Modélisation, Conception Système d'Architectures Hétérogènes pour les Applications Embarquées

Sébastien Bilavarn

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Université Nice Sophia Antipolis

cnrs

HABILITATION À DIRIGER DES RECHERCHES

Modélisation, Conception Système d'Architectures Hétérogènes pour les Applications Embarquées

Éléments d'amélioration de l'efficacité énergétique des systèmes sur puce de silicium

Sébastien BILAVARN

Laboratoire d'Electronique Antennes et Télécommunications - LEAT

Présentée en vue de l'obtention d'une HDR en discipline STIC, Université Côte d'Azur

Soutenue le:
28/09/2018

Devant le jury, composé de :
Bertrand GRANADO, Professeur, Sorbonne Université
Russel TESSIER, Professor, University of Massachussets
Gilles SASSATELLI, DR CNRS, Université Montpellier II
Olivier SENTIEYS, Professeur, Université de Rennes I
Michel AUGUIN, DR CNRS, Université de Nice Sophia Antipolis
Eric DEBES, Expert invité, Thales Avionics

Partie II – Travaux de recherche détaillés et perspectives

A ma famille.

Sommaire

The following selection of papers reflects the backbone of this habilitation thesis. We provide here a map of the document outline, with references to the corresponding publications given in Appendix, in a way to make it more accessible for an international audience.

« Modeling and System Level Design of Heterogeneous Architectures for Embedded Applications – Elements of energy efficiency improvement for System-on-Chips »

[1] Cécile Belleudy and Sébastien Bilavarn, **Power Models and Strategies for Multi-processor Platforms**, Design Technology for Heterogeneous Embedded Systems, G. Nicolescu, I. O'Connor, C. Piguet, Springer, 2012.

[2] Taheni Damak, Imen Werda, Sébastien Bilavarn and Nouri Masmoudi, **Fast Prototyping H.264 Deblocking Filter using ESL Tools**, Transactions on Systems, Signals & Devices, Shaker Verlag, Vol. 8, No 3, pp.345-362, December 2013.

[3] Robin Bonamy, Sébastien Bilavarn, Daniel Chillet and Olivier Sentieys, **Power Consumption Models for the use of Dynamic and Partial Reconfiguration**, Microprocessors and Microsystems, Elsevier, February 2014.

[4] Robin Bonamy, Sébastien Bilavarn, Daniel Chillet and Olivier Sentieys, **Power Modeling and Exploration of Dynamic and Partially Reconfigurable Systems**, Journal of Low Power Electronics, American Scientific Publishers, September 2016.

[5] François Duhem, Fabrice Muller, Robin Bonamy and Sébastien Bilavarn, **FoRTReSS: a Flow for Design Space Exploration of Partially Reconfigurable Systems**, Design Automation of Embedded Systems, Springer, February 2015.

[6] Sébastien Bilavarn, Jabran Khan Jadoon, Cécile Belleudy and Muhammad Khurram Bhatti, **Effectiveness of Power Strategies for Video Applications: a Practical Study**, Journal of Real-Time Image Processing, Springer, January 2014.

[7] Robin Bonamy, Sébastien Bilavarn and Fabrice Muller, **An Energy-Aware Scheduler for Dynamically Reconfigurable Multi-Core Systems**, In Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2015, Bremen, Germany, 29/06-01/07 2015.

[8] Sébastien Bilavarn, Andrea Castagnetti and Laurent Rodriguez, **A Video Monitoring Application for Wireless Sensor Networks using IEEE 802.15.4**, In Proceedings of the 2nd Workshop on Ultra-Low Power Sensor Networks, WUPS 2011, Como Italy, November 2011.

[9] Robin Bonamy, Sébastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet and Fabrice Lemonnier, **Energy Efficient Mapping on Manycore with Dynamic and Partial Reconfiguration: Application to a Smart Camera**, International Journal of Circuit Theory and Applications, Wiley, June 2018.

[10] Joel Wanza, Sébastien Bilavarn, Maarten De Vries, Said Derradji and Cécile Belleudy, **Efficiency Modeling and Analysis of 64-bit ARM Compute Nodes for HPC**, Microprocessors and Microsystems, Elsevier, September 2017.

Travaux de recherche

« Modélisation, Conception Système d'Architectures Hétérogènes pour les Applications Embarquées – Éléments d'amélioration de l'efficacité énergétique des systèmes sur puce de silicium »

1 Objectifs scientifiques et enjeux technologiques

Les enjeux d'une meilleure efficacité énergétique sont un moteur essentiel de l'évolution des technologies semi-conducteur depuis la popularisation des systèmes répartis, nomades et embarqués. L'exemple le plus parlant est celui des téléphones portables ces vingt dernières années et les perspectives technologiques autour des objets connectés (Internet des Objets, *IoT*) renforcent encore plus la nécessité de maîtrise de la consommation d'énergie pour le déploiement de quelques milliards d'objets autonomes et connectés attendus à l'horizon 2020. On estime aussi par exemple qu'il faut améliorer l'efficacité énergétique d'un facteur 20 à 30 pour construire la prochaine génération de calculateurs haute performance (Exascale, 10^{18} FLOPS), qui sont des vecteurs fondamentaux d'avancées pour de nombreuses disciplines scientifiques (simulations numériques, infrastructure *cloud* pour l'IoT, etc.). Sur ce point d'ailleurs, le déploiement de la 5G pose également des défis énormes puisqu'elle devra supporter une augmentation du trafic de données estimée à plus d'un facteur 1000 pour une consommation énergétique réduite de moitié (source NGMN). Une autre conjecture très étudiée concerne les technologies d'intégration nanométriques qui atteignent une densité de transistors telle qu'elles ne permettent plus l'alimentation électrique de toute la puce dans des conditions de température tolérables (*dark silicon*).

De nombreux efforts ont donc été entrepris autour de l'idée d'introduire et d'exploiter une hétérogénéité à différents niveaux d'un Système sur Puce (SoC) pour augmenter les performances tout en bénéficiant d'une meilleure efficacité énergétique, et résoudre par ailleurs les problèmes de densité de chaleur et de puissance posées par les technologies semi-conducteur à court terme. Un exemple représentatif de cette tendance est la généralisation de la technologie *big.LITTLE* intégrée par ARM dans leurs architectures multiprocesseur haute performance produites actuellement. Néanmoins, le niveau d'hétérogénéité atteint par les plateformes est tel qu'il accroit considérablement la complexité de déploiement et d'utilisation pour les applications et systèmes à venir. Si les espoirs sont prometteurs, c'est à la condition expresse d'une utilisation efficace car une mauvaise exploitation de l'hétérogénéité (e.g. charge CPU mal répartie) risque de conduire au résultat inverse et à une plus forte consommation. Or il ne s'agit plus simplement de savoir sur quel type de cœur allouer les tâches d'une (ou plusieurs) application(s), mais aussi d'intégrer les aspects dynamiques de l'exécutif tels que la gestion énergétique (DVFS, ordonnancement) et les opportunités matérielles associées (accélération, parallélisme) dans une profusion de décisions visant à garantir le

contrôle et l'amélioration de la consommation globale à différents niveaux (conception, exécutif, application).

Depuis de nombreuses années, une réflexion continue a été accordée aux technologies reconfigurables qui représentent plus que jamais un sujet de recherche très effervescent. Si les avancées dans le domaine, motivées par l'énorme potentiel en accélération matérielle de certains traitements, ont été très innovantes et significatives en termes de technologie, d'outils et de langage ces vingt dernières années, leur exploitation efficace se heurte toujours en pratique à la difficulté d'établir une abstraction des processus très complexes de conception et de déploiement matériels. Les travaux de recherche menés depuis la thèse abordent cette question par l'étude d'outils et de méthodes de conception des systèmes numériques en se focalisant sur les contraintes énergétiques et les technologies embarquées. En particulier les recherches effectuées ces quinze dernières années interviennent toutes au sein de projets collaboratifs ambitieux avec des industriels représentatifs du domaine (Intel, Thales, STMicroelectronics, etc.) sur les aspects énergétiques. Malgré les difficultés à poursuivre aujourd'hui une étude fondamentale sur la durée dans un contexte global de recherche appliquée (projets collaboratifs financés à trois ou quatre ans sur des enjeux fortement influencés par la R&D industrielle), c'est une démarche délibérée visant à intégrer les objectifs d'appels d'offre court terme dans un objectif scientifique plus large qui a permis d'identifier et d'approfondir des idées novatrices sur des aspects plus fondamentaux. Les contributions qui ont été développées pour répondre à ces préoccupations sont donc organisées autour de trois axes qui reflètent une décomposition cohérente de la réflexion:

1. La modélisation et la conception système (déploiement)
2. La gestion des ressources et de l'énergie (exécutif)
3. Les éléments d'une amélioration significative de l'efficacité énergétique

Le premier axe traite des méthodologies de conception d'architectures hétérogènes en mettant l'accent sur la définition d'une modélisation énergétique globale qui capture les caractéristiques importantes des plateformes d'exécution (multi cœur, DVFS, hétérogénéité logicielle, accélération matérielle, …) en considérant certaines perspectives technologiques (processeurs multi/many cœurs, reconfiguration dynamique partielle, *3D stacking*, hiérarchie mémoire, …). Il se base sur différents travaux de caractérisation et de mesures énergétiques menés sur une variété de plateformes multiprocesseur et de composants reconfigurables dans le cadre des projets PHERMA (ANR), Open-PEOPLE (ANR), et COMCAS (EUREKA CATRENE). Les résultats de ces projets ont ainsi permis de développer progressivement les éléments d'une modélisation assez complète qui a pu être évaluée dans différents contextes (environnements de conception, applications, plateformes) et serviront avantageusement à une prise de décision cohérente au niveau de la conception, du déploiement (exploration) et de l'exécutif (ordonnancement, gestion énergétique).

Le deuxième axe s'intéresse en détail au second élément essentiel qui conditionne l'efficacité énergétique, en suivant toujours une démarche pragmatique, par l'étude et l'expérimentation de politiques de gestion énergétique et de stratégies d'ordonnancement faible consommation. Les résultats obtenus et les conclusions énoncées reposent ainsi sur des politiques énergétiques avancées qui ont été spécifiées, simulées et surtout développées jusqu'au prototypage complet sur différentes plateformes (réelles et virtuelles), ce qui représente un travail conséquent étant donné la complexité des développements de ce type au sein du noyau des OS (technique sous dépôt de brevet international [10]). Cet effort

supplémentaire par rapport à l'approche beaucoup plus répandue qui se base sur des simulations est extrêmement riche d'enseignements et permet de beaucoup mieux appréhender les gains tangibles mais aussi les conditions réelles d'efficacité et d'amélioration. Les conclusions ont ainsi conduit à proposer une approche avancée pour l'amélioration de l'efficacité énergétique qui exploite la définition de stratégies dédiées grâce à une solution visant à faciliter leur intégration aux OS en pratique.

Enfin, il découle de ces deux axes une synthèse des conditions d'efficacité aux niveaux clés de la conception (déploiement, exploration) et de l'exécutif (stratégies énergétiques, ordonnancement) par une étude de leur couplage plus élaboré pour un meilleur niveau de résultats. Ces éléments d'amélioration ont conduit à envisager une méthodologie centrée sur l'efficacité énergétique qui reflète un concept de conception intégrée approfondi dans le cadre du projet BENEFIC (EUREKA CATRENE) avec des études de cas montrant des réductions énergétiques très significatives sur des démonstrateurs concrets.

2 Etudes doctorales et post-doctorales

Mon parcours en recherche a débuté en 1997-1998 par une première année de formation doctorale (DEA), qui préparait au doctorat avant son remplacement par le master recherche avec la réforme LMD de 2004. En particulier, le stage à vocation d'orientation vers la recherche qui y était associé a été réalisé au sein de la société Thomson Multimedia R&D à Rennes. Outre les bénéfices d'une expérience très formatrice de quelques mois en R&D industrielle, il a été l'occasion de ma toute première confrontation avec des composants reconfigurables par le prototypage d'un algorithme de compensation de mouvement sur un FPGA de chez Altera. La compensation de mouvement utilisée était une version spécifique, non basée sur les standards MPEG de l'époque, développée par Thomson Multimedia pour ses besoins propres dans le domaine de la télévision numérique haute définition. Les composants reconfigurables, mais aussi la compression vidéo resteront deux constantes caractéristiques dans tous mes travaux qui suivront jusqu'à aujourd'hui.

Stage de DEA: « Implantation en VHDL d'un algorithme de compensation de mouvement sur FPGA »

Il m'a ensuite été donné la possibilité de poursuivre en thèse au laboratoire LESTER (aujourd'hui Lab-STICC) à l'Université de Bretagne Sud dans le domaine plus général des méthodologies de conception logicielles / matérielles (Codesign). L'objectif global des recherches du laboratoire étaient alors de définir des approches qui permettent de faciliter et d'automatiser le déploiement d'applications sur des architectures hétérogènes, c'est à dire qui peuvent être composées d'unités de natures différentes (processeurs, DSP, FPGA, ASIC). Le travail de la thèse a consisté à développer une méthodologie d'exploration et d'estimation qui permette, à partir d'une spécification applicative comportementale (C, C++, CDFG), d'explorer automatiquement l'espace de conception de fonctions matérielles pour fournir un ensemble de coûts possibles sur FPGA (performance, occupation) comme support utile aux décisions de partitionnement logiciel / matériel. Une exigence à remplir dans ce contexte était alors d'éviter les approches de synthèse complète (HLS) pour réduire la complexité et pouvoir explorer facilement un large éventail de solutions d'implantation. L'approche originale développée s'est attachée à combiner plusieurs techniques d'estimation existantes (au niveau traitement, mémoire, contrôle) dans une approche globale et cohérente estimant le coût total d'une architecture complète sur FPGA. Elle se compose de deux étapes. Une première, dite d'estimations structurelles, définit un certain nombre de paramètres architecturaux au niveau RTL (nombre d'opérateurs, d'accès mémoires, d'états de contrôle, etc.). Une deuxième étape, dite d'estimations physiques, établit une projection physique de ces paramètres sur chaque unité de l'architecture (traitement, mémoire, contrôle) à partir de librairies décrivant les ressources du composant cible (opérateurs, mémoires RAM et ROM).

Doctorat: « Exploration Architecturale au Niveau Comportemental – Application aux FPGAs »

A la suite du doctorat, en Juin 2002, j'ai entamé un postdoc à l'EPFL afin de travailler sur une collaboration académique / industrielle entre le Laboratoire de Traitement des Signaux (LTS, EPFL) et le System Technology Lab d'Intel Corp. (Santa Clara). Le sujet d'étude et du projet, « Processor Enhancements for

Media Delivery in Heterogeneous Environments », dans lequel j'ai exercé la principale responsabilité scientifique est idéalement complémentaire avec l'aspect matériel des recherches précédentes en approfondissant la dimension logicielle (au sens processeur) avec Intel. Le LTS, dirigé à l'époque par le Pr Murat Kunt, est un laboratoire de recherche reconnu internationalement pour ses contributions dans les domaines de l'image numérique et de la compression vidéo. A l'époque où je l'ai rejoint, l'équipe du LTS2 travaillait activement sur une technique de codage vidéo très bas débit appelée *Matching Pursuit* qui offre des propriétés particulièrement adaptée (compression, scalabilité) à la diffusion de contenu vidéo sur le net (video streaming, visio conférence, etc.). La contrepartie de ces algorithmes est néanmoins leur très importante complexité qui dépasse largement la capacité des processeurs de l'époque, même spécialisés. L'objectif de la collaboration s'est donc porté sur la recherche de solutions architecturales pour les processeurs généralistes (PC, serveurs, ordinateurs portables, appareils de poche) dans le contexte général des algorithmes de traitement vidéo et video streaming. Les solutions proposées explorent l'utilisation de coprocesseurs et d'accélérateurs matériels reconfigurables. Le potentiel de la reconfiguration étant déjà bien connu, mais toujours techniquement très difficile à exploiter, et notamment du point de vue du modèle de programmation (qui reste encore aujourd'hui le problème principal), deux axes de recherches ont été définis afin de développer des solutions réalistes pour Intel en considérant l'état de l'art de l'époque:

1. La définition d'une architecture de coprocesseur reconfigurable dynamiquement (gros grain et faible consommation), exploitable par un modèle de programmation réaliste par un concept d'instructions configurables pour le traitement multimédia.

2. L'étude de flots de compilation efficaces par des techniques d'exploration et de Synthèse de Haut Niveau (HLS) pour apporter des solutions aux problèmes de génération de code vers des accélérateurs de type FPGA. Ces aspects en particulier représentent la base des travaux réalisés à l'issu de ma prise de fonction en tant que Maître de Conférences. Ce projet initialement financé pour un an a été finalement reconduit deux fois (pour une durée totale de trois ans) suite à l'intérêt porté par Intel à ces travaux, à l'époque en 2002. Il est aujourd'hui assez gratifiant de relever l'aspect novateur du sujet à l'époque chez Intel, surtout en considérant le très fort investissement d'Intel récemment dans le domaine des méthodologies de codesign et du reconfigurable (achat d'ALTERA, Cofluent, etc).

Postdoc: « Processor Enhancements for Media Delivery in Heterogeneous Environments »

3 Modélisation, conception système

3.1 Modélisation de plateformes multiprocesseur

L'étude décrite par la suite a été développée dans le cadre du projet PHERMA (*Parallel Heterogeneous Energy efficient Real-time Multiprocessor Architecture*, ANR). Elle aborde la modélisation en performance et en énergie d'architectures multiprocesseur symétriques (SMP) sur des applications vidéo représentatives et par une approche pragmatique. Les études développées ici en termes de recherche d'efficacité énergétique ont ensuite été logiquement étendues aux architectures hétérogènes (section 3.4), puis aux stratégies de gestion énergétique multiprocesseur (section 4.1) sur des applications et plateformes concrètes. Le choix de l'application utilisée pour ce travail, dont le code est entièrement fonctionnel, a été développé par Thales Communication & Security. Il sera détaillé dans la mesure où ce décodeur parallélisé a été central dans plusieurs travaux suivants.

3.1.1 Evaluation de décodeurs H.264 sur ARM11 MPCore

Dans un premier temps, différentes versions de décodeurs H.264 ont été évaluées sur plateforme CT11 MPCore test chip, une des premières plateformes multiprocesseur pour l'embarqué (2005 [1]). Cette plateforme homogène est constituée de quatre cœurs basés sur l'architecture ARM11. Chaque cœur est associé à un cache de niveau 1 (32 KB instructions, 32 KB données) et le système inclut une unité pour la gestion de la cohérence mémoire avec la cache unifiée de niveau 2 (1 MB). Pour nos besoins, le MPCore est utilisé en mode symétrique sous le contrôle d'un noyau Linux SMP (*Symmetric Multi-Processing*), mais peut également fonctionner en mode AMP (*Asymmetric Multi-Processing*). La plateforme supporte la désactivation en ligne de cœurs inactifs ainsi que la modification de la tension d'alimentation et de la fréquence des cœurs, permettant théoriquement l'adaptation de la puissance consommée en cours d'exécution. Ces possibilités de gestion énergétique sont contrôlées par des ressources matérielles spécifiques dont l'utilisation est détaillée en section 3.1.2.

ARM11 MPCore

Séquence foreman 300 images

Décodeur	Ref ITU-T	Univ. Chemnitz	Thales Part/fonct	Thales Multislice
Perf (img/sec)	1,7	6,0	1,6	6,3
Tex (sec)	173,6	49,9	189,5	47,7

Tableau 1 - Performances de différents décodeurs H.264. Mesures réalisées pour 1 cœur actif à 250 MHz.

Les quatre versions suivantes du décodeur ont été considérées pour évaluation de leurs performances sur ARM11 MPCore: le décodeur de référence ITU-T [2], une version open source développée à l'Université de Chemnitz [3], et deux versions développées par Thales qui se basent sur des parallélisations par fonctions ("Part/fonct") et par tranches ("Multislice"). Les performances mesurées sont exprimées en nombre d'images décodées par seconde (Tableau 1). Le décodeur de référence ITU-T n'est pas la version la plus

lente, elle est toutefois loin d'atteindre une qualité acceptable (1,7 img/sec) car le code n'a pas été développé dans une optique de performances. La version Chemnitz est une version plus simple mais fonctionnelle qui permet déjà de décoder des flux de référence en profil *Baseline* avec une efficacité de compression proche de MPEG-4 ASP. Cette version a elle été développée avec un objectif d'implantation (sur PC et DSP) ce qui explique les performances de la version de base (6,0 img/sec). Un autre avantage est qu'elle est plus facile à modifier, les deux parallélisations considérées dans la suite ont été déduites de cette version.

La version parallélisée par fonction se base sur une décomposition en sept sous-fonctions : *TG* (Traffic Generator), *Nal* (gestion des paquets réseaux), *Slice* (décodage d'une tranche par macrobloc), *Render1* (moitié supérieure d'un macrobloc), *Render2* (motié inférieure d'un macrobloc), *Libu* (gestion des macroblocs) et *Ramdac* (gestion de l'affichage). L'application est décrite par un ensemble de tâches logicielles associées à chaque fonction qui communiquent par l'intermédiaire de *FIFOs*. Cette configuration implique donc des transferts de données importants entre tâches (donc entre cœurs) qui sont peu adaptés à l'architecture symétrique. La Figure 1 montre les performances de cette version en fonction du nombre de cœurs activés. L'accélération atteint un facteur 4,8 pour quatre cœurs. Ce résultat ne correspond toutefois qu'à une vitesse de décodage de 7,5 img/sec, ce qui est nettement inférieur aux performances de la version parallélisée par tranches (20,6 img/sec).

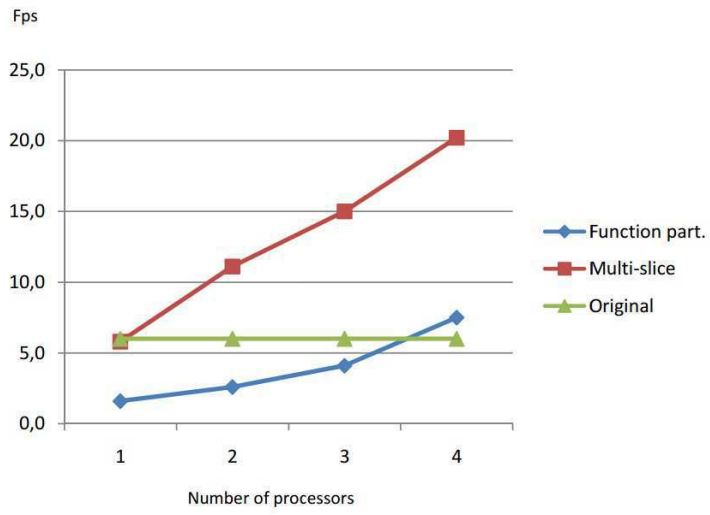


Figure 1 - Performances en exécution multiprocesseur. Configurations: 1, 2, 3, 4 cœurs à 250 MHz, séquence *foreman* 300 images.

La parallélisation par tranches du décodeur se base sur la possibilité de décoder simultanément plusieurs tranches (*slices*) d'une image. Chacune des tranches travaille sur une zone indépendante de l'image et peut référencer les tranches d'une image précédente. Une tranche est ainsi prise en charge par une tâche POSIX et subit le processus de décodage dans son intégralité et indépendamment des autres tranches de la même image. Les dépendances de données sont donc réduites, il n'existe de dépendances qu'au niveau des structures d'entrée qui recueillent le flux vidéo et en sortie à la reconstruction de l'image. Le décodeur fonctionne en créant autant de tâches qu'il y a de tranches dans une image qui sont alors traitées en parallèle. Cette solution bénéficie par ailleurs d'une meilleure localité des données qui peut être exploitée par la mémoire cache L1. Les résultats de la Figure 1 démontrent la très bonne faculté de passage à l'échelle de ce décodeur en configuration de huit tranches dont les performances augmentent linéairement de 6,3 à 20,6 img/sec en fonction du nombre de cœurs utilisés (1, 2, 3, 4). L'accélération maximale est alors d'un facteur 3,27 pour quatre cœurs.

Le niveau tranche (*slice*) est celui qui permet d'obtenir le meilleur rapport performance/complexité du contrôle à rajouter, ce qui constitue un point important pour une exécution sur une plateforme embarquée. Par ailleurs, la décomposition homogène par tranche d'image se prête idéalement à un

schéma d'exécution SMP en permettant d'équilibrer la charge sur les cœurs (1 tranche = 1 tâche). Toutefois, le traitement indépendant des tranches d'une image réduit la fenêtre de recherche du mouvement possible à l'étape de compression (estimation de mouvement), ce qui a pour conséquence de dégrader les taux de compression. La Figure 2 montre l'évolution de la taille de la vidéo compressée en fonction du nombre de tranches. Afin de garder une efficacité acceptable, nous ne considérerons pas plus de huit tranches (i.e. huit tâches), ce qui correspond à une augmentation de 10% par rapport au meilleur taux de compression possible (1 tranche). Par la suite, nous considérons quatre versions du décodeur (1, 2, 4 et 8 tranches/tâches) qui permettront de faire varier la charge du CT11 MPCore pour différents nombre de cœurs utilisés.

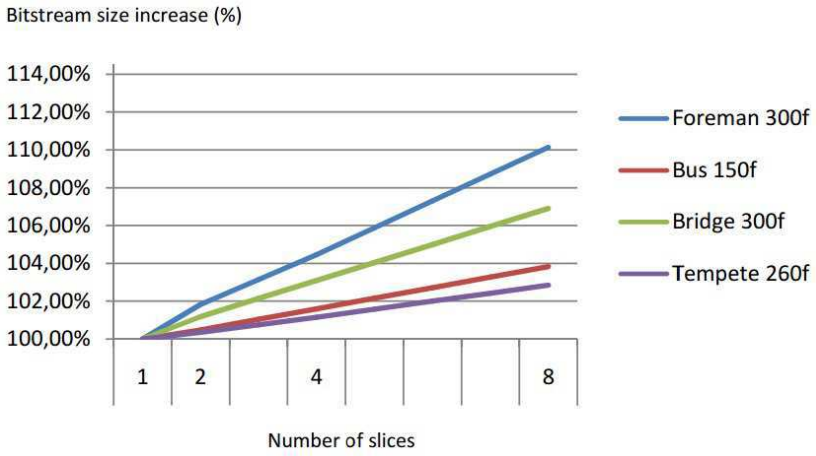


Figure 2 - Augmentation de la taille du *bitstream* vidéo pour 1, 2, 4, 8 tranches et différentes séquences vidéo (foreman, bus, bridge, tempete).

3.1.2 Analyse des performances

Les performances du décodeur *multislice* sont mesurées pour plusieurs séquences vidéo (*foreman* 300, *bus* 150, *bridge* 300, *tempete* 260 images) et dans différentes configurations du décodeur (1, 2, 4, 8 tranches). Les séquences sont en format 256*256 pixels et les quatre cœurs de la plateforme sont activés pour ces mesures. Les résultats de la Figure 3 montrent la vitesse de décodage et l'accélération obtenues par rapport à l'exécution d'une version *monoslice* sur un seul cœur.

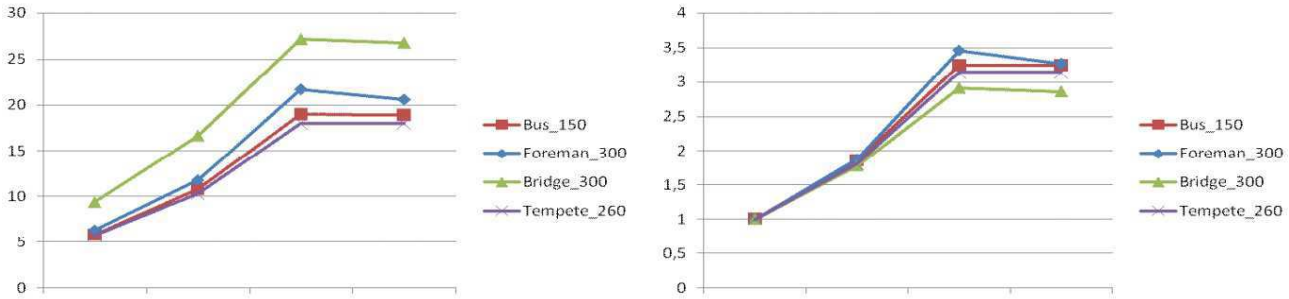


Figure 3 - Performances (fps) et accélération pour différentes configurations (nombre de slices) et séquences vidéo sur quatre cœurs.

La première observation est que la courbe d'accélération suit une tendance linéaire croissante lorsque le nombre de *slices* augmente jusqu'à quatre. Le facteur d'accélération vaut 3,19 dans ce cas (moyenne sur toutes les séquences vidéo), ce qui est proche du maximum théorique de 4. Ce point dénote une bonne répartition des tâches POSIX, donc de la charge de travail sur les cœurs, et renforce l'intérêt de la décomposition par *slices* pour l'architecture SMP. La faible dégradation en performance est liée à des pénalités de traitement du flux d'entrée (avant de générer les tâches parallèles), de synchronisation des tâches et aussi certainement à l'influence du système d'exploitation. Ces facteurs conditionnent également

la perte d'efficacité observable dans une configuration de huit *slices*. Dans ce cas, on ne bénéficie plus de cœurs supplémentaires pour accélérer le décodage et les performances sont même moins bonnes que pour une configuration de quatre *slices*. Le nombre de tâches par cœur étant de deux, l'effet de changements de contextes est aussi certainement à prendre en compte.

En examinant les temps de décodage par séquence vidéo, on remarque d'importantes variations. Des différences sont par exemple sensibles entre les séquences *bridge* (27,2 fps) et *tempete* (18 fps). La première séquence est composée d'un large arrière plan fixe avec quelques objets de petite taille en translation (faible complexité et quantité de mouvement). La deuxième séquence comporte un arrière plan mobile avec de nombreux objets en déplacement rapide (forte complexité et quantité de mouvement). Les performances sont fortement dépendantes des propriétés de mouvement des séquences vidéo (+/- 20% sur ces deux exemples), ce qui peut être exploité avantageusement pour définir une adaptation fine de la fréquence des cœurs en fonction de la vitesse de décodage.

3.1.3 Analyse de la consommation

Les profils de consommation sont analysés pour différentes configurations du décodeur et de la plateforme MPCore. Les mesures se basent sur l'utilisation des registres internes du testchip MPcore qui à un instant donné fournissent directement la consommation en tension et en courant des cœurs. Il faut noter ici que les points de fonctionnement (couples tension/fréquence) ne sont pas définis en pratique sur cette plateforme et qu'elle ne supporte par ailleurs qu'un changement statique de la fréquence (pas de DVFS strictement). Les configurations consistent en plusieurs couples tension/fréquence que nous avons définis nous-mêmes afin d'en évaluer l'impact sur la consommation. Cette influence tient compte de la charge des cœurs qui varie pour différentes configurations du décodeur (1, 2, 4, 8 slices et à vide). Le testchip MPCore autorise des variations de la tension d'alimentation qui vont de 0,95V à 1,45V. Pour respecter les spécifications recommandées par ARM (1,20V +/-20%), nous avons fixé les couples tension/fréquence suivants: 0,95V/150MHz, 1,08V/200MHz, 1,20V/250MHz (nominal) et 1,32V/300MHz. Les mesures effectuées sont également paramétrées par nombre de cœurs utilisés (1, 2, 3 et 4). Les cœurs sont rendus inactifs par l'utilisation du mode *WFI* (*Wait For Interrupt*) dans lequel l'horloge d'un cœur est stoppée, réduisant ainsi sa part sensible de puissance dynamique consommée.

Devant le nombre important de mesures relevées, nous analysons tout d'abord les profils de consommation dans les conditions nominales d'utilisation (1,20V/250/MHz). La Figure 4 montre les résultats correspondants pour différentes charges et configurations de la plateforme. Pour des raisons de commodité de visualisation, nous n'avons représenté que les profils de consommation pour la séquence *foreman*, les profils des autres séquences vidéo révèlent les mêmes tendances de variation: la puissance consommée oscille entre une valeur minimum et une valeur maximum en fonction de l'activité des cœurs. Pour des configurations charge/ cœur équilibrées (e.g. 2 slices / 2 cœurs), la distribution de la puissance reste proche de la valeur maximale ce qui indique un fonctionnement efficace des cœurs. Pour des configurations non équilibrées (e.g. 4 slices / 3 cœurs), la consommation est plus imprévisible mais toujours située entre le minimum et le maximum précédemment identifiés. Elle correspond logiquement à une charge CPU qui est plus variable dans ce cas puisque certaines tâches s'exécutent sur des cœurs différents au gré de leur disponibilité. L'amplitude des variations dépend du nombre de cœurs: moins de 5% de la valeur moyenne pour un cœur, jusqu'à plus de 50% pour quatre cœurs. Ce dernier point peut avoir de l'importance pour des systèmes alimentés sur batterie qui est alors soumises à de plus fortes contraintes.

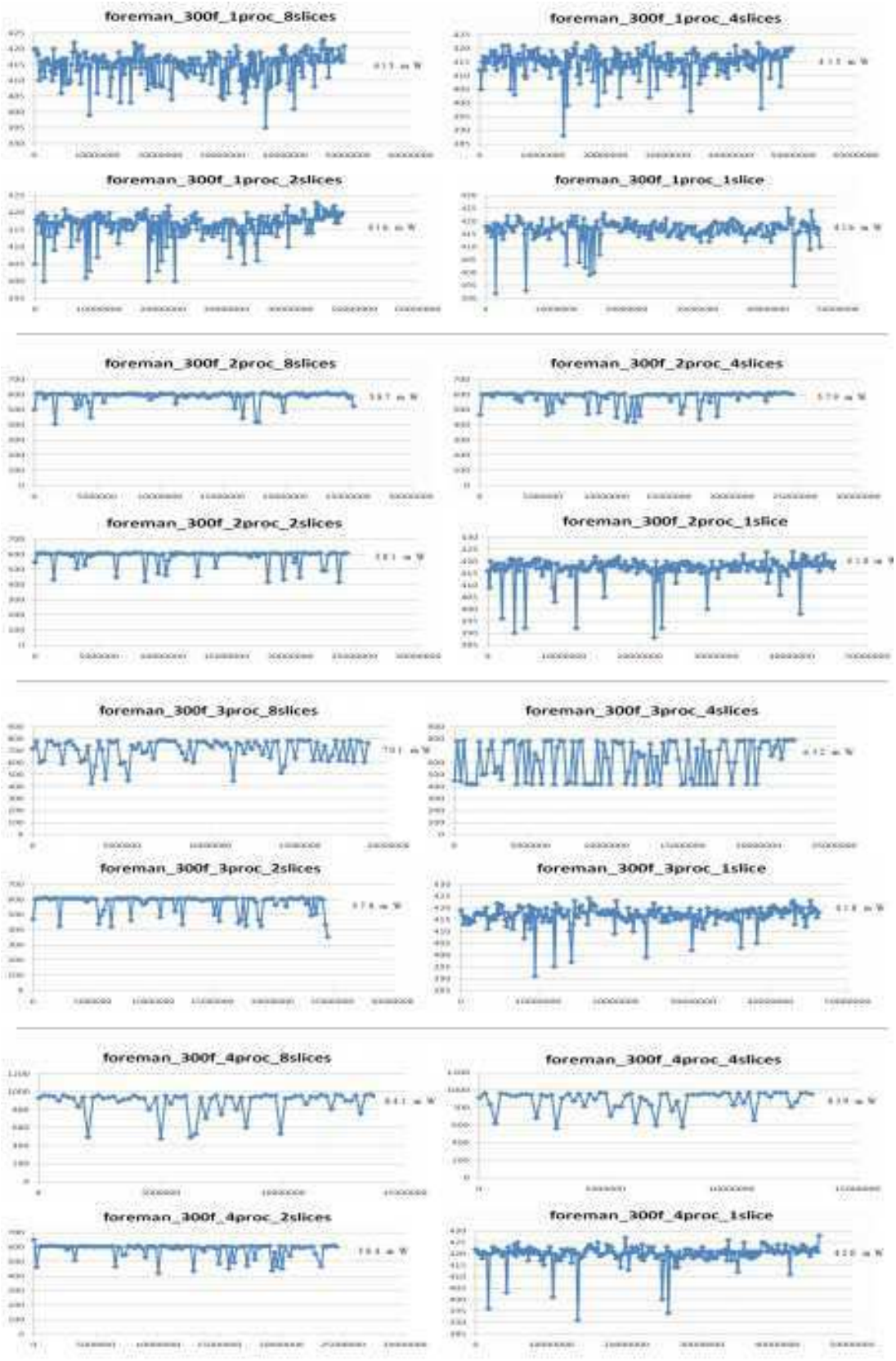


Figure 4 - Profils de consommation (mW/µs) à 1,20V/250MHz pour plusieurs configurations thread / CPU.

Si nous analysons la puissance moyenne correspondante (section 1,20V/250MHz du Tableau 2), nous pouvons observer qu'elle est facilement identifiable et prédictible lorsque les charges des cœurs sont équilibrées : 840mW / 4 cœurs, 580mW / 2 cœurs, 415mW / 1 cœur.

1,32V/300MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**à vide**
1 cœur	633	634	636	636	361
2 cœurs	885	878	880	637	364
3 cœurs	1034	932	882	637	363
4 cœurs	1256	1187	867	642	355

1,20V/250MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**à vide**
1 cœur	415	415	416	416	241
2 cœurs	587	579	581	418	241
3 cœurs	701	632	578	418	238
4 cœurs	841	839	584	420	238

1,08V/200MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**à vide**
1 cœur	270	270	270	271	156
2 cœurs	381	376	375	271	148
3 cœurs	462	408	381	272	153
4 cœurs	561	568	380	273	154

0,95V/150MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**à vide**
1 cœur	160	160	160	160	93
2 cœurs	225	224	225	160	89
3 cœurs	276	241	224	161	89
4 cœurs	335	335	223	162	92

Tableau 2 - Puissance moyenne (mW) en fonction de la configuration charge/processeur pour plusieurs configurations tension/fréquence.

On remarque également que dans les configurations où le nombre de slices est inférieur au nombre de cœurs, la consommation est ramenée à celle du nombre de cœurs actifs. Par exemple : 2, 4, 8 slices sur 2 cœurs, ou 2 slices sur 2, 3, 4 cœurs ont une consommation stable, qui oscille entre 578 et 587 mW. Cela signifie que l'OS désactive lui-même les cœurs inutilisés par le mode WFI pour diminuer la consommation globale. Ces résultats montrent qu'il est possible de définir une modélisation simple de la puissance, à condition d'exclure certaines configurations sous optimales.

Par la suite, pour étendre la caractérisation de la consommation du décodeur sur le testchip MPCore, nous considérons les valeurs de tension/fréquence suivantes: 0,95V/150MHz, 1,08V/200MHz et 1,32V/300MHz. Compte tenu de l'analyse précédente, nous nous limitons aux valeurs moyennes de puissance par configuration charge/cœur (Tableau 2), incluant la consommation des cœurs à vide, lorsqu'aucune tâche utilisateur n'est exécutée. Dans chaque cas, le comportement de la consommation au cours de l'exécution suit les mêmes tendances que celles constatées en fonctionnement nominal (1.20V/250MHz). Les mesures effectuées confirment les observations précédentes concernant l'effet de la charge sur les valeurs moyennes de consommation. En outre, les variations de consommation s'étendent de 160 mW (1 cœur / 150 MHz) à 1256 mW (4 cœurs / 300 MHz), ce qui laisse entrevoir une possibilité intéressante d'adaptation en cours d'exécution de la tension/fréquence des cœurs en fonction de la charge de traitement. Il est également possible de vérifier que la puissance suit bien une fonction linéaire croissante de V^2F, conformément au modèle théorique [4]. On peut noter enfin que la consommation à vide est identique quelle que soit le nombre de cœurs activés (à tension / fréquence donnée). En pratique, les cœurs non utiles sont placés en mode *WFI* par le noyau Linux, un cœur unique demeure actif dans ce cas pour les besoins du système d'exploitation. Le mode *WFI* déconnecte l'horloge d'un cœur (i.e. la puissance dynamique associée) permettant de réduire la consommation totale en proportion. Selon ARM, une part importante de la puissance due aux courants de fuite est associée au CPU dans notre test chip. Il y a également une partie de la puissance dynamique qui provient du reste de la puce. Ces surcoûts et leurs effets sont étroitement liés au testchip qui est une plateforme d'évaluation, en comparaison à un circuit de production qui pourrait par ailleurs fonctionner à une fréquence beaucoup plus élevée. En supposant que les mesures incluent les consommations statiques et dynamiques des cœurs et du reste du circuit (cache L2, PLL...), on peut faire l'estimation suivante:

$$P_{HORS\_CPU} = 2 * P_{1CPU} - P_{2CPU} \qquad (1)$$

Cette expression correspond à 256 mW à 1,20V/250MHz, qui est une valeur très proche de la consommation à vide dans cette configuration (240mW). Cela semble donc confirmer que la consommation mesurée à vide est essentiellement due à la consommation statique et dynamique hors CPU, et que la consommation dynamique du seul cœur resté actif dans ce cas est faible. Elle représente la consommation minimale et incompressible quand aucune tâche utilisateur n'est exécutée sur le MPCore, ce qui est une donnée importante du point de vue du modèle énergétique.

3.1.4 Modèle énergétique multiprocesseur homogène

Les résultats précédents permettent de définir une modélisation énergétique simple d'architectures multiprocesseur homogène sous certaines hypothèses. En supposant en particulier que la charge des cœurs est homogène (qui repose sur une parallélisation efficace du code), la consommation prend des valeurs caractéristiques qui dépendent de la configuration de la plateforme: nombre de cœurs, état (à vide, en charge) et fréquence des cœurs. Cette étude devra être confrontée à d'autres plateformes et étendue à

d'autres paramètres, comme le DVFS par cœur qui n'est pas supporté par le testchip MPCore. Il est important de noter que les mesures tiennent compte de la cache L1 associée à chaque cœur mais pas de la cache unifié L2. De manière plus générale, la modélisation énergétique des aspects mémoire, qui est aussi un aspect complexe, n'est pas abordée ici.

Ces éléments de modélisation seront utilisés par la suite dans la définition de plusieurs méthodologies d'analyse de l'efficacité énergétique (projets ANR Open PEOPLE, CATRENE BENEFIC, environnement FoRTReSS) et étendus à l'hétérogénéité des plateformes multiprocesseurs en considérant d'autres types de cœurs, les possibilités de DFVS et d'accélération matérielle et de reconfiguration dynamique qui seront développés dans les deux sous chapitres suivants. Cette étude est également approfondie par le développement d'une stratégie DVFS pour du traitement vidéo sur plateforme ARM1176-JZFS monoprocesseur, supportant un DVFS fonctionnel, et qui est décrit au chapitre 4 relatif aux stratégies de gestion des ressources et de l'énergie.

3.2 Accélération matérielle et reconfigurable

La méthodologie présentée par la suite est un flot de conception ESL (*Electronic System Level Design*) développé par une collaboration avec le Laboratoire d'Electronique et de Technologies de l'Information (LETI, Ecole Nationale d'Ingénieurs de Sfax) à travers le co-encadrement de la thèse de Mme Taheni Damak. L'étude de cas repose sur un décodeur H.264 entièrement développé en C++ par le LETI (différent des versions présentées section 3.1.1.) pour permettre un déploiement plus facile sur différentes cibles logicielles / matérielles. Cette collaboration s'intéresse plus particulièrement à l'accélération matérielle de certaines fonctionnalités du décodeur par l'utilisation de la synthèse de haut niveau (*High Level Synthesis*, HLS) afin d'en évaluer la pertinence (automatisation), les limites (modifications du code et génération d'une interface bus) et de proposer des solutions. Le flot de conception se base sur une coopération d'outils de CAO existants pour permettre de concevoir une architecture logicielle/matérielle jusqu'à la génération complète et l'exécution des accélérateurs sur plateformes processeur/FPGA réelles. Il se révèlera très utile pour des études ultérieures concernant la modélisation énergétique d'accélérateurs matériels et de la reconfiguration dynamique partielle en permettant des mesures de consommation concrètes sur plusieurs familles de FPGAs Xilinx. Ces modèles sont pensés pour pouvoir être associés à la modélisation multiprocesseur précédente afin d'aboutir à un modèle énergétique multiprocesseur hétérogène réaliste et global qui puisse être utilisé efficacement dans la recherche d'une efficacité énergétique optimale.

3.2.1 Analyse du décodeur H.264 LETI

Le décodeur H.264 considéré ici a été développé par le laboratoire LETI à l'ENIS. Le code C++ a été validé sur différents bitstreams standards pour pouvoir servir de modèle de référence aux étapes de développement suivantes. Une phase d'analyse du code (*profiling*) a permis d'identifier et d'analyser la pertinence de certaines fonctions élémentaires candidates à une accélération matérielle. Les résultats sont visibles à la Figure 5.

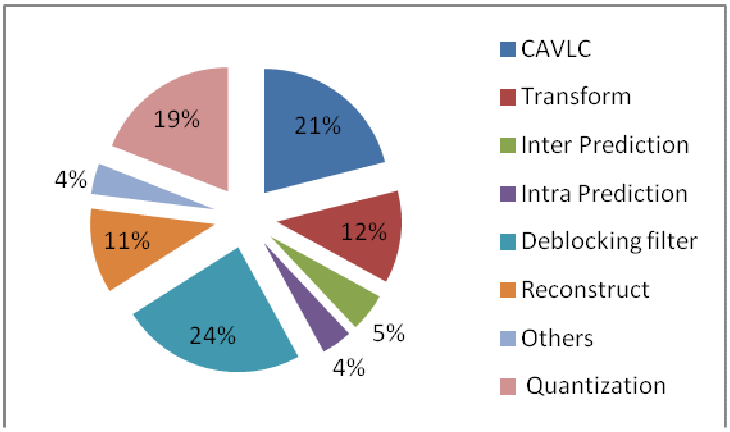


Figure 5 – Profil du décodeur H.264 LETI.

La fonction la plus coûteuse de ce décodeur est le filtre de sortie anti-bloc (*Deblocking filter*) qui représente 24% du temps total de décompression d'image. Viennent ensuite la fonction de décodage entropique CAVLC (*Context Adaptive Variable Length Coding*, 21%), la quantification inverse (19%) et la transformée entière inverse (12%). Le filtre anti-bloc par sa nature fortement calculatoire peut limiter la vitesse de traitement, généralement de 25 ou 30 images par seconde. Le coût et la nature de cette fonctionnalité en

font le premier candidat sur la liste des fonctions candidates à une accélération matérielle possible. Les trois autres fonctions seront elles aussi accélérées, seul l'exemple du filtre sera détaillé ici.

3.2.2 Filtrage anti-bloc

Le filtrage adaptatif utilisé est une technique de post-traitement appliquée à une image reconstruite, à la fois par le décodeur et l'encodeur. Il contribue grandement à améliorer la qualité visuelle pour l'utilisateur et améliore de façon importante le compromis distorsion-performance des applications basées sur H.264/AVC. Le module de filtrage anti-bloc intervient après les étapes de transformée et de quantification inverse dans le processus de reconstruction d'image. Ces deux étapes s'appliquent en effet sur des macroblocs de 16*16 pixels, puis sur des blocs de 4*4 pixels, générant des artefacts au niveau bloc qui affectent la qualité vidéo. Comme illustré à la Figure 6, le filtre anti-bloc est appliqué sur chaque arête d'un bloc 4*4 à l'intérieur et entre les macroblocs 16*16. Il est appliqué conditionnellement pour adoucir les arêtes de bloc horizontales et verticales. Les arêtes verticales sont d'abord examinées de gauche à droite, puis les arêtes horizontales du haut vers le bas. La condition d'application dépend d'une métrique BS (*Boundary Strength*) et du gradient des pixels à chaque frontière. Le filtrage se déroule donc en deux étapes: le calcul de la métrique BS et l'application effective du filtrage.

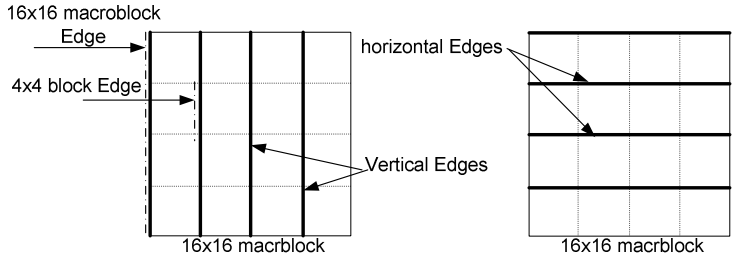


Figure 6 – Arêtes horizontales et verticales d'un macrobloc.

La première étape génère 32 valeurs BS pour 16 arêtes horizontales et 16 arêtes verticales d'un macrobloc. Une valeur BS est ainsi calculée sur 32 bits pour chaque arête gauche ou supérieur d'un bloc 4*4 pour évaluer la nécessité de filtrer et choisir le type de filtre adéquat (*Standard*, *Strong*). L'algorithme de calcul BS dépend de différents paramètres tels que le mode de prédiction, les vecteurs mouvement et blocs de références. Il dépend aussi du type d'arête: bloc ou macrobloc. La seconde étape effectue le filtrage des pixels. Il s'applique sur un macrobloc extrait du buffer d'image en reconstruction en fonction du type de filtre et du paramètre BS calculés précédemment. Les filtres *Standard* et *Strong* sont définis par des expressions mathématiques et appliquées aux pixels d'un macrobloc. Le processus global de filtrage, bien que conditionnel, est répétitif et peut être codé par des structures de boucles déterministes pour couvrir toutes les arêtes et directions d'un macrobloc. La spécification en C++ correspondante a été développée en vue de faciliter une implantation matérielle, en particulier à l'aide d'outils de synthèse RTL. Par la suite, une méthodologie développée sur l'utilisation de l'outil *Catapult C Synthesis* (Mentor Graphics) est utilisée pour permettre l'exécution du décodeur H.264 avec des accélérateurs complètement opérationnels sur des plateformes réelles combinant processeurs et FPGA.

3.2.3 Flot de conception ESL

La méthodologie employée pour obtenir une architecture logicielle / matérielle complète et fonctionnelle se base sur une coopération d'outils de CAO existants: *Catapult C Synthesis*, *Xilinx Platform Studio* et *Mentor Precision Synthesis*. Le filtre anti-bloc constitue le point d'entrée de l'outil de synthèse de haut

niveau (HLS). Une version *Mentor Catapult C Synthesis 2009a Release* est utilisée pour générer automatiquement le code RTL VHDL ciblant les contraintes d'un FPGA Xilinx Virtex-5 XC5VFX70T. Le processeur *PowerPC* intégré, qui peut fonctionner jusqu'à 400MHz, est utilisé pour contrôler l'accélérateur connecté au bus système PLB cadencé à 100MHz. Il permettra également de calculer le gain en performance par rapport à une exécution logicielle. *Xilinx Platform Studio* est utilisé pour assembler les composantes logicielles et matérielles. Un modèle d'interface bus en VHDL (*User logic*), originalement fourni par Xilinx est étendu dans le but d'équiper une IP RTL générée par HLS d'une interface compatible avec le protocole PLB mappée en mémoire. Les étapes de génération de code plus bas niveau se basent sur *Precision Synthesis* pour la synthèse logique de l'IP RTL avec interface PLB, et *Xilinx Platform Studio* pour l'ensemble de la plateforme.

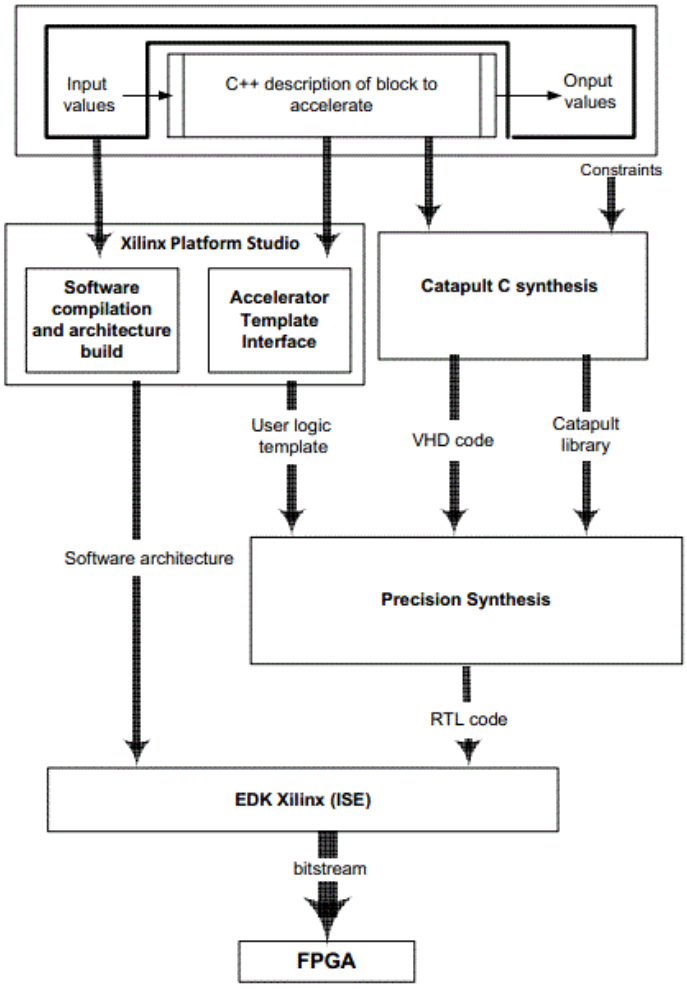


Figure 7 – Flot de conception ESL

L'outil *Catapult* constitue le point central de la méthodologie étant donné que l'accélérateur et son interface bus sont les éléments principaux de l'effort de conception. Sa contribution essentielle est de permettre la génération automatique du code RTL à partir de spécifications en C/C++. Il supporte des options d'optimisation de boucle propres aux outils de HLS tels que la fusion, le déroulage et l'exécution pipeline, qui sont utilisées pour explorer rapidement différents compromis de ressources/performances. Néanmoins, le code source doit se conformer à des contraintes importantes avant d'être synthétisable. D'abord, certaines possibilités du langage C/C++ doivent être éliminées car elles n'ont pas de correspondance simple avec une réalisation matérielle (variables globales, allocations de mémoire

dynamique, certaines utilisations de pointeurs, de structures, de types de données, etc.). Ensuite, certains styles d'écriture se prêtent mieux à une exposition du parallélisme qui puisse être exploité par l'architecture RTL (e.g. boucles déterministes). Enfin, les arguments des fonctions accélérées doivent être compatibles avec un modèle générique d'interface bus défini pour leur mise en œuvre dans des mémoires locales accessibles à la fois par le bus et l'accélérateur. Dans ce modèle, chaque argument de fonction de type tableau est mappé vers une RAM simple port tandis qu'un argument de type scalaire est utilise un registre. A ce stade, les transferts vers les mémoires RAM des arguments doivent être optimisés pour préserver les gains de l'accélération matérielle. Dans le cas du filtre anti-bloc, les pixels se présentent sous forme d'octets de données. Leur transfert natif vers les mémoires accélérateur représente une sous utilisation de 25% du bus PLB (32-bit). Une étape de concaténation (quatre pixels vers un entier 32-bit non signé) est donc nécessaire pour ne pas pénaliser les transferts d'un facteur quatre. Par ailleurs, un module DMA est utilisé pour améliorer encore ces transferts. Les principales sorties de *Catapult* sont des fichiers VHDL au niveau RTL, les librairies associées, un bloc schéma hiérarchique et différents rapports de synthèse avec des métriques utiles concernant les ressources et performances du bloc RTL généré. Celui-ci, sous réserve de compatibilité avec les contraintes d'argument tableaux/scalaires précédents, peut ensuite être encapsulé dans le modèle d'interface bus générique décrit ci-après.

Ciblant une technologie Xilinx, les outils *Xilinx Platform Studio* (XPS) sont utilisés pour réaliser les développements logiciels et matériels. XPS fournit un générateur de code VHDL pour aider l'interfaçage de modules matériels dédiés (*user logic*) avec le bus PLB. Le code généré n'est que partiel et doit être modifié pour correspondre au modèle d'interface bus, en particulier pour supporter l'accès des mémoires et registres (qui recevront les arguments de fonction), soit par le bus pour qu'un pilote puisse envoyer/recevoir les arguments, soit par l'IP qui lit/écrit données. A partir de ce modèle d'interface bus générique, il ne reste qu'à ajouter une instance du code RTL Catapult de l'accélérateur et à connecter toutes les entrées/sorties de l'entité aux RAMs et registres mappés en mémoire du modèle d'interface bus. Du point de vue logiciel, un pilote générique est également développé pour pouvoir remplacer la fonction logicielle originale par une fonction similaire, mais qui transfert les arguments et exécute l'accélérateur. L'utilisation de la topologie d'arguments de fonction définie et des modèles logiciels (pilotes) matériels (interface bus générique) permettent de générer un accélérateur entièrement fonctionnel en l'espace de quelques heures, et le processus de génération de code correspondant pourrait être en grande partie automatisé.

A l'issu de ces étapes d'élaborations logicielles/matérielles, l'IP RTL du filtre anti-bloc est entièrement opérationnel sur le FPGA cible. Par la suite, nous analyserons les performances et l'accélération par rapport à une exécution logicielle du même code. Le processeur PowerPC considéré ici fonctionne à 400MHz, et le bloc matériel connecté au bus PLB fonctionne à 100MHz. Par ailleurs, plusieurs solutions de parallélisme sont considérées en générant huit versions d'accélérateurs ayant chacune des caractéristiques de déroulage de boucle différente (Figure 8). La solution *db_filter_v1* correspond à une version où aucun déroulage n'intervient, la solution *db_filter_v4* correspond à un déroulage complet de toutes les boucles. Il est à noter ici que la génération d'une version de parallélisme nouvelle ne nécessite pas de modifier l'interface bus puisque les arguments de fonction restent inchangés, et peut donc être réalisée très rapidement par une synthèse RTL (Catapult), une synthèse logique (Precision) et une synthèse plateforme (XPS).

Le filtre anti-bloc s'avère être un candidat intéressant à l'accélération matérielle avec une accélération maximale de 55 pour la plateforme PowerPC/Virtex5, utilisant le modèle d'interface bus générique (courbe

opt. spu V5/PPC/PLB). Il est aussi intéressant de constater l'impact quasiment nul du déroulage de boucle sur les performances malgré l'augmentation significative des ressources FPGA (jusqu'à un facteur 2,7), qui s'explique entièrement par l'effet des transferts de données avec l'accélérateur. Ces temps de transfert représentent en effet plus de 95% du temps d'exécution dans le cas du filtre anti-bloc, ce qui limite à la fois l'accélération maximale possible (autour de 55) et l'effet des optimisations au niveau de l'IP RTL pure (hors transferts). Ces accélérateurs ont par la suite été adaptés pour une plateforme Virtex6 / MicroBlaze (courbe *opt. spu V6/MB/PLB*) avec une modification de l'interface bus pour supporter le standard AXI (courbe *opt. spu V6/MB/AXI*). Cet exemple est représentatif de l'importance de communications au niveau bus extrêmement performants et montre l'efficacité du modèle d'interface bus.

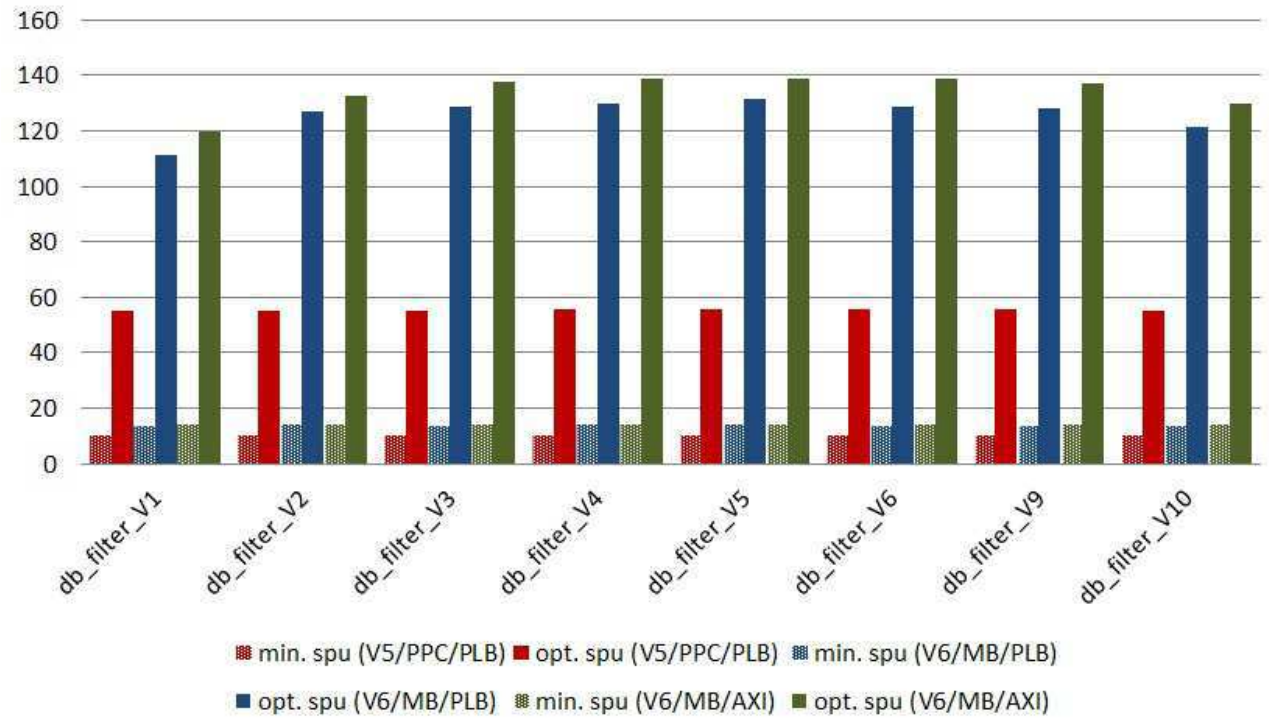


Figure 8 – Accélération du filtre anti-bloc pour différents déroulage de boucle.

3.2.4 Conclusions sur l'exploitation de l'accélération matérielle

La méthodologie d'accélération a été appliquée à trois autres fonctionnalités du décodeur H.264 LETI regroupées en deux blocs matériels: un bloc pour la quantification et transformée entière inverse et un bloc pour le décodage entropique (CAVLC). Elle permet ainsi une accélération effective des trois blocs matériels et du décodeur dans sa globalité d'un facteur 3,3 (PowerPC) et 3,8 (MicroBlaze). Il est à noter que ces résultats pourraient être améliorés en accélérant d'autres fonctions du décodeur.

Cette étude de cas concrète montre donc l'intérêt indéniable de l'utilisation de la synthèse de haut niveau par son automatisation, mais aussi ses limites (modifications du code et génération d'une interface bus) qui sont certainement les raisons principales d'un manque d'adoption général en pratique. La génération de l'interface bus en particulier est un point extrêmement complexe. Cette étape est aussi coûteuse en développement et en pise au point que la synthèse RTL elle-même. Par ailleurs, les transferts de données se faisant vers des mémoires locales de l'accélérateur, ils doivent être fortement optimisés sous peine de voir les bénéfices de l'accélération matérielle réduits, voire perdus. Une solution visant à systématiser la

génération d'une interface bus a été proposée et validée sur trois accélérateurs et une application réelle. Il est à noter que ce travail intervient avant l'intégration de l'outil Vivado HLS à la chaîne d'outils Xilinx, et que Xilinx travaille activement sur ce problème depuis 2014 avec la série d'environnements « SDx » (SDNet, SDAccel, SDSoC), mais dont nous n'avons pas encore pu faire une évaluation notamment concernant le problème de génération d'interface matériel et logiciel car cet outil est en version de test par une poignée d'utilisateurs sélectionnés.

La méthodologie développée précédemment a été très activement utilisée dans les travaux qui sont abordés par la suite, en particulier pour développer une modélisation énergétique d'unités reconfigurables et d'unités reconfigurables dynamiquement, qui constituent des techniques prometteuses pour l'amélioration significative de l'efficacité énergétique.

3.3 Reconfiguration Dynamique Partielle

Minimiser les ressources et la consommation énergétique sont deux contraintes caractéristiques dans la conception de systèmes de traitement embarqués et alimentés sur batterie. La Reconfiguration Dynamique Partielle (RDP) est une technique associée aux composants FPGA récents qui ouvre des possibilités intéressantes par sa capacité à réduire à la fois le temps d'exécution par l'accélération matérielle, et les ressources pour des fonctionnalités caractérisés par du calcul intensif. Néanmoins, la reconfiguration partielle induit des interactions complexes qui rendent les gains énergétiques très difficiles à analyser. En particulier, il est essentiel de pouvoir quantifier le coût des pertes énergétiques associées car le processus de reconfiguration engendre lui-même une consommation. Cette section décrit une étude approfondie des coûts énergétiques et en puissance associés aux différents mécanismes impliqués dans le processus de RDP. Ils entrent dans le cadre de la thèse de Mr. Robin BONAMY menés dans le projet Open-PEOPLE (*Open-Power and Energy Optimization PLatform and Estimator,* ANR).

3.3.1 Procédure expérimentale

Une plateforme Xilinx ML550 Virtex5 LXT intégrant des points de mesures accessibles est utilisée pour toutes les expérimentations qui suivent. Cinq connecteurs sont présents sur cette carte pour permettre de mesurer le courant consommé par le FPGA et ses périphériques. La mesure de la puissance FPGA se base sur celle des rails d'alimentation du cœur. Un amplificateur haute précision sur mesure est utilisé pour améliorer le niveau du signal qui peut ensuite être observé à l'aide d'un oscilloscope numérique avancé. Grâce à cette procédure, il est possible de mesurer des valeurs de puissance de l'ordre de 0,1 mW. Cette précision est nécessaire (et suffisante) pour permettre d'identifier clairement les différentes étapes du processus de reconfiguration dynamique partielle.

La plateforme ML550 est configurée avec le projet de référence Xilinx décrit dans [5], composé d'un processeur *MicroBlaze*, un contrôleur mémoire pour la *Compact_Flash* et un contrôleur de reconfiguration *xps_hw_icap*. Le processeur *MicroBlaze* et le contrôleur de reconfiguration opèrent tous les deux à la fréquence de 100MHz. Les requêtes de reconfiguration sont gérées par le *MicroBlaze* qui lit le fichier de configuration (*bitstream*) dans la *Compact_Flash* et l'envoie au contrôleur *xps_hw_icap* pour appliquer une configuration à la zone reconfigurable (*Partial Reconfiguration Region*, PRR) concernée.

Task	BRAMs	Slices	DSPs	Idle power (mW)
T_1	8	169	2	26
T_2	8	154	1	24
T_{blank}	0	0	0	0
FPGA without task				402

Tableau 3 – Ressources FPGA et consommation à vide des tâches utilisées.

Comme plusieurs tâches différentes seront utilisées dans les expérimentations suivantes, une PRR compatible avec la taille de la plus grosse tâche a été définie. Elle a par ailleurs été choisie dans la partie haute et supérieure du plan de masse du FPGA et occupe une surface complète couvrant deux domaines

d'horloge, et correspond à un *bitstream* de configuration d'une taille fixe de 227700 octets. Lors d'une mesure, le cœur FPGA est alimenté à 1V et on s'assure de la stabilité de la température du composant autour de 35°C afin de ne se focaliser que sur la consommation engendrée par la RDP. Les mesures sont effectuées avec la PRR configurée pour trois tâches possibles. La première tâche T_1 est une multiplication de matrice. La seconde tâche T_2 est une version parallélisée de la multiplication de matrice. Les deux implémentations RTL de la multiplication de matrice sont générées par un outil Synthèse de Haut Niveau avec différents paramètres de déroulage de boucle, dont les ressources FPGA sont données au Tableau 3. La troisième tâche est un *bitstream* « blanc », dénommé T_{blank} par la suite. Un *bitstream* blanc correspond à la configuration d'une PRR vide. Cette possibilité permet d'effacer la configuration d'une PRR lorsqu'elle est inutilisée, diminuant ainsi sa consommation. Par la suite nous analysons comment la configuration d'une tâche vers une autre affecte la consommation du cœur FPGA.

3.3.2 Analyse de la consommation

Le processus global de RDP consiste principalement à transférer les données vers les différents composants impliqués (*xps_hw_icap* et mémoire de configuration). La procédure complète peut se décomposer de la façon suivante, les données de configuration sont successivement :

- Lues dans un fichier stocké dans la *Compact_Flash* et mises en mémoire locale BRAM du *MicroBlaze*.
- Ecrites depuis la mémoire locale BRAM du *MicroBlaze* versle contrôleur de reconfiguration *xps_hw_icap*.
- Ecrites depuis *xps_hw_icap* vers le port d'accès de configuration interne (*Internal Configuration Access Port*, ICAP).
- Ecrites depuis l'ICAP vers la mémoire de configuration.
- Appliquées depuis la mémoire de configuration aux ressources configurables (Slices, BRAM, DSP, interconnections).

Dans ces conditions, la puissance globale consommée par la reconfiguration est le résultat d'une combinaison de deux éléments principaux : (i) le contrôle de la reconfiguration qui implique principalement des accès aux données de configuration et (ii) la configuration effective des ressources FPGA.

Le haut de la Figure 9 montre les profils de consommation du cœur FPGA lors d'une reconfiguration de T_2 vers T_1 (noir) et de T_1 vers T_2 (gris). En analysant de près le processus de reconfiguration complet sur cette figure, on peut observer les variations de puissance dues à la RDP. La courbe du bas est un zoom sur les dix premières millisecondes. Pour une analyse plus poussée, des marqueurs déclenchés de manière logicielle ont été introduits au niveau du contrôle de la reconfiguration afin de faire ressortir les étapes du processus. Sept étapes sont ainsi clairement identifiées :

1. Réception de l'ordre de reconfiguration
2. Ouverture du fichier de configuration sur la *Compact_Flash*.
3. Lecture de l'en-tête du fichier de configuration.

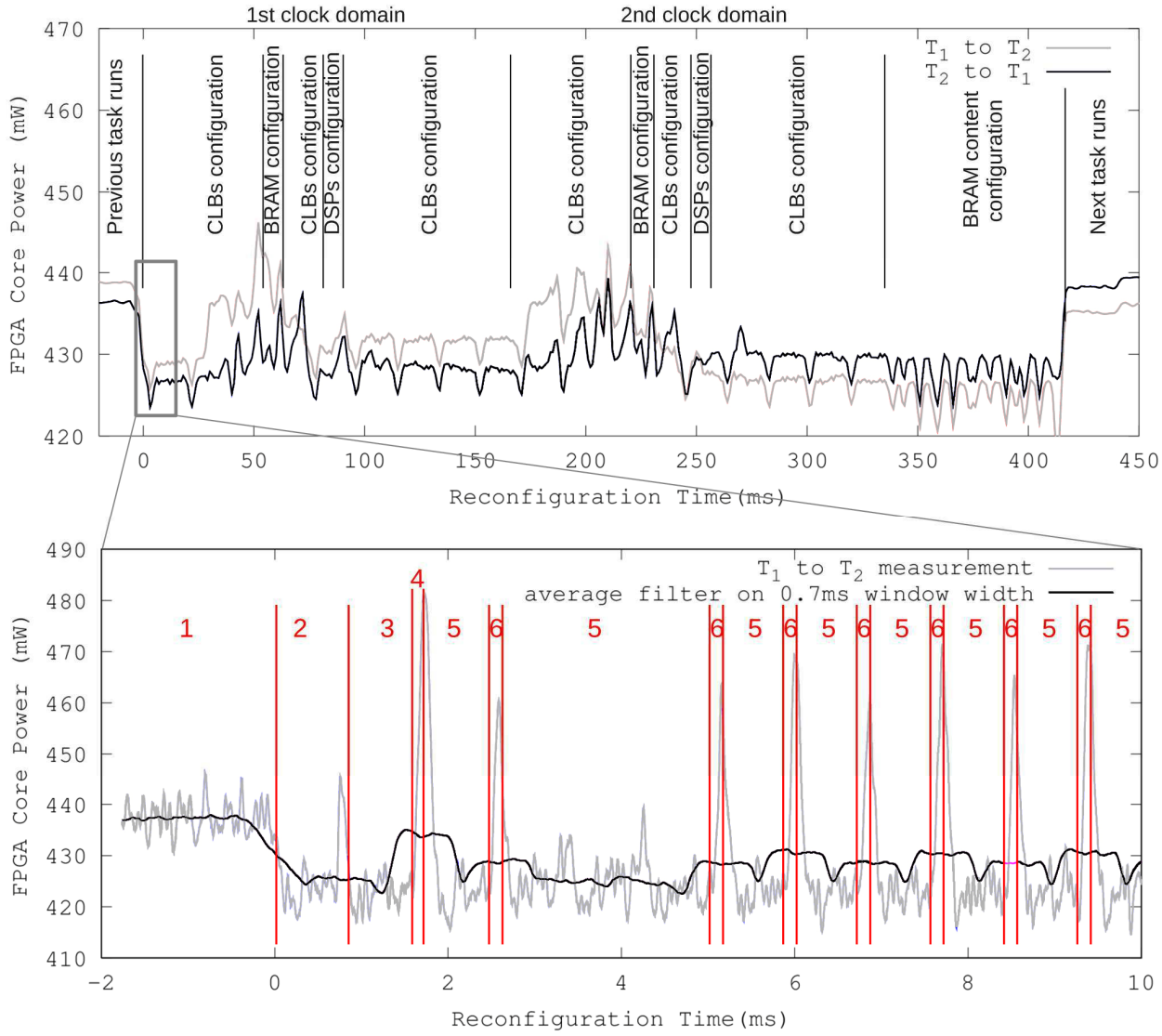


Figure 9 – Haut de la figure: consommation du cœur FGPA pour une reconfiguration de T_2 à T_1 (noir) et de T_1 à T_2 (gris). La composition du *bitstream* est mise à l'échelle pour correspondre au temps de reconfiguration et faire ressortir les étapes de la reconfiguration. Le bas de la figure est un zoom sur 10ms au début de la reconfiguration.

4. Vérification de la validité du fichier de configuration.
5. Lecture d'un fragment de fichier sur la *Compact_Flash*.
6. Ecriture des données de configuration dans *xps_hw_icap*.
7. Répétition des étapes 5 et 6 jusqu'à la fin du fichier

On remarque que les niveaux les plus bas des profils de puissance de la Figure 9 (courbe du bas) correspondent à des accès en lecture à la *Compact_Flash* (phases 3 et 5). A cause de la relative lenteur de la *Compact_Flash*, le processeur *MicroBlaze* (qui a la part de consommation la plus importante du FPGA) est la majeure partie du temps en attente des données et ne génère qu'une faible activité. Néanmoins l'écriture (phase 6) engendre plus d'activité par du trafic sur le bus et par le contrôle et la configuration des

ressources FPGA. Le coût en puissance correspondant est d'environ 45 mW, ce qui représente plus de 10% de la consommation totale du FPGA. Il existe un autre pic de consommation lors de la vérification de la validité du bitsream (phase 4) Cette surconsommation est également significative car elle résulte d'une activité du processeur *MicroBlaze*. Ces pics ne sont pas visibles dans la courbe supérieure de la Figure 9 à cause d'un filtrage des données permettant un meilleur affichage. Finalement, la lecture des données de configuration depuis la *Compact_Flash* et leur écriture dans *xps_hw_icap* sont répétées jusqu'à la fin du bitstream.

L'analyse précédente démontre l'influence des transferts des données de configuration sur la consommation. Néanmoins, les effets des opérations de lecture et d'écriture n'apportent pas de variations significatives au profil global de consommation (haut de la Figure 9). Si on applique une moyenne fenêtrée sur 0,7 ms (courbe noire, bas de la Figure 9), le profil résultant est régulier et quasi constant. Or le profil n'est pas régulier sur la durée complète du processus de reconfiguration: premièrement, deux surconsommations interviennent autour de 50 ms et 200 ms; deuxièmement, les niveaux de puissance au début et à la fin de la reconfiguration sont différents. Cette différence de consommation semble être liée à la transition entre la tâche précédente et la suivante.

Ces deux effets indiquent que les profils de puissance dépendent aussi d'autres paramètres que l'accès aux données de configuration. On suppose par la suite que les variations observées sont aussi dues au contenu du bitstream, en distinguant deux effets : les « vagues » et les « sauts ». Une reconfiguration peut causer des surconsommations dues à l'activité résultant des différences avec la configuration précédente (vagues de consommation). Puis l'application de la configuration active ou désactive des signaux et ressources qui peuvent causer des non linéarités (sauts de puissance).

Dans le profil de consommation de la Figure 9 montrant une reconfiguration de T_2 vers T_1 (noir) et de T_1 vers T_2 (gris), les deux courbes ont une allure générale similaire et qui est principalement générée par les opérations d'écriture dans la mémoire de configuration. En pratique, lorsqu'une tâche est configurée à partir d'une tâche précédente, la reconfiguration consiste à ré-écrire les données sur le contenu existant de la mémoire de configuration. Si les deux nièmes mots des bitstreams de la tâche précédente et suivante sont identiques, le contenu mémoire correspondant ne change pas et la consommation reste faible. En revanche si ces deux mots sont exactement complémentaires, chaque bit du contenu mémoire change ce qui induit une forte consommation. A partir de cette observation, on peut supposer que la consommation pendant le processus de reconfiguration est liée aux différences entre le bitstream de la tâche précédente et celui de la tâche suivante. Ces différences peuvent être caractérisées par une métrique comme la distance de Hamming.

Sur les deux courbes de la Figure 9, deux zones remarquables sont présentes: la première se situe autour de 50 ms et la seconde autour de 200 ms. L'amplitude de ces surconsommations est d'environ 15 mW. L'analyse du contenu du bitstream indique que ces deux zones correspondent à la configuration de deux groupes de BRAMs. Ces BRAMs appartiennent à deux domaines d'horloge distincts qui sont à l'origine des surconsommations. Par corrélation avec une visualisation du plan de masse par les outils Xilinx (*PlanAhead*), on s'aperçoit que la densité des cellules logiques utilisées (*slices*) augmente et est plus importante à proximité de ces groupes de BRAMs. Ce placement de ressources logiques autour des blocs BRAMs s'explique par les algorithmes de placement et routage qui tendent à regrouper les ressources logiques utilisées afin de limiter le coût en interconnexions.

Le haut de la Figure 10 montre le profil de reconfiguration d'une PRR de T_2 à T_1. Le bas de la figure montre la distance de Hamming par mot de configuration 32-bit calculé entre les deux bitstreams concernés.

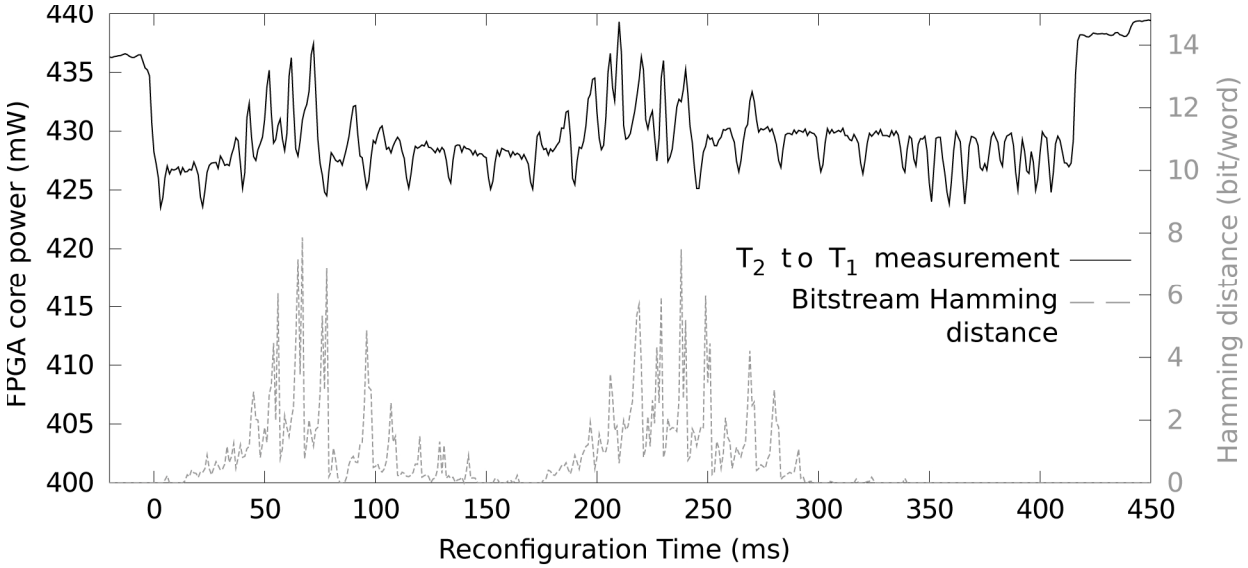


Figure 10 – Consommation lors d'une RDP de T_2 vers T_1 (noir) et distance de Hamming par mot de configuration 32-bit entre les *bitstreams* de T_2 vers T_1 (gris).

L'axe des abscisses est passé à l'échelle pour correspondre au déroulement temporel de la reconfiguration. On remarque sur ces courbes que l'allure de la distance de Hamming est fortement corrélée à celle du profil de consommation. La distance de Hamming atteint deux pics très voisins des surconsommations observées, autour de 50 ms et 200 – 250 ms. Ceci tend à confirmer le lien entre les différences de configuration et la présence de surconsommations.

En analysant d'avantage la Figure 9, on peut noter que les niveaux de puissance au début et à la fin de la reconfiguration sont différents. Cette différence est cohérente avec la consommation des tâches présentées au Tableau 3 où la puissance globale est de 402 mW pour le FPGA à vide, et 428 mW ou 426 mW respectivement quand les tâches T_1 ou T_2 sont configurées. T_1 et T_2 n'ont pas la même puissance au repos (puissance *idle*) qui dépend de leur taille. Logiquement, l'utilisation de deux tâches ayant des niveaux de puissance idle très différents devrait confirmer cette hypothèse. La Figure 11 montre ainsi le profil de puissance obtenu pour une reconfiguration dynamique partielle d'une PRR de T_{blank} vers T_2, où T_{blank} est une tâche vide comme défini précédemment. Cette figure montre explicitement deux sauts en puissance à 55 ms et 220 ms. Ces sauts élèvent progressivement le niveau de puissance depuis celui de la tâche précédente (T_{blank}) jusqu'à celui de la tâche suivante (T_2). La Figure 11 fait également apparaître la composition du bitstream correspondant pour un composant Virtex5, ce qui révèle que les sauts apparaissent juste avant une étape de configuration de BRAMs. Ce phénomène est typique d'une activation/désactivation d'éléments et suggère qu'il existe certainement un lien entre la consommation des mémoires BRAMs et leur état actif associé.

Nous avons ainsi identifié deux éléments importants pour une modélisation pertinente de la RDP en consommation. Il existe tout d'abord une contribution du contrôle de reconfiguration impliquant

principalement des accès aux données de configuration, puis une contribution générée par l'application effective de la reconfiguration.

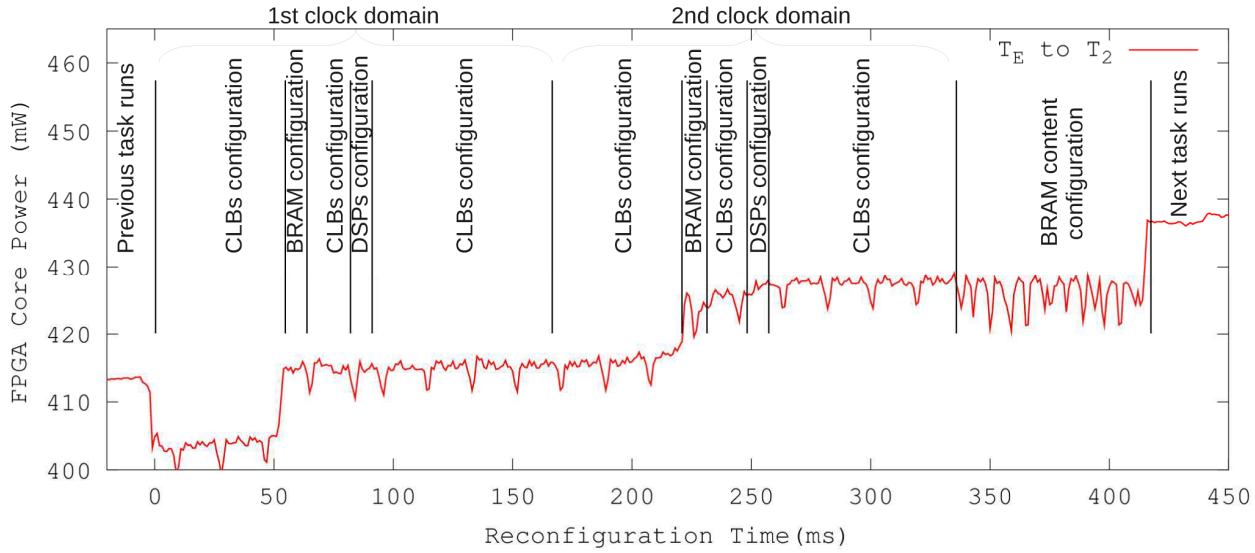


Figure 11 – Consommation lors de la RDP de T_{blank} vers T_2 et composition du *bitstream* correspondant.

Cette application de la reconfiguration fait apparaître des vagues de consommation, résultant des différences entre les configurations précédentes et suivantes, et des sauts de puissance qui résultent de l'activation de nouvelles ressources dans la configuration de la tâche suivante. A partir de ces contributions, nous pouvons établir plusieurs modèles de consommation de la RDP avec différents niveaux de précision, et qui sont présentés par la suite.

3.3.3 Estimation de la consommation

La façon la plus sûre d'évaluer la consommation d'une PRR consiste à effectuer plusieurs mesures d'une même configuration et d'en déduire une moyenne. La puissance idle peut ensuite être soustraite de cette valeur pour ne garder que la composante due à la seule reconfiguration. Elle peut ainsi être considérée comme la consommation qui résulte du contrôle de la reconfiguration et sera notée $P_{control}$ par la suite. La consommation idle du FPGA avant reconfiguration P_{FPGA} est définie par la consommation idle du FPGA quand la configuration de la PRR est effacée (*blank*). La puissance idle de la PRR configurée précédemment P_{prev} est obtenue par soustraction de la puissance globale mesurée quand T_{prev} est configurée et de P_{FPGA}. Dans ces conditions, un premier modèle d'estimation gros grain (*Coarse Grained*, CG) est donné par l'expression suivante:

$$P_{CG} = P_{FPGA} + P_{prev} + P_{control}$$

Ce modèle présente l'avantage d'être très simple à définir, mais des écarts importants peuvent se produire s'il existe une différence importante entre les niveaux de puissance idle des tâches précédentes et suivantes sur une PRR.

Une amélioration du modèle précédent peut être faite en réalisant une interpolation linéaire entre les puissances idle avant et après la reconfiguration. Elle nécessite de connaître la puissance idle des tâches précédentes et suivantes configurées, ainsi que la puissance du contrôle de reconfiguration. Le modèle d'estimation grain moyen (*Medium Grained*, MG) correspondant est donné par :

$P_{MG}(\tau) = P_{FPGA} + P_{prev} + P_{control} + \tau*(P_{next} - P_{prev})/(B_{Ssize}*T_{word})$

où $0 < \tau \leq B_{Ssize}*T_{word}$

τ est une unité de temps correspondant à l'accès en lecture d'un mot de configuration 32-bit, BS_{size} est la taille en octets du bitstream de la PRR configurée, T_{word} est le temps nécessaire à la configuration d'un mot 32-bit et P_{next} est la puissance idle de la PRR configurée avec la nouvelle tâche. L'erreur d'estimation devrait être moins importante que précédemment puisque le modèle intègre les puissances à vide des deux tâches. Néanmoins, ce modèle ne considère pas le contenu du bitstream qui génère les sauts de puissance évoqués précédemment. Un troisième modèle grain fin (*Fine Grained*, FG) est donc défini en tenant compte de la distance de Hamming.

Ce dernier modèle de reconfiguration est basé sur les mêmes paramètres précédents, mais le profil défini dans ce cas évolue par paliers qui dépendent du contenu du bitstream. De plus, les phénomènes de vagues sont pris en compte par la distance de Hamming entre les mots de configuration des deux bitstreams. Le modèle résultant implique l'utilisation de paramètres avancés dont la définition est détaillée dans [6]. Il est défini par l'équation suivante :

$P_{FG}(\tau) = P_{FPGA} + P_{prev} + P_{control} + steps(\tau)*(P_{next} - P_{prev}) + \alpha*d_{Hamming}(\tau, T_{prev}, T_{next})$

où $steps(\tau)$ est une fonction dont le résultat prend une valeur entre 0 et 1 qui dépend du FPGA et de la taille de la PRR. Dans nos mesures, cette fonction est liée à la position des BRAMs. Les valeurs sont déterminées à partir des ressources de la PRR, de la taille du bitstream et de la vitesse de reconfiguration. $d_{Hamming}(\tau, T_{prev}, T_{next})$ est la distance de Hamming entre les configurations précédentes et suivantes d'une PRR appliquée sur les mots de configuration 32-bit. Enfin α est un coefficient d'ajustement pour un FPGA donné qui reflète le poids de la distance de Hamming sur la consommation. La section suivante compare la précision des trois modèles définis avec les résultats de mesures réelles, à la fois en termes d'énergie et de profils de puissance.

3.3.4 Validation des modèles

Les profils de puissance des trois modèles précédents sont représentés Figure 12, Figure 13, Figure 14 et Figure 15. Les lignes continues représentent les profils mesurés, les lignes utilisant des marqueurs 'x', '+' et 'o' sont les traces de puissance estimées des modèles gros grain, grain moyen et grain fin respectivement.

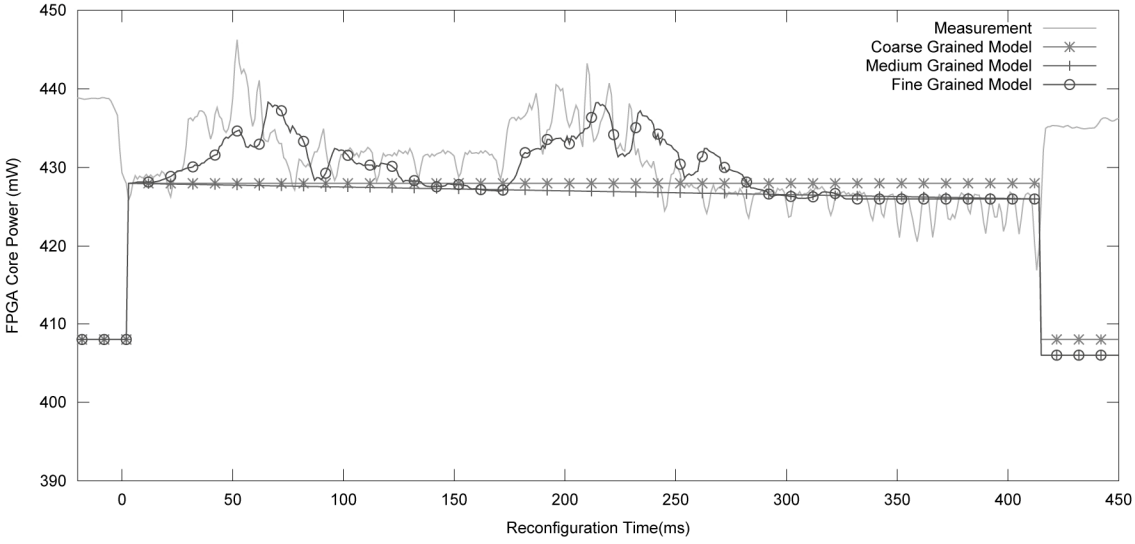


Figure 12 – Comparaison des modèles RDP et des mesures pour une reconfiguration de T_1 à T_2.

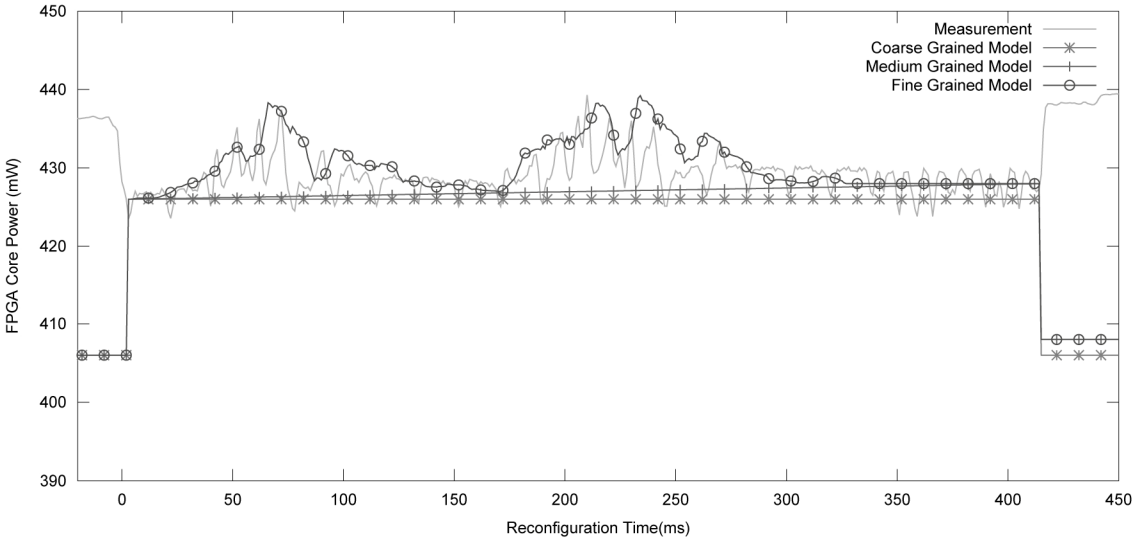


Figure 13 - Comparaison des modèles RDP et des mesures pour une reconfiguration de T_2 à T_1.

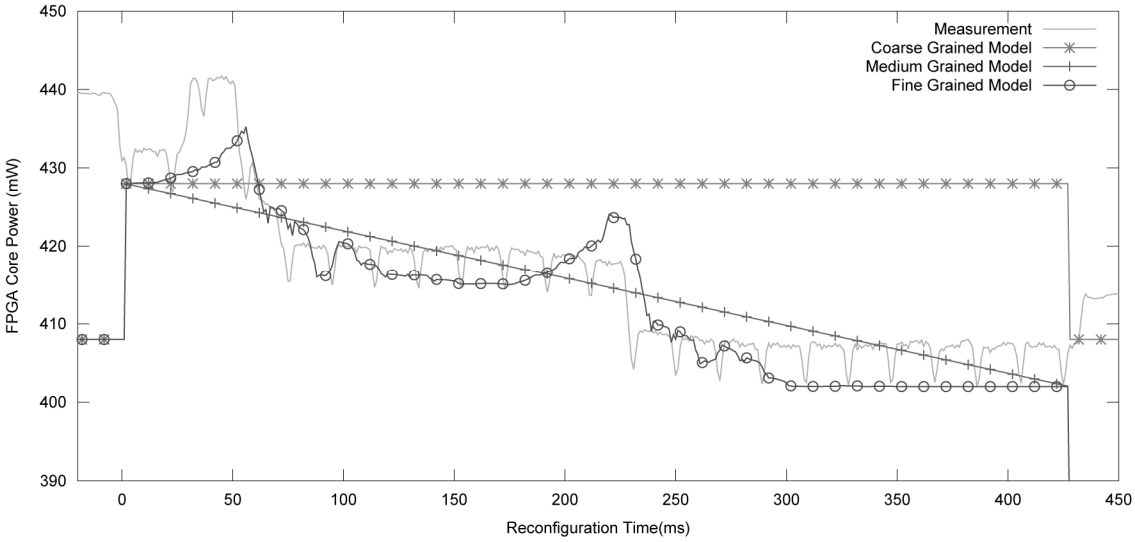


Figure 14 - Comparaison des modèles RDP et des mesures pour une reconfiguration de T_1 à T_{blank}.

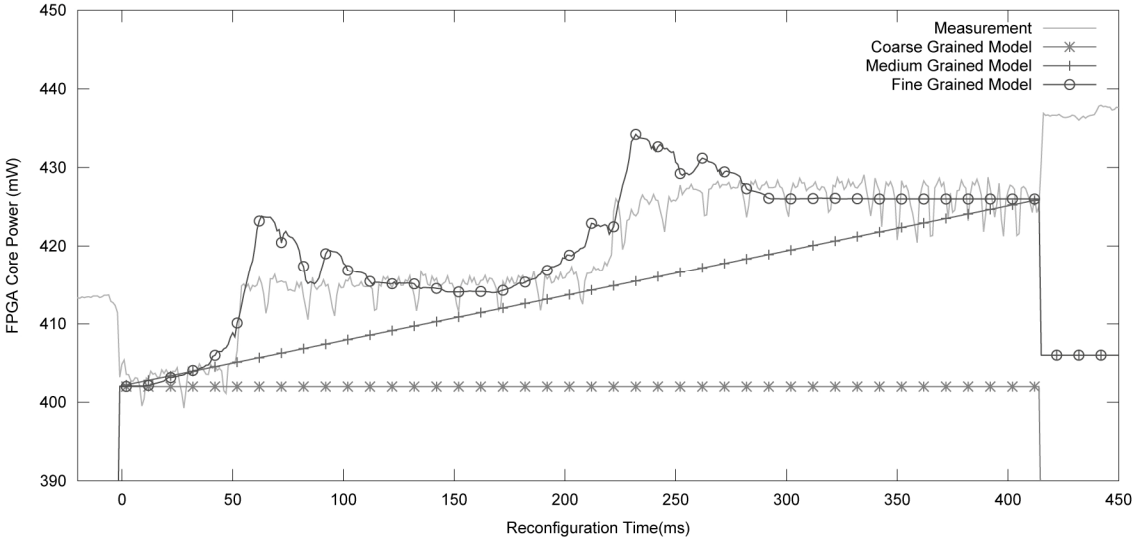


Figure 15 - Comparaison des modèles RDP et des mesures pour une reconfiguration de T_{blank} à T_2.

Quatre scénarios de reconfiguration sont considérés : reconfiguration de T_1 à T_2, de T_2 à T_1, de T_1 à T_{blank} et de T_{blank} à T_2. La précision de chaque modèle est d'abord évaluée en termes de consommation énergétique en calculant l'erreur moyenne absolue et l'erreur quadratique moyenne (Tableau 4). Puisque nous utilisons la même PRR, les temps de reconfiguration sont identiques dans les quatre cas et l'énergie est comparable à la puissance moyenne.

Config.	M (mJ)	CG error (mJ)	MG error (mJ)	FG error (mJ)
T_2 to T_1	9,12	-1,3 (-13,7%)	-0,84 (-9,2%)	0,5 (5,5%)
T_1 to T_2	9,58	-0,89 (-9,2%)	-1,3 (-13,5%)	-0,22 (-2,3%)
T_1 to T_{blank}	7,97	6,2 (77%)	0,59 (7,4%)	-0,13 (-1,7%)
T_{blank} to T_2	10,41	-7,1 (-67,9%)	-2,1 (-19,7%)	0,5 (5%)
Average	9,27	-0,77 (-8,3%)	-0,9 (-9,8%)	-0,16 (-1,8%)
RMSE		4,77 (51,5%)	1,34 (14,5%)	0,38 (4,0%)

Tableau 4 – Erreur moyenne absolue et erreur quadratique moyenne des estimations énergétiques pour les modèles gros grain (CG), grain moyen (MG) et grain fin (FG) comparés aux mesures réelles (M).

Comme on peut le constater sur les profils de T_1 à T_2 (Figure 12) et de T_2 à T_1 (Figure 13), les modèles gros grain et grain moyen donnent de bons résultats avec moins de 14% d'erreur en énergie. L'erreur est faible dans ce cas parce que la différence de puissance idle entre T_1 et T_2 est faible (2 mW). Si on considère une différence plus importante, par exemple pour la reconfiguration de T_1 à T_{blank} (Figure 14) ou de T_{blank} à T_2 (Figure 15), le modèle gros grain est beaucoup moins précis avec respectivement 77% et 68% d'erreur en énergie. La plus grande part de l'imprécision résulte de l'absence de la puissance idle des tâches précédentes et suivantes dans ce modèle. En outre, le modèle grain moyen réduit l'erreur à 20% en tenant compte de cette puissance idle. Finalement, le modèle grain fin fournit les meilleurs résultats avec moins de 6% d'erreur dans tous les cas, en intégrant les effets de vagues et de sauts dans le modèle de reconfiguration. On observe ainsi une forte similitude entre le profil grain fin estimé et les mesures réelles dans les quatre cas considérés.

Conjointement à l'analyse de la précision des estimations énergétiques, la capacité des profils de puissance à prédire une trace correcte constitue un second critère de qualité important des modèles. Les variations des modèles de puissance par rapport aux mesures peuvent être évaluées en calculant l'écart des modèles avec la réalité pour chaque échantillon, puis en prenant la moyenne de ces écarts. Les valeurs de ces variations sont données au Tableau 5 pour chaque modèle et scénario de reconfiguration. Les résultats montrent que le modèle gros grain présente un écart important de 6,5 mW, ce qui représente 17,6 % de la puissance moyenne de reconfiguration. Cette erreur résulte des effets de vagues et de sauts qui ne sont pas pris en considération. Le modèle à grain moyen réduit l'écart moyen à 4,05 mW (10,5%) en tenant compte de manière simple des sauts de puissance. Finalement, le modèle à grain fin est légèrement meilleur avec une déviation moyenne de 3,75 mW, soit 9,6% de la puissance moyenne de reconfiguration. Il est à noter ici une relative mésestimation des phénomènes de vagues dans ce cas qui semble être accentué

par la différence des niveaux de puissance résultant de l'utilisation de T_{blank} que l'on peut clairement observer à la Figure 14 et à la Figure 15.

Config.	CG (mW)	MG (mW)	FG (mW)
T_2 to T_1	2,7 (5,8%)	2,7 (5,8%)	3,2 (6,9%)
T_1 to T_2	4,9 (10,2%)	4,6 (9,5%)	3,8 (7,9%)
T_1 to T_{blank}	10 (31,8%)	5 (16%)	4,3 (13,5%)
T_{blank} to T_2	8,2 (22,4%)	3,9 (10,7%)	3,7 (10%)
Average	6,5 (17,6%)	4,05 (10,5%)	3,75 (9,6%)

Tableau 5 – Ecart des modèles gros grain (CG), grain moyen (MG) et grain fin (FG) par rapport aux mesures.

De plus, il existe d'autres variations plus fines de la puissance qui ne sont pas estimées et contribuent à augmenter l'erreur. Néanmoins, les valeurs pic sont présentes et l'allure générale de la trace en puissance est globalement assez bien estimée.

3.3.5 Pertinence des modèles

Les trois modèles de RDP proposés sont utiles pour ajuster la précision des estimations au niveau d'analyse et de détail nécessaire. Dans tous les cas, le modèle à grain fin est préférable en terme de précision, mais ce n'est pas toujours le meilleur choix en pratique par l'importante complexité de mise en œuvre qu'il nécessite (distance de Hamming, fonction step(τ) et paramètres de calibration). De plus, la précision du modèle grain fin n'est pas utile dans de nombreux cas. En particulier, la modélisation fine des détails devient secondaire lorsque le surcoût engendré par la reconfiguration est optimisé et devient négligeable devant les temps d'exécution et puissances des tâches matérielles (ce qui représente également une condition d'efficacité énergétique de la RDP). Ainsi le choix d'un modèle est déterminé par i) l'existence d'un modèle grain fin pour un composant ou une technologie, et ii) un compromis entre précision et complexité d'élaboration du modèle. En l'absence d'un modèle grain fin, les modèles gros grain et grain moyen sont plus faciles à utiliser et assez pertinents pour fournir des estimations énergétiques et profils de puissance justes dans un grand nombre de cas de figures.

Il est important de souligner que les performances du contrôleur de reconfiguration sont déterminantes pour une utilisation valable de la RDP. Dans la procédure expérimentale de cette étude basé sur un projet de référence Xilinx, les performances de la reconfiguration dépendent de l'utilisation du contrôleur de reconfiguration *xps_hw_icap*, du processeur *MicroBlaze* et d'une mémoire *Compact_Flash*. De bien meilleures performances sont possibles en utilisant un environnement matériel optimisé tel que [7] et [8]. Dans [8] par exemple, la reconfiguration est beaucoup plus rapide mais les variations de puissance sont beaucoup plus faibles et difficiles à mesurer à cause des limitations des sondes de courants existantes. Les contraintes de mesures sont donc principalement les raisons du choix du contrôleur de reconfiguration *xps_hw_icap* utilisé dans cette étude. Néanmoins, nous avons également participé au développement de contrôleurs de reconfiguration optimisés avec une connaissance précise des mécanismes et techniques utilisées. Les étapes qui interviennent dans un contrôleur amélioré sont essentiellement identiques, les modèles précédents peuvent donc être appliqués. L'utilisation d'un contrôleur optimisé impacte

principalement les paramètres des équations et dépend du modèle de RDP concerné. Pour le modèle gros grain, la puissance du contrôleur de reconfiguration $P_{control}$ est le seul paramètre qui est relativement simple à établir. La vitesse du contrôleur T_{word} doit aussi être spécifiée pour le modèle à grain moyen. En ce qui concerne le modèle à grain fin, une reconfiguration plus rapide pourrait diminuer les pics du courant d'alimentation à cause de l'effet capacitif des rails d'alimentation. Ce phénomène peut avoir un effet sur la prise en compte de la distance de Hamming et nécessiter un ajustement de la moyenne fenêtrée pour corriger l'amplitude des pics de puissance. Une déviation relative peut donc se produire pour l'utilisation d'un contrôleur de reconfiguration optimisé, mais dans ce cas elle n'interviendrait que pour le modèle à grain fin.

Cette étude étend la modélisation de plateformes multiprocesseur à la reconfiguration dynamique partielle. Elle pose également les bases d'un modèle réaliste pour les accélérateurs reconfigurables tiré des nombreuses mesures effectuées sur des exemples concrets. Les travaux suivants décrivent cet effort qui est pensé dans le souci d'établir un modèle global et appliqué à un cas d'étude réel avec le décodeur H.264 LETI. Ils se déclinent par une première approche simple plutôt centrée sur l'établissement d'un modèle formel (PRE-Explorer), et une seconde approche d'exploration plus méthodique de la RDP développée au LEAT (FoRTReSS).

3.4 Méthodologies pour la conception et le déploiement

Les différentes modélisations étudiées précédemment fournissent une base utile assez complète pour permettre l'étude et la définition d'approches de conception et de déploiement au niveau système qui soit efficaces et adaptées aux contraintes des applications et architectures des *SoCs* hétérogènes actuelles et à venir. Les travaux détaillés par la suite décrivent ainsi deux méthodologies couvrant la conception d'architectures multiprocesseur hétérogènes et l'analyse du déploiement d'applications, qui ont été développées au cours de projets collaboratifs et s'appuient très fortement sur les contributions précédentes, en particulier en termes de modèles énergétiques. Le projet ANR Open-PEOPLE (*Open-Power and Energy Optimization PLatform and Estimator*) est d'abord concerné par des extraits et résultats de travaux de thèse de Mr. Robin BONAMY publiés dans des revues internationales. La poursuite récente de ces recherches est ensuite abordée à travers la contribution à l'environnement FoRTReSS (*Flow for Reconfigurable archiTectures in Real-time SystemS*) en intégrant des aspects énergétiques avancés en étroite collaboration avec mon collègue Fabrice MULLER au LEAT.

3.4.1 Formalisme

Les travaux précédents sur la modélisation de la reconfiguration dynamique partielle ont permis d'identifier les composantes essentielles de la consommation d'accélérateurs matériels reconfigurables: la consommation à vide (FPGA, PRRs), la consommation au repos et en activité des tâches, le contrôleur de reconfiguration. Ce modèle est étendu ici pour pouvoir supporter la caractérisation d'une architecture multiprocesseur hétérogène complète et réaliste, d'une application et de ses caractéristiques en termes de déploiement.

Variable	Range	Definition	
N^{core}	$\in \mathbb{N}^*$	Number of software execution units	Platform topology
N^{PRR}	$\in \mathbb{N}^*$	Number of hardware execution units	
N^{EU}	$= N^{PRR} + N^{core}$	Total number of execution units	
EU_j	$\forall j = 1, ..., N^{EU}$	The j^{th} execution unit	
SoC	$= \{EU_j\}$	A platform is a set of execution units	
N_j^{freq}	$\in \mathbb{N}^*$	Number of frequencies for software EU_j	Execution units
$F_{j,k}$	$\forall k = 1, ..., N_j^{freq}$	The k^{th} frequency of software EU_j	
$P_{j,k}^{empty}$	$\in \mathbb{R}^+$	Empty power consumption for software EU_j at $F_{j,k}$	
$P_{j,k}^{run}$	$\in \mathbb{R}^+$	Running power consumption for software EU_j at $F_{j,k}$	
N_j^{cell}	$\in \mathbb{N}^*$	Number of logic cells for hardware EU_j	
N_j^{bram}	$\in \mathbb{N}^*$	Number of RAM blocks for hardware EU_j	
N_j^{dsp}	$\in \mathbb{N}^*$	Number of DSP blocks for hardware EU_j	
P_j^{empty}	$\in \mathbb{R}^+$	Empty power consumption for hardware EU_j	
T^{1cell}	$\in \mathbb{R}^+$	Time required to reconfigure one logic cell	DPR
E^{1cell}	$\in \mathbb{R}^+$	Energy required to reconfigure one logic cell	

Tableau 6 – Paramètres du modèle d'architecture hétérogène

La topologie générale considérée est celle d'une architecture hétérogène qui peut être composée de cœurs logiciels et d'accélérateurs reconfigurables dynamiquement. Chaque type d'unité d'exécution est formalisé

par un ensemble de paramètres spécifiques qui capturent les informations nécessaires pour l'analyse du déploiement. Ces paramètres peuvent être classés en trois catégories (Tableau 6) : topologie de la plateforme, unités d'exécution et reconfiguration dynamique partielle. La description topologique divise les unités d'exécution en deux catégories : un nombre N^{core} d'unités d'exécution logicielles (cœurs de processeurs) et N^{PRR} unités d'exécution matérielles, qui sont les régions reconfigurables (PRRs). Le nombre total d'unités d'exécution N^{EU} est donc défini par N^{core} + N^{PRR} et la $j^{ième}$ unité d'exécution est dénommée EU_j où $j \epsilon [1, N^{EU}]$. Dans cette représentation formelle, une plateforme hétérogène est représentée par l'ensemble de toutes ses unités d'exécution { EU_j } et cette composition est supposée fixe.

La taille de l'unité d'exécution EU_j est caractérisée en termes de ressources logiques par les paramètres N_j^{cell}, N_j^{bram}, N_j^{dsp} pour une caractérisation réaliste et indépendante du vendeur. La terminologie *cell* désigne la principale ressource configurable de la logique programmable (e.g. Xilinx *Slices*, Altera *Logic Elements*). En ce qui concerne la caractérisation énergétique, une distinction est faite entre les différentes composantes de chaque EU_j. La puissance à vide P_j^{empty} ou $P_{j,k}^{empty}$ reflète la puissance consommée quand aucune tâche applicative ne s'exécute, respectivement pour les PRRs et les cœurs (à la fréquence $F_{j,k}$). Il est notable concernant les PRRs qu'une tâche configurée accumule deux contributions : $P_{i,k}^{idle}$ quand la tâche est au repos, et $P_{i,k}^{run}$ quand la tâche s'exécute, qui sont toutes deux dépendantes de l'implémentation et décrites dans la section « implémentations des tâches » du Tableau 7. La caractérisation P_j^{empty} est également très utile pour analyser les possibilités de *blanking* dans le déploiement des tâches. Pour les unités logicielles, $P_{i,k}^{run}$ est la puissance d'un cœur EU_j à la fréquence $F_{j,k}$ (à pleine charge). L'énergie d'une tâche logicielle peut ainsi être calculée facilement à partir de $P_{i,k}^{run}$ et du temps d'exécution correspondant.

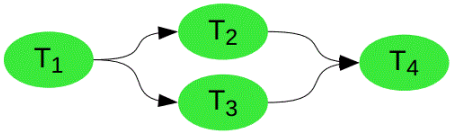
Variable	Range	Definition	
N^T	$\in \mathbb{N}^*$	Number of tasks of the application	Task graph
T_i	$\forall i = 1, ..., N^T$	The i^{th} task of the application	
$\mathcal{G}$	$= \{T_i\}$	Task-dependency graph	
$T^{eq}_{i1,i2}$	$\in \{0,1\}\ \forall i1, i2 = 1, ..., N^T$	Tasks equivalence matrix	
N_i^{imp}	$\in \mathbb{N}^*$	Number of implementations of T_i	Task implementation
$T_{i,k}$	$\forall k = 1, ..., N_i^{imp}$	The k^{th} implementation of T_i	
$I_{i,j,k}$	$\in \{0,1\}$	Defines if $T_{i,k}$ is instantiable on EU_j	Task execution
$C_{i,j,k}$	$\in \mathbb{R}^+$	Execution time of $T_{i,k}$	
$E_{i,j,k}$	$\in \mathbb{R}^+$	Energy consumption of $T_{i,k}$ on EU_j	
$P^{idle}_{i,k}$	$\in \mathbb{R}^+$	Idle power consumption of $T_{i,k}$	
$P^{run}_{i,k}$	$\in \mathbb{R}^+$	Running power consumption of $T_{i,k}$	

Tableau 7 – Paramètres du modèle d'application

Le coût de la reconfiguration dynamique partielle est modélisé en termes de latence et d'énergie. La latence de reconfiguration dépend essentiellement de la vitesse du contrôleur de reconfiguration et de la taille du bitstream de configuration. Elle peut être calculée efficacement à partir d'un paramètre T^{1cell} qui représente le temps nécessaire pour configurer une cellule logique et reflète les performances du contrôleur de reconfiguration. La puissance est caractérisée de façon similaire par un paramètre E^{1cell} qui est l'énergie nécessaire à la configuration d'une cellule logique. Comme les bitstreams de configuration dépendent principalement du nombre de cellules logiques d'une PRR en pratique, le coût en temps et en énergie de la reconfiguration d'une PRR peuvent être calculé rapidement.

Différentes caractéristiques des tâches applicatives doivent être connues pour l'exploration et l'estimation. Ces informations sont formalisées par un ensemble de paramètres (Tableau 7) qui peuvent être classés en trois catégories : le graphe de tâches, les implémentations de tâches et les caractéristiques d'exécution des tâches.

Un graphe de dépendances de tâches est utilisé pour permettre une exploitation efficace du parallélisme dans l'analyse du déploiement. Les dépendances entre tâches sont énumérées par une matrice d'adjacence pour pouvoir être traité facilement par un algorithme d'analyse. La matrice d'adjacence est une matrice N^T * N^T utilisée pour représenter les dépendances entre tâches.



	1	2	3	4
1	\	-1	-1	0
2	1	\	0	-1
3	1	0	\	-1
4	0	1	1	\

Figure 16 – Exemple de graphe de tâche et matrice d'adjacence.

Pour une ligne *i*, la valeur à la colonne *j* vaut 1 si T_i dépend de la tâche T_j, sinon elle vaut 0. La matrice est asymétrique et prend des valeurs -1 pour traduire les dépendances en sens inverse (Figure 16). Par ailleurs, la matrice d'adjacence est complétée par une autre information appelée matrice d'équivalence de tâches utilisée pour indiquer si plusieurs tâches sont identiques, et peuvent ainsi réutiliser la même implémentation (en particulier matérielle).

Pour analyser le déploiement à partir de la représentation précédente, il est aussi nécessaire de décrire les implémentations possibles pour chaque tâche. Plusieurs implémentations logicielles (cœur, fréquence) et matérielles (PRR, parallélisme) peuvent être décrites pour une même tâche pour traduire différents compromis performance / consommation dans l'analyse du déploiement. Le nombre total d'implémentations logicielles et matérielles pour la tâche T_i est N_i^{imp} et $T_{i,k}$ est utilisé pour représenter la $k^{ième}$ implémentation de la tâche T_i avec $k \epsilon [1, N_i^{imp}]$.

Concernant les caractéristiques d'exécution des tâches, une variable $I_{i,j,k}$ est utilisée pour indiquer si $T_{i,k}$ peut s'exécuter sur l'unité EU_j (0 faux, 1 vrai). Quand la tâche T_i s'exécute sur une unité logicielle (EU_j à la fréquence $F_{j,k}$), le temps d'exécution et l'énergie correspondants sont définis par $C_{i,j,k}$ et $E_{i,j,k}$. L'énergie peut être déduite de la caractérisation en puissance du cœur : $E_{i,j,k} = P_{j,k}^{run} * C_{i,j,k}$. Un modèle légèrement différent s'applique pour les tâches matérielles. Quand la $k^{ième}$ implémentation d'une tâche matérielle est déployée sur une PRR, elle ajoute une part de consommation idle. Cette contribution est référencée par $P_{i,k}^{idle}$ pour la $k^{ième}$ implémentation de la tâche T_i. Une autre contribution $P_{i,k}^{run}$ en puissance se rajoute pour la tâche matérielle en exécution. La puissance totale consommée par la PRR EU_j quand la tâche $T_{i,k}$ est configurée et en exécution est: $P_j^{empty} + P_{i,k}^{idle} + P_{i,k}^{run}$.

3.4.2 PRE-Explorer

Une première analyse du déploiement exploitant la formalisation précédente est décrite ici. Cette approche simple réalise une analyse exhaustive de toutes les solutions de déploiement possibles sur l'architecture hétérogène afin d'identifier les plus intéressantes. L'application considérée est le décodeur H.264/AVC décrit au chapitre 3.2. Le filtre de sortie anti-bloc (*DB_Filter*), le décodage entropique CAVLC (*Inv_CAVLC*) et la quantification / transformée inverse (*Inv_QTr*) contribuent ensemble à 76% du temps d'exécution global sur un processeur simple cœur. Ces blocs représentent les trois fonctionnalités du décodeur qui peuvent

être exécutées de façon logicielle ou matérielle dans notre étude de cas. En plus des possibilités d'accélération matérielle, on souhaite aussi pouvoir exploiter un parallélisme supporté par des architectures multicœur. On considère à cet effet une version multitâche du décodeur utilisant une décomposition par tranche (cf. 3.1.1) supporté dans le standard H.264/AVC. La version utilisée est une décomposition en deux tranches où deux flux correspondants chacun à une moitié de l'image sont traités en parallèle (Figure 17).

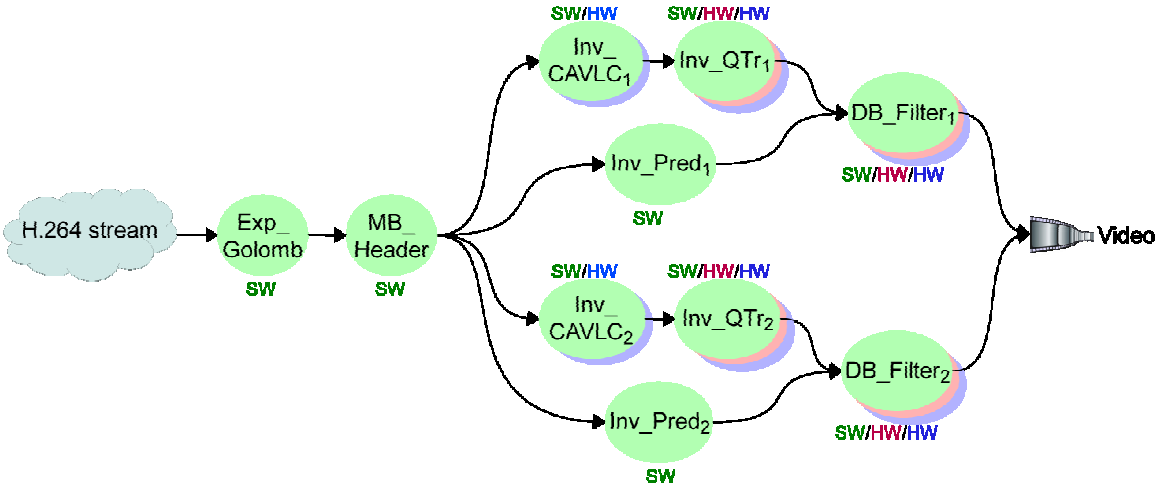


Figure 17 – Graphe de tâche du décodeur H.264 parallélisé en deux tranches.

Ce graphe peut être déployé en utilisant jusqu'à six accélérateurs et deux cœurs sur la plateforme cible. En l'absence de plateforme supportant toutes les caractéristiques nécessaires au début de ces travaux (multicœur, mesure de puissance, Reconfiguration dynamique, Linux, DVFS, etc.), cet exemple de caractérisation se base sur une plateforme fictive qui associerait des cœurs ARM CortexA8 et un FPGA Xilinx Virtex-6 LX240T. Les modèles de puissance sont définis par des mesures physiques sur une carte Mistral Texas Instruments TI OMAP 3530 EVM (65 nm) pour les cœurs logiciels et une carte Xilinx EK-V6-ML605-G (40 nm) pour l'accélération matérielle et la reconfiguration dynamique partielle. Le Tableau 8 montre les valeurs des paramètres du modèle d'architecture hétérogène exposé précédemment pour la plateforme considérée.

Platform	N^{core}	N^{PRR}	N^{EU}	Description
	2	3	5	CortexA8/V6LX240T
Cores	EU_j	$F_{j,k}$ (MHz)	$P^{empty}_{j,k}/P^{run}_{j,k}$ (mW)	Description
	EU_1	600	24/445	core #1
	EU_2	600	24/445	core #2
PRRs	EU_j	$N^{cell}_j; N^{bram}_j; N^{dsp}_j$	P^{empty}_j (mW)	Description
	EU_3	1200; 8; 0	50	PRR #1
	EU_4	3280; 8; 0	137	PRR #2
	EU_5	2000; 8; 0	83	PRR #3

Tableau 8 – Paramètres du modèle de la plateforme cible (CortexA8/Virtex-6 LX240T).

La méthode décrite dans [9] est utilisée pour déterminer un ensemble optimal de PRRs qui permette de respecter les contraintes de performances et le plan de masse du FPGA, en tenant compte des implémentations matérielles des tâches. Trois PRRs de 1200, 3280 et 2000 *slices* sont ainsi définies et

permettent de réduire le nombre de *slices* et de BRAMs de 49% et 33% respectivement par rapport à une implémentation statique de tous les accélérateurs. Cette configuration sera considérée comme le partitionnement par défaut du FPGA pour l'analyse du déploiement qui suit. A partir de la valeur de la puissance à vide de 47µW/slice mesurée sur le FPGA, on déduit la consommation à vide de chaque PRR: P_3^{empty} = 50 mW, P_4^{empty} = 137 mW, P_5^{empty} = 83 mW. La puissance à vide des cœurs est issue de mesures similaires sur l'OMAP 3530: $P_{j,k}^{empty}$ = 24 mW et $P_{j,k}^{run}$ = 445 mW pour un cœur ARM CortexA8 à $F_{j,k}$ = 600 MHz. Par ailleurs, le contrôleur de reconfiguration utilisé est une version optimisée dénommée UPaRC qui supporte une vitesse de reconfiguration de 400 MB/s pour une puissance de $P_{controller}$ = 150 mW [8]. Ces valeurs correspondent à T^{1cell} = 0,41 µs et E^{1cell} = 61,5 mJ à partir desquelles il est facile de calculer le temps et l'énergie de reconfiguration d'une PRR.

T_i	$T_{i,k}$	$I_{i,j,k} = 1$	$C_{i,j,k}$(ms)	$E_{i,j,k}$(mJ)	$P_{i,k}^{idle}/P_{i,k}^{run}$(mW)	$N_{i,k}^{cell}$; $N_{i,k}^{bram}$; $N_{i,k}^{dsp}$	Description
T_1	$T_{1,1}$	$I_{1,1,1}$ $I_{1,2,1}$	5.00	2.23	24/445	–	Exp_Golomb
T_2	$T_{2,1}$	$I_{2,1,1}$ $I_{2,2,1}$	4.92	2.19	24/445	–	MB_Header
T_3	$T_{3,1}$	$I_{3,1,1}$ $I_{3,2,1}$	11.03	4.91	24/445	–	Inv_CAVLC_1
	$T_{3,2}$	$I_{3,4,2}$	7.45	1.46	55.1/4.4	3118; 6; 0	
T_4	$T_{4,1}$	$I_{4,1,1}$ $I_{4,2,1}$	11.03	4.91	24/445	–	Inv_CAVLC_2
	$T_{4,2}$	$I_{4,2,4}$	7.45	1.46	55.1/4.4	3118; 6; 0	
T_5	$T_{5,1}$	$I_{5,1,1}$ $I_{5,2,1}$	5.10	2.27	24/445	–	Inv_QTr_1
	$T_{5,2}$	$I_{5,3,2}$	2.46	0.24	34.2/11.47	1056; 7; 0	
	$T_{5,2}$	$I_{5,4,2}$	2.46	0.45	34.2/11.47	1056; 7; 0	
	$T_{5,2}$	$I_{5,5,2}$	2.46	0.32	34.2/11.47	1056; 7; 0	
	$T_{5,3}$	$I_{5,3,3}$	1.97	0.21	42.2/12.87	1385; 7; 0	
	$T_{5,3}$	$I_{5,4,3}$	1.97	0.38	42.2/12.87	1385; 7; 0	
	$T_{5,3}$	$I_{5,5,3}$	1.97	0.27	42.2/12.87	1385; 7; 0	
T_6	$T_{6,1}$	$I_{6,1,1}$ $I_{6,2,1}$	5.10	2.27	24/445	–	Inv_QTr_2
	$T_{6,2}$	$I_{6,3,2}$	2.46	0.24	34.2/11.47	1056; 7; 0	
	$T_{6,2}$	$I_{6,4,2}$	2.46	0.45	34.2/11.47	1056; 7; 0	
	$T_{6,2}$	$I_{6,5,2}$	2.46	0.32	34.2/11.47	1056; 7; 0	
	$T_{6,3}$	$I_{6,3,3}$	1.97	0.21	42.2/12.87	1385; 7; 0	
	$T_{6,3}$	$I_{6,4,3}$	1.97	0.38	42.2/12.87	1385; 7; 0	
	$T_{6,3}$	$I_{6,5,3}$	1.97	0.27	42.2/12.87	1385; 7; 0	
T_7	$T_{7,1}$	$I_{7,1,1}$ $I_{7,2,1}$	5.39	2.40	24/445	–	Inv_Pred_1
T_8	$T_{8,1}$	$I_{8,1,1}$ $I_{8,2,1}$	5.39	2.40	24/445	–	Inv_Pred_2
T_9	$T_{9,1}$	$I_{9,1,1}$ $I_{9,2,1}$	17.49	7.78	24/445	–	DB_Filter_1
	$T_{9,2}$	$I_{9,3,2}$	1.57	0.14	33.4/6	686; 5; 0	
	$T_{9,2}$	$I_{9,4,2}$	1.57	0.28	33.4/6	686; 5; 0	
	$T_{9,2}$	$I_{9,5,2}$	1.57	0.19	33.4/6	686; 5; 0	
	$T_{9,3}$	$I_{9,4,3}$	1.55	0.29	40.3/7.4	1869; 5; 0	
	$T_{9,3}$	$I_{9,5,3}$	1.55	0.20	40.3/7.4	1869; 5; 0	
T_{10}	$T_{10,1}$	$I_{10,1,1}$ $I_{10,2,1}$	17.49	7.78	24/445	–	DB_Filter_2
	$T_{10,2}$	$I_{10,3,2}$	1.57	0.14	33.4/6	686; 5; 0	
	$T_{10,2}$	$I_{10,4,2}$	1.57	0.28	33.4/6	686; 5; 0	
	$T_{10,2}$	$I_{10,5,2}$	1.57	0.19	33.4/6	686; 5; 0	
	$T_{10,3}$	$I_{10,4,3}$	1.55	0.29	40.3/7.4	1869; 5; 0	
	$T_{10,3}$	$I_{10,5,3}$	1.55	0.20	40.3/7.4	1869; 5; 0	

Tableau 9 – Paramètres du modèle de tâches du décodeur H.264.

Le Tableau 9 expose le détail des paramètres du modèle de tâches de l'exemple du décodeur H.264. Ce décodeur est composé de dix tâches parmi lesquelles six peuvent être exécutées en matériel. Pour chaque

tâche matérielle possible (T_3, T_4, T_5, T_6, T_9, T_{10}), deux versions avec des compromis coût / performance différents sont produits en exploitant le déroulage de boucle par l'outil de Synthèse de Haut Niveau. Si on considère l'exemple de la fonction *Inv_QTr1* (T_5), trois exécutions ($T_{5,1}$, $T_{5,2}$, $T_{5,3}$) sont possibles en respectivement 5,10ms (CortexA8 600MHz), 2,46 ms (solution matérielle #1) et 1,97 ms (solution matérielle #2 avec déroulage de boucle). Pour chacune des deux implémentations matérielles, trois allocations de PRR possibles sont décrites avec leur coût énergétique associé. Cette caractérisation réaliste de différentes implémentations de tâches basée sur des données issues de mesures améliore la qualité des estimations et des résultats d'exploration, qui sont discutés par la suite.

Sous les conditions précédentes d'application, de plateforme et de reconfiguration dynamique, le résultat global de l'exploration est tracé à la Figure 18.

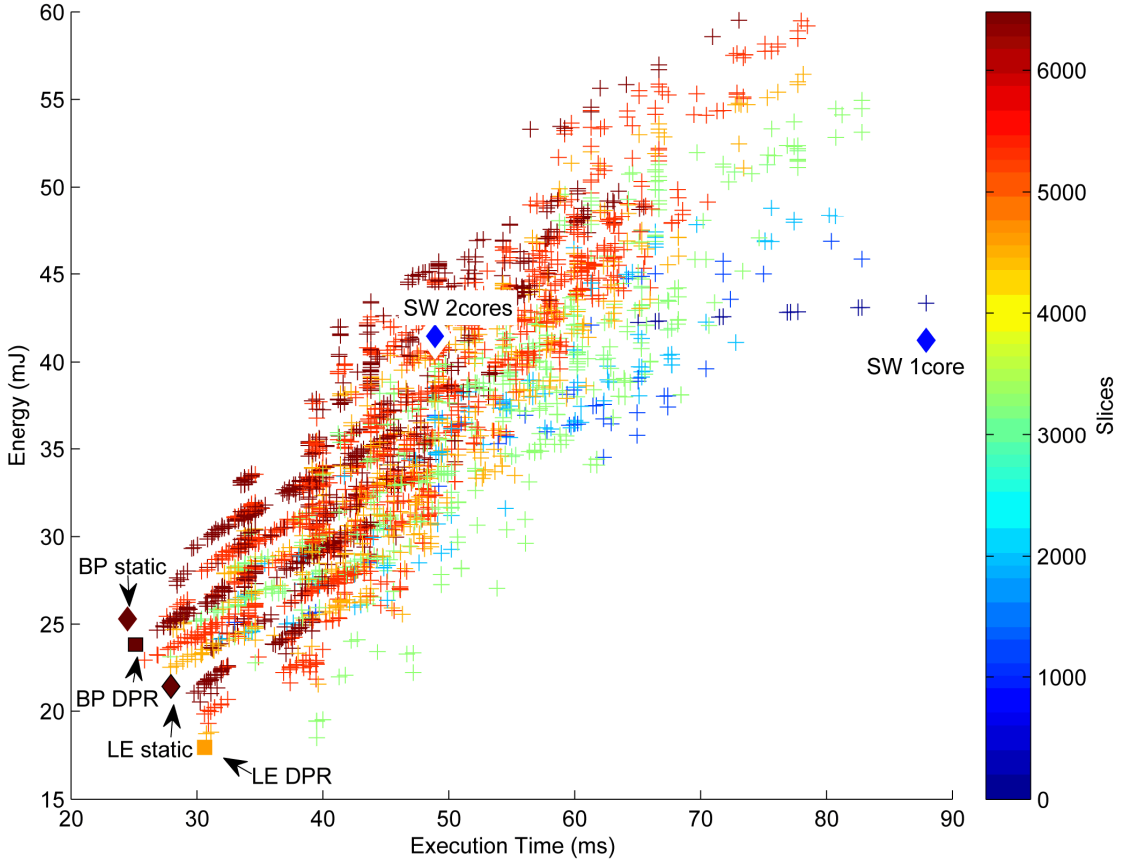


Figure 18 – Résultats d'exploration du décodeur H.264.

On peut tout d'abord noter le très grand nombre de solutions analysées, avec plus d'un million de déploiements possibles évalués. Cet exemple est traité en quelques secondes avec une station de travail équipée d'un processeur Intel Core i5. Six solutions sont mises en avant: (i) une solution logicielle à base d'un cœur (*SW_1core*), (ii) une solution logicielle utilisant deux cœurs (*SW_2cores*), (iii) la solution la moins consommatrice avec des accélérateurs *statiques* (*LE_Static*), (iv) la solution la plus performante avec des accélérateurs *statiques* (*BP_Static*), (v) la solution la moins consommatrice utilisant la reconfiguration dynamique (*LE_DPR*) et (vi) la solution la plus performante utilisant la reconfiguration dynamique (*BP_DPR*). Les détails de ces solutions caractéristiques sont résumés dans le Tableau 10. Etant donné que la solution *SW_1core* est presque deux fois plus lente que *SW_2cores* pour le même coût énergétique, cette dernière sera considérée comme point de comparaison pour l'évaluation des améliorations relatives des solutions

matérielles. Pour permettre une analyse plus poussée, l'exploration permet également de tracer les profils d'ordonnancement, d'allocation et de puissance qui sont illustrés Figure 19 et Figure 20 pour *BP_Static*, *LE_Static*, *BP_DPR* et *LE_DPR.*

Implementation results	$Energy(mJ)$	$T_{EX}(ms)$	$CPUs$;	$Slices$;	$DSPs$;	$BRAMs$
SoftWare 1core	41.23	87.92	1;	0;	0;	0
SW 2cores(reference)	41.47	48.92	2;	0;	0;	0
Lowest Energy - static	21.40 (-48%)	27.93 (-43%)	1;	6480;	0;	18
Best Performance - static	25.30 (-39%)	**24.49 (-50%)**	2;	6480;	0;	18
Lowest Energy - DPR	**17.94 (-57%)**	30.62 (-37%)	1;	4480;	0;	16
Best Performance - DPR	23.84 (-43%)	25.13 (-49%)	2;	6480;	0;	24

Tableau 10 – Détails des résultats d'exploration du décodeur H.264.

On peut noter que les quatre solutions accélérées sont toutes plus intéressantes que les solutions logicielles, à la fois en termes de performance et d'énergie. L'exécution matérielle améliore significativement l'efficacité du traitement en déchargeant les cœurs logiciels, en réduisant 50% du temps d'exécution et 39% de l'énergie au niveau du décodeur global. La reconfiguration dynamique introduit un léger ralentissement à cause des temps de reconfiguration, néanmoins la solution RDP la plus performante n'est plus lente que de 2,6% par rapport à une implémentation statique. En terme d'énergie, la solution RDP la moins consommatrice réduit le coût énergétique de 57% par rapport à la références *SW_2cores*, et de 16% par rapport à la solution *LE_Static*.

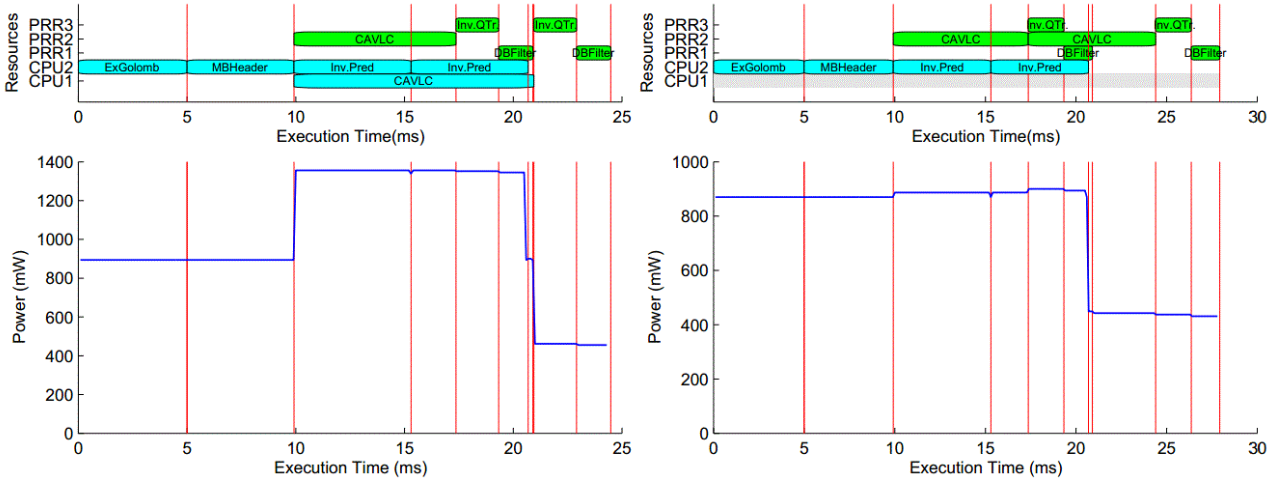


Figure 19 – Ordonnancement et profil de puissance des solutions *BP_Static* (gauche) et *LE_Static* (droite)

Si on analyse le détail de l'ordonnancement et de l'utilisation des ressources à la Figure 19 et à la Figure 20, on peut noter que ce sont les solutions les plus performantes qui utilisent le plus de ressources tandis que les solutions énergétiques utilisent moins d'unités d'exécution en améliorant leur taux d'utilisation. Par exemple, la consommation de la solution *BP_Static* est réduite en déchargeant l'exécution de la fonction *CAVLC* du premier cœur sur *PRR2*, ce qui conduit à une réduction de 3,9 mJ (-15%) pour une perte en performance de 3,4 ms (+14%). Il en est de même pour l'implémentation BP_DPR dans laquelle les exécutions sur le premier cœur et *PRR3* sont évitées pour économiser 4,9 mJ (-25%) en contrepartie de 5,5 ms (+22%). La minimisation du nombre de reconfigurations (représentées en rouge dans les profils d'ordonnancement) est aussi un facteur important qui impacte l'énergie et le temps d'exécution. Dans les deux implémentations du décodeur qui utilisent la reconfiguration dynamique, la configuration de *PRR2* est

préservée pour l'exécution consécutive de deux instances de *Inv_CAVLC*, et une démarche identique s'applique pour l'exécution de *DB_Filter* sur *PRR1*. En outre, *Inv_QTr* est déployée sur *PRR1* à sa première exécution et sur *PRR2* à sa deuxième. Les dépendances d'exécution entre *DB_Filter* et *Inv_QTr* ne permettent pas l'économie d'une reconfiguration de *Inv_QTr* sans ralentir le temps d'exécution global, ce qui impacte aussi l'énergie. La seconde instance de *Inv_QTr* est donc exécutée sur *PRR2*.

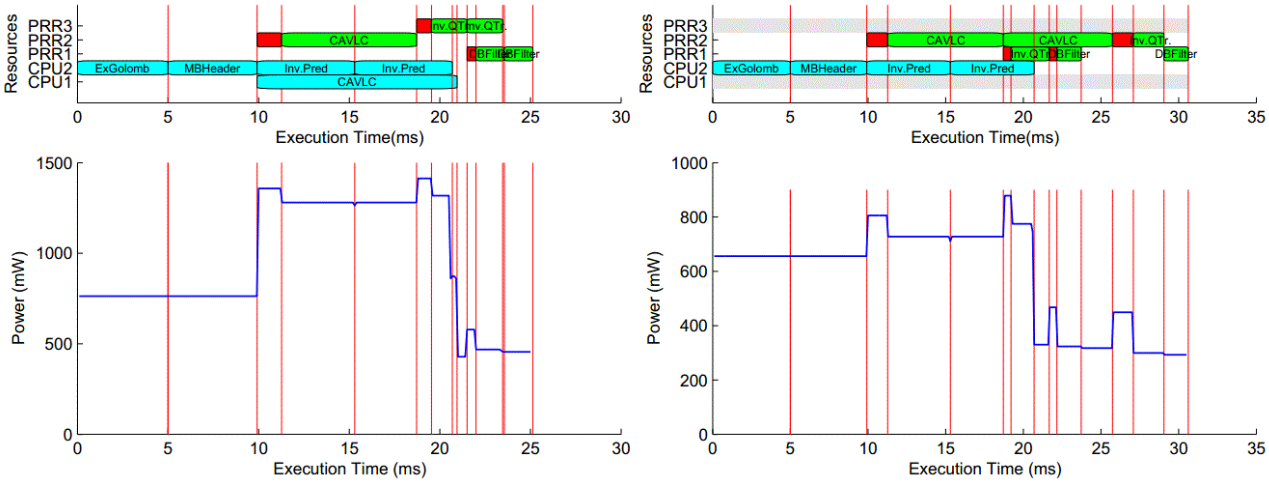


Figure 20 - Ordonnancement et profil de puissance des solutions *BP_DPR* (gauche) et *LE_DPR* (droite)

En termes de bénéfices de la reconfiguration dynamique par rapport à la reconfiguration statique, l'exemple du décodeur H.264 montre que la reconfiguration dynamique réduit le coût énergétique de 16% et les ressources FPGA (*slices*) de 31%, pour une perte de 10% en performance. Les gains en énergie proviennent de la réduction de surface statique active occupée par la logique programmable et la puissance idle associée, qui décroit de 318 mW à 210 mW (34%). Ces résultats aident aussi à évaluer les gains possibles par la reconfiguration dynamique en pratique, et doivent tenir compte du fait qu'il est possible d'améliorer ces résultats en utilisant plus de fonctions matérielles. Enfin, il est aussi intéressant de noter que dans cet exemple, aucune des quatre solutions mise en valeur n'exploite la possibilité de *blanking* (effacement) d'une *PRR*. Dans la solution *LE_DPR*, les unités d'exécution matérielles ne sont pas libres assez longtemps pour compenser le surcoût d'une reconfiguration. Ces estimations permettent donc aussi d'évaluer l'intérêt d'utiliser ou non les techniques de blanking dans un projet.

Le meilleur résultat pour l'architecture cible considérée est donc un déploiement utilisant un seul cœur avec une reconfiguration dynamique de six accélérateurs sur deux *PRRs*. L'implémentation correspondante permet 57% et 37% d'amélioration en performance et énergie par rapport à une exécution logicielle à base de deux cœurs, et une réduction de 16% en énergie pour une perte en performance de 10% par rapport à une exécution matérielle statique.

L'approche décrite ici montre l'utilité en pratique des modèles et de la formalisation pour faciliter l'exploration de l'espace de conception et de déploiement. L'analyse exhaustive, utilisée ici en premier lieu pour établir une preuve de concept sur un exemple relativement limité en nombre de tâches et de ressources, doit ainsi être amélioré pour pourvoir être appliqué à des plateformes et applications plus complexes. C'est l'objectif de l'intégration de cette modélisation dans la méthodologie FoRTReSS qui décrite ci-après.

3.4.3 FoRTReSS

La deuxième approche décrite ici aborde la principale faiblesse du flot précédent (complexité d'une recherche exhaustive) en effectuant une analyse du déploiement plus méthodique. Elle se base sur la méthodologie FoRTReSS développée au LEAT par l'équipe de Fabrice Muller et à laquelle les travaux précédents de modélisation énergétique de la RDP s'associent logiquement dans le projet BENEFIC.

FoRTReSS (*Flow for Reconfigurable archiTectures in Real-time SystemS*) est initialement un environnement de conception qui permet à l'utilisateur d'explorer le déploiement d'applications sur des systèmes multiprocesseur reconfigurables dynamiquement. A partir d'une application ou d'un ensemble d'applications, il recherche un partitionnement FPGA (i.e. définit les régions reconfigurables) et un ensemble de cœurs logiciels par une simulation *time-acurate* du déploiement des tâches applicatives matérielles et logicielles qui se base sur des stratégies d'ordonnancement réelles. Chaque tâche peut disposer de plusieurs implémentations possibles, chacune correspondant à un compromis ressource performance différent. Ces implémentations peuvent être matérielles (exécutées sur une région reconfigurable d'un FPGA) ou logicielles (exécutées sur un cœur de processeur). Une implémentation matérielle est définie par les ressources et les performances d'une tâche qui peuvent être déterminées par génération des accélérateurs (en utilisant la méthodologie décrite au chapitre 3.2). Il peut y avoir plusieurs implémentations matérielles d'une même tâche pour refléter des niveaux de parallélisme différents. Une implémentation logicielle est principalement caractérisée par son temps d'exécution sur un cœur, éventuellement aux différentes fréquences supportées par le processeur.

Task	SW_{ex}	HW_{ex}			
	WCET(ms)	WCET(ms)	SLICE	DSP	BRAM
Exp_Golomb	1.96	n/a	n/a	n/a	n/a
MB_Header	1.96	n/a	n/a	n/a	n/a
Inv_CAVLC	20.56	5.05	3383	0	6
Inv_QTr	30.35	15.48	1202	3	7
Inv_Pred	8.81	n/a	n/a	n/a	n/a
DB_Filter	23.50	6.50	701	0	5

Tableau 11 – Paramètres des tâches H.264 (1 tranche) sur Zynq-7000 EPP.

Task	SW_{ex}	HW_{ex}			
	WCET(ms)	WCET(ms)	SLICE	DSP	BRAM
Exp_Golomb	1.96	n/a	n/a	n/a	n/a
MB_Header	1.96	n/a	n/a	n/a	n/a
Inv_CAVLC	10.28	2.53	3383	0	6
Inv_QTr	15.18	7.74	1202	3	7
Inv_Pred	4.41	n/a	n/a	n/a	n/a
DB_Filter	11.75	3.25	701	0	5

Tableau 12 – Paramètres des tâches H.264 (2 tranches) sur Zynq-7000 EPP.

L'architecture cible est donnée par une description précise des ressources FPGA et un ensemble de cœurs logiciels. A partir de la description des tâches applicatives et de l'architecture cible, le simulateur *SystemC/TLM* intégré (RecoSim) est utilisé pour rechercher les meilleures régions reconfigurables capables

d'héberger rigoureusement les tâches sous des contraintes d'exécution en temps-réel. En utilisant des caractéristiques temporelles supplémentaires (performance, *deadline*, période, etc.), le simulateur réalise plusieurs déploiements (et ordonnancements associés) de l'application complète sur différentes combinaisons de régions reconfigurables et de cœurs. Une description plus détaillée de la méthodologie est fournie par exemple dans [9]. Par la suite, nous détaillerons l'analyse du déploiement sur deux études de cas pour comprendre les extensions des méthodologies *PRE-Explorer* et *FoRTReSS* qui ont été développées. La première application étudiée est le décodeur H.264/AVC parallélisé par tranches décrit au chapitre 3.4.2. La seconde application est un encodeur H.265/HEVC issu du projet open-source *x265* [11].

Le décodeur H.264 considéré est celui de la Figure 17, parallélisé par tranches (1 ou 2 tranches). Les différentes fonctions logicielles et matérielles sont ici caractérisées sur une plateforme Xilinx Zynq-7000 EPP (Artix-7, dual MPCore CortexA9). Les paramètres des tâches logicielles et matérielles correspondantes sont reportés au Tableau 11 et au Tableau 12. Ces paramètres servent de point d'entrée à la méthodologie FoRTReSS. Les caractéristiques des tâches matérielles (colonne HW_{ex}) sont issues de la génération complète des accélérateurs sur la plateforme Zynq et celles des tâches logicielles (colonne SW_{ex}) résultent de l'exécution sur un cœur CortexA9 à 667MHz. L'objectif est de déterminer s'il est possible de décoder un flux vidéo H.264 en temps-réel, c'est-à-dire en 30 images par seconde. Les résultats produits par FoRTReSS sont résumés au Tableau 13.

Implementation	Framerate (fps)	Resource type	Static area	Number of RRs	PR area (columns)	PR area	Improvement (%) Raw	Total
Full SW 1 slice 1 core	11.4	Slice BRAM DSP	n/a	n/a	n/a	n/a	n/a	n/a
Full SW 2 slices 2 cores	21.7	Slice BRAM DSP	n/a	n/a	n/a	n/a	n/a	n/a
HW/SW 1 slice 1 core	23.3	Slice BRAM DSP	5551 18 3	1	72 1 1	3600 10 20	35.15 44.4 -566.7	29.40 44.4 -566.7
HW/SW 2 slices 1 core	28.2	Slice BRAM DSP	11102 36 6	4	140 4 3	7000 40 60	36.95 -11.11 -900	25.45 -11.11 -900
HW/SW 2 slices 2 cores	34.1	Slice BRAM DSP	11102 36 6	4	140 4 3	7000 40 60	36.95 -11.11 -900	25.45 -11.11 -900
HW/SW 2 slices 2 cores	30	Slice BRAM DSP	11102 36 6	2	88 2 1	4400 20 20	60.37 44.44 -233.3	54.62 44.44 -233.3

Tableau 13 – Performances et utilisation des ressources reconfigurables pour le décodeur H.264.

Les résultats en surface (i.e. l'occupation des ressources reconfigurables) concernent à la fois les solutions reconfigurables statiques et dynamiques pour mieux juger des améliorations relatives de la RDP. L'amélioration brute (colonne *raw improvement*) est calculée en comparant les ressources d'une solution statique (où trois accélérateurs matériels sont implémentés sans RDP) par rapport à celles occupées par l'ensemble des régions reconfigurables. Une amélioration globale (colonne *total improvement*) est également définie en tenant compte des ressources reconfigurables utilisées par le contrôleur de

reconfiguration (*Fast Reconfiguration Manager*, FaRM [7]). La seconde colonne *framerate* indique que la première solution qui respecte la contrainte des 30 images par seconde est une implémentation logicielle / matérielle d'une version parallélisée en deux tranches impliquant deux cœurs (34,1 img/sec). L'amélioration de l'utilisation des ressources est de 25,45%, -11,11% et -900% pour les *slices*, BRAM et blocs DSP respectivement. Cela signifie que la RDP a permis de diminuer le nombre de *slices* nécessaires mais les besoins en BRAM et blocs DSP ont augmenté. Néanmoins ces résultats peuvent être améliorés: la vitesse de décodage dépasse sensiblement les 30 img/sec et cela peut être exploité pour assouplir la contrainte d'exécution globale (deadline) à 33,3 ms (30 img/sec). Procéder de la sorte permet une amélioration de 54% en nombre de *slices* (contre 25,45% précédemment) et de 44% en BRAM (dernière ligne du Tableau 13).

Implementation	Framerate	Core (%)		Reconfigurable Region (%)					
	(fps)	Core1	Core2	RR1	RR2	RR3	RR4	RR5	RR6
HW/SW (2 slices/cores)	34.1	48.42	43.37	26.39	11.08	26.39	11.08	n/a	n/a
HW/SW (2 slices/cores)	30	88.09	73.40	n/a	n/a	n/a	n/a	9.74	30.77

Tableau 14 – Taux d'utilisation des ressources logicielles / matérielles pour le décodeur H.264.

Pour permettre de mieux comprendre ce résultat, le Tableau 14 montre les taux d'occupation des cœurs et des régions reconfigurables des deux dernières solutions du Tableau 13. Les différences proviennent de la vitesse de décodage et des allocations possibles de la fonction *Inv_CAVLC* (cœurs ou région reconfigurables). La réduction de performance laisse à l'ordonnanceur assez de marge pour utiliser une implémentation plus lente (logicielle) de la tâche à la place d'une exécution matérielle. La solution à 30 img / sec montre de meilleurs taux d'utilisation avec 73,4% et 88,09% contre 43,37% et 48.42% respectivement pour deux cœurs.

A travers les différentes solutions du Tableau 13, on note une utilisation excessive de blocs DSP. Elle s'explique par le faible besoin en blocs DSP de l'application (seulement trois) et par la granularité de la reconfiguration. Chaque région reconfigurable doit couvrir un domaine d'horloge complet ce qui contraint les ressources d'une colonne. Chaque colonne est constituée de 20 blocs *DSP48* qui est la limite maximum pour un composant Zynq-7000. Il y a donc inévitablement un surcoût important (multiple de 20 en fonction des régions reconfigurables définies) en regard du petit nombre de blocs DSP des fonctions matérielles du décodeur (au total 3). Ces résultats peuvent être améliorés en forçant l'utilisation de cellules logiques (*slices* dans le cas de Xilinx) à la place de blocs DSP lors de la synthèse des accélérateurs.

Dans une deuxième étude de cas, nous nous intéressons à l'aptitude de cette approche à faciliter l'analyse du déploiement, très tôt à partir de spécifications préliminaires d'une application. Nous avons considéré pour cela un encodeur H.265/HEVC distribué récemment par le projet open source *x265* dont le code est disponible depuis juillet 2014 [11]. La Figure 21 montre le graphe de tâche issu du code C++ de référence. Idéalement, les caractéristiques des fonctions matérielles indispensables à l'exploration (au nombre de trois et identifiées par SW/HW dans la Figure 21) résultent de la synthèse Haut Niveau (HLS), dans le cas où le code C/C++ de référence est exploitable. Cela n'est généralement pas le cas pour la plupart des applications, y compris *x265*, car seul un sous ensemble de C/C++ peut être supporté en pratique par les outils de HLS. Le code source *x265* fait un usage généralisé de techniques d'allocation dynamique de mémoire et d'indirections arbitraires (pointeurs qui ne sont pas des tableaux statiques) qui nécessiteraient un effort très important pour les faire totalement disparaître (quasiment une ré-écriture complète du

code). Néanmoins plusieurs travaux se sont déjà penchés sur l'implémentation matérielle des fonctionnalités critiques de H.265/HEVC depuis la finalisation de l'annexe technique début 2013.

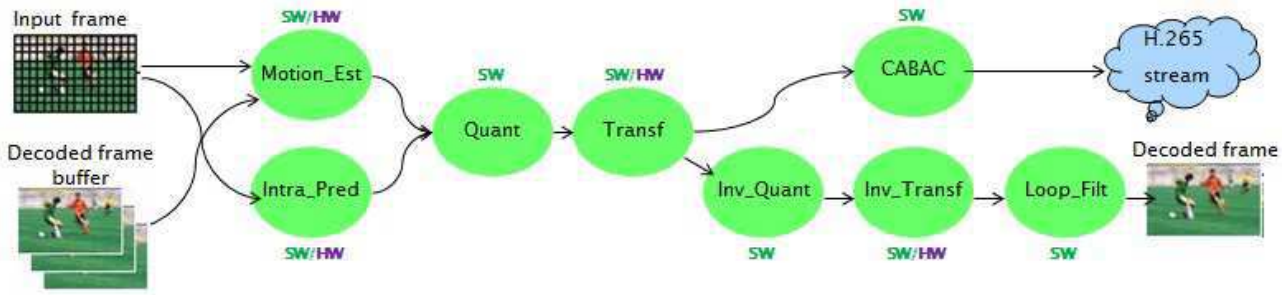


Figure 21 – Graphe de tâche de l'encodeur H.265/HEVC.

Le Tableau 15 montre les caractéristiques des trois fonctions matérielles sur des composants de la famille Virtex (Xilinx) décrites dans la littérature et qui peuvent être avantageusement utilisées pour établir une analyse préliminaire du potentiel de reconfiguration dynamique partielle sur cette application. Le bloc d'estimation de mouvement [12] constitue la partie la plus coûteuse du processus d'encodage, puis viennent un bloc matériel de prédiction intra haute performance [13] et un accélérateur dédié qui supporte les transformées et transformée inverse 2D [14].

Task	SW_{ex} WCET(ms)	HW_{ex} WCET(ms)	SLICE	DSP	BRAM
Motion_Est	2418	9.61	14067	0	297
Intra_Pred	126	1.43	472	0	4
Quant	37	n/a	n/a	n/a	n/a
Tranf	21	0.28	3595	0	0
CABAC	139	n/a	n/a	n/a	n/a
Inv_Quant	5	n/a	n/a	n/a	n/a
Inv_Transf	5	0.28	3595	0	0
Loop_Filt	23	n/a	n/a	n/a	n/a

Tableau 15 - Paramètres des tâches x265 (1 tranche) sur Zynq-7000 EPP.

Task	SW_{ex} WCET(ms)	HW_{ex} WCET(ms)	SLICE	DSP	BRAM
Motion_Est	1209	4.81	14067	0	297
Intra_Pred	63	0.72	472	0	4
Quant	19	n/a	n/a	n/a	n/a
Tranf	11	0.14	3595	0	0
CABAC	70	n/a	n/a	n/a	n/a
Inv_Quant	3	n/a	n/a	n/a	n/a
Inv_Transf	3	0.14	3595	0	0
Loop_Filt	12	n/a	n/a	n/a	n/a

Tableau 16 - Paramètres des tâches x265 (2 tranches) sur Zynq-7000 EPP.

Toutes les implémentations et caractéristiques associées concernent des composants de la famille Virtex et ont été extrapolées pour se conformer à la même résolution vidéo et une fréquence de fonctionnement de

tous les accélérateurs à 100MHz. Nous considérons la même plateforme Zynq-7000 EPP que précédemment. Comme pour le standard H.264, H.265 permet une décomposition par tranches pour permettre un encodage parallélisé sur plusieurs cœurs. Les paramètres de tâches correspondant à une décomposition en deux tranches sont donnés au Tableau 16.

Comme pour l'étude du décodeur H.264, on analyse différentes configurations en termes de parallélisation (1 ou 2 tranches) et de nombre d'unités d'exécution (nombre de cœurs et de régions reconfigurables). Les résultats correspondants sont présentés au Tableau 17. L'amélioration en performance est sensible pour les trois solutions logicielles / matérielles définies (jusqu'à 8,88 img / sec en utilisant deux cœurs et trois régions reconfigurables) par rapport aux deux solutions purement logicielles (respectivement 0,36 et 0,7 img / sec pour 1 tranche / 1 cœur et 2 tranches / 2 cœurs). Ce cas d'étude est très affecté par la présence du très important bloc de calcul que représente l'estimation de mouvement, qui occupe 65% du nombre de *slice* total occupé par les accélérateurs (14067 *slices*). Il existe peu de marge pour partager une même région capable d'héberger toutes les tâches, ainsi qu'un important surcoût temporel (temps de reconfiguration de 26,8 ms). Dans ces conditions, la meilleure solution est basée sur l'utilisation de trois régions, une pour chaque type de fonction (les fonctions *Transf* et *Inv_Transf* partagent le même accélérateur). Il n'y a pas d'amélioration possible par la reconfiguration dynamique dans cet exemple. Il existe même une légère augmentation du nombre de ressources logiques qui provient d'un surdimensionnement inévitable des régions pour pouvoir héberger les tâches.

Implementation	Framerate (fps)	Resource type	Static area	Number of RRs	PR area (columns)	PR area	Improvement (%) Raw	Total
Full SW 1 slice 1 core	0.36	Slice BRAM DSP	n/a	n/a	n/a	n/a	n/a	n/a
Full SW 2 slices 2 cores	0.7	Slice BRAM DSP	n/a	n/a	n/a	n/a	n/a	n/a
HW/SW 1 slice 1 core	4.58	Slice BRAM DSP	22673 301 0	3	552 32 0	27600 320 0	-21.7 -6.31 0	-26 -6.31 0
HW/SW 1 slice 2 cores	5.33	Slice BRAM DSP	22673 301 0	3	552 32 0	27600 320 0	-21.7 -6.31 0	-26 -6.31 0
HW/SW 2 slices 2 cores	8.82	Slice BRAM DSP	22673 301 0	3	552 32 0	27600 320 0	-21.7 -6.31 0	-26 -6.31 0

Tableau 17- Performances et utilisation des ressources reconfigurables pour l'encodeur x265.

Néanmoins, cette étude de cas montre l'intérêt de la méthodologie pour évaluer très tôt la possibilité d'utiliser la reconfiguration dynamique dans une application. De plus en approfondissant les résultats obtenus, il est possible d'analyser les causes et d'émettre des possibilités d'amélioration. Par exemple, le détail de l'ordonnancement de la solution à 8,82 img / sec (Figure 22) fait clairement apparaître que la fonction CABAC devient le problème majeur après accélération par les trois fonctions matérielles précédentes.

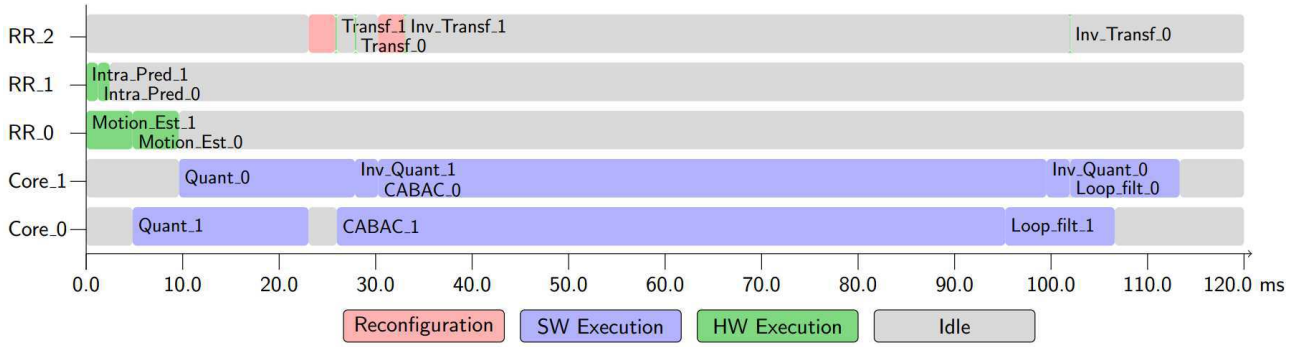


Figure 22 – Ordonnancement de la solution x265 la plus performante.

Une implémentation matérielle de cette fonction, ainsi que du filtrage anti-blocs (dans une moindre mesure), pourraient peut être permettre d'obtenir un traitement en temps réel (30 img / sec) et en fonction de leur taille, remettre en question le problème de temps de reconfiguration précédent. Cette possibilité n'a pas été approfondie en l'absence de données décrivant l'implémentation matérielle de ces fonctions dans la littérature au moment de ce travail.

3.5 Conclusion

Les différentes études et projets précédents ont successivement contribué à établir les bases d'une méthodologie d'analyse en efficacité énergétique de plateformes multiprocesseur hétérogènes. L'approche résolument applicative et la diversité de plateformes utilisées est riche d'enseignements utiles sur la modélisation système et les conditions d'une meilleure efficacité énergétique dans la pratique. Les deux dernières méthodologies en particulier fournissent une contributions sur l'accélération matérielle reconfigurable dont il ressort deux points : le potentiel de la reconfiguration dynamique en terme d'efficacité énergétique et la nécessité de méthodologies spécifiques pour en exploiter le potentiel. L'approche PRE-Explorer montre sur le cas concret du décodeur H.264 une amélioration par la RDP de 57% et 37% en performance et énergie par rapport à une exécution purement logicielle, et une réduction de 16% en énergie pour une perte en performance de 10% par rapport à une solution accélérée mais sans la RDP. La méthodologie *FoRTReSS* montre sur le même exemple une amélioration des performances pour une réduction sensible des ressources reconfigurables (54% en nombre de *slices* et 44% en BRAM). Dans les deux cas, les gains peuvent être encore améliorés en augmentant le nombre de fonctions matérielles (qui renforce aussi l'intérêt de la RDP pour limiter la taille de la zone reconfigurable).

Les extensions intégrées à *FoRTReSS* qui ont permis cette étude (support multi- cœur et pluri-applications) le rendent ainsi très adapté à l'analyse du déploiement d'applications complexes sur des plateformes multiprocesseur reconfigurables. Une perspective intéressante est donc naturellement d'intégrer la modélisation énergétique des travaux précédents pour permettre une analyse avancée de la consommation. Cette extension a été développée dans le cadre du projet européen CATRENE BENEFIC (*Best ENergy EFficiency solutions for heterogeneous multI-core Communicating systems*) et sera abordée au chapitre 5.3.

4 Gestion des ressources et de l'énergie

4.1 Stratégies multiprocesseur faible consommation

Les recherches décrites dans cette partie ont été menées dans le cadre du projet européen CATRENE COMCAS (*COmmunication-centric heterogeneous Multi-Core ArchitectureS*) et s'intéressent à l'utilisation de stratégies énergétiques avancées expérimentées sur des plateformes multicœur représentatives, pour des applications basées sur le standard vidéo H.264 (encodeur et décodeur). Ces aspects se basent sur les travaux de thèse Mr. Jabran KHAN JADOON au LEAT et le postdoc de Mr. Muhammad Khurram BHATTI, financés sur le projet COMCAS. Les plateformes utilisées combinent l'utilisation d'une plateforme réelle *ARM 1176JZF-S Platform Baseboard* qui possède des ressources intégrées pour la mesure de consommation, et deux plateformes virtuelles basées sur l'émulateur QEMU incluant des modèles du cœur ARM1176JZF-S précédent et ARM Cortex A9 (basé sur le SoC ST-Ericsson Nova A9500 Application Processor), développés en collaboration avec le TIMA, Thales Communication & Security et ST-Ericsson. Etant donné que les plateformes virtuelles développées au sein du projet COMCAS permettent des simulations SystemC *cycle accurate* de SoCs multiprocesseurs, leur utilisation a été profitable pour compenser l'absence de plateforme équivalente réelle au début de ce projet.

Trois types de stratégies différentes basées sur des techniques DVFS (*Dynamic Voltage and Frequency Scaling*) et DPS (*Dynamic Power Switching*, i.e. exploitation dynamique des modes repos) sont utilisées. La première est une stratégie DVFS conçue pour être adaptée aux contraintes spécifiques des applications vidéo (adaptation de la fréquence processeur en fonction de la vitesse de décodage). Les deux suivantes appartiennent à une classe particulière de politique d'ordonnancement dite "à contraintes d'échéances" (*deadline scheduling*) prototypées grâce à une technique brevetée de d'ordonnancement en espace utilisateur Linux [10]. Deux ordonnanceurs sont considérés: une stratégie DVFS appelé DSF (*Determinitic Stretch-to-Fit*) et une stratégie DPS dénommé AsDPM (*Assertive Dynamic Power Management*). Par la suite, nous étudierons en détail l'efficacité de ces trois stratégies en utilisant un décodeur H.264 opérationnel sur la plateforme *PB ARM1176JZF-S*, et un modèle de tâche d'un encodeur H.264 sur la plateforme virtuelle, développé par Thales Communication & Security, pour les simulations basées QEMU.

Les gains énergétiques des techniques DVFS et DPS sont ainsi mesurés, comparés et analysés par l'utilisation de ces différentes plateformes et applications et dans plusieurs configurations. L'analyse détaillée des nombreux résultats permet de mieux appréhender les gains possibles, les conditions nécessaires et de proposer des perspectives intéressantes pour la définition de stratégies énergétiques plus efficaces, en particulier pour un domaine d'application représentatif et exigeant comme celui de la compression vidéo.

4.1.1 Stratégie DVFS vidéo

Les travaux précédents réalisés sur le décodeur H.264 (section 3.1.2) ont montré que la vitesse de décodage était sensible aux caractéristiques du mouvement des séquences vidéo, avec des variations qui atteignent ±20%. Le principe de la stratégie présentée ici est ainsi d'exploiter ces variations en stabilisant la vitesse de décodage par l'utilisation du DVFS. Le processeur peut être ralenti lorsque la vitesse de décodage est supérieure à la moyenne, pour économiser de l'énergie. Cette adaptation se base sur une contrainte de

vitesse à satisfaire (par exemple 9 images par secondes) qui permet de définir des paliers (en termes d'images par secondes) distincts de fonctionnement pour chacune des fréquences processeur possibles. Les seuils délimitant ces zones peuvent être déterminés de la façon suivante: $tresh_i$ = adaptation_const * f_{nom} / f_i , où f_{nom} représente la fréquence nominale CPU. Pour la plateforme *PB ARM1176JZF-S*, quatre seuils (13.50, 10.05, 9.0 and 8.15 fps) peuvent donc être déduits des quatre points de fonctionnement possibles (160, 215, 240 et 265 MHz) pour une contrainte d'adaptation de 9 img/s par exemple. Ainsi, si le décodeur fonctionne entre 0.0 and 8.15 img/s pendant un certain temps (défini à 250 images), le point de fonctionnement choisi est de 265MHz. S'il se situe entre 8.15 et 9.0 img/s, le point de fonctionnement choisi est de 240MHz, etc. Par la suite, nous allons analyser les gains énergétiques obtenus par cette stratégie en pratique sur le décodeur vidéo H.264.

Un résumé des mesures sur la plateforme *PB ARM1176JZF-S* est donné au Tableau 18, avec les valeurs d'énergie totale, de puissance moyenne, de temps de décodage (en secondes, sur une séquence de 1944 images), et le temps moyen de décodage pour une image (ms) pour différentes contraintes d'adaptation. Ces résultats montrent une augmentation du coût en énergie pour des contraintes d'adaptation de 4, 8 et 9 img/s. Etant donné que dans chaque cas, la contrainte se situe en dessous de la performance moyenne (11,6 img/s à la fréquence nominale de 240MHz), la fréquence CPU est bien réduite par la stratégie mais cela ne se traduit pas ici par une diminution de l'énergie.

Adaptation Constraint (fps)	Total Energy (J)	Average Power (mW)	Decoding time (s)	Average time/frame (ms)	fps	Energy gain (%)
Not used	52	311	168	86.4	11.6	0
4	64	262	244	126.1	7.9	-23
8	62	270	228	117.6	8.5	-19
9	56	290	196	101.4	9.9	-8
11	52	311	168	86.5	11.6	0
16	50	325	157	81.2	12.3	4

Tableau 18 - Gains en énergie de la stratégie DVFS vidéo.

La raison de ce résultat inattendu tient à l'augmentation du temps d'exécution (résultant de la diminution de la fréquence CPU) qui augmente donc l'énergie consommée au point d'annuler la diminution de puissance attendue par la réduction de fréquence. De par la relation E= P*T, la diminution de puissance n'est pas suffisante pour compenser l'augmentation du temps d'exécution sur la plateforme *PB ARM1176JZF-S*. Cet effet adverse est dû aux caractéristiques des points de fonctionnement sur cette plateforme. Par exemple, le ralentissement de 240 à 215 MHz implique une augmentation du temps d'exécution de 10,4% pour une diminution de 6,45% en puissance (courbe bleue Figure 23). Ainsi, le produit P*T correspondant reste supérieur après le passage de 240 à 215 MHz. Par la suite, nous analysons plus en détail cet effet grâce à la plateforme virtuelle qui permet de modifier le niveau de puissance associé aux points de fonctionnement. La Figure 23 montre les caractéristiques des points de fonctionnement originaux de la plateforme *PB ARM1176JZF-S* (courbe bleue). Trois configurations supplémentaires dénommées

config$_1$, *config$_2$* et *config$_3$* sont définies pour traduire différents niveaux de puissance en charge pour les fréquences possibles du processeur (160, 215, 240 et 265MHz).

Les trois configurations sont définies pour faire varier de façon régulière la différence en niveaux de puissance pour chaque fréquence. Par exemple, le passage de 240 à 215 MHz correspond à une réductiond e 20, 30, 50 ou 90 mW respectivement pour les configurations *ARM1176*, *config$_1$*, *config$_2$* et *config$_3$*.

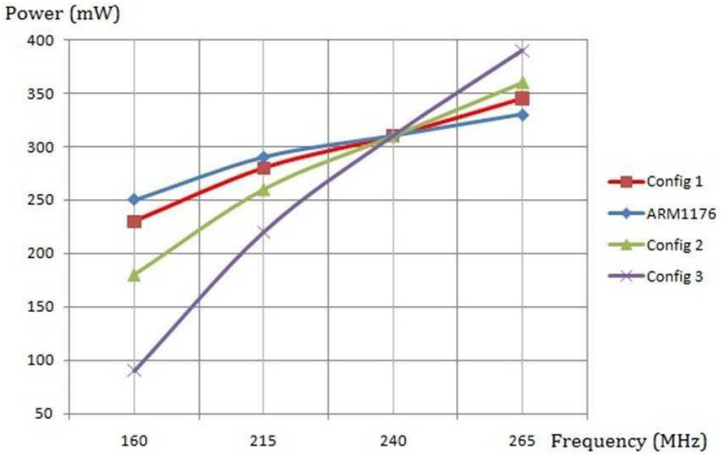


Figure 23 - Configuration des points de fontionnement sur la plateforme virtuelle.

Les gains énergétiques correspondants sont visibles à la

Figure **24**, pour des contraintes d'adaptation de 4, 6 et 8 img/s. Comme les trois contraintes se situent en dessous de la vitesse de décodage nominale de 11,6 img/s, la stratégie DVFS procède à une diminution de la fréquence dans les trois cas. Néanmoins, les résultats indiquent que seules les configurations *config$_2$* et *config$_3$* réduisent le coût énergétique, ceci parce que, dans le cas des configurations *ARM1176* et *config$_1$*, les différences de niveaux de consommation des points de fonctionnement consécutifs ne permettent pas de compenser l'effet de l'augmentation du temps d'exécution.

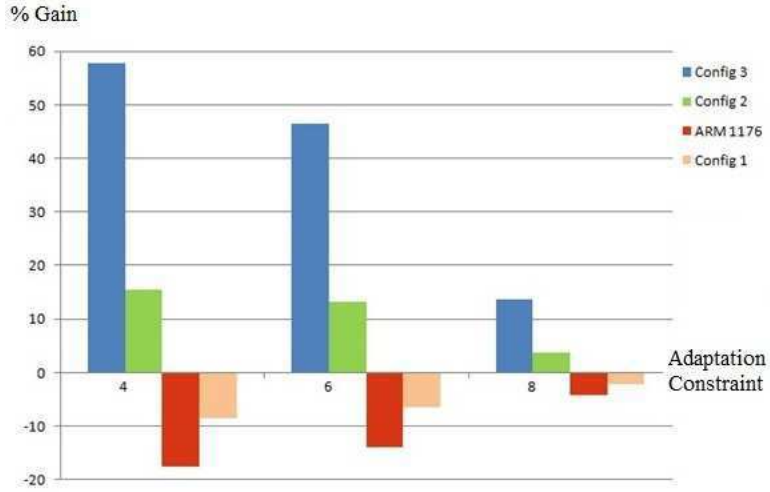


Figure 24 - Gains énergétiques en fonction de la contrainte d'adaptation pour différentes configurations.

Ces résultats soulèvent trois points importants. Tout d'abord, les platefomes réelles ne garantissent pas toujours, en pratique, une réduction de l'énergie lorsque l'on diminue la fréquence, à cause de l'inefficacité possible des points de fonctionnement. C'est le cas de la plateforme *PB ARM1176JZF-S* utilisée pour cette expérimentation, mais aussi pour d'autres plateformes pour différentes raisons matérielles. Cette plateforme d'évaluation est en effet une version de développement spécifique qui n'opère pas à fréquence représentative en raison des contraintes de prototypage et de debug (utilisation d'un FPGA pour certaines fonctionalités de contrôle). La fréquence maximale est limitée à 265 MHz alors qu'une version de production pourrait fonctionner jusqu'à 800 MHz, ce qui affecte la définition et la pertinence des points de fonctionnement (cf étude sur le CT11 MPCore section 3.1.3). Ensuite, les caractéristiques de ces points de fonctionnement sont des paramètres déterminants dans l'efficacité du DVFS. En particulier, les différences de niveau de puissance entre fréquences consécutives ont un impact majeur sur le niveau des gains énergétiques possibles. Dans les conditions de mesures précédentes, la variation des gains énergétiques

atteint des valeurs très significatives (jusqu'à 3,6 fois moins entre $config_2$ et $config_3$). Finalement, les résultats montrent également jusqu'à 50% de gains énergétiques possibles en fonction du paramètre applicatif régulant dont dépend la stratégie (ici, une certaine vitesse de décodage). Les stratégies standards existantes dans les systèmes d'exploitation sont des politiques generales (dans le sens où elles peuvent être utilisées pour n'importe quelle application) qui sont basées sur la charge CPU pour la plupart. Pour des charges CPU importantes comme c'est typiquement le cas pour les applications vidéo, elles conduisent à opérer à la fréquence la plus importante sur toute la durée du traitement, et donc à une consommation maximale. Cette observation montre le potentiel de stratégies dédiées, plus spécifiques, qui s'adaptent finement au type de traitement. Elles sont potentiellement plus efficaces car optimisées pour une application ou classe d'applications, par rapport à des stratégies standards et particulièrement pour les applications haute performance. C'est par ailleurs ce que nous allons vérifier sur les deux stratégies suivantes qui ciblent elles une classe plus étendue d'applications ayant des contraintes d'exécution temps-réel.

4.1.2 Stratégie DVFS temps réel : DSF

La stratégie DSF (*Determintic Stretch-to-Fit*) est une stratégie d'ordonnancement dite "à contraintes d'échéances" (*deadline scheduling*) dont le principe est de réduire la fréquence CPU et la puissance associée grâce au DVFS. Elle exploite le fait que le temps d'exécution effectif d'une tâche (*Actual Execution Time*, AET) est inférieur ou égal au pire cas (Worst Case Execution Time, WCET) en pratique. Le principe est illustré à la Figure 25 où la terminaison avancée de la tâche A produit une marge exploitable pour l'exécution de la tâche suivante (*slack time*). Le temps supplémentaire disponible pour la tâche B permet de ralentir le processeur concerné et donc de diminuer la consommation associée.

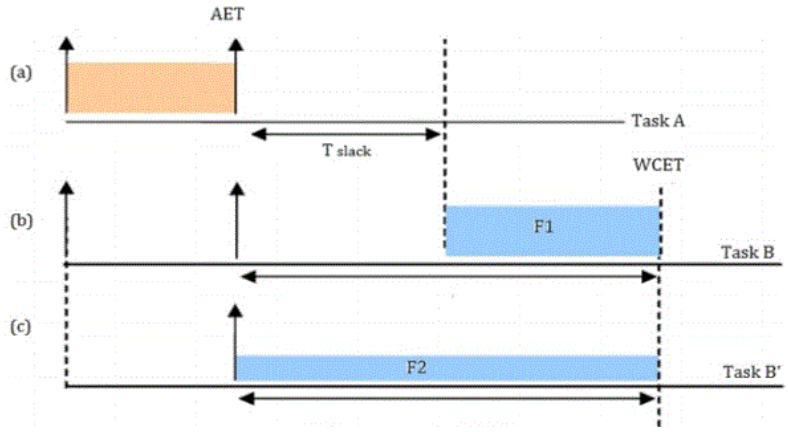


Figure 25 - Principe de la stratégie DSF.

Pour vérifier l'efficacité de cette stratégie en pratique, DSF est porté sur les deux plateformes virtuelles QEMU ARM1176 et QEMU Cortex A9 grâce à une technique brevetée en espace utilisateur Linux [10]. L'application test utilisée dans ce cas est un encodeur vidéo H.264 parallélisé pour une exécution *dual core* dont le bloc diagramme est donné Figure 26. Dans cette version, le module d'estimation de mouvement, qui représente la part la plus complexe de l'encodeur, est divisé en deux parties pour paralléliser les traitements. Un modèle de tâche de l'encodeur avec les mêmes caractéristiques temporelles que l'encodeur réel (développé par Thales Communications & Security), mais réalisant des calculs simplifiés, est utilisé en pratique pour faciliter les simulations de la plateforme virtuelle. Les valeurs temporelles de ce modèle de tâche (WCET, BCET, deadline et période) sont celles d'un encodeur réel profilé sur plateforme

ST-Ericsson Nova A9500 (Dual Cortex A9). Ainsi, la variation de *slack* du BCET au WCET pour les deux tâches d'estimation de mouvement T1 et T2 nous permet de d'analyser la variation des gains énergétiques pour des cœurs ARM1176 et Cortex A9 en configuration *dual core*.

La Figure 27 montre les gains énergétiques correspondants, qui varient de 5,93% à 16,70% sur QEMU ARM1176 et de 10,21% à 51,46% sur QEMU CortexA9. QEMU CortexA9 permet une efficacité énergétique 1,8 fois meilleure en moyenne, sous les mêmes conditions en termes d'applications (*slack*) et de mesures.

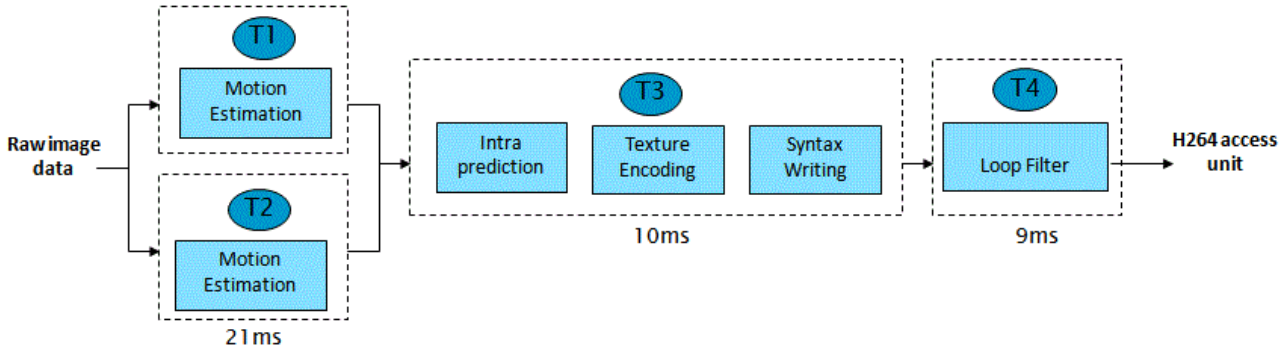


Figure 26 - Bloc diagramme encodeur H.264 .

Comme la fréquence est déterminée à partir du *slack* de la tâche précédente et du WCET de la tâche suivante, les niveaux de puissance consommée associés aux fréquences CPU sont seuls responsables des différences de gains énergétiques observées.

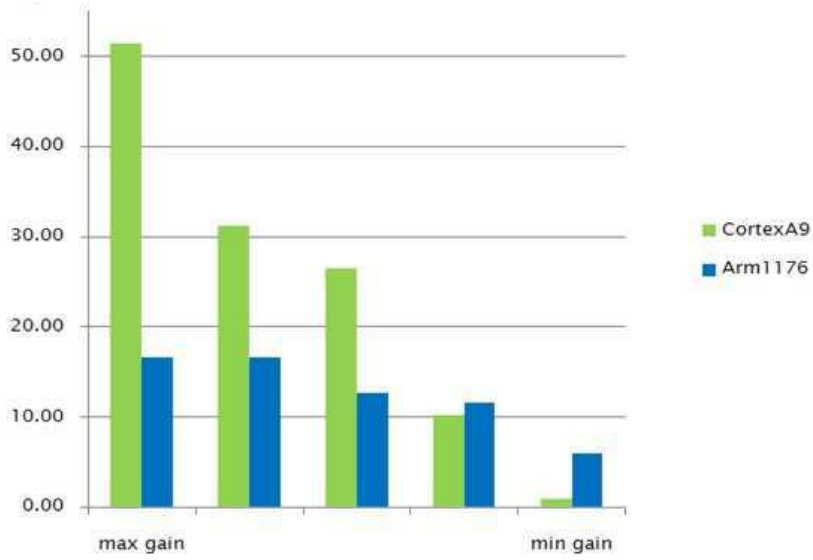


Figure 27 - Gains énergétiques DSF pour l'encodeur H.264.

En effet, on peut vérifier avec le Tableau 19 que comparativement, les points de fonctionnement ont des caractéristiques très différentes sur les deux plateformes. Par exemple, quand le Cortex A9 passe de la fréquence maximum (1 GHz) à la fréquence minimum (300 MHz), la puissance active consommée (en charge) diminue de 177 mW. La diminution de puissance correspondante est de 80 mW pour l'ARM1176 (pour passer de 265 à 160 MHz). Comme les différences de niveaux de puissance sont plus importantes entre les points de fonctionnement du Cortex A9, la puissance peut être réduite d'avantage que sur ARM1176 par une diminution de fréquence et les gains énergétiques sont meilleurs. Un autre facteur qui

conditionne l'efficacité de la stratégie DSF est illustré à la Figure 28. Pour ces mesures, nous avons utilisé un simple exemple (2 tâches, 1 cœur) pour vérifier l'effet des latences de changement de fréquence par rapport à la granularité temporelle des tâches applicatives.

Platform	Freq. (MHz)	Idle P. (mW)	Load P. (mW)
ARM1176	160	223	250
	215	238	290
	240	245	310
	265	252	330
Cortex A9	300	38	143
	600	60	215
	1000	90	320

Tableau 19 - Puissance à vide et puissance en charge pour QEMU ARM1176 et QEMU Cortex A9.

Les temps d'exécution des tâches sont modulés par un facteur multiplicatif de 10^0, 10^1, 10^2, 10^3. Il apparaît clairement dans les résultats que les tâches de l'ordre de quelques millisecondes provoquent rapidement une perte d'efficacité énergétique. Elle vient du fait que, à mesure que l'on se rapproche des latences de changement de fréquence (de l'ordre de quelques centaines de microsecondes), le coût énergétique associé au changement de fréquence devient significatif en proportion et altère l'efficacité de la stratégie DSF.

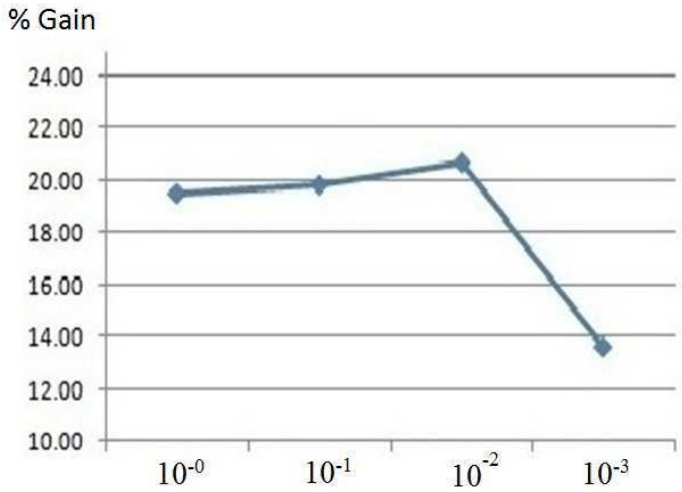


Figure 28 - Gains énergétiques en fonction de la granularité temporelle des tâches.

Ces différents résultats de mesure montrent la capacité de la stratégie DSF à réduire le coût énergétique pour deux plateformes, avec des gains énergétiques allant jusqu'à 18% pour ARM1176 et 52% pour Cortex

A9. Ces résultats soulignent encore le potentiel de stratégies avancées pour réduire d'avantage l'énergie, là où une approche généraliste basée sur la charge CPU (*workload*) conduirait à la fréquence et à la consommation maximale pour des charges importantes comme celles que représentent typiquement les algorithmes de compression vidéo. Le paramètre applicatif régulant la stratégie DSF est le *slack* dynamique. Sous les mêmes conditions de mesures et de *slack* applicatif, il existe un facteur notable de 1,6 en efficacité énergétique entre ARM1176 et Cortex A9. Comme pour la stratégie DVFS vidéo précédente, les gains énergétiques sont fortement conditionnés par les caractéristiques de la plateforme, et en particulier des différences de niveaux de puissance entre les points de fonctionnement. Enfin, les temps d'exécution des tâches applicatives entrent en compte. Elles doivent être suffisamment grandes devant les délais de changement de fréquences pour qu'une stratégie DVFS soit applicable. En d'autres termes, toutes les applications ne sont pas forcément compatibles avec une stratégie DVFS, en particulier lorsque les temps de tâches sont de l'ordre de la centaine de microsecondes, voire de la milliseconde en fonction de la qualité des drivers DVFS de la plateforme d'exécution.

4.1.3 Stratégie DPS temps réel : AsDPM

AsDPM (*Assertive Dynamic Power Management*) est une autre stratégie d'ordonnancement faible consommation multiprocesseur. Son principe consiste à grouper l'exécution des tâches applicatives sur un nombre minimal de processeurs dans le but de placer les cœurs les moins actifs en mode repos le plus longtemps possible. L'exemple de la Figure 29 illustre ce principe par la distribution de trois tâches sur deux cœurs. L'inactivité des tâches est segmentée dans l'ordonnancement EDF original (Figure 29a).

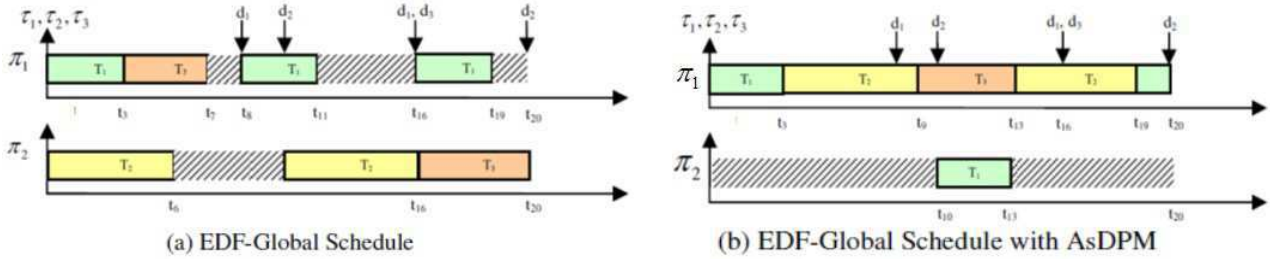


Figure 29 - Principe de la stratégie AsDPM.

L'ordonnanceur AsDPM groupe l'exécution des tâches sur le processeur $\prod_1$; ainsi une économie d'énergie importante peut être faite en plaçant le processeur $\prod_2$ dans un mode endormi sur une durée étendue (Figure 29b).

Par la suite, nous examinerons l'efficacité de cette stratégie d'ordonnancement portée sur la plateforme virtuelle multiprocesseur grâce à la technique de développement en espace utilisateur Linux mentionnée précédemment [10]. D'après la définition précédente, AsDPM nécessite au moins deux processeurs pour être applicable. L'ordonnanceur est développé sur les deux plateformes virtuelles QEMU ARM1176 et QEMU Cortex A9 (chacune en configuration *dual core*) et appliqué à l'exemple d'encodeur H.264 précédent. Au début d'une hyper période, l'encodeur utilise les deux cœurs pour exécuter les deux tâches parallèles d'estimation de mouvement (T1 et T2, Figure 26). Tandis que T3 et T4 sont séquentielles et ne nécessitent qu'un seul cœur. Le second processeur peut donc être endormi pour une durée qui dépend des valeurs des *slacks* dynamiques produits par T1 et par T2. Nous avons défini six valeurs de temps d'exécution pour T1 et T2, qui correspondent aux six solutions représentées à la Figure 30, de façon à caractériser les gains minimum et maximum en énergie. Cette façon de procéder concernant la variation du *slack* permet également d'établir une comparaison fiable des gains énergétiques sur les deux plateformes.

La Figure 30 montre que AsDPM permet de réduire l'énergie consommée pour les deux plateformes, de 24,05% à 46,73% pour ARM1176 et de 15,32% à 42,72% pour Cortex A9. AsDPM est 1,4 fois plus efficace sur ARM1176, alors que DSF était meilleur précédemment sur Cortex A9. Ces différences viennent encore des caractéristiques des points de fonctionnement, mais cette fois en termes de puissance à vide (*idle power*) et en charge (*load power*). Des mesures ont été effectuées sur les deux plateformes à fréquence maximale, respectivement 265 MHz et 1 GHz pour ARM1176 et Cortex A9 (Tableau 19). Il est intéressant de constater que pour ARM1176, la puissance à vide (252 mW) est importante par rapport à la puissance en charge (330 mW), alors qu'il y a une différence plus sensible pour Cortex A9 (90 et 320 mW). Par conséquent, le fait de placer un cœur dans un état où la puissance est très réduite voire proche de zéro, réduit une plus grande proportion de puissance sur ARM1176 que sur Cortex A9. La consommation à vide importante sur ARM1176 résulte comme précédemment de l'utilisation d'une plateforme d'évaluation avec certaines limitations matérielles. Néanmoins, ces résultats montrent la capacité de la stratégie AsDPM à être efficace pour des systèmes ayant une forte part de consommation *idle*.

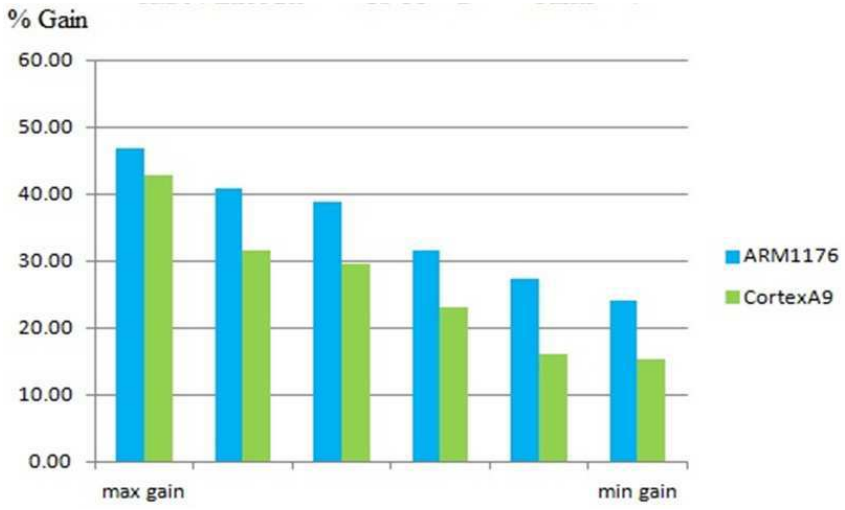


Figure 30 - Gains énergétiques AsDPM pour un encodeur H.264.

L'efficacité pratique de la stratégie AsDPM a été démontrée sur la base d'un prototypage réel de l'ordonnanceur sur les deux plateformes virtuelles, avec des gains énergétiques allant jusqu'à 46% pour ARM1176 et 42% pour Cortex A9 en fonction des variations du paramètre applicatif régulant (*slack* dynamique). Le niveau des gains énergétiques est aussi déterminé par les niveaux de consommation à vide de la plateforme qui dépendent des points de fonctionnement. AsDPM est particulièrement adapté pour des systèmes ayant une forte proportion de consommation *idle*, qui est étroitement liée aux courants de fuites caractéristique des technologies d'intégration nanométriques récentes. Comme pour les stratégies DVFS, les latences d'entrée et de sortie des différents modes repos doivent pouvoir être négligées par rapport aux temps d'exécution des tâches applicatives. Si cette condition n'est pas respectée, le coût d'un changement d'état actif/endormi dégrade l'efficacité énergétique de la stratégie, et aboutira aussi probablement à une mauvaise exécution de l'application. Comme ces latences sont en général plus importantes que celles d'un changement de fréquence DVFS, AsDPM et plus généralement les techniques exploitant les modes repos (DPS) sont susceptible s'appliquer moins souvent que les stratégies DVFS, c'est le cas en particulier des applications de compression vidéo qui doivent satisfaire des contraintes typiques de 15 à 40ms par image.

4.1.4 Conclusion

A partir de ces trois stratégies étudiées sur des plateformes de type ARM représentatives, les résultats permettent d'avoir une meilleure compréhension des contraintes qui affectent l'efficacité des stratégies de gestion énergétique en pratique et d'envisager plusieurs possibilités d'amélioration. Les gains réels en énergie sont très fortement dépendants des caractéristiques matérielles de la plateforme d'exécution, en particulier des niveaux et latences de transition associés aux points de fonctionnement, à la fois pour les stratégies DFVS et DPS. Ceci souligne l'importance d'une analyse préliminaire des caractéristiques matérielles pour définir des stratégies efficaces en pratique. Si les conditions matérielles sont satisfaites, l'efficacité des stratégies dépend également de leur spécialisation et de leur pertinence face aux traitements. La capacité de politiques plus dédiées (aux applications ou à un domaine d'application) à exploiter une connaissance fine des applications permet de réaliser des gains supérieurs, jusqu'à 50% d'après nos expérimentations sur des applications vidéo, là où des stratégies généralistes existantes seraient bien moins efficaces et consommeraient plus. Ces résultats suggèrent aussi que des réductions très importantes de la consommation sont possibles en combinant l'usage de différentes stratégies en fonction du contexte. Pour une utilisation plus efficace des possibilités, une gestion énergétique intelligente serait par exemple d'utiliser des stratégies DVFS/DPS dédiées pour des charges CPU très critiques (e.g. vidéo), et des politiques généralistes dans les autres cas.

4.2 Ordonnancement hétérogène faible consommation

Les travaux décrits par la suite abordent l'étude de stratégies énergétiques pour des plateformes multiprocesseur hétérogènes, qui intègrent en particulier des accélérateurs matériels reconfigurables et la possibilité de Reconfiguration Dynamique Partielle. Les décisions de l'ordonnanceur sont un élément encore plus déterminant dans ce contexte, car si l'hétérogénéité permet une meilleure efficacité des traitements en théorie, de mauvais choix d'allocation sur les unités d'exécution ont de fortes chances d'augmenter la consommation au lieu de la réduire. L'approche qui est développée pour garantir de meilleurs choix se base donc sur l'utilisation d'une modélisation en puissance et énergie assez efficace pour permettre une prise de décision à la fois rapide et fiable pour répartir les tâches entre les cœurs et les régions reconfigurables en fonction de l'impact énergétique réel. Ce coût est calculé sur la base des modèles et de la formalisation développés précédemment, qui ont été entièrement intégré à l'environnement FoRTReSS [9] au cours du projet CATRENE BENEFIC à travers le postdoc de Mr. Robin BONAMY.

4.2.1 Spécification de la stratégie

Le processus d'ordonnancement se déroule principalement en deux étapes: l'établissement d'une liste de tâches prêtes et triées selon un critère de priorité (e.g. EDF), puis une ressource d'exécution est allouée à l'exécution d'une implémentation compatible de la tâche prioritaire en accord avec un critère d'efficacité énergétique. Ce procédé est dénommé *Energy Aware Heterogeneous Scheduling* (EAHS) par la suite.

Une grande partie des stratégies d'ordonnancement usuelles se basent sur des stratégies de type FIFO (premier arrivé, premier servi), des stratégies d'ordonnancement contraintes par échéances (e.g. EDF) et des stratégies d'ordonnancement par priorités statiques. Les principes fondamentaux de ces stratégies peuvent être modifiés pour être plus adaptés aux contraintes d'un ordonnancement hétérogène. Par exemple, une tâche en tête de la liste des tâches prêtes peut être bloquée par l'absence d'une unité d'exécution compatible et disponible, qui peut bloquer l'exécution d'autres tâches sur d'autres ressources d'exécution libres. Ceci entraînerait à la fois un ralentissement et une sous-utilisation des ressources. Nous proposons ici une approche d'ordonnancement assouplie qui consiste à utiliser une des stratégies précédentes, mais en laissant la possibilité d'ignorer une tâche bloquée à l'exécution de son implémentation (e.g. parce que le contrôleur de reconfiguration est occupé, ou parce qu'aucune unité d'exécution compatible n'est disponible).

Lors du déploiement des tâches sur les unités d'exécution, le choix d'une implémentation a une influence majeure sur la consommation. Une fonction de coût est ainsi définie pour permettre d'identifier la meilleure solution énergétique parmi les différentes possibilités. Ce coût est évalué à chaque fois qu'une tâche est prête à être exécuté (pour chacune de ses implémentations possibles) en se basant sur des estimations du temps d'exécution et de l'énergie des implémentations analysées. L'estimation énergétique est donnée par l'expression :

$$\forall j \ ; \ E_j^{cost} = \left\{ \begin{array}{ll} E_{i,j}^{run} & \text{when } \rho_{i,j} = 0 \\ E_{i,j}^{run} + E_j^{conf} & \text{when } \rho_{i,j} = 1. \end{array} \right\}$$

où

$$E_{i,j}^{run} = P_{i,j}^{run} \times T_{i,j}$$
$$E_{j}^{conf} = T_{RR_j}^{reconf} \times P^{reconf}$$

et $\rho_{i,j}$ formalise le besoin de faire une reconfiguration. Par exemple, si la même implémentation est déjà configurée, $\rho_{i,j} = 0$ signifie que la région reconfigurable RR_j peut être directement utilisée pour exécuter la tâche *i*. Autrement $\rho_{i,j} = 1$ et une reconfiguration est nécessaire. Pour toutes les ressources non reconfigurables, $\rho_{i,j} = 0$.

Pour exécuter une nouvelle tâche, l'ordonnanceur doit d'abord vérifier si l'unité d'exécution EU_j en cours d'évaluation est libre. Si ce n'est pas le cas, la tâche pourra être implémentée plus tard et le retard impliqué doit être pris en compte. Le temps d'exécution de la tâche est ainsi estimé par l'équation :

$$\forall j \; ; \; T_j^{cost} = T_{i,j} + T_{RR_j}^{reconf} \times \rho_{i,j} + T_j^{busy}$$

où T_j^{busy} représente le temps durant lequel EU_j est occupé par l'exécution d'une autre tâche *i'*. Il est déduit du temps d'exécution de $T_{i',j}$, du début de la tâche en cours et du temps au moment de l'ordonnancement.

Le coût final pour la ressource EU_j est donné par :

$$\forall j \; ;$$
$$Cost_j = \alpha \frac{E_j^{cost}}{max(E^{cost})} + (1-\alpha) \frac{T_j^{cost}}{max(T^{cost})}$$

où α est un paramètre qui peut prendre une valeur réelle entre 0 et 1 de façon à favoriser les performances (α proche de 0) ou la consommation (α proche de 1). Cette fonction de coût est évaluée pour toutes les implémentations et unités d'exécution de la tâche en tête de la liste des tâches prêtes. La valeur de $Cost_j$ la plus faible est retenue et la tâche est implémentée sur la ressource EU_j. Néanmoins, si EU_j est occupée, la tâche reste en attente et sera implémentée plus tard.

4.2.2 Validation de la stratégie

L'application considérée dans cette étude de cas se base sur le décodeur vidéo H.264/AVC LETI (section 3.2.1). A partir d'une analyse du code (*profiling*), six fonctions principales sont identifiées et définissent le graphe de tâche suivant :

Γ = *{Exp_Golomb, MB_Header, Inv_CAVLC, Inv_QTr, Inv_Pred, DB_Filter}*

La méthodologie de conception décrite au chapitre 3.2 est utilisée pour produire rapidement des accélérateurs entièrement opérationnels pour les trois fonctions matérielles possibles. Le filtre de sortie anti-bloc (*DB_Filter*), le décodage entropique CAVLC (*Inv_CAVLC*) et la quantification / transformée inverse (*Inv_QTr*) contribuent ensemble à 76% du temps d'exécution global sur un processeur mono cœur. Ces blocs représentent les trois fonctionnalités du décodeur qui peuvent être générées par Synthèse de Haut Niveau (HLS) pour accélération matérielle et peuvent donc donner lieu à une exécution logicielle ou matérielle.

Le standard H.264 offre par ailleurs des possibilités de parallélisation avec une décomposition de l'image en tranches. Une tranche H.264 est une zone indépendante d'une image, comme le sont par exemple deux moitiés horizontales d'une image. Ainsi, le traitement d'une tranche (d'une image) est indépendant de celui

d'une autre tranche (de la même image). Le décodeur peut alors traiter différentes tranche H.264 d'une image en parallèle, ce qui constitue une optimisation très intéressante pour des architectures multi processeur symétriques (SMP). Nous avons ainsi considéré une décomposition de l'image où quatre flux procèdent au décodage de quatre tranches H.264 en parallèle.

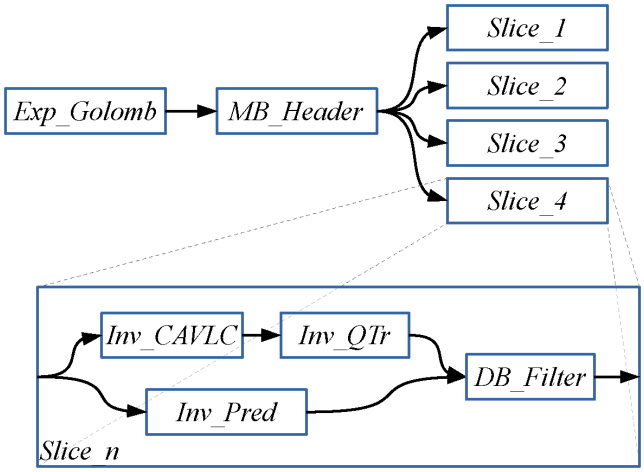


Figure 31 – Graphe de tâches du décodeur H.264 parallélisé par slices.

La carte de développement utilisée dans cette étude applicative est une plateforme Zynq ZC702 composée d'un processeur dual cœur ARM Cortex-A9 et d'un FPGA Artix-7 Xilinx. Cette carte possède des ressources intégrées pour permettre de mesurer la consommation de l'alimentation du système CPU/FPGA. Par la suite, une campagne de mesure est réalisée sous les conditions suivantes: tension d'alimentation 1V, température ambiante 22°C, fréquence des cœurs ARM 667MHz et fréquence FPGA 100MHz. Les valeurs de puissance reportées au Tableau 20 sont issues des mesures directes sur la carte d'évaluation ZC702.

Model	Value	Model	Value
P_{core}^{static}	89,82 mW	P^{reconf}	72 mW
P_{core}^{idle}	55,83 mW	T_{RR1}^{reconf}	1,36 ms
P_{core}^{run}	119,4 mW	T_{RR2}^{reconf}	1,36 ms

Tableau 20 - Paramètres du modèle pour la plateforme Zynq ZC702

Le graphe de tâches correspondant à l'exemple du décodeur H.264 parallélisé par *slices* est illustré à la Figure 31. Dans ces conditions, les régions reconfigurables (*Partially Reconfigurable Regions*, PRRs) définies pour la reconfiguration dynamique partielle sont dimensionnées pour pouvoir accueillir chacune toute tâche matérielle de l'application en maximisant le parallélisme, en suivant la méthodologie FoRTReSS. Les deux PRRs ainsi définies sont ensuite implémentées et caractérisées en termes de puissance et de temps d'exécution sur la plateforme ZC702. Les paramètres logiciels et matériels du modèle de tâche sont donnés au Tableau 21.

La stratégie d'ordonnancement proposée est comparée à quatre autres stratégies. Le paramètre α est mis à la valeur 0,8 pour favoriser l'efficacité énergétique sans négliger complètement les performances. L'analyse de l'ordonnancement du décodeur H.264 est réalisée par le simulateur SystemC/TLM intégré à FoRTReSS, dont les résultats sont résumés au Tableau 22 et à la Figure 32. Ils fournissent l'occupation des ressources FPGA (*slices* Xilinx) en fonction du temps d'exécution et l'énergie consommée pour décoder une image. Le

produit énergie performance est également calculé pour fournir une mesure supplémentaire de l'efficacité énergétique.

Task i	EU j	$P_{i,j}^{run}$ (mW)	$P_{i,j}^{idle}$ (mW)	$T_{i,j}$ (ms)
Exp_Golomb	$core_1$ $core_2$	119,4	--	6
MB_Header	$core_1$ $core_2$	119,4	--	5,9
Inv_CAVLC	$core_1$ $core_2$	119,4	--	6,6
	PRR_1 PRR_2	33,5	55,1	4,5
Inv_QTr	$core_1$ $core_2$	119,4	--	3,1
	PRR_1 PRR_2	5,8	34,2	1,5
Inv_Pred	$core_1$ $core_2$	119,4	--	3,2
DB_Filter	$core_1$ $core_2$	119,4	--	10,5
	PRR_1 PRR_2	6,4	44,3	0,9

Tableau 21 – Paramètres logiciels et matériels du modèle de tâche pour le décodeur H.264 sur la plateforme Zynq.

Deux solutions ressortent en ce qui concerne l'ordonnancement EDF pour une exécution uniquement logicielle (*EDF Software*). La première est définie pour une plateforme cible composée de deux cœurs mais ne respecte pas la contrainte temps réel (58,7 ms par image). Comme l'hyper période doit se trouver sous la limite des 40 ms pour cette application, le modèle pour la plateforme est étendu jusqu'à un maximum de quatre cœurs ARM Cortex-A9. Une autre solution caractéristique est donnée par l'ordonnancement qui fait intervenir les cœurs avec des accélérateurs « statiques », c'est à dire sans reconfiguration dynamique (*EDF static*). En effet dans cette configuration de parallélisme, chaque *slice* H.264 utilise trois accélérateurs ce qui conduirait à un total de 4*3 accélérateurs (39536 *slices* Xilinx) difficiles à implémenter sur un composant existant. Comme il peut y avoir une réutilisation des accélérateurs entre les tranches H264, une autre solution d'ordonnancement est définie (*EDF SmartStatic*) en se basant sur l'utilisation d'un seul accélérateur pour chaque fonction matérielle (*DB_Filter*, *Inv_CAVLC*, *Inv_QTr*) quelque soit le nombre de tranches H264.

Scheduler	N# cores	FPGA area (eq. slices)	Execution time (ms)	Energy consumption (mJ)	Energy-delay product
EDF Software	2	0	**58,7**	29,67	1742
EDF Software	**4**	0	38,54	35,02	1350
EDF Static	2	**39536**	19,31	21,09	407
EDF SmartStatic	2	6372	35,4	19,47	690

EAHS	2	6600	29,8	16,04	478

Tableau 22 – Comparaison de différentes stratégies d'ordonnancements sur le décodeur H.264.

L'exécution logicielle consomme une grande quantité d'énergie (35 mJ) alors que la solution statique bénéficie d'une meilleure efficacité matérielle avec 21,1 mJ (*EDF Static*) et 19,5 mJ (*EDF SmartStatic*). La solution *EDF SmartStatic* (35,4 ms) est néanmoins plus lente que la solution *EDF Static* (19,3 ms) car le nombre limité d'instances accélérateur limite aussi les possibilités d'exécution en parallèle. Le résultat est par ailleurs visible sur le produit énergie – performance: *EDF Static* obtient le meilleur score (407) suivi par *EDF SmartStatic* (960) et *EDF Software* (1350). En ce qui concerne les solutions RDP (deux cœurs, deux PRRs), le score énergie – performance d'*EAHS* est proche de celui d'*EDF Static* (478). Dans ce cas, la consommation d'énergie diminue de 21,1 mJ à 16 mJ par rapport à *EDF Static* tandis que le temps d'exécution augmente à cause des reconfigurations et du nombre de ressources matérielles plus faible. De plus, il est intéressant de noter qu'*EAHS* définit une solution qui consomme 3,4 mJ de moins en étant 5,6 ms plus rapide qu' *EDF SmartStatic* grâce à une meilleure utilisation des ressources reconfigurables.

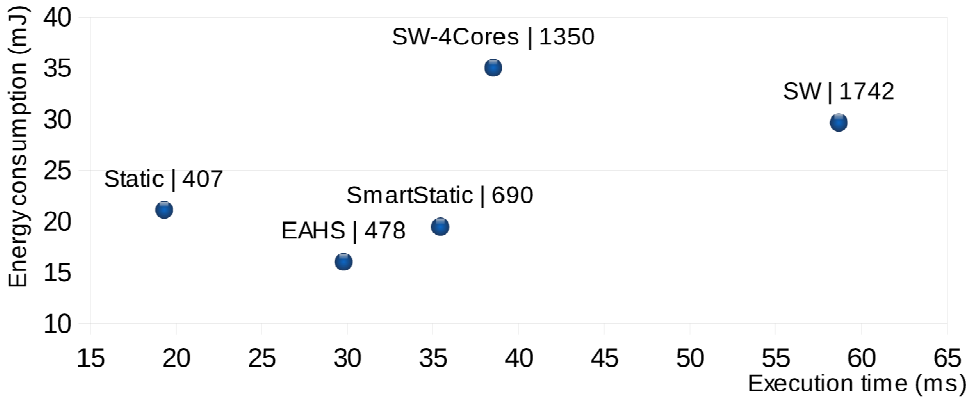


Figure 32 – Energie consommée et produit énergie - performance pour différents ordonnancements.

Bien que ces résultats soient indicateurs d'une amélioration de l'efficacité énergétique, ils ne sont pas tout à fait réalistes compte tenu des contraintes d'exécution réelles qui consistent à traiter les images périodiquement, toutes les 40 ms typiquement. Dans ce cas, l'objectif n'est plus tellement de minimiser le produit énergie – performance mais de satisfaire la contrainte temps-réel au coût énergétique le plus faible. Les résultats précédents sont donc ajustés au Tableau 23 pour considérer le coût énergétique sur une période de 40 ms par image.

40 ms period energy (mJ)	Energy gain	EDF Sw 4 cores	EDF Static	EDF SmartStatic	EAHS
35,87	**EDF Sw 4 cores**	--	9,9%	-66,7%	-78,8%
39,81	**EDF Static**	-11,0%	--	-85,0%	-98,5%
21,52	**EDF SmartStatic**	40,0%	45,9%	--	-7,3
20,06	**EAHS**	**44,1%**	**49,6%**	**6,8%**	--

Tableau 23 – Comparaison de la consommation pour différents ordonnanceurs sur le décodeur H.264.

vraisemblableLes chiffres montrent qu'*EDF Static* consomme plus d'énergie que les quatre autres ordonnancements, avec près de 40 mJ. Ces résultats sont dus à l'importante consommation statique et *idle* qui découle du grand nombre d'accélérateurs lorsqu'ils ne sont pas utilisés. Cette solution, qui avait précédemment le meilleur produit énergie – performance, se trouve en réalité être la plus mauvaise si on considère l'effet de la consommation *idle*. *EAHS* tire avantage de l'utilisation de moins de ressources reconfigurables pour atteindre une consommation de 20,06 mJ, c'est-à-dire environ 50% moins qu'*EDF Static* et 6,8% moins qu'EDF *SmartStatic*. Le profil d'ordonnancement EAHS est présenté à la Figure 33 pour montrer l'utilisation de chaque unité d'exécution au cours du temps sur deux hyper périodes de 40 ms. Comme il a été défini dans le modèle de la plateforme, deux cœurs Cortex A9 sont utilisés avec deux régions reconfigurables. Les reconfigurations sont bien visibles (en couleur rose) entre des exécutions de tâches différentes sur les deux PRRs.

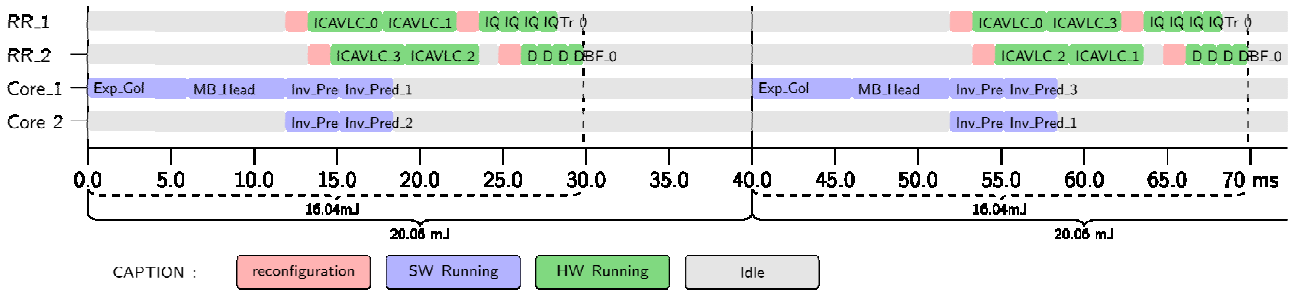


Figure 33 – Allocation et ordonnancement EAHS du décodeur H.264 parallélisé par slices.

Les quatre appels de chaque accélérateur matériel sont configurés successivement. Les premières quatre invocations de la fonction *Inv_CAVLC* sont exécutées en utilisant les deux PRRs, de façon à pouvoir être exécutées en parallèle par paires. Ensuite, une PRR est utilisée pour exécuter séquentiellement les quatre appels à la fonction *Inv_Qtr*, l'autre PRR est utilisée pour les quatre appels à la fonction *DB_Filter*. On voit très bien dans cet exemple applicatif la capacité de l'ordonnanceur à favoriser une exécution matérielle (meilleure efficacité énergétique), à tirer avantage d'exécutions en parallèle et à minimiser le nombre de reconfigurations (qui ont un coût énergétique non négligeable). Tous ces facteurs représentent des conditions essentielles pour l'amélioration de l'efficacité énergétique par la reconfiguration dynamique partielle dans les applications pratiques.

4.2.3 Conclusion

Les résultats montrent une amélioration sensible des gains énergétiques par la reconfiguration dynamique partielle, par rapport à une exécution uniquement logicielle (44,1%) mais aussi par rapport à une solution accélérée sans reconfiguration dynamique (49,6%). Cette étude montre globalement qu'il existe un potentiel prometteur dans l'exploitation de la RDP pour améliorer l'efficacité énergétique, sous réserve que l'ordonnanceur puisse se baser sur des décisions fiables pour identifier clairement les meilleurs choix d'allocation des tâches. Il est par ailleurs vraisemblable que le potentiel d'accélération permette d'améliorer encore les gains en développant des accélérateurs supplémentaires, sans augmenter le coût en ressources reconfigurables. Des améliorations intéressantes sont possibles concernant l'extension de l'ordonnanceur EAHS à l'exploitation du DVFS, incontournable si l'on considère des processeurs multicœur actuels, mais aussi à l'utilisation de la possibilité de *blanking* pour réduire l'effet très important de la consommation *idle* mise en évidence dans les résultats.

Ainsi, plusieurs conditions essentielles peuvent être énoncées afin d'établir une exploitation efficace de la RDP. L'utilisation d'un contrôleur de reconfiguration performant est la première exigence. Un partitionnement minimisant le nombre et la surface des zones reconfigurables joue un rôle essentiel qui doit être soigneusement défini. Ensuite vient la définition d'un ordonnanceur capable de recouvrir un maximum de contributions potentielles: exécution matérielle privilégiée, minimisation du nombre de reconfigurations, exploitation du parallélisme des régions reconfigurables, utilisation du *blanking* pour neutraliser les régions inactives, DVFS des unités logicielles (et matérielles éventuellement). La définition d'un tel ordonnancement représente donc un effort significatif qui se poursuit actuellement notamment pour la définition d'une technique qui supporte un véritable DVFS par cœur au niveau des unités logicielles.

4.3 Autres stratégies énergétiques

Les deux études précédentes sur des stratégies de gestion énergétique mieux spécialisées pour une application ou une classe d'application (e.g. vidéo, temps-réel) ont montré qu'elles ouvraient des possibilités prometteuses pour contrôler la consommation au plus près. Cette approche, dont le principe rejoint celui de la spécialisation pour améliorer l'efficacité de certains traitements (MMX, DSP, etc.), consiste à définir des stratégies qui adaptent plus finement les possibilités de traitement dynamique (e.g. DVFS) aux caractéristiques applicatives. Une stratégie similaire mais adaptée pour les systèmes distribués a été étudiée au cours du projet GEODES (ITEA2) dans une application concrète avec le développement d'un dispositif de surveillance vidéo par les Réseaux de Capteur Sans Fils (RCSF). Une deuxième stratégie exploitant les récupérateurs d'énergie développée dans le cadre du projet BENEFIC (Eureka CATRENE) est également présentée. Il est à noter que les stratégies présentées dans cette section ont été étudiées pour la réalisation de démonstrateurs physiques, et sont donc parfaitement utilisables dans des applications réelles.

4.3.1 Stratégie WSN

Cette étude porte sur la définition d'une stratégie de gestion énergétique pour les réseaux de capteurs sans fil (*Wireless Sensor Networks*, WSN). Elle a été développée dans le cadre du projet GEODES en étroite collaboration avec l'équipe CAIRN de l'IRISA pour aboutir à un démonstrateur développé principalement par la contribution de M. Laurent Rodriguez en tant qu'ingénieur d'étude sur le projet. Un des objectifs a été d'étudier les possibilités de surveillance environnementale ultra faible consommation par les réseaux de capteurs sans fil. La solution définie expérimente ainsi l'utilisation du standard IEEE 802.15.4 pour réaliser des communications sans fil à très faible consommation et l'exploitation de stratégies énergétiques DVFS spécialisées dans ce contexte.

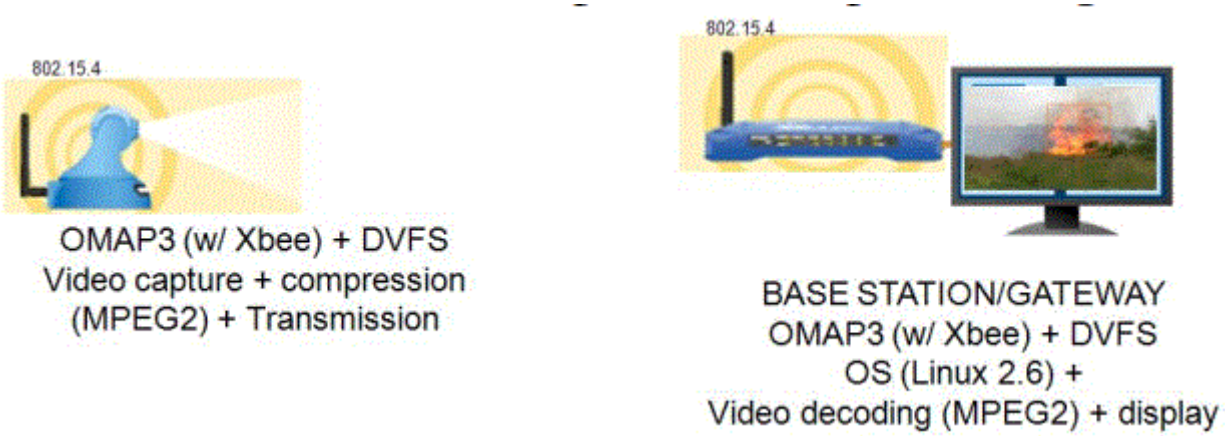


Figure 34 – Démonstrateur de surveillance vidéo WSN (GEODES)

Le scénario d'utilisation consiste à déployer des capteurs vidéo pour la surveillance d'un environnement et la détection de feu. On suppose donc que plusieurs capteurs sont disposés dans la zone d'intérêt (bâtiment, parc forestier, etc.) et que l'information est ensuite transportée jusqu'à un utilisateur final (e.g. centre de commande) qui peut éventuellement visualiser et analyser ce qui se passe. La Figure 34 donne un aperçu du système qui se compose de nœuds caméra (Texas Instruments OMAP3530) qui peuvent transmettre des

séquences vidéo observées vers une station de base pour affichage et/ou traitement de l'information. La transmission 802.15.4 se fait par un module *XBee* à très bas débit (quelques KB/sec) et nécessite une compression des flux vidéo (MPEG2) réalisée sur le processeur OMAP.

Pour définir une stratégie DVFS pertinente, nous avons d'abord procédé à l'exécution des nœuds caméra / station de base dans différentes configurations. Les phases de compression (caméra) et décompression MPEG (station de base) ont été mesurées sur le processeur OMAP pour chacun de ses points de fonctionnement (couples tension / fréquence défini par Texas Instruments). Les résultats correspondants (Tableau 24) montrent que le débit maximum possible limite la capacité de traitement globale à 5,61 img/sec du côté de l'encodeur.

OMAP 3530 (beagleboard)						
Freq. (MHz)	125	250	500	550	600	720
Volt. (V)	0.991	1.072	1.17	1.235	1.3	–
Encoder (fps)	0.95	1.92	3.80	4.16	4.76	5.61
Decoder (fps)	23	43	92	101	111	130

Tableau 24 – Performances encodeur / décodeur MPEG2 à différentes fréquences du processeur OMAP 3530

Cela implique que le décodeur pourra toujours opérer à la fréquence minimum de 125 MHz, de façon à garder une consommation minimale (le processeur fonctionne à 500MHz par défaut). De même, il est probable que les faibles débits propres au standard 802.15.4 limitent la quantité de données traitées (donc la fréquence de fonctionnement des nœuds), notamment du côté de la caméra et de la compression MPEG2 associée. Des mesures précises faites à ce niveau montrent un débit effectif maximum de 4KB/sec pour une taille d'image compressée de 2KB en moyenne ce qui correspond à un traitement de 2,3 img/sec maximum (sur des images au format QCIF). Comme pour le décodeur, on peut éviter d'exécuter l'encodeur à plus de 500 MHz puisque les modules *XBee* ne pourraient pas transmettre toutes les données avec le débit disponible.

Les considérations précédentes sur l'encodeur et les vitesses de transmission permettent ainsi d'élaborer une stratégie intéressante. Etant donné que les débits réels varient et dépendent de plusieurs facteurs en pratique (débit réel disponible, état du réseau, du trafic, interférences, etc.), il est possible d'adapter la fréquence de fonctionnement du processeur à la vitesse de transmission réellement disponible. Ce principe est illustré à la Figure 35 : si le débit est inférieur à 2KB/sec le processeur fonctionne à 125MHz, si le débit est compris entre 2KB/sec et 4KB/sec, le processeur fonctionne à 250MHz et si le débit est supérieur à 4KB/sec le processeur fonctionne à 500MHz.

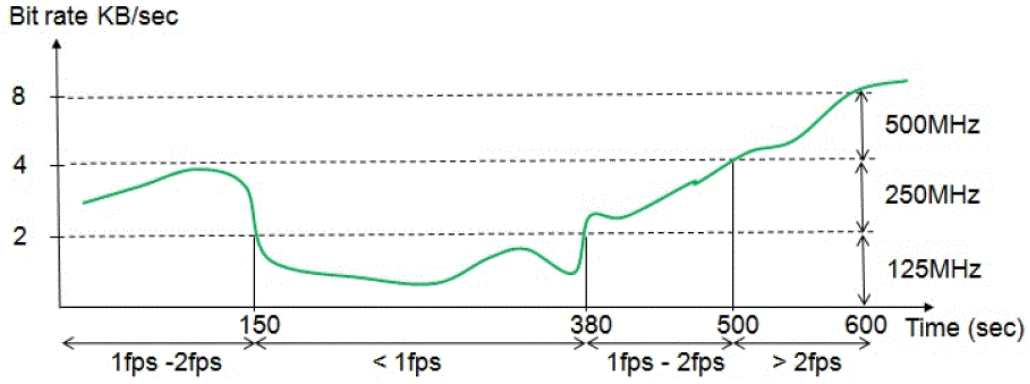


Figure 35 – Principe de la stratégie WSN vidéo

De cette façon, la puissance de calcul est adaptée finement et précisément à la quantité de données qui peut être transmise réellement sur le réseau. Le potentiel de réduction de consommation est alors très intéressant si on considère que sans cette stratégie, il faudrait faire fonctionner le nœud encodeur à 500 MHz pour couvrir tous les cas de figure. Le Tableau 25 donne la consommation mesurée sur l'OMAP 3530 à pleine charge pour chaque point de fonctionnement. On peut déduire une estimation des gains énergétiques et de l'extension de la durée de vie du nœud à partir de ces données.

OMAP 3530 (beagleboard)						
Freq. (MHz)	125	250	500	550	600	720
P (mW)	57	130	303	371	445	–

Tableau 25 – Consommation OMAP3530 aux différentes fréquences

Si on applique ces mesures à l'exemple de la Figure 35, il est possible de calculer une estimation de l'énergie consommée avec stratégie (150*0.13+130*0.057+120*0.13+100*0.303 = 72.81J) et sans stratégie (600*0.303 = 181.8J). Le gain associé atteint un facteur 2.5. Ces résultats représentent une amélioration significative, ce qui témoigne encore du potentiel de stratégies spécialisées sur les caractéristiques applicatives.

4.3.2 Stratégie ENO

Le travail suivant porte sur le développement d'un autre démonstrateur qui s'intègre dans un deuxième aspect auquel nous avons contribué dans le projet CATRENE BENEFIC, qui se base sur le concept de neutralité énergétique (*Energy Neutral Operation*, ENO). Ce travail s'est déroulé à travers le stage master de M. Khalil HARRANE (co-encadré avec mon collègue M. Alain PEGATOQUET) et une collaboration étroite entre STMicroelectronics qui a fournit la plateforme matérielle (STM32 Nucleo), Thales Communications & Security qui a développé des librairies de pilotes périphériques optimisées (*Hardware Dependent Software*, HDS) et le LEAT qui a travaillé sur la politique énergétique du système.

Le principe général consiste à récupérer de l'énergie à partir de l'environnement ambiant (lumière, chaleur, vibration, …) pour recharger les batteries et ainsi éviter de les remplacer. La principale difficulté est que la disponibilité de l'énergie ambiante varie, par exemple pour un panneau solaire suivant le contexte (conditions météorologiques, jour, nuit). Ces problèmes peuvent être surmontés en appliquant une (ou plusieurs) stratégie(s) de gestion d'énergie pertinente(s). Il est en effet possible d'adapter la consommation énergétique à la quantité d'énergie disponible et de ce fait, de satisfaire une condition de neutralité énergétique (ENO). En d'autres termes, l'énergie consommée est égale à l'énergie récupérée sur une certaine période de temps. Deux stratégies basées sur [21] et exploitant ce principe seront utilisées et abordées plus amplement par la suite.

Le démonstrateur se compose de quatre modules principaux: une source pour la récupération d'énergie (panneau solaire en premier lieu, et *rectenna* par la suite), le Power Manager matériel, la carte ST sur laquelle s'exécute la stratégie ENO et un module de communication (BLE, LORA). Le bloc diagramme de la Figure 36 présente les différents modules ainsi que leur interconnexion. Le panneau solaire utilisé délivre une puissance de 1,2W et un courant de 150mA. Son rôle est de récupérer l'énergie pour alimenter les éléments du démonstrateur. Le Power Manager matériel est composé essentiellement de trois parties: un

régulateur *Buck*, un régulateur *Buck-Boost* et un circuit de commutation qui sélectionne le régulateur à utiliser. Il dispose aussi d'une pile CR2032 et d'une super capacité de 0,09F. Le Power Manager matériel a pour fonctions de stabiliser la tension reçue du panneau solaire à l'aide du régulateur *Buck*, de charger la super capacité (en sortie du *Buck-Boost*) et de commuter entre celle-ci, la pile et le régulateur *Buck* (connecté au panneau solaire) suivant la disponibilité de l'énergie.

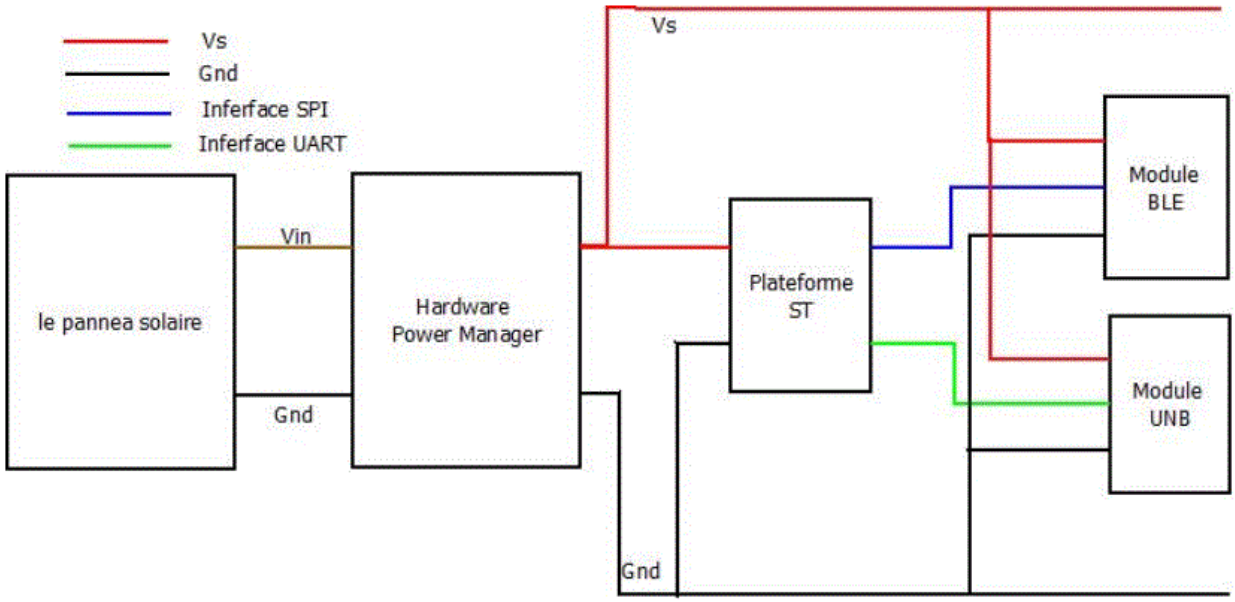


Figure 36 – Démonstrateur ENO développé au LEAT (BENEFIC)

La carte STM32L0 est dotée d'un microcontrôleur ARM Cortex M0+ qui peut fonctionner de 32KHz à 32MHz, ainsi que de nombreux périphériques tel que des convertisseurs ADC et DAC, GPIO, interfaces de communication USART, SPI, I2C et de nombreux autres dispositifs. Elle est aussi dotée de 64KB de mémoire Flash. Cette plateforme est très intéressante du point énergétique car elle ne consomme que 0,8mA en mode veille et peut être alimentée sous 1,65V.

La plateforme STM32L4 est équipée elle d'un microcontrôleur ARM Cortex M4 beaucoup plus puissant qui peut fonctionner de 32KHz à 80MHz et intègre une Unité de Calcul Flottant (FPU). Ce microcontrôleur embarque les mêmes types de périphériques, mais en plus grand nombre et/ou plus performants. Par exemple, elle comporte trois convertisseurs ADC avec une fréquence d'échantillonnage de 5MHz contre un seul convertisseur à 1,14MHz pour la carte L0. De plus la carte L4 dispose d'autres périphériques tels qu'un module de chiffrement AES matériel (*Advanced Encryption Standard*), un générateur de nombres aléatoires, et comporte 1 MB de mémoire Flash. Bien qu'elle consomme davantage en mode stop (1,4mA), la carte L4 reste plus intéressante que la L0 par son rapport consommation/fréquence qui est en moyenne très proche à basse fréquence. Les besoins mémoire et applicatifs en termes de sécurité ont finalement conduit Thales à retenir la plateforme L4 pour le démonstrateur final.

Afin de permettre au nœud de communiquer avec la station de base, il est muni d'un module *Bluetooth Low Energy* (BLE) et d'un module *Ultra Narrow Band* (UNB). Le module Bluetooth utilisé est un module BLE de STMicroelectronics faible consommation qui consomme 7,3mA en réception et 8,2mA en émission (à 0 dBm). Ce module constitue l'élément de communication principal du démonstrateur étudié dans ce travail, mais évoluera ensuite vers l'utilisation d'un module UNB LORA dans le démonstrateur final du projet finalisé par Thales. Par ailleurs, il intègrera principalement un panneau solaire pour la récupération d'énergie mais aussi un prototype d'antenne redresseuse ou *rectenna* (développée par l'IMEC) par la suite.

La politique de neutralité énergétique utilisée est issue de [21] et doit permettre une autonomie indéfinie (dans l'idéal) et fonctionner pour plusieurs sources d'énergie (solaire, radiofréquence ou *rectenna*). Deux stratégies ENO ont finalement été déployées sur le démonstrateur pour être plus efficaces selon le type de récupérateur actif: une stratégie polyvalente et une stratégie périodique. La première comme son nom l'indique est une stratégie généraliste qui s'applique pour différents types de récupérateur. La deuxième est une approche optimisée pour les sources d'énergie périodiques (e.g. solaire avec cycles jour / nuit sur 24h).

La stratégie non périodique se base sur l'utilisation d'un superviseur (*power monitor*) qui évalue l'énergie consommée. Cette estimation se base sur une caractérisation préliminaire de la consommation des éléments de la plateforme réalisée hors ligne, et sur l'historique de l'énergie récupérée.

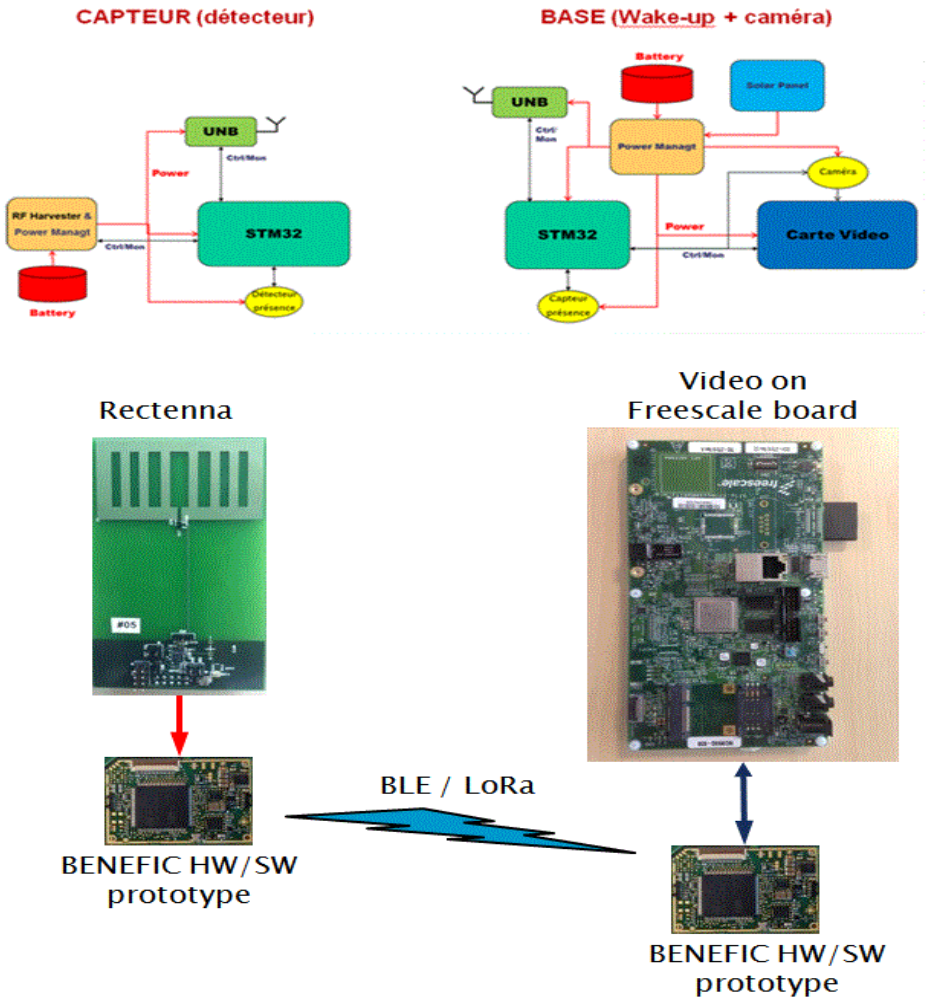


Figure 37 – Démonstrateur ENO finalisé par Thales Communications & Security (BENEFIC)

Le superviseur prédit l'énergie qui sera récupérée pendant l'intervalle d'activité suivant en s'appuyant également sur un filtre adaptatif pour améliorer les résultats. Grâce à cette estimation et en connaissant la tension de la super capacité Vs, on peut évaluer l'énergie disponible pour les activités futures. En appliquant la condition de neutralité énergétique (Energie consommée = Energie récupérée) il est ainsi possible de déterminer la prochaine période de réveil.

La stratégie ENO pour une source d'énergie périodique est définie pour mieux équilibrer la qualité de service du système sur la durée d'un cycle (par exemple sur une période de 24h dans le cas d'un panneau solaire). Pour être plus efficace, cette stratégie fonctionne selon deux modes : un mode *positif* qui est activé en présence d'énergie et un mode *négatif* qui fonctionne en absence d'énergie. Le mode positif est similaire à la stratégie non périodique décrite précédemment. Elle se distingue au niveau de la prédiction de l'énergie récupérée qui se base sur un filtre adaptatif et la connaissance du rapport entre les périodes d'absence et de présence d'énergie pour calculer les périodes de réveil et assurer une alimentation électrique permanente du système. Le mode négatif est activé durant les périodes d'absence d'énergie. Il estime la durée d'absence d'énergie restant pour répartir correctement l'énergie encore disponible dans la super capacité. Il en découle le nombre de réveils possibles ainsi que la prochaine période de réveil.

Les deux stratégies ont été intégrées dans le démonstrateur préliminaire du LEAT (Figure 36) et celui finalisé par Thales (Figure 37) pour la revue finale du projet BENEFIC. Les résultats ont démontré que la stratégie non périodique opère correctement pour le récupérateur solaire (mais en régime diurne uniquement) et pour la *rectenna* avec une période de réveil minimale. Cette stratégie devient vite inutilisable dans le cas du solaire en régime nocturne à cause de la variation importante des caractéristiques lumineuses entre le cycle de jour et le cycle de nuit. En effet, cet algorithme suppose que l'énergie récupérée ne varie pas entre deux slots temporels consécutifs. C'est précisément pour mieux répondre à ces particularités du solaire que la stratégie périodique a été définie. Celle-ci a été développée et testée sur le démonstrateur intégrant le panneau solaire avec de très bons résultats et une bonne répartition de la qualité de service entre les périodes de présence et d'absence d'énergie.

4.4 Conclusion

Ce chapitre montre ainsi à travers des applications concrètes i) le rôle essentiel et complexe joué par les dispositifs de gestion énergétique sur la consommation CPU effective (DVFS, modes repos, récupération d'énergie, etc.) et ii) qu'il est possible de beaucoup mieux les exploiter par des stratégies grain fin et spécialisées.

Les approches existantes sont effet majoritairement des stratégies polyvalentes qui se basent sur l'observation de la charge CPU. Pour faire simple, elles consistent à augmenter la fréquence (plus ou moins rapidement) lorsque la charge du processeur augmente. Cette approche présente l'avantage d'être applicable quelque soit l'application, mais la contrepartie est qu'elle est souvent inefficace. Par exemple, si un utilisateur souhaite visualiser de la vidéo, l'OS va rapidement faire passer le CPU à sa fréquence maximum et conduire à une consommation maximum pendant toute la durée d'exécution. Or des stratégies plus judicieuses et complexes comme celles exposées section 4.1.1 (stratégie DVFS vidéo), 4.1.2 et 4.1.3 (stratégies DVFS et DPS temps-réel) permettent elles d'améliorer l'efficacité énergétique jusqu'à plus de 50%, comme il l'a été montré sur du traitement vidéo. Enfin l'extension de ce concept à un ordonnanceur eco-énergétique pour des architectures hétérogènes intégrant de la reconfiguration dynamique partielle laisse entrevoir un potentiel d'amélioration encore plus important qui sera largement approfondi et développé au chapitre suivant.

En fin de compte, les approches génériques basées *workload* conviennent bien pour une gestion simple (gros grain) de la consommation CPU, c'est à dire à une échelle temporelle de l'ordre de la minute et plus. L'utilisation de stratégies dédiées consiste elle adapter plus finement la capacité de traitement CPU aux traitements en cours (à une échelle temporelle de l'ordre de la fraction de seconde). Elles sont ainsi potentiellement beaucoup plus efficaces mais ne sont utilisables que pour certaines applications puisqu'elles exploitent leurs spécificités. La définition de telles stratégies dédiées pose également des difficultés importantes pour leur utilisation en pratique car les infrastructures de gestion énergétique sont complexes et très souvent noyées dans les méandres d'un code noyau au sein des OS. C'est pour résoudre ce problème que nous avons développé une technique visant à faciliter l'implantation d'ordonnanceurs dédiés en espace utilisateur Linux. Ce procédé fait l'objet d'un dépôt de brevet international depuis 2014 qui suit son cours actuellement [10], et permet la définition d'ordonnanceurs dédiés sur n'importe quelle plateforme supportant Linux et son infrastructure ACPI *CPUFreq*. On assiste d'ailleurs depuis très récemment à l'extension des solutions de gestion énergétique existantes dans le sens d'ordonnanceurs spécialisés. C'est le cas par exemple de Linux avec l'introduction d'une nouvelle stratégie (ou gouverneur) dénommée *Schedutil* depuis la version 4.7 du noyau, qui se base directement sur des données issues de l'ordonnanceur. Il est néanmoins encore trop tôt pour donner une évaluation de cette technique qui est toujours en phase de développement et d'amélioration.

Les deux chapitres précédents font ainsi ressortir deux facteurs prépondérants en termes d'efficacité énergétique, en particulier pour des topologies hétérogènes: des aspects liés à l'exploration architecturale et à un déploiement applicatif méthodique, et le rôle déterminant des stratégies de gestion énergétique. Le chapitre suivant étudie en détail dans quelle mesure il est possible de combiner plus efficacement ces deux éléments de façon à permettre une amélioration très significative de l'efficacité énergétique.

5 Eléments d'amélioration de l'efficacité énergétique

Si on considère la généralisation d'architectures multiprocesseur (multi-cœur, many-cœur) et les tendances et perspectives d'hétérogénéité, un modèle de plateforme qui peut être raisonnablement envisagé dans les perspectives à venir est celui le la Figure 38. Ce modèle fait l'hypothèse d'une hétérogénéité assez maîtrisée, on peut envisager que les architectures multi-cœur et many-cœur hétérogènes conserveront un certain degré d'homogénéité pour ne pas trop complexifier l'exploitation de l'architecture (e.g. technologie *big.LITTLE* chez ARM constituée d'un cluster de quatre CortexA7 et d'un cluster de quatre CortexA15). En effet, si l'hétérogénéité permet en théorie une meilleure efficacité de traitement en affectant des tâches ou des applications différentes aux ressources d'exécution les plus adaptées, c'est à la condition d'une gestion en ligne et d'une prise de décision efficace. Or plus l'hétérogénéité est importante, plus le déploiement devient complexe et critique, car dans le cas d'une charge CPU mal répartie (cf. chapitre 3.1), le risque est d'augmenter la consommation au lieu de la réduire. Il est donc souhaitable de garder au moins une partie de la plateforme constituée de cœurs homogènes (*clusters*) pour faciliter l'exploitation et la répartition de la charge CPU. Par ailleurs, il est plus raisonnable de limiter la complexité qui augmente de façon importante avec le nombre d'unités d'exécution (différents types de cœurs, gestion DVFS et modes repos, accélération matérielle, reconfiguration dynamique, etc.). La réalisation d'une meilleure efficacité énergétique peut s'obtenir dans ces conditions par une combinaison efficace des facteurs influents sur la consommation.

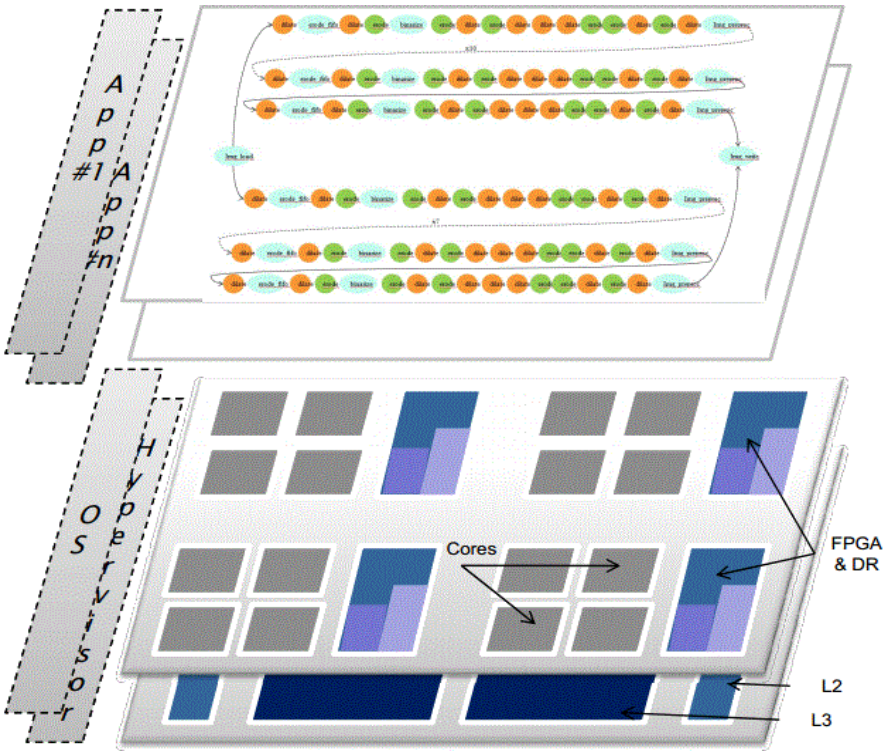


Figure 38 – Vue d'un déploiement multi applicatif sur un SoC hétérogène 3D.

A la lumière des études présentées, l'exploitation de l'hétérogénéité repose sur trois éléments importants : une analyse méthodique du déploiement (exploration) afin d'identifier les gains tangibles et les conditions architecturales et applicatives associées, la spécialisation de stratégies exécutives pour permettre des gains énergétiques plus élevés, et enfin une méthodologie de développement intégrée (incluant logiciel,

matériel, modèles et stratégies énergétiques spécialisées) pour mieux coupler l'exploration et l'exécutif, et garantir la faisabilité et les caractéristiques du déploiement attendu. Ces aspects sont encore principalement traités de façon séparée jusqu'à maintenant. Les bénéfices d'une coopération plus organisée sont étudiés et évalués dans les sections suivantes.

5.1 Analyse du déploiement hétérogène

5.1.1 Modélisation système

Une recherche plus méthodique de l'efficacité énergétique passe d'abord par une modélisation système pertinente qui permette un support aux prises de décision par des estimations rapides et suffisamment fiables. La définition d'un tel compromis doit être globale et intégrer tous les aspects importants qui déterminent la consommation et les performances en pratique. Le développement d'un modèle pragmatique issu d'une large expérimentation et la caractérisation en consommation associée a été étudié au chapitre 3 en couvrant les plateformes multiprocesseur symétriques jusqu'à la prise en compte de l'accélération matérielle et de la reconfiguration dynamique partielle. D'autres modèles existent mais ne couvrent généralement que partiellement certains aspects du système (e.g. performance ou consommation, modélisation logicielle ou matérielle) ou le niveau d'abstraction. La modélisation proposée représente une base effective comme l'ont démontré les études applicatives, mais néanmoins perfectible (par exemple dans le cas d'une hiérarchie mémoire avancée, la prise en compte d'autres types d'unités d'exécution ou de technologies d'intégration 3D, etc.). Elle offre néanmoins un formalisme de base réaliste, global (performance et consommation) et unifié (logiciel et matériel) permettant des estimations fiables et rapides au niveau système. La caractérisation complète en performance, puissance et énergie des unités d'exécution (logicielles et matérielles) tient compte d'un ensemble de paramètres énergétiques de base tels que les points de fonctionnement et les modes repos des cœurs, le partitionnement du FPGA, ou encore le parallélisme des accélérateurs.

5.1.2 Programmation parallèle

Le déploiement efficace de tâches applicatives sur des unités hétérogènes dépend étroitement de deux éléments : une répartition équilibrée de la charge et l'utilisation de stratégies dédiées.

L'étude de la modélisation et de la conception au niveau système du chapitre 3 fourni des résultats utiles concernant les spécifications applicatives en particulier du point de vue d'une parallélisation efficace qui favorise l'efficacité des traitements. Un premier point concerne l'importance d'une répartition équilibrée de la charge pour un système multi-cœur. Les résultats du chapitre 3.1 sur une plateforme SMP montrent une moins bonne efficacité énergétique dans les situations où les tâches applicatives, très homogènes, ne sont pas réparties équitablement sur les cœurs (e.g. quatre tâches sur trois cœurs) ou lorsqu'une charge inégale (e.g. décodeur H.264 parallélisé par fonctions) est répartie sur les cœurs. Dans l'hypothèse précédente d'une architecture ayant une faible hétérogénéité des cœurs, une décomposition homogène facilitera la répartition des tâches sur des grappes de cœurs homogènes. Par ailleurs, une décomposition homogène permet une modélisation énergétique plus simple qui facilite la prédictibilité et les prises de décision. Une parallélisation régulière qui facilite la répartition de la charge et la prédictibilité des modèles de déploiement sont des éléments essentiels qui contribuent à l'amélioration de l'efficacité énergétique.

Un deuxième point concerne l'hétérogénéité qui n'est traitée ici que du point de vue des accélérateurs matériels. La maturité des outils et de la technologie facilite aujourd'hui l'utilisation d'accélérateurs

matériels (FPGA, GPU). Les recherches exposées plus particulièrement ici abordent le cas des accélérateurs matériels reconfigurables et de la reconfiguration dynamique partielle. Les résultats du chapitre 3.3 montrent en effet un potentiel d'amélioration énergétique significatif par l'utilisation d'accélérateurs reconfigurables qui peut être encore amélioré par la reconfiguration dynamique. L'utilisation de ce potentiel implique une décomposition fonctionnelle pertinente de l'application qui fasse apparaître des traitements propices à une accélération matérielle. Il est à noter que le cas d'une hétérogénéité logicielle (utilisation de cœurs différents) n'a pas encore été pris en compte dans nos applications, mais pourra l'être car elle a été prévue dans les modèles.

Un autre aspect lié au logiciel concerne l'utilisation de stratégies exécutives dédiées (ordonnancement, gestion énergétique) pour affiner en cours d'exécution l'adéquation des ressources au traitement en exploitant la variabilité des applications. Cette possibilité est abordée en détail à la section suivante.

5.2 Spécialisation de stratégies exécutives

5.2.1 Analyse des conditions d'efficacité DVFS et DPS

La majorité des stratégies énergétiques DVFS se font sur la base d'une réduction de la fréquence processeur pour économiser de l'énergie. Bien que cette hypothèse fondamentale se vérifie dans la plupart des cas, il a été démontré section 4.1.1 que ce n'est pas toujours le cas à cause des points de fonctionnement. En effet leurs caractéristiques déterminent la capacité à réduire l'énergie en pratique et par extension, les niveaux de gains potentiels. Un facteur déterminant dans l'efficacité du DVFS est la différence relative des niveaux de puissance en charge pour les différentes fréquences possibles. D'importantes différences sont une condition essentielle pour des gains sensibles, alors que de petites différences réduisent les gains potentiels, allant même jusqu'à un DVFS inefficace dans certains cas. Les stratégies DPS sont sujettes à des conditions de plateformes similaires. Un critère d'efficacité proéminent dans ce cas est le niveau de puissance à vide des modes inactifs (consommation *idle*). Les techniques DPS sont plus efficaces pour des systèmes ayant de fortes consommations à vide, qui sont typiques de technologies d'intégration récentes (moins de la vingtaine de nanomètres) et présentent une forte proportion de consommation statique.

L'applicabilité des stratégies est par ailleurs très fortement conditionnée par les coûts des transitions entre les états CPU. L'efficacité des politiques DVFS se base implicitement sur le fait que les latences de changement de fréquence, typiquement autour de quelques centaines de microsecondes, peuvent être négligées pour une application donnée. Cela ne sera pas le cas pour des applications vidéo par exemple si une plateforme présente des latences de quelques millisecondes, étant donné les contraintes de traitement typiques de quelques dizaines de millisecondes par image. Par ailleurs les latences de changement d'état DPS sont souvent plus importantes, à cause de la complexité des processus de mise en veille et de réveil, ce qui par conséquent restreint l'applicabilité des techniques DPS par rapport au DVFS, pour des applications avec des temps de tâche faibles. Les latences de transition d'état doivent donc être soigneusement analysées par rapport aux contraintes temporelles applicatives avant tout développement d'une stratégie DVFS ou DPS.

Les politiques d'OS existantes, qui sont des stratégies généralistes basées *workload* (charge CPU) pour la plupart (e.g. Linux *OnDemand* ou *Conservative*), conduisent typiquement à placer la fréquence CPU à son maximum pour des traitements intensifs, mais de façon indifférenciée quel que soit l'application. Des

stratégies avancées, basées DVFS ou DPS, exploitent une connaissance plus spécifique des applications pour réaliser une adaptation plus fine, et peuvent ainsi être beaucoup plus efficaces. Les résultats des trois stratégies expérimentées, adaptées a du traitement vidéo, montrent un potentiel de réduction qui peut aller jusqu'à 50%, là où une stratégie généraliste augmenterait la fréquence et donc, la consommation. L'efficacité de ces stratégies basées application tient à la possibilité d'exploiter une certaine forme de variabilité à l'exécution des tâches. Elles sont basées sur un paramètre applicatif (e.g. vitesse de décodage, *slack* dynamique) dont les variations au cours de l'exécution déterminent en pratique les niveaux de gains énergétiques. Ainsi, le montant des réductions énergétiques que l'on peut raisonnablement espérer de l'emploi efficace de stratégies DVFS et DPS se situe entre 0% et un maximum de 50%, en fonction de l'exécution effective qui dépend des données traitées. Cette efficacité est aussi sujette aux conditions énoncées précédemment concernant la granularité temporelle des tâches qui doit permettre de négliger les latences de transitions d'états CPU.

Une question qui se pose naturellement est de savoir laquelle d'une politique DVFS ou DPS peut être la plus efficace. Il est bien sûr impossible de répondre de manière catégorique et généralisable mais la comparaison des stratégies DSF et AsDPM sur l'exemple commun d'encodeur H.264 (chapitre 4.1.2 et 4.1.3) fourni les éléments d'une réflexion de base intéressante. Les gains énergétiques des deux stratégies ont été présenté Figure 27 et Figure 30 respectivement pour DSF et AsDPM, sur ARM1176 et Cortex A9. Ils montrent qu'AsDPM obtient de meilleures réductions énergétiques que DSF dans tous les cas. Néanmoins, ils montrent aussi des différences en efficacité qui varient beaucoup d'une plateforme à l'autre. Comme il a été mentionné précédemment, un facteur important est la différence entre les niveaux de puissance en charge des points de fonctionnement pour une stratégie DVFS, et les niveaux de consommation à vide et en charge pour une stratégie DPS. Bien qu'il soit concevable de considérer que la désactivation d'un cœur inutilisé (DPS) soit plus efficace qu'un ralentissement (DVFS), toute extrapolation doit être considérée avec prudence. En effet, les caractéristiques matérielles d'une plateforme (points de fonctionnement, niveaux de puissance et latences de transitions d'états CPU) peuvent inverser la tendance, voire ne pas permettre l'application d'une stratégie dans certains cas. DPS devrait être plus efficace mais plus limité en termes d'applicabilité, par exemple pour être compatible avec les contraintes temporelles des tâches applicatives.

5.2.2 Analyse des conditions d'efficacité de la RDP

Concernant la RDP, une part importante des gains énergétiques proviennent en premier lieu de l'exécution matérielle, potentiellement beaucoup plus efficace qu'une exécution logicielle. Par ailleurs, la RDP permet aussi de réduire la taille de la logique programmable en utilisant la même surface pour l'exécution de tâches séquentielles différentes. Des gains supplémentaires sont alors possibles grâce à la réduction de surface statique active occupée par la logique programmable. Ces gains dépendent étroitement de la surface réduite et de la consommation *idle* associée (fonction de la technologie d'intégration). Le partitionnement du FPGA en régions optimales (compromis taille / parallélisme) ainsi que leur utilisation (ordonnancement) sont donc un point déterminant. L'utilisation de techniques de *blanking* est quasiment indispensable pour réduire au maximum la consommation *idle* des régions non actives et garantir de meilleurs gains.

Les résultats sur le décodeur H.264 montrent des gains énergétiques variables et relativement modestes (16% sur Virtex6) de la RDP par rapport à une accélération matérielle statique, qui s'explique par un nombre d'accélérateurs relativement limité (trois fonctions) et donc un taux d'utilisation des régions faible.

Ainsi le potentiel en efficacité de la RDP augmente très rapidement avec la proportion de fonctions matérielles, comme il sera démontré au chapitre 5.3.

Pour une application constituée par un ensemble de tâches dont certaines sont matérielles, les cas de figure possibles mèneront à des caractéristiques très différentes en termes de performance, surface, énergie: (i) utiliser des tâches matérielles statiques, (ii) utiliser des tâches matérielles dynamique et reconfigurer quand nécessaire, (iii) utiliser des tâches uniquement logicielles et (iv) utiliser des tâches logicielles et des tâches matérielles statiques et/ou dynamiques avec différents compromis coût-performance (parallélisme matériel). Le problème se complexifie avec l'hétérogénéité logicielle (cœurs, DVFS). A cela s'ajoutent de nombreux paramètres tels que les performances du contrôleur de reconfiguration, l'exploitation du *blanking*, l'utilisation d'un ordonnanceur spécialisé (pour favoriser l'exécution matérielle, tirer avantage d'exécutions en parallèle, minimiser le nombre de reconfigurations), mais aussi la technologie silicium (consommation statique et *idle*). Comme l'a démontré l'exemple de l'encodeur x265 au chapitre 3.4.3, la pertinence de la spécification applicative elle-même à une accélération matérielle n'est pas toujours évidente. Les coûts doivent donc être analysés de manière méthodique afin de savoir s'il y a un intérêt ou non à utiliser la reconfiguration dynamique partielle, et en tenant compte éventuellement d'une possible gestion énergétique des cœurs (DVFS, modes repos). La définition d'une telle méthodologie qui soit capable de combiner au mieux les gains possibles à ces différents niveaux fait l'objet du chapitre suivant.

5.3 Conception intégrée

5.3.1 Extension de la méthodologie FoRTReSS

Dans la méthodologie FoRTReSS initiale (chapitre 3.4.3), l'objectif principal est de garantir un niveau de performance à l'application, qui peut être temps-réel. L'étude qui a été menée au cours du projet BENEFIC a permis l'extension du flot original, centré sur l'exploitation de la RDP pour des contraintes temps-réel, pour se focaliser sur l'efficacité énergétique. Les principes fondateurs de cette approche se basent sur la complexité inhérente de l'hétérogénéité impliquée par les exécutions logicielles et matérielles et par les aspects dynamiques des plateformes actuelles, qui s'intensifie si la RDP est concernée (partitionnement FPGA, cœurs logiciels, DVFS, *clock gating*, *power gating*). La méthodologie développée se base ainsi sur un meilleur couplage de deux éléments déterminants et étroitement dépendants lors de tels déploiements hétérogènes en général, mais encore plus lorsque la RDP est impliquée: l'exploration de l'espace de conception (et/ou l'analyse du déploiement applicatif) pour identifier les meilleurs paramètres architecturaux (e.g. combien de cœurs, types de cœurs, nombre, taille et forme des régions reconfigurables, …) et la définition de stratégies exécutives (ordonnancement, gestion énergétique) spécialisées ou dédiées pour améliorer encore plus finement le déploiement à l'exécution. En effet, la pertinence de l'analyse du *mapping* dépend étroitement de l'ordonnancement dont le rôle est essentiel dans l'exploitation des capacités de l'architecture. La définition de stratégies d'ordonnancement dédiées et avancées qui se basent sur une meilleure connaissance de l'application est également une possibilité incontournable pour espérer améliorer les gains énergétiques de façon très significative. De plus, exploration et ordonnancement nécessitent une aide à la prise de décision rapide et fiable pour i) permettre l'analyse d'un vaste espace de conception/déploiement et ii) d'évaluer rapidement les choix d'ordonnancement en cours d'exécution. Ces deux étapes essentielles peuvent bénéficier grandement des mêmes estimateurs pour améliorer la cohérence des décisions de conception/déploiement et d'ordonnancement.

5.3.2 Application à une caméra intelligente

L'étude de cas suivante illustre l'application de cette approche sur une application réelle dans le domaine des caméras intelligentes. Il s'agit d'un système de détection et de reconnaissance de plaque minéralogique (*License Plate Recognition*, LPR par la suite) destiné au contrôle automatique du trafic à l'entrée d'un péage développé par Thales Research & Technology. L'identification des plaques d'immatriculation est un enjeu important dans l'industrie du transport par ses applications à la gestion du trafic et aux systèmes de surveillance, de sécurité et de contrôle d'accès.

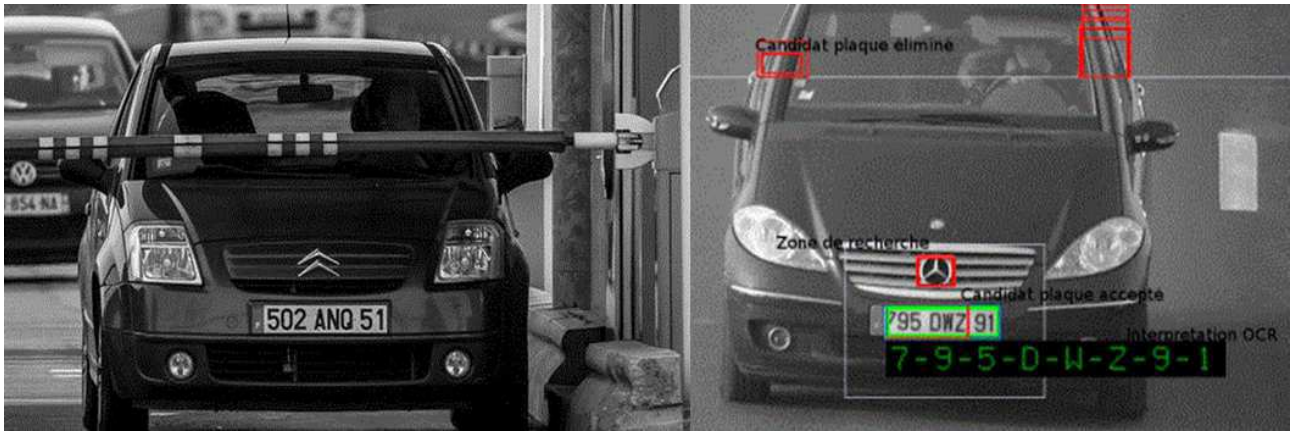


Figure 39 – Détection et reconnaissance de plaque simultanée sur deux voies (2-lane LPR).

Dans notre cas, l'application LPR est conçue pour la détection et l'identification automatique de véhicules franchissant un péage. Les caméras (une par voie de péage) sont disposées pour détecter et suivre les plaques d'immatriculation avant des véhicules de chaque voie. Le système peut ainsi analyser plusieurs voies simultanément, comme illustré à la Figure 39 pour deux voies. On considèrera par la suite que le système de reconnaissance LPR peut être configuré pour 1, 2, 4 ou 8 voies simultanément (*1-, 2-, 4- and 8-lane LPR*). La partie inférieure de l'image est utilisée pour suivre la plaque de la voiture courante tandis que la partie supérieure sert à détecter la plaque de la voiture suivante. Les deux régions sont soumises à une succession d'opérations morphologiques (érosion, dilatation, etc.) qui visent à améliorer la qualité de l'image pour les étapes de détection et de reconnaissance. Une séquence d'opérations (*dilate*, *erode_fifo*, *erode*, *binarize*, *preprocessing*) est appliquée sept fois pour la zone de détection et dix fois pour la zone de reconnaissance comme le montre le graphe de tâches de la Figure 40. Ce graphe traduit les opérations réalisées pour le traitement d'une voie, et peut être facilement dupliqué pour le traitement de voies supplémentaires.

Dans la région de détection ainsi améliorée, un processus de reconnaissance optique de caractères (*Optical Character recognition*, OCR) est utilisée pour identifier les caractères alpha-numériques à l'aide d'un réseau de neurones multi-couche (FANN [15]). En dehors de la phase d'apprentissage, l'analyse de l'exécution montre que la phase de reconnaissance du réseau de neurones est très négligeable par rapport au reste des traitements morphologiques. En particulier, trois fonctions représentent à elles seules 79% du temps d'exécution total : *erode_fifo* (36.68%), *dilate* (25.27%) et *erode* (16.71%). L'accélération maximum pour l'ensemble de l'application LPR peut donc atteindre théoriquement un facteur 4,7 qui s'ajoute aux gains énergétiques de l'exécution matérielle des fonctions *erode_fifo*, *dilate* et *erode* pour permettre une amélioration significative de l'efficacité énergétique. Celle-ci sera analysée en terme de réduction du produit énergie-performance (*Energy Delay Product*, EDP) par rapport à une exécution complètement logicielle (ou accélération × gain énergétique).

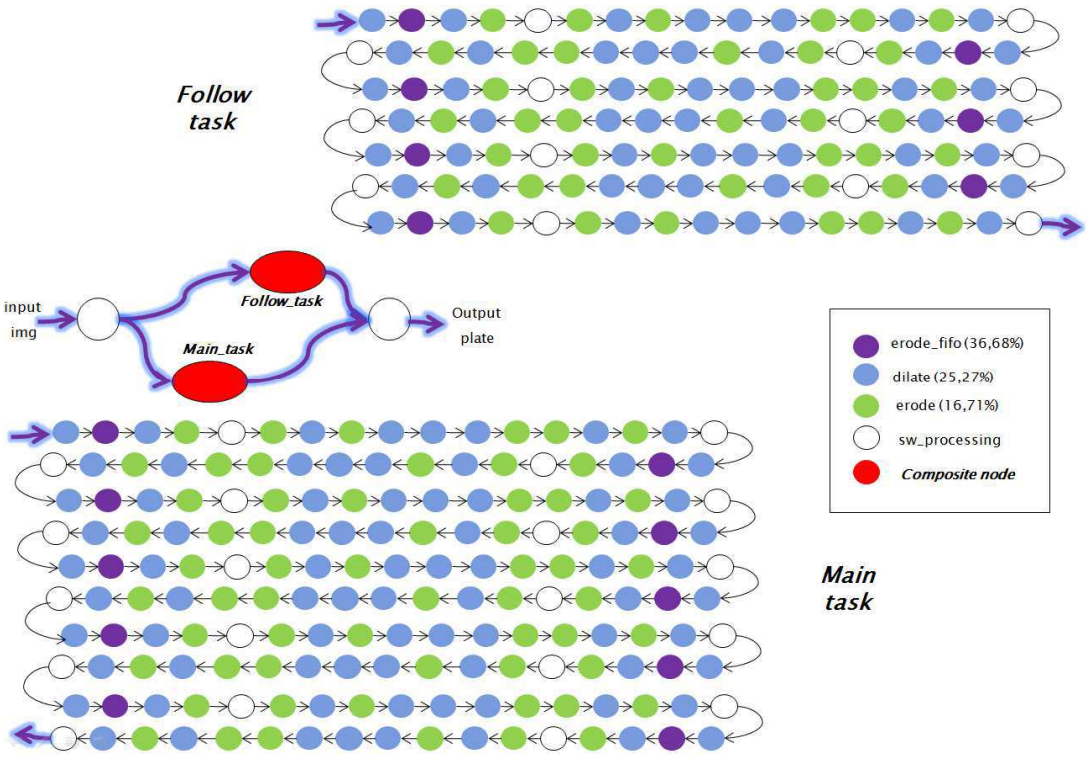


Figure 40 – Graphe de tâches de l'application (1-lane LPR)

On s'intéresse par la suite au déploiement de cette application sur différentes plateformes afin d'analyser en détail les performances, la consommation et l'efficacité énergétique dans plusieurs configurations (1-, 2-, 4- , 8-lane LPR). On considère d'abord un FPGA de la famille Virtex-6 de Xilinx avec des cœurs *softcore* MicroBlaze comme plateforme d'exécution. Les caractéristiques en performances et en consommation de toutes les tâches matérielles et logicielles sont issues de mesures sur une carte d'évaluation ML605 (en s'appuyant sur la méthodologie décrite en 3.2 pour la génération des accélérateurs). Chaque solution logicielle de référence (une pour chaque configuration LPR) correspond à l'exécution multiprocesseur la plus efficace, c'est-à-dire celle qui permet la meilleure performance pour le parallélisme d'une configuration donnée (2 cœurs pour 1-lane LPR, 4 cœurs pour 2-lane LPR, 8 cœurs pour 4-lane LPR, 16 cœurs pour 8-lane LPR).

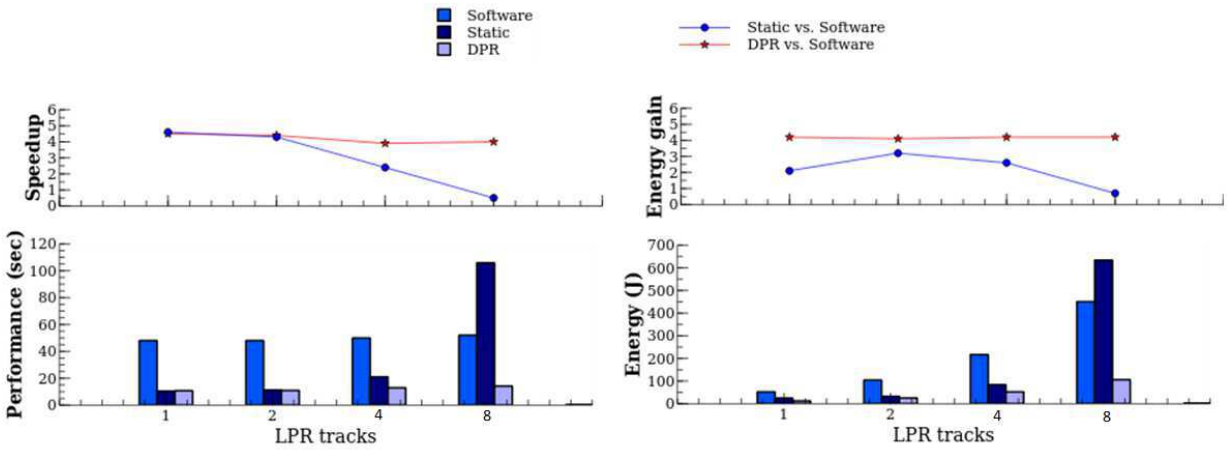


Figure 41 – Accélération et gain énergétique des solutions statiques et RDP par rapport à une exécution logicielle (XC6VLX240T)

Le bas de la Figure 41 montre que les performances des solutions logicielles restent constantes lorsque l'on passe l'application à l'échelle (1-lane LPR / 2 cœurs, 2-lane LPR / 4 cœurs, etc.). L'énergie des solutions logicielles double logiquement avec chaque configuration successive (qui nécessite deux fois plus de cœurs que la précédente), tandis que les solutions RDP sont toujours plus performantes avec moitié moins de cœurs. Pour 8-lane LPR par exemple, l'accélération RDP (8 cœurs + 6 RRs) permet des résultats nettement au-dessus de la meilleure solution logicielle (16 cœurs) avec des performances 4 fois supérieures pour 4,4 fois moins d'énergie consommée (haut de la Figure 41). Il est intéressant de voir que l'accélération et les gains énergétiques des solutions RDP par rapport aux solutions logicielles restent tous les deux très stables dans les quatre configurations LPR, et que l'efficacité énergétique (ici, le produit énergie-performance) a été améliorée jusqu'à un facteur 19 (4,5 × 4,2) dans la configuration 1-lane LPR.

Dans les différentes configurations définies, les solutions dites *statiques* combinent un nombre de cœurs minimum avec trois accélérateurs statiques (un pour *dilate*, un pour *erode_fifo*, un pour *erode*). Nous avons intentionnellement limité le nombre d'accélérateurs statiques à trois, quel que soit la configuration LPR, car le FPGA serait rapidement à cours de ressources autrement. Cela se traduit par des performances plus variables des solutions statiques qui sont globalement moins bonnes que pour la RDP. Plus précisément, l'écart se creuse à mesure que les configurations et le parallélisme augmentent. Pour 1-lane LPR par exemple, la RDP est plus lente de 2,6% par rapport à l'accélération statique, car elle est affectée par un peu moins de parallélisme (1 RR contre 3 accélérateurs) et des latences de reconfiguration dynamique. Néanmoins, la diminution des ressources logiques obtenue par la RDP permet de gagner 50,3% en énergie. Pour 8-lane LPR, la RDP bénéficie d'un parallélisme optimal (8 cœurs + 6 RRs) et peut atteindre une accélération de 8,1 avec une réduction énergétique d'un facteur 6 par rapport à une solution statique (8 cœurs + 3 accélérateurs). Une implémentation statique est inefficace dans ce cas, même par rapport à la solution logicielle de référence (16 cœurs), car elle est fortement désavantagée par la limitation à trois accélérateurs pour traiter huit processus LPR en parallèle. Au bout du compte, l'accélération statique améliore l'efficacité énergétique d'un facteur 13,9 (4,3 × 3,2) à 0,3┐ (0,5 × 0,7) par rapport à l'exécution logicielle, là où la RDP l'améliore d'un facteur 16,1 (3,9 × 4,2) à 19 (4,5 × 4,2). La même analyse a été réalisée sur deux plateformes Zynq XC7Z020 et XC7Z045 (85K et 350K cellules logiques respectivement) incluant des cœurs d'IP hard ARM Cortex-A9. Les résultats sont amplement détaillés dans [16] et montrent également des améliorations significatives de l'efficacité énergétique d'un facteur 9,3 à 11,7.

5.3.3 Analyse des conditions d'efficacité

Les principales conclusions de l'application précédente sont d'abord qu'une variété de conditions influence une exploitation correcte de l'hétérogénéité et le niveau des gains possibles par la RDP. L'accélération par la RDP n'est pas applicable à tous les types de traitement. Une part importante de fonctions matérielles, plus de 75% de l'application, est la première condition nécessaire (mais pas suffisante) pour des gains significatifs. Le parallélisme des tâches, c'est-à-dire leur capacité à s'exécuter parallèlement, influence aussi fortement la qualité des résultats. Si les bénéfices de l'accélération statique ou de la RDP sont garantis la plupart du temps par rapport à l'exécution logicielle, l'existence d'un gain potentiel de la RDP par rapport à l'accélération statique est beaucoup moins triviale. Pour véritablement améliorer l'efficacité de traitement, la plateforme doit fournir les ressources nécessaires à l'exploitation du parallélisme. Sur le cas d'étude de l'application LPR, la RDP permet une amélioration jusqu'à un facteur 3 de l'efficacité énergétique par rapport à des accélérateurs statiques, dans les conditions de parallélisme opportunes (e.g. 4-lane LPR).

Les conditions essentielles pour obtenir ces résultats sont l'ordonnancement et le partitionnement FPGA. Pour tenter d'en donner une mesure, nous avons configuré une simulation où un ordonnanceur EDF standard (*Earliest Deadline First*) est utilisé à la place de l'ordonnanceur EAHS (section 4.2). Pour la configuration 4-lane LPR, les résultats montrent une baisse de 63,4% de l'efficacité énergétique pour XC7Z045 / Cortex-A9 et 14,6% pour XC6VLX240T / MicroBlaze. Ces comparaisons donnent une idée de l'impact potentiel de l'ordonnancement et du partitionnement dont l'efficacité énergétique dépend étroitement. Cette étude montre ainsi l'importance d'une approche holistique méthodique d'exploration et de déploiement pour exploiter efficacement les techniques de RDP. Alors que des bénéfices sur une exécution logicielle sont relativement faciles à obtenir, la véritable question est de savoir si la RDP mérite l'effort de conception supplémentaire par rapport à l'accélération statique. En cas de mauvais choix statiques (dimensionnement et partitionnement) et dynamiques (ordonnancement et déploiement), le potentiel de la RDP sera sous exploité car un grand nombre de paramètres statiques et dynamiques complexes entrent en jeu dans l'utilisation efficace de l'hétérogénéité. Par exemple, un point crucial porte sur l'identification d'un compromis en termes technologiques (types de cœurs, FPGA, technologie silicium) et de partitionnement (nombre, taille et ressources de la logique programmable) car une surface reconfigurable trop petite ne permettra pas toujours d'exploiter le maximum de parallélisme, tandis qu'une surface trop grande consommera beaucoup en raison de la puissance statique relativement importante des FPGAs. Pour une même application, cela peut engendrer des variations très significatives des résultats d'une plateforme à une autre. Si on reprend l'exemple de l'application LPR, la configuration 1-lane LPR est deux fois plus éco énergétique en reconfiguration dynamique qu'en reconfiguration statique sur XC6VLX240T / MicroBlaze, grâce à la réduction en taille de la logique programmable (1 RR contre 3 accélérateurs) et compte tenu de l'importante consommation statique du composant. Sur Zynq-7000, les deux solutions ont une efficacité énergétique similaire, ce qui rend moins attractive la RDP parce qu'elle est plus complexe à mettre en œuvre.

Ces résultats soulignent l'influence prédominante de l'exploration et de l'ordonnancement dans la complexité croissante des décisions liées à l'exécution sur des plateformes multiprocesseur hétérogènes. La méthodologie définie à partir de l'intégration étroite de ces deux étapes a permis de déterminer des déploiements à base de RDP sensiblement plus éco efficaces que l'accélération statique et, de manière tout aussi importante, de savoir *si* et *comment* la reconfiguration dynamique peut être utilisée correctement. Grâce à cette approche, l'efficacité énergétique a pu être amélioré jusqu'à un facteur 19 (au niveau de l'application globale) par rapport à une exécution logicielle, ce qui ouvre de nombreuses perspectives pour les systèmes soumis à des contraintes énergétiques critiques tels que les systèmes embarqués (IoT, sécurité) ou le calcul haute performance (Exascale), l'intelligence artificielle (embarquée), …. L'étude suivante s'intéresse justement aux énormes défis du calcul HPC (*High Performance Computing*) dans ce domaine par une collaboration avec Bull Atos technologies menée à travers la thèse CIFRE de M. Joël WANZA dont les contributions s'intègrent directement dans la série des projets FP7 et H2020 Mont-Blanc [17].

5.4 Etude de cas: les défis de l'Exascale

5.4.1 Introduction

Les performances des super calculateurs ont traditionnellement suivi une croissance continue en suivant la loi de Moore et les avancées du calcul parallèle, tant que les contraintes énergétiques pouvaient être considérées comme un problème secondaire. Mais il est rapidement devenu évident que la consommation

allait être le défi principal du passage à la prochaine génération (horizon 2020 - 2025). On estime en effet qu'il faudra améliorer l'efficacité énergétique d'un facteur 20, au minimum, pour pouvoir mettre en œuvre les premiers supercalculateurs de classe *Exascale* (10^{18} FLOPS). L'idée alors de réutiliser des concepts issus des technologies embarquées s'est donc naturellement imposée. Les premiers prototypes basés sur l'utilisation d'un très grand nombre de cœurs faible consommation (jusqu'à plusieurs millions) au lieu de cœurs performants mais plus complexes ont commencé à être explorés, mettant en évidence certaines améliorations indispensables au niveau nœud pour satisfaire les exigences du HPC. Ces travaux ont couvert une variété de cœurs RISC 32-bit qui s'étend des processeurs ARM Cortex-A8 et Cortex-A9 jusqu'aux plus récents Cortex-A15 et Cortex-A7. Les premiers résultats ont ainsi révélé plusieurs limitations pour satisfaire les exigences du HPC, en termes de précision en calcul flottant, de contrôleur mémoire 32-bit (limite de l'espace adressable), de mémoire ECC (*Error Correcting Code*) pour les calculs scientifiques et financiers, et de communications rapides. Certains travaux ont montré que la variabilité en performance et énergie sont dues en grande part aux calculs flottant et SIMD, et aux interactions avec le système mémoire. D'autres recherches effectuant une comparaison explicite avec des systèmes basés sur des cœurs x86 notent aussi ce manque de performance et concluent que l'avantage en coût des cœurs ARM diminue progressivement pour les applications avec de forts besoins en puissance de calcul (e.g. application serveur Web dynamique, transcodage vidéo). Certains considèrent même que les clusters x86 peuvent atteindre la même efficacité énergétique, en fonction de l'adéquation des traitements avec les ressources micro architecturales du cœur.

De ces différentes recherches sur la pertinence des cœurs ARM pour le calcul HPC, les efforts se sont largement concentrés sur les performances au niveau nœud en utilisant des mini ou des micro applications simples. Peu d'études ont pu expérimenter des systèmes à plus grande échelle dépassant quelques cœurs bien que les performances de nœuds *clusterisés* multi cœurs soient un aspect essentiel des futurs calculateurs Exascale. Si on tient compte par ailleurs de l'apparition récente de nouvelles générations de cœurs *ARMv8-A* qui apportent des améliorations spécifiquement développées pour le HPC (arithmétique et espace adressable 64-bit, communications hautes performances, hiérarchie mémoire optimisée), le travail qui suit est un des premiers à décrire des résultats d'une exploitation de l'ISA ARM64-bit pour du HPC. Il s'attache premièrement à effectuer une évaluation des outils, modèles et plateformes qui peuvent être utilisés pour définir les fondations d'une exploration système méthodique, avec des applications HPC représentatives, et jusqu'à 128 cœurs ARM 64-bit. La méthodologie est employée par la suite à examiner différentes hypothèses architecturales autour d'un partitionnement au niveau puce (SoC) pour réduire la complexité, le coût et la consommation.

5.4.2 Méthodologie d'exploration

Juno ARM Development est la première plateforme disponible qui supporte l'ISA *ARMv8-A*. L'intérêt réside premièrement dans le cluster performance (dual Cortex-A57 du processeur *MPCore big.LITTLE*), le réseau d'interconnexion CCI-400 (*Cache Coherent Interconect*) et le contrôleur mémoire DDR3-1600. Dans un effort de caractérisation de plateformes réelles, nous avons aussi utilisé une carte *AMD Seattle* composée de quatre clusters de deux cœurs ARM Cortex-A57 avec un réseau d'interconnexion cohérent AMD à 2 GHz et deux contrôleurs mémoire DDR4-3733. Nous avons également considéré la plateforme *AppliedMicro* (APM) X-Gene1 basée sur quatre clusters de deux cœurs 64-bit fonctionnant à 2,4 GHz, un réseau d'interconnexion APM cohérent et un contrôleur mémoire DDR3 (16GB).

L'utilisation de plateformes virtuelles permet ensuite d'étendre l'espace d'exploration par le passage à une large échelle et l'évaluation de différents cœurs 64-bit (e.g. Cortex-A72). La plateforme *ARM Fast Models* (AFM) est la plus grande qui puisse être configurée à cet égard, jusqu'à 48 cœurs A57 / A72. Cette limitation à 48 vient du réseau CCN-512 qui supporte un maximum de douze clusters (de quatre cœurs) cohérents. *Virtual Processing Unit* (VPU) *& Task Modeling*, qui font parti de la méthodologie *Synopsys Platform Architect*, ont été utilisés pour le passage à plus grande échelle ainsi que pour l'exploration du système mémoire et de l'*interconnect* cohérent. Ces outils se basent en effet sur une abstraction plus élevée de systèmes multicœurs en SystemC / TLM, reposant sur une génération de trafic (issues de simulations AFM) et de modèle TLM du réseau de communications, et permettent d'explorer des configurations de clusters plus complexes.

Les deux plateformes AFM et VPU sont exploitables sous *Platform Architect* pour permettre une analyse des performances et de la consommation. La Figure 42 illustre une vue générale de la méthodologie de simulation.

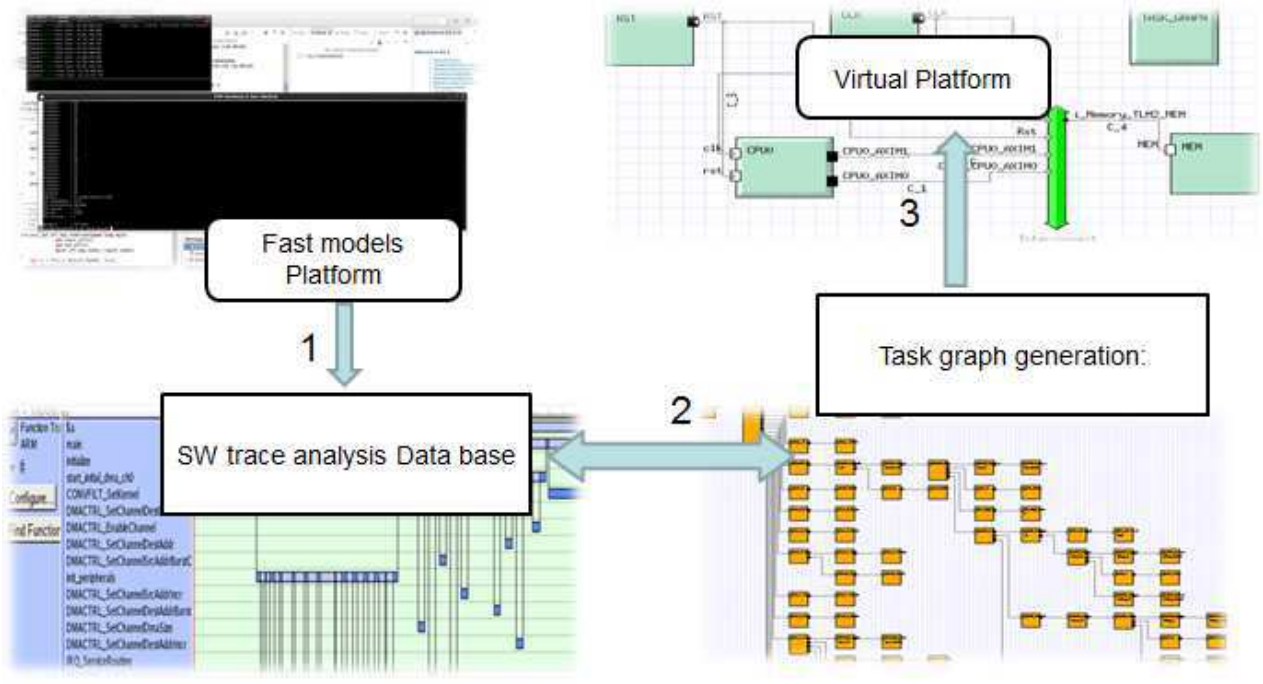


Figure 42 – Méthodologie d'exploration

Celle-ci se base premièrement sur une combinaison d'exécutions sur plateformes réelles et virtuelles quand le système ne comporte que quelques nœuds. Les graphes de tâches sont ensuite produits à partir des résultats et utilisés pour simuler jusqu'à 128 cœurs par la modélisation VPU. En plus de disposer des deux plateformes virtuelles dans le même environnement, cet outil permet une évaluation des modèles ARM existants et une analyse architecturale plus avancée grâce à la complémentarité des plateformes AFM et VPU. Pour les besoins de cette étude, ces deux plateformes ont été paramétrées dans des configurations qui correspondent aux plateformes réelles décrites précédemment, dans le but de vérifier en premier lieu la corrélation en termes de performances (GFLOPS, cycle par instruction, etc.) et de hiérarchie mémoire (statistiques au niveau cache, bande passante). Les détails et résultats de cette confrontation des modèles sont décrits en détail dans [18]. Nous nous focaliserons par la suite sur l'analyse du problème d'exploration architecturale en vue d'améliorer l'efficacité énergétique et les traitements HPC.

5.4.3 Exploration architecturale

Cette exploration étudie en particulier les possibilités de partitionnement au niveau SoC. Cette approche originale se justifie par l'énorme complexité (et les coûts associés) des technologies SoCs monolithiques à envisager. Sur la base d'une puce classique qui intègrerait 128 cœurs, une technologie mémoire rapide (HBM/HMC) et le *Bull Exascale computing network interface controller* (BXI NIC), l'idée est donc d'évaluer des topologies multi-SoC cohérentes (deux SoCs / 64 cœurs, quatre4 SoCs / 32 cœurs) qui communiquent par des connexions point-à-point via des *proxy* cohérents (haut de la Figure 43).

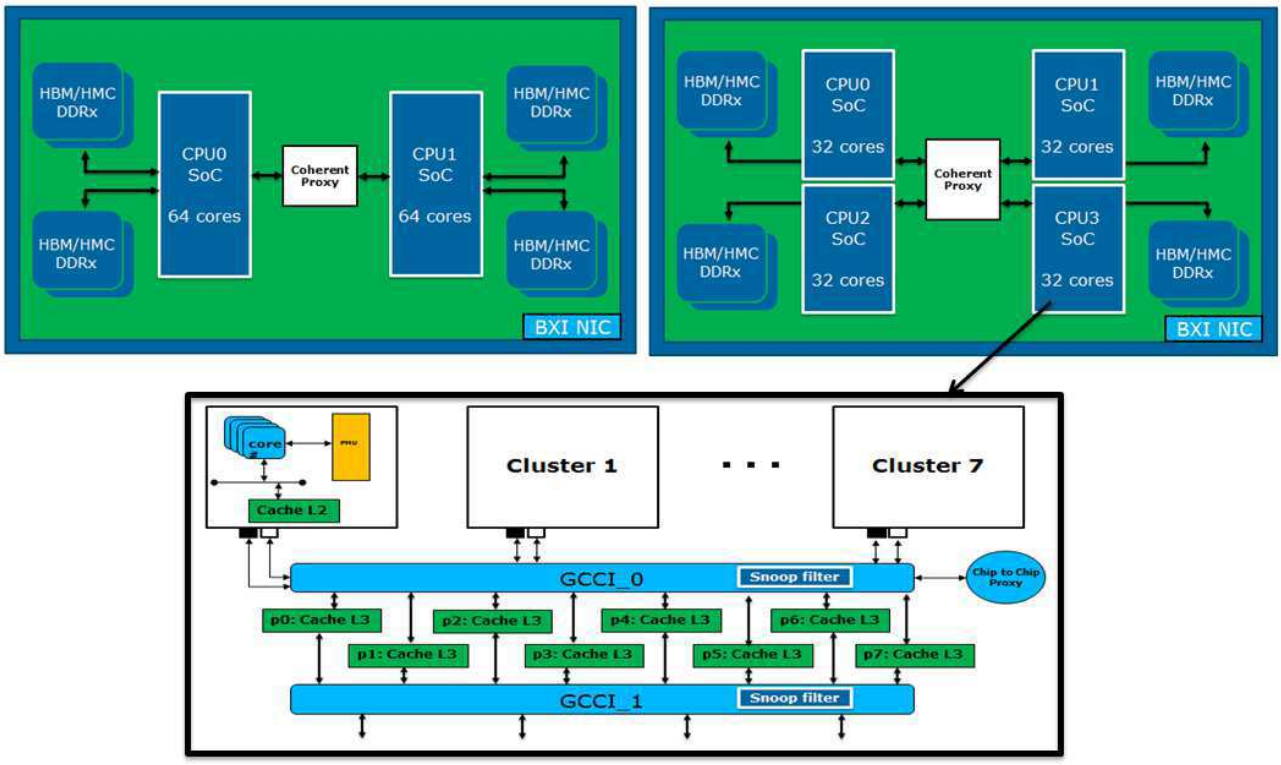


Figure 43 – Vue interne niveau SoC et cluster (2 SoCs / 64 cœurs, 4 SoCs / 32 cœurs)

En considérant ce modèle d'architecture global et la hiérarchie mémoire associée, un engorgement apparait au niveau de la cohérence de cache car les performances des protocoles utilisés supportent mal le passage à l'échelle, en particulier pour du HPC qui exploite un très grand nombre de cœurs. C'est pourquoi Bull a souhaité introduire un protocole de cohérence par répertoire [19] dans leur réseau d'interconnexion SCI (*SoC Interconnect*) afin de rendre possible le partitionnement cohérent inter puces. Contrairement à un protocole classique basé sur l'observation des modifications de données faites par les autres processeurs, l'utilisation d'un répertoire permet de stocker la localisation de chaque modification et d'envoyer les requêtes de cohérence uniquement vers les contrôleurs concernés. Outre la réduction des coûts de développement qui est le premier objectif, la réduction de complexité obtenue doit aussi permettre de diminuer l'activité et l'énergie consommée par l'*interconnect* et la mise à jour de caches distants.

Deux configurations sont définies pour le partitionnement d'une topologie mono-SoC 128 cœurs en deux et quatre puces (Figure 44). Tous les processeurs dans un scénario partitionné doivent pouvoir communiquer comme s'ils étaient sur le même réseau d'interconnexion cohérent *on chip*. Les puces sont donc connectées entre elles par des *proxys* cohérents. Leur rôle principal est précisément de garantir la cohérence des transactions avec les *sockets* voisins et les zones mémoire externes. Le maintien de la cohérence est géré

par un module *snoop controller* présent dans chaque *proxy* du SCI. Par exemple, si le SCI est configuré avec 32 ports (32 *clusters* de quatre cœurs), chaque *snoop controller* doit pouvoir communiquer avec les 31 autres contrôleurs. Dans l'hypothèse d'un partitionnement en deux SoCs, il ne communique plus qu'avec 16 contrôleurs (15 autres ports et le *proxy*) ce qui permet de réduire sensiblement leur complexité et leur coût.

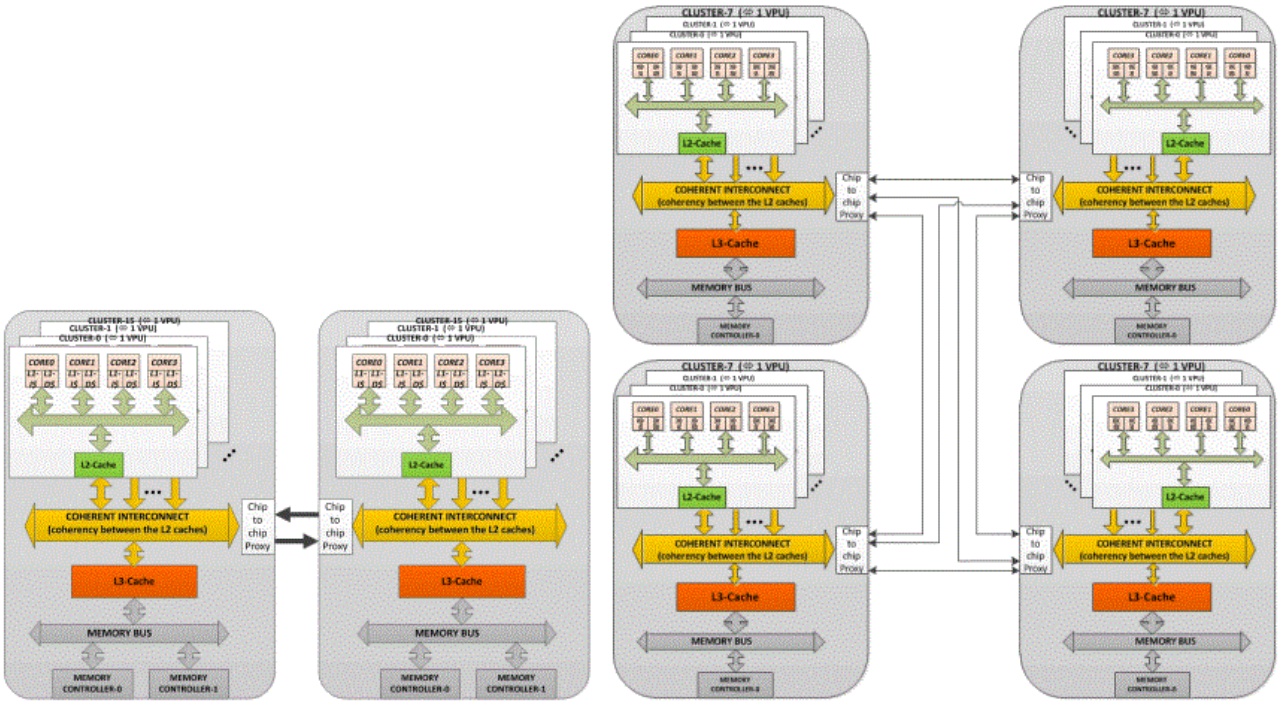
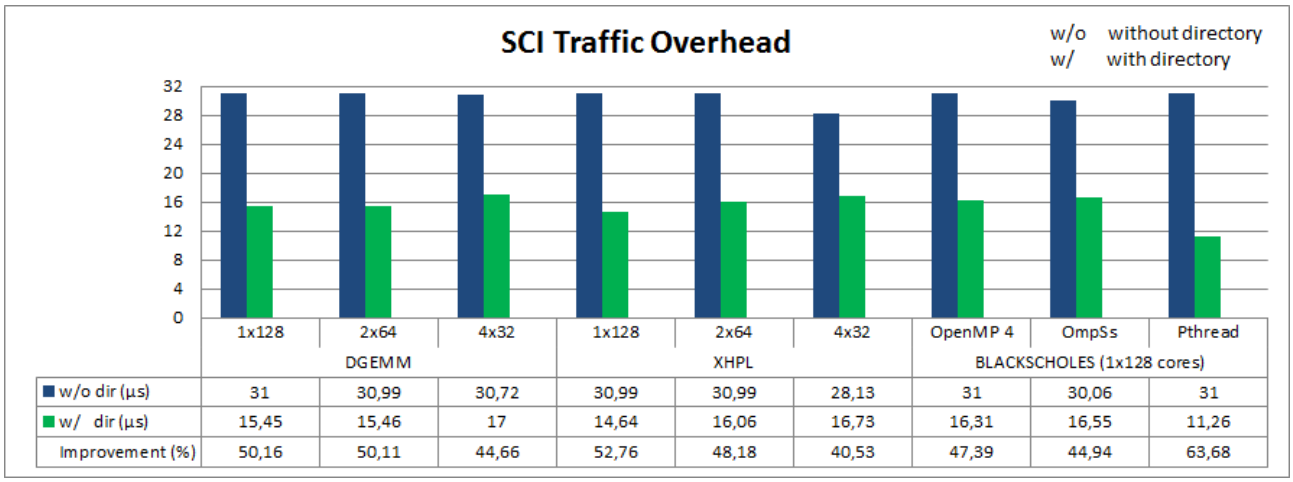


Figure 44 – Partitionnements 2x64 et 4x32 par proxy cohérents

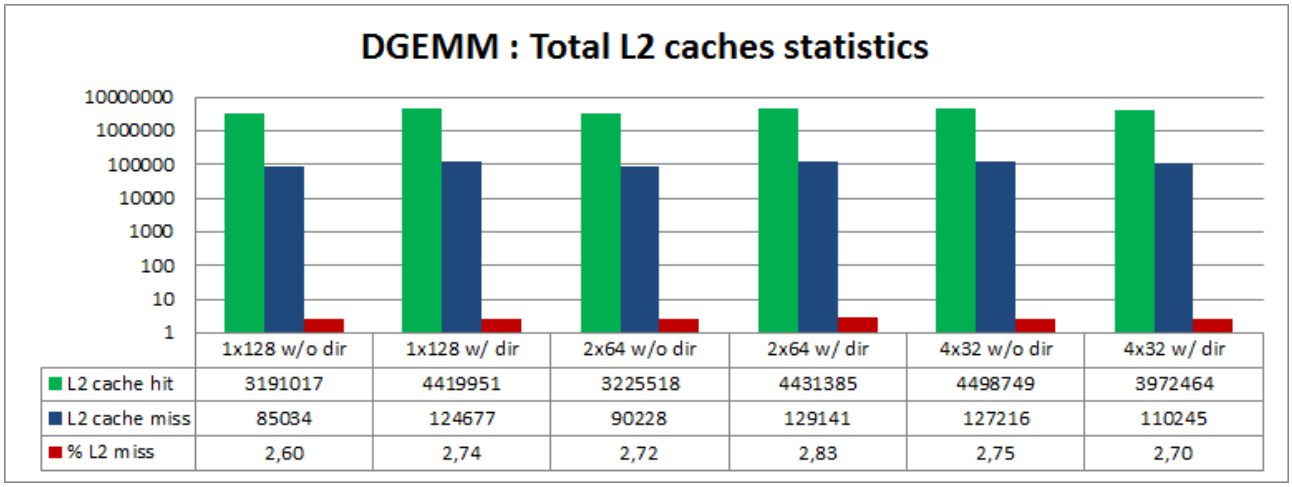
Les simulations qui suivent visent à évaluer la réduction de ces coûts. On s'intéresse en premier lieu à vérifier l'aptitude du *snoop controller* à diminuer le trafic dû à la cohérence de cache et à son impact sur les performances. On analysera ensuite la pertinence d'un partitionnement en deux (2x64 cœurs) ou quatre SoCs (4x32 cœurs) par rapport à une solution monolithique 1x128, sans et avec le mécanisme de cohérence par répertoire. Enfin on considèrera l'effet de différents modèles de programmation parallèles sur le trafic interne et les performances correspondantes.



	1x128	2x64	4x32	1x128	2x64	4x32	OpenMP 4	OmpSs	Pthread
	DGEMM			XHPL			BLACKSCHOLES (1x128 cores)		
w/o dir (µs)	31	30,99	30,72	30,99	30,99	28,13	31	30,06	31
w/ dir (µs)	15,45	15,46	17	14,64	16,06	16,73	16,31	16,55	11,26
Improvement (%)	50,16	50,11	44,66	52,76	48,18	40,53	47,39	44,94	63,68

Figure 45 – Nombre de transactions générées pour différentes configurations multi-SoC

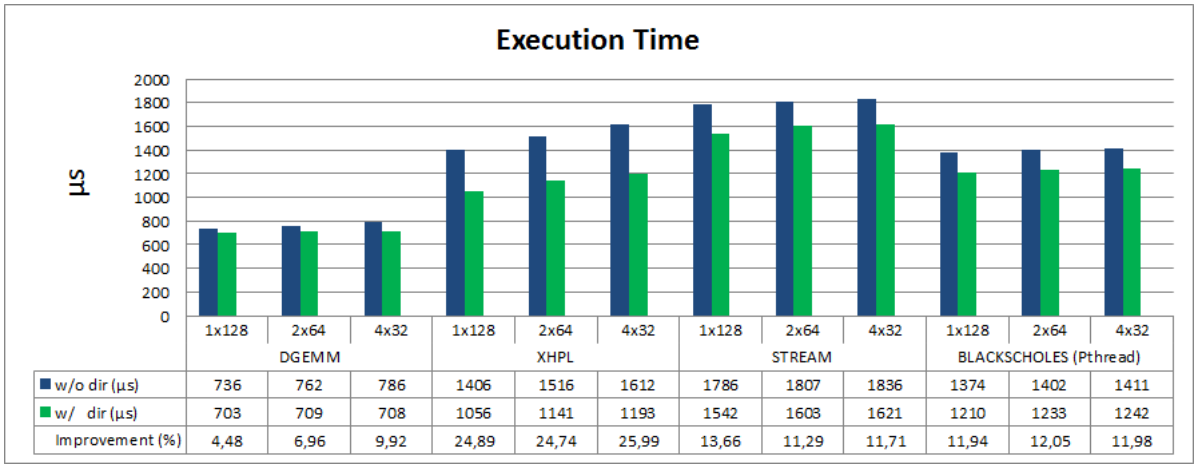
La Figure 45 montre le nombre de transactions liées à la cohérence de cache générées, sans et avec répertoire, pour chacune des trois configurations de 32 maîtres (1x128, 2x64, 4x32). Dans une configuration conventionnelle (1x128 sans répertoire), 31 transactions doivent être générées pour chaque défaut de cache (ou requête d'invalidation) et mettre à jour chaque copie de la donnée concernée. L'utilisation d'un répertoire réduit le trafic correspondant dans le SCI entre 40% et 63%, ce qui est conforme à la réduction en taille d'un facteur deux du *snoop controller* passant de 31 à 16 ports de sortie. La réduction du nombre de transactions impacte à son tour les temps d'exécution applicatifs en réduisant les pénalités de défaut de cache L2.



	1x128 w/o dir	1x128 w/ dir	2x64 w/o dir	2x64 w/ dir	4x32 w/o dir	4x32 w/ dir
L2 cache hit	3191017	4419951	3225518	4431385	4498749	3972464
L2 cache miss	85034	124677	90228	129141	127216	110245
% L2 miss	2,60	2,74	2,72	2,83	2,75	2,70

Figure 46 – Statistiques de cache (exemple DGEMM)

La Figure 47 compare les temps d'exécution des mêmes exemples, sans et avec répertoire, pour examiner ces gains. L'utilisation d'un répertoire dans le modèle multi-SoC partitionné apporte en moyenne une amélioration de 14% (4% à 26%) des temps d'exécution. Malgré la réduction du nombre de transactions d'un facteur deux, les gains en performance sont limités par le nombre relativement peu élevé de défauts de cache en pratique dans les applications réelles (autour de 2,7% pour l'exemple DGEMM, Figure 46).



	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32
	DGEMM			XHPL			STREAM			BLACKSCHOLES (Pthread)		
w/o dir (µs)	736	762	786	1406	1516	1612	1786	1807	1836	1374	1402	1411
w/ dir (µs)	703	709	708	1056	1141	1193	1542	1603	1621	1210	1233	1242
Improvement (%)	4,48	6,96	9,92	24,89	24,74	25,99	13,66	11,29	11,71	11,94	12,05	11,98

Figure 47 – Performances pour différentes configurations multi-SoC

Maintenir la consistance des données partagées entre les processeurs est une tâche particulièrement complexe dans les systèmes de calcul grande échelle. L'utilisation d'un filtrage par répertoire dans les trois configurations multi-SoC permet à la fois d'assurer la consistance des données entre puces mais aussi

d'améliorer l'efficacité en réduisant les transactions de cohérence au strict nécessaire. Le surcoût entraîné pour chaque transaction entrante est ainsi réduit de plus de 40% ce qui est un aspect important pour le passage à l'échelle au niveau du système mémoire. Il est par ailleurs assez notable que les performances se trouvent légèrement améliorées dans le même temps. La consistance mémoire entre *clusters* est donc assurée avec sensiblement moins de trafic à la lumière des résultats. Malgré un bénéfice relativement limité en performances nettes, les avantages de la cohérence par répertoire suffisent à justifier un partitionnement multi-SoC avec des latences de communication inter puce suffisamment faibles. C'est précisément ce qui fait l'objet des mesures suivantes.

Si on se focalise maintenant sur une comparaison des configurations multi-SoC (*2x64, 4x32 w/o dir*) et mono-SoC (*1x128 w/o dir*) dans les résultats de la Figure 47, on observe une légère dégradation légitime des performances en l'absence de répertoire, qui augmente progressivement en fonction du partitionnement du fait des latences de transactions inter puce. Envisager un partitionnement ne semble donc pas être une approche très efficace à priori avec jusqu'à 15% de baisse en performance sur l'exemple XHPL pour un partitionnement 4x32. Néanmoins en présence du filtrage par répertoire, les performances des configurations mono et multi-SoC s'améliorent toutes les deux (respectivement 13,7% et 14,3%) puisque les transactions sont réduites au minimum nécessaire pour garantir la cohérence. La complexité supplémentaire du *snoop controller* est largement compensée par la suppression des temps d'attente internes et externes. Le gain varie en fonction du type de traitement et s'étend de 4% à 25%. XHPL, qui nécessite un espace mémoire important et qui est très sensible aux latences, fourni un exemple idéal pour montrer l'impact du répertoire sur l'*interconnect on-chip* avec 25% d'amélioration en performance. Une autre caractéristique intéressante est la stabilité des performances en présence du répertoire, avec une variation moyenne de 1,9% sur les différentes configurations SoC et exemples d'applications. Si on compare les topologies partitionnées (*2x64, 4x32 w/ dir*) et non partitionnées (*1x128 w/ dir*), on peut vérifier que l'impact du partitionnement sur les temps d'exécution a été efficacement limité par le filtrage par répertoire (4,3% pour 2x64 et 5,4% pour 4x32). Les performances sont plus stables dans les deux configurations partitionnées parce qu'il existe toujours, pour une donnée partagée, une copie dans les caches voisines, ce qui évite les défauts de *snoop*.

Ces résultats laissent donc entrevoir une dégradation moyenne des performances de 5,2% si on envisage de partitionner le SoC (en deux ou en quatre pour 128 cœurs). Toutefois l'emploi d'un mécanisme de filtrage par répertoire est suffisamment efficace pour réduire les transactions inter puce et éliminer cette perte de performance, avec même un gain moyen possible de 10% (par rapport à un mono-SoC sans filtrage). Le mécanisme d'extension de cohérence ouvre ainsi des possibilités très intéressantes telles que l'intégration de nœuds de calcul (*compute nodes*) plus nombreux directement sur un interposeur *System-in-Package*, en exploitant éventuellement des technologies 3D *Through Silicon Vias* (TSVs) et *High Memory Bandwidth* (HBM) qui peuvent aussi permettre d'améliorer l'efficacité globale.

Un autre facteur susceptible d'affecter l'intérêt d'un partitionnement multi-SoC est lié au parallélisme logiciel. La question ici est de savoir comment minimiser au mieux les transactions sortantes entre les puces partitionnées aux niveaux programmation et modèle de programmation. Nous avons ainsi considéré trois modèles de programmation parallèles (*OpenMP*, *OmpSs* et *Threads POSIX*) sur un exemple d'application *blackscholes* (qui fait parti de la série de *benchmarks* PARSEC [20]) pour examiner leur influence sur l'efficacité du filtrage par répertoire.

OpenMP, *OmpSs* et *Threads POSIX* sont des APIs (*Application Programming Interface*) pour la programmation parallèle multiplateformes à modèle de mémoire partagée. *OpenMP* fournit des mécanismes de gestion de tâches parallèles haut niveau utilisant des annotations de code et de graphe pour leur exécution et synchronisation. *Threads POSIX* est une API de plus bas niveau offrant des fonctionnalités très spécifiques de contrôle de tâches POSIX.

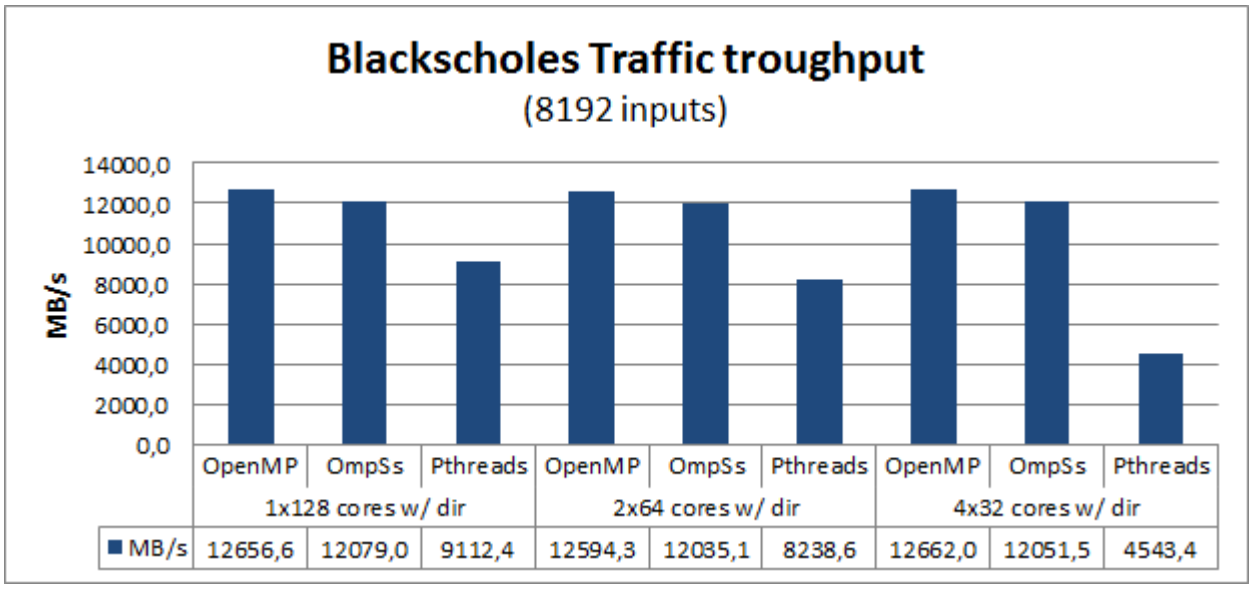


Figure 48 – Impact du modèle de programmation sur le débit (*blackscholes*)

Contrairement à *OpenMP*, l'utilisation de *Pthreads* nécessite une expression explicite du parallélisme dans le code. *OmpSs* est une combinaison d'*OpenMP* et de *StarSs*, un modèle de programmation développé par le *Barcelona Supercomputing Center* pour du HPC. Il fournit un ensemble d'extensions à *OpenMP* pour permettre de supporter des tâches asynchrones, l'hétérogénéité (accélérateurs) et de tirer plus de performances des architectures parallèles homogènes et hétérogènes.

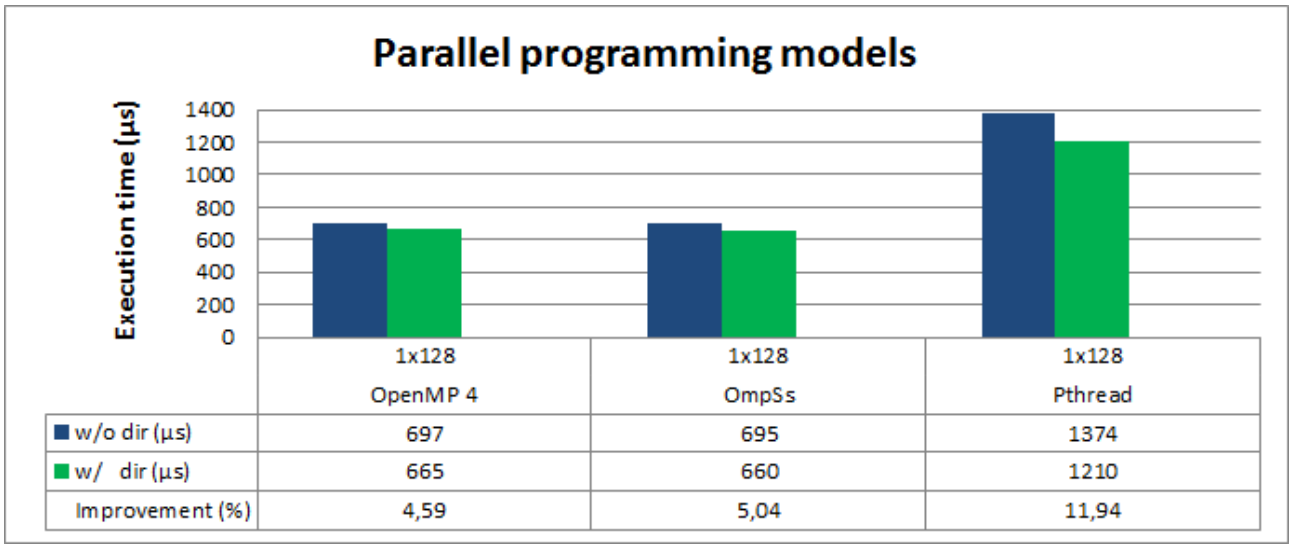
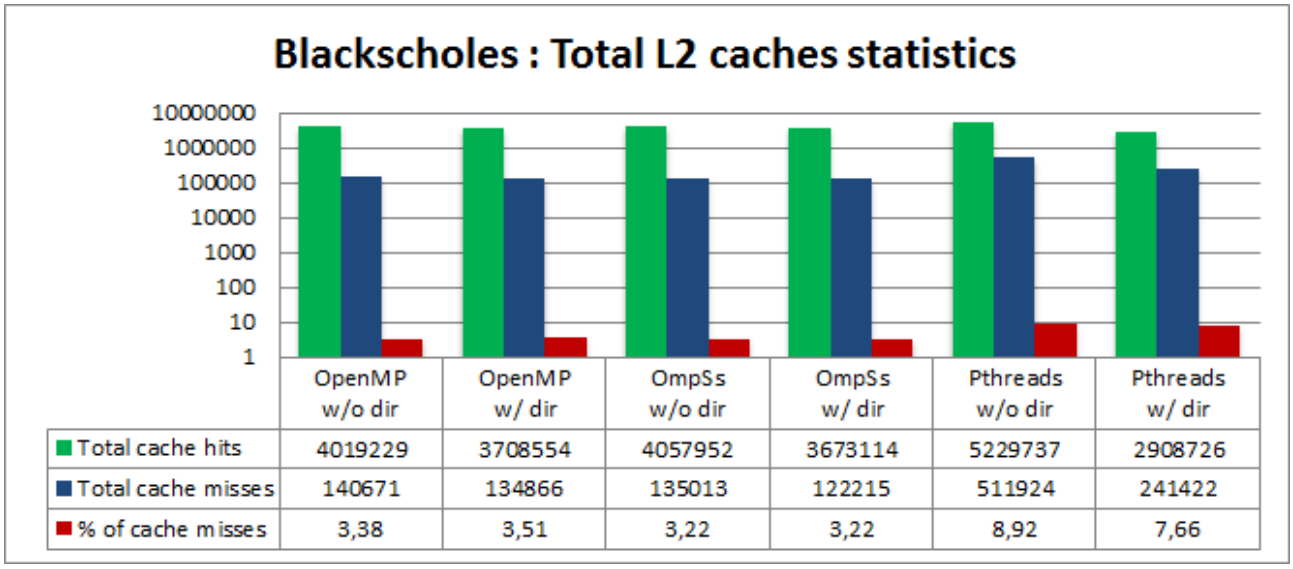


Figure 49 – Impact du modèle de programmation sur les performances (*blackscholes*, 1x128 cœurs)

Les résultats précédents montrant une faible influence du partitionnement en présence d'un répertoire, l'analyse suivante est limitée à la configuration mono-SoC. La Figure 49 montre les temps d'exécution de

l'exemple *blackscholes* pour chaque modèle de programmation. Il y a comparativement peu de différences entre les résultats d'*OpenMP* et *OmpSs* du fait de leur similarité. L'application tire peu de profit des extensions spécifiques d'*OmpSs* pour exploiter ce modèle d'architecture.

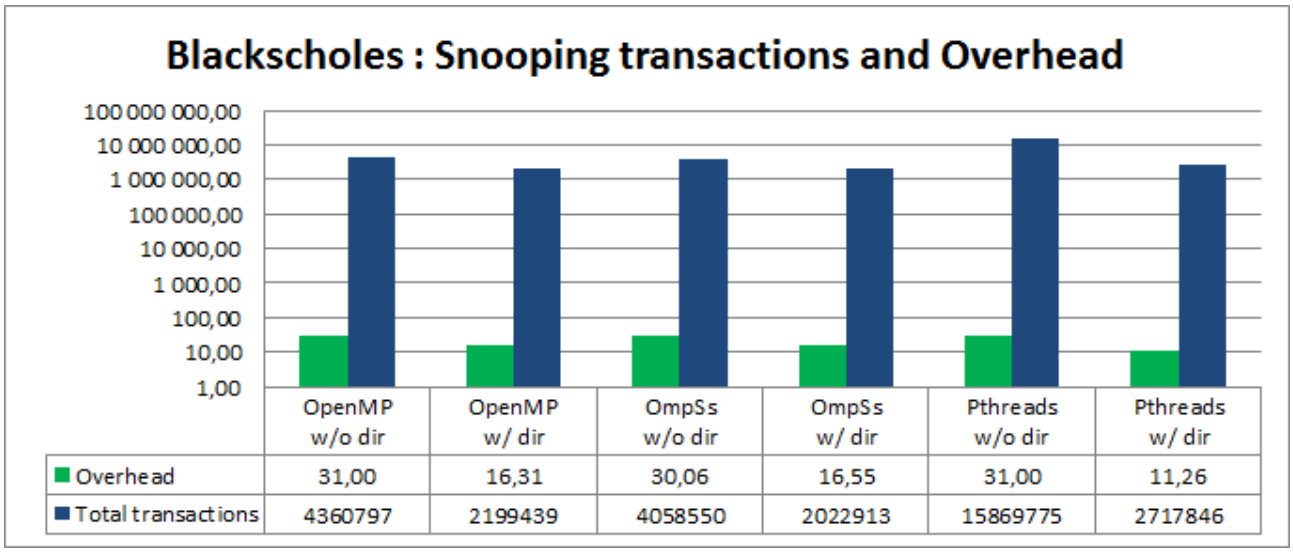


	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
Total cache hits	4019229	3708554	4057952	3673114	5229737	2908726
Total cache misses	140671	134866	135013	122215	511924	241422
% of cache misses	3,38	3,51	3,22	3,22	8,92	7,66

Figure 50 – Impact du modèle de programmation sur le nombre de défauts de cache L2 (*blackscholes*, 1x128 cœurs)

Néanmoins, les Figure 48 et Figure 49 montrent clairement deux points d'inflexion à 82,6% en performance et 40,9% en débit pour *Pthreads* par rapport à *OpenMP* et *OmpSs*. Une analyse plus approfondie montre que *Pthreads* a 28% de débit en moins qu'*OpenMP* (*1x128 w/ dir*) pour 79% de défauts de cache en plus (Figure 50). Ainsi les défauts de cache sont responsables de la génération de 23,5% plus de trafic dans le SCI par rapport à *OpenMP* (Figure 51).

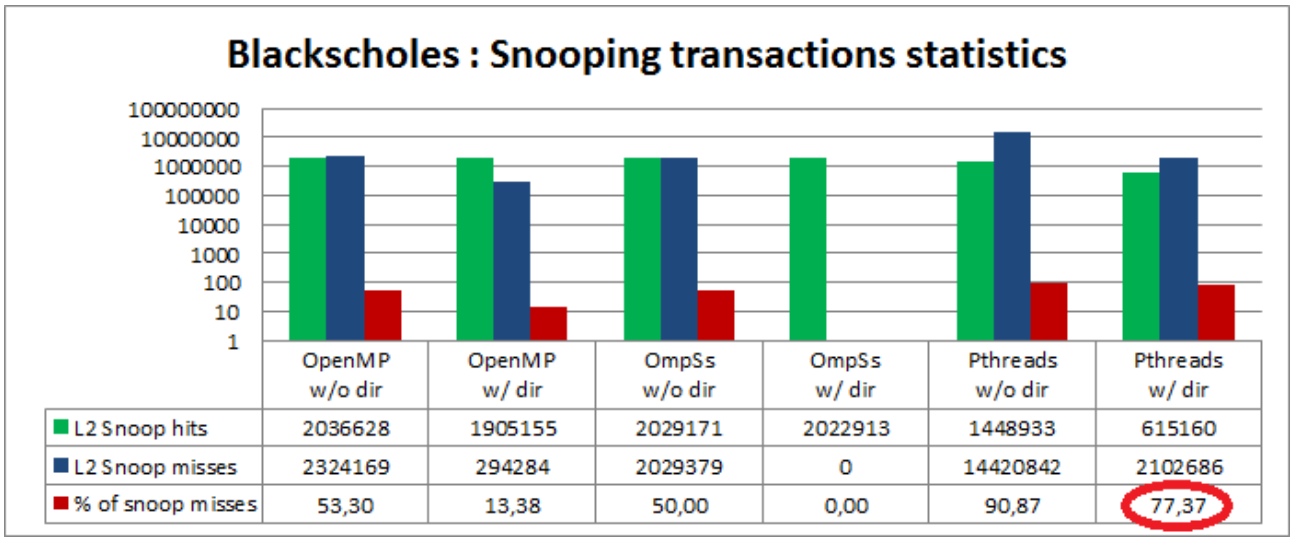
La Figure 52 confirme que *Pthreads* ne fait pas bon usage du filtrage par répertoire. Avec 77,4% de défaut de *snoop*, l'*interconnect* doit gérer cinq fois plus de transactions qu'*OpenMP*. Cette faiblesse tient principalement à la capacité du modèle de programmation à exploiter d'importants degrés de parallélisme.



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
Overhead	31,00	16,31	30,06	16,55	31,00	11,26
Total transactions	4360797	2199439	4058550	2022913	15869775	2717846

Figure 51 – Impact du modèle de programmation sur le nombre de transactions (*blackscholes*, 1x128 cœurs)

Dans les versions parallélisées du graphe de tâche de l'exemple *blackscholes* utilisées pour *OpenMP* et *OmpSs*, le travail est mieux réparti en unités de taille prédéfinie qui permettent d'obtenir plus d'instances de tâches et une meilleure équilibre de la charge qu'avec *Pthread*. Les effets d'un parallélisme moins facile à exploiter peuvent alors augmenter significativement (jusqu'à un facteur deux) à l'échelle de 128 cœurs.



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
L2 Snoop hits	2036628	1905155	2029171	2022913	1448933	615160
L2 Snoop misses	2324169	294284	2029379	0	14420842	2102686
% of snoop misses	53,30	13,38	50,00	0,00	90,87	77,37

Figure 52 – Statistiques de *snoop* (*blackscholes*, 1x128 cœurs)

Malgré le fait qu'il soit difficile de tirer des conclusions sur l'efficacité réelle d'un modèle de programmation par rapport à un autre, ce qui dépend aussi de la façon dont le logiciel est partitionné et codé, cette étude montre toutefois le potentiel d'une analyse logicielle / matérielle fine au niveau de l'exploitation du parallélisme de l'application. Les résultats sont utiles pour affiner l'architecture et le développement des algorithmes et du logiciel, pour identifier et corriger les éventuelles lacunes de programmation et pour améliorer l'efficacité du traitement parallèle.

5.4.4 Stratégies énergétiques spécialisées HPC

Les stratégies utilisées en pratique dans le domaine du HPC se basent généralement sur les politiques d'OS existantes, qui sont des stratégies généralistes simples consistant dans leur majorité à placer le système dans un état de fonctionnement ajusté (modes repos, P-States) en fonction du *workload* (section 5.2). Il existe des approches qui contraignent la réalisation des traitements sous une contrainte de coût énergétique (Power Cap), mais ce sont plus des techniques de limitation de la consommation maximum que de véritables stratégies explicites de réduction de la consommation. Or il existe certainement, comme dans le domaine de l'embarqué, un potentiel dans l'adaptation fine du matériel aux variations et aux particularités des applications en pratique. Dans le domaine de l'embarqué, l'objectif est généralement de minimiser le coût énergétique absolu pour traiter une application, et d'allonger l'autonomie. Les gains peuvent généralement s'exprimer par une métrique d''efficacité énergétique comme le produit énergie-performance. Le problème dans le domaine du HPC est différent car l'exigence absolue est toujours d'exécuter l'application avec les performances maximales. Cela implique que l'objectif est d'économiser l'énergie tant que cela n'affecte pas les performances, ou du moins en acceptant une dégradation tolérable (très faible).

Une première application intuitive serait par exemple d'exploiter la variation des temps d'exécution de tâches parallèles. Une condition fondamentale pour un système de traitement parallèle optimal est

d'équilibrer la charge CPU. Différents travaux qui s'exécutent sur plusieurs ressources se terminent en pratique à des moments différents. Comme le temps d'exécution global est souvent déterminé par la terminaison du travail le plus lent, il est alors possible de ralentir l'exécution des travaux plus rapides par DVFS. La réduction de fréquence qui en découle permet de diminuer la puissance consommée tout en préservant le niveau de performance, ce qui améliore l'efficacité énergétique en termes de performance par watt du système. Cette première approche sera désignée « soft big.LITTLE » par la suite en référence au concept d'équilibre de charge multiprocesseur hétérogène développé chez ARM que cette stratégie rejoint.

Une deuxième possibilité intéressante repose sur les communications. Du fait de l'existence inhérente d'un très grand nombre de tâches parallèles, le cas de figure où un travail final de rassemblement des résultats a de fortes chances de se produire dans les applications HPC. Ce scénario implique de nombreux mécanismes de synchronisation et de communication qui ne nécessitent pas d'utiliser les processeurs à leur vitesse maximale. Cette deuxième stratégie HPC sera dénommée « Blocking point strategy » par la suite. Par ailleurs, étant donné que les primitives de communication et de synchronisation sont utilisées sous forme de bibliothèques, la mise en place d'une stratégie « Blocking point » peut leur être directement intégrée ce qui la rend transparente pour l'application et facilite son utilisation. Par la suite, nous analyserons l'efficacité des deux stratégies proposées en simulation avec GEM5 et sur une plateforme représentative (Cavium ThunderX2 96 cœurs).

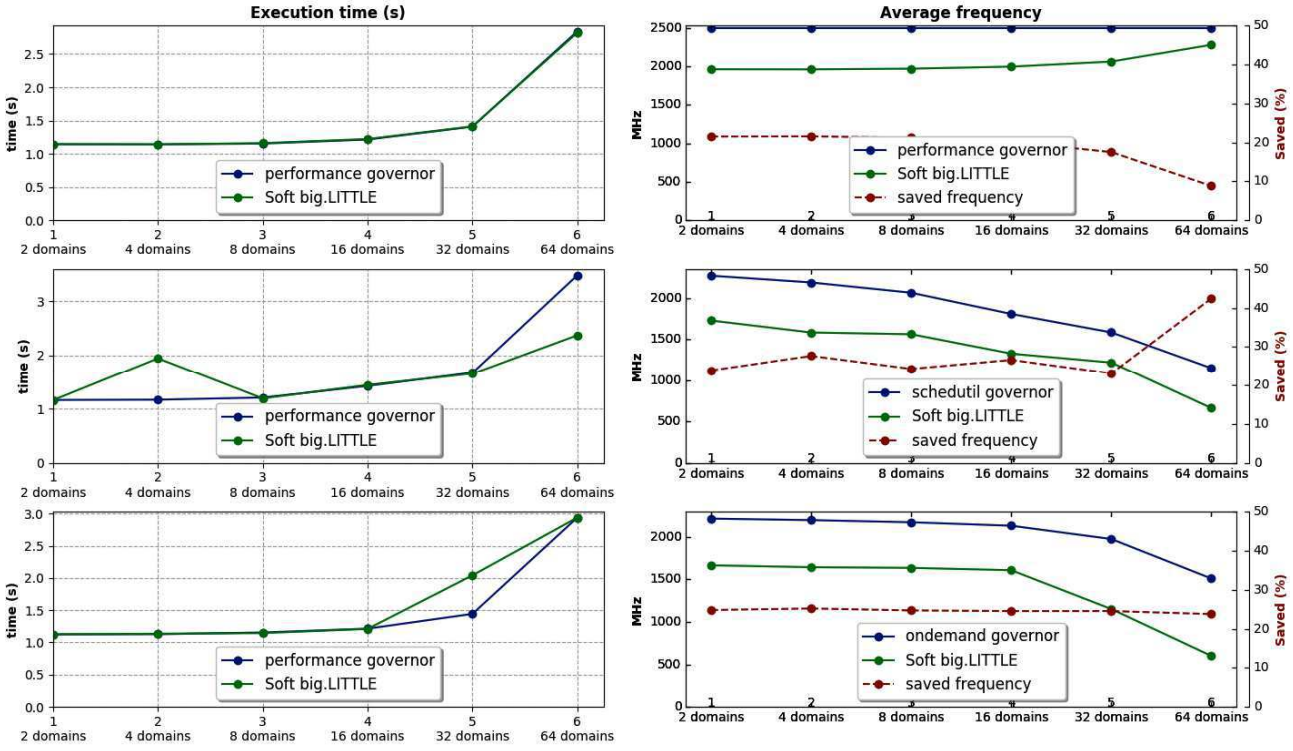


Figure 53 – Temps d'exécution et fréquence CPU moyenne pour la stratégie *soft big.LITTLE* (GEM5)

Les exemples utilisés pour ces évaluations de stratégies HPC sont basés sur la multiplication de matrice flottante. Deux versions ont été définies pour obtenir des traitements massivement parallèles. La première associe simplement la multiplication complète de deux matrices de dimension N à une tâche, et lance un nombre configurable de tâches en parallèle (version dénommée « matrix_duplicate_N_N » par la suite). Cette parallélisation équilibrée et indépendante se prête particulièrement bien à la stratégie « soft big.LITTLE ». La seconde version parallélisée réalise la multiplication d'une ligne de la première matrice par

une colonne de la seconde matrice dans une tâche distincte. Elle est dénommée « matrix_slice_N_N » par la suite et se prête quand elle très bien à l'évaluation de la stratégie « Blocking point » puisque les traitements séparés ligne * colonne doivent être rassemblés à la fin pour obtenir le résultat final en mémoire.

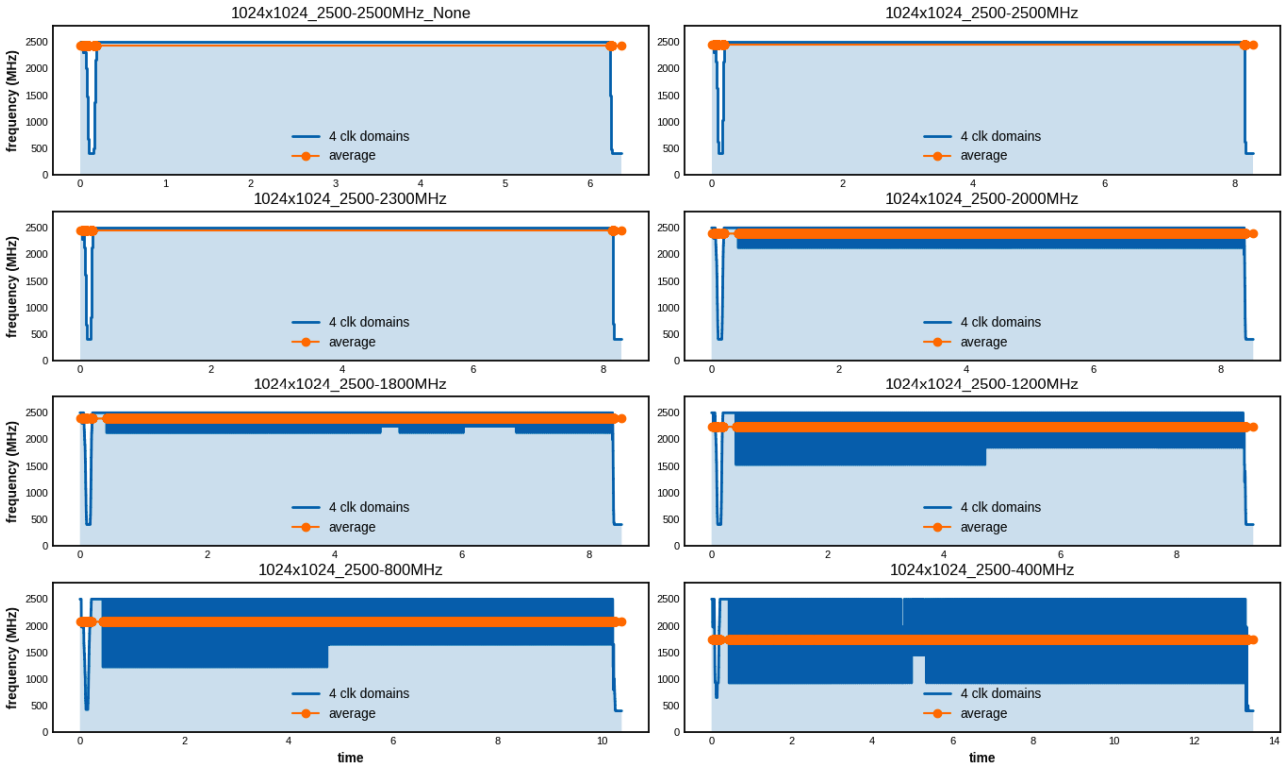


Figure 54 – Profils de fréquence CPU pour la stratégie *Blocking point* (GEM5)

Les simulations suivantes analysent les variations de fréquence de plusieurs domaines d'horloge sur l'exemple *matrix_duplicate_256_256*. Pour produire une variation contrôlée des temps d'exécution, la multiplication est configurée de sorte à distinguer le temps de traitement des tâches de numéros pairs et celui des tâches de numéros impairs. Le traitement des tâches impaires est défini pour avoir la moitié du temps d'exécution des tâches paires pour émuler des temps d'exécution différents. Ces variations sont nécessaires pour évaluer la stratégie dont le principe consiste à compenser les différences par DVFS, en ralentissant les tâches rapides quand c'est possible. Ainsi les tâches impaires de l'exemple *matrix_duplicate_256_256* peuvent s'exécuter à une fréquence réduite de moitié par rapport aux tâches paires. La Figure 53 montre l'exécution de *matrix_duplicate_256_256* pour 2, 4, 8, 16, 32 et 64 domaines d'horloges et plusieurs stratégies (*soft big.LITTLE* et Linux *Performance*, *Schedutils*, *onDemand* pour comparaison). La plateforme GEM5 est configurée de sorte qu'un domaine d'horloge correspond à deux cœurs, chaque cœur exécute une tâche et peut fonctionner à une fréquence distincte (DVFS par cœur). Les fréquences possibles correspondent à 400, 800, 1200, 1800 et 2500 MHz. La partie gauche de la Figure 53 reflète les temps d'exécution où l'on peut vérifier que les performances sont du même ordre pour les quatre stratégies. La partie droite de la Figure 53 révèle quand à elle le comportement en fréquence qui est différent. Pour simplifier les traces, seule la moyenne des fréquences sur les domaines d'horloge est représentée, ce qui correspond aussi à la tendance de la puissance dynamique moyenne. La stratégie *soft big.LITTLE* permet de réduire dans tous les cas la fréquence par rapport aux gouverneurs Linux évalués (*Performance*, *Schedutils*, *onDemand*) et montre des gains moyens (représentés en rouge) autour de 20%

pour cet exemple. Ces valeurs issues de la simulation sont données à titre indicatif et seront par la suite mieux évaluées par des mesures sur une plateforme réelle.

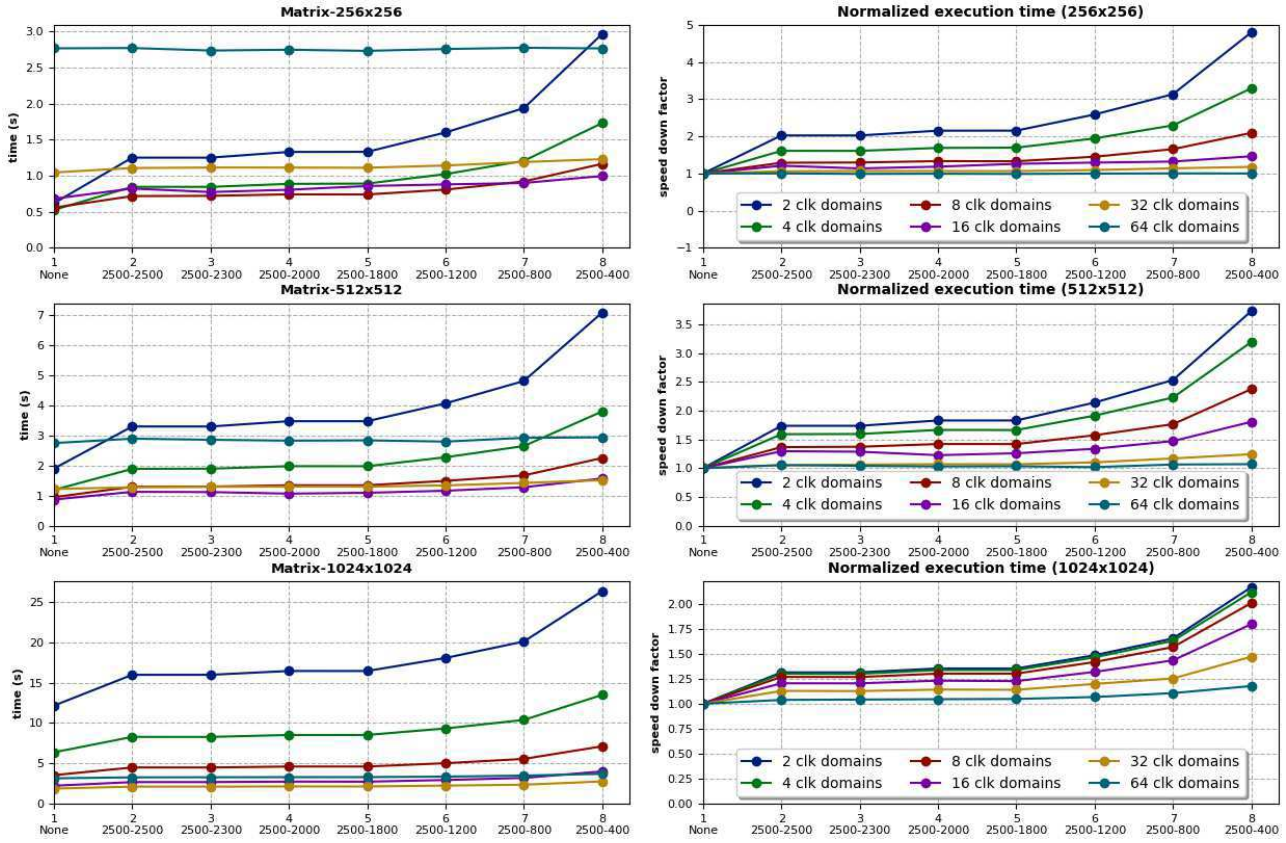


Figure 55 – Temps d'exécution pour la stratégie *Blocking point* (GEM5)

Les simulations GEM5 qui suivent concernent la stratégie « Blocking point ». La Figure 54 montre le comportement en fréquence CPU de la stratégie sur l'exemple *matrix_slice_1024_1024* pour une plateforme composée de quatre domaines d'horloge. La trace bleue représente l'évolution de la fréquence moyenne des quatre domaines en cours d'exécution, tandis que la trace orange représente la moyenne de la trace précédente sur toute l'exécution, ce qui donne un aperçu de la réduction de puissance (dynamique) moyenne consommée en comparant avec la fréquence maximale de 2500 MHz. Le premier profil relate le comportement du système sans stratégie, où la fréquence reste constamment au maximum de 2500 MHz. Les sept profils suivants relatent le comportement du système avec stratégie, où les fréquences utilisées pour le ralentissement des cœurs passent de 2500 à 2500 MHz, de 2500 à 2300 MHz, de 2500 à 2000 MHz, de 2500 à 1800 MHz, de 2500 à 1200 MHz, de 2500 à 800 MHz et de 2500 à 400 MHz. On voit bien sur cette dernière courbe en particulier l'effet plus marqué de la diminution de fréquence, qui intervient à chaque terminaison / reprise d'une tâche de multiplication ligne / colonne. La stratégie *Blocking point* permet de réduire dans tous les cas la fréquence (par rapport au cas normal où la fréquence resterait à 2500 MHz) et montre des gains moyens (estimés sur la trace de fréquence moyenne orange) autour de 28% ((2500 - 1800) / 2500) pour cet exemple. Ces valeurs issues de la simulation sont données à titre indicatif et seront par la suite mieux évaluées par des mesures sur une plateforme réelle.

La Figure 55 examine les temps d'exécution afin de mesurer si les gains précédents ne se font pas au détriment des performances. Les simulations traduisent un passage à l'échelle pour des multiplications de taille 256*256, 512*512 et 1024*1024 en utilisant 2, 4, 8, 16, 32 et 64 domaines d'horloge. Les trois

courbes de gauche relatent le temps d'exécution absolu, celles de droite le facteur de ralentissement qui doit rester proche de 1 pour valider la stratégie. On note clairement ici qu'il existe une zone de validité pour cette stratégie qui correspond à des configurations où il y a beaucoup de cœurs et de parallélisme. Dans les conditions inverses (par exemple, deux domaines d'horloges en bleu foncé), le temps d'exécution augmente et il le fait d'autant plus fortement que la plateforme est ralentie (2500 vers 400 MHz au maximum) et que le nombre de tâches parallèles augmente (de 256*256 à 1024*1024). Cela s'explique par la quantité de communications entre domaines d'horloge différents car ce sont ces opérations inter domaines qui génèrent les attentes les plus longues et peuvent bénéficier d'un ralentissement des cœurs. Donc plus le nombre de domaines augmente, plus la stratégie est efficace sans affecter les performances, ce qui correspond bien aux contraintes des architectures HPC très fortement parallèles.

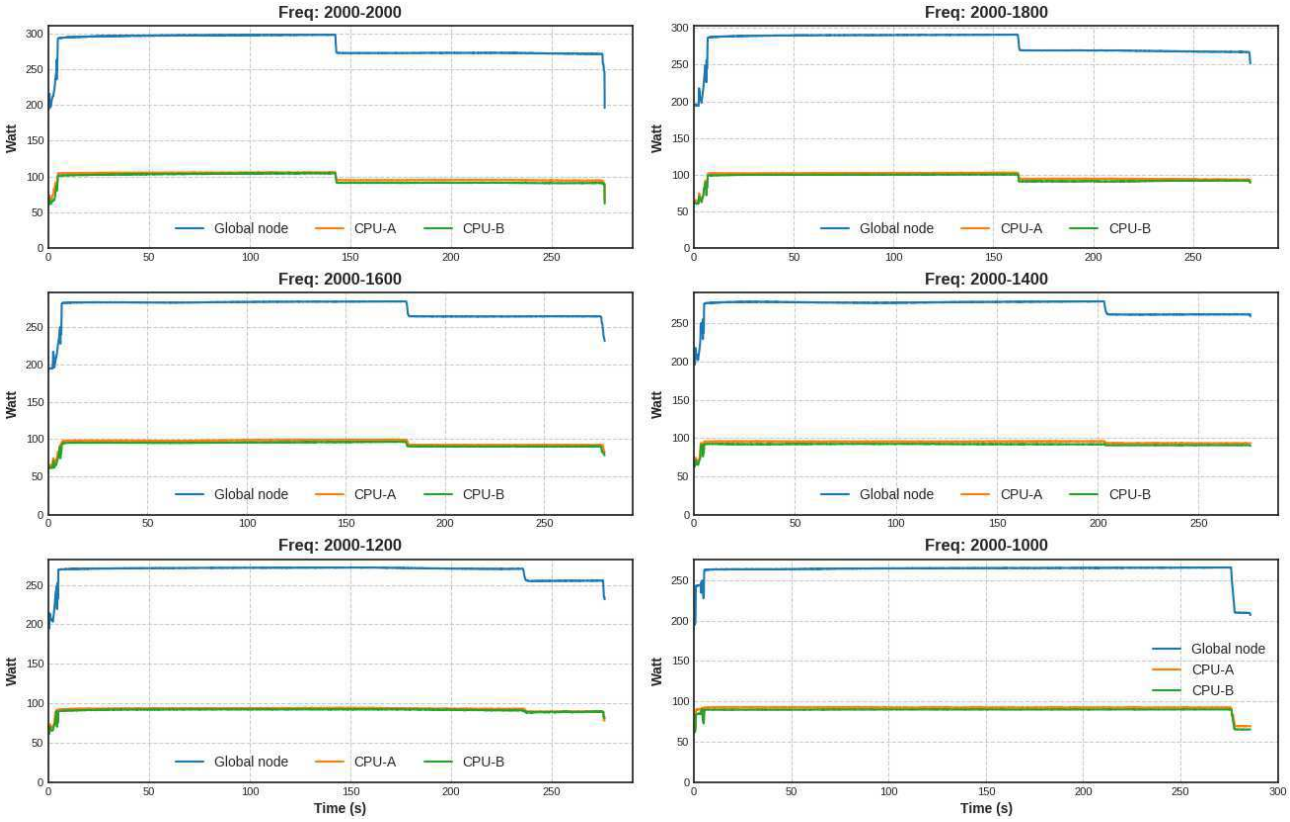


Figure 56 – Profils de puissance mesurés pour la stratégie *soft big.LITTLE* (Cavium ThunderX2)

Enfin, les évaluations suivantes s'attachent à évaluer les deux stratégies HPC proposées sur une plateforme représentative (Cavium ThunderX2) basée sur le dernier démonstrateur Mont-Blanc 3. Cette plateforme dénommée « Dibona » possède deux *compute nodes* qui communiquent via un réseau éthernet (contrôle) et Infiniband Melanox (données). Chaque nœud contient 48 cœurs ARMV8 64-bit sous licence répartis dans les sockets CPU-A (cœurs 0 - 23) et CPU-B (cœurs 24 - 47). Le système permet à chaque cœur de fonctionner aux fréquences de 2000, 1800, 1600, 1400, 1200 et 1000 MHz et les mesures de consommation sont possibles grâce à des capteurs intégrés qui donnent séparément la consommation des nœuds 1 et 2 (courbes oranges et vertes de la Figure 56), ainsi que la puissance globale (courbes bleues).

Les mesures suivantes concernent la stratégie *soft big.LITTLE* sur la plateforme Cavium avec 96 instances de tâches *matrix_duplicate_256_256*. Les 48 tâches de numéro pairs sont allouées sur 2*24 cœurs de numéro pairs des deux *compute nodes*, et les es 48 tâches de numéro impairs (temps d'exécution deux fois

moindre) sont allouées sur les 2*24 autres cœurs impairs.

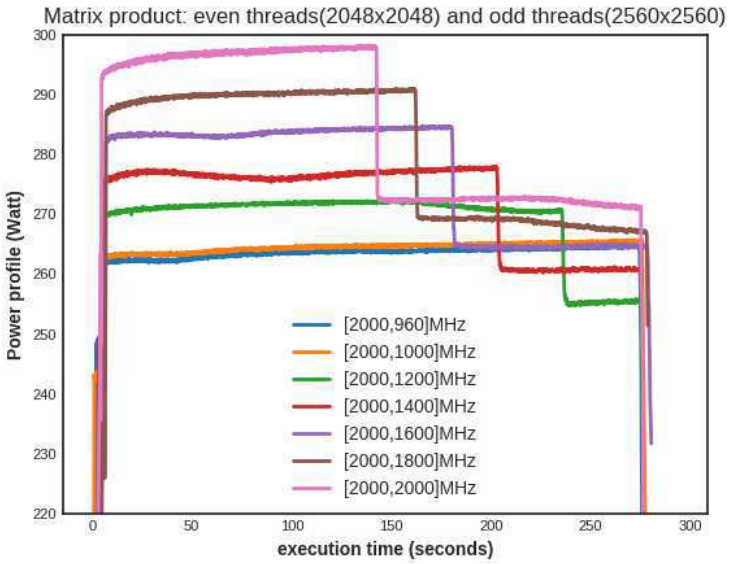


Figure 57 – Profils de puissance globaux pour la stratégie *soft big.LITTLE* (Cavium ThunderX2)

C'est ce que l'on peut observer en haut de la Figure 56 (Freq 2000 - 2000) où les cœurs pairs / impairs sont placés à la même fréquence de 2 GHz pour servir de référence aux prochaines mesures. Le système fonctionne dans ce cas avec une consommation maximale (290W) jusqu'à 145s, puis les cœurs impairs passent en mode *idle* (270W) jusqu'à la fin de la multiplication de matrice à 280s.

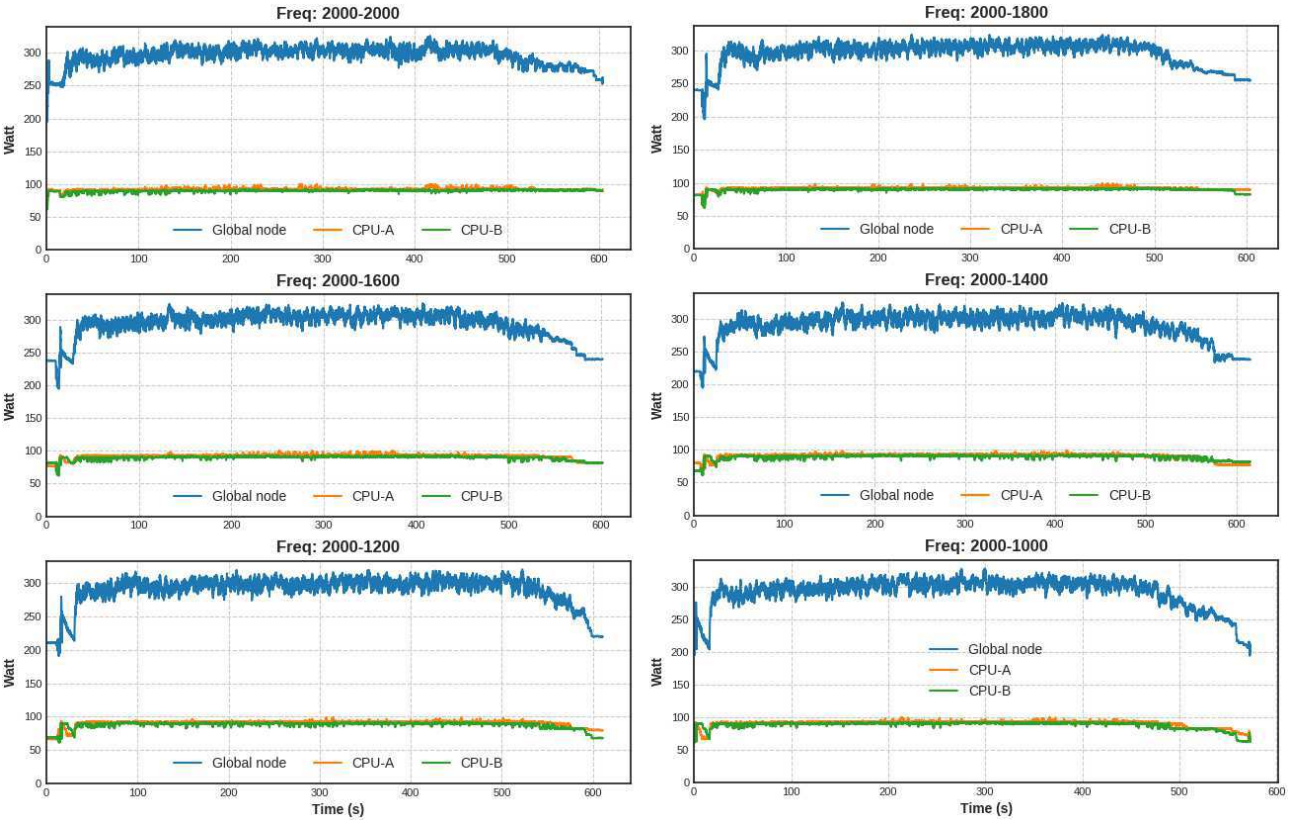


Figure 58 – Profils de puissance mesurés pour la stratégie Blocking point (1 nœud)

Le principe de la stratégie est donc d'exploiter le fait que les tâches impaires n'ont pas ici besoin de s'exécuter à la vitesse maximale. C'est précisément ce qui utilisé dans la stratégie pour ralentir les cœurs

impairs et diminuer la consommation globale tout en conservant le même temps d'exécution global (déterminé par les tâches paires lentes allouées aux cœurs pairs à 2 GHz).

Frequencies	[2000,1000]	[2000,1000]	[2000,1200]	[2000,1200]	[2000,2000]	[2000,2000]
Node ids	#1	#2	#1	#2	#1	#2
Execution time	1255.68	1256.99	1269.09	1260.17	1294.85	1279.82
Global power	241.21	256.44	247.66	259.47	270.72	270.09
CPU-A power	77.19	79.83	78.78	81.91	91.30	90.00
CPU-B power	78.24	83.57	79.83	84.23	92.89	87.50
Memory power	52.96	51.90	52.54	51.79	52.76	51.46
Infiniband power	12.13	12.93	12.13	12.93	12.12	12.94

Tableau 26 – Détail des mesures en puissance pour la stratégie *Blocking point*

Nous avons ensuite appliqué des fréquences décroissantes de 2 GHz à 1 GHz sur les cœurs impairs pour mesurer les gains en puissance possibles. Cela se traduit Figure 56 et Figure 57 par des paires de valeurs indiquant la fréquence CPU de départ et la fréquence secondaire utilisée pour ralentir un cœur. On observe la baisse de performance correspondante pour les tâches rapides impaires dont le temps d'exécution augmente de 180s à 280s (notons que le temps d'exécution global est toujours celui des tâches paires lentes, i.e. 280s). Ces profils de puissance sont étendus à la consommation globale de la plateforme et moyennés sur la durée de l'exécution pour obtenir les gains réalisés par la stratégie (Tableau 26). La Figure 57 montre les profils globaux correspondants pour chaque paire de fréquence appliquée aux cœurs pairs et impairs.

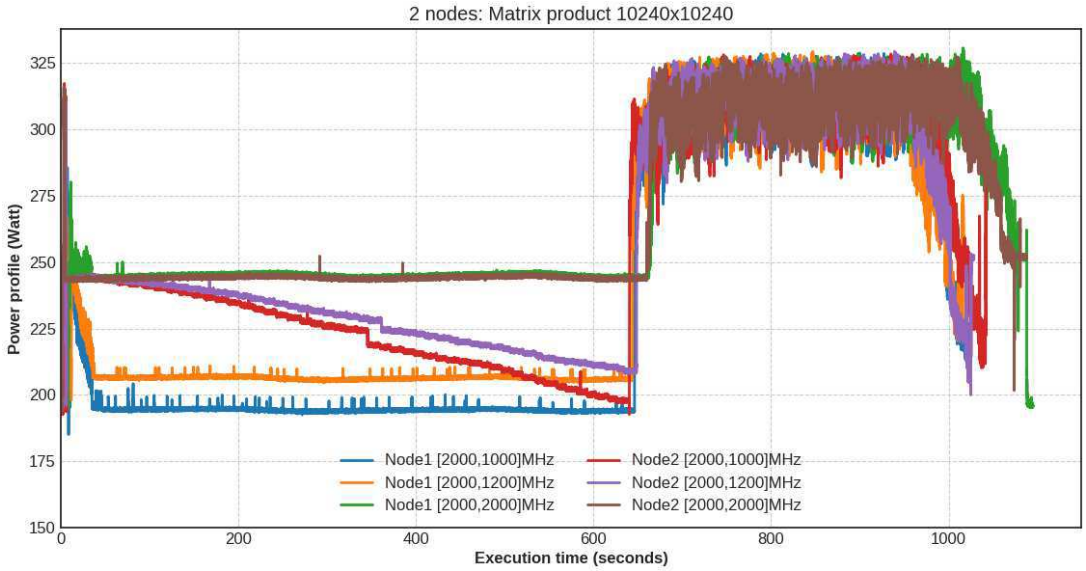


Figure 59– Profils de puissance mesurés pour la stratégie Blocking point (2 nœuds)

On s'intéresse maintenant à la stratégie *Blocking point* sur la même plateforme Cavium avec l'exemple *matrix_slice_10240x10240*. La stratégie exploite dans ce cas les phases de synchronisation (exécution et données) dans le traitement des tâches parallèles. Une première barrière de synchronisation est présente au début pour attendre la fin du processus de création des tâches et démarrer le traitement. Une autre synchronisation se produit à la fin pour attendre la fin de toutes les tâches et écrire le résultat final de la

multiplication en mémoire, avec des attentes potentiellement importantes où les processeurs peuvent être placés à faible fréquence. Dans un premier temps, on analyse l'effet de la stratégie sur un seul nœud (48 cœurs) à la Figure 58. Les profils CPU-A, CPU-B et globaux ne montrent pas d'effets visibles de la stratégie dans ce cas. Cela provient du fait que tous les cœurs au sein du même nœud communiquent à travers des mémoires partagées, les latences de communications sont donc faibles et offrent peu de possibilités au DVFS. Néanmoins, ce n'est pas le cas pour deux nœuds (Figure 59) où les communications inter nœuds sont plus lentes et impliquent des attentes actives beaucoup plus significatives. Une phase de faible activité (0 - 650s) peut être observée pour la synchronisation au démarrage des tâches. Ensuite un pic d'activité où les cœurs passent à 2GHz se produit jusqu'à la terminaison des tâches autour de 1000s, où une barrière de synchronisation intervient à nouveau pour attendre la fin de la dernière tâche et traiter le résultat global de la multiplication. C'est ici que se produit la diminution de fréquence et on peut observer en particulier que la puissance passe de 245W (pour la paire de fréquence 2000 - 2000MHz) à 190W (pour la paire 2000 - 1000MHz), économisant ainsi plus de 20% pour quasiment le même temps d'exécution (moins de 2% d'écart).

6 Conclusion et perspectives

La course aux performances des processeurs, dictée par la loi de Moore depuis les années 1970, a longtemps dominé les exigences de conception des circuits intégrés où les aspects énergétiques pouvaient être considérés comme secondaires. Avec les avancées technologiques continues dans le domaine des semi-conducteurs et notamment celui de l'embarqué depuis 30 ans, il est apparu rapidement que la consommation allait constituer une contrainte prédominante à chaque nouvelle technologie d'intégration silicium. Un nombre considérable d'efforts académiques et industriels ont été réalisés pour apporter des solutions à différents niveaux du problème: technologique, architectural, composant, outils, exécutif, applicatif, etc.

Un bon exemple des tendances en la matière est donné par ARM et Intel. On assiste chez ARM à une hétérogénéisation plutôt "logicielle" (au niveau processeur / cœurs) avec l'essor de la technologie *big.LITTLE* et son extension *DynamIQ* plus récemment, alors qu'Intel semble accorder plus d'intérêt à l'accélération matérielle (coprocesseurs *Xeon* et FPGAs Altera). Le premier objectif de ces innovations est d'améliorer très significativement l'efficacité de traitement des processeurs, c'est-à-dire leur capacité à fournir de meilleures performances en consommant moins d'énergie, ou encore l'efficacité énergétique. Les perspectives en termes de contraintes applicatives convergent toutes dans ce sens: IoT, 5G, HPC (Exascale), Robotique, Intelligence artificielle, ingénierie automobile, etc.

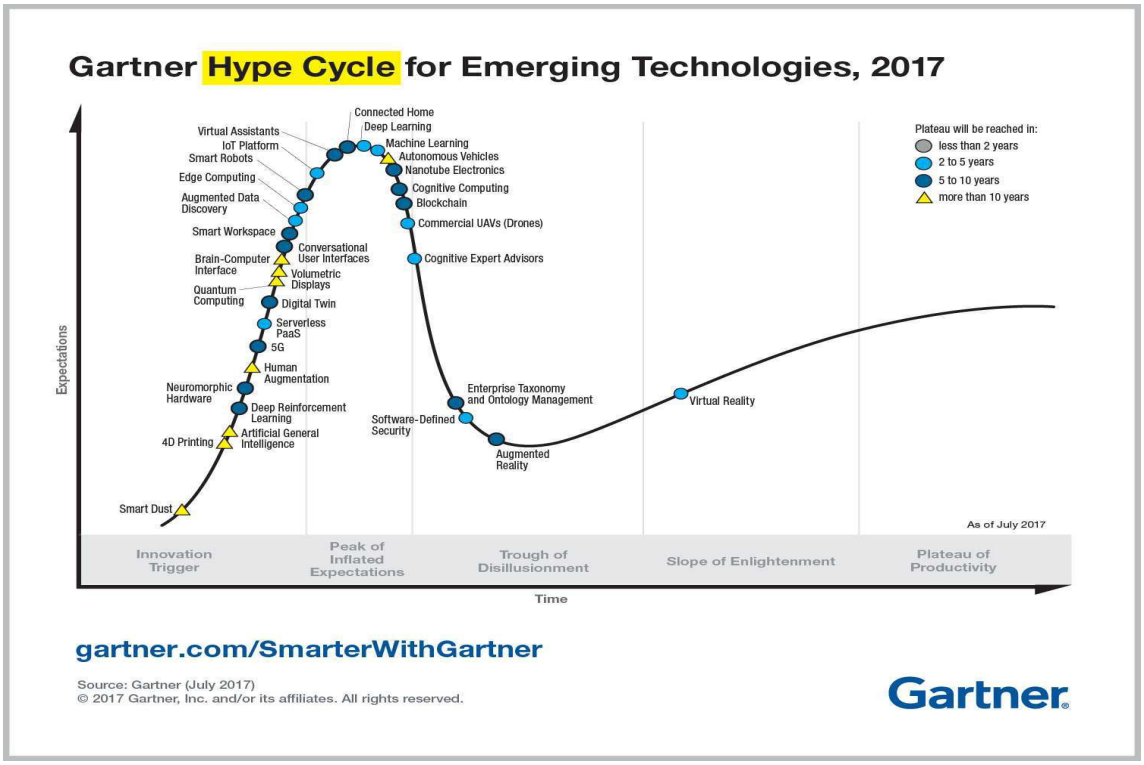


Figure 60 – Estimation de l'évolution de l'intérêt pour les nouvelles technologies (Gartner)

Dans quasiment tous les domaines d'applications à venir, le besoin en efficacité énergétique est fondamental pour que les espoirs technologiques ambitieux (tels que ceux de la Figure 60) soient réalistes, ne serait-ce que d'un point de vue de l'alimentation électrique impliquée par la profusion de systèmes

électroniques connectés et l'infrastructure associée, si ce futur voit effectivement le jour. Pour atteindre un tel niveau d'efficacité (que l'on peut estimer à une amélioration de plus d'un facteur 100 par rapport aux possibilités actuelles), tous les éléments de l'infrastructure devront être fondamentalement conçus pour tirer parti de l'hétérogénéité matérielle et de la complexité applicative de la manière la plus fine et la plus profitable possible. C'est sur cette question essentielle que les différents résultats, qui sont tous tirés d'applications industrielles concrètes, nous orientent. L'exemple des stratégies de gestion énergétique est parlant avec une utilisation en pratique qui est en dessous du potentiel réel, principalement à cause de la complexité inhérente au développement pratique de stratégies de gestion énergétique avancées (e.g. EDF, EDF-DVFS). Des ordonnanceurs faible consommation plus évolués que l'existant sont par exemple possibles en gérant de façon extrêmement fine la consommation au *runtime* et permettre des améliorations jusqu'à 50% (chapitre 4). Par ailleurs, l'étude d'une combinaison efficace de différentes techniques, comme par exemple pour allier véritablement l'hétérogénéité logicielle (au niveau processeur / cœurs, DVFS) et matérielle (accélérateurs matériels, GPU, RDP), est une direction incontournable pour atteindre cet objectif. Sur ce point par exemple, le potentiel de méthodologies système, conçues spécifiquement pour mieux associer des aspects logiciels (stratégies) et matériels (cœurs spécialisés, accélérateurs, RDP) montre des gains en efficacité énergétique de plus d'un facteur dix sur des cas d'étude réels (chapitre 5). Pour qu'un tel résultat soit possible, la prise de décision joue un rôle clé dans l'exploitation de l'hétérogénéité tant sur le plan statique (exploration/déploiement) que dynamique (ordonnancement, DVFS, RDP), et qui repose sur une modélisation globale cohérente, robuste et compatible avec une forte abstraction des plateformes et des applications. Le très important effort de caractérisation énergétique des technologies reconfigurables développé représente une contribution notable dans ce sens, qui permet désormais de beaucoup mieux appréhender le véritable potentiel d'optimisation de la reconfiguration et de la RDP par rapport à une exécution purement logicielle sur des architectures multi-cœurs (chapitre 3), mais aussi par rapport à une accélération matérielle reconfigurable classique sans reconfiguration dynamique (chapitre 5).

Sur la question méthodologique, l'étude et le développement de plusieurs environnements de conception, de modélisation et d'exploration à travers de nombreux projets ambitieux au cours des vingt dernières années nous a conduit à entrouvrir une voie pour l'amélioration de l'efficacité énergétique qui est peu considérée à l'heure actuelle. Elle se base sur une approche coopérative plus intégrée (de la spécification jusqu'au déploiement concret), s'appuie sur une abstraction / modélisation système pour une prise de décision efficace, et s'applique à un niveau plus global (incluant par exemple une gestion énergétique spécialisée). Ainsi les derniers travaux réalisés et publiés sur la méthodologie *FoRTReSS* (chapitre 5) démontrent le potentiel d'une telle approche du problème pour favoriser une meilleure combinaison de techniques existantes à différents niveaux (conception, modélisation, ordonnancement, DVFS, RDP, etc.) et permettre de tenir compte véritablement des interdépendances de plus en plus complexes impliquées par l'hétérogénéité statique et dynamique des SoCs. Un enseignement intéressant est donc qu'il existe un potentiel important dans l'étude de méthodologies qui sont explicitement centrées sur l'efficacité énergétique, par rapport aux approches existantes. Celles-ci se divisent grossièrement en deux catégories: celles qui cherchent d'abord à répondre aux exigences de performances (ou de temps-réel) puis sont étendues pour caractériser la consommation, et celles qui reposent sur de nouvelles techniques et technologies. La profusion de techniques énergétiques cohabitant dans un SoC aujourd'hui pose le problème de leur coopération efficace pour repousser encore les limites des niveaux de gain. Il est cependant assez clair que le niveau d'efficacité extrême requis par les espoirs technologiques (amélioration d'un facteur 100 et plus) ne peut passer que par une combinaison efficace, plus méthodique de plusieurs techniques, et dépend dans l'absolu de progrès sur les trois aspects suivants: de nouvelles méthodologies

système spécifiquement conçues et dédiées à une réduction massive de la consommation, l'amélioration de techniques existantes (e.g. gestion énergétique complexe et avancée) et le développement de nouvelles technologies (e.g. mémoires non volatiles).

Les perspectives portent précisément sur les deux premiers points. Les premiers travaux envisagés par la suite sont donc des efforts à court terme pour consolider la méthodologie [16], en particulier par la confrontation pratique dans les domaines d'applications les plus exigeants en contraintes énergétiques amenés à se développer dans un futur proche (IoT, 5G, HPC, intelligence artificielle, Robotique, ingénierie automobile). Les études applicatives très concrètes privilégiées dans tous ces travaux ont montré par ces nombreux résultats qu'elles représentent une base de données d'une importance cruciale pour avancer concrètement sur des questions de conception et d'analyse au niveau système. Ces nouvelles applications ambitieuses permettront par exemple à terme la définition d'une méthodologie qui arrive à combiner complètement hétérogénéité logicielle (cœurs, DVFS, modes repos, etc.) et accélération matérielle (RDP, parallélisme, GPU, etc.), à un niveau d'analyse système pour appréhender l'énorme complexité matérielle et surtout aborder de manière beaucoup plus méthodique la combinaison de techniques et technologies différentes.

Le deuxième aspect concerne la complexité croissante des SoCs et des applications, et l'amélioration significative des solutions de gestion énergétique dans ces conditions. Une direction qui parait en effet indispensable à étudier aujourd'hui est celle de stratégies exploitant des techniques d'apprentissage et d'intelligence artificielle. La maturité récente des travaux dans ce domaine (*deep learning*, *machine learning*, etc.) ouvre un champ prometteur, mais extrêmement complexe à expérimenter et qui ne peut se mener autrement que sur du long terme. Les contraintes de financement actuelles (à court terme et principalement sur de la R&D industrielle) sont certainement une raison importante pour laquelle très peu de travaux cherchent à explorer ces possibilités. J'ai pour ma part acquis la conviction que toutes les promesses technologiques en vogue actuellement n'ont que peu de chances d'aboutir et d'être viables sans une amélioration énergétique extrême (facteur 100 à 1000), étant donné le volume de systèmes (plus de 20 milliards d'objets connectés à l'horizon 2020 et l'infrastructure associée), l'explosion des communications numériques, le coût énergétique et l'impact environnemental complètement sous-estimés que cela représente. Les perspectives technologiques sont largement optimistes en termes de faisabilité et elles ne pourront voir le jour sans des efforts beaucoup plus importants, plus amont, sur du long terme, et focalisées de manière plus exclusive sur ce défi énergétique fondamental, pour des systèmes électroniques de manière globale (i.e. incluant analogique, RF, puissance signal, récupération d'énergie, etc.). Ces aspects restent à l'heure actuelle traités de manière complètement séparée et/ou secondaire (e.g. méthodologies existantes centrées majoritairement sur les performances, moins sur l'énergie). De manière générale, il est à parier que les espoirs technologiques seront bien en dessous des promesses affichées dans la réalité car ils manquent de soutien et de vision globale pour véritablement alimenter la révolution méthodologique qu'ils nécessitent, compte-tenu des ambitions, et des politiques de recherche actuelles qui ne sont pas adaptées, favorisent une vision partielle, ne mettent pas l'effort nécessaire sur le problème énergétique, se focalisent sur des défis sociétaux, mais aussi des modes de financements de la Recherche et leur volatilisation depuis quelques années.

7 Références

[1] ARM Limited, **Core Tile for ARM11™ MPCore™ User Guide**, http://infocenter.arm.com/, 2005

[2] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, **H.264/MPEG-4 AVC Reference Software Manual of JM latest H.264 Reference Decoder**, October 2005.

[3] M. Fiedler, **Implementation of a basic H.264/AVC Decoder**, Seminar paper, June 2004.

[4] T. Mudge, **Power: A First-class Architectural Design Constraint**, IEEE Computer Society, 34(4), pp. 58–52, 2001.

[5] Xilinx, UG744, **PlanAhead Software Tutorial: Partial Reconfiguration of a Processor Peripheral**, 2011.

[6] R. Bonamy, S. Bilavarn, D. Chillet and O. Sentieys, **Power Consumption Models for the use of Dynamic and Partial Reconfiguration**, Microprocessors and Microsystems, Elsevier, February 2014.

[7] F. Duhem, F. Muller, P. Lorenzini, **FaRM: fast reconfiguration manager for reducing reconfiguration time overhead on FPGA**, Reconfigurable Computing: Architectures, Tools and Applications, Springer, Berlin/ Heidelberg, 2011.

[8] R. Bonamy, H.-M. Pham, S. Pillement, D. Chillet, **UPaRC – Ultra fast Power Aware Reconfiguration Controller**, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.

[9] F. Duhem, F. Muller, R. Bonamy, and S. Bilavarn, **FoRTReSS: A flow for design space exploration of partially reconfigurable systems**, Design Automation for Embedded Systems, Springer, 2015.

[10] S. Bilavarn, M. K. Bhatti, and C. Belleudy, **Method for scheduling with deadline constraints, in particular in linux, carried out in user space**, International Application No PCT/IB2013/059916, 2014.

[11] x265, **x265 HEVC Encoder / H.265 Video codec**, http://x265.org/, 2014.

[12] T.P.K.C. D'huys , S. Momcilovic, F. Pratas, L. Sousa, **Reconfigurable data flow engine for HEVC motion estimation**, IEEE International Conference on Image Processing (ICIP), 2014.

[13] E. Kalali, Y. Adibelli, I. Hamzaoglu, **A high performance and low energy intra prediction hardware for high efficiency video coding**, IEEE 22nd International Conference on Field Programmable Logic and Applications (FPL 2012), 2012.

[14] N.R. Tiago Dias, L. Sousa, **Unified transform architecture for AVC, AVS, VC-1 and HEVC high-performance codecs**, EURASIP Journal on Advances in Signal Processing, 2014.

[15] N. Steffen, Fast artificial neural network library, http://leenissen.dk/fann/wp/, 2013.

[16] Robin Bonamy, Sébastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet and Fabrice Lemonnier, **Energy Efficient Mapping on Manycore with Dynamic and Partial Reconfiguration: Application to a Smart Camera**, International Journal of Circuit Theory and Applications, Wiley, June 2018.

[17] Mont-Blanc, **European Approach Towards Energy Efficient High Performance**, http://montblanc-project.eu/, 2011 – 2018.

[18] J. Wanza, S. Bilavarn, M. De Vries, S. Derradji and C. Belleudy, **Efficiency Modeling and Analysis of 64-bit ARM Compute Nodes for HPC**, Microprocessors and Microsystems, Elsevier, September 2017.

[19] A. Moshovos, G. Memik, B. Falsafi and A. Choudhary, **Jetty: filtering snoops for reduced energy consumption in SMP servers**, Symposium on High-Performance Computer Architecture (HPCA'01), 2001, Nuevo Leone, Mexico.

[20] C. Bienia, S. Kumar, J. P. Singh and K. Li, **The PARSEC Benchmark Suite: Characterization and Architectural Implications**, Parallel Architectures and Compilation Techniques (PACT), 2008, Toronto, Canada.

[21] T. N. Le, **Global power management system for self-powered autonomous wireless sensor node**, PhD Thesis, Université de Rennes 1, 2014.

Publications

Revues internationales avec comité de lecture

Robin Bonamy, Sébastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet and Fabrice Lemonnier, **Energy Efficient Mapping on Manycore with Dynamic and Partial Reconfiguration: Application to a Smart Camera**, International Journal of Circuit Theory and Applications, Wiley, June 2018.

Joel Wanza, Sébastien Bilavarn, Maarten De Vries, Said Derradji and Cécile Belleudy, **Efficiency Modeling and Analysis of 64-bit ARM Compute Nodes for HPC**, Microprocessors and Microsystems, Elsevier, September 2017.

Robin Bonamy, Sébastien Bilavarn, Daniel Chillet and Olivier Sentieys, **Power Modeling and Exploration of Dynamic and Partially Reconfigurable Systems**, Journal of Low Power Electronics, American Scientific Publishers, September 2016.

François Duhem, Fabrice Muller, Robin Bonamy and Sébastien Bilavarn, **FoRTReSS: a Flow for Design Space Exploration of Partially Reconfigurable Systems**, Design Automation of Embedded Systems, Springer, February 2015.

Robin Bonamy, Sébastien Bilavarn, Daniel Chillet and Olivier Sentieys, **Power Consumption Models for the use of Dynamic and Partial Reconfiguration**, Microprocessors and Microsystems, Elsevier, February 2014.

Sébastien Bilavarn, Jabran Khan Jadoon, Cécile Belleudy and Muhammad Khurram Bhatti, **Effectiveness of Power Strategies for Video Applications: a Practical Study**, Journal of Real-Time Image Processing, Springer, January 2014.

Taheni Damak, Imen Werda, Sébastien Bilavarn and Nouri Masmoudi, **Fast Prototyping H.264 Deblocking Filter using ESL Tools**, Transactions on Systems, Signals & Devices, Shaker Verlag, Vol. 8, No 3, pp.345-362, December 2013.

Rym Cheour, Sébastien Bilavarn and Mohamed Abid, **Exploitation of EDF Scheduling in Wireless Sensor Networks**, International Journal on Measurement Technologies and Instrumentation Engineering, 2011.

Dominique Blouin, Daniel Chillet, Eric Senn, Sébastien Bilavarn, Robin Bonamy, and Christian Samoyeau, **AADL Extension to Model Classical FPGA and FPGA Embedded within a SoC**, International Journal of Reconfigurable Computing, Hindawi, 2010.

Fateh Boutekkouk, Mohammed Benmohammed, Sébastien Bilavarn and Michel Auguin, **UML2.0 Profiles for Embedded Systems and Systems on a Chip (SoCs)**, Journal of Object Technology , january-february 2009.

Fateh Boutekkouk, Mohammed Benmohammed, Sébastien Bilavarn and Michel Auguin, **UML for Modelling and Performance Estimation of Embedded Systems**, Journal of Object Technology, march-april 2009.

Fateh Boutekkouk, Sébastien Bilavarn, Michel Auguin and Mohammed Benmohammed, **Rewriting Logic Semantics for SystemC Scheduler**, International Review on Computers and Software, march 2009.

Jean-Philippe Diguet, Guy Gogniat, Jean-Luc Philipp e, Yannick Le Moullec, Sébastien Bilavarn, Christian Gamrat, Karim Ben Chehida, Michel Auguin, Xavier Fornari, Anne-Marie Fouillart and Philippe Kajfasz, **EPICURE: A Partitioning and Co-design Framework for Reconfigurable Computing**, Microprocessors and Microsystems, Elsevier, 2006.

Sébastien Bilavarn, Guy Gogniat, Jean-Luc Philippe and Lilian Bossuet, **Design Space Pruning through early Estimations of Area / Delay Trade-offs for FPGA Implementations**, IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems, 2006.

Sébastien Bilavarn, Eric Debes, Pierre Vandergheynst and Jean-Philippe Diguet, **Processor Enhancements for Media Streaming Applications**, Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology , Springer, September 2005.

Revues nationales avec comité de lecture

Sébastien Bilavarn, Thibault Dupont, Cécile Belleudy, Michel Auguin and Anne-Marie Fouillart, **Implantation d'un décodeur H.264 sur plateforme Multiprocesseur avec Gestion Energétique**, Revue Technique et Science Informatiques, 2010.

Brevets

Sébastien Bilavarn, Muhammad Khurram Bhatti and Cécile Belleudy, **Method for scheduling with deadline constraints, in particular in linux, carried out in user space**, International Patent Application N° PCT/IB2013/059916, 15 may 2014.

Chapitres de livres

Cécile Belleudy and Sébastien Bilavarn, **Power Models and Strategies for Multi-processor Platforms**, Design Technology for Heterogeneous Embedded Systems, G. Nicolescu, I. O'Connor, C. Piguet, Springer, 2012.

Conférences internationales avec comité de lecture

Joel Wanza, Sébastien Bilavarn, Said Derradji, Cécile Belleudy and Sylvie Lesmanne, **Efficiency Modeling and Analysis of 64-bit ARM Clusters for HPC**, In Proceedings of Euromicro Conference on Digital System Design, DSD 2016, Limassol Cyprus, September 2016.

Robin Bonamy, Sébastien Bilavarn and Fabrice Muller, **An Energy-Aware Scheduler for Dynamically Reconfigurable Multi-Core Systems**, In Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2015, Bremen, Germany, 29/06-01/07 2015.

Taheni Damak, Sébastien Bilavarn and Nouri Massmoudi, **HLS based design of a mixed architecture for H.264/AVC CAVLD**, In Proceedings of the 12th International Multi-Conference on Systems, Signals and Devices Program, SSD 2015, Mahdia, Tunisia, March 16 - 19, 2015.

Jabran Khan Jadoon, Sébastien Bilavarn, Muhammad Khurram Bhatti and Cécile Belleudy, **Energy Analysis of a Real-Time Multiprocessor Control of Idle States on ARM Platforms**, In Proceedings of the 3rd International Conference on Pervasive and Embedded Computing and Communication Systems, PECCS 2013, Barcelona Spain, 2013.

Robin Bonamy, Daniel Chillet, Sébastien Bilavarn and Olivier Sentieys, **Power Consumption Model for Partial Dynamic Reconfiguration**, In Proceedings of the International Conference on ReConFigurable Computing and FPGA, RECONFIG 2012, Cancun Mexico, December 2012.

Daniel Chillet, Eric Senn, Olivier Zendra, Cécile Belleudy, Sébastien Bilavarn, Rabbie Ben Atitallah, Christian Samoyeau and Agnès Fritsch, **Open-PEOPLE: Open Power and Energy Optimization Platform and Estimator**, In Proceedings of the 15th Euromicro Conference on Digital System Design, DSD 2012, Cesme Izmir Turkey, September 2012.

Jabran Khan Jadoon, Sébastien Bilavarn and Cécile Belleudy, **Energy Analysis of a DVFS Power Strategy on ARM Platforms**, In Proceedings of the 11th IEEE Faible Tension Faible Consommation, FTFC 2012, Paris France, June 2012.

Jabran Khan Jadoon, Sébastien Bilavarn and Cécile Belleudy, **Impact of Operating Points on DVFS Power Management**, In Proceedings of the 7th International conference on Design & Technology of Integrated Systems in nanoscale era, DTIS 2012, Gammarth Tunisia, May 2012.

Sébastien Bilavarn, Andrea Castagnetti and Laurent Rodriguez, **A Video Monitoring Application for Wireless Sensor Networks using IEEE 802.15.4**, In Proceedings of the 2nd Workshop on Ultra-Low Power Sensor Networks, WUPS 2011, Como Italy, November 2011.

Bassem Ouni, Cécile Belleudy, Sébastien Bilavarn and Eric Senn, **Embedded Operating Systems Energy Overhead**, In Proceedings of the International Conference on Design & Architectures for Signal & Image Processing, DASIP 2011, Tampere Finland, November 2011.

Robin Bonamy, Daniel Chillet, Olivier Sentieys and Sébastien Bilavarn, **Towards a Power and Energy Efficient Use of Partial Dynamic Reconfiguration**, In Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip , ReCoSoC 2011, Montpellier France, June 2011.

Robin Bonamy, Daniel Chillet, Sébastien Bilavarn and Olivier Sentieys, **Parallelism Level Impact on Energy Consumption in Reconfigurable Devices**, In Proceedings of the 2nd International Workshop on Highly Efficient Accelerators and Reconfigurable Technology, HEART 2011, London UK, June 2011.

Dominique Blouin, Eric Senn, Robin Bonamy, Daniel Chillet, Sébastien Bilavarn and Christian Samoyeau, **FPGA Modeling for SoC Design Exploration**, In Proceedings of the 2nd International Workshop on Highly Efficient Accelerators and Reconfigurable Technology, HEART 20 11, London UK, June 2011.

Taheni Damak, Nouri Masmoudi, Sébastien Bilavarn, **Fast Prototyping H.264 Deblocking filter using ESL tools**, In Proceedings of the 8th International Multi-Conference on Systems, Signals & Devices (SSD'11), Conference on Communication & Signal Processing, Mar 2011, Sousse, Tunisia.

Andrea Castagnetti, Cécile Belleudy, Sébastien Bilavarn and Michel Auguin, **Power Consumption Modeling for DVFS Exploitation**, In Proceedings of Euromicro Conference on Digital System Design, DSD 2010, Lille France, September 2010.

Rym Cheour, Sébastien Bilavarn and Mohammed Abid, **EDF Scheduler Technique for Wireless Sensor Networks: Case Study**, In Proceedings of the 4th International Conference on Sensing Technology, ICST 2010, Lecce Italy, January 2010.

Fateh Boutekkouk, Sébastien Bilavarn, Michel Auguin and Mohammed Benmohammed, **UML Profile for Estimating Application Worst Case Execution Time on System-on-Chip**, In Proceedings of the International Symposium on System-on-Chip, SOC 2008, Tampere Finland, November 2008.

Sébastien Bilavarn, Cécile Belleudy, Michel Auguin, Thibault Dupont and Anne-Marie Fouillart, **Embedded Multicore Implementation o f a H.264 Decoder with Power Management Considerations**, In Proceedings of the 11th Euromicro Conference on Digital System Design, DSD 2008, Parma Italy, September 2008.

Yannick Le Moullec, Søren Skovgaard Christensen, Wen Chenpeng, Peter Koch and Sébastien Bilavarn, **Fast Prototyping of Reconfigurable Architectures from a C Program**, In Proceedings of the 5th International Conference on Information, Communications and Signal Processing, ISICS 2005, Bangkok Thailand, December 2005.

Yannick Le Moullec, Søren Skovgaard Christensen, Wen Chenpeng, Peter Koch and Sébastien Bilavarn, **Fast System-Level Design of Wireless Applications**, In Proceedings of the 8th Wireless Personal Multimedia Communications, WPMC 2005, Aalborg Denmark, September 2005.

Sébastien Bilavarn, Jean-Philippe Diguet, Eric Debes and Pierre Vandergheynst, **Reconfigurable Coprocessor for Media Streaming**, In Proceedings of the 2004 IEEE International Conference on Multimedia and Expo, ICME 2004, Taipei Taiwan, June 2004.

Sébastien Bilavarn, Guy Gogniat, Jean-Luc Philippe and Lilian Bossuet, **Fast Prototyping of Reconfigurable Architectures from a C Program**, In Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, ISCAS 2003, Bangkok Thailand, May 2003.

Sébastien Bilavarn, Guy Gogniat, Jean-Luc Philippe and Lilian Bossuet, **Fast Prototyping of Reconfigurable Architectures: An Estimation and Exploration Methodology from System-Level Specifications**, In Proceedings of the 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays, FPGA 2003, Monterey CA USA, February 2003.

Sébastien Bilavarn, Guy Gogniat and Jean-Luc Philippe, **FPGA Area Time Power Estimation for DSP Applications**, In Proceedings of the International Conference on Signal Processing Applications and Technologies, ICSPAT 2000, Dallas TX USA, November 2000.

Sébastien Bilavarn, Guy Gogniat and Jean-Luc Philippe, **Area Time Power Estimation for FPGA Based Designs at a Behavioral Level**, In Proceedings of the 7th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2000, Beirut Lebanon, December 2000.

Sébastien Bilavarn, Guy Gogniat and Jean-Luc Philippe, **A hardware Software Co-design Methodology for Heterogeneous Architecture Estimation**, In Proceedings of the International Conference on Signal Processing Applications and Technologies, ICSPAT 1999, Orlando FL USA, November 1999.

Conférences nationales avec comité de lecture

Sébastien Bilavarn, Thibault Dupont, Nabil Mounir, Cécile Belleudy, Michel Auguin et Anne-Marie Fouillart, **Une Analyse de Performances et de Consommation du Décodage H.264 sur ARM MPCore**, SYMPosium en Architectures nouvelles de machines, SympA 2008, Fribourg Suisse, Février 2008.

Sébastien Bilavarn, Guy Gogniat et Jean-Luc Philippe, **Estimation de Performances à un Niveau Comportemental pour l'Implantation sur Composants FPGAs**, 7ème SYMPosium en Architectures nouvelles de machines, SYMPA7, Paris France, Avril 2001.

Sébastien Bilavarn, Guy Gogniat et Jean-Luc Philippe, **Méthode de Conception d'Architectures Hétérogènes pour les Applications de Traitement Numérique du Signal**, 3ème Journées Nationales du Réseau Doctoral de Microélectronique, JNRDM 2000, Montpellier France, Mai 2000.

Sébastien Bilavarn, Guy Gogniat et Jean-Luc Philippe, **Estimation d'Architectures Hétérogènes pour les Applications de Traitement Numérique du Signal**, 2ème Colloque du GDR CAO de Circuits Intégrés et Systèmes, Aix-en-Provence France, Mai 1999.

Conférences nationales sans comité de lecture

Sébastien Bilavarn, **Linux Embarqué - Développement de pilotes périphériques**, 13èmes journées pédagogiques du CNFM, Saint-Malo, France, Nov 2014.

Thèse

Sébastien Bilavarn, **Exploration Architecturale au Niveau Comportemental – Application aux FPGAs**, Université de Bretagne Sud, 28 Février 2002.

Logiciels

Lab-STICC, **Design Trotter**, https://designtrotter.univ-ubs.fr/

LEAT, **FoRTReSS**, http://fortress-toolbox.unice.fr/

Résumé / abstract

Les enjeux d'une meilleure efficacité énergétique sont un moteur essentiel de l'évolution des technologies semi-conducteur depuis la popularisation des systèmes répartis, nomades et embarqués. L'exemple le plus parlant est celui des téléphones portables ces vingt dernières années et les perspectives technologiques (IoT, 5G, HPC, intelligence artificielle, Robotique, ingénierie automobile) renforcent encore plus la nécessité de la maîtrise de la consommation d'énergie. Les travaux de recherche qui ont conduit à cette HDR abordent cette question par l'étude d'outils et de méthodes de conception des systèmes numériques en se focalisant sur les contraintes énergétiques et les technologies embarquées. En particulier, les recherches effectuées ces quinze dernières années interviennent toutes au sein de projets collaboratifs ambitieux avec des industriels représentatifs du domaine (Intel, Thales, STMicroelectronics, etc.) sur les aspects énergétiques. Malgré les difficultés à poursuivre aujourd'hui une étude fondamentale sur la durée dans un contexte global de recherche appliquée (projets collaboratifs financés à trois ou quatre ans sur des enjeux fortement influencés par la R&D industrielle), c'est une démarche délibérée visant à intégrer les perspectives à court terme des appels d'offre dans un horizon scientifique plus large qui a permis d'identifier et d'approfondir des idées originales sur des aspects plus fondamentaux. Ces éléments d'amélioration ont conduit à envisager une méthodologie centrée sur l'efficacité énergétique qui reflète un concept de conception intégrée avec des études de cas montrant des améliorations significatives sur des démonstrateurs concrets.

Next level of energy efficiency is an essential concern in the evolution of semiconductor technologies since the widespread development of distributed, nomad and embedded systems. The best example is that of mobile phones the last twenty years but consumer electronics perspectives (IoT, 5G, HPC, Artificial Intelligence, Robotics, automotive engineering) reinforce the need for further power consumption control. The research carried out for this habilitation defense addresses this question by investigating tools and methodologies for digital system design focusing on energy constraints and embedded technologies. In particular, all the investigations developed over the last fifteen years are part of ambitious collaborative projects with representative industrial partners (Intel, Thales, STMicroelectronics, etc.) on different aspects of energy concerns. Despite the difficulty faced today to proceed with a fundamental study on the long run in a global context of applied research (collaborative projects funded for 3 or 4 years with objectives highly driven by industrial R&D), it is a deliberate approach aimed at setting the short-term views of project proposals in a broader scientific horizon which has made it possible to identify and develop original ideas on more fundamental questions. These elements of improvement have led to propose a methodology centered on energy efficiency reflecting a concept of integrated design methodology with case studies reporting significant improvements on concrete demonstrators.

Annexe

Chapter 21

POWER MODELS AND STRATEGIES FOR MULTIPROCESSOR PLATFORMS

Cécile BELLEUDY

Sébastien BILAVARN
University of Nice-Sophia Antipolis, LEAT- CNRS, Bat.4, 250 rue albert Einstein 06560 Valbonne - France {belleudybilavarn}@unice.fr

Abstract: Emerging applications in the field of multimedia like new video standards require the capabilities of multiprocessor architectures which are a high source of heat dissipation. In this chapter, we present the definition of a power model suited for multiprocessor power management based on the study of a H264/AVC decoder implementation. We will provide a detailed analysis of multiprocessor execution including the influence of several possible operating points of frequency and voltage. These results will be used to propose a power strategy suited to video processing. Since the consumption of the main memory becomes more and more considerable, we also address the impact of the memory architecture on the energy cost.

Key words: multiprocessor platform, energy model, energy aware strategies, H.264 decoder

1. INTRODUCTION

Multiprocessor systems are a promising solution to answer the need for more processing power, memory and network bandwidth in future mobile applications. At the same time, this solution implies high power dissipation and one of the key challenges is to control the power consumption, especially for embedded systems where a critical constraint is to extend the battery lifetime of a device. There are other goals which may also need to be considered, some of which could be to limit the cooling requirements or to

reduce the financial cost. But finding ways to reduce the energy consumption becomes crucial because of power-related environmental concerns. Each year, approximately 160,000 tons of portable (consumer) batteries are sold in the European Union. The portable battery market consists of general purpose batteries, button cells and rechargeable batteries. These batteries contain metals, which may pollute the environment at the end of their life. Reducing the power consumption limits the pollution.. The problem of managing the energy consumed by electronic systems can be addressed at different levels of integrated systems development, ranging from technological level (circuits, architectures) to application and system software capable of adapting to the available energy source. Many research and industrial efforts are currently underway to develop energy-aware scheduling as well as power services in operating systems. In the literature, many works describe the energy behavior from theoretical studies or from (essentially) monoprocessor measurement results [1]. In the following, we will experiment with multiprocessor platforms in various possible configurations.

Increasingly powerful SOC designs require ever-larger main memory, which leads to a heavy increase of the energy consumption. In particular this can be a limiting factor in many embedded system designs. For future technologies, the static power will become the dominant factor of power consumption for the off-chip memory. In order to solve this issue, an architectural solution is to adopt a multi-bank memory system instead of a monolithic (single-bank) memory system, such as the RAMBUS-DRAM (RDRAM) technology [2] or the Mobile-RAM (SDRAM) technology [3]. In this case, significant power reductions are expected with the help of an efficient policy to manage the low-power modes of the different banks (Standby, Nap, Power-Down). In these modes, the power supply of the different parts of line and row decoders are switched off progressively. In the last low-power mode, the self-refresh mode is disabled and the content of the memory is lost.

In this chapter we propose to analyze the impacts on energy considering the application allocation in the multibank memory. We will also present experiments and power measurements obtained from executing a H264/AVC decoder on a multiprocessor platform in different configurations. We will discuss an efficient exploitation of the results for the definition of power-aware strategies controlled by the operating system. Two problems will be addressed: the power consumption of processors and the one of the main memory. The remainder of this chapter is organized as follows. We survey related work in section 2. An overview of our power experimentation and detailed analysis of measures are provided in section 3. A power model of the main memory is exposed in section 4 with a quantitative study of the

energy variation according to the memory mapping of the application. Section 5 then concludes the chapter.

2. RELATED WORK

To date, energy management in a processor relies mainly on two mechanisms: DPM (Dynamic Power Management) to switch off the power supply of a part of the circuit and DVFS (Dynamic Voltage and Frequency Scaling) to tune a processor clock speed and its corresponding voltage according to requirements such as the workload (actual or expected) or the battery charge. The management of these mechanisms through an operating system requires software that must be able, first, to identify the various operating points of the processor(s) and second, to assess requirements derived from the application and its environmental context.

The longest-standing approaches were developed to extend the autonomic capacity of laptops. They rely on DPM (e.g. ACPI in Linux) to manage the power saving modes available efficiently. The number of modes can actually range from 1 to 6 according to the processor family, and power consumption in each mode can be divided by a factor from 3 to 1000. The corresponding wakeup latencies vary from microseconds to milliseconds. In these approaches [4][5], the processor is modeled by a state graph where nodes represent the operating points (processor + peripherals) and edges are the transition conditions with their associated penalties (for example, time and energy overheads). A user can thus define its own energy management policy by monitoring application parameters on the processor like the number of instructions executed on a time window. The application can also force some execution requirements by constraints such as the bit rate or the resources needed. This is also applicable for environmental parameters like the battery level, which can impose limitations on the power consumption or on the set of minimum functionalities to maintain.

In embedded systems, reducing energy is one of the prime concerns. We can find several initiatives developed by processor manufacturers and dedicated to families of embedded architectures. For example, ARM [6] defined a technique called IEM (Intelligent Energy Manager), the role of which is to handle system configuration according to the actual and/or predicted workload. The IEM works together with a hardware unit called IEC (Intelligent Energy Controller) to retrieve information for a running period and select suited power modes using the two mechanisms DPM and DVFS. Intel [7] proposed a similar technology called SpeedStep that was integrated into the XScale family of Intel PXA27x [8]. These initiatives also have in common that they target monoprocessor platforms.

Another important field of investigation addresses strategies for low-power scheduling. The first techniques based on DPM proposed scheduling policies for sets of tasks (dependent or independent) given an application domain. The goal was to optimize CPU idle times and to reduce the number of processors returning to active modes. Many contributions exist for single processors, and some of them have been extended to multiprocessor systems. Most approaches trace the history of task executions to predict the idle periods and turn hardware to a low-power state. For example, in [9], the authors use a regressive analysis on the running tracks. [10][11] use Markov chains: processor execution is modeled by a state graph where the probabilities of mode transitions are revalued according to the actual execution of tasks. In practice, these approaches are difficult to implement in real time systems. To bring a solution, some approaches like [12] introduce the mobility of tasks (time separating the earliest and latest dates of execution) to obtain a better sequencing of the tasks on the same processor.

The majority of these approaches do not consider all the power modes available or the energy overhead due to the wakeup of processors. We can however note that such techniques based on DPM are increasingly used today to solve thermal dissipation problems [13]. Nowadays, the use of low-power or sleep modes is often associated with techniques for the dynamic control of processor speed (DVFS). Such techniques have a very significant potential for reducing power. Some previous works take place in a purely static context [14][15][16][17]: the number of processors in active mode, as well as the operating frequency of processors, are set once and before execution, on the basis of the worst case execution time (WCET). Dynamic approaches [18] start with a similar static analysis based on WCET, and carry out a re-evaluation of the frequency at runtime by reconsidering the effective execution time of the tasks. Finally, some other approaches are essentially defined in a dynamic (on-line) context without preliminary information on the system [19][20].

A majority of these works have been developed for homogeneous platforms where the clock speed is controlled either globally for all processors (chip_wide) [20], either locally by processor [14][15][16]. Heterogeneity in more realistic systems can be seen as a set of processors with DVFS or not [20], as a set of processors that can operate at different speeds, or as a set of different types of processors resulting in varying task execution time [21]. A power-aware strategy can thus have different objectives, for example minimizing the total energy consumption resulting from the execution of a task set [15][17], or reducing the peak power consumption leading to important solicitations of the system battery [14][22].

Among these strategies, some are said to be "global" [19]: all tasks can be allocated to different processors allowing a comprehensive management of energy consumption, which is optimal if the number of task migrations is not too high. The issue for real time applications and multiprocessor systems is that the global schedulability test is only proved for a workload of approximately *m/2* for *m* processors [23]. This sub-optimality leads to the development of approaches said to be "local" [15]: task allocation is performed statically, and each processor controls its running speed or low-power mode according to its own workload. Finally, in mixed approaches [24], some of the tasks are allocated to processors, while the other tasks are distributed over all the processors depending on their workload. Only these latter tasks are allowed to migrate.

In many approaches, only the dynamic power consumption is considered. This assumption is very restrictive especially for technologies below 90nm. The authors of [16][21] introduce static power consumption in their model. The processor speed is then calculated considering the two sources of energy dissipation. The authors show that, because of the static power consumption, some operating points, depending on the allocated time slots, do not gain more energy. Moreover, it should be noticed that the penalties due to the switch of voltage-frequency couples are very rarely considered in the majority of these theoretical works. However, on some platforms like the IMX31 and PXA27x, transition times of some milliseconds are observed. This setting cannot be neglected. A set of non-experimental works have been presented so far. The majority of these works assesses the power savings by using a simulation platform or by developing an *ad hoc* evaluation based on a theoretical model of the power consumption.

Another critical factor impacting power is the memory. Consumption of the main memory is becoming considerable and it is expected to increase for future hardware. Some approaches associate the power management of processors with the power modes of the main memory. Several techniques exploiting multibank memory architectures with low-power mode management were proposed. From access data pattern analysis, they try to determine when to power down and into which mode it is possible to switch the memory banks. These memory controller policies can be compiler-based data mapping [25][26][27], hardware-assisted [28], or operating system oriented [2][29].

At the compiler level, [25] studied the impact of loop transformations on banked memory architectures. In [30], it was proposed to keep data blocks in the on-chip memory space in a compressed form. The compiler's job is to analyze the application code and extract information on data access patterns to determine the data blocks to compress/decompress every time. In [27], the authors proposed to perform extra computations if doing so makes it

unnecessary to reactivate a bank which is in the low-power operating mode. Most of these approaches operate on data, exploit one low-power mode of the memory and predict the memory bank usage at compile time for the target application. In [26], the presented approach automatically places data elements that are likely to be accessed simultaneously in the same memory bank by tuned data migration based on the temporal affinity of data. For hardware-assisted techniques [30], the self-monitored hardware transits automatically banks to low-power modes based on the information collected by the supporting hardware. These techniques showed better performance than the compiler based approach but they are not flexible and need extra hardware, which in itself consumes energy.

The operating system based approach has the advantage of a global view of the system, without introducing any performance or energy overhead. Lebeck, et al. [31] described a scheme for reducing DRAM energy by power aware page allocation algorithm. Delaluz, et al. [32] proposed a scheduler-based approach. They use a bank usage table (BUT) which is managed by the operating system, and reset all banks' power modes at the context switches. The BUT gives the bank usage information at the previous time a process was scheduled. This information helps in selecting which banks to set to a low-power mode when the scheduler picks this process the next time. This approach cannot take into account the energy-aware policy applied to the processor, such as the frequency variations.

To reduce the power consumption of the main memory, we propose a new approach based on the task-bank locality in SMP multiprocessor architectures and which is complementary to the energy-aware scheduling policy.

Firstly, in the following section, power characterization of a multiprocessor platform is described according to different operating points and a proposal for power management strategy is given.

3. DESIGN OF A MULTIPROCESSOR POWER MANAGEMENT STRATEGY

3.1 Development environment

We present in the following the design and evaluation of a strategy based on DVFS (Dynamic Voltage and Frequency Scaling) for multiprocessor power management. The application domain is in the field of video processing and the platform used for all developments is the CT11 MPCore from ARM. In a first step, we analyze in detail the performance, power and

energy consumption of a representative video application, namely H.264/AVC decoding, in different execution conditions: platform configuration (number of CPUs activated, voltage/frequency), workload (number of threads of the decoder) and input data (video sequences). The next section describes the experimental procedure on which this characterization, and power measurement in particular, is based.

3.1.1 Platform description

The multiprocessor development platform is composed of two parts: the multicore platform itself which is an ARM MPCore test chip, and an Emulation Baseboard on top of which it is connected. The Emulation Baseboard is in charge of the power supply, memory system, bus control (AMBA AXI) and peripherals. The MPCore test chip is composed of 4 ARM11 processors, each having 32KB instruction memory and 32KB data memory. The system includes hardware support for memory system coherence with the unified L2 cache (1MB). For our needs, the MPCore is used in symmetric mode under the control of Linux SMP, but it can also operate in asymmetric mode (AMP). Adaptive shutdown of unused processors is supported as well as dynamic voltage scaling. But concerning frequency switching, it is only supported statically with the Emulation Baseboard. All these features are controlled by hardware resources that are detailed in the following.

Most development boards allow power measurements using standard probes which are a convenient way to provide fast global values, but in our case another solution can be used. Specific built-in registers labeled $SYS\_VOLTAGE_x$ exist in the MPCore to perform direct sampling of the voltage and current of the chip or the PLL [33]. By writing into these registers, it is also possible to force the supply voltage of the cores.

Register $SYS\_VOLTAGE_0$ has two fields DAC_DATA and ADC_DATA that are used respectively to force and read the voltage of the cores. DAC_DATA is an 8-bit field that allows specifying one of 256 possible values ranging from 0.95 to 1.45V. This value is sent to the digital to analog converter and sets the supply voltage of the four cores.

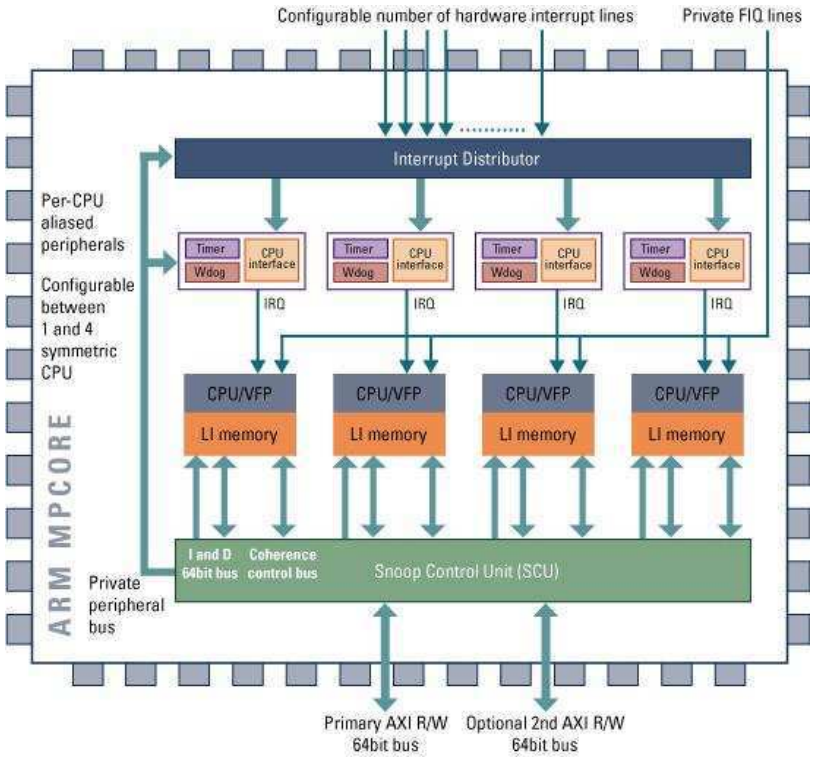


Figure 1. ARM11 MPCore processor

The other ADC_DATA field, of 12 bits, returns the voltage of the cores. Register SYS_VOLTAGE2 operates like SYS_VOLTAGE0 except that it has read-only accessibility. Its purpose is to return a voltage measure from a resistor network of 0.025 Ohms in a way to indicate the supply current of the CPUs, and thus to derive the corresponding instantaneous power consumption of the MPCore. Finally, we also used another register called SYS_PLD_INIT, to process a static modification of the processor frequency. This register sets the initialization of two parameters: the Test Chip PLL Control register and the Test Chip Clock divider register. A reset of the emulation baseboard makes the change effective in SYS_PLD_INIT. The above registers let us set the voltage and frequency of the entire test chip (corresponding to figure 1), and there is no support for separate clock and voltage scaling for each processor. Also we must keep in mind in the

following the fact that the Emulation Baseboard allows the voltage to be scaled dynamically, but frequency only statically.

To use these hardware resources, we developed Linux modules that could meet our specific needs of power measurements in different platform configurations (a module to set voltage and frequency statically, and a module for power and energy monitoring). In addition we developed shell scripts that automate other configurations, like the number of processors activated, and power measurements possibly selecting among different video sequences. During the execution of an application, the power module samples the monitoring registers. It is activated every 200 ms by an external timer in such a way as to minimize a possible time overhead on the application. With this method, we observed less than 1% execution time differences with and without using the power module. In the following, we present in detail the multithread H.264 decoder that has been used to benchmark the MPCore processor.

3.1.2 Application mapping

The parallelization of the decoder exploits the possibility of slice decomposition of frames in the H264/AVC standard. Indeed a slice represents an independent zone of a frame: it can reference other slices of previous frames for decoding; therefore decoding one slice (of a frame) is independent from another (slice of the same frame). In our implementation, a slice is handled by a POSIX thread through the entire slice decoding process.

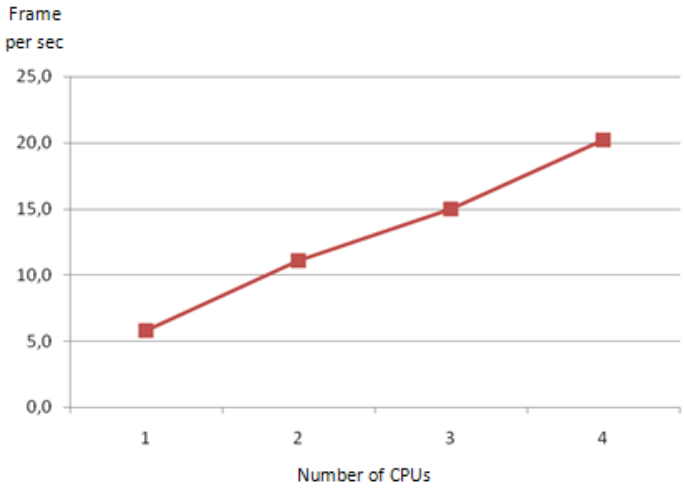


Figure 2. Slice partitioning: frame rate of an 8 slices configuration using 1, 2, 3, 4 CPUs

This way, the decoder can process different slices of a frame in parallel. There are very few data dependencies this way, and sequential execution remains only at the input stream level because of the sequential nature of reading NAL packets (Network Abstraction Layer). So this parallelization is expected to scale performances according to the number of slices and processors. Given a number of slices, we proceed in creating as many threads that can run concurrently according to the number of CPUs available. The implied data locality of this solution also benefits greatly from the L1 cache of the core, provided the amount of slice data fits the cache size. The performance results of figure 2 emphasize the scalability of a decomposition of eight slices: the decoding speed increases from 6.3 to 20.6 fps with the number of CPUs (1, 2, 3, 4), and the maximum speedup corresponds to a value of 3.27 using four CPUs. Slice partitioning thus represents a good trade-off between performance and complexity, both in terms of implementation and processing, since there is very little dependence between slices to manage.

Another advantage of slice decomposition is its regularity and homogeneity which is suited to SMP implementations and makes balancing the workload between CPUs very easy (1 slice = 1 thread). Nevertheless, processing independent slices in a frame has a counterpart: it reduces the range of motion search within a frame, and compression efficiency is thus altered when increasing the number of slices [34].

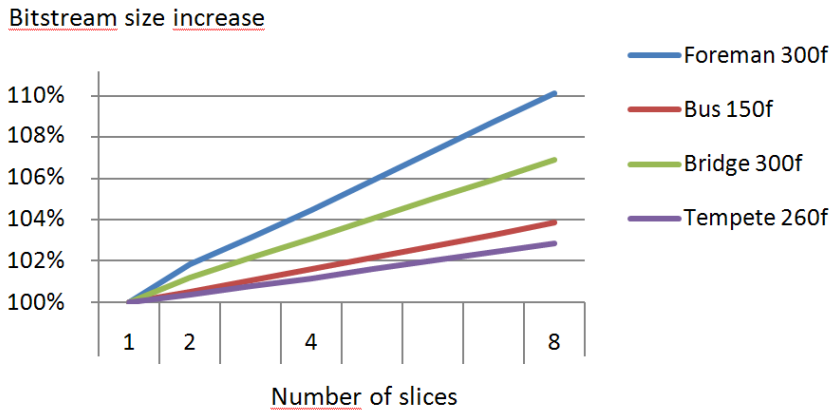


Figure 3. Evolution of the bitstream size for 1, 2, 4, 8 slices and different video sequences

Figure 3 shows the evolution of the bitstream size versus the number of slices. In a way to keep acceptable compression efficiency, we will not consider configurations of more than eight slices because it implies an

increase of more than 10% in the bitstream size (compared to a single slice configuration). Given this, we will consider the decoder only in the four following configurations 1, 2, 4, and 8 slices (or threads) in a way to provide the platform with different workloads.

3.2 Performance and power characterization

This section presents and discusses the performance and power measurements of the parallelized decoder on the MPCore platform in different configurations (1, 2, 4, 8 slices) and for several video sequences in 256x256 resolution (foreman, bus, bridge and tempete with 300, 150, 300, and 260 frames respectively). As explained previously, the parallelization associates one thread to each slice of a frame, so we can refer to the parallelism level in terms of the number of threads, which is exactly the number of slices per frame. The available configurations of 1, 2, 4 and 8 slices allow us to study the effect of different workloads on the platform. The next section addresses the performance analysis.

3.2.1 Performance analysis

Figure 4 reports the speedup results of the decoder in configurations of 1, 2, 4, 8 threads when four CPUs are activated, compared to the execution of a monoslice version using a single CPU. The platform is configured in nominal conditions at 1.20V/250/MHz. Figure 5 shows the corresponding performance in terms of number of frames decoded per second (fps).

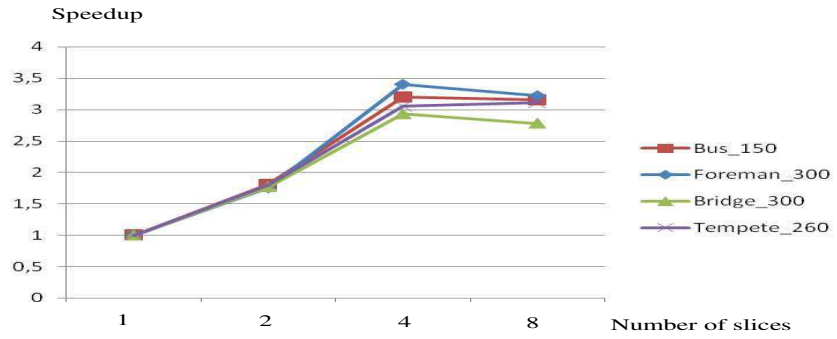


Figure 4. Speedup of the multislice decoder for 1, 2, 4, 8 slices, different video sequences, using 4 CPUs (250MHz).

The first observation is a linear acceleration growing from one to four slices. The execution speedup reaches a value of 3.19 (average on all

sequences), which is close to the theoretical maximum of four. This denotes a good balance of the workload (i.e. threads) between the processors and shows the relevance of slice partitioning for SMP implementations. The loss in performance with respect to the theoretical maximum is due (i) to the sequential nature of processing the input stream (before the creation of slice threads), (ii) to thread synchronizations and to a lesser extent (iii) to the influence of the operating system. These reasons are also responsible for the performance penalty in configurations of eight slices. In this case, the extra number of slices does not benefit from additional CPUs to speedup execution. Indeed, the number of threads per processor exceeds one, and therefore the effect of context switching and the associated L1 cache penalties may also have to be considered.

If we have a close look at the frame rates per video sequence (figure 5), we can notice important variations. Those variations are sensitive for instance between the bridge (27,2 fps) and the tempete sequence (18 fps). In the first one, small objects are moving in slow motion over a fixed background (low motion complexity). In the second one, the background is mobile and a group of objects are moving randomly and with fast motion (high motion complexity).

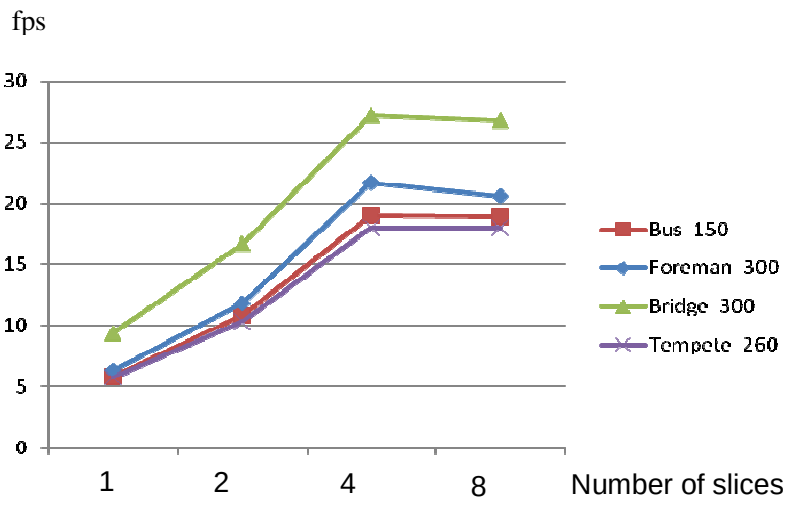


Figure 5. Frame rate of the multislice decoder for 1, 2, 4, 8 slices, different video sequences, using 4 CPUs (250MHz).

The frame rate appears to be very sensitive to motion properties in the video, with variations of about +/- 20%. This can be exploited to define a dynamic power strategy based on a frame rate adaptation using frequency scaling, which is discussed in section 3.3.

3.2.2 Power analysis

The following section analyzes the power consumption in different configurations of the decoder (workload) and the MPCore platform (voltage/frequency, number of CPUs). The test chip allows voltage variations ranging from 0.95V to 1.45V. To be compliant with ARM specifications (V_{max} = 1.20 + 10%), we have set the following operating points: 0.95V/150MHz, 1.08V/200MHz,1.20V/250MHz (nominal), 1.32V/300MHz. In these conditions, we have also considered various configurations of the platform (1, 2, 3, 4 CPUs) and workload (8, 4, 2, 1, 0 threads) to analyze the power consumption of the platform in different operational scenarios. Given the large quantity of results, we will first focus on the power consumption in nominal conditions presented in figure 6 (4 CPUS, 1.20V/250MHz). These measurements have shown very few differences in behavior between different video sequences, so we have represented only the power profiles of a foreman sequence. Indeed we can observe the same trends of power consumption in each condition of workload: variations occur between a well-defined minimum and maximum value, and the distribution of points depends on the processor activity. For balanced CPU/workload configurations (e.g. 1, 2, 4, 8 thread / 4 CPUs, figure 6), the power distribution is close to the maximum indicating a homogeneous and maximum load of processors. The average power consumption is well identified in this case. When the workload is unbalanced, for instance in a configuration of 4 threads/3 CPUs (figure 7), random variations occur between the minimum and maximum, indicating unequal demands on the processors resulting from a heterogeneous distribution of four threads on three CPUs. In this case, processing suffers from penalties caused by thread migrations and the associated L1 cache updates. As a consequence, these configurations are not energy efficient; the average power consumption is high and less predictable.

If we focus on the average power consumption for different configurations of CPU/workload (section 1.20V/250MHz of table 1), we can see that it is easily predictable when CPU loads are balanced: 840mW for 4 CPUs, 580mW for 2 CPUs and 415mW for 1 CPU. In the case of 3 CPUs, there are higher variations (typically in the 4 and 8 thread configurations) because of the non-homogeneous distribution of threads that leads to more unpredictable and irregular CPU activity. When there are fewer threads than active cores, we observe clearly that power consumption results from the number of active CPUs. For instance: 2, 4, 8 threads / 2 CPUs, or 2 threads / 2, 3, 4 CPUs exhibit very similar power consumption figures between 578 and 587 mW (1.20V/250MHz section of table 2). This results from the operating system's ability to disable the unused cores using a sleeping mode

called Wait For Interrupt (WFI). Finally, we have set other voltage-frequency couples in order to extend the power characterization of the platform.

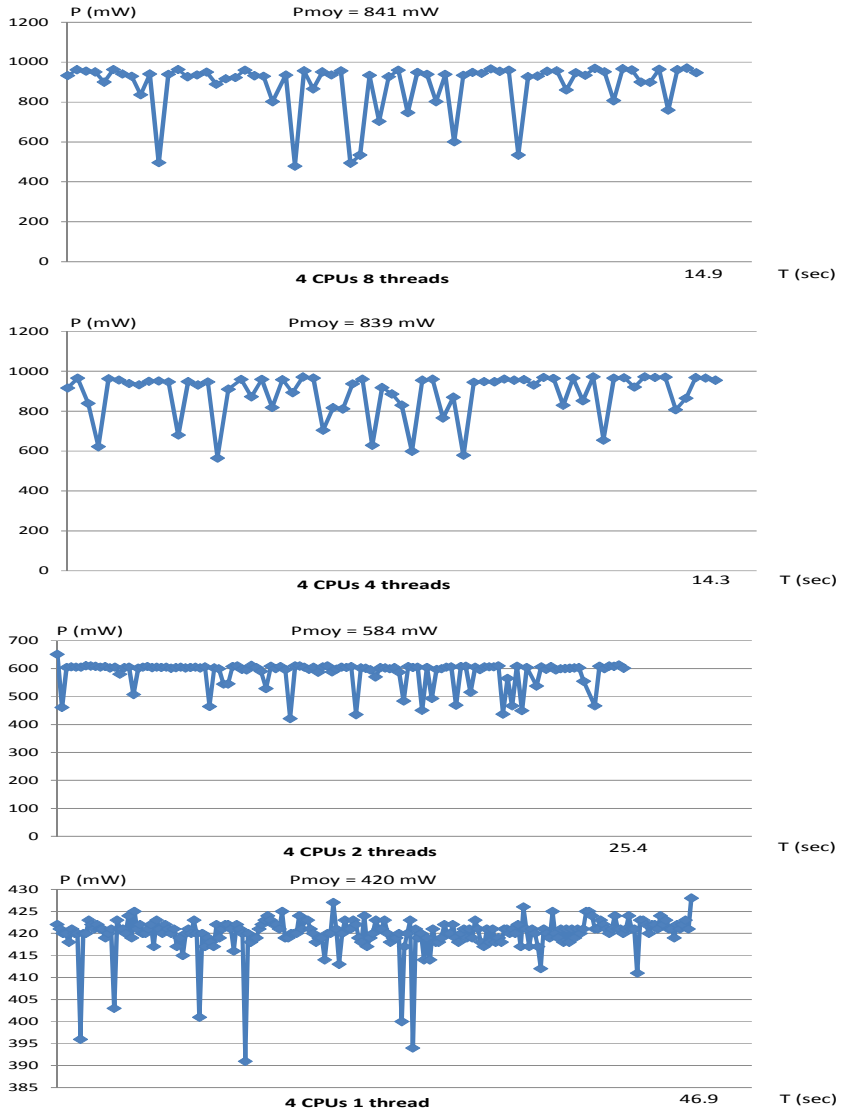


Figure 6. Power profiles of 4 CPUs (1.20V/250MHz) in different workload configurations (8, 4, 2, 1 threads) using the foreman sequence.

The following values have been considered: 0.95V/150MHz, 1.08V/200MHz and 1.32V/300MHz. Regarding the previous analysis, we expose the average power consumption and execution time in these different

voltage/frequency configurations, and for different workloads including when no user thread is run (8, 4, 2, 1, 0 threads). In each case, power profiles follow the same trends as those observed in the nominal case (figures 6 and 7).

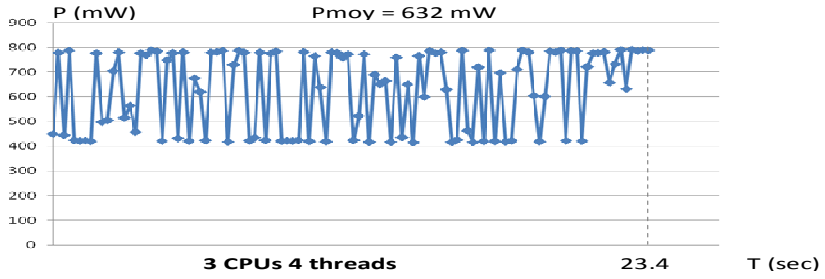


Figure 7. Power profile of 3 CPUs (1.20V/250MHz) in a workload configuration of 4 threads using a foreman sequence.

These results show that it is possible to derive a first simple yet accurate power model, provided we exclude sub-optimal configurations where the workload is unequally balanced between processors. Such a basic model is able to provide reliable power predictions from the number of threads and the voltage/frequency configuration of the platform. We address other key results that are discussed in terms of development of a power management strategy in the following section.

MPCore config	Decoder config				
1,32V/300MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**0 thread**
1 CPU	633/44.0	634/43.1	636/41.7	636/41.5	361/--
2 CPUs	885/22.6	878/22.0	880/22.2	637/40.3	364/--
3 CPUs	1034/17.2	932/20.5	882/21.6	637/40.1	363/--
4 CPUs	1256/13.0	1187/12.2	867/21.5	642/40.7	355/--
1,20V/250MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**0 thread**
1 CPU	415/50.7	415/50.8	416/49.3	416/48.9	241/--
2 CPUs	587/26.6	579/25.9	581/25.6	418/47.0	241/--
3 CPUs	701/19.8	632/23.4	578/25.6	418/57.6	238/--
4 CPUs	841/14.9	839/14.3	584/25.4	420/46.9	238/--

1,08V/200MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**0 thread**
1 CPU	270/62.1	270/61.9	270/60.3	271/59.8	156/--
2 CPUs	381/32.2	376/31.5	375/31.5	271/58.2	147/--
3 CPUs	462/23.7	408/28.7	381/30.6	272/57.5	153/--
4 CPUs	561/17.9	568/17.0	380/31.1	273/57.5	154/--
0,96V/250MHz	**8 threads**	**4 threads**	**2 threads**	**1 thread**	**0 thread**
1 CPU	160/82.9	160/80.9	160/79.2	160/78.4	93/--
2 CPUs	225/42.0	224/41.2	225/41.5	160/75.7	89/--
3 CPUs	276/30.9	241/36.5	224/41.1	161/76.2	89/--
4 CPUs	335/23.0	335/22.3	223/40.3	162/75.9	92/--

Table 1. Average power (mW) and decoding time (sec) of a foreman sequence (300 frames) in different configurations of platform (number of CPUs, voltage/frequency) and workload (number of threads).

3.3 Power management design and implementation

3.3.1 Impact of results

An interesting point from a DVFS perspective is the variation of about 40% in decoding speed resulting only from the motion properties of the video. Indeed this can be exploited to smooth the frame rate by a dynamic adaptation of processor frequency. Other results can also be exploited. Table 2 reports total performance variations ranging from 3.6 to 24.6 fps (extracted from table 1) when considering all the possible configurations of execution (platform, decoder). This gives room for adapting the performance to different video qualities: 15, 20 or 25 fps for example.

Performance variations					
Nb CPUs	Nb threads	V/F(V/MHz)	P(mW)	Perf.(fps)	E(mJ)
1	8	0.95/150	160	**3.6**	**13**
4	4	1.32/300	1187	**24.6**	**14.4**

Table 2: Differences of configurations (Number of CPUs, workload, voltage/frequency) and performances for an equivalent level of energy consumption.

We can also notice that the energy cost is comparable in both performance extrema (3.6 to 24.6 fps), and indicates that there are suboptimal configurations in terms of energy efficiency among all possibilities. We can also verify the intuitive assumption that a combination of a high number of active cores with low operating frequency is better from an energy point of view. To confirm this, we can consider a decoding constraint of 13.5 fps that can be satisfied in two configurations: four CPUs at 0.95V/150MHz and two CPUs at 1.32V/300MHz. The difference reaches a factor of two between the corresponding energy consumptions (table 3). This shows that it is more interesting to increase the number of CPUs while minimizing the frequency, for a given level of performance (13.5 fps in this case).

Energy efficiency				
Nb CPUs	V/F(V/MHz)	Perf.(img/sec)	P(mW)	E(mJ)
4	**0.95/150**	13.4	335	**7.51**
2	**1.32/300**	13.5	880	**18.80**

Table 3. Energy efficiency: effect of increasing the number of CPUs vs. frequency (at an equivalent performance level of 13.5 fps).

Another aspect concerning energy efficiency is illustrated in figure 8, which relates the energy consumption measured in a configuration of four tasks for different number of cores and voltage/frequency. These results show that energy efficiency increases when using more CPUs, by a factor of 1.8 from 1 to 4 processors.

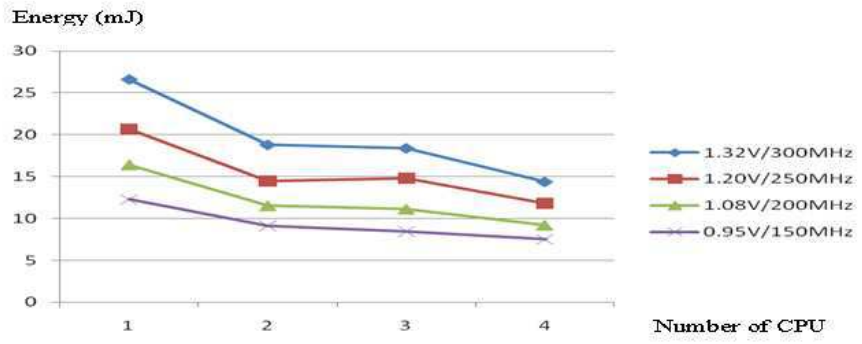


Figure 8. Impact of load balancing on energy efficiency (configuration of 4 threads using 1, 2, 3, 4 CPUs in 4 voltage/frequency settings).

We can also notice that energy efficiency is very sensitive to the distribution of the workload with a clear inflection point in each configuration of 3

CPUs, showing that energy efficiency is altered. In these conditions, it is more energy efficient to maximize the number of CPUs while minimizing frequency, but also to pay attention to workload balancing in order to minimize the energy waste.

From these considerations, we have derived the following strategy: first, a static adaptation of the number of active CPUs to the workload according to a quality constraint, and second, a finer adjustment of performance variations using frequency scaling. In this case, we chose to use four CPUs to benefit from the maximum video quality (25 fps), and then adjust the decoding speed by dynamic frequency scaling, around 20 or 15 fps, which will result in lowering the power consumption. It could also be possible to use 2 CPUs (12 fps at 250MHz) and adapt the decoder speed at a lower value, such as 8 fps for example.

3.3.2 Implementation of the adaptation strategy

The principle of the adaptation strategy is to control the decoder speed around a frame rate constraint slightly lower than the average performance in nominal conditions (to decrease the operating frequency). The adaptation is based on changing the frequency (thus voltage) when the frame rate is between two defined thresholds. The number and values of these thresholds depend on the operating points (voltage-frequency couples) and on the performance constraint to satisfy. To implement this adaptation on the MPCore, we used seven fps thresholds that were derived from the following operating points: 0.96V/150MHz, 1.02V/175MHz, 1.08V/200MHz, 1.14V/225MHz, 1.20V/250MHz, 1.26V/275MHz, 1.32V/300MHz. Each threshold $tresh_i$ is associated with a given frequency f_i which is computed as follows:

$tresh_i = adaptation\_const * f_{nom} / f_i$ where f_{nom} is the nominal frequency (250MHz)

For an adaptation constraint of 20 fps, we have the following thresholds: 33.3, 28.6, 25, 22.2, 20, 18.2, 16.7 fps defined respectively for 150, 175, 200, 225, 250, 275, 300 MHz. When the decoder speed remains in a zone delimited by two consecutive thresholds during a sufficient amount of time (to minimize the number of clock switching), the processor frequency is switched to the value associated with this zone. As an illustration, if the frame rate is between 28.6 and 33.3 fps during 250 frames, the operating point is set to 0.96/150MHz.

Changing the supply voltage is done as explained in section 3.1.1. Concerning frequency, it can not be set dynamically, but only statically after a reset of the Emulation Baseboard. Whereas dynamic clock switching is not supported on the platform, we have nevertheless implemented an adapted

version of the DVFS strategy defined above in order to check the possible energy gains. The energy consumption could be measured this way using the procedure of section 3.1.1, but considering dynamic voltage scaling only. Frequency scaling could not be included in the strategy we developed for evaluation, but we have tried to implement a strategy as close as possible to the DVFS strategy described above.

We have thus developed a DV(F)S driver for Linux with the following characteristics. It handles the switching of operating points with respect to the decoding speed. It samples the frame rate at regular time intervals (every $T_{monitor}$ images), and makes the decision of switching or not the operating point every T_{eval}*$T_{monitor}$ images. A T_{switch} parameter delay is also simulated to take the effect of PLL setting times into account. To analyze the adapted performances of the decoder, the driver computes a trace of the frame rate which is extrapolated from the actual performance at 250 MHz and the frequency that should have been changed to by the driver. Power and energy are measured using the procedure of section 3.1.1; results are reported in the next section.

3.3.3 Results

The power and performance measures for two regulations at 20 and 15 fps of a 1 minute 20 second video sequence are given in figures 9 and 10. In nominal conditions (1.20V/250MHz), the sequence requires 62 Joules to be decoded at an average speed of 24.8 fps. Both figures show traces of the original and regulated frame rates as well as the trace of power consumption. On the power profiles, we can clearly observe different voltage/frequency domains, thus the DVS switches that can be identified by distinct maximum values.

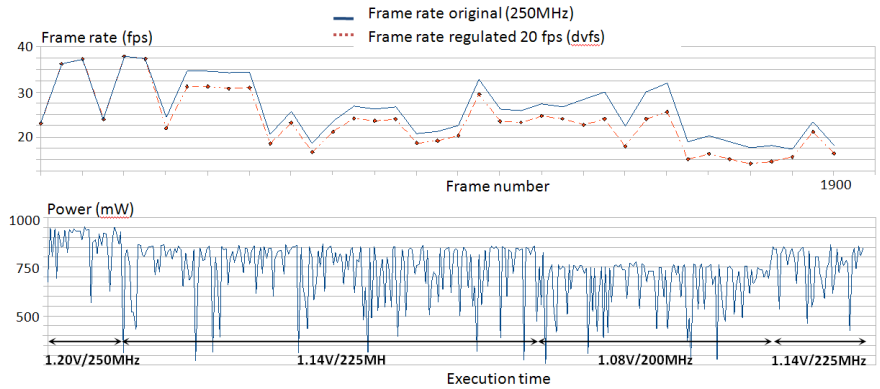


Figure 9. Frame rate and power profile of a 20 fps regulation (9.7% energy gains)

Energy gains are good in both cases with respectively 9.7 and 30.6% for 20 and 15 fps. Because the platform remains at an operating frequency of 250 MHz, we must emphasize that the energy reported does not consider two effects that would result from an effective frequency scaling: the energy consumption should be higher because the execution time increases when frequency decreases. At the same time, the energy consumption decreases with frequency and voltage because dynamic power consumption is proportional to αCV^2F [35]. These opposing effects will result in compensation, so we can reasonably assume that the energy gains measured are close to the results of an effective DVFS technique.

The performance profiles of figures 9 and 10 allow comparing the evolution of the decoder performances at 250MHz (full line) versus scaling frequency (dotted line). Each point is evaluated every $T_{monitor}$ images and a decision of changing the frequency or not is made every $T_{eval}*T_{monitor}$ images. In these conditions, three configuration switches are operated for a 20 fps adaptation (1,14V/225MHz - 1,08V/200MHz - 1,14V/225MHz) and only two switches but at lower frequency (0,96V/150MHz - 1,02V/175MHz) for a 15 fps adaptation, which results in better energy gains.

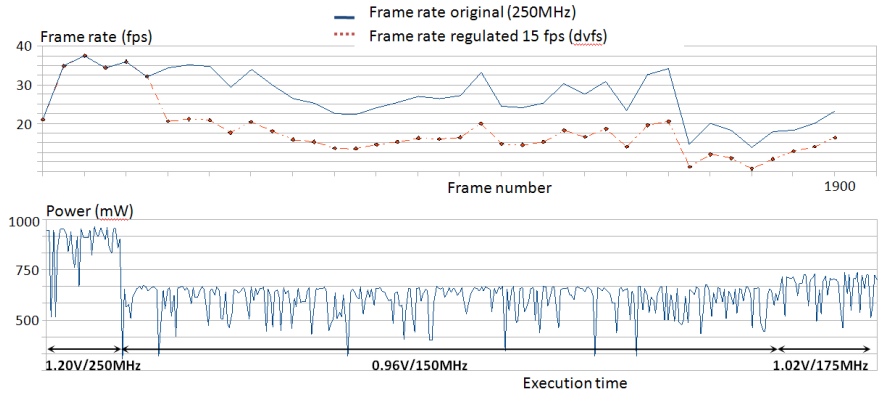


Figure 10. Frame rate and power profile of a15 fps regulation (30.6% energy gains)

Obviously the potential energy gains are linked to the performance. This is due to the fact that performance improvement allows proportional frequency (and voltage) reduction. As a consequence, there is still room for optimizing the decoder using SIMD instructions in particular (1.5/2x speedups), that could permit to run the platform at lower frequency (150MHz, 335 mW), or even to meet fps constraints using less than four CPUs.

3.4 Conclusion

The work exposed in this section provides many results on power and performance that have stressed the impact of load balancing on energy efficiency of multiprocessor platforms. They have led us to propose a management strategy for video processing that has been implemented with reported energy savings of up to 30.6%. Whereas the achievable gains depend on the level of application performance optimization and video quality required, the exploitation of strategies that has been tuned for domain specific applications lead us to anticipate high energy savings, compared to general purpose strategies. It is highly probable that power management in the future will have to adapt strategies to different application domains in order to achieve important energy gains. Another potential source of high power reduction concerns the memory system that has to be controlled under similar constraints. This is also a promising field of investigation which is the purpose of the following section.

4. MEMORY POWER MODEL

4.1 Memory architecture

Processor power management using techniques such as DVFS tends to increase application run time and therefore to stretch the active time and the energy consumption of the main memory. As an example, for a given processing rate, we can operate with 4 CPUs at a frequency of 300MHz (1.32V) for a duration of T/2 and afterwards go into WFI mode; or alternatively for a duration T with a frequency of 150MHz (0.95V). In the latter case, we notice that the power consumption of the MPcore is reduced by a factor of 2.32. If we add the memory consumption, for instance, let us consider one bank of RDRAM memory [2] operating in its first low-power mode, such that the power saving factor falls to 1.53. If a deeper low-power mode is used, this factor decreases again, consequently reducing the benefit that could be achieved by frequency tuning.

Moreover, for future technologies, static power is expected to become the dominant contribution to energy consumption in off-chip memory. To solve this issue, one solution consists of using a multi-bank memory system with an efficient management policy of low-power modes of the different banks. A methodology for allocation of tasks into the multi-bank memory could minimize the energy consumption by extending the time spent in low-power modes and/or by reducing the number of bank wake-ups. To service a

memory request (read or write), a bank must be in active mode which consumes most of the power. When a bank is inactive, it can be put in any low-power mode (for instance standby, nap, power-down mode). Each mode is characterized by its power consumption and the time that it takes to transit back to the active mode (resynchronization time). The lower the energy consumption of the low-power mode, the higher the resynchronization time becomes. The addressed SMP architecture (like the MPcore) has a two level cache arrangement with a bus-based communication model. A shared multi-bank memory is linked to the processors as shown in figure 10. Each bank can be controlled independently and placed into one of the available low-power modes. Each low-power mode is characterized by the number of components being disabled to save energy. In the following, our aim is to show that an appropriate memory allocation can enable significant power savings and to propose a strategy in relation to the energy-aware scheduling policy.

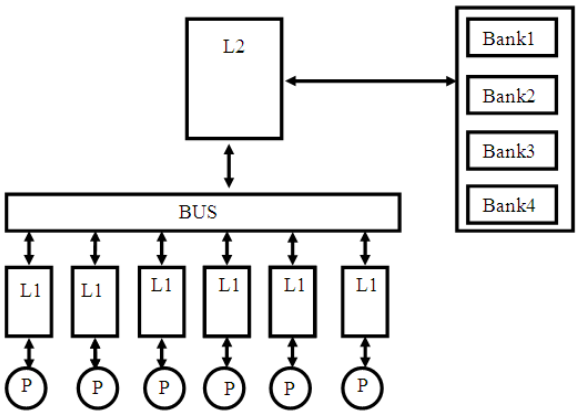


Figure 10. Architecture of the memory system

4.2 Multi-bank Memory and system model

The memory is described by architectural parameters: the number of banks (NB_{mem}), the bank size (SB_{mem}) and the number of low-power mode (N_{LP}). In active mode, a bank is characterized by its power consumption (P_{active_mem}) and the access time for read/write operation (T_{access}). In each low-power mode *l*, the memory features to consider are: power consumption ($P_{lp\_mode\text{-}l}$), wakeup time ($T_{lp\_mode\text{-}l}$) and an energy overhead when the bank goes to the active mode ($E_{mode\_switch\text{-}l}$). A dynamic system will be able to model the different modes of the memory as in the ACPI [4] and to assess the power savings in order to apply an efficient power strategy.

The application is composed of a set of tasks. Each task t_i is characterized by its worst case execution time $WCET_{ti}$ (or the average execution time). In order to evaluate the memory power consumption, some additional parameters must be taken into account: the task memory size S_{tj} (code and data) and the number of main memory accesses $L2\_M_{tj}$. These last parameters correspond to the number of L2 cache misses and can be collected by profiling. Usually, this is the average number of cache misses derived from simulations or previous executions. We assume that the task size is less than or equal to the size of a memory bank.

4.3 Energy model

The energy consumption of a multi-bank memory depends on the power dissipated in the different modes and the time spent in these modes. In the following equation, the energy model is given for the worst case execution time ($WCET_{ti}$) of the task but it can be replaced by the actual execution time. Usually the worst case execution time is specified for the maximum speed of the processor and need to be adjusted when the frequency is scaled down: $WCET_{ti}$ (fcpu). This time is divided into two time intervals: the first one, for read and write operation (bank is accessed), and the second one, when the bank is not accessed but still active: $T_{ti} = T_{access\text{-}ti} + T_{noaccess\text{-}ti}$.

The access time is the result of the multiplication of the number of memory accesses and the read/write time: $T_{access\text{-}ti} = L_2\_M_{ti} * T_{access}$. $T_{access\text{-}ti}$ is independent of the processor frequency and therefore $E_{access\text{-}ti}$ is a constant as regards the selected processor frequency. The time spent when no memory access occurs ($T_{noaccess\text{-}ti}$) is the difference between the access time and the worst case execution time and consequently is a function of the processor frequency: $T_{noaccess\text{-}ti}(f_{CPU}) = WCET_{ti}(f_{CPU}) - T_{noaccess\text{-}ti}$.

An allocation function noted φ has to be defined which associates each task t_i belonging to a set of N tasks to a bank b_j belonging to a set of k banks.

$$\varphi: \{t_1,t_2,\ldots t_N\} \rightarrow \{b_1,b_2,\ldots b_k\}$$

$$\varphi(t_i) = b_j$$

The energy consumption of a memory of k banks and a given allocation of N tasks to these banks is evaluated by the sum of the energies consumed in the active mode (E_{active}), in the low-power modes (E_{mode_switch}) and to switch between the modes (E_{lp_mode}): $E_{memory} = E_{active} + E_{mode\_switch} + E_{lp\_mode}$. We now describe the individual contributions in detail.

E_{active}:

The memory bank is assumed in an active mode during the whole duration of execution of a task. The time spent in active mode is divided into two time intervals: the first one for the read/write operation (the bank is

accessed) and the second one, when the bank is not accessed but still active: $E_{active} = E_{access} + E_{noaccess}$. The equation for E_{access} is given by:

$$E_{access} = \sum_{b_j / j=1}^{k} \left(\sum_{t_i /(\varphi(t_i))=(b_i)}^{N} L_2 \_ M_{t\,i} * E_{mem\_access} \right)$$

$E_{noaccess}$ is the energy due to the co-activation of the different banks of the memory when these banks are active but not servicing any read or write operation. The time spent in this configuration is called, for a bank b_j, $T_{no\_access\_bj}$.

$$E_{noaccess} = \sum_{bj / j=1}^{k} (T_{noaccess-bj} * P_{noaccess})$$

When two tasks, stored in a same bank, are running in parallel, the no access time is the interval between the start time and the end time of the two tasks minus the access times of the two tasks. As an example let us consider two tasks running according to the scheme below.

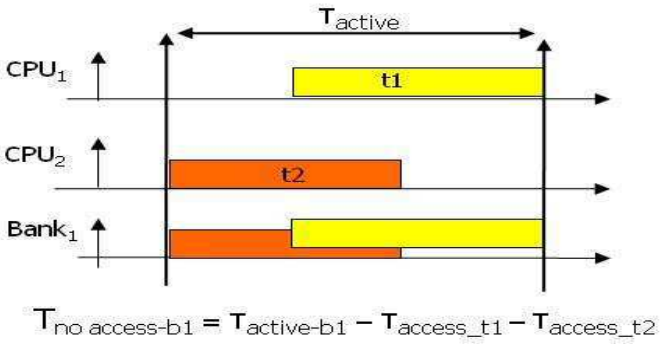


Figure 11. Example of memory mapping and active time

The no access time is dependent on the processor frequency and also increases the no access energy. For instance, if a processor is slowed down by a factor *S*, the no access energy of the memory is multiplied by the same factor. This drawback can be mitigated by grouping tasks running in parallel.

E_{mode_switch}:

This energy is dissipated during the transition of the memory bank from a low-power mode to the active mode:

$$E_{mode\_switch} = \sum_{b_i / j=1}^{k} (\sum_{l=1}^{N_{LP}} N_{mode\_switch-l}(b_j) * E_{mode\_switch-l})$$

with $N_{mode\_switch\text{-}l}$ (bj), the number of switches per bank.

E_{LP_mode} *:*

This energy is consumed in the memory banks in low-power modes:

$$E_{lp\_mode} = \sum_{B_i / j=1}^{k} (\sum_{l=1}^{N_{LP}} T_{LP\_mode-l}(b_j) * P_{LPmode-l})$$

$T_{lp\_mode\text{-}l}$ is the time spent by the memory bank b_j in low-power mode *l.*

4.4 Energy-aware memory allocation

4.4.1 Energy-aware memory allocation: principle

In most cases, the energy in the active mode is the major part of the memory energy consumption. In order to minimize this part of the energy and also to decrease the effect of the processor slowdown, our strategy is to try to allocate tasks running in parallel to the same bank. We can consider two kinds of DVFS in a multiprocessor platform: chip-wide (all the processors run with the same frequency) and per-core (the processors can run at different frequencies). Our approach is implemented with two queue structures consisting in the running task queue (RUQ) and ready task queue (REQ) and the knowledge of the memory requirements. Each bank is described by the list of running tasks which are allocated to it, the end time of these tasks (based on the WCET or the average actual execution time) and the remaining memory size. When a task begins its execution, the attributes of bank is updated. The scheduler in charge of each task selects a processor, a frequency and a memory bank. In the case of a chip-wide DVFS system, all the tasks are slowed down. The end time of all the running tasks must be updated. For a per-core DVFS system, only the considered task is affected by the slowdown factor. The recovery time between two tasks is defined as the time for which these two tasks are running in parallel. A task is allocated to the active bank that has the highest recovery time. In the reverse case, no active bank has enough memory to store the task, two choices are envisaged: the task execution is delayed or a bank in low-power mode is woken up. In

order to assess the power saving of these two ways, the memory needs of the task in REQ are analyzed. The memory size of the tasks belonging to the REQ is compared to the memory size freed by the task that will finish their execution on a window time equal to the wakeup time of the bank. If the memory needs are inferior or equal to the freed memory size, the tasks are delayed. Otherwise, a bank is woken up and a time overhead is added.

4.4.2 Results

In order to illustrate the benefit of our approach, we consider a simple example, with four tasks t_1, t_2, t_3, t_4 with respectively an execution time of T/2, T, T/2 and T at the maximum frequency (300MHz). The memory size of these tasks is 3Mb. The access time is ten percent of the execution time (average value for a H264 decoder). The memory is a RDRAM described in [2] and is composed of four banks of 8Mb. The low-power mode used for this example is standby. When a processor is not busy, it goes into WFI mode. Three configurations of the MPcore were analyzed in table 5 that gives the overall energy consumption of the memory system and the processors (MPcore).

Normalized energy	Without energy-aware allocation	With energy –aware allocation
4 CPUs (300MHz, 1.32V)	1	0.92
4 CPUs (150 MHz, 0.95V),	0.78	0.65
2 CPUs (300 Mhz, 1.32V)	0.85	0.82

Table 5. Normalized energy for three configurations of the MPcore

This example shows that the memory consumption is significantly reduced with an adapted power management and it is complementary to processor power management. Other examples show that some operating points don't enable power savings if we take into account the memory consumption. This fact leads us to believe that the energy benefit of an operating point must be assessed on the overall system.

5. CONCLUSION

Existing approaches of power management are mostly generic. For example, *cpu-freq* implemented in Linux is based on monitoring the CPU workload. When the workload decreases, processor frequency is reduced, and so is the power consumption. In case of video processing for example, this approach is inefficient because the application represents most of the

time a maximum load which results in the processor operating at the maximum clock frequency. The results we have described demonstrate the need for domain specific strategies (for example multimedia or networking) in addition to general purpose strategies to enable higher levels of energy gains. This means that the software infrastructure of power management and the operating system must be flexible and able to handle different policies. Another point concerns the possibility of other techniques alternative to DPM and DVFS. For instance, interesting extensions to investigate should include memory and/or dedicated (possibly dynamically reconfigurable) accelerators, within power management strategies. For a simple example, we have shown the benefit of including a memory energy-aware strategy that is complementary to a scheduling policy that exploits the DVFS technique.

REFERENCES:

1. Contrebas G., Martonosi M., Power Prediction for Intel XScaler Processors Using Performance Monitoring Unit Events, ISPLED 2005.
2. Fan X., Ellis C., Lebeck A., Modeling of DRAM Power Control Policies Using Deterministic and Stochastic Petri Nets, lecture Note in Computer Science, volume 2325/2003, pp. 37-41.
3. Infineon Inc, Mobile-RAM data sheet" 2004.
4. ACPI, Advanced Configuration and Power Interface, http ://www.acpi.info/, 2006.
5. IBM, MontaVista, Dynamic Power Management for embedded systems, 2002.
6. ARM, Intelligent Energy Controller Technical Reference Manual, ARM Limited, http://infocenter.arm.com/, 2008.
7. Intel, Wireless Intel SpeedStep Power Manager, White paper, 2004.
8. Intel, Intel PXA270 Processor, Electrical, Mechanical, and Thermal Specification, http: www.intel.com, 2005.
9. Hwang C..,Wu A., A predictive system shutdown method for energy saving of event-driven computation, International Conference on Computer-Aided Design, p. 28-32, November, 1997.
10. Benini L., Bogliolo A., Micheli G. D., A survey of design techniques for system-level dynamic power management, IEEE Transaction on Very Large Scale Integration System, p.299-316, 2000.
11. Rong P., Pedram M., Determining the Optimal Timeout Values for a Power-Managed System based on the Theory of Markovian Processes: Offline and Online Algorithms, Proc. of Design Automation and Test in Europe, 2006.
12. Qiu Q., Liu S., Wu Q., Task merging for dynamic power management of cyclic applications in Real-Time multiprocessors systems, ICCD 04.
13. Merkel A., Bellosa F., Energy Power consumption in Multiprocessor systems, EuroSys2006, 2006.
14. Luo J., Jha N. K., Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems, International Conference on VLSI Design, January, 2002.
15. Chen J.-J., Kuo T. W., Energy efficient scheduling of Periodic Real-time tasks over homogeneous Multiprocessors, PARC05, 2005.

16. De Langen P., Jurlink B., Vassiliadis S., Multiprocessor Scheduling to Reduce Leakage Power, 17th International Conference on Parallel and Distributed Symposium, 2006.
17. Benini L., Bertozzi D., Guerri A., Milano M., Allocation, Scheduling and Voltage Scaling on Energy Aware MPSoCs, LNCS Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, vol. 3990/2006, p. 44-58, June, 2006.
18. Choudhury P., Chakrabarti C., Kumar R., Online Dynamic Voltage Scaling using Task Graph Mapping Analysis for Multiprocessors, VLSI Design, 2007.
19. Zhu D., Melhem R., Childers B., Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems, IEEE Trans. on Parallel & Distributed Systems, vol. 14, no. 7, pp. 686 - 700, 2003.
20. Chen J., Yang C.Y., Kuo T., Shih C.-S., Energy efficient real-time system task Scheduling in multiprocessor DVS systems, ASP-DAC, January, 2007.
21. Yang C., Jian-Jia Chen, Tei-Wei Kuo, Lothar Thiel, An Approximation Scheme for Energy-Efficient Scheduling of Real-Time Tasks in Heterogeneous Multiprocessor Systems, DATE 09, April 2009.
22. Chou P. H., Liu J., Li D., Bagherzadeh N., IMPACCT : Methodology and tools for power aware embedded systems, Design Automation for Embedded Systems, Kluwer International Journal, Special Issue on Design Methodologies and Tools for Real-Time Embedded Systems, p. 205-232, 2002.
23. Srivastava M., Chandrakasan A., Brodersen R., Predictive system shutdown and other architectural techniques for energy efficient programmable computation, IEEE Trans. VLSI Systems, Vol. 4, pp. 42-55, Mar. 1996.
24. Bhatti M.K., Muhammad F., Belleudy C., Auguin M., Improving resource utilization under EDF-based mixed scheduling in multiprocessor real-time systems, 16th IFIP/IEEE Int. Conf. on Very Large Scale Integration, VLSI-SOC'2008, Rhodes, Greece.
25. Kandemir M., Kolcu I., Kadayif I.,Influence of loop optimizations on energy consumption of multi-bank memory systems, in Proc. Compiler Construction, April 2002.
26. Ozturk O., Kandemir M., Irwin M.J., Increasing OnChip Memory Space Utilization for Embedded Chip Multiprocessors through Data Compression, CODES05.
27. Koc H., Ozturk O., Kandemir M., E. Ercanli, Minimizing Energy Consumption of Banked Memories Using Data Recomputation, ISLPED'06.
28. Kandemir M., Ozturk O., Non-uniform banking for reducing memory energy consumption, DATE'05 Munch, Germany 2005.
29. BenFradj H., Belleudy C., Auguin M., Multi-Bank Main Memory Architecture with Dynamic Voltage Frequency Scaling for System Energy Optimization, 9th Euromicro, September, 2006.
30. Delaluz V., Kandemir M., Vijaykrishnan N., Sivasubramaniam A., Irwin M.J., DRAM Energy Management Using Software and Hardware Directed Power Mode Control, International Symposium on High Performance Computer Architecture, 2001 pp.159-170.
31. Lebeck A., Fan X., Ellis C., Memory controller policies for DRAM power management, International Symposium on Low Power Electronics and Design, 2001.
32. Delaluz V., Sivasubramaniam A., Kandemir M., Vijaykrishnan N., Irwin M.J., Scheduler Based DRAM Energy Management, DAC 2002.
33. ARM, Core Tile for ARM11 MPCore User Guide, ARM Limited, http://infocen-ter.arm.com/, 2005.

34. Roitzsch M., Slice-balancing H.264 video encoding for improved scalability of multi-core decoding, 7th ACM & IEEE International conference on Embedded software (EM-SOFT)p. 269-278, September, 2007.
35. Mudge T., Power : A First-Class Architectural Design Constraint, IEEE Computer Society, vol. 34, n° 4, p. 58-52, 2001.

1861-5252/© 2013 TSSD

Transactions on
Systems, Signals & Devices
Vol. 8, No. 3, pp.345-362

Fast prototyping H.264 deblocking filter using ESL tools

T. Damak,[1] I. Werda,[1] S. Bilavarn[2] and N. Masmoudi[1]

[1] *Laboratory of Electronics and Information Technologies (LETI). Sfax University, National Engineering School of Sfax, Tunisia.*

[2] *Laboratory of Electronics, Antennas and Telecommunications (LEAT). Nice-Sophia Antipolis University, Nice, France.*

Abstract This paper presents a design methodology for hardware/software (HW/SW) architecture using Electronic System Level (ESL) tools. From C++ descriptions, the design flow is able to produce hardware blocks that can fully operate with software on FPGA based prototyping platforms from Xilinx. An efficient cooperation of a High Level Synthesis (HLS) tools :Catapult C Synthesis, a logic synthesis and a Xilinx developement tools is detailed to create the mixed architecture in shorter design time. An application/design study of this methodology is made on an optimized deblocking filter function, which is part of a complete H.264/AVC video coding system. Various performance / area tradeoffs are easier to explore thanks to this methodology, and this has led to hardware acceleration up to 95,5% against the original software execution in the considered example. The corresponding throughput is 180 cycles/macroblock at 100 Mhz.

Keywords: H.264, Deblocking filter, ESL, Catapult C, Hardware acceleration, FPGA.

1. Introduction

Recently, the intensifying demand of multimedia services and applications has led to the need of embedded systems supporting ever-increasing functionality and flexibility [1]. Because of this evolution, embedded

media processing systems are more difficult to design and develop under shorter time to market constraints.

The H.264 video coding standard [2] is a representative example of this trend, especially when considering typical performance and low-power constraints of embedded implementations. To speedup performances and to overcome the processing complexity of H.264 codec, several works have been proposed considering both software optimization [3] and hardware acceleration [4]. Comparison between the two approaches, hardware and software, is made difficult because design characteristics are very different. Software implementation does not allow exploiting the same level of parallelism that hardware design can. However, when the complexity of a hardware function grows, hand coding at Register-Transfer Level (RTL), which is already low and error prone, adds debugging and verification overheads that impact severely the time and costs of development. Therefore, Electronic System Level (ESL) brings a solution to decrease the design time of dedicated hardware and keep the high abstraction level of software development. The automation of the RTL generation process also results in the exploration of a wider design space. Some works reported five time shorter development time against manual RTL coding for comparable quality of results [5]. As a result, many commercial and academic high-level synthesis tools are available today that mostly automate the generation of RTL code from C/C++/systemC descriptions [6]: AccelFPGA (AccelChip), SystemC Compiler (Synopsys), SPARK, Cathedral, Catapult-C (Mentor Graphics).

In this paper, a specific methodology is used to develop all software and hardware needed to deploy an accelerated application on a reconfigurable platform from pure C specifications. First, the full software code of a H.264 video decoder has been developed in C++ [7]. Then, the elementary functions of the video decoder have been analyzed in detail to identify the most attractive candidates for hardware acceleration. Profiling results pointed out that the most expensive software task in C++ implementation of the decoder was the deblocking filter which is used to reduce the edge artefacts created by macroblock (MB) processing. The following parts describe how the previously mentioned methodology has been used to develop a fully operational dedicated accelerator that offloads the CPU from processing the deblocking filter function. The remaining of the paper is organized as follows. In Section 2, an overview of the H.264 decoder and the corresponding profiling results are presented. Details on the deblocking filter and its implementation are given in Section 3.Section 4 explains the proposed design methodology in two sub sections. The first one describes the design of the hardware accelerator

using Catapult C Synthesis, while the second sub section presents the design of the mixed hardware-software architecture. Section 5 exposes and discusses the implementation results obtained on a Virtex-5 FPGA with a PowerPC Embedded Processor and the dedicated filter accelerator in different parallelism configurations. A comparison with other hardware implementations of the deblocking filter is provided and discussed in section 6. Finally, a conclusion and perspectives are drawn in Section 7.

2. H.264/AVC decoder overview

The H.264/AVC (known as MPEG-4 part 10) [2] is a video compression standard which provides gains in compression efficiency of up to 50% over a wide range of bit rates and video resolutions compared to previous standards. At the input of the H.264 decoder, an encoded video file is first processed by a Context-based adaptive variable-length coding (CAVLC) and Exp-Golomb depending on data types. The resulting data elements are reordered to produce a set of quantized coefficient arrays that are then rescaled and inverse transformed to find back the original residual macroblock. A macroblock is an array of 16×16 pixels. A block is a 4×4 pixels array.

Using the header information decoded from the bitstream, distinct prediction modes are generated for luminance (luma) and chrominance (chroma) components to recompose the respective prediction residues. Depending on the prediction mode, either motion compensation reconstructs each macroblock from the reference frames (inter frame prediction). Either, the macroblock is reconstructed from the neighbouring macroblocks in the same frame (intra frame prediction). For intra prediction, there are in total 9 optional prediction modes for each 4×4 luma block, 4 optional modes for a 16×16 luma macroblock and 4 modes for chrominance samples are supported in this implementation. For inter frame prediction, the reference frame number and motion vector for each 4×4 block within a macroblock is obtained from the bitstream. In this implementation, only previous frame is considering as reference frame.

Predicted pixels macroblocks are then sent to the reconstruction module to be added with their corresponding residual macroblocks. This process is applied for each 16×16 macroblock until the whole frame is reconstructed. Finally, a smoothing filter called deblocking filter is applied to the resulting frame in order to reduce the visual artefacts of macroblock processing.

The proposed decoder has been validated against H.264 encoded video files to serve as a C++ reference model for the next steps. From this reference code, profiling was used to analyze the suitability of elementary functions for further optimization. Results are shown in Fig. 1.

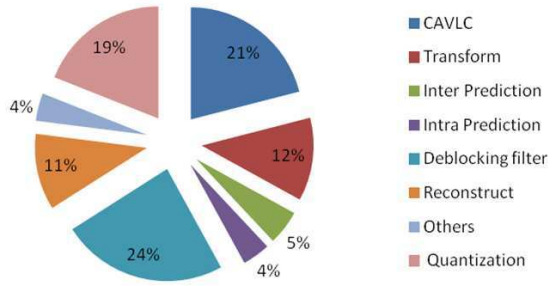


Fig. 1. H.264 decoder Profile.

Proposed decoder partitioning is approved by different works [8, 9]. In [8] implementation, deblocking filter takes 33% of decoder while it takes 38% in [9]. Despite of difference percentages between the proposed implementation and [8, 9], deblocking filter constantly occupied the lion share of decoder time execution. Compared with others literature results, CAVLC and inter prediction modules are the most complex parts of decoder. Such as in [10], deblocking filter takes only 15%, while CAVLC module takes 27% and motion compensation takes 29% of decoder time processing.

For the proposed decoder, CAVLC and motion compensation take respectively only 21% and 5% (Fig. 1). Both modules are not gourmand in term of time consuming. In fact, the proposed CAVLC is optimized in previous work [11]. It has taken 42% of decoder time execution before algorithmic optimization and it was the most consuming module. In addition, proposed decoder applied only one frame reference in inter prediction and smaller block size for motion vector is 8x8 instead of 4×4. These proprieties, which are detailed in [12], alleviate motion compensation complexity. The most expensive task in decoder profile is the deblocking filter which has an average share of 24% of the overall decoding time. Due to its highly computationally intensive nature, the deblocking filter can limit the ability to reach standard frame rate requirements of 25 or 30 frame per second. Based on the profiling results

of Fig. 1, the deblocking filter is the first promising function to take advantage of dedicated hardware acceleration and to reduce the overall processing time of a frame.

3. Deblocking filter algorithm

Adaptive deblocking filter is a post-processing technique of H.264/AVC applied to the reconstructed frame both in the encoder and the decoder. This filter greatly contributes to provide a better visual quality for the user and to enhance the rate-distortion performance of H.264/AVC-based video coding systems. Deblocking filter module is applied after transform and quantization steps.

Those last steps treat data by MB (16×16 pixels) and then by block (4×4 pixels). Consequently, they create significant blocking artifact that affect the video quality. As shown in Fig. 2, the deblocking filter is applied to each edge of a 4×4 block inside and between 16×16 macroblocks. It is conditionally applied to smooth horizontal and vertical block edges: vertical edges are first processed from left to right, and then horizontal edges from top to bottom. The condition depends on the boundary strength (BS) and the pixel gradient across the boundary. The applying of deblocking filter to a macroblock is processed in two steps: BS computation and applying the filter.

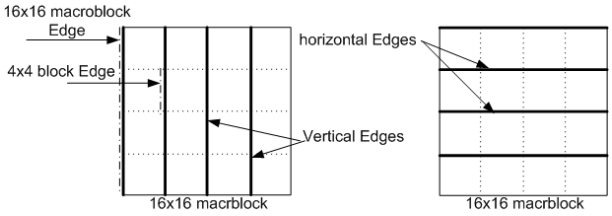


Fig. 2. Horizontal and vertical edges in a macroblock.

The first step generates 32 BS values for 16 horizontal edges and 16 vertical edges within a macroblock. As presented in Fig. 3, a 32-bit BS value is computed for each left or top edge of a 4×4 block within a given MB to evaluate the necessity of filtering and to choose adequate type (Standard or Strong).The BS algorithm depends on different parameters such as the prediction mode, motion vectors and reference blocks. It also depends on the edges type: block edge or a macroblock edge.

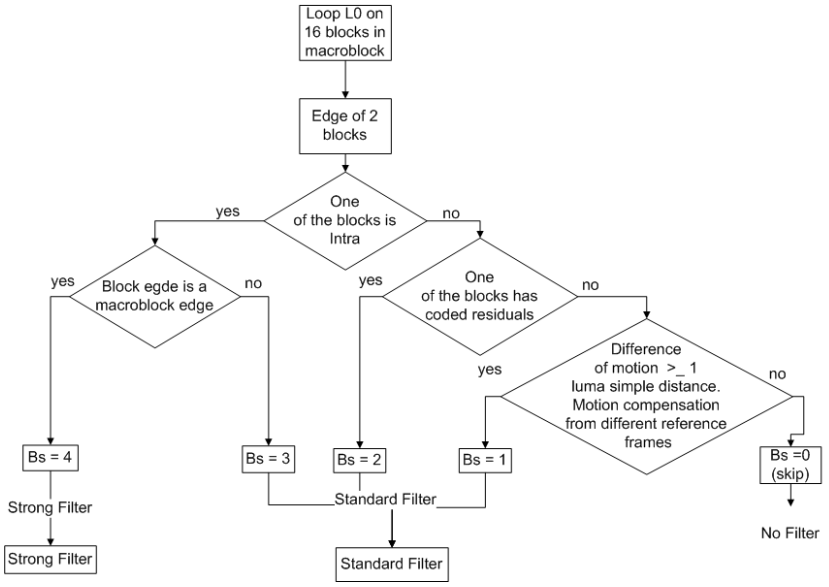


Fig. 3. Boundary strength algorithm for macroblock.

Second step is the filter module. It acquires an input macroblock from the reconstructed buffer and processes it according to the filter type decided by BS parameter. Standard and Strong filter processing are defined by mathematical equations [2] which are applied to the pixel values of a macroblock. Since the deblocking filter process is repetitive, a for-loop structure is used to process all possible directions and edges of a macroblock. More details are given in the diagram of Fig. 4.

Filter module is used for 4 edges in each direction (vertical/horizontal). In each edge, 16 pixels are filtered. In fact, a first loop is applied to repeat filter operation in vertical direction and then in horizontal one. This loop is called L1 and it is activated tow times. HOR_VER_counter parameter is the number of iteration of L1. Under L1, filtering operation treats 4 edges. L2 is the corresponding loop. The EDGE_counter parameter, initialized to three, is the counter of L2. Iteration of L2 loop consists of applied defined equations on luma and chroma pixels in both cases: Standard and Strong filters. L3 and L4 are the loops used respectively for luma and chroma pixels in Strong filter case. L5 and L6 are the loops used respectively for luma and chroma pixels in Standard filter case. Mathematical equations of both filter types are detailed in standard description [2].

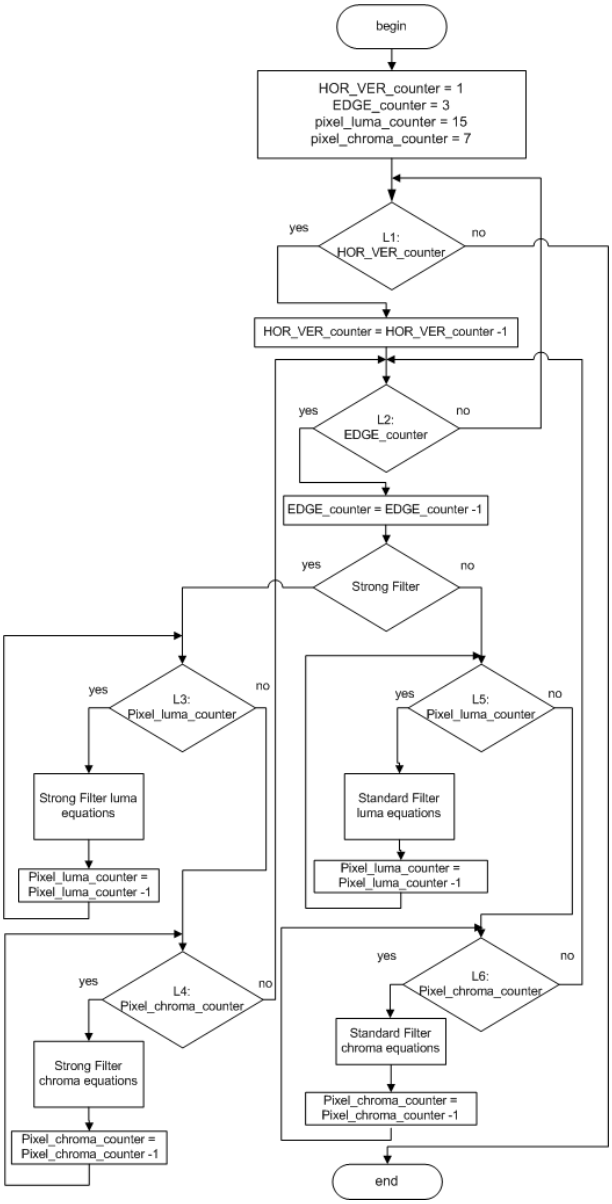


Fig. 4. H.264 decoder Profile.

As described above, the proposed implementation of the deblocking filter relies on various deterministic nested loop structures that can efficiently benefit from HLS loop unrolling optimizations. In the following, the exploitation of this using Catapult C Synthesis and the impact of unrolling loops L1 to L6 on different area and performance tradeoffs are described.

4. Design methodology

The methodology used to develop a mixed architecture is based on an efficient cooperation of different tools: Catapult C Synthesis [13], Xilinx Platform Studio [14] and Mentor Graphics Precision RTL, as shown in Fig. 5. The H.264 deblocking filter, written in C++, is the HLS entry point. A Mentor Graphics Catapult C Synthesis 2009a Release is used to automatically develop the corresponding RTL VHDL code considering the resource constraints of the target FPGA (Virtex-5 XC5VFX70T). The 400 MHz PowerPC [15] of this device is used to control the accelerator subsystem connected to the 100MHz Processor Local Bus (PLB) [16], and to compute its relative speedup over software processing. Xilinx Platfom Studio is used to generate and implement both software and hardware components. An accelerator template interface is developed by Xilinx in order to provide the RTL IP generated by HLS with a memory mapped interface compliant with the PLB protocol. Low level synthesis steps are based on Mentor Precision for the wrapped accelerator and Xilinx ISE for the rest of the platform, and the whole process leads to a bitstream file to program the FPGA and binaries for the processor subsystem. The remaining of this section is split into two sub-sections in order to detail the contributions of each tool.

4.1 Hardware accelerator design using Catapult C Synthesis

Catapult C is the central point in this methodology since accelerator and interface are the main elements for the design. Its primary contribution is the automation of RTL generation from algorithmic specifications written in ANSI-standard C/C++. Catapult also supports typical optimizations including loop merging, unrolling and pipelining which are used to quickly explore different resources and performance tradeoffs. The original source code has to comply with a few constraints. First, not all C constructs are supported by Catapult.

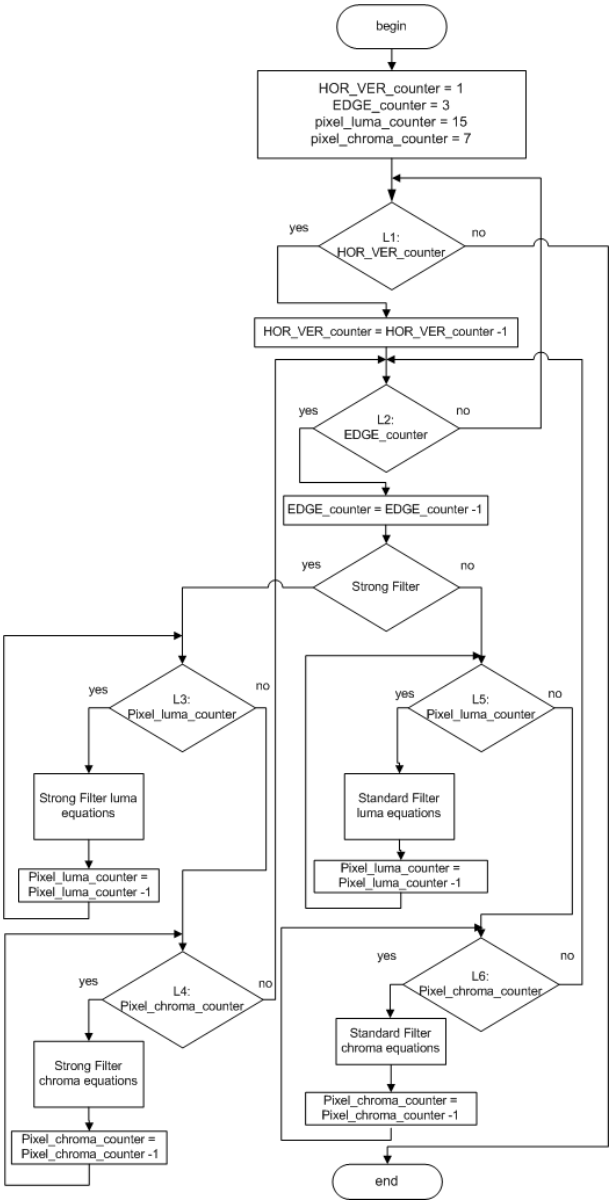


Fig. 5. H.264 decoder Profile.

Second, some C/C++ design styles can be used to expose more parallelism for the RTL architecture [17]. Finally, the software arguments of the function must cope with the accelerator interface template to be further connected to the system bus.

In a standard hardware design flow, the accelerator development starts by a manual definition of the architecture. From there, a lot of time is also needed for RTL optimization and verification. However, HLS requires only the setting of few implementation constraints to produce correct-by-construction RTL outputs. Therefore Catapult can simply operate from the specification of a clock value and the mapping of C function arguments to interface resources like memories and / or registers. Transformations such as loop unrolling and pipelining can be used to expose loop level parallelism. In the following, several configurations of loop unrolling have been used to explore different area and performance tradeoffs for the deblocking filter.

In order to transfer data in and out of the accelerator subsystem, the RTL block has to be connected to the onchip memories. In this work, input and output arguments arrays are thus mapped to single port RAM memories while other I/O scalar parameters arguments are mapped to registers. The main outcomes of Catapult synthesis are RTL VHDL, necessary libraries, hierarchical schematics and various useful metrics and reports about resources and performances of the design. Catapult RTL is then wrapped into a bus compliant module as explained in the next section. Since the resulting RTL VHDL can be complex, it is not always supported by Xilinx logic synthesis so Mentor Precision is used instead of.

4.2 Mixed hardware-software architecture design

Xilinx Platform Studio [14] is used for both hardware and software developments of the architecture on a Virtex-5 FXT device. XPS provides VHDL templates to help interfacing dedicated hardware modules (called user logic) with the PLB system bus.

This template has to be modified in order to add an instance of the Catapult generated RTL code and to connect all signals of its top level entity to the memory mapped resources of the interface template (RAM, registers). At this point, data transfers need to be optimized in order to benefit from full hardware acceleration potential. In the deblocking filter, pixels are represented as bytes of data. Without consideration, moving pixels uses only 25% of the bus speed which is 32-bit wide. Four pixels are then packed into 32-bit unsigned integer to decrease data transfers by

a factor of four. After this step, the resulting user logic can be directly connected to the Processor Local Bus in order to be accessible by the 32-bit hard processor : the PowerPC.

Although the PLB bus can be clocked at 133 MHz, it is set to 100MHz which is the maximum possible PLB frequency when the PowerPC is at 400MHz. As a consequence, the accelerator operates at 100MHz. On the software side, an accelerator driver is developed to be able to replace the original software function with a similar but accelerator-based function. All source codes are then compiled to produce PowerPC binaries and FPGA bitstream that can be finally uploaded and run on the target by Xilinx Platform Studio debug and programming tools.

5. Implementation and performances results

After completing all previous development steps, a full IP of the deblocking filter is available for test and experiment on the target FPGA. In the following, performances are measured and presented for this accelerator prototype. Next section discusses the results against other reference works.

As detailed in section 4, the 400MHz PowerPC processor and the deblocking filter accelerator are connected to the PLB bus which can run at a maximum frequency of 100MHz in this configuration. It should be noted here that is why the accelerator is limited to 100MHz. During RTL design, Catapult has been set explicitly to generate 100MHz solutions in order to be further connected to the PLB bus. So higher accelerator frequencies are certainly possible but have not been explored in the context of this work.

To get better performance, we have used an optimization feature of Catapult that allows fast exploration of loop parallelism. For each deterministic loop in the source code, an unrolling parameter can be adjusted to provide different area and performance tradeoffs. By default, all loops are not unrolled which corresponds to a low area and low performance solution (solution 1 in Tab. 1). Five other solutions corresponding to five different loop unrolling configurations have been set and fully synthesized. This process is very fast as the RTL modules generated by Catapult has all the same top level interface and can directly reuse the exact same template of section 4.2 (user logic) to comply with the target system bus. Loops in the deblocking filter are processing three scales per macroblock (direction, edge and pixel scale) for both luminance and chrominance processing. The loops over pixel scale (L3, L4, L5 and L6) are unrolled as follows: Solution 2 unrolls both chrominance loops L4

and L6. Solution 3 unrolls standard filter luminance loop L5. Solution 4 unrolls all luminance and chrominance loops (L3, L4, L5 and L6). Solution 5 unrolls the edge loop L2. Finally all loops are unrolled in solution 6 to process all loops in parallel. As observed in Tab. 1, unrolling these loops provides sensitive execution time reduction against area increase.

Table 1. Catapult loop unrolling configuration.

	Execution time (cycles)	Execution speedup	Accelerator area (Slices)	System area (Slices)
Solution 1	475	88,26 %	5167	6223
Solution 2	402	90.07 %	7235	7276
Solution 3	382	90,5 %	9070	9919
Solution 4	324	91.99 %	10253	11024
Solution 5	260	93.57 %	11972	12952
Solution 6	180	95.55 %	13440	14323

However, we are not limited by area in this case since Solution 6 which is the largest IP occupies only 40% of the target FPGA main resources (Slices). As it is also the best performance solution, it will be considered in the following section for comparison with other reference works.

6. Comparison with previous works

A comparison with state-of-the-art implementations of the H.264 deblocking filter is summarized in Tab. 2. These implementations are classified in the ascending order of throughput performance in terms of macroblock per second. Both [4, 18] works propose a processing order for deblocking filter. Parlak et al. [4] use a novel ordering edge in a macroblock to prevent unnecessarily waiting for the pixels that will be filtered become available. While design in [18] reduces the requirement of on-chip SRAM bandwidth and increases the throughput of the filter processing by making good use of data dependence between neighboring 4×4 blocks. Both architectures are described in Verilog HDL at different frequency. Architecture in [4] operates at 72 MHz when the working frequency is set to 100MHz for the implementation of [18]. Frequency of [19, 20] is also set to 100MHz. Shih and al. [19] suggest a novel filtering order and a data reuse strategy. It results in significant saving in filtering time, local memory usage, and memory traffic. Every 16×16 macroblock requires 192 filtering operations. After a few initialization cycles, one filtering operation per cycle is performed. The hardware ar-

chitecture proposed in [20] derives an integration-oriented algorithm that can be reconfigured as the in-loop or post-loop filter in deblocking filter processing. Moreover, a hybrid filtering schedule is developed to reach a lower bound of processing cycles. In particular, the filtering order is rescheduled. It reuses the intermediate pixels when the deblocking filter switches the filtered edges from vertical to horizontal direction. The proposed work doesn't achieve the highest throughput because of difference in frequency, technology and decoding filter algorithm. In fact, the proposed accelerator, [18–20] operate at the same frequency (100 MHz). An efficient comparison is justified here: the throughput of the proposed solution is better in term of the number of cycles per MB processing.

Loukil et al. [21] solution achieves a better throughput compared with the proposed solution in this work. They use a hand coded design with a well considered architecture exploiting the maximum of parallelism in the deblocking filter algorithm. Consequently, results are effective and much optimized compared to the proposed automated approach, at the expense of a very long development time. In [21] design time to market needs is not evaluate even it is a major factor in design flow. Furthermore, using HLS tools allows larger possibilities of amending and improving solutions implemented on FPGA technology. It allows an easily generation of different architectural solutions for the same accelerator. It also allows system portability which is more difficult with other tools. In fact, the reuse of the same accelerator when changing platform or technology is not always evident if design accelerator depends on target. However in HLS flows, designer has just to change the platform constraints in the input of Catapult.

Messaoudi et al. [22] implement deblocking filter by two ways which differ on the utilized data width. The first implementation solution uses 32 bits while the second one is based on 128 bits width. The second solution ensures a high degree of parallelism and avoids the use of transpose circuits and the intermediate buffers between the elementary modules in the filter. Like the proposed work, parallelization in data transfer is also considered in work of [22] . Data was concatenated to minimize time of communication between processor and hardware accelerator.

For occupied memory, the proposed implementation uses less RAM then [22]. It is also less memory consumed than [18–20]. Only [4, 21] present less memory blocks than the proposed one. But, it doesn't mean that they use less memory space, since memory can be taken from FPGA Luts and not from RAM block. In this case, synthesis results of number of gate are affected instead of number of used RAM. The synthesis results obtained in terms of equivalent gates count are presented in the fourth

line of Tab. 2. The proposed accelerator requires significantly less area than the other. It takes only 19 Kgates for worst case in term of area. In addition PowerPC is a hardware core that means no area gate is added for the processor.

Table 2. Implementation comparison.

	[4]	[18]	[19]	[20]
Technology	0.15 um	0.25 um	0.18 um	0.18 um
RAM type	Dual port	Dual port	Dual port	Dual port
RAM size	18 x21	32x64 32x96	32x32, 32x 1.5x1280x720	2x32x(1280 x720 +12)
Occupied area	-	24 Kgates	20.9 Kgates	21.1 Kgates
Processing cycles/MB	-	446	214 or 246	243
Max frequency (MHz)	72	100	100	100
Throughput (kilosMB/s)	-	224	406 or 467	411

	Proposed	[21]	[22]
Technology	65 nm	NA	LX110T FPGA
RAM type	Single port	NA	NA
RAM size	912x8	1792	256x128
Occupied area	19 Kgates	23 Kgates	7 506 LUTs
Processing (cycles/MB)	180	105	55-71
Max frequency (MHz)	100	150	171
Throughput (kilosMB/s)	555,5	1428	2400

7. Conclusion and perspectives

This paper presents a design flow using ESL tools to implement mixed hardware-software architecture of a H.264 decoder. The implementation was done on a Virtex-5 FPGA using a PowerPC processor at 400 Mhz and a hardware accelerator, the deblocking filter, at 100 Mhz. The IP was developed using Catapult C Synthesis. It provides reduced development time and allows designers to enhance performances applications

from a high abstraction level when controlling implementation steps. Implementation results are satisfactory when comparing with state-of-art. Improvements are in time of development which is sometimes at the expense of an optimal solution. But in this work performed results defends design use. In addition, many hardware solutions of deblocking filter accelerator were offered by different Catapult C configurations. The speeder one effectuated in 180 cycles with a throughput of 555,5 kilos Macroblock/second and it occuped 40% of FPGA area. The other solutions took more time execution and less area on FPGA. As perspectives, others time consuming modules of the H.264 decoder given by profiling of Fig. 1 can be accelerated using the proposed design flow. Development time can be shorter of further IP block since design flow is defined. Consequently, the global video throughput will be accelerated.

References

[1] S. Fei, S. Ravi, A. Raghunathan, and N.K. Jha. Application-specific heterogeneous multiprocessor synthesis using extensible processors. *IEEE Trans. CAD*, 25(9), 1589-1602, 2006.

[2] ITU-T Recommendation H.264 & ISO/IEC 14496-10. *Advanced Video Coding for Generic Audiovisual Services*. Version 4, 2005.

[3] Z. Wei, K. L. Tang, and K. N. Ngan. Implementation of H.264 on Mobile Device. *IEEE Trans. on Consumer Electronics*, 53(3):1109-1116, 2007.

[4] M. Parlak and I. Hamzaoglu. An efficient hardware architecture for H.264 adaptive deblocking filter algorithm. *Conf. Adaptive Hardware Syst.*, :381-385, 2006.

[5] F.Wang, Y.Xie, A.Takach. Variation-Aware Resource Sharing and Binding in Behavioral Synthesis. In *IEEE Design Automation Conf., ASP-DAC*, :79–84, 2009.

[6] E. Casseau, L. Gal, P. Bomel, C. Jego, S. Huet and E. Martin. C-based rapid prototyping for digital signal processing. 13th *European Signal Processing Conference (EUSIPCO)*, 2005.

[7] I.Werda, T.Dammak, T.Grandpierre, M.A. Ben Ayed and N. Masmoudi. Real-time H.264/AVC baseline decoder implementation on TMS320C6416 *J. Real-Time Image Processing* 7(4):215–232, 2010.

[8] M. Horowitz, A. Joch, F. Kossentini and A. Hallapuro. H.264/avc baseline profile decoder complexity analysis. *Circuits and Systems for Video Technology*, 13:704–716, 2003.

[9] S.-W. Wang, S.-S.Yang, H.-M.Chen, C.-L.Yang and J.-L. Wu. A multi-core architecture based parallel framework for h.264/avc deblocking filters. *J. of Signal Processing Systems*, 57:195–211, 2009.

[10] S.-H. Wang , W.-H. Peng, Y. He, Y. Guan, Y. Cheng, S.-C Chang, C.-N. Wang and T. Chiang. A Software-Hardware Co-Implementation of MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining. *J. of VLSI Signal Processing* 41:93–110, 2005.

[11] T.Damak, I.Werda, M.A.Ben Ayed and N.Masmoudi. An Efficient Zero Length Prefix Algorithm for H.264 CAVLC Decoder on TMS320C64. In*IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2010.

[12] I.Werda, H.Chaouch, A.Samet, M.A.Ben Ayed and N.Masmoudi. Optimal DSP Based Integer Motion Estimation Implementation for H.264/AVC Baseline Encoder. *Int. Arab J. of Information Technology*, 7(1), 2010.

[13] M. G. Datasheet. *Catapult SynthesisMentor Graphics*, Tech. Rep,2008, http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/index.cfm.

[14] EDK Reference Guide. *EDK Concepts, Tools, andTechniques*. A Hands-On Guide to Effective Embedded System Design XTP013 EDK10.1.

[15] PowerPC Processor Reference Guide. *Embedded Development Kit*. EDK 6.1 September 2, 2003.

[16] *PLB Reference Guide Processor Local Bus* (PLB) v4.6 (v1.00a), DS531 August 9, 2007.

[17] *Catapult-C Manual and C/C++ style guide*, Mentor Graphics, 2004.

[18] B. Sheng, W. Goo and D. Wu. An implemented architecture of deblocking filter for H.264/AVC. *Int. Conf. on Image Pfocessing (ICIP)*, 2004.

[19] S.-Y. Shih, C.-R. Chang and Y.-L. Lin. A near optimaldeblocking filter for H.264 advanced video coding *Asia and South Pacific Conf. on Design Automation*, :24–27, 2006.

[20] T. Ming Liu, W. Ping Lee and C.-Yi Lee. An In/Post-loop deblocking filter with hybrid filtering schedule. *IEEE Trans. on Circuits and Systems for Video Technology*, 17(7):937–943 , 2007.

[21] H. Loukil , A. Ben Atitallah and N. Masmoudi. Hardware architecture of H.264/AVC deblocking filter algorithm. *Int. Multi-Conf. Systems, Signals and Devices, IEEE*, 1–6,2009.

[22] K. Messaoudi, E. Bourennane, S. Toumi and G. Ochoa. Performance comparison of two hardware implementations of the deblocking filter used in H.264 by changing the utilized data width. *Int. Workshop on Systems, Signal Processing and their Applications (WOSSPA)*, :55–58, 2011.

Biographies

Taheni Damak received her degree in Electrical Engineering and her master in Electronic Engineering from the National School of Engineers of Sfax (ENIS), Tunisia, in 2006 and 2007 respectively. In 2007, she joined the High Institute of Electronics and Communication of Sfax, Tunisia, as an Assistant. She is a member of Sfax Laboratory of Electronics and Information Technology. Since 2011, she is an Assistant in High Institute of Computer sciences and communication techniques Hammem Sousse, Tunisia. Her current research interest is Codesign implementation of H.264 video coding standard.

Imen Werda received her degree in Electrical Engineering in 2003 and her Masters in Electronic Engineering in 2004from the National School of Engineers of Sfax (ENIS), Tunisia. In 2004, she joined the Ubvideo Tunisia Inc. as an R&D engineer. In 2006, she joined the High Institute of Technology of Sousse, Tunisia, as a technical assistant. She prepared his thesis also at the National School of Engineers of Sfax (ENIS) in 2011. She is currently a professor assistant at the High Institute of Sciences and Technological Studies of Sousse ISSATS. She is a member of the Sfax Laboratory of Electronics and Information Technology. Her current research interests include DSP and hardware implementation of H.264 video coding standard for video conference applications.

Sebastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis - CNRS. His research interests are in design, exploration and optimisation from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of french, european and international collaborative research projects.

Nouri Masmoudi received electrical engineering degree from the Faculty of Sciences and Techniques - Sfax, Tunisia, in 1982, the DEA degree from the National Institute of Applied Sciences-Lyon and University Claude Bernard-Lyon, France in 1984. From 1986 to 1990, he prepared his thesis at the laboratory of Power Electronics (LEP) at the National School of Engineers of Sfax (ENIS). He received his PhD degree from the National School Engineering of Tunis (ENIT), Tunisia in 1990. From 1990 to 2000, he is currently a professor at the electrical engineering department -ENIS. Since 2000, he has been a group leader "Circuits and Systems" in the Laboratory of Electronics and Information Technology. Since 2003, He is responsible for the Electronic Master Program at ENIS. His research activities have been devoted to several topics: Design, Telecommunication, Embedded systems, Information technology, Video Coding and Image Processing.

Microprocessors and Microsystems 38 (2014) 860–872

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Power consumption models for the use of dynamic and partial reconfiguration

R. Bonamy [a,*], S. Bilavarn [b], D. Chillet [a], O. Sentieys [a]

[a] IRISA – CNRS UMR6074, University of Rennes 1, Lannion, France
[b] LEAT – CNRS UMR7248, University of Nice Sophia-Antipolis, France

ARTICLE INFO

Article history:
Available online 1 February 2014

Keywords:
Dynamic and partial reconfiguration
Power model
FPGA
Virtex

ABSTRACT

Minimizing the energy consumption and silicon area are usually two major challenges in the design of battery-powered embedded computing systems. Dynamic and Partial Reconfiguration (DPR) opens up promising prospects with the ability to reduce jointly performance and area of compute-intensive functions. However, partial reconfiguration management involves complex interactions making energy benefits very difficult to analyze. In particular, it is essential to realistically quantify the energy loss since the reconfiguration process itself introduces overheads. This paper addresses this topic and presents a detailed investigation of the power and energy costs associated to the different operations involved with the DPR capability. From actual measurements considering a Xilinx ICAP reconfiguration controller, results highlight other components involved in DPR power consumption, and lead to the proposition of three power models of different complexity and accuracy tradeoffs. Additionally, we illustrate the exploitation of these models to improve the analysis of DPR energy benefits in a realistic application example.

1. Introduction

Run-time reconfiguration, i.e. the ability to modify hardware execution resources to ensure specific functions arrangement during execution, has been a promising field of research since the 1990s [1]. The technology, now referred to as Dynamic and Partial Reconfiguration (DPR), is fully operational and available in up to date devices from the two major FPGA manufacturers, Xilinx and Altera. Run-time reconfiguration allows sharing a piece of silicon area for the implementation of different hardware accelerators when tasks are sequentially executed. This results in significantly less FPGA resources for reconfigurable processing.

Whereas many tools are available for the actual management and programming of DPR functionalities, there are comparatively few methodologies helping to explore and evaluate these benefits. However, the costs of reconfiguration and hardware implementation of tasks involve complex issues that need to be well understood to determine whether or not the gains exceed the costs. Considering for example an application as a set of tasks with possible hardware implementations, different execution scenarios will influence the actual performance, area and energy tradeoff of a solution: (i) use static hardware tasks, (ii) use dynamic hardware tasks and reconfigure when necessary, (iii) use software tasks and (iv) use a mix of software and hardware tasks, possibly statically or dynamically configurable, possibly at different cost-performance tradeoffs for hardware tasks. In order to assess this very large opportunity of choices, fast and accurate models of DPR reconfiguration must be defined, especially in terms of power and energy. This paper addresses this problem and investigates the definition, development and use of such models that can be able firstly, to explore combinations of task implementations, and secondly to provide the associated schedule that will allow minimizing the energy consumption. Therefore, providing reliable power models is the main contribution of this work which is based on actual experimentation of the DPR process in Xilinx devices. Fine measurements on a Virtex5 FPGA have led to define three models of different accuracy/complexity tradeoffs. Additionally, we also address an application study of the proposed models on a representative real life example to show their usefulness in early design space exploration for energy efficiency.

The outline of the paper is the following. First, we present the context of this work with a state of the art on power consumption related to the DPR ability, an introduction to FPGA architectural features for DPR and the experimental setup developed for accurate

* Corresponding author.
E-mail addresses: robin.bonamy@irisa.fr (R. Bonamy), sebastien.bilavarn@unice.fr (S. Bilavarn), daniel.chillet@irisa.fr (D. Chillet), olivier.sentieys@irisa.fr (O. Sentieys).

http://dx.doi.org/10.1016/j.micpro.2014.01.002
0141-9331/

power measurements. Extensive measurement results are then analyzed in detail in Section 3, and exploited in Section 4 to define different models for power estimation that are further validated in Section 5. The following Section 6 proposes a case study of these models in the design space exploration of a HW/SW implementation of a H.264/AVC profile video decoder application. Finally, Section 7 summarizes the main conclusions and presents next directions of research.

2. Work context

2.1. State of the art

DPR is a technique enabled in reconfigurable hardware devices like FPGAs to improve their processing flexibility. Indeed their configuration can be changed during execution according to user constraints or environmental needs [2]. This run-time tasks configuration ability comes with various opportunities for energy saving. First, dedicated hardware allows the definition of optimal implementations in terms of processing and energy efficiency. DPR can then be widely used to increase the resource usage [3], further reducing the size of reconfigurable units and the associated static power consumption. Dynamic reconfiguration also lets the modification of clock configuration, i.e. it provides dynamic frequency adaptation and variation of performances, to adjust power consumption and energy efficiency on demand [4]. In addition, DPR can also be used to disable the routing of clock signals to some of the FPGA resources, thus to implement a low overhead clock gating technique with interesting results [5].

An important drawback of DPR in most applications is the unavailability of the region involved during the reconfiguration process. Limiting the reconfiguration time is thus a critical requirement that can be addressed with different techniques. A first one is to improve the reconfiguration speed to reach the maximum reconfiguration port throughput [6]. For example, using higher performance memory for bitstream storage (DDR) and DMA can greatly reduce memory access times to the bitstream. Another solution is to reduce the size of the configuration data, based for instance on coding only the differences with previous configuration instead of the completely new configuration information. In any case, the overhead of reconfigurations is important to consider in the design process. This has been shown in many recent works [7,8], but reconfiguration delays are not the only issue. The energy cost of DPR is also an important design parameter to manage, especially to ensure actual energy gains from its utilization. We can find a few studies on the power consumption of dynamic reconfiguration in the literature like [9,10] and more recently [11]. These works generally address the reconfiguration controller and throughput optimization but do not provide a thorough analysis and modeling of power consumption during partial reconfiguration.

The fine investigation of power and energy consumption and modeling during DPR is the main focus of this paper. This contribution is part of a more global design space exploration methodology developed in the context of a platform project focusing on power measurement, estimation and optimization for heterogeneous hardware–software computing systems [12]. In the following, we detail the elaboration of power estimations at different accuracy and complexity tradeoffs from actual and fine power measurements. We also show the applicability and usefulness of the proposed models in adapting the level of estimation complexity to the best suited accuracy imposed by the application analysis. It is then demonstrated how these models greatly help the analysis of difficult implementation choices including static hardware tasks, software execution and dynamic reconfiguration using the design space exploration methodology mentioned previously.

2.2. FPGA architecture and partial reconfiguration

Effective DPR is available in Xilinx FPGAs since the VirtexII pro series and more recently in Altera Stratix V devices. This work is based on Xilinx technology and targets more specifically the Virtex5 XC5VLX50T device of a ML550 Evaluation platform for power measurement reasons. A view of the corresponding layout is presented in Fig. 1. This FPGA is organized in six clock domains where each clock domain is composed of several frames. These frames are grouped in columns containing either CLBs (slices), DSPs, BRAMs or other specific blocks. The number of frames in one column is fixed by the FPGA architecture and is dependent on the type of these

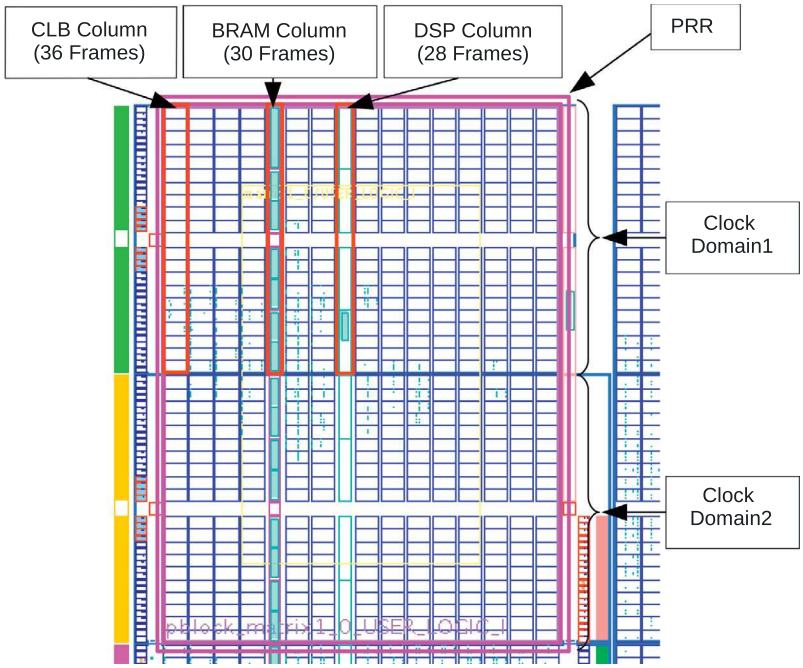


Fig. 1. Top left hand of Virtex-5 XC5VLX50T FPGA layout (Xilinx PlanAhead 12.1).

columns. The minimum addressable reconfiguration area is a frame whose configuration requires 41 words of 4 bytes (164 bytes). The minimum recommended reconfiguration area is a column, so a Partial Reconfigurable Region (PRR) must be a multiple of the number of frames and mainly contains CLBs, DSPs and BRAMs.

Fig. 2 represents the organization of the configuration file (bitstream) to configure the PRR example of Fig. 1. The first few words represent the header containing information on the bitstream and configuration startup procedure. The next word is the address of the first frame to configure, followed by the configuration words for the CLB frames. After four CLB columns comes the successive configuration of a BRAM column, two CLB columns and a DSP column. The configuration of this first clock domain ends with CLB data up to the right bound of the PRR. The second clock domain starts with another frame address corresponding to the first column and follows the same column pattern (CLB, BRAM, CLB, DSP). Finally the BRAM content is addressed and a few additional words end the configuration file.

2.3. Experimental setup

2.3.1. Power measurement

A Virtex-5 LXT ML550 Networking Interface & Power Measurement Platform [13] is used in the following experiments. The choice of this platform was motivated by the availability of built-in resources that greatly helps analyzing power consumption. Indeed five connectors are present on the board to access the currents consumed by the FPGA and its peripherals. FPGA power measurement is based on monitoring the power rail of the core. As represented in Fig. 3, a high-precision amplifier is used to improve the signal level which is then sent to a digital oscilloscope. With this procedure, it is possible to measure power values as low as 0.1 mW. As it will be shown, this precision is sufficient to clearly identify the different steps of a dynamic reconfiguration process and derive accurate power models.

2.3.2. Platform setup

The ML550 platform is configured using the Xilinx Reference Design described in [14]. The system is composed of a `MicroBlaze` processor, a `Compact-Flash` memory controller and a `xps_hw_icap` reconfiguration controller, as shown in Fig. 4. The `MicroBlaze` processor and the reconfiguration controller operate both at 100 MHz. Reconfiguration requests are managed by the `MicroBlaze` which reads a configuration bitstream from the `CompactFlash` and sends it to the `xps_hw_icap` to apply the corresponding configuration to a PRR.

Since different tasks are used in the following experiments, one PRR fitting the area of the biggest task is defined. We selected a PRR that holds in the top left hand of the FPGA layout as shown in Fig. 1, and occupies the complete area over two clock domains. Doing this

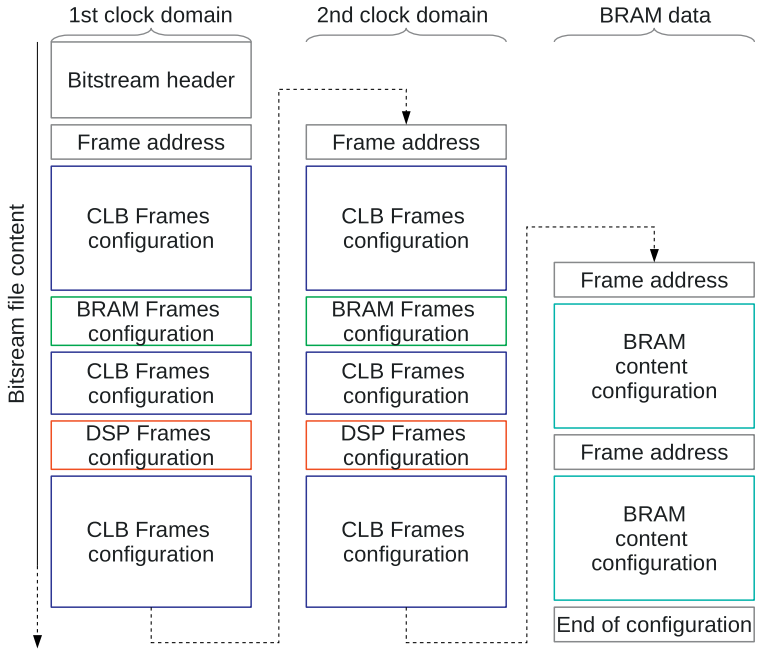


Fig. 2. Bitstream composition to configure a PRR (Xilinx ISE 12.1).

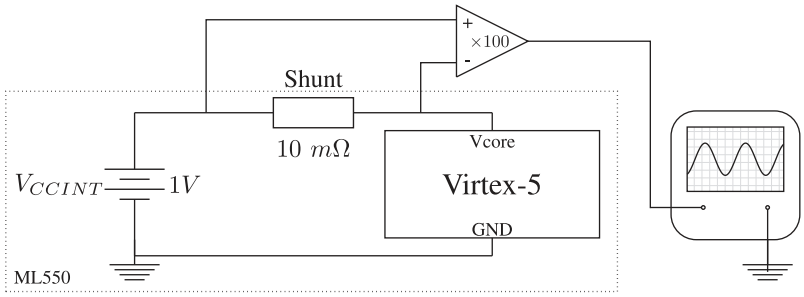


Fig. 3. Power measurement procedure using a ML550 platform, high-precision amplifier and digital oscilloscope.

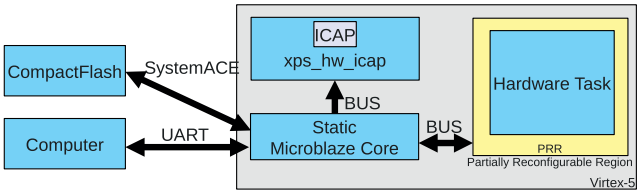


Fig. 4. Reference Design used during measurement procedure.

corresponds to a configuration bitstream with a fixed size of 227,700 Bytes. During a measurement process, the FPGA core is powered at 1 V and we make sure that the die temperature keeps stable around 35 °C in order to focus on power consumption variations only due to the DPR. As it is visible for example in Fig. 5, we can clearly detect the distinct phases of the reconfiguration process thanks to this procedure.

2.3.3. PRR and tasks

Power measurements are made using previous PRR that can be configured with three different tasks. The first task T_1 is a matrix multiplication. The second task T_2 is a parallel version of the matrix multiplication. The two RTL implementations of the matrix multiplication are derived from High Level Synthesis with different loop unrolling settings [15]. They correspond to a sequential and a parallel solution which FPGA resources are given in Table 1. The third task is called a *blank* bitstream and will be referred to as T_{blank} in the following. A *blank* bitstream corresponds to the configuration of an empty PRR. This is useful to clear the configuration of a PRR when it is unused and thus to reduce the associated power consumption [16]. In the following, we analyze in detail how configuring from one task to another impacts the FPGA core power consumption.

3. Power consumption analysis

The overall DPR process consists mainly in transferring data to the different components involved (`CompactFlash`, `MicroBlaze` local BRAM memory, `xps_hw_icap` reconfiguration controller and configuration memory). The full procedure can be split into the following basic steps. The configuration data are successively:

- Loaded from a file stored on the `CompactFlash` and buffered to the `MicroBlaze` local memory.
- Written from the `Microblaze` local memory to the `xps_hw_icap` reconfiguration controller.
- Written from the `xps_hw_icap` to the Internal Configuration Access Port (ICAP).
- Written from the ICAP to the configuration memory.
- Applied from the configuration memory to the configurable resources (CLB, BRAM, DSP, interconnect).

Under these circumstances, the global reconfiguration power is the result of a combination of two main elements: (i) reconfiguration control involving mainly configuration data accesses and (ii) the actual configuration of the FPGA resources.

Table 1
FPGA resources and idle power of tasks used for measurements.

Task	BRAMs	Slices	DSPs	Idle power (mW)
T_1	8	169	2	26
T_2	8	154	1	24
T_{blank}	0	0	0	0
FPGA without task				402

3.1. Configuration data access

Top of Fig. 5 reports the power profiles of the FPGA core during DPR from T_2 to T_1 (black), and from T_1 to T_2 (gray). If we inspect closely the full reconfiguration process in this figure, we can observe the power variations due to the DPR procedure. The bottom plot of Fig. 5 is a zoom on the first 10 ms of DPR. To provide even finer power analysis, we have introduced software triggered

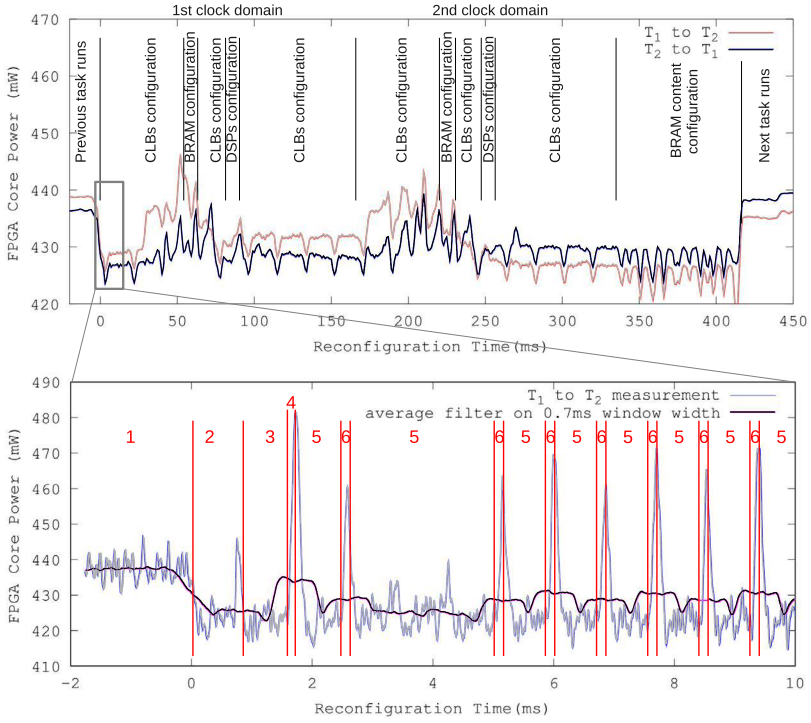


Fig. 5. Top plot is the FPGA core power consumption during DPR from T_2 to T_1 (black) and from T_1 to T_2 (gray). Bitstream composition is scaled to the reconfiguration time to outline the reconfiguration steps. Bottom plot is a zoom on the first 10 ms of DPR.

markers in the reconfiguration control to highlight each step of the process. Seven steps are thus clearly identified and presented in the sequence diagram of Fig. 6:

1. Receive reconfiguration order.
2. Open bitstream file on the `CompactFlash`.
3. Read bitstream file header.
4. Check validity of bitstream file header.
5. Read file fragment on the `CompactFlash`.
6. Write data in `xps_hw_icap`.
7. Repeat steps 5 and 6 until the end of file.

We can notice that the lowest levels in the power profile of Fig. 5 (bottom plot) correspond to read accesses to the `CompactFlash` (phases 3 and 5). Due to the relative slowness of the `CompactFlash`, the `MicroBlaze` processor (which has the largest part of FPGA power) is mostly waiting for data during these accesses and does not generate a lot of activity. However writing implies much more work (phase 6) involving bus traffic, reconfiguration controller activity and FPGA resources configuration. The corresponding power overheads are about 45 mW which is more than 10% of the overall FPGA power consumption. There is another power overhead during bitstream validity check (phase 4). This overconsumption is also significant as it is processed by the `MicroBlaze` processor. Note that these power peaks are not visible on top of Fig. 5 because of an average filtering scheme used for display convenience. Finally reading file fragments and writing configuration data to the `xps_hw_icap` are repeated up to the end of the bitstream.

3.2. Configuration application

Previous zoom highlighted the influence of configuration data transfers on power consumption. Nevertheless, the effects of these read and write operations should not bring significant variations on global power profiles (top of Fig. 5). Indeed, if we compute an average sliding window over 0.7 ms (black curve, bottom of Fig. 5), the resulting profile is very regular and not far from constant. However, global power profiles are clearly not constant over the full duration of the reconfiguration process. Firstly, two power overconsumptions located around 50 and 200 ms are present and look like two "waves". Secondly, power levels at the beginning and at the end of the reconfiguration are different. This power difference seems to be related to the transition from previous to the next task.

These two effects indicate that power profiles also depend on other parameters than configuration data accesses. We assume that the observed power variations are also due to the bitstream contents and we separate two kind of effects: power surges and power steps. A reconfiguration can cause overconsumptions due to the activity resulting from differences with previous configuration (power surges). Then, configuration application enables or disables signals and blocks that can cause power consumption breaks (power steps). The following sections detail the analysis of power profiles that justifies these assumptions.

3.2.1. Power surges

In the power profiles of Fig. 5 showing the reconfiguration of T_2 from T_1 (gray) and T_1 from T_2 (black), both curves have the same overall shape which is mainly driven by the write operation in the reconfiguration memory. Practically, when a task is configured over a previous task, reconfiguration consists in re-writing data on the existing content of the configuration memory. If both nth words in the bitstreams of previous and next tasks are the same, the memory cells do not change and the power consumption is low. Inversely, if the two words in the bitstreams are exactly complementary, each bit of the memory cell changes and this leads to a larger power consumption. From this observation, we assume that

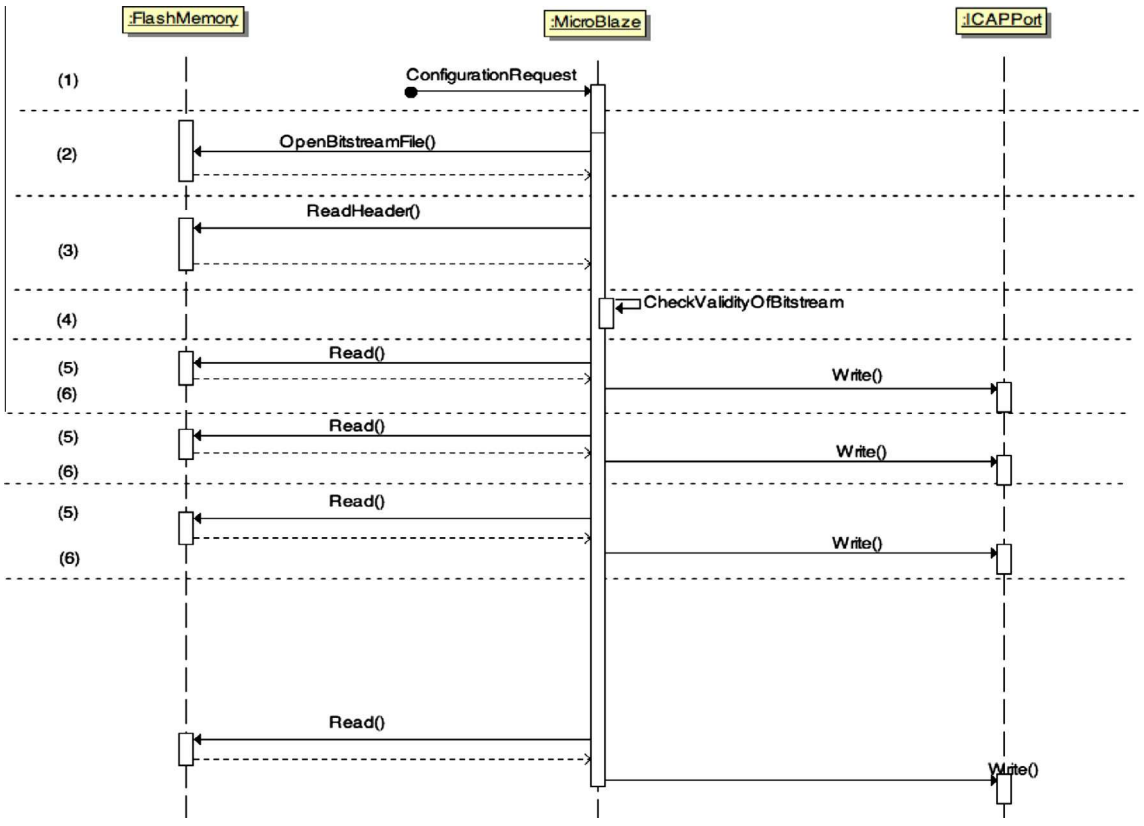


Fig. 6. Sequence diagram of the reconfiguration process.

power consumption during the reconfiguration process is linked to the differences between the bitstreams of previous and next task configurations. These differences can be quantified by a metric like the Hamming distance.

On both curves of Fig. 5, two remarkable zones are present: the first one is located around 50 ms and the second one around 200 ms. The amplitude of these overconsumptions is about 15 mW. Further analysis of the bitstream content shows that these two zones correspond to the configuration of two BRAM frames. The BRAM frames belong in two distinct clock domains which explains the presence of two overconsumption zones. By correlating this information with the design view of Xilinx floorplanning tool (PlanAhead), we can notice that the density of slices is higher close to the BRAM frames. This placement of logic resources around the BRAM blocks can be explained by the Place and Route algorithms which probably try to group the resources to limit the cost of interconnections.

Top of Fig. 7 shows the power profile of the PRR reconfiguration from T_1 to T_2. Bottom of this figure shows the Hamming distance per configuration word of 32-bit between the two corresponding bitstreams. Abscissa values are adjusted to tie in the reconfiguration time (ms). We can notice on these curves that the shape of the Hamming distance is highly correlated to the shape of the power consumption profile. The Hamming distance peaks at the same time when the overconsumptions are present, around 50 ms and 200–250 ms. This tends to confirm the link between configuration differences and overconsumption values (power surges).

These results suggest that overconsumptions result from differences between previous and next configurations, which in turn correspond to the activation and deactivation of interconnects between FPGA resources. This may cause unwanted connections and perhaps small short circuits while the PRR is not fully configured, as suggested by [10] for an ATMEL device. Modifications of the FPGA interconnect are indicated in the bitstream and reflected in the Hamming distance.

3.2.2. Power steps

Looking more closely at Fig. 5 reveals that the power levels at the beginning and the end of the reconfiguration are different. This difference is coherent with the power consumption of tasks presented in Table 1 where the global power is 402 mW for an empty FPGA, and 428 mW or 426 mW respectively when tasks T_1 or T_2 are configured. T_1 and T_2 do not have the same idle power consumption which depends on their size. Logically, using two tasks with more idle power differences should emphasize this observation. As an illustration, Fig. 8 presents the power profile during the partial dynamic reconfiguration of a PRR from T_{blank} to T_2, where T_{blank} is an empty or blank task as defined previously. This figure explicitly shows two power steps at 55 ms and 220 ms. These steps raise progressively the power level from the idle power of previous task T_{blank} to the idle power of next task T_2. Fig. 8 also highlights the corresponding bitstream composition of the Virtex5 device which reveals that two steps appear just before the configuration of BRAMs. This behavior is typical of activating or deactivating elements and suggests that there is probably a link with the power consumption of BRAM memories and their associated active state.

In this section, we have identified the main elements involved. First there is a contribution from reconfiguration control involving mainly configuration data accesses, and second from the actual application of a configuration. In the actual configuration application, power surges are present because of differences between previous and next configuration, and power steps result from the activation of new resources for the next task configuration. From these contributions, the following section defines three power models of DPR with different accuracy levels.

4. Power consumption estimation

4.1. Coarse grained DPR model

The easiest way to estimate the power consumption of a PRR reconfiguration is to record measurements of multiple reconfigurations and to consider the average value. Idle power is subtracted from this value before reconfiguration to keep only the component related to the reconfiguration. It can thus be considered as the power consumption resulting from the control of the reconfiguration and will be referred to as $P_{control}$ in the following. The FPGA idle power consumption before reconfiguration is defined by the FPGA idle power consumption when the configuration of the PRR is cleared (*blank*): P_{FPGA}. The idle power P_{prev} of the PRR configured previously is obtained by subtracting P_{FPGA} from the global FPGA idle power measurement when T_{prev} is configured on the PRR. In these conditions, the corresponding coarse grained power model is given by the following expression:

$$P_{CG} = P_{FPGA} + P_{prev} + P_{control} \quad (1)$$

This model has the advantage of being simple to setup, but deviations can occur when there is a significant difference of idle power between the previous and the next task configured on a given PRR.

4.2. Medium grained DPR model

An improved model can be defined by an interpolation between the idle power before and after the reconfiguration. This requires to

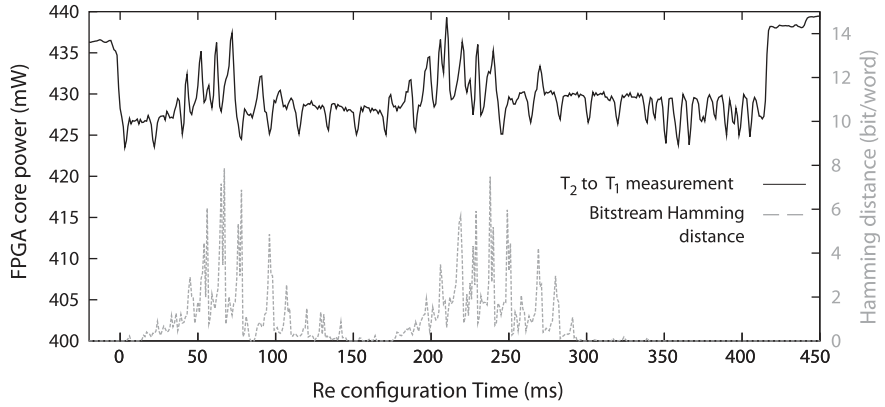


Fig. 7. Power consumption during DPR from T_2 to T_1 (straight black line) and Hamming distance per configuration word of 32-bit in T_2 and T_1 bitstreams (dashed gray plot).

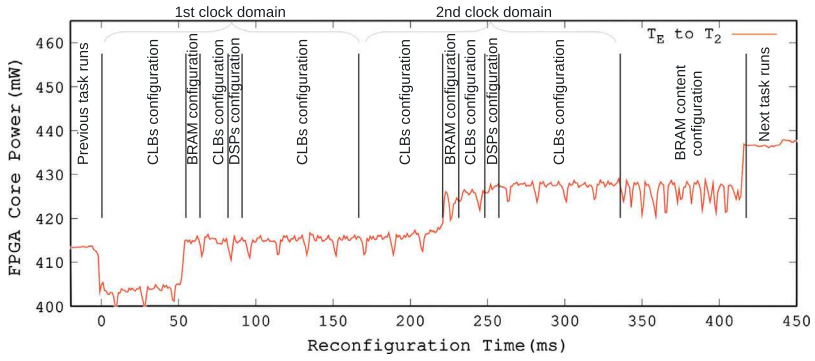


Fig. 8. Power consumption during DPR from T_{blank} to T_2 and corresponding bitstream composition.

know the idle power of previous and next tasks configured, in addition to the power of the reconfiguration control. The resulting linear equation is given in the following:

$$P_{MG}(\tau) = P_{FPGA} + P_{prev} + P_{control} + (P_{next} - P_{prev}) \times \frac{\tau}{BS_{size} \times T_{word}} \forall\ 0 < \tau \leqslant BS_{size} \times T_{word} \quad (2)$$

where τ is a time unit corresponding to the read access of a configuration word of 32-bit, BS_{size} is the bitstream size in Bytes of the PRR configured, T_{word} is the time required to process a configuration word of 32-bit and P_{next} is the idle power consumption of the PRR with the new task. The estimation error should be less important than previous model since the idle powers of tasks are present. However the model does not consider the bitstream content which is responsible of power steps as explained in Section 3.2.2. In the next model, this can be addressed by the Hamming distance.

4.3. Fine grained DPR model

In this section, the reconfiguration model is based on the same previous parameters, but the profile defined in this case evolves by steps which are dependent on the bitstream content. In addition, power surges are also considered with the Hamming distance reflecting the differences between configuration words of both bitstreams. The resulting model involves advanced parameters whose determination is further detailed in Section 5.1.

The resulting model is defined by the following equation:

$$P_{FG}(\tau) = P_{FPGA} + P_{prev} + P_{control} + steps(\tau) \times (P_{next} - P_{prev}) + \alpha \times d_{Hamming}(\tau, T_{prev}, T_{next}) \quad (3)$$

where $steps(\tau)$ is a function whose result is between 0 and 1 depending on the FPGA device and PRR size. In our measures, this function is linked to the position of BRAMs. This value is computed from the PRR resources, bitstream size and reconfiguration speed. Multiple intermediate values are possible depending on the PRR composition.

$d_{Hamming}(\tau, T_{prev}, T_{next})$ is the Hamming distance between previous and next configurations of a PRR, applied on configuration words of 32-bit. Finally α is a coefficient adjusted for a given FPGA device, reflecting the weight of the Hamming distance in power consumption.

This section has presented three power models with growing accuracies progressively improving the matching with the actual dynamic reconfiguration power. In the following section, we set the parameters of these models for the FPGA and task setup described in Section 2.3 in order to compare the estimation accuracy with actual measurement results.

5. Model validation

5.1. Model calibration

Under the experimental conditions presented in Section 2.3, this section details how the values of parameters in the different models are measured or computed to enable power estimation.

5.1.1. Coarse grained DPR model

P_{FPGA} is the FPGA idle power consumption with an empty PRR (402 mW). P_{prev} and P_{next} are the idle power of the PRR before and after the reconfiguration, they depend on the tasks involved T_{prev} and T_{next}. These values are measured on the FPGA and the corresponding results come from Table 1. $P_{control}$ is the average extra power required for the reconfiguration control to transmit reconfiguration data from the storage memory to the configuration memory through the ICAP port. This power has been measured by averaging power measurements of multiple reconfigurations of the same blank task to avoid power variations due to configuration differences. The corresponding power consumption is $P_{control} = 20$ mW.

5.1.2. Medium grained DPR model

Two additional parameters are used in the medium grained model: BS_{size} and T_{word}. BS_{size} is the size of the reconfiguration bitstream for the PRR considered, in this case 227,700 Bytes. T_{word} is the time needed to configure one word (4 Bytes) of the bitstream. Its actual value is 7.4 ms which is derived from the reconfiguration time of the bitstream and its size:

$$T_{word} = \frac{T_{Reconfiguration}}{BS_{size}} \times 4 \quad (4)$$

5.1.3. Fine grained DPR model

The fine grained model expressed in (3) requires three more parameters: $d_{Hamming}(\tau, T_{prev}, T_{next})$, α and $steps(\tau)$.

$d_{Hamming}(\tau, T_{prev}, T_{next})$ is the function that returns the Hamming difference computed word per word between the previous configuration and the next configuration data. This value is filtered with a sliding window average over 100 words. This average is required because the same difference between two words does not necessarily result in the same overconsumption (depending on the type of resource under configuration in the bitstream position). Thus, the average of the Hamming distance should be performed on at least one frame to allow a better estimation of the real power consumption required during reconfiguration. However, this averaging process does not include the end of the bitstream as it represents exclusively BRAM data. $d_{Hamming}(\tau, T_{prev}, T_{next})$ returns 0

in this case since BRAM content configuration does not have a significant power contribution.

α is a technology dependent parameter used to weight the Hamming distance. This parameter is determined with an iterative optimization algorithm to minimize the average absolute power error between the estimated power and actual measurements. Considering T_2 to T_1 configuration, the optimal value for α is 2.98 mW.

Finally, the determination of the power step function $steps(\tau)$ is derived from the FPGA layout. Presented in Section 2.3.2, the PRR used occupies two columns of BRAMs in two distinct clock domains. As seen in Section 3.2.2, power steps occur before the BRAM interconnect configuration, so there are logically two power steps equally shared in this case. The position of these steps correspond to the beginning of BRAM configuration frames. Since four CLB columns precede the first BRAM configuration frame, and considering that CLB column requires 36 frames and one frame is 41 words [17], the first step is located at the 5904th word ($41 \times 36 \times 4$) of the bitstream. The second step is at the same position in the second clock domain, i.e. at the 31,898th word of the bitstream which ends at the 56925th word. The resulting function *steps* is defined as follows:

$$steps(\tau) = 0 \quad \forall \tau \in [0; 5903]$$

$$steps(\tau) = 0.5 \quad \forall \tau \in [5904; 31,897]$$

$$steps(\tau) = 1 \quad \forall \tau \in [31,898; 56,925].$$

The power profiles resulting from these models are represented in Figs. 9–12. Straight lines represent the measured power profiles. "×", "+" and "o" marked lines are the estimated power traces of the coarse, medium and fine grained models respectively. These results are further analyzed in terms of energy and power accuracy in the following sections.

5.2. Accuracy of energy estimations

Four reconfiguration scenarios are considered to evaluate the accuracy of energy estimations: reconfiguration from T_1 to T_2, from T_2 to T_1, from T_1 to T_{blank} and from T_{blank} to T_2. The accuracy of each model is evaluated with regard to energy consumption by computing the average error and root mean square error (RMSE), reported in Table 2.

Since we consider the same PRR, reconfiguration times are the same in the four considered cases and energy is comparable to the average power. As visible in the power profiles of T_1 to T_2 (Fig. 9) and T_2 to T_1 (Fig. 10), the coarse grained and medium grained models perform well with less than 14% error on energy. The error is low in this case because the difference of idle power between T_1 and T_2 is fairly limited (2 mW). If we consider higher differences of idle power, e.g. reconfiguring from T_1 to T_{blank} (Fig. 11) or from T_{blank} to T_2 (Fig. 12), the coarse grained model is significantly more inaccurate with 77% and 68% of estimation error on energy. Most of this inaccuracy comes from the fact that the coarse grained model does not consider the idle power of previous and next tasks. However, the medium grained model limits this estimation error to 20% by considering the idle power of previous and next tasks on the given PRR. Finally, the fine grained model provides the best estimation results with less than 6% of error in all cases, by including power surges and power steps to the model. We can thus observe in Figs. 9–12 a close match between the power estimated by the fine grained model ("o" round marked line) and the real measured power (straight line).

5.3. Accuracy of power profiles

In addition to the accuracy of energy estimations presented above, the ability of power profiles to predict a correct behavior is also an important quality criterion for the models. The variations of power models from actual power measurements can be indicated by computing the differences of the models with reality on each point of the profiles and taking an average. These variation values are presented in Table 3 for the three models and all reconfiguration scenarios.

As expected, the results show that the coarse grained model has an important deviation of 6.5 mW, which is 17.6% of the average reconfiguration power. This error comes from not considering power surges and power steps. The medium grained model reduces the average deviation to 4.05 mW (10.5%) by taking into account roughly the power steps. Finally, the fine grained model is slightly better with an average deviation of 3.75 mW which is 9.6% of the average power of the reconfiguration. It might be noted here that there is an over-estimation of power surges in the computation of the Hamming distance in this case. This is caused by the dissymmetry of power levels resulting from the configuration involving T_{blank} that can be clearly seen in Figs. 11 and 12. Moreover, finer variations of power are not estimated and contribute to increase the error. However, most peak values are present and the general power trend is globally well estimated.

5.4. Relevance of DPR models

The three models proposed for DPR are useful to adjust the estimation accuracy to the level of analysis and detail needed. In all cases, the fine grained model is preferable in terms of precision, but it is not always the best choice in practice due to its high

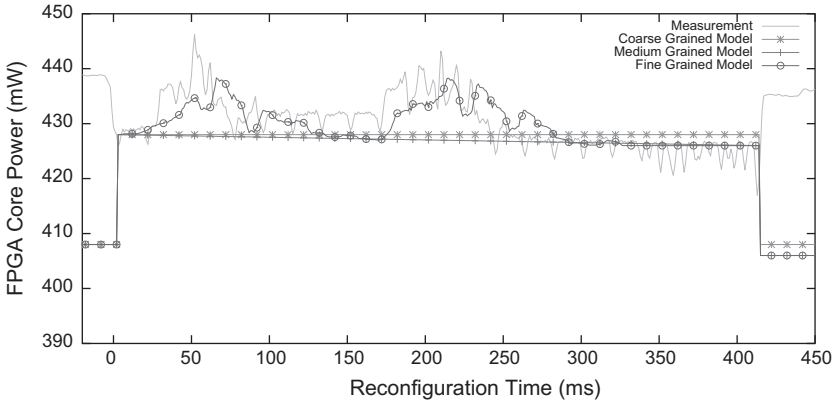


Fig. 9. DPR models vs. measurements for the reconfiguration of T_1 to T_2.

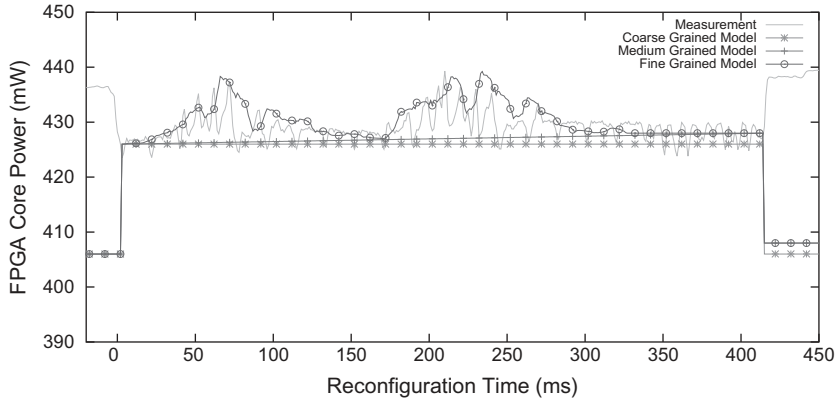


Fig. 10. DPR models vs. measurements for the reconfiguration of T_2 to T_1.

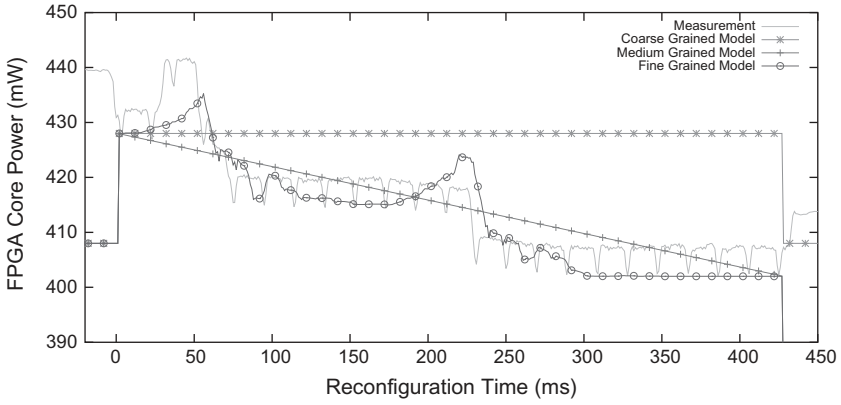


Fig. 11. DPR models vs. measurements for the reconfiguration of T_1 to T_{blank}.

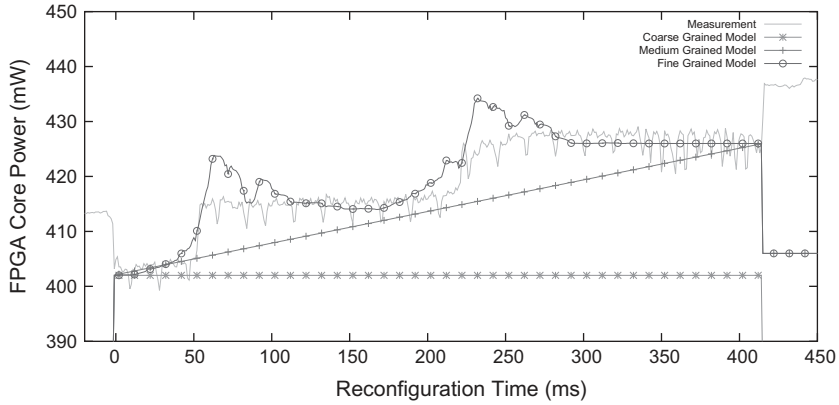


Fig. 12. DPR models vs. measurements for the reconfiguration of T_{blank} to T_2.

Table 2
Average standard error and root mean square error of energy estimations for the coarse grained (CG), medium grained (MG) and fine grained (FG) models compared to real measures (M).

Config.	M (mJ)	CG error (mJ)	MG err. (mJ)	FG err. (mJ)
T_2 to T_1	9.12	−1.3 (−13.7%)	−0.84 (−9.2%)	0.5 (5.5%)
T_1 to T_2	9.58	−0.89 (−9.2%)	−1.3 (−13.5%)	−0.22 (−2.3%)
T_1 to T_{blank}	7.97	6.2 (77%)	0.59 (7.4%)	−0.13 (−1.7%)
T_{blank} to T_2	10.41	−7.1 (−67.9%)	−2.1 (−19.7%)	0.5 (5%)
Average	9.27	−0.77 (−8.3%)	−0.9 (−9.8%)	−0.16 (−1.8%)
RMSE		4.77 (51.5%)	1.34 (14.5%)	0.38 (4.0%)

Table 3
Power profiles deviation for coarse grained (CG), medium grained (MG) and fine grained (FG) models, compared to measurements.

Config.	CG (mW)	MG (mW)	FG (mW)
T_2 to T_1	2.7 (5.8%)	2.7 (5.8%)	3.2 (6.9%)
T_1 to T_2	4.9 (10.2%)	4.6 (9.5%)	3.8 (7.9%)
T_1 to T_{blank}	10 (31.8%)	5 (16%)	4.3 (13.5%)
T_{blank} to T_2	8.2 (22.4%)	3.9 (10.7%)	3.7 (10%)
Average	6.5 (17.6%)	4.05 (10.5%)	3.75 (9.6%)

elaboration complexity (Hamming distance, step function and calibration parameters). In addition, the accuracy of the fine grained model is not necessary in many cases. Especially, fine power details become secondary when the reconfiguration overhead is optimized and low compared to execution times and power of hardware tasks (which is also a requirement for actual DPR benefits). Therefore the choice of a model is driven by (i) the availability of a fine grained model for a device or technology, and (ii) a tradeoff between accuracy and elaboration complexity. In the absence of a fine grained model, coarse grained and middle grained models are easier to use and relevant enough to provide fair energy estimations and power profiles in a large number of cases. Next section will illustrate the practical consideration of two of these models for a realistic application example.

It is important to stress that the performance of a reconfiguration controller has also a very large impact on the relevance of DPR in a real design. In the proposed experimental setup of Section 2.3 based on the Xilinx Reference Design, reconfiguration performance is dependent on the use of Xilinx `xps_hw_icap`, `MicroBlaze` and `CompactFlash`. Higher reconfiguration performance can be reached using an optimized framework such as [18,11]. In [11] for example, reconfiguration is significantly faster but power variations are very difficult to detect in this case because of the low level of currents involved and bandwidth limits of current probes. Measurement is the main reason why a standard Xilinx setup is used in this study. However, we have also developed optimized DPR controller IPs under closely related works and we have a precise knowledge of the mechanisms and techniques used. The steps involved in an optimized DPR process are essentially the same, therefore the same power models apply. The use of an optimized controller is thus expected to affect mainly the parameters of equations and depends on the DPR model concerned. For the coarse grained model, power of the controller ($P_{control}$) is the only parameter impacted in Eq. (1), performance (T_{word}) has also to be modified for the medium grained model in Eq. (2). When considering the fine grained model of Eq. (3), $P_{control}$ is affected. In addition, faster reconfiguration could also decrease peaks of supply current due to the capacitance effect of power rails. This can in turn impact the Hamming distance and require an adjustment of the sliding window used for computation, to correct the amplitude of power peaks. Considering this, a relative deviation can occur for optimized controller models but in this case, this would mainly relate to the fine grained model.

6. Application study

6.1. Overview

6.1.1. AADL exploration and modeling framework

In this section, we show the usefulness of previous DPR models for the analysis of a relevant application example. The design framework used for this purpose is a system level approach based on the Architecture Analysis and Design Language (AADL [19]) developed in the scope of the Open-PEOPLE platform project [12]. In this context, a methodology automating the Design Space Exploration (DSE) of energy and performance tradeoffs in dynamically reconfigurable systems has been proposed, allowing the evaluation of multiple application implementations on a heterogeneous System-on-Chip composed of processor(s) and dynamically reconfigurable unit(s). It starts from the description of execution resources (CPU cores, FPGA) and application tasks with their possible hardware and software instantiations. Then a greedy algorithm searches for all mapping solutions and computes the associated costs in terms of energy and performance. Reliable analysis of dynamic hardware implementations in particular requires relevant energy models of the dynamic and partial reconfiguration process. In the following, we illustrate the use of previous DPR models in this DSE and modeling framework for the realistic application example described in the following.

6.1.2. H264/AVC decoder

The application which is considered for this validation study is a H.264/AVC profile video decoder. High Level Synthesis (HLS) is used to provide values of cost performance tradeoffs for possible hardware functions, which serve as an entry point to the exploration methodology. The H.264 decoder used corresponds to the block diagram of Fig. 13 which is a version derived from the ITU-T reference code [20] to comply with hardware design constraints and HLS. From the original C++ code, a profiling step identifies four main functionalities for acceleration that are, in order of importance, the deblocking filter (24%), the inverse context-adaptive variable-length coding (Inv. CAVLC 21%), the inverse quantization (Inv. Quant. 19%) and the inverse integer transform (Inv. Transf. 12%). To achieve better results, we have merged the inverse quantization and integer transform into a single block (Inv. QTr.). It might be noted here that CAVLC was not added to this block because the HLS tool (Catapult C Synthesis 2009a Release) could not handle the complexity of the resulting C code. Therefore, this results in three potential hardware functions representing 76% of the total processing time.

The deblocking filter, inverse CAVLC, and inverse quantization and transform block are the three functionalities of the decoder that can be either implemented in software or in dedicated hardware. For hardware implementations, an exploration of loop level parallelism was carried out with the HLS tool in order to target varying performance and resource requirements. We have selected from the results the fastest and the slowest solutions for each accelerator, except for CAVLC where too little parallelism could be exploited by the HLS tool. Table 4 shows the corresponding hardware and software task parameters that will be the inputs for the exploration example. In this table, hardware execution parameters (section HW_{ex}) are measured on a Virtex-6 LX240T FPGA (Xilinx ML605 development board). Software tasks (section SW_{ex}) are described over a 600 MHz ARM CortexA8 processor (Texas Instruments BeagleBoard) rather than MicroBlaze in order to provide results and discussions that are more relevant of video processing constraints (25 fps).

6.2. Exploration results

With the previously described exploration framework, we analyze different execution possibilities of the H264 decoder envisaging a heterogeneous computing platform possibly composed of ARM CortexA8 cores and a Virtex-6 FPGA (supporting DPR). The exploration algorithm analyzes the power and time required for running each task using a CPU or a hardware resource (PRR). Reconfiguration costs are considered based on the estimation models presented in Section 4. Exploration results highlights one solution corresponding to the lowest energy consumption and execution time for the entire application. We chose to setup the reconfigurable resource in two PRRs of different sizes derived from the area of available hardware tasks described in Table 4. The size of *PRR*1 and *PRR*2 are respectively 1200 and 3200 slices in such a way that all hardware tasks can fit in the largest PRR (*PRR*2). The smallest PRR (*PRR*1) is suitable only for hardware sequential implementations (HW_{seq}) of Inv. QTr. and DB Filter. As the reconfiguration speed of the `xps_hw_icap` cannot exceed 15 MB/s, which is too slow to meet video processing constraints, we set a faster reconfiguration throughput of 40 MB/s. In these conditions, the corresponding exploration results are reported considering the Coarse grained and fine grained DPR models.

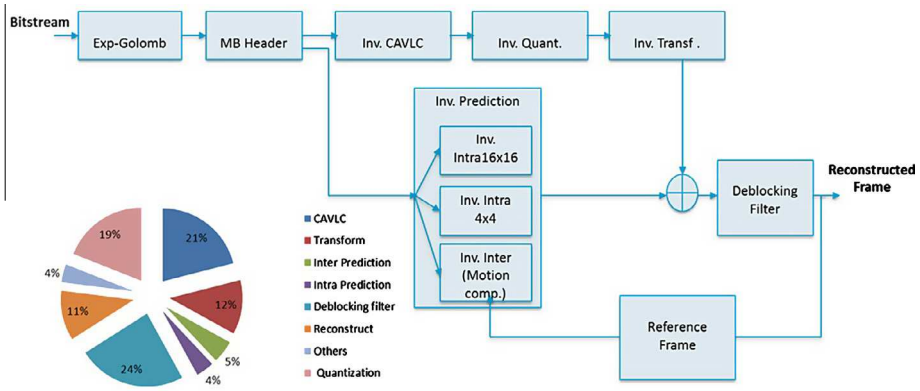


Fig. 13. H.264/AVC decoder block diagram and profiling.

Table 4
H264 task parameters.

Task	SW_{ex}	HW_{ex}			
		HW_{seq}		HW_{par}	
	T_{ex} (ms)	T_{ex} (ms)	Slices	T_{ex} (ms)	Slices
Exp-Golomb	5	–	–	–	–
MB_Header	4.92	–	–	–	–
Inv_CAVLC	22.06	14.90	3118	–	–
Inv_QTr	10.19	4.92	1056	3.93	1385
Inv_Pred	10.77	–	–	–	–
DB_Filter	34.98	3.14	686	3.11	1869

6.2.1. Coarse grained DPR model

Exploration results using the coarse grained model for the first execution of a hyper-period of the H264 decoder highlighted a best energy and performance DPR solution whose details are represented in Fig. 14. Top of this figure is the mapping of hardware and software tasks respectively on PRRs and CPUs against time. For this solution, three tasks are executed on a unique CPU (in blue) and three tasks are executed using the two PRRs (in green). We can notice the presence of dynamic and partial reconfiguration phases (in red) prior to each hardware task execution. As expected, reconfiguration time overheads are important (13 ms for *PRR*2, 5 ms for *PRR*1) but hardware DPR still outperforms software execution by 37%. On the corresponding estimated power profile (bottom of Fig. 14), we can see the processor power consumption when it goes from running to idle state when Inv. Pred. task is finished (21 ms). Power consumption of the reconfigurations, based on the coarse grained model (constant power), are represented before hardware execution (10–23 ms, 37–42 ms and 47–52 ms).

6.2.2. Fine grained DPR model

The best energy and performance DPR solution provided by the exploration results, using now the fine grained model, is reported in Fig. 15. Changing the DPR model does not change the mapping of tasks which is exactly the same as described previously in Fig. 14. Both overall power profiles of Figs. 14 and 15 are very similar except in some details resulting from the sophistication of the finer grained model, putting into light additional variations impacting the power profiles. Power surges, estimated with the Hamming distance, are apparent in the global SoC power consumption. Power steps are also present e.g. at 12 ms and 19 ms when configuring CAVLC and at 38 ms and 40 ms for the configuration of Inv. QTr. These power steps are not visible for DB Filter since idle power of Inv. QTr. and DB Filter are quite similar (33.4 mW and 34.2 mW).

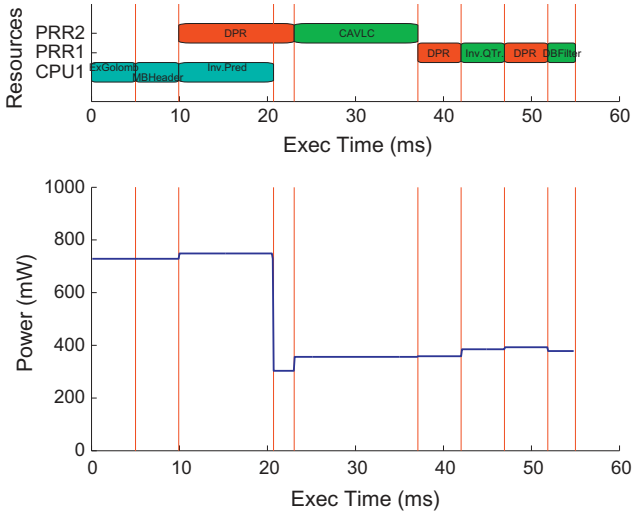


Fig. 14. Task mapping, scheduling and power profile estimation using the coarse grained model.

The overall energy, execution time and maximum power estimated with both models are reported in Table 5 for further discussion in the following.

6.2.3. Result analysis

Table 5 summarizes key features of the best hardware DPR solution highlighted by exploration, for both coarse grained and fine grained estimation results. It also compares these values against full software execution (one CPU, no accelerator) and static hardware implementation (no DPR, all IPs are statically instantiated on the FPGA). First, the difference in energy estimations is only 0.62 mJ (2%) between the fine grained and the coarse grained models. The reduced deviation comes from the efficient modeling of power steps and power surges in the reconfiguration process. Secondly, the estimated execution time is the same for both models, which shows that small energy differences did not affect mapping choices during exploration. Finally, the maximum peak power estimated by the fine grained model is 68 mW (9%) higher than the peak power estimated by the CG model. As mentioned in Section 5.4, the fine grained model illustrates its ability to consider more advanced level of characteristics, global peak power in this case. In this example, the maximum SoC power consumption is visibly impacted by DPR, this can have to be considered in the development of power sensitive applications. On the flip side, the coarse grained model provides simple but accurate enough model (2%

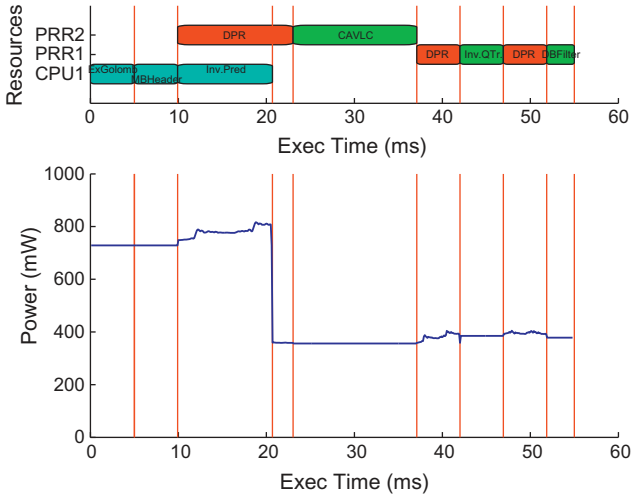


Fig. 15. Task mapping, scheduling and power profile estimation using the fine grained model.

Table 5
Characteristics of exploration results.

	Energy (mJ)	Time (ms)	Max power (mW)
Coarse Grain	31.91	54.99	748.48
Fine Grain	32.53 (1.9%)	54.99	816.5 (+9.1%)
Full software	39.12	87.92	445
HW (no DPR)	25.1	31.11	953

energy difference) to achieve early system level estimation and exploration.

In this case study, DPR hardware/software execution of the application provides noticeable acceleration and energy reduction compared to a full processor solution (based on single core Cortex-A8). Decoding rate is 18 frames per second (fps) while single processor execution reaches 11 fps, which is 64% faster. The energy gain is 17% for DPR against mono processor execution. However, compared to a static implementation of all accelerators (no DPR), the lowest energy DPR solution consumes more, 32.53 mJ vs. 25.10 mJ. The corresponding execution times are 31 ms for the static solution and 55 ms for the DPR solution. It should be noted here that 24 ms of this execution time (43%) is devoted to the reconfiguration overhead. Using a more efficient controller would greatly reduce this and the energy cost as well. For example, an optimized reconfiguration controller such as [11] has a reconfiguration speed of 400 MB/s for 384 nJ/KB, compared to 96 KB/s for 213 uJ/KB in a Xilinx ICAP based setup. Introducing these parameters in the coarse grained model for exploration reduces the total part of reconfiguration time from 43% to 9% of the total execution time. This also results in a lower power/energy overhead and reduces the overall energy of the best energy solution to 19.45 mJ, which is then 22.5% better than a static solution.

Table 7
Power profiles deviation for coarse grained (CG), medium grained (MG) and fine grained (FG) models, compared to measurements.

Config.	CG (*mW*)	MG (*mW*)	FG (*mW*)
Inv_QTr_{par} to T_{blank}	15.2 (142%)	5 (47%)	10.1 (94%)
T_{blank} to Inv_QTr_{par}	13.7 (59%)	6.3 (27%)	10.1(44%)
Inv_QTr_{par} to Inv_QTr_{seq}	17.7 (91%)	10.5 (54%)	12 (62%)
Inv_QTr_{seq} to *DB_Filter*	6.6 (24%)	6.5 (23%)	6.8 (23%)
DB_Filter to Inv_QTr_{par}	12.7 (30%)	16.5 (39%)	15 (35%)
Inv_QTr_{par} to T_1	16.9 (53%)	13.7 (43%)	13.5 (43%)
Average	10.8 (44%)	7.5 (31%)	8.3 (34%)

6.3. Further analysis of estimation accuracy

To complete the analysis of estimation accuracy (Section 5), six additional reconfiguration scenarios using H.264 hardware tasks with a PRR of 1440 slices have been considered in Tables 6 and 7: Inv_QTr_{par} to T_{blank}, T_{blank} to Inv_QTr_{par}, Inv_QTr_{par} to Inv_QTr_{seq}, Inv_QTr_{seq} to *DB_Filter*, *DB_Filter* to Inv_QTr_{par}, Inv_QTr_{par} to T_1. When compared to the measures, energy estimations show larger deviations: 27% for FG, 43% for MG and 89% for CG in average. Power profiles follow the same trend: 34% for FG, 31% for MG and 44% for CG (average). Closer analysis shows that the size of tasks increases disparities of the models (e.g. Inv_QTr_{par} with 1385 slices representing 96% of the PRR's resources). This means that the Hamming distance and power step modeling functions (Section 4) have to be refined for PRRs and tasks of very important complexities. Additional modeling and analysis would be required to further increase the precision of DPR estimates which depends on detailed low level DPR knowledge (bitstream, PRR structure) that are protected information not easy to investigate.

However, the estimation models are intended to be used at system level exploration where the current accuracy is sufficient. We chose not to develop finer DPR modeling as it is not essential for our goals and would additionally lead to technology dependent models. In previous exploration example, the improvement of FG over CG is due to a better matching with real power peaks (which grew by 9% using FG), and results in more accurate energy estimations. However, the benefits of DPR FG modeling regarding the full application level is relatively limited (1.9% better energy accuracy over CG for the H.264 decoder example). DPR overheads in this example represent up to 43% of the total execution time due to the use of an unoptimized DPR controller for measurement reasons. In a practical implementation, reconfiguration time would be reduced to a smaller fraction of the global application execution time, limiting even further the impact of model deviations to a level that can be considered negligible (e.g. significantly below 1.9% for the H.264 application example).

7. Conclusion and perspectives

Based on a reference procedure of dynamic and partial reconfiguration for Xilinx FPGAs, this paper has thoroughly investigated

Table 6
Average standard error and root mean square error of energy estimations for the coarse grained (CG), medium grained (MG) and fine grained (FG) models compared to real measures (M).

Config.	M (mJ)	CG error (mJ)	MG err. (mJ)	FG err. (mJ)
Inv_QTr_{par} to T_{blank}	7.4	24 (324%)	6.4 (86.8%)	6 (82%)
T_{blank} to Inv_QTr_{par}	16	−12.9 (−80.6%)	2.2 (−13.7%)	10 (67%)
Inv_QTr_{par} to Inv_QTr_{seq}	13.5	11.8 (87.5%)	0.4 (2.8%)	2.7 (19.6%)
Inv_QTr_{seq} to *DB_Filter*	19.2	−12.6 (−65.8%)	−5.4 (−28%)	2.9 (15%)
DB_Filter to Inv_QTr_{par}	29.1	−22.9 (−78.6%)	−15.2 (−52.4%)	−2.1 (−7.2%)
Inv_QTr_{par} to T_1	21.9	−1.4 (−6.6%)	−8 (−36.7%)	−1.7 (−7.6%)
RMSE		12.9 (89%)	6.1 (43%)	4 (27%)

the measurements and modeling of power and energy during run-time partial reconfiguration. Results have shown that power consumption was not as straightforward as expected. Components other than the reconfiguration controller have been identified, like the effect of previous configuration and the resources of reconfigurable regions. Three analytic models have been proposed in order to help analyzing the power contribution of dynamic reconfiguration at different levels of detail. The applicability and usefulness of these models have been illustrated on a representative example of design space exploration for a H264/AVC decoder. In this application study, the ability of models to determine if exploiting DPR in a SoC actually leads to an overall energy saving or to a loss has also been shown.

The perspectives from this work[1] are to continue developing the applicability, especially for more generic model calibration and interaction between models and tools. Another issue in this context is also to investigate online multiprocessor scheduling policies that would be able to support efficient execution of dynamic hardware and software tasks. The expected results from these efforts are a complete system level approach for the design space exploration of reconfigurable heterogeneous multicore System-on-Chips with power issues.

References

[1] E. Lemoine, D. Merceron, Run time reconfiguration of fpga for scanning genomic databases, in: Proceedings, IEEE Symposium on FPGAs for Custom Computing Machines, 1995.

[2] V. Tadigotla, L. Sliger, S. Commuri, FPGA implementation of dynamic run-time behavior reconfiguration in robots, in: Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE.

[3] J.G. Eldredge, B.L. Hutchings, Run-time reconfiguration: a method for enhancing the functional density of SRAM-based FPGAs, J. VLSI Signal Process. 12 (1996) 67–86.

[4] X. Zhang, H. Rabah, S. Weber, Dynamic slowdown and partial reconfiguration to optimize energy in fpga based auto-adaptive sopc, in: DELTA 2008, 4th IEEE International Symposium on Electronic Design, Test and Applications, 2008.

[5] L. Sterpone, L. Carro, D. Matos, S. Wong, F. Fakhar, A new reconfigurable clock-gating technique for low power SRAM-based FPGAs, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011.

[6] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, J. Becker, A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput, in: FPL 2008, International Conference on Field Programmable Logic and Applications, 2008.

[7] M. Rullmann, R. Merker, A cost model for partial dynamic reconfiguration, in: SAMOS 2008, International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008.

[8] F. Duhem, F. Muller, P. Lorenzini, Reconfiguration time overhead on field programmable gate arrays: reduction and cost model, Comput. Digital Techn. IET 6 (2012) 105–113.

[9] J. Becker, M. Huebner, M. Ullmann, Power estimation and power measurement of xilinx virtex fpgas: trade-offs and limitations, in: SBCCI 2003, Proceedings, 16th Symposium on Integrated Circuits and Systems Design, 2003.

[10] M. Lorenz, L. Mengibar, M. Valderas, L. Entrena, Power consumption reduction through dynamic reconfiguration, in: Field Programmable Logic and Application, Springer, Berlin/Heidelberg, 2004.

[11] R. Bonamy, H.-M. Pham, S. Pillement, D. Chillet, UPaRC – ultra fast power aware reconfiguration controller, in: Design, Automation & Test in Europe Conference & Exhibition (DATE).

[12] Open-PEOPLE – Open-Power and Energy Optimization PLatform and Estimator, 2013. <http://www.open-people.fr/>.

[13] Virtex-5 LXT ML550 Networking Interface & Power Measurement Platform, 2013. <http://www.xilinx.com/products/boards-and-kits/HW-V5-ML550-UNI-G.htm>.

[14] Xilinx, UG744 – PlanAhead Software Tutorial: Partial Reconfiguration of a Processor Peripheral, 2011.

[15] R. Bonamy, D. Chillet, O. Sentieys, S. Bilavarn, Parallelism level impact on energy consumption in reconfigurable devices, ACM SIGARCH Comput. Architect. News 39 (2011) 104–105.

[16] S. Liu, R.N. Pittman, A. Forin, Energy reduction with run-time partial reconfiguration, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays.

[17] Xilinx, UG191 – Virtex-5 FPGA Configuration User Guide, 2010.

[18] F. Duhem, F. Muller, P. Lorenzini, FaRM: fast reconfiguration manager for reducing reconfiguration time overhead on FPGA, in: Reconfigurable Computing: Architectures, Tools and Applications, Springer, Berlin/Heidelberg, 2011.

[19] Architecture Analysis & Design Language (AADL), version 2, 2010. <http://standards.sae.org/as5506a/>.

[20] ISO/IEC 14496-10, Advanced Video Coding for Generic Audiovisual Services, ITU-T Recommendation H.264, Version 4, 2005.

[1] This work was carried out under the Open-PEOPLE project, a platform project funded within the framework of the Embedded Systems and Large Infrastructures program (ARPEGE) from ANR, the French National Agency for Research.

Robin Bonamy received his M.E. degree in Electronics and Embedded Systems from University of Rennes, France in 2009. He is currently pursuing his Ph.D. at IRISA, France. His current research interests are reconfigurable devices, power and energy models and design space exploration especially for energy consumption reduction.

Sebastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis – CNRS. His research interests are in design, exploration and optimization from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of French, european and international collaborative research projects.

Daniel Chillet is member of the Cairn Team, which is an Inria Team located between Lannion and Rennes in France. He received the Engineering degree and the M.S. degree in electronics and signal processing engineering from University of Rennes 1, respectively, in 1992 and in 1994, the Ph.D. degree in signal processing and telecommunications from the University of Rennes 1 in 1997, and the habilitation to supervise PhD in 2010. He is currently an Associate Professor of electrical engineering at the Enssat, engineering school of University of Rennes 1. Since 2010, he is the Head of the Electronics Engineering department of Enssat. His research interests include memory hierarchy, reconfigurable resources, real-time systems, and middleware. All these topics are studied in the context of MPSoC design for embedded systems. Low power design based on reconfigurable systems is one important topic and spatio-temporal scheduling, memory organization and operating system services have been previously addressed on several projects.

Olivier Sentieys joined University of Rennes (ENSSAT) and IRISA Laboratory, France, as a full Professor of Electronics Engineering, in 2002. He is leading the CAIRN Research Team common to INRIA Institute (national research institute in computer science) and IRISA Lab. (research institute in computer science and random systems). Since September 2012 he is on secondment at INRIA as a Senior Research Director. His research activities are in the two complementary fields of embedded systems and signal processing. Roughly, he works firstly on the definition of new System-on-Chip architectures, especially the paradigm of reconfigurable systems, and their associated CAD tools, and secondly on some aspects of signal processing like finite arithmetic effects and cooperation in mobile systems. He is the author or coauthor of more than 150 journal publications or peer-reviewed conference papers and holds 5 patents. He is the head of the "Architecture" department of IRISA.

Journal of
Low Power Electronics
Vol. 12, 1–14, 2016

Power Modeling and Exploration of Dynamic and Partially Reconfigurable Systems

Robin Bonamy[1], Sébastien Bilavarn[2,*], Daniel Chillet[1], and Olivier Sentieys[1]

[1] *CAIRN, University of Rennes 1, CNRS, IRISA*
[2] *LEAT, 930 Route des Colles, University of Nice Sophia Antipolis, CNRS, F-06903, Sophia Antipolis, France*

(Received: 14 March 2016; Accepted: 11 July 2016)

Although fairly known for a long time, the vast potential of Dynamic and Partial Reconfiguration (DPR) for high energy efficiency is still difficult to exploit, for reasons that are more methodological than purely technical. This work addresses this problem and provides a contribution by seeking to improve energy efficient deployment and analysis for embedded heterogeneous multiprocessor platforms representative of current and upcoming systems. This paper explores the potential energy efficiency improvements of DPR on the concrete implementation of a H.264/AVC video decoder. The methodology used to explore the different implementations is presented and formalized. This formalization is based on pragmatic power consumption models of all the tasks of the application that are derived from real measurements. Results allow to identify low energy/high performance mappings, and by extension, conditions at which partial reconfiguration can achieve energy efficient application processing. The improvements are expected to be of 57% in energy and 37% in performance over pure software execution, corresponding also to 16% energy savings over static implementation of the same accelerators for 10% less performance.

Keywords: Dynamic and Partial Reconfiguration, Power Modeling, Energy Efficiency, SoC, eFPGA, Design Space Exploration, Video Processing.

1. INTRODUCTION

Minimizing power consumption and extending battery life are major concerns in popular consumer electronics like mobile handsets and wireless handheld devices. Silicon chips embedded in these products face challenging perspectives with the rise of processing heterogeneity introduced to address heat and power density problems. Efficient methodologies and tools are thus necessary to help the mapping and execution of applications on such complex platforms considering high demands for performance at less energy costs.

In parallel, in the context of mobile devices, video decoding is known as part of intense computation, which leads to an important impact on energy consumption. Considering this context and the need of methodology to help the designers of such systems, it is essential that a subset of relevant solutions can be quickly identified and evaluated from early steps of development. This process usually relies on high level modeling and estimations to help analyzing many complex interactions between a variety of implementation choices. To address this topic, performance models of FPGAs have been widely defined and proposed but power consumption has been less investigated in comparison, especially regarding Dynamic and Partial Reconfiguration (DPR).

DPR is a feature introduced to further improve the flexibility of reconfigurable based hardware accelerators. It exploits the ability to change the configuration of a portion of a FPGA while other parts are still running. By sharing the same area for the execution of different sequential tasks, DPR allows reducing the size of active programmable logic required for a given application.[1,2] Therefore, improving the area efficiency also results in decreasing static power consumption which is directly linked to the number of transistors in the chip. The counterpart is that an additional part of power is needed to switch from a current configuration to the next one. Thus, this DPR cost needs to be properly assessed and analyzed in order to check if there is an actual power consumption benefit at the complete system level, for a given application. This paper investigates the exploitation of this on a relevant case study addressing the implementation of a H.264/AVC video decoder in search of the best global energy efficiency at the complete system level. From this

*Author to whom correspondence should be addressed.
Email: bilavarn@unice.fr

concrete example, we define a pragmatic formalization of the problem that can be used to analyze mapping combination of tasks on hardware and software units with DPR, and to provide reliable evaluations of execution time, area, power and energy. In particular, it is shown how the different model parameters are defined, applied and exploited based on a real-life deployment study and from the deriving possible power measurement and breakdown analysis opportunities. The defined power characterization strongly grounded in experimentation therefore is shown to ensure applicability and relevance of the proposed software/DPR hardware energy modeling scheme and to afford the ability to produce pertinent estimation numbers of DPR benefits from large system level exploration based on its use.

The paper firstly presents state of the art techniques in the general field of dynamic reconfiguration in Section 2, with a special emphasis on energy analysis and optimization. Then Section 3 details formal models for application, platform and dynamic reconfiguration underlying the mapping analysis of the H.264/AVC decoder. The automatic analysis of different mappings and scheduling of hardware and software application tasks is described in Section 4. Then, a result analysis of the full video application deployment is presented along with estimation values that are discussed in Section 5. Finally, we conclude the paper and suggest future directions for research.

2. DYNAMIC PARTIAL RECONFIGURATION AND ENERGY EFFICIENCY

2.1. Energy Efficient DR Systems

Embedded systems have to cope with numerous challenges such as limited power supply, space and heat dissipation. Extensive research efforts are currently carried out to address these problems. For low production volumes, reconfigurable architectures and FPGAs have been attractive in embedded systems due to their flexibility, allowing faster development at lower costs than Application Specific Integrated Circuits (ASICs). However, the energy efficiency and the maximum frequency of reconfigurable hardware are also impaired by their flexible interconnect which does not allow to reach the power performance level of custom ASICs.[3,4]

As the focus of this work is on energy aware analysis of dynamic and partially reconfigurable systems, power modeling is central to the problem. One can find a variety of works spanning the general field of Dynamic Reconfiguration, among which a few primarily address power and energy efficiency concerns. Among relevant works,[5] presents an approach that investigated power management for Reconfigurable Video Coding (RVC) involving coarse grained reconfigurable systems. As far as DPR is not concerned, power estimations are based on SoC libraries and power issues reported here are quite specific to the RVC methodology and cannot be directly extend to FPGA and DPR implementations. In Ref. [6] an evolvable hardware system is used to optimize the power consumption of a DPR platform at run time. A singularity of this method is that the search space is explored at run time from measured power consumption profiles to adapt the genes of an evolutionary algorithm. However results reported are 5% to 10% battery lifetime benefits and no further energy analysis is provided. The approach in Ref. [7] describes an architecture exploration of hardware-based processing units with DPR support integrated in an OpenCL framework. Despite the undeniable interest of addressing an important weak point of reconfigurable hardware (programming model), support for the OpenCL programming mode in the study overshadow energy analysis and results and do not provide yet relevant DPR power and energy efficiency concerns.

From these relatively few number of works, it can be observed that all fall into delineated perimeter of applications. A large majority focuses on hardware accelerators (leaving aside the questions of software units) and none deal with both FPGA reconfiguration (DPR) and software execution. Many works address relatively simple applications and face the difficult problem of power analysis. Therefore, it can be said that the exploitation of energy efficiency for DPR systems still lacks of a solid grasp of the complex mapping opportunities for representative computing (multiprocessor) platforms, and this is mainly due to a global power modeling problem. As there is up to our knowledge, not much study addressing explicitly the definition of abstract power models, we describe in the following a set of studies that nonetheless provide valuable insights on the exploration of DPR based solutions in terms of energy efficiency.

2.2. DPR Related Techniques for Energy Efficiency

DPR allows a better use of hardware resources by sharing and reusing reconfigurable regions (PRRs) during execution, thus less area and energy consumption are expected. A variety of other techniques can be associated to this inherent DPR capability. For instance, it is possible to clear the configuration data of a PRR (referred to as *blank configuration* in the following) when it is unused which leads to decrease the share of static power associated with PRRs.[8] As an important part of power consumption also comes from clock signals, some techniques investigated the use of dynamic reconfiguration to reduce clock related impacts. A low overhead clock gating implementation based on dynamic reconfiguration has been proposed in Ref. [9], achieving 30% power reductions compared to standard FPGA clock-gating techniques based on LUTs. Another approach has been developed to modify the parameters of clock tree routing at run time reconfiguration to moderate clock propagation in the whole FPGA and to decrease dynamic power.[10] Finally, self-reconfiguration also permits online modification of clock frequency with low resource overhead by directly acting on clock management units from the reconfiguration controller.[11]

Dynamic and partial reconfiguration faces the difficult problem of task placement, both spatially on reconfigurable regions and temporally in terms of scheduling. Therefore DPR demands specific requirements to support this type of execution. It is generally the responsibility of a task scheduler, like,[12] to decide online which resource will support the execution of a task. The question becomes even more critical when addressing context saving issues related to the preemption and relocation of hardware tasks as discussed for example in Ref. [13]. In terms of power, scheduling must state when task reconfiguration occurs such as to avoid unnecessary idle consumption prior to execution.[14] It is also responsible for choosing to use *blank* configurations or not while ensuring this decision actually leads to an overall energy gain.[15]

Another important element in DPR optimization is related to hardware implementation and parallelism. Parallelism has the potential to drastically decrease the execution time of a hardware implementation. A technique to exploit this potential is to apply code transformations such as those available in High Level Synthesis (HLS) tools. Previous works reported two times energy reductions between sequential and unrolled loops of a hardware matrix multiplication implementation.[16] This parallelism exploitation is especially relevant for DPR since it can help the adaptation of a better area power performance tradeoff at run time.

However, all these opportunities add many dimensions to the DPR implementation problem for which there are currently few design analysis support, especially concerning energy and power consumption. It is thus extremely difficult to

(i) identify the most influential parameters in the design and

(ii) understand the impact of their variations in search of energy efficiency.

In the following, we detail the deployment analysis of a H.264/AVC decoder on a representative performance execution platform (multicore with DPR), and in the most energy efficient way. To support this, we present first formal models of application, platform and mapping to allow a more systematic design space exploration and to help the evaluation of their associated impacts. Relevant power and energy models of DPR represent another essential condition to provide early reliable evaluations. The power and energy models that are used in the proposed exploration are based on actual measurements of the DPR process which are further described in Ref. [17]. Finally, a greedy exploration heuristic is made out of this base and described in details. It is shown in the result analysis how a set of relevant energy and performance tradeoffs can be identified and compared against characteristic solutions (best performance, static hardware, full software execution, etc.).

3. PROBLEM MODELING

Previous work addressed a detailed study of DPR energy modeling which led to identify significant parameters of dynamic reconfiguration (FPGA and PRRs idle power, DPR control).[17] This section fully extends the model to support a full and realistic platform (hardware, software execution units), application (hardware, software tasks) and mapping characterization which is defined from a set of actual power values measured on real platforms reported in Sections 5.2 and 5.3. Therefore the assessment (i.e., practical applicability) of the proposed global modeling scheme is shown on a H.264 video decoder for an assumed dual core reconfigurable platform (CortexA8/Virtex-6 LX240T).

3.1. Target Platform Model

As previously stated, the target platform is a heterogeneous architecture composed of processors and dynamically reconfigurable accelerators. Each type of execution unit is formalized by a set of specific parameters that captures all the information needed for deployment exploration. These parameters can be broadly classified in three categories that are listed in Table I: platform topology, execution units and dynamic and partial reconfiguration characterization.

3.1.1. Platform Topology

Execution Units (EU) are divided in two categories: software and hardware execution units. The system is supposed composed of a number N^{core} of software execution units (processor cores) and N^{PRR} hardware execution units. The hardware execution units are defined as the partially reconfigurable regions of the FPGA. Therefore, the total number of hardware and software execution units N^{EU} in the architecture is $N^{\text{core}} + N^{\text{PRR}}$, and the jth execution unit of the architecture is tagged by EU_j with $j \in [1, N^{\text{EU}}]$. In this abstract representation, a heterogeneous *SoC* platform is represented by the set of all its execution units EU_j and this composition is considered fixed at run-time.

3.1.2. Execution Units

The size of a hardware unit EU_j is characterized in terms of logic resources with parameters N_j^{cell}, N_j^{bram} and N_j^{dsp} to ensure realistic resource representation. The *cell* terminology refers to the main configurable resource of the programmable logic (e.g., Xilinx *Slices*, Altera *Logic Elements)*.

Concerning the power model, a distinction is made between different components of each EU_j. The empty power consumption, P_j^{empty} or $P_{j,k}^{\text{empty}}$, reflects the power consumed when no application task is loaded, respectively

Table I. Parameters used for architecture formalization.

for PRRs and cores (at frequency $F_{j,k)}$. It is worth noting concerning PRRs that a task configured on it accumulates two contributions: $P_{i,k}^{\text{idle}}$ when the task is idle, and $P_{i,k}^{\text{run}}$ when it is running, which are both implementation dependent and therefore described in Section 3.2.3. The characterization of P_j^{empty} is additionally useful to consider PRR blanking opportunities in the analysis of task deployments. For software units, $P_{j,k}^{\text{run}}$ is the power of a core EU_j running at $F_{j,k}$ (full load). The energy of a software task can then be computed from $P_{j,k}^{\text{run}}$ and the corresponding execution time.

3.1.3. Dynamic and Partial Reconfiguration

Taking into account the cost of dynamic and partial reconfiguration involves two types of overhead: delay and energy. As the reconfiguration delay is mainly dependent on the speed of the reconfiguration controller and size of the configuration bitstream, it can be efficiently described by parameter T^{1cell} representing the time needed to configure one logic cell and reflects the performance of the reconfiguration controller. Power is addressed in a similar way by parameter E^{1cell} reflecting the energy needed to configure one logic cell. As configuration depends mostly on the number of logic cells composing a PRR in practice, delay and energy overheads are fairly easy to compute.

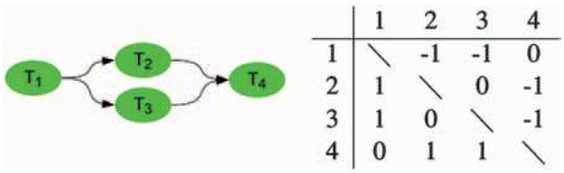
3.2. Application Power and Mapping Model

Different features of the application tasks need to be known for exploration and estimation. These characteristics are formalized by a set of parameters exposed in Table II. These parameters can be classified in three categories: task graph, task implementations and task execution characterization.

3.2.1. Task Graph

A task-dependency graph ς is used to reflect execution concurrency in the mapping problem. The dependencies between tasks are enumerated by an adjacency matrix representation which is convenient to process by an analysis algorithm. The adjacency matrix is a $N^T * N^T$ matrix used to represent dependencies between tasks. Considering a row i, the value in each column is 1 if T_i is dependent on the task represented by the column index, otherwise this value is 0. This adjacency matrix is asymmetric and -1 values are used to represent an inverted edge direction of the graph (Fig. 1). In addition, the adjacency matrix is completed with another information called the task equivalence matrix $T_{i1,i2}^{\text{eq}}$, which is used to indicate the identicalness of two or more tasks, meaning that they have the same execution code or bitstream. This equivalence matrix is useful to minimize execution units and improve their utilization rate, which is an important condition for DPR efficiency as it will be pointed out in the results.

Table II. Parameters used for application formalization.



	1	2	3	4
1	\	-1	-1	0
2	1	\	0	-1
3	1	0	\	-1
4	0	1	1	\

Fig. 1. Task graph example and associated adjacency matrix.

3.2.2. Task Implementations

From previous representation, it is also required to describe the possible implementations for each task. As we aim to explore combinations of mappings to the execution units, various implementations of the same task can be described. Different software (CPU core, frequency) and hardware (PRR) implementations can be specified to reflect several power performance tradeoffs in the exploration. The total number of possible hardware and software implementations for task T_i is N_i^{imp}, and $T_{i,k}$ is used to represent the kth implementation of task T_i with $k \in [1, N_i^{\text{imp}}]$.

3.2.3. Task Execution

First, a variable $I_{i,j,k}$ is used to express if $T_{i,k}$ can be executed on EU_j (0 false, 1 true). When task T_i is running on a software execution unit (EU_j at frequency $F_{j,k)}$, the corresponding execution time and energy consumption are defined by $C_{i,j,k}$ and $E_{i,j,k}$. Energy can be derived from the power characterization of a CPU core: $E_{i,j,k} = P_{j,k}^{\text{run}} * C_{i,j,k}$.

A slightly different model applies for hardware tasks (Fig. 2). When the kth implementation of a hardware task is mapped on a PRR, it comes with a part of idle power. This contribution is referenced by $P_{i,k}^{\text{idle}}$ for the kth implementation of task T_i. The remaining power contribution $P_{i,k}^{\text{run}}$ is added when the task is running. Therefore, the total power of PRR EU_j when $T_{i,k}$ is configured and running is $P_{i,k}^{\text{idle}} + P_j^{\text{empty}} + P_{i,k}^{\text{run}}$.

3.3. Model Assessment

We apply previous modeling on the H.264 decoder in a way to evaluate the extent of the formalization defined and show the actual setting of model parameters from a clear measurement process. In the absence of an available platform supporting all features necessary for these

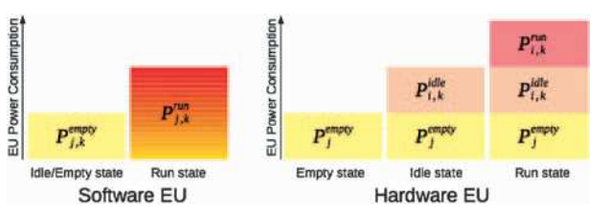


Fig. 2. Power contributions of an execution unit.

investigations (multiprocessor, power measurement, DPR, Linux, DVFS, etc.) at the beginning of this work, this characterization example is based on a dual CortexA8/Virtex-6 LX240T assumed platform. Power model parameters were set upon physical measurements on a Mistral Texas Instruments TI OMAP 3530 EVM (65 nm) board for software cores and a Xilinx EK-V6-ML605-G (40 nm) board for DPR acceleration. Further details of the specification graph and functions of the video decoder are given in Section 5.

3.3.1. Power Measurement Procedure

The FPGA device which is addressed in the following of this study is a Xilinx Virtex-6 LX240T. All measurements to set up the different parameters of the models are thus made on a Xilinx ML605 platform, including a built-in shunt resistor that can let us monitor the current through the FPGA core. Figure 3 shows the experimental setup for power measurements. We use the Virtex core shunt with a high-precision amplifier to handle current and power measurements that are logged with a digital oscilloscope. This setup allows measuring dynamic variations of current and power consumptions as low as milliamps and milliwatts during the execution of the device.

3.3.2. Platform

A first issue is to determine a set of relevant PRRs in terms of number (N^{PRR}) and size (N_j^{cell}, N_j^{bram}, N_j^{dsp}). We do not handle this partitioning in our approach and rely on the methodology of Ref. [15] which defines a systematic way to achieve this. Then, the empty power of PRRs (P_j^{empty}) can be derived from the empty power per logic cell, which is the power that can be measured when the full FPGA is powered but does not contain any configuration (at voltage and internal temperature constant), averaged by the number of cells. For instance, the Virtex-6 device used has a measured empty power of 1.57 W (at 1 V, 35 °C) for a capability of 37680 slices, which leads to a parameter of 41.7 μW/slice. It is then easy to derive the empty power of a PRR from its size. The configuration of a task on a PRR also adds contributions that are implementation dependent ($P_{j,k}^{\text{idle}}$, $P_{i,k}^{\text{run}}$), the determination of these parameters is thus described in the following section.

Empty and running power of CPU cores come from similar measurements. It is worth noting here that the model let the specification of different types of cores and frequencies ($F_{j,k}$). A software implementation can be associated to a core frequency k in this case. $P_{j,k}^{\text{empty}}$ and $P_{j,k}^{\text{run}}$ are the consumptions measured respectively when the core is idle (no application task) and running (assuming 100% CPU load). As an illustration, these values are 24 mW and 445 mW for an OMAP3530 based platform operating at 600 MHz.[19]

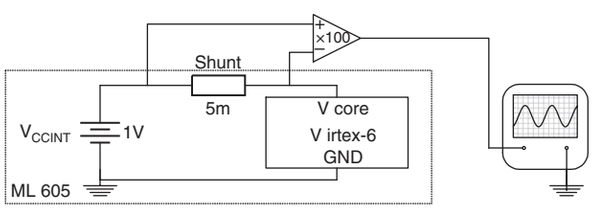


Fig. 3. Current measurement schematics on ML605 board using a high-precision amplifier.

The power model used for DPR reconfiguration is the *Coarse Grained DPR estimation model* detailed in Ref. [17]. In the example of Section 5.2, this model is calibrated for an optimized reconfiguration controller called UPaRC[20] supporting 400 MB/s at an average power of 150 mW. The minimum reconfiguration region in a Virtex-6 device (one *cell* = one *slice)* is one frame, where one frame is 324 bytes and contains two slices.[21] In these conditions, the corresponding T^{1cell} is (324B/400MB/s)/2 = 0.41 μs. In addition, the related E^{1cell} is $T^{\text{1cell}} * 150{\sim}\text{mW} = 61.5$ nJ. From these values, it is convenient to derive reconfiguration delay and energy for PRRs of different shapes and sizes.

3.3.3. Application

In the formalism of Table II, hardware and software task mapping parameters can be settled by defined implementations and measures. Classical profiling can be used to set out software execution time $C_{i,j,k}$ and derive the associated energy cost $E_{i,j,k}$ from the power $P_{j,k}^{\text{run}}$ of the executing core at frequency k (e.g., $E_{1,1,1} = 445$ mW $* 5$ ms $=$ 2.23 mJ for T_1 on core EU_1 in Table V).

As for hardware tasks, they are fully generated using an ESL (Electronic System Level) methodology described in Ref. [22]. Hardware mapping parameters are derived from measurements made possible by full accelerator implementation. $P_{i,k}^{\text{idle}}$ is the consumption measured when $T_{i,k}$ is configured but not running. This power is supposed to be independent from PRRs in our model. $P_{i,k}^{\text{run}}$ is the fraction of dynamic power added when $T_{i,k}$ is running (also supposed independent from PRRs), that can be determined in practice by subtracting the consumption of a configuration where $T_{i,k}$ is running from the consumption of a configuration where $T_{i,k}$ is idle. Therefore, the total power of a hardware task T_i is the sum of P_j^{empty} of a PRR and $P_{i,k}^{\text{idle}}$ when the task is idle, plus an additional contribution $P_{i,k}^{\text{run}}$ when the task is running. For example, the first hardware implementation $I_{5,4,2}$ of $T_{5,2}$ on EU_4 (PRR2) in Table V has a total energy cost $E_{5,4,2}$ computed from P_4^{empty} of PRR2, $P_{5,2}^{\text{idle}}/P_{5,2}^{\text{run}}$ of $T_{5,2}$ and the corresponding execution time $C_{5,4,2}$:

$$\begin{aligned} E_{5,4,2} &= (P_4^{\text{empty}} + P_{5,2}^{\text{idle}} + P_{5,2}^{\text{run}}) * C_{5,4,2} \\ &= (137\ \text{mW} + 34.2\ \text{mW} + 11.47\ \text{mW}) * 2.46\ \text{ms} \\ &= 0.45\ \text{mJ} \end{aligned}$$

This view is actually integrated in a more global framework for power modeling and analysis called OpenPEOPLE.[23] In particular, the open power platform supports

remote measurements and therefore let the definition of previous parameters reducing the need for equipment, devices and the usually complex monitoring procedures associated. The following section shows how using previous models developed from a set of accurate and concrete measurements can help defining relevant and reliable deployment exploration analysis.

4. DEPLOYMENT ANALYSIS

Based on previous modeling and formalization, more methodological approaches can be defined to explore the mapping space and provide relevant evaluations.

Execution time, area (in terms of programmable logic resources), energy and power profiles can be computed from the full characterization of the system available, including description and power models of execution units, SoC platform, Hw/Sw implementations of tasks, Dynamic Partial Reconfiguration and Partial Reconfigurable Regions. Figure 4 depicts the exploration methodology used for this energy efficiency study, and is further detailed in the following.

4.1. Exploration Inputs

4.1.1. System Description

The estimation flow starts with descriptions of application tasks and execution resources (Fig. 4-1). Tasks dependencies are specified using the aforementioned task-dependency graph (ς, N^T, T_i, $T^{\text{eq}}_{i1,i2}$), which is further processed using graph traversal techniques. Platform resource information like the size, number of PRRs and CPUs must be considered to determine different possible allocations. We assume here that the definitions of PRRs have been done so far (Section 3.3.2). Hw/Sw implementations of tasks and SoC platform characteristics required to compute power estimations come from specific libraries that are described in the following.

4.1.2. SoC Libraries

Power consumption and execution time of tasks for each possible execution unit are described in *Tasks Implementation libraries* (Fig. 4-2). For hardware tasks, these settings can be estimated by hardware dedicated power estimators, like the Xilinx Power Estimator, and execution time can be derived from timing reports produced by high level synthesis. Energy of software tasks can also be based on measurements or derived from data released by processor manufacturers. However for the sake of precision, power and execution times in the following come from real implementation and measurement for both hardware and software tasks (Section. 3.3.3).

Specific power and energy models are used to estimate the overheads resulting from PRR reconfigurations. These models have enough accuracy to estimate the latency, energy, and power profile of a reconfiguration from the characteristics of the reconfiguration controller, PRR and tasks involved.[17]

Another aspect in the estimation model is the power consumed by a hardware task present on a PRR, but not in use (idle power). An idle task power model is used to compute this contribution from sizes of tasks and PRRs. An improvement here is to configure a PRR with an empty task to reduce the associated idle power. This possibility is included in the mapping exploration process (Fig. 4-8).

Finally, device parameters such as the size, process technology and external adjustments like voltage and frequency are also present in *SoC Parameters*. This type of information is not currently used in the computation of estimations. Since previous libraries have been derived from measurements on a specific device (Virtex-6), we kept these characteristics in a way to derive implementations and models for different technologies.

4.2. Ordered Execution Lists

Task scheduling is a first requirement to compute application performance and energy estimations. We use information derived from the task-dependency graph to define a preliminary order for the execution of tasks. This is the role of the *Ordered Execution List* (OEL), which is a list of tasks similar to the task graph except that it sequentially determines which tasks must be launched for a time slot (τ_n). Each time slot corresponds to the end of a task and the beginning to, at least, one other task. The OEL is a representation of a static task schedule which is useful to derive at low complexity a number of feasible deployments. This ensures keeping enough workstation capacity to process the analysis of extensive task mappings along with fine complete power characterizations.

However, one OEL is not always enough to cover the best scheduling solution, especially if the application supports a lot of parallel tasks. An example is shown in Figure 4-3 where tasks *Inv_Pred* and *Inv_QTr* are executed in the same time slot, whereas *Inv_Pred* could also be run in parallel with *Inv_CAVLC*. In such a case, several OELs can be extracted from ς and explored one by one. The mapping definition process from an OEL depends on the formal parameters of Table III and is further described in the next section.

4.3. Mapping Exploration

4.3.1. Implementation Selection

Task mapping is the second requirement to compute performance and energy consumption. From the scheduling given by an OEL, for each task beginning at the current time slot τ_n, an implementation is selected from the list of possibilities (hardware or software) available in the task implementation library (Fig. 4-4).

$\text{Map}_{i,j,k,\tau n}$ is a variable which represents the implementation choice by its value: 1 at τ_n when $T_{i,k}$ is mapped

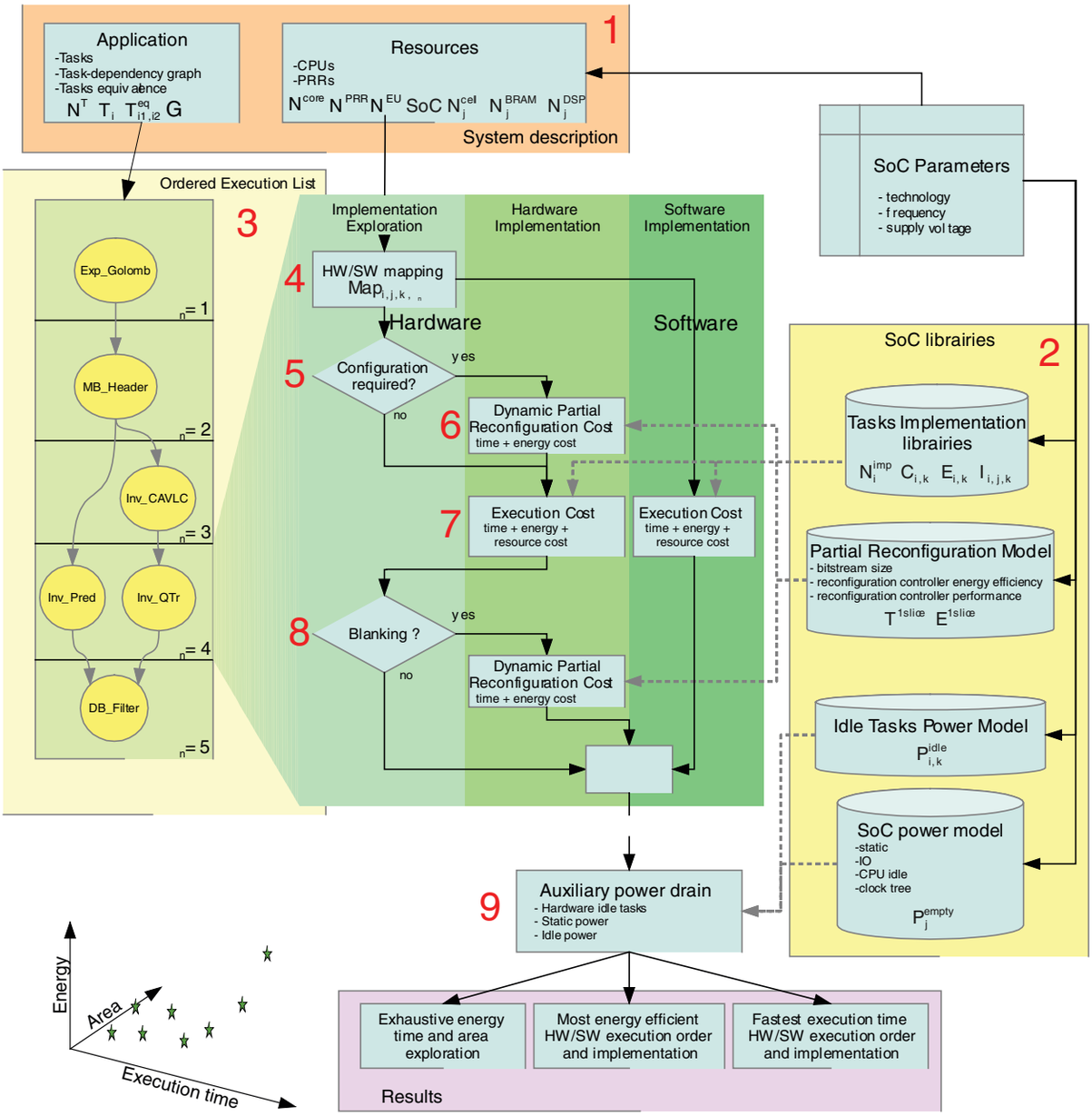


Fig. 4. Global exploration and power/performance estimation flow.

on EU_j. This variable is also set under the following constraints:

— one task is mapped only once

$$\sum_{j,k,\tau_n} \mathrm{Map}_{k,i,j,\tau_n} = 1 \quad \forall i \in 1,\ldots,N_T \tag{1}$$

Table III. OEL parameters used in exploration.

Variable	Range	Definition
N_{OEL}	$\in N^*$	Number of OLEs extracted from $\mathcal{G}$
N_n^{TS}	$\forall n = 1,\ldots,N^{\mathrm{OEL}}$	Number of time slots in the nth OLE
τ_n	$= 1,\ldots,N_n^{\mathrm{TS}}$	Current time slot in the nth OLE
$\mathrm{OLE}_{\tau_n,i}$	$\in \{0,1\}$	Presence of T_i for τ_n

— one EU can run only one task at the same time

$$\sum_{i,j} \mathrm{Map}_{k,i,j,\tau_n} \leq 1 \quad \forall i \in 1,\ldots,N^{\mathrm{EU}},\ \forall \tau_n \in 1,\ldots,N_n^{\mathrm{TS}} \tag{2}$$

For a mapping choice expressed by *Map*, estimations of power and execution time are then computed subsequently.

4.3.2. Partial Reconfiguration Cost

Reconfiguration is likely to occur when a task is mapped on a hardware execution unit (PRR), except if this task is already configured on this PRR (Fig. 4-5). In the situation where a reconfiguration is needed (Fig. 4-6), time and

energy overheads are computed respectively by:

$$T_j^{\text{conf}} = T^{1\text{cell}} \times N_j^{\text{cell}} \quad (3)$$

$$E_j^{\text{conf}} = E^{1\text{cell}} \times N_j^{\text{cell}} \quad (4)$$

It is worth noting here that, although BRAM and DSP blocks are involved in computing the use of FPGA resources (Section 3.1.2), they are not considered in the cost of reconfiguration control because it would incur low level layout and device dependent considerations for relatively few accuracy benefit (reconfiguration is a fraction of hardware task execution).

4.3.3. Task Execution Cost

Contributions of the actual execution of tasks (hardware and software) are then added to previous cost estimation (Fig. 4-7). The execution time for the current implementation of task T_i is given by $C_{i,j,k}$ while the energy consumption for this implementation is defined by $E_{i,j,k}$.

4.3.4. Blanking Analysis

When a hardware resource is not used for some time, blanking is an opportunity that can also be considered to save power (Fig. 4-8). However this technique comes with an added cost that has to be estimated.[12] This is determined by comparing the energy with and without blanking using the following expressions:

$$E_j^{\text{blanking}} = E_j^{\text{conf}} + E_j^{\text{empty}} = E_j^{\text{conf}} + P_j^{\text{empty}} * (T_j^{\text{idle}} - T_j^{\text{conf}}) \quad (5)$$

$$E_{i,j,k}^{\text{idle}} = P_{i,j}^{\text{idle}} * T_j^{\text{idle}} \quad (6)$$

where T_j^{idle} is the time during which EU_j is idle, waiting for a new task to begin. If $\text{E}_j^{\text{blanking}} < E_{i,j,k}^{\text{idle}}$ then blanking is an acceptable solution.

4.4. Auxiliary Power Drain

Auxiliary power contributions are also considered to fully characterize the energy consumption (Fig. 4-9). These contributions are from the static leakage power and from the idle power of the clock tree, both considered in P^{SoCidle}. The portion of power consumed by the execution units even when they are not in use ($P_{j,k}^{\text{empty}}$) is also added. We consider that power of a blank PRR is included in P_j^{empty}. However, hardware tasks already configured and idle also lead to power drains that are considered ($P_{i,k}^{\text{idle}}$).

4.5. Global Cost Characterization

At the end of an OEL analysis, energy contributions and execution times are added for each global mapping solution. An exhaustive search is currently used to enumerate the possible deployments of tasks on the execution units. This process consists in generating progressively at each time slot different branches for the OEL mapping. The end of a branch corresponds to a global deployment solution with its associated estimation of energy, area (resources) and performance. Notable solutions minimizing energy or performance are highlighted from a scatter plot representation of the results to help comparing the mappings explored (Fig. 6). Scheduling and the corresponding power profiles are computed as well to further analyze and implement a particular solution, as illustrated in Figure 8. Next section outlines these results that are derived from the application to a parallelized accelerated H.264/AVC decoder.

5. APPLICATION STUDY AND RESULTS

This section details a case study addressing a H.264/AVC decoder, exposing firstly how all application and platform model parameters are setup, here from concrete measurements (these could be also determined using estimators, at the expense of accuracy), and secondly the results of previously described exploration flow based on their use. This will additionally lead to a better valuation of DPR usefulness, potential energy efficiency benefits and conditions of effectiveness.

5.1. H.264/AVC Decoder

The application which is considered in this validation study is a H.264/AVC profile video decoder specification modified to comply with parallel software (multicore)/hardware (reconfigurable) execution. An ESL design methodology[22] is used to provide real implementations for the possible hardware functions, which serve as an entry point to the exploration flow of Section 4. The input specification code used is a version derived from the ITU-T reference code[24] to better cope with hardware design constraints.

The deblocking filter (*DB_Filter*), inverse CAVLC (*Inv_CAVLC*), and inverse quantization and transform block (*Inv_QTr*) contribute together to 76% of the global execution time on a single CPU core. They represent the three functionalities of the decoder that can be either software or hardware executed.

In addition to these acceleration opportunities, we aim at exploring solutions mapped onto parallel architectures including multicore CPUs. For this, we consider a multithreaded version of the decoder exploiting the possibility of slice decomposition of video frames supported in the H.264/AVC standard. Indeed a slice represents an independent zone of a frame, it can reference other slices of previous frames for decoding; therefore decoding one slice (of a frame) is independent from another (slice of the same frame). This way, the decoder can process different slices of a frame in parallel. We have thus considered a decomposition of the image where two streams process two halves of a same frame (Fig. 5). The corresponding task graph is defined by ς as:

$$\varsigma = \{Exp\_Golomb, MB\_Header, Inv\_CAVLC1, Inv\ CAV\_LC2, Inv\_QTr1, Inv\_QTr2, Inv\_Pred1, Inv\_Pred2, DB\_Filter1, DB\_Filter2\} \quad (7)$$

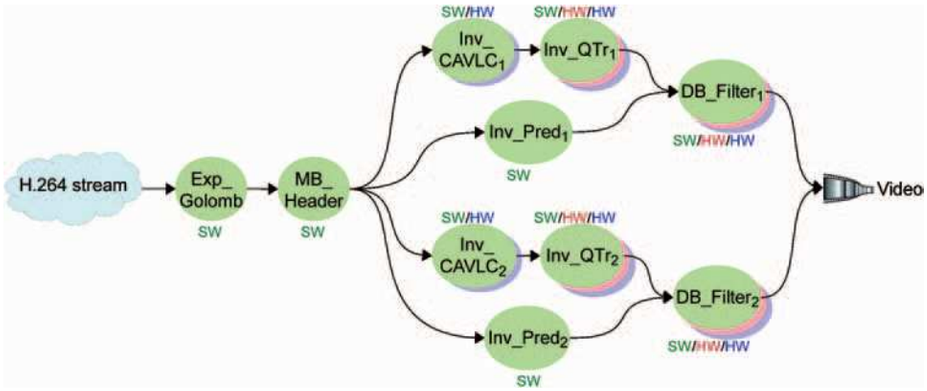


Fig. 5. H.264 decoder task flow graph.

This graph can be deployed using up to six accelerators and two processors on the underlying platform. Accelerators are fully generated from a reference C code at this level in a way to derive precise performance, resource, and power information, and to define relevant reconfigurable regions for DPR execution (Table IV). Some functions are reported with two implementations (sequential and parallel resulting from HLS loop unrolling) to account for the impacts of parallelism on energy efficiency. Software tasks are characterized in a similar way by running the code on CPU cores to derive execution times and energy, possibly at the different supported frequencies (Table V).

5.2. Target Platform

The execution platform is based on two ARM CortexA8 cores and an eFPGA, assuming a Virtex-6 device model supporting DPR. Table IV shows the corresponding platform parameters that have been set as exposed in Section 3.3.2. The method of Ref. [18] is used to identify an optimal set of PRRs under performance and FPGA layout constraints, considering all the hardware implementations of tasks previously considered. Three PRRs of 1200, 3280 and 2000 slices were found to reduce slice and BRAM count from respectively 49% and 33% over a purely static implementation of all hardware tasks. Therefore this configuration is used as a basic PRR setup in the following deployment analysis.

Table IV. Model parameters for a dual CortexA8/Virtex-6 LX240T assumed platform.

Platform	N^{core}	N^{PRR}	N^{EU}	Description
	2	3	5	CortexA8/V6LX240T
Cores	EU_j	$F_{j,k}$ (MHz)	$P^{empty}_{j,k}/P^{run}_{j,k}$ (mW)	Description
	EU_1	600	24/445	core #1
	EU_2	600	24/445	core #2
PRRs	EU_j	$N^{cell}_j; N^{bram}_j; N^{dsp}_j$	P^{empty}_j (mW)	Description
	EU_3	1200; 8; 0	50	PRR #1
	EU_4	3280; 8; 0	137	PRR #2
	EU_5	2000; 8; 0	83	PRR #3
DPR	Conf. Ctlr	T^{1cell} (μs)	E^{1cell} (nJ)	Description
		0.41	61.5	UPaRC

From $P^{\text{empty}}_{V6LX240T} = 41.7$ μW/slice measured previously on the device, it is possible to derive the empty power consumption for each defined PRR: $P^{\text{empty}}_3 = 50$ mW, $P^{\text{empty}}_4 = 137$ mW and $P^{\text{empty}}_5 = 83$ mW. The empty power of CPU cores come from similar measurements on an OMAP3530 based development board: $P^{\text{empty}}_{j,k} = 24$ mW and $P^{\text{run}}_{j,k} = 445$ mW for a CortexA8 core at $F_{j,k} = 600$ MHz.

Additionally, the reconfiguration controller used is an optimized IP called UPaRC supporting a reconfiguration speed of 400 MB/s for a power of $P_{\text{controller}} = 150$ mW.[20] This corresponds to $T^{1\text{cell}} = 0.41$ μs and $E^{1\text{cell}} = 61.5$ mJ (Section 3.3.2) from which reconfiguration time and energy of a PRR are easy to compute.

5.3. Application Model Parameters

Hardware power models are based on the power consumed by the whole task implemented on a PRR. In our validation and model characterization approach, this power

Table V. Task parameters for an H.264/AVC decoder application.

T_i	$T_{i,k}$	$I_{i,j,k} = 1$	$C_{i,j,k}$(ms)	$E_{i,j,k}$(mJ)	$P^{idle}_{i,k}/P^{run}_{i,k}$(mW)	$N^{cell}_{i,k}; N^{bram}_{i,k}; N^{dsp}_{i,k}$	Description
T_1	$T_{1,1}$	$I_{1,1,1}$ $I_{1,2,1}$	5.00	2.23	24/445	–	*Exp_Golomb*
T_2	$T_{2,1}$	$I_{2,1,1}$ $I_{2,2,1}$	4.92	2.19	24/445	–	*MB_Header*
T_3	$T_{3,1}$	$I_{3,1,1}$ $I_{3,2,1}$	11.03	4.91	24/445	–	*Inv_CAVLC*$_1$
	$T_{3,2}$	$I_{3,4,2}$	7.45	1.46	55.1/4.4	3118; 6; 0	
T_4	$T_{4,1}$	$I_{4,1,1}$ $I_{4,2,1}$	11.03	4.91	24/445	–	*Inv_CAVLC*$_2$
	$T_{4,2}$	$I_{4,2,4}$	7.45	1.46	55.1/4.4	3118; 6; 0	
T_5	$T_{5,1}$	$I_{5,1,1}$ $I_{5,2,1}$	5.10	2.27	24/445	–	*Inv_QTr*$_1$
	$T_{5,2}$	$I_{5,3,2}$	2.46	0.24	34.2/11.47	1056; 7; 0	
	$T_{5,2}$	$I_{5,4,2}$	2.46	0.45	34.2/11.47	1056; 7; 0	
	$T_{5,2}$	$I_{5,5,2}$	2.46	0.32	34.2/11.47	1056; 7; 0	
	$T_{5,3}$	$I_{5,3,3}$	1.97	0.21	42.2/12.87	1385; 7; 0	
	$T_{5,3}$	$I_{5,4,3}$	1.97	0.38	42.2/12.87	1385; 7; 0	
	$T_{5,3}$	$I_{5,5,3}$	1.97	0.27	42.2/12.87	1385; 7; 0	
T_6	$T_{6,1}$	$I_{6,1,1}$ $I_{6,2,1}$	5.10	2.27	24/445	–	*Inv_QTr*$_2$
	$T_{6,2}$	$I_{6,3,2}$	2.46	0.24	34.2/11.47	1056; 7; 0	
	$T_{6,2}$	$I_{6,4,2}$	2.46	0.45	34.2/11.47	1056; 7; 0	
	$T_{6,2}$	$I_{6,5,2}$	2.46	0.32	34.2/11.47	1056; 7; 0	
	$T_{6,3}$	$I_{6,3,3}$	1.97	0.21	42.2/12.87	1385; 7; 0	
	$T_{6,3}$	$I_{6,4,3}$	1.97	0.38	42.2/12.87	1385; 7; 0	
	$T_{6,3}$	$I_{6,5,3}$	1.97	0.27	42.2/12.87	1385; 7; 0	
T_7	$T_{7,1}$	$I_{7,1,1}$ $I_{7,2,1}$	5.39	2.40	24/445	–	*Inv_Pred*$_1$
T_8	$T_{8,1}$	$I_{8,1,1}$ $I_{8,2,1}$	5.39	2.40	24/445	–	*Inv_Pred*$_2$
T_9	$T_{9,1}$	$I_{9,1,1}$ $I_{9,2,1}$	17.49	7.78	24/445	–	*DB_Filter*$_1$
	$T_{9,2}$	$I_{9,3,2}$	1.57	0.14	33.4/6	686; 5; 0	
	$T_{9,2}$	$I_{9,4,2}$	1.57	0.28	33.4/6	686; 5; 0	
	$T_{9,2}$	$I_{9,5,2}$	1.57	0.19	33.4/6	686; 5; 0	
	$T_{9,3}$	$I_{9,4,3}$	1.55	0.29	40.3/7.4	1869; 5; 0	
	$T_{9,3}$	$I_{9,5,3}$	1.55	0.20	40.3/7.4	1869; 5; 0	
T_{10}	$T_{10,1}$	$I_{10,1,1}$ $I_{10,2,1}$	17.49	7.78	24/445	–	*DB_Filter*$_2$
	$T_{10,2}$	$I_{10,3,2}$	1.57	0.14	33.4/6	686; 5; 0	
	$T_{10,2}$	$I_{10,4,2}$	1.57	0.28	33.4/6	686; 5; 0	
	$T_{10,2}$	$I_{10,5,2}$	1.57	0.19	33.4/6	686; 5; 0	
	$T_{10,3}$	$I_{10,4,3}$	1.55	0.29	40.3/7.4	1869; 5; 0	
	$T_{10,3}$	$I_{10,5,3}$	1.55	0.20	40.3/7.4	1869; 5; 0	

is measured (it could also be estimated) and therefore includes logic cells, DSP and BRAM blocks as well, if the corresponding hardware task makes use of such dedicated resources.

Table V shows the application model parameters in details for the H.264 decoder example. The decoder is composed of ten tasks among which six can be run in hardware. All tasks are characterized in terms of software and hardware mapping following the generation and measurement procedure of Section 3.3.3.

For each hardware task (T_5, T_6, T_9, T_{10}), two versions of different cost and performance tradeoffs are produced using HLS loop level parallelism. Therefore, if we consider the example of Inv_QTr_1 (T_5), three implementations $T_{5,1}$, $T_{5,2}$, $T_{5,3}$ are possible with respectively 5.10 ms (CortexA8 600 MHz), 2.46 ms (hardware #1) and 1.97 ms (hardware #2 with loop unrolling).

For each of the two hardware implementations, three possible PRR mappings are described along with the associated energy cost (computed as shown in Section 3.3.3). This realistic characterization of different task implementations based on practical data improves the reliability of estimations and exploration results which are addressed in the following.

5.4. Exploration Results and Analysis

Under previous conditions of application, SoC architecture and dynamic reconfiguration, the primary output of the exploration flow is plotted in Figure 6. It is worth noting first the very important quantity of solutions analyzed, over 1 million possible mappings are evaluated for this design. This exploration example is processed in a matter of seconds with an Intel Core i5 based workstation.

Table VI. Highlights of exploration results.

Implementation results	Energy (mJ)	T_{EX} (ms)	CPUs;	Slices;	DSPs;	BRAMs
Software 1core	41.23	87.92	1;	0;	0;	0
SW 2cores (reference)	41.47	48.92	2;	0;	0;	0
Lowest energy -static	21.40 (−48%)	27.93 (−43%)	1;	6480;	0;	18
Best performance -static	25.30 (−39%)	**24.49** (−**50**%)	2;	6480;	0;	18
Lowest energy-DPR	**17.94** (−**57**%)	30.62 (−37%)	1;	4480;	0;	16
Best performance -DPR	23.84 (−43%)	25.13 (−49%)	2;	6480;	0;	24

Six solutions are highlighted from the results:

(i) full software implementation using one CPU core (*SW_1Core*)

(ii) full software execution using two cores (*SW_2Cores*),

(iii) the lowest energy solution using *static* accelerators (*LE_Static*),

(iv) the best performance solution using *static* accelerators (*BP_Static*),

(v) the lowest energy solution using dynamic reconfiguration (*LE_DPR*) and

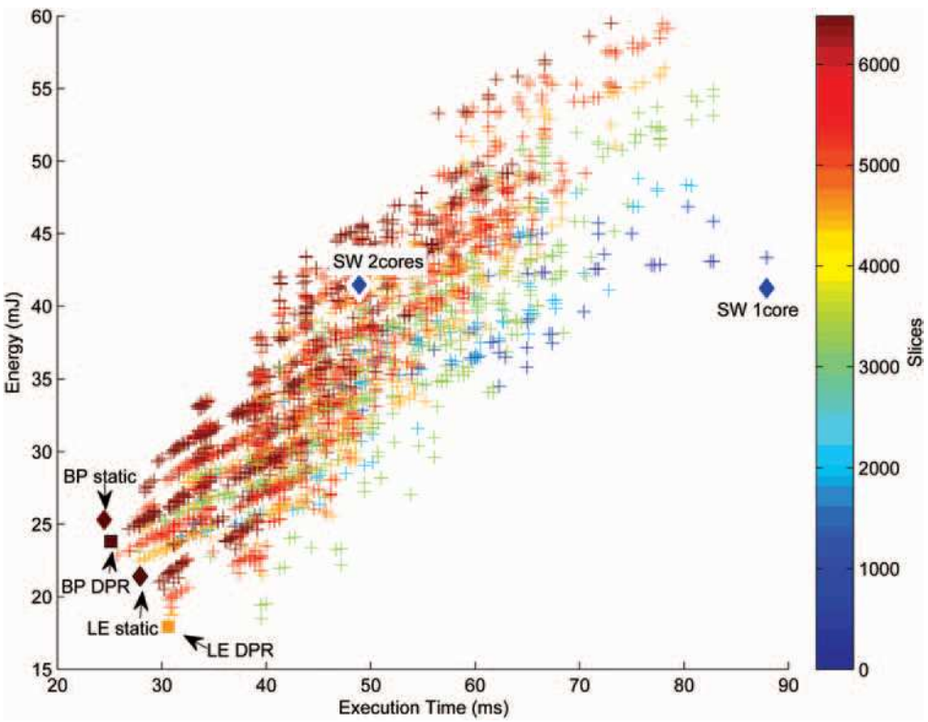


Fig. 6. Energy versus execution time exploration results. Colors represent the number of FPGA resources (slices) of a solution.

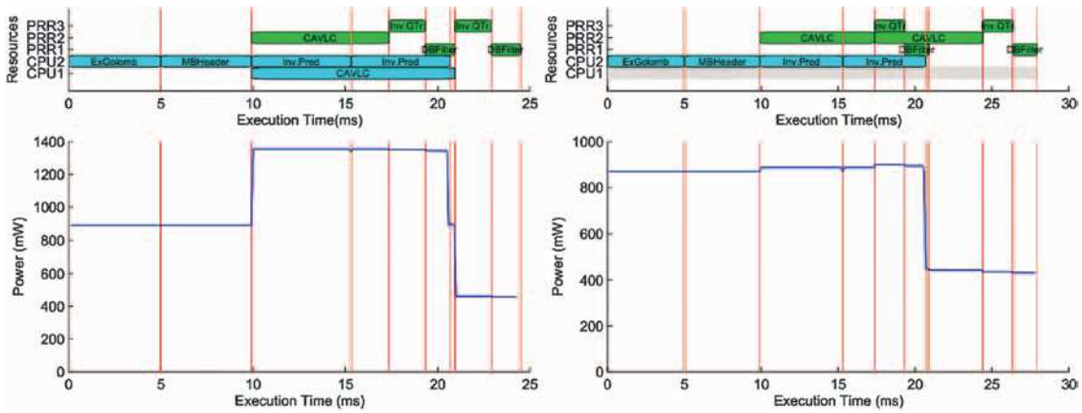


Fig. 7. Scheduling and power profile of *BP_Static* (left) and *LE_Static* (right) solutions.

(vi) the best performance solution using dynamic reconfiguration (*BP_DPR*).

Details of these characteristic solutions are summarized in Table VI. Since *SW_1Core* is almost twice slower for the same energy compared to *SW_2Cores*, *SW_2Cores* is considered as a reference result in the following to let the comparison of relative improvements from hardware accelerated solutions. To further help this analysis, exploration results also output scheduling, allocation and power profiles that are illustrated in Figures 7 and 8 for *BP_Static*, *LE_Static*, *BP_DPR* and *LE_DPR*.

We can firstly note that the four accelerated solutions perform better compared to the reference software execution, both in terms of performance and energy. Hardware significantly improves processing efficiency while offloading CPU cores which results in 50% faster execution and 39% energy savings at the global decoder application level.

Dynamic reconfiguration introduces a slight performance penalty due to reconfiguration delays, however the best performance solution based on DPR is only 2.6% slower than a static implementation. In terms of energy, the lowest energy solution using DPR is 57% more efficient than the reference *SW_2Cores* and 16% more energy efficient than a static implementation.

Inspecting the schedule and resources usage of Figures 7 and 8 emphasizes the fact that performance solutions make use of a maximum of resources, while low energy implementations tend to use less execution units and improve their utilization rate. For example, the energy of *BP_Static* is reduced by offloading the execution of function CAVLC from the first CPU core to PRR2 which results in an energy gain of 3.9 mJ (−15%) for a performance penalty of 3.4 ms (+14%). The same applies for DPR implementation *BP_DPR* in which Core1 and PRR3 can be removed to save 5.9 mJ (−25%) while increasing execution time by 5.5 ms (+22%).

Minimizing the number of reconfigurations (represented in red in the scheduling profiles) is also an important factor impacting execution time and energy consumption. In the DPR solutions of the decoder, the configuration of PRR2 is kept to execute two consecutive instances of *Inv_CAVLC*, and the same applies for the execution of *DB_Filter* on PRR1. However, *Inv_QTr* is mapped to PRR1 for the first instance and to PRR2 for the second. Execution dependencies between *DB_Filter* and *Inv_QTr* do not allow to

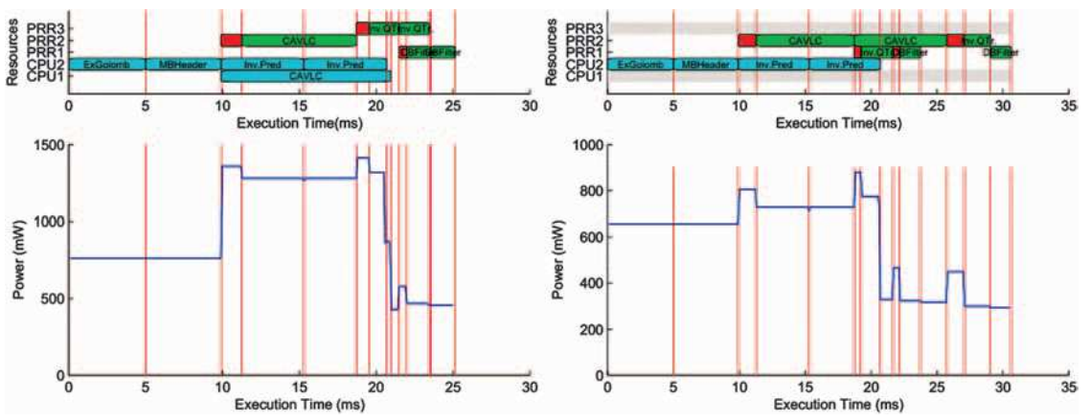


Fig. 8. Scheduling and power profile of *BP_DPR* (left) and *LE_DPR* (right) solutions.

save a reconfiguration of *Inv_QTr* without a penalty on global execution time, impacting also energy. The second instance of *Inv_QTr* is thus executed on PRR2.

In terms of DPR benefits over static implementation, the H.264 decoder example shows that DPR brings 16% energy improvement for 31% FPGA resource (slice) reduction and an execution time increase of 10%. Energy gains come from the reduction of the static area of the programmable logic and the associated idle power, decreasing from 318 mW to 210 mW (34%). These results help evaluating the practical benefits of dynamic reconfiguration, considering also that there is room for improvement on the H.264 decoder by moving more functions to hardware.

Finally, it is also interesting to note that for this application, none of the four hardware solutions highlighted exploits PRR blanking. In the *LE_DPR* solution, hardware execution units are not free for a sufficient period of time to compensate the energy overheads implied by PRR reconfigurations. Therefore these estimations provide a possible assessment to know whether or not to use blanking in a design.

The end result for a target platform possibly made of two CPUs and a FPGA fabric is a solution based on a single core execution with dynamic reconfiguration of six hardware accelerated functions on two PRRs. The corresponding implementation represents 57% and 37% performance and energy improvements over a dual core software execution which is also 16% more energy efficient over a static hardware implementation of the same accelerators with 10% less performance.

6. CONCLUSION AND PERSPECTIVES

This paper addressed the evaluation of the interest of dynamic and partial reconfiguration in the context of low power optimizations. Indeed, the possibility to reduce area (by reusing the same resources for several sequential tasks) is an interesting approach to save static power and/or to use a smaller FPGA. But this capability doesn't come for free, and this paper proposes a complete formalism of tasks execution on a platform based on processor cores and reconfigurable area. This formalism is then used as a support for design space exploration including tasks mapping in the different available execution resources. Previous detailed results report different potential energy efficiency improvements on a representative video processing application. DPR benefits are sensitive over pure software (dual core) execution with 57% energy gains for 37% better performance. There are comparatively less limited benefits against static (no DPR) hardware acceleration with 16% energy gains, but for 10% less performance (resulting from the overheads of reconfiguring partial regions). In addition to these numbers, we can derive a set of conditions that are essential for practical DPR effectiveness. First, the cost of reconfiguration is high both in terms of delay and energy. Thus all reconfiguration overheads have to be minimized as much as possible which means to support high speed reconfiguration control and to reach a schedule minimizing the number of reconfigurations. Second, hardware execution being to a very large extent significantly more energy efficient than software, accelerated functions are likely to be employed. On top of this, minimizing the number of regions will improve the results, both because it reduces the inherent power (especially the idle power), but also because it improves usage of the available regions. Therefore, there is still room for improving the H.264 decoder, in which only three functions are considered for acceleration, as quality of results will grow when increasing and sharing the number of hardware functions on a limited number of regions.

From these considerations, a first perspective is to address further energy gains with the definition of run-time scheduling policies supporting energy-aware execution of dynamic hardware and software tasks. Indeed exploration is likely to provide overestimated performances at design time since it has to be based on (static) worst case execution times. Therefore, there is room for complementary energy savings by exploiting dynamic slacks resulting from lesser execution times at run-time, and these scheduling decisions can benefit from the same models used for mapping exploration. Finally, another direction of research will be to build on this exploration and scheduling base to achieve efficient cooperation with existing processor level techniques (e.g., DVFS) and converge towards an advanced heterogeneous power management scheme.

Acknowledgments: This work was carried out under the Open-PEOPLE project, a platform project funded within the framework of the Embedded Systems and Large Infrastructures program (ARPEGE) from ANR, the french National Agency for Research. This work is also carried out under the BENEFIC project (CA505), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

References

1. J. G. Eldredge and B. L. Hutchings, Run-time reconfiguration: A method for enhancing the functional density of SRAM-based FPGAs. *Journal of VLSI Signal Processing* 12, 67 (**1996**).
2. K. Latif, A. Aziz, and A. Mahboob, Deciding equivalances among conjunctive aggregate queries. *Computers and Electrical Engineering* 37, 1043 (**2011**).
3. I. Kuon and J. Rose, Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 203 (**2007**).
4. A. Amara, F. Amiel, and T. Ea, FPGA versus ASIC for low power applications. *Microelectronics Journal* 37, 669 (**2006**).
5. F. Palumbo, C. Sau, and L. Raffo, Coarse-grained reconfiguration: Dataflow-based power management. *IET Computers and Digital Techniques* 9, 36 (**2015**).
6. B. López, J. Valverde, E. de la Torre, and T. Riesgo, Power-aware multi-objective evolvable hardware system on an FPGA,

NASA/ESA Conference on Adaptive Hardware and Systems (AHS) **(2014)**, pp. 61–68.
7. A. Rodriguez, J. Valverde, and E. de la Torre, Design of OpenCL-compatible multithreaded hardware accelerators with dynamic support for embedded FPGAs, *International Conference on ReConFigurable Computing and FPGAs (ReConFig)* **(2015)**, pp. 1–7.
8. T. Tuan and B. Lai, Leakage power analysis of a 90 nm FPGA. *IEEE Custom Integrated Circuits Conference* **(2003)**.
9. L. Sterpone, L. Carro, D. Matos, S. Wong, and T. Ea, FPGA versus ASIC for low power applications. *Microelectronics Journal* 669 **(2006)**.
10. Q. Wang, S. Gupta, and J. H. Anderson, Clock power reduction for virtex-5 FPGAs, *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* **(2009)**, pp. 13–22.
11. K. Paulsson, M. Hubner, and J. Becker, Dynamic power optimization by exploiting self-reconfiguration in Xilinx Spartan 3-based systems. *Microprocessors and Microsystems* 46 **(2009)**.
12. T. T. O. Kwok and Y. K. Kwok, Practical design of a computation and energy efficient hardware task scheduler in embedded reconfigurable computing systems, *Proceedings IEEE 20th International Symposium on Parallel and Distributed Processing* **(2006)**.
13. H. Kalte and M. Porrmann, Context saving and restoring for multitasking in reconfigurable systems, *Proceedings IEEE International Conference on Field Programmable Logic and Applications* **(2005)**, pp. 223–228.
14. P. H. Yuh, C. L. Yang, C. F. Li, and C. H. Lin, Leakage-aware task scheduling for partially dynamically reconfigurable FPGAs. *ACM Transactions on Design Automation of Electronic Systems* 14, 1 **(2009)**.
15. S. Liu, R. N. Pittman, and A. Forin, Energy reduction with runtime partial reconfiguration, *Proceedings ACM/SIGDA International Symposium on Field Programmable Gate Arrays* **(2010)**.
16. R. Bonamy, D. Chillet, O. Sentieys, and S. Bilavarn, Parallelism level impact on energy consumption in reconfigurable devices. *ACM SIGARCH Computer Architecture News* **(2011)**.
17. R. Bonamy, D. Chillet, O. Sentieys, and S. Bilavarn, Power consumption models for the use of dynamic and partial reconfiguration. *Microprocessors and Microsystems, Elsevier* **(2014)**.
18. F. Duhem, F. Muller, R. Bonamy, and S. Bilavarn, FoRTReSS: A flow for design space exploration of partially reconfigurable systems, Design Automation for Embedded Systems, Springer **(2015)**.
19. F. Kriegel, F. Broekaert, A. Pegatoquet, and M. Auguin, Power optimization technique applied to real-time video application, *Proceeding of 13th Sophia Antipolis Microelectronics Forum* **(2010)**.
20. R. Bonamy, H. M. Pham, S. Pillement, and D. Chillet, UPaRC-ultra fast power aware reconfiguration controller, *Design, Automation and Test in Europe Conference and Exhibition* **(2012)**.
21. Xilinx Inc, UG360–Virtex-6 FPGA Configuration User Guide (v3.1), Technical Report **(2010)**.
22. T. Damak, I. Werda, S. Bilavarn, and N. Masmoudi, Fast prototyping H.264 deblocking filter using ESL tools. *Transactions on Systems, Signals and Devices, Issues on Communications and Signal Processing* 8, 345 **(2013)**.
23. E. Senn, D. Chillet, O. Zendra, C. Belleudy, R. B. Atitallah, A. Fritsch, and C. Samoyeau, Open-people: An open platform for estimation and optimizations of energy consumption, *Design and Architectures for Signal and Image Processing Conference* **(2012)**.
24. ISO/IEC 14496-10, Advanced Video Coding for Generic Audiovisual Services, ITU-T Recommendation H.264, Version 4 **(2005)**.

Robin Bonamy

Robin Bonamy *received his M.E. degree in Electronics and Embedded Systems in 2009 and the Ph.D. degree from University of Rennes, France, in 2013. His research interests are reconfigurable devices, power and energy models and design space exploration especially for energy consumption reduction. He is currently focused on low power, wireless devices and he is consultant in innovation based on digital electronic technologies.*

Sébastien Bilavarn

Sébastien Bilavarn *received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis-CNRS. His research interests are in design, exploration and optimization from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of french, european and international collaborative research projects.*

Daniel Chillet

Daniel Chillet *is member of the Cairn team which is an Inria team located between Lannion and Rennes in France. He received the Engineering degree and the M.S. degree in electronics and signal processing engineering from University of Rennes 1, respectively, in 1992 and in 1994, the Ph.D. degree in signal processing and telecommunications from the University of Rennes 1 in 1997, and the habilitation to supervise Ph.D. in 2010. He is currently a Professor of electrical engineering at Enssat, engineering school of University of Rennes 1. Since september 2014, he is director of the master "Information and Communication Technology" of the University of Science and Technology of Hanoi. His research interests include memory hierarchy, reconfigurable resources, real-time systems, and middleware. All these topics are studied in the context of MPSoC design for embedded systems. Low power design based on reconfigurable systems is one important topic and spatio-temporal scheduling, memory organization and operating system services have been previously addressed on several projects.*

Olivier Sentieys

Olivier Sentieys *joined University of Rennes (ENSSAT) and IRISA Laboratory, France, as a full Professor of Electronics Engineering, in 2002. He is leading the CAIRN Research Team common to INRIA Institute (national research institute in computer science) and IRISA Lab. (research institute in computer science and random systems). Since September 2012 he is on secondment at INRIA as a Senior Research Director. His research activities are in the two complementary fields of embedded systems and signal processing. Roughly, he works firstly on the definition of new System-on-Chip architectures, especially the paradigm of reconfigurable systems, and their associated CAD tools, and secondly on some aspects of signal processing like finite arithmetic effects and cooperation in mobile systems. He is the author or coauthor of more than 150 journal publications or peer-reviewed conference papers and holds 5 patents. He is the head of the "Architecture" department of IRISA.*

Des Autom Embed Syst
DOI 10.1007/s10617-015-9160-2

FoRTReSS: a flow for design space exploration of partially reconfigurable systems

François Duhem · Fabrice Muller · Robin Bonamy · Sébastien Bilavarn

Received: 28 June 2014 / Accepted: 20 February 2015

Abstract In this paper, we present a flow enabling design space exploration for partially reconfigurable systems with real-time constraints, called FoRTReSS. FoRTReSS allows estimating mixed hardware/software implementations of an application where the hardware design space, the floorplanning of reconfigurable regions placed on the FPGA, is automatically inferred from application resources information, interface constraints and the target device. Real-time constraints are verified by a highly configurable SystemC simulator, RecoSim, handling applications described as control data flow graphs (CDFGs). We demonstrate our approach on an H.264 video decoder and an H.265 encoder targeting the latest Zynq-7000 platforms from Xilinx, embedding a Cortex-A9 dual-core processor. We show that an hardware/software implementation of the H.264 decoder using both processor cores and slice decomposition is possible under real-time constraints, effectively achieving a framerate of 30 frames per second while reducing area requirements compared to a static implementation, using 54 % less slice resources and 44 % less BRAM resources. Additionally we report the ability of the methodology to address very early analysis from high level application specification on the example of an H.265 encoder.

Keywords Reconfigurable architecture · Design space exploration · Real-time systems · Partial reconfiguration · Field programmable gate arrays

F. Duhem · F. Muller (✉) · R. Bonamy · S. Bilavarn
University of Nice-Sophia Antipolis - CNRS/LEAT, 930 Route des Colles, BP 145,
06903 Sophia Antipolis Cedex, France
e-mail: Fabrice.Muller@unice.fr

F. Duhem
e-mail: duhemfrancois@gmail.com

R. Bonamy
e-mail: robin.bonamy@gmail.com

S. Bilavarn
e-mail: Sebastien.Bilavarn@unice.fr

Published online: 27 February 2015

1 Introduction

The last few decades saw the emergence of reconfigurable computing through Field Programmable Gate Arrays (FPGA). These devices provide a high level of parallelism along with programmability, bridging the gap between programmable processors and high performance, but expensive, application specific integrated circuits (ASIC). Over generations of devices, FPGAs had more and more computing power so that entire systems can now be built into a single device, including processors, hardware accelerators, memory controllers, I/O peripherals and so on. They are called System on Programmable Chip (SoPC). Processors included in the design can be either soft cores (i.e. using the FPGA fabric as resources, for instance MicroBlaze or Nios cores) or hard cores (i.e. integrated within the die, hardwired to the FPGA). In the latter case, the processors are much more powerful than soft cores, hence providing designers with high performance processor-centric architectures like the Xilinx Zynq-7000 devices, based on a dual ARM Cortex-A9 MPCore [1]. However, the hardware part of the Zynq-7000 does not offer many logic cells compared to state-of-the-art Virtex-7 FPGAs built with the same 28 nm technology.

At first, these devices could only be programmed in an all-or-nothing style such that all services and IP cores are stopped during reconfiguration. In the case of communication services, pieces of data might be lost and degrade the Quality-of-Service (QoS). In critical applications, this cannot be tolerated. This led to the development of partial reconfiguration (PR), introduced with the Xilinx XC6200 series, which allows modifying the behaviour of pre-defined reconfigurable regions (RR) without affecting the remaining logic. The circuit functionality can thus be modified at runtime depending on the application requirements and/or execution. Since not all the resources are present on the FPGA persistently, area requirements are reduced. If a careful study is made at design time, it is possible either to switch to a smaller and cheaper FPGA or to add extra features on the target device [2,3].

Despite promising features, PR is still not widely spread in the industry [4]. The major issue concerns designing the systems satisfying application requirements as well as technology-specific constraints inherent to PR systems. For instance, mutualising reconfigurable region resources between multiple tasks is possible, but implies PR-compatible scheduling that takes into account reconfiguration times, which cannot be neglected for applications with severe real-time constraints [5,6]. Moreover, design space exploration (DSE) still remains a great challenge: existing design flows do not allow DSE during early development stages but rather during the late FPGA implementation stage when the cost implied by any architecture modification can be prohibitive. Hence, FPGA engineers prefer relying on existing, well-known and reliable design flows, even if the solution is sub-optimal in terms of logic resources or device cost.

Our contribution in this paper is an extension of previous work described in [7]. The methodology consists of FoRTReSS, a Flow for Reconfigurable archiTectures in Real time SystemS that enables both hardware and software design space exploration for partially reconfigurable applications with real-time constraints, along with its graphical user interface, FoRTReSS Toolbox. Our approach is based on two main steps: first, an architecture is generated in terms of processors and reconfigurable regions. RRs are inferred from synthesis results in the form of netlists and text reports and/or XML (eXtensible Markup Language) task descriptions. The second step consists of the simulation of this architecture using RecoSim, a Reconfigurable simulator written in SystemC. This step automatically verifies whether the application real-time constraints are met with a given QoS.

Since our previous work, a lot of significant improvements have been made on RecoSim and new features have been introduced. First of all, the model of computation changed a lot as

RecoSim now handles Control Data Flow Graphs (CDFGs) instead of DFGs. Diagrams may contain cycles and loopbacks, while tasks can be periodic or not. We also added type information to the interfaces (e.g. AXI, PLB, FIFO...) and refined our communication model in order to provide the user with a more accurate description. Another important extension consists in adding software considerations into FoRTReSS. It is now possible to add processors to the design and defining several implementations for one task. This feature is illustrated by new use cases: an H.264 video decoder and H.265 video encoder with multiple hardware and software implementations of tasks. FoRTReSS provides several Application Programming Interfaces (APIs) to easily modify and/or develop task and scheduling algorithms. The paper also targets a state-of-the-art Zynq device. In fact, any device, already existing or not, can be targeted using a flexible device description using XML.

The remainder of the paper is structured as follows: in Sect. 2, we discuss works related to FPGA floorplanning and existing PR design flows. Section 3 introduces our methodology. In Sect. 4, our approach is validated using two representative applications with performance and architecture results. Finally, future works and conclusions are outlined respectively in Sects. 5 and 6.

2 Related work

2.1 FPGA floorplanning

One primordial problem is the task placement on FPGA led by a placement algorithm. Two categories of algorithms are clearly identified: off-line and on-line. In the first category, it is possible to investigate a near-optimal or optimal solution because the off-line scenario is found before the execution of the system. In the second category, the placement decision must be taken quickly because time is very critical. Anyway, RR placement for partially reconfigurable systems is necessarily defined during the design phase. Hence, we only discuss below on off-line floorplanning techniques.

The placement of hardware tasks on an FPGA is an NP-Complete problem and the time to reach a solution depends mainly on the number of the tasks. The work in [8,9] introduces an exact resolution of scheduling problems. These approaches are very time-consuming and are not scalable. Rather than finding the best solution, it is preferable to find a near-optimal solution in a reasonable amount of time, based on heuristics. For instance, authors in [10] define 3D FPGA templates in time and space dimensions and use simulated annealing and greedy research heuristics to place Reconfigurable Functional Unit Operations (or RFUOPs). Moreover, Lodi et al. propose in [11,12] different off-line approaches to resolve hardware task placement as 2D bin-packing problem for instance Floor-Ceiling algorithm and Knapsack packing algorithm. However, these approaches are not adapted for real-time embedded systems because the scheduling of tasks has to be known at compile time.

Authors in [13] introduce an approach based on simulated annealing in order to find out the bigger common area between two reconfigurable zones. This common area will not be modified and the reconfiguration overhead will decrease. Authors also consider traffic congestion in the design brought by placing reconfigurable regions close to each other, which helps to remove infeasible solutions in many designs.

In [14], authors present a resource- and configuration-aware floorplacement framework that uses metrics such as external wire length (total length of wires connecting reconfigurable regions) to qualify a solution. They report an average improvement of 50 % when using this metric. Their main objective is to group reconfigurable units together without taking

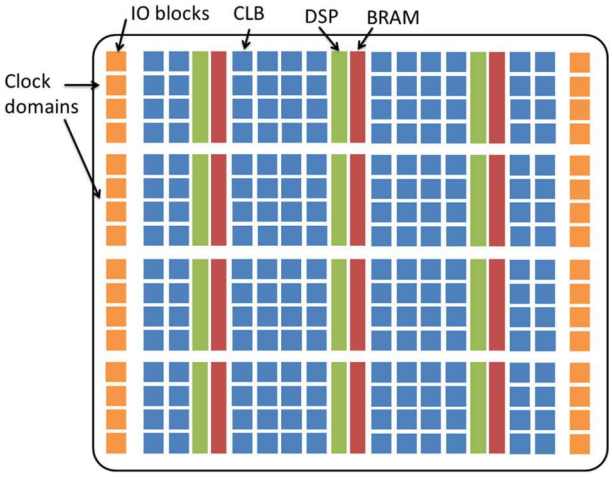


Fig. 1 Physical disposition of resources inside an FPGA

into account the heterogeneity of FPGA resources. Indeed, current FPGAs (Fig. 1) include different partially reconfigurable resources such as logic blocks (e.g. Configurable Logic Blocks), memories (e.g. Block Random Access Memory) and Digital Signal Processing blocks. A column includes only one kind of resource. Hence, a flexible model is essential because the structure of FPGAs differs.

A methodology for an architecture-aware and reconfiguration-centric floorplanning is introduced by [15]. To obtain the solution, a cost function is defined by the total wire length and wastage of resources without taking into account the task dependencies or timing constraints for the application. However, the real-time aspect is essential and must be considered for partially reconfigurable systems. A lot of application domains such as robotics, video streaming, automotive, avionics and so on, depend on these constraints.

An approach similar to FoRTReSS has been developed by authors in [16]. This methodology, called FoRSE, (standing for Formulation-level partial Reconfiguration design Space Exploration) performs a mathematical-based exploration of the design space, looking for a Pareto optimum for the application, considerably reducing the exploration time compared to methodologies requiring implementation (implementation-level versus formulation-level). Nevertheless, this formal method does not take into account potential real-time constraints. However, it is worth noting that this methodology relies on Xilinx FPGA models to represent an FPGA device.

According to our knowledge, the works on floorplanning for partial reconfiguration are often treated separately or even are not studied at all. We think that these two problems must be considered together, as the floorplan problem is often meaningless without the communication channels to support it.

2.2 PR design flows

To complete this overview of methods for partial reconfiguration, it is also important to give a clear picture of partial reconfiguration design flows. Xilinx was the first company to introduce such a feature for their FPGAs [17]. Their flow inspired Virginia Tech's open-source tool, OpenPR [18]. Altera also recently unveiled a new version of Quartus enabling

dynamic and partial reconfiguration for their state-of-the-art Stratix V FPGAs [19] with a PR flow pretty much similar to Xilinx's one. Another interesting flow is GoAhead [20], an academic tool providing some new partial reconfiguration features such as module relocation (an extensively adressed subject [21,22]). GoAhead also allows mapping two reconfigurable modules simultaneously inside the same PR region (which is also possible using only Xilinx tools but with an important extra design effort). We believe that a design methodology should not be (or the least possible) technology-dependent and also extendable to virtual FPGAs and architecture exploration of physical FPGAs.

FoRTReSS addresses these issues by providing designers with a feasible floorplan for state-of-the-art heterogeneous devices and a task scheduling that satisfies the application timing constraints. The FoRTReSS device model is flexible in order to be compliant with future device architectures or virtual FPGA platforms. Finally, FoRTReSS can handle task and RR interfaces to represent various communication models (e.g. point to point, shared bus...).

3 Our approach

3.1 FoRTReSS overview

FoRTReSS is a tool providing the user a way to explore the partial reconfiguration design space and ultimately proposing a set of reconfigurable regions and processors that will ensure a certain Quality-of-Service (QoS) for a given application. The term quality of service refers to the rate of task executions that respected their deadline. For instance, hard real time applications would typically require a QoS of 100 % whereas in applications such as video streaming, it is acceptable to lose some packets and have a degraded QoS.

Figure 2 shows an overview of the FoRTReSS flow. It is based upon a Y-chart approach where application and architecture are described separately. The application is described as a control data flow graph (CDFG) or a set of periodic tasks with dependencies. Each task has some timing characteristics such as a deadline or a period (zero for non-periodic tasks), and a set of possible implementations with different performance, resource and energy trade-offs. These implementations can be hardware (i.e. to be mapped on a reconfigurable region) or software (i.e. to be mapped on a processor core) and share a set of parameters such as the task best/worst case execution time (BCET/WCET). Some other parameters are different from one type of implementation to another. Typically, hardware implementations are defined by their resource requirements resulting from synthesis. This information can be extracted from Xilinx synthesis reports or from XML-based files (.tsk extension) developed for compliance with other synthesis tools. Describing task hardware implementation is mandatory in order to determine a reconfigurable region set which might fit the application. FoRTReSS might also require full netlists of each implementation (Xilinx NGC or standard EDF) in cases where compressed bitstreams are used to optimise reconfiguration times (see Sect. 3.3.6 for more details). On the other hand, software implementations are characterised similarly by the time required to load the binary executable into the instruction memory of the processor.

The target architecture is described separately as an FPGA and a set of processor cores that can be either on the same die (Virtex-5 with integrated PowerPC or the latest Xilinx Zynq-7000 SoC with a CortexA9 processor), external to the FPGA or soft cores instantiated with its configurable logic resources (e.g. MicroBlaze). The FPGA architecture is also described using an XML-based file format in order to be compatible with existing devices as well as custom, virtual or non-existent (future) architectures.

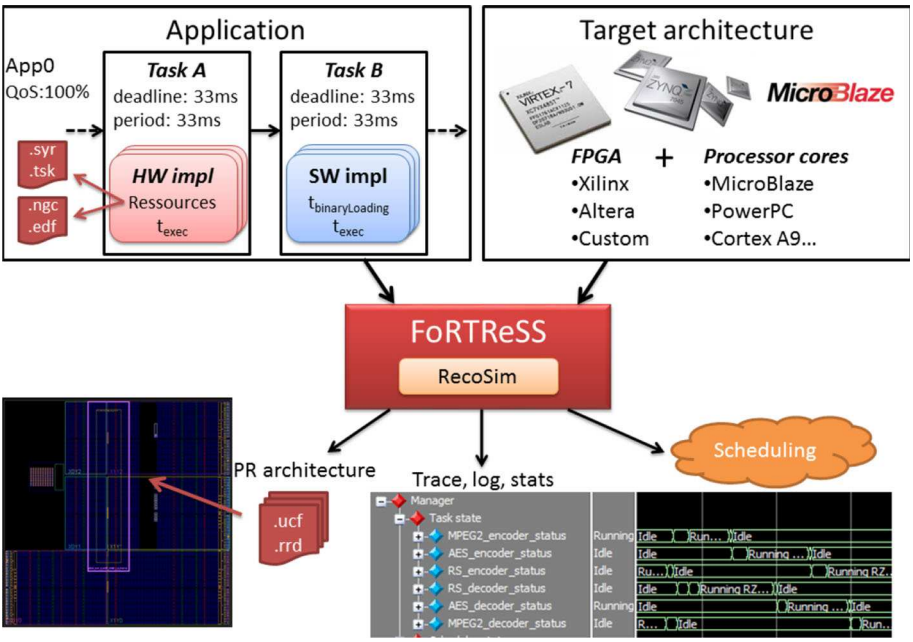


Fig. 2 PR flow using FoRTReSS

FoRTReSS uses the application resource requirements and FPGA description to find potential reconfigurable regions. The validation of the application quality-of-service is carried out using a SystemC based simulator called RecoSim. Tasks are scheduled under a standard earliest-deadline-first (EDF) policy integrated in this simulator and that can be modified or extended to other scheduling policies due to a dedicated application programming interface (API). RecoSim also generates traces, statistics and log files for every simulation for debug purposes. Finally, FoRTReSS provides an architecture fully defined in terms of RR by an UCF file (User Constraints File, used by Xilinx) and an XML representation of the regions (.rrd extension). The resulting floorplan can be viewed using the Xilinx PlanAhead design tool as shown in Fig. 2.

The SystemC simulator has already been introduced in a previous publication [23]. However, the model of computation has been improved since in order to better process the simulation of real time constraints. The following sections give an update on these features and a description of the underlying FoRTReSS flow in order to better understand the set of parameters impacting the design space exploration process.

3.2 RecoSim overview

RecoSim, for reconfigurable simulator, is a SystemC/TLM simulator that verifies if an architecture can satisfy real-time constraints of an application. From a description of possible mappings of tasks on the execution units, RecoSim uses transaction-level models of the system to ensure fast simulation while considering abstract communication details, reconfiguration overheads (that can be inferred from cost models such as [24]), context switches, hardware and software preemptions, for a wide range of architectures.

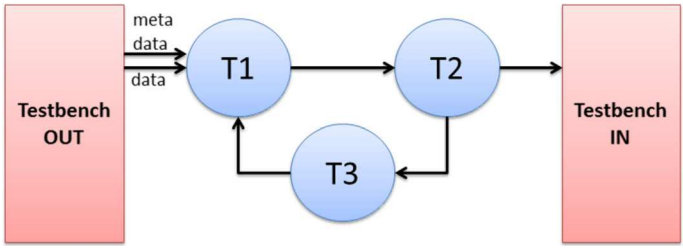


Fig. 3 Example of CDFG handled by RecoSim

3.2.1 RecoSim model of computation

The main input of RecoSim is an application description given in the form of a Control Data Flow Graph, which means that communications from a module to another can be conditional. An example of specification that can be simulated by RecoSim is shown in Fig. 3. Compared to standard data flow graphs (DFGs), CDFGs introduce control at the task level to make conditional communications possible. For instance, task T2 in Fig. 3 can send data to Testbench IN and task T3 independently: it is possible to make T2 send n packets to T3 after each execution (i.e. when new data are generated by the task algorithm) while sending data to the testbench once every m executions only. Another feature of this CDFG model is the ability to describe cyclic applications: on the first iteration, no relevant data are provided by T3 and hence T1 only waits for data incoming from the testbench. Finally, it is also possible to simulate several communication channels linking the same two tasks (see connections between testbench and task T1 in Fig. 3). This can perfectly describe separate data and metadata channels to evaluate different performance between these connections. For example, a metadata channel would use a low speed bus (AXI Lite bus) while the data channel would need a more efficient bus (AXI bus).

The application must be surrounded by two testbenches as depicted in Fig. 3, separating stimuli generation from result verification. Unlike other tasks, we suppose here that testbenches are persistent in the system (i.e. not dynamically reconfigurable). They also have no implementation explicitely defined, however testbench algorithms, which decribes the behavior of data sending or receiving, might be modified using a dedicated API.

Applications can be composed of periodic tasks. The main difference with non-periodic applications resides in the way tasks are started: for non-periodic applications, the task is launched whenever all incoming sockets have sent their data (according to the task algorithm controlling which socket is required for this task execution). In case of periodic applications, it is also required that a new period has started. If not, the task has to wait for this new period and execution is delayed. However, its absolute deadline is still calculated with regard to the task relative deadline (except whenever incoming data are ready like in full dataflow applications).

Either way, the application has to be simulated long enough to ensure that the architecture maintains the required quality of service. The minimum simulation time for periodic applications is a hyperperiod. For DFGs, it is defined as the Least Common Multiple (LCM) of task periods whereas for CDFGs, the additional control part changes this hyperperiod and it is not possible to predict its value automatically. Moreover, in order for every tasks to be running in the same hyperperiod, the system should be in a steady state. We consider that the upper limit for the time required to enter this steady state can be estimated as the sum of

every worst case execution times. Adding this time to the hyperperiod gives an estimation of the minimum simulation time. Note that it is the responsibility of the designer to ensure that this simulation time is respected and can be modified in FoRTReSS flow. The uncertainty brought by CDFGs into the computation of the hyperperiod leads FoRTReSS to notify the user when the simulation is relevant, but the designer might want to manually compute the minimum simulation time (or over estimate it) when designing the testbench.

Let us note $WCET(T_i)$ the Worst Case Execution Time of task i. Equation (1) gives us the minimal simulation time for the system.

$$t_{simulation,min} = LCM(T_1..T_n) + \sum_{i=1}^{n} WCET(T_i) \tag{1}$$

3.2.2 Mapping tasks to processing units

Recosim simulates the mapping of application tasks to the processing units and the corresponding execution (i.e. run-time allocation and scheduling). These processing units are either hardware (reconfigurable regions infered by FoRTReSS for the target FPGA) or software (user-defined processor cores). The mapping decision is made by the reconfiguration manager considering the different software and hardware implementations associated with each task as well as the availability of reconfiguration units: the default behaviour consists in getting the most out of the architecture by using As Many units As Possible (AMAP mapping). The choice of an implementation also depends on parameters such as the configuration time, execution time or energy consumption. A dedicated API is defined to help the definition of specific scheduling and allocation techniques.

3.2.3 Task preemption and context switches

We have already presented the management of preemption for hardware tasks in [23]. Our preemption model is based on a request/grant system where preemption points are explicitely defined in the task implementation code as depicted in Fig. 4. The reconfiguration manager does not preempt the task but it is rather the task that issues a preemption request (cooperative multitasking). This behaviour fits well with the execution of hardware tasks which preemption is more complex than software tasks. Furthermore, context evolves during task execution as the number of registers used to store important data are changing. Hence, it is understandable that context switches, operated by register save/restore operations, should be performed when it has the fewest impacts.

This approach is also compatible with classic software preemption by defining as many preemption points as instructions in the executable. This way, the task will notify the manager after each instruction, making the task preemptible at every moment of its execution, emulating preemptive multitasking. However, the simulation time overhead inherent to this

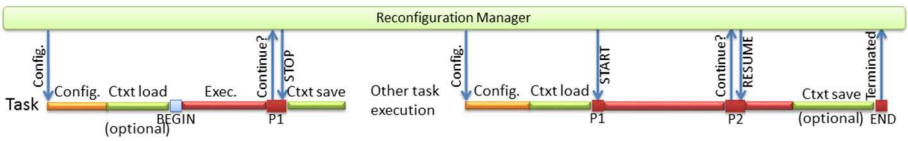


Fig. 4 Task preemption within RecoSim

technique (due to an increase of communication between the module and the reconfiguration manager) can be fairly important but considered reasonable compared to accuracy. For example, we simulated an application composed of 30 tasks with a scenario representing an execution time of 2 s on a Core I7-3740QM running at 2.7GHz with 8GB of RAM. With a preemption resolution of 1 us, the execution time on the computer is 810 s against 1.53 s without preemption. This is a significant increase, but considering the complexity of preemption is at this cost. This overhead will be much closer to 1.53 s in real cases as real applications will only require a few preemption points per task. Nevertheless, this overhead is necessary for an accurate modeling of software tasks. In fact, the relationship between these two quantities is shown in Eq. 2, where N_i is the number of preemption points for the task i.

$$Overhead\ (in\ seconds) = \sum_{i=0}^{TaskNumber} 0.4 \times N_i \tag{2}$$

Using (2), the designer has to perform a trade-off between extreme precision of software preemption, quality of the results (some good solutions might be missed when losing precision in some corner cases) and overall flow execution time.

Figure 4 also illustrates an example of context switch for a task execution. Context save and restore operations are required whenever a task is preempted. There is also an optional context switch at the beginning and at the end of task execution due to the distinction between hardware and software execution. Software context switches are processor dependent whereas for hardware implementations, this context switch depends on the implementation (two implementations of the same task might have different context switch times due to a different number of registers to save/restore, possibly some memories). Some work has been done on context switch for hardware tasks such as [8,25,26]. However, context switching is not required after a task configuration, but might be necessary for proper task execution (for instance, IP configuration). Hence, the designer is given the possibility to enable context switching after configuration for each implementation.

3.2.4 RecoSim task finite-state machine

RecoSim is based on the Finite-State Machine (FSM) depicted in Fig. 5. This FSM represents the states and transitions of the application tasks:

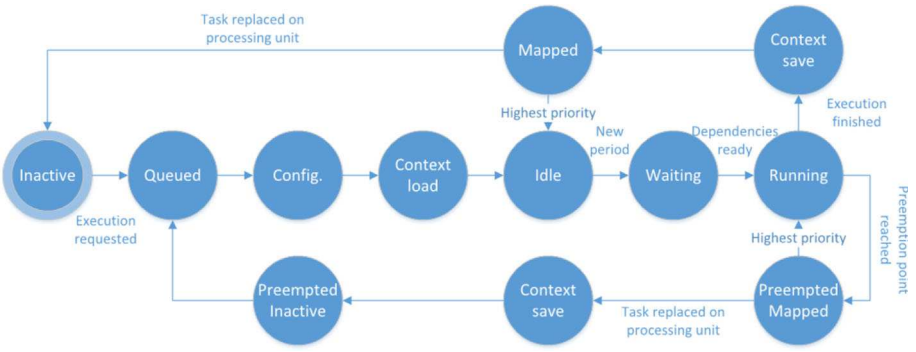


Fig. 5 Task finite-state machine

- *Inactive* task is not running nor placed on the FPGA.
- *Queued* after an execution request has been granted by the scheduler and a processing unit has been chosen, the task is queued, waiting for reconfiguration, ordered in a first-come-first-served basis since task priority is handled within the scheduler waiting queue.
- *Configuration* task is being configured on the FPGA.
- *Context load* loads the task context (for instance after being preempted or if necessary before task execution)
- *Idle* task is idle, waiting for the beginning of next period (transitory for non-periodic applications)
- *Waiting* task is waiting to receive communications from its predecessors
- *Running* task is running
- *Preempted/mapped* task reached a preemption point and notified the reconfiguration manager. Task is not preempted yet.
- *Context save* saves the task context when the task is being preempted. It may also be needed after task execution.
- *Preempted/inactive* task has been actually preempted by the reconfiguration manager and has been replaced on the processing unit. It is brought back into the waiting queue in order to be configured again and resume execution.
- *Mapped* task is placed on a reconfigurable region but not running. It can be safely replaced by a higher priority task by the reconfiguration manager.

3.2.5 Interfaces and communications

It is possible to use different types of interface (e.g. AXI, PLB, FIFO...). However, some tasks might have many interfaces (for instance, task 1 from Fig. 3 has four interfaces: two interfaces with the testbench, one with task 2 and one with task 3) and it seems rather inappropriate to consider exhaustively all possible types of interface (AXI, PLB, FIFO...) for each reconfigurable region hosting this task (especially in case of several bus interfaces). In order to reduce the overall number of interfaces that should be implemented on a reconfigurable region, we introduce the concept of physical and virtual interfaces. Virtual interfaces are the ones required by the task as described on the diagram (the four interfaces of task 1). On the other hand, physical interfaces are the ones actually used in the implementation. The number of physical interfaces can vary from one implementation to another. A situation can occur when there is less physical than virtual interface in the application diagram. In such a case, since it is not possible to use the same interface for distinct but simultaneous data transfers, accesses to the physical interfaces should be made sequentially, using a first-come, first-served policy. In the case where no physical interfaces have been defined for the task implementation, RecoSim automatically uses the information from the diagram to provide the task with as many interfaces as declared in the diagram, to maintain the best performance.

As a matter of fact, virtual interfaces might require a permanent access to a physical channel to optimize performance. Typically, task T1 from Fig. 3 has two virtual input interfaces from Testbench OUT that can represent data and metadata channels. The data channel can be defined as a priority channel: a physical interface is dedicated to this virtual interface while potential other interfaces share the remaining common interfaces. This choice, which is up to the designer, is a mean to reduce data transfer latency, as the cost of an additional resource overhead for the dedicated physical interface.

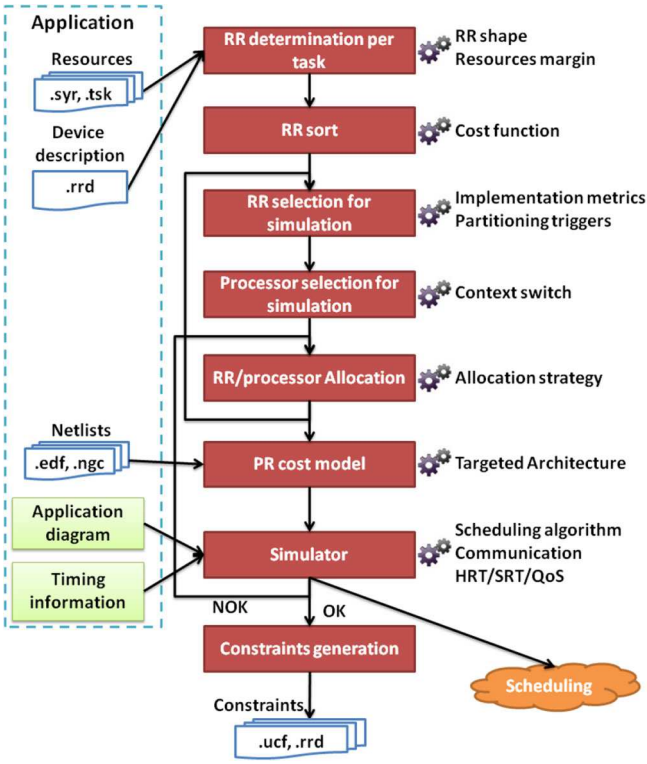


Fig. 6 FoRTReSS flow

3.3 FoRTReSS flow

Figure 6 shows the different steps composing the FoRTReSS flow. Since the original flow is described in [7], only major features and enhancements are described here.

3.3.1 RR determination per task

The first step in the architecture definition process is to determine a pool of reconfigurable regions that are able to host one or more tasks of the application in terms of resources. This reference pool is built by browsing the XML-based representation of the target device resources. These RRs are shaped and mapped on the reconfigurable device with regard to the heterogeneity of the device resources, with possible RR overlapping to have a wider choice of RRs later. They are also built to be as close as possible to the resource requirements of the current task, and trying to waste the smallest amount of resources (this is called internal fragmentation). However, experience shows that defining a region based on minimising the number of resources are generally results in a failure during the routing phase. Furthermore, resources information from the synthesis step does not take routing into account. For instance, Xilinx recommends adding 5 % extra resources to the reconfigurable region. For a more accurate description, FoRTReSS has a parameter for global routing margin that can be overriden for every hardware implementation that might require more routing ressources.

FoRTReSS also takes care of interfaces: as we already mentioned, it is possible to define a set of interfaces for task implementations. In case of hardware implementations, they are also used to constrain physical task placement on the FPGA. These restrictions are inferred from interfaces placed on the FPGA by the designer. It is also possible to prevent a region from being used by any reconfigurable region (static area). Typically, these features can be used to force the placement of reconfigurable regions around the actual location of interfaces (located for instance between the FPGA and Cortex cores in a Zynq-7000 device).

The search for reconfigurable regions is very useful when starting from scratch, but it is also possible to use FoRTReSS when the placement and number of reconfigurable regions are already defined. For this purpose, an XML-based file format describing predefined regions can be read to avoid being forced to explore the reconfigurable region space.

3.3.2 RR sorting

Once we have determined a pool of compatible RRs for each task, it is sorted according to a cost function in order to select the best regions for the application. The cost function is described in Eq. 3.

$$\begin{aligned} Cost_{RR} = {} & k_1 * Cost_{shape} \\ & + k_2 * Cost_{compliance} \\ & + k_3 * Cost_{fragmentation} \end{aligned} \tag{3}$$

The metric $Cost_{RR}$ is formulated around three components. First, it depends on the shape of the RR with the metric $Cost_{shape}$. As mentioned in previous subsection, regions can have different shapes. However, the more you complexify the shape, the less likely routing is to be efficient, this can possibly lead to failure at the floorplanning step. Therefore, we penalize regions with complex shapes by counting their vertices.

$Cost_{compliance}$ refers to the concept of Application Architecture Adequacy (often noted AAA). It takes into account the number of tasks that can be mapped on the reconfigurable region. Bigger regions might host a more important set of tasks, hence giving more freedom to the scheduler for runtime mapping of the application.

Finally, the last metric $Cost_{fragmentation}$ corresponds to the internal fragmentation and is used to penalize regions that have been built with too many resources compared to the task they host. This cost tries to avoid the waste of resources inside a reconfigurable region. This component actually reflects the percentage of unused resources inside a reconfigurable region.

Since all components have different amplitudes ($Cost_{shape}$ takes values from 4 to 10 vertices, $Cost_{compliance}$ from 0 to n incompatible tasks, n being the number of tasks in the application), it is important to weight them in order to share the same dynamics. On top of that, k_1, k_2 and k_3 in Eq. 3 correspond to parameters defined in our flow that can be tuned to promote either the shape, the compliance or the fragmentation component of the cost function. Default values are set to one to give the same importance to all three components.

3.3.3 RR selection for simulation

During this step, the tool picks the RRs that will constitute the system architecture. It searches for the minimum number of RRs that will make the simulation step succeed in order to optimize the partially reconfigurable area. FoRTReSS also addresses the external fragmentation which represents the physical distance between the reconfigurable regions, calculated as

the sum of all Manhattan distances between regions. Low external fragmentation reduces the total wire length and thus provides better results during the implementation phase: the regions are packed on a small part of the device, optimizing the remaining area for static logic.

There are two main approaches for the minimisation of fragmentation: reconfigurable regions should be placed as close as possible to each other or there should exist a minimum distance between them. The last option considers congestion at the interconnect level if regions are too close [13], inducing a drop on the frequency that can be reached by the system. We decided to let this choice up to the designer with a parameter representing the minimum distance between reconfigurable regions.

In order to reduce resource requirements for PR systems, the application is partitioned. If tasks have very different resource needs, i.e. have very different resource needs (which is the case most of the time), small tasks hosted within big regions are in some ways resource inefficient. To prevent this situation, we try to group tasks with similar resource requirements: tasks within the same group will share the same reconfigurable regions. Therefore, tasks with small requirements will not be placed on regions defined for bigger tasks. The first step to partition the application consists in sorting the tasks according to a resource cost function that penalizes the waste of scarce resources. The cost of a reconfigurable resource is inversely proportional to the amount of resources available on the device, while fixing CLB (logic elements) cost to 1.

Then, the tasks are split into three categories according to pre-defined trigger values: *optimum*, *acceptable* and *unacceptable* mapping to reflect adequacy between a task and the biggest RR. An example is given in Fig. 7. *Optimum* tasks are the more expensive in terms of resources and their mapping to bigger reconfigurable regions is relevant (tasks t_5 and t_6 in Fig. 7). *Acceptable* tasks do not waste too much resources within the reconfigurable region (tasks t_3 and t_4) while *unacceptable* tasks represent a clearly bad allocation scheme (tasks t_1 and t_2). Reconfigurable regions from the initial simulation subset (and the biggest ones according to the cost computed in previous section) will host both *optimum* and *acceptable* tasks. New regions are created based on the maximum resource needs of *acceptable* and *unacceptable* tasks, replacing some of the biggest reconfigurable regions in the initial set. These regions cannot physically host the biggest tasks, explaining the area savings that can be obtained by partitioning the application.

Trigger values are user-defined with arbitrary default values of 33 and 66 % (percentage of RR resources use). The designer should set the *optimum* trigger value in order to isolate resource demanding tasks from the others. However when many tasks are isolated, more reconfigurable regions should be used and less area optimisation is also expected in this case. The second trigger prevents tasks with low requirements from being mapped to big regions and should be set so that all partitions are balanced (i.e. containing similar number of tasks). The trigger values should then be updated from one exploration to the other depending on the previous results.

3.3.4 Processor selection for simulation

While FoRTReSS focus is on the determination and placement of reconfigurable regions, it is also possible to design mixed systems including one or more software processing units. However, these elements are not processed exactly the same way as hardware reconfigurable regions. Processor cores are added statically at the beginning of the exploration process while reconfigurable regions are dynamically added to the simulation subset. In a future release, we aim to extend exploration to be able to analyse automatically software and hardware

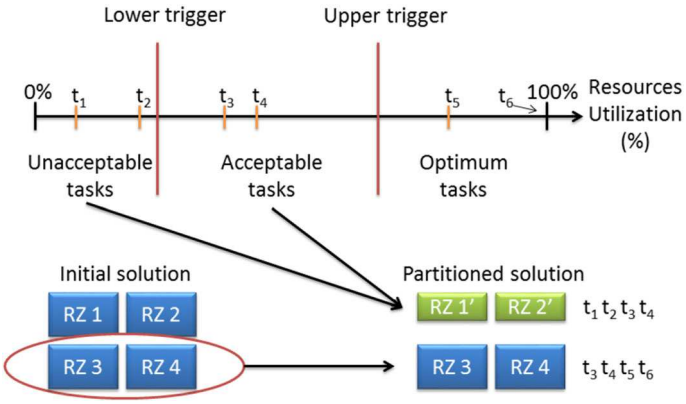


Fig. 7 Partitioning step

implementation opportunities. In the current version, several simulation iterations are needed to consider different number of cores.

A processor type is associated with each software implementation to determine on which core it can be mapped during simulation (e.g. MicroBlaze, PowerPC, Cortex A9 for Zynq platforms). We also defined a context switch parameter to represent the time required to save and restore an execution context. This parameter is processor-dependent.

3.3.5 RR and processor allocation

When an RR is selected for simulation, the tool defines which region is used for each task. This allocation step is the most complex since it is the one giving the greatest degree of freedom to the designer. The most obvious constraint is that the task should fit the RR in terms of resources. However, if the task can be placed on different RRs, this will lead to as many bitstreams stored in memories since bitstream relocation is not currently supported (using the same configuration bitstream for one task on several reconfigurable regions). The memory footprint associated with many configuration bitstreams cannot be neglected in embedded systems where memory can be a scarce resource. Also, larger memories such as DDR have higher access times than FPGA internal memories that can affect reconfiguration times. Hence, it is important not to map tasks to every possible RR but rather limiting the task-RR association to a mimimum for a better memory footprint.

There are two distinct phases in the allocation process. The first phase consists of finding a viable solution, i.e. finding the mimimum number of reconfigurable regions and processors for the application to reach the required QoS. Hence, the first allocation is pretty straightforward and consists in allowing every combination of tasks on RRs to maximise the freedom of the scheduler for on-line task placement. If in spite of this freedom simulation fails, it means that trying to improve the mapping is meaningless and that is it necessary to add another region to the simulation subset. Upon first simulation success, task allocation may be optimized. Furthermore, the first step is greedy in terms of memory usage since there could be a lot of bitstreams to store.

An optimisation based on removing pairs of task/region from the solution is applied using the following strategies:

- *Least used allocation* removes the couple task/RR that is the least used during the simulation (i.e. the association that is most likely to be removed without altering the system performance).
- *Highest internal fragmentation* removes the task wasting the most resources on a reconfigurable region.
- *Highest memory cost* removes the task having the biggest bitstream.

Because memory footprint improvement differs from one use case to another, there is no best optimization strategy. Therefore, the designer is left with the possibility to select the strategy that best suits to the application needs. Note that for the last strategy, involving bitstream size, we do not consider bitstream relocation (using one bitstream for configuring several regions). We will include bitstream relocation management in a future release of FoRTReSS.

Interface related constraints are not taken into account during this step: we consider it is a result of the allocation step. It is thus possible to find situations where an IP with a FIFO interface and an IP with an AXI interface would share the same reconfigurable region. In such case, the reconfigurable region must access both AXI and FIFO interfaces. A common approach is to separate the IP core from the communication interfaces in order to reduce reconfiguration time and resource overheads. Interfaces are actually connected to their associated IP core by a router which is parameterised at configuration time. As grouping IPs with different interfaces are known to be inefficient, improvements are foreseen on these aspects.

This step allocates software implementations to processor cores as well. Compared to hardware mapping, the difference lies in the optimisation strategies than can not be strictly the same: processor level optimization is based on the *Least used allocation* strategy which is the only one implemented in this case but could be extended by another strategies such as a low power strategy.

3.3.6 PR cost model

Before simulating the solution, it is necessary to calculate the reconfiguration times associated with every task-RR association. For this purpose, we use the cost model developped for an optimised reconfiguration controller called fast reconfiguration manager (FaRM) [24]. FaRM also allows for bitstream compression to further reduce the memory footprint without degrading configuration performance.

3.3.7 Simulation with RecoSim

At this point, we can simulate a solution to check whether this architecture is fulfilling the timing constraints of the application. The application is considered frozen and the only parameters that can be modified are those related to the scheduler. For instance, FoRTReSS comes with an earliest deadline first (EDF) scheduling strategy and as many as possible (AMAP) mapping strategy (using as many processing units as possible), but the designer can define custom strategies using dedicated APIs. The time spent by the scheduler to decide the next move is simulated in order for the simulation to be time-accurate. The main objective is to obtain a realistic schedule and, through measurements or an accurate cost model, to ensure compliance with real-time constraints. However, this scheduler is currently under implementation based on previous work described in [27].

RecoSim can simulate the simultaneous mapping of several applications on the target FPGA (an application being a sequence of tasks started and ended by a testbench, just like the example of Fig. 3). In this case, all applications are controlled by the same reconfiguration manager and the same scheduler that can be implemented either in software and hardware. Either way, schedulers are considered static and placed on a dedicated unit.

3.3.8 Layout constraints generation

When simulation has completed, a user constraint file (UCF) is produced describing the placement of reconfigurable regions on the device, compliant with Xilinx design tools. For non-Xilinx devices, FoRTReSS also generates an XML file representing the chosen reconfigurable regions.

3.4 About the solution chosen by FoRTReSS

The solution chosen and validated by FoRTReSS is one amongst many other correct solutions in the design space. FoRTReSS mainly focuses on whether or not the architecture satisfies the application real-time constraints. Once the constraints are met, it is not necessary to continue evaluating other solutions and possibly find a better solution since we already found a valuable one. Still, it is possible to manually search for an optimal solution by running several times the FoRTReSS flow while reducing the timing constraints.

In future versions of the flow, we plan on integrating energy minimisation. The same behaviour is expected: there will be an energy consumption constraint that should be respected as well as the timing constraint. We will not be looking for a trade-off between both metrics but FoRTReSS will rather evaluate the system's feasibility. Therefore, the same approach can be preserved.

3.5 FoRTReSS Toolbox

In order to ease the joint use of FoRTReSS and RecoSim, we developed a graphical user interface (GUI), called FoRTReSS Toolbox [28] which allows the graphical specification of application diagrams and exploration parameters. The eclipse graphical modeling framework (GMF) [29] is used to create user interfaces based on Eclipse editor and eclipse modeling framework (EMF). This environment is used to generate the C++ source code required by FoRTReSS and RecoSim using JDOM [30]. Code generation, compilation and simulation are handled within FoRTReSS Toolbox GUI. Interactions with other design tools such as Xilinx PlanAhead or Mentor Graphics ModelSim are possible to examine details of the simulation results. FoRTReSS Toolbox is compatible with Linux-based and Windows operating systems.

4 Application study & results

This section illustrates the application of FoRTReSS methodology for system level exploration of hardware software mappings involving dynamic and partial reconfiguration. A reasonable specification assumption can be based on using C/C++ as a high level input code. Since the relevance of RR definition depends greatly on reliable characteristics of hardware accelerators, we rely on the use of High Level Synthesis (HLS) made possible by the use

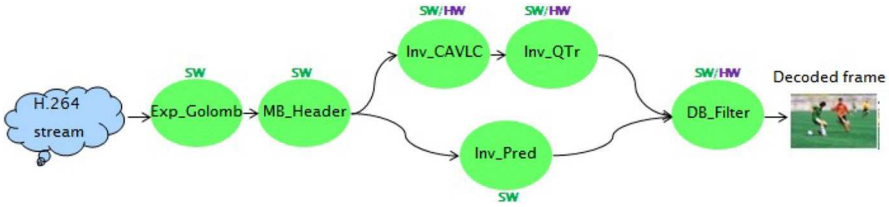


Fig. 8 H.264/AVC decoder block diagram

of C/C++ code. This approach is fully illustrated on the analysis example of an H.264/AVC decoder.

The entire system has not been implemented on the Zynq platform. Each hardware block was synthesized to extract resources and was placed and routed separately on the Zynq device to evaluate its performance accurately (i.e. WCET). Moreover, each software block was run on a Cortex-A9 (performed ten times) in order to have the most accurate worst case execution time. About the data transfer, data is located in main memory (e.g. on-board DDR3 memory). A program, running on the Cortex-A9 and using a DMA, moves data to the accelerator. Regarding software IPs, they fetch data directly from memory. We also integrated this data transfer time in the WCET. Therefore, this leads to a very good accuracy for the simulation in our tool in terms of resource and time.

However, in practice C/C++ code requires a lot of time and effort to be actually compliant with HLS rules, sometimes going as far as complete application rewriting. Therefore in a second validation study, we address the mapping exploration of an H.265/HEVC encoder from a reference C++ code released by the x265 open-source project [31]. As this type of code is often not able to comply with HLS requirements, we show how exploration can nevertheless be usefully processed from existing software and relevant hardware implementations reported in the literature as explained in Sect. 4.4, in order to assess early mapping opportunities and their impact on performance.

4.1 H.264/AVC decoder overview

The first application which is considered for this validation study is an H.264/AVC profile video decoder. An Electronic System Level (ESL) design methodology [32] is used here to provide values of cost performance tradeoffs for possible hardware functions, which serve as an entry point to the exploration methodology of FoRTReSS. The H.264 decoder used corresponds to the block diagram of Fig. 8 which is a version derived from the ITU-T reference code [33] to comply with hardware design constraints and HLS. From the original C++ code, a profiling step identifies four main functionalities for acceleration that are, in order of importance, the deblocking filter (24 %), the inverse context-adaptive variable-length coding (Inv. CAVLC 21 %), the inverse quantization (Inv. Quant. 19 %) and the inverse integer transform (Inv. Transf. 12 %). To achieve better results, we have merged the inverse quantization and integer transform into a single block (Inv. QTr.). It might be noted here that CAVLC was not added to this block because the HLS tool (Catapult C Synthesis 2009a Release) could not handle the complexity of the resulting C++ code. Therefore, this results in three potential hardware functions representing 76 % of the total processing time.

The deblocking filter, inverse CAVLC, and inverse quantization and transform block are the three functionalities of the decoder that can be either implemented in software or in dedicated hardware. In addition to these accelerating oportunities, we also consider a par-

Table 1 H264 task parameters on Zynq-7000 EPP (no slice decomposition)

$Task$	SW_{ex}	HW_{ex}			
	WCET (ms)	WCET (ms)	SLICE	DSP	BRAM
Exp_Golomb	1.96	n/a	n/a	n/a	n/a
MB_Header	1.96	n/a	n/a	n/a	n/a
Inv_CAVLC	20.56	5.05	3383	0	6
Inv_QTr	30.35	15.48	1202	3	7
Inv_Pre	8.81	n/a	n/a	n/a	n/a
DB_Filter	23.50	6.50	701	0	5

Table 2 H264 task parameters on Zynq-7000 EPP (two slice decomposition)

$Task$	SW_{ex}	HW_{ex}			
	WCET (ms)	WCET (ms)	SLICE	DSP	BRAM
Exp_Golomb	1.96	n/a	n/a	n/a	n/a
MB_Header	1.96	n/a	n/a	n/a	n/a
Inv_CAVLC	10.28	2.53	3383	0	6
Inv_QTr	15.18	7.74	1202	3	7
Inv_Pred	4.41	n/a	n/a	n/a	n/a
DB_Filter	11.75	3.25	701	0	5

allelization of the video decoder which exploits the possibility of slice decomposition of frames in the H264/AVC standard. A slice represents an independent zone of a frame, it can reference other slices of previous frames for decoding; therefore decoding one slice (of a frame) is independent from another (slice of the same frame). This way, the decoder can process different slices of a frame in parallel. We have thus considered two versions of the decoder corresponding to i) the original decoder (no slice decomposition), and ii) a two slice decomposition of the image where two streams can be processed in parallel on two halves of a same frame. This will allow considering implementations up to six accelerators and two processors for exploration. The corresponding hardware and software task parameters are reported in Table 1 for the original decoder and Table 2 for a two slice decomposition.

These parameters constitute the inputs for the exploration methodology. Hardware execution parameters (section HW_{ex} in Tables 1, 2) are derived from the full implementation of the three hardware functions identified previously on a Xilinx Zynq-7000 Extensible Processing Platform. Software tasks (section SW_{ex} in Tables 1, 2) are described over the 667MHz ARM CortexA9 processor. The following section provides exploration results, analysis and discussion relevant to video processing constraints for this application example.

4.2 H.264/AVC decoder exploration results

The aim of this design space exploration is to determine if it is possible to decode an H.264 video stream in real-time, i.e. processing 30 frames per second (fps). For this purpose, we studied different use cases, from full software implementations towards mixed solutions, with or without making use of the slice decomposition possibility. Since FoRTReSS does

Table 3 Performance and area results of H.264 decoder implementations

Implementation	Framerate (fps)	Resource type	Static area	Number of RRs	PR area (columns)	PR area	Improvement (%)	
							Raw	Total
Full SW	11.4	Slice	n/a	n/a	n/a	n/a	n/a	n/a
1 slice		BRAM						
1 core		DSP						
Full SW	21.7	Slice	n/a	n/a	n/a	n/a	n/a	n/a
2 slices		BRAM						
2 cores		DSP						
HW/SW	23.3	Slice	5551	1	72	3600	35.15	29.40
1 slice		BRAM	18		1	10	44.4	44.4
1 core		DSP	3		1	20	−566.7	−566.7
HW/SW	28.2	Slice	11,102	4	140	7000	36.95	25.45
2 slices		BRAM	36		4	40	−11.11	−11.11
1 core		DSP	6		3	60	−900	−900
HW/SW	34.1	Slice	11,102	4	140	7000	36.95	25.45
2 slices		BRAM	36		4	40	−11.11	−11.11
2 cores		DSP	6		3	60	−900	−900
HW/SW	30	Slice	11,102	2	88	4400	60.37	54.62
2 slices		BRAM	36		2	20	44.44	44.44
2 cores		DSP	6		1	20	−233.3	−233.3

not automatically explore the number of processor units, it was run with different projects increasing the number of processors. Reconfigurable regions are then automatically adjusted to fit with the configuration.

Performance and area results reported by FoRTReSS are summed up in Table 3. Area results are provided for both static and partially reconfigurable solutions in a way to put forward the relative improvements of dynamic reconfiguration. Raw improvement is computed by comparing resources of a static solution against resources required by reconfigurable regions. A total improvement is also computed considering the additional resources used by the reconfiguration controler (FaRM). For every use case presented here, we took care of choosing the shortest possible deadline, hence achieving the best framerate for the H.264 decoder. When considering a HW/SW implementations, global performance is usually limited by the number of processor units and reconfigurable regions which are directly related to the size of the target device. In this application study, we may use both Cortex-A9 cores of the Zynq-7000 platform.

The second column labelled "Framerate" in Table 3 shows that the first solution complying with a 30fps constraint is HW/SW implementation using two CPUs for a two slice decomposition (34.1 fps). For this solution, the resource improvements from the use of partial reconfiguration are 25.45, −11.11 and −900 % respectively for slice, BRAM and DSP blocks. This means that BRAM and DSP requirements have actually increased while the number of slices decreased in a small fraction. Therefore, this overhead might not promote the use of PR. However, this solution can be improved: the maximum framerate that can be reached with this architecture is more than the targeted framerate of 30 frames per second.

Table 4 Occupation rate results of H.264 decoder implementations

Implementation	Framerate (fps)	Core (%)		Reconfigurable Region (%)					
		Core1	Core2	RR1	RR2	RR3	RR4	RR5	RR6
HW/SW (2 slices/cores)	34.1	48.42	43.37	26.39	11.08	26.39	11.08	n/a	n/a
HW/SW (2 slices/cores)	30	88.09	73.40	n/a	n/a	n/a	n/a	9.74	30.77

Hence, the application deadline can be increased to 33.3 ms: this is the last use case in Table 3. Proceeding this way brings significant improvement with 54 % slice improvement (less than half of previous slice requirements, from just reducing the framerate constraint) and even 44 % BRAM improvement.

Across all implementations, we notice a important overhead for DSP resources in Table 3. It is due to the limited DSP requirements (only 3 DSP48s) and reconfiguration granularity: each reconfigurable region must span an entire clock domain, hence constraining every resource within the column. For DSP resources, each column is composed of 20 DSP48s, which is the minimum number of DSP48s that can be set for a Zynq-7000 device. Therefore, there will always be an important resource overhead for DSP elements. If this is unacceptable to the designer (for instance, if the remaining static logic is very DSP-consuming), it is always possible to change synthesis parameters in order to prohibit the synthesizer from inferring DSP blocks and use logic elements instead.

To better understand the results obtained when reducing the targeted framerate, the Table 4 shows the occupation rates of the last two solutions of the Table 3 depending on the framerate and the possible ICAVLC allocation (RR or core). Reducing the framerate gives the scheduler the opportunity to use a slower task implementation (the software one) rather than a faster hardware implementation, but with a resource overhead in return. The 30 fps use case shows much higher occupation rates than the 34.1 use case (73.4 and 88.09 % compared to 43.37 and 48.42 % respectively for both cores).

4.3 Adding interfaces constraints to FoRTReSS

Solutions found by FoRTReSS did not take into account any location constraints and hence, placement of the reconfigurable regions on the FPGA is arbitrary. However, our application mixes both hardware and software tasks communicating with each other. Hence, task placement can be optimized by placing the reconfigurable regions closer to the Cortex-A9 processors.

Constraining regions is automated within FoRTReSS: for every hardware implementation in the design, the interface requirements are specified so that every region that can host the task must also provide this type of interface. In our case, all accelerators use a 32-Bit AXI high-performance slave port (at 100MHz) between the programmable logic zone and the processing system. Then, the location of potential interfaces has to be specified by the user. Figure 9 shows the location of the AXI interfaces as well as the resulting floorplan of the Zynq-7000 platform for previous use case (two RRs and two processor cores working on two slices at 30 frames per second). In this case, we defined two possible locations for AXI interfaces, close to the Cortex-A9 processor. In order to comply with the reconfiguration granularity, the possible interface locations span an entire clock domain. We can see that both reconfigurable regions found by FoRTReSS effectively include one AXI interface, ensuring enhanced performance during the FPGA implementation phase.

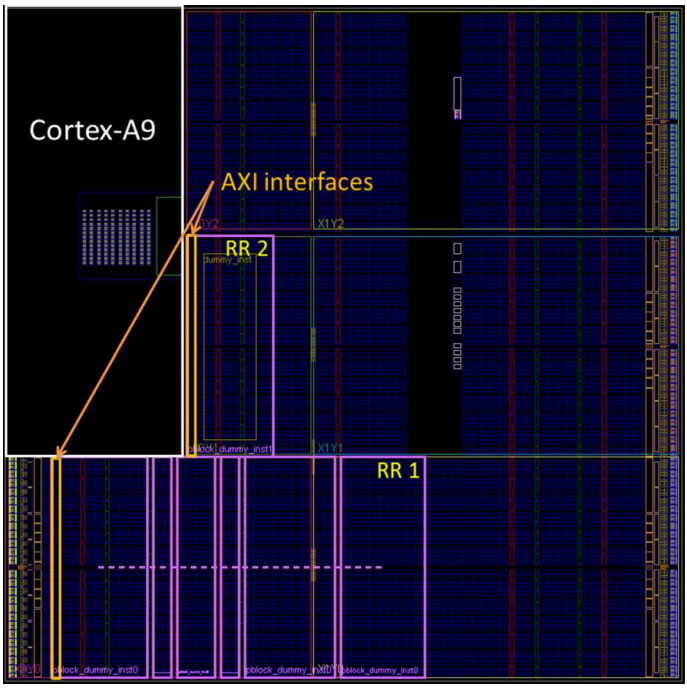


Fig. 9 Floorplan with interface constraints on a Zynq-7000 platform

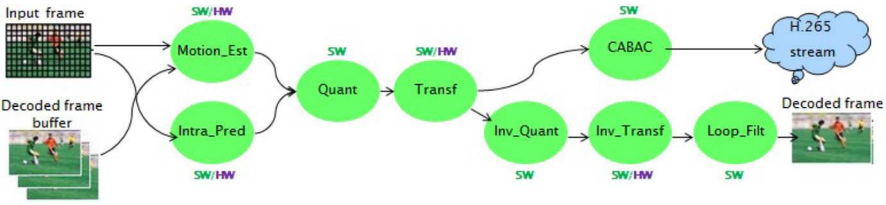


Fig. 10 H.265/HEVC encoder flow graph

4.4 H.265/HEVC encoder

In this second application study, we show the ability of the method to help system level mapping analysis from early-stage application specifications. The application considered is a recently released H.265/HEVC encoder from the x265 open source project which software is available since July 2014 [31]. Figure 10 shows the application graph derived from the reference C++ code.

Ideally the characteristics of hardware mappings of tasks needed for exploration can originate from HLS, provided that the C/C++ source can be processed. This is generally not the case for most applications, including x265, as only a subset of C/C++ is usually supported for hardware acceleration. The x265 code makes widespread use of dynamic memory allocation and arbitrary indirection (pointers that are not static arrays) that will require extensive effort to remove. However many works have already been devoted to the

Table 5 x265 task parameters on Zynq-7000 EPP (no slice decomposition)

$Task$	SW_{ex}	HW_{ex}			
	WCET(ms)	WCET(ms)	SLICE	DSP	BRAM
Motion_Est	2418	9.61	14,067	0	297
Intra_Pred	126	1.43	472	0	4
Quant	37	n/a	n/a	n/a	n/a
Tranf	21	0.28	3595	0	0
CABAC	139	n/a	n/a	n/a	n/a
Inv_Quant	5	n/a	n/a	n/a	n/a
Inv_Transf	5	0.28	3595	0	0
Loop_Filt	23	n/a	n/a	n/a	n/a

Table 6 x265 task parameters on Zynq-7000 EPP (two slice decomposition)

$Task$	SW_{ex}	HW_{ex}			
	WCET (ms)	WCET (ms)	SLICE	DSP	BRAM
Motion_Est	1209	4.81	14,067	0	297
Intra_Pred	63	0.72	472	0	4
Quant	19	n/a	n/a	n/a	n/a
Tranf	11	0.14	3595	0	0
CABAC	70	n/a	n/a	n/a	n/a
Inv_Quant	3	n/a	n/a	n/a	n/a
Inv_Transf	3	0.14	3595	0	0
Loop_Filt	12	n/a	n/a	n/a	n/a

implementation of critical processing blocks of H.265/HEVC since the technical content of HEVC was finalized at the beginning of 2013.

Table 5 reports the characteristics of three relevant hardware implementations on Virtex FPGAs that can be used to process an early exploration of the acceleration potential with dynamic reconfiguration: a motion estimation engine [34] which is the most computational part of the encoding process, a high performance intra prediction hardware [35] and an accelerator supporting fast forward and inverse two-dimensional transforms [36]. All implementation and performance results come from Xilinx Virtex devices and have been extrapolated to comply with the same video resolution and a running frequency of 100MHz. We target the same platform (Zynq-7000 EPP) as depicted in Sect. 4.1. As for H.264, H.265 supports slice decomposition to allow frame level parallelism when encoding using multiple cores. Task parameters corresponding to a two slice decomposition are thus also considered and depicted in Table 6.

4.5 H.265/HEVC encoder exploration results

Like for the previous H.264 application study, we cover different configurations from full software to hardware software implementations with or without slice decomposition. The corresponding results are reported in Table 7.

Table 7 Performance and area results of x265 encoder implementations

Implementation	Framerate (fps)	Resource type	Static area	Number of RRs	PR area (columns)	PR area	Improvement (%)	
							Raw	Total
Full SW	0.36	Slice	n/a	n/a	n/a	n/a	n/a	n/a
1 slice		BRAM						
1 core		DSP						
Full SW	0.7	Slice	n/a	n/a	n/a	n/a	n/a	n/a
2 slices		BRAM						
2 cores		DSP						
HW/SW	4.58	Slice	22,673	3	552	27,600	−21.7	−26
1 slice		BRAM	301		32	320	−6.31	−6.31
1 core		DSP	0		0	0	0	0
HW/SW	5.33	Slice	22,673	3	552	27,600	−21.7	−26
1 slice		BRAM	301		32	320	−6.31	−6.31
2 cores		DSP	0		0	0	0	0
HW/SW	8.82	Slice	22,673	3	552	27,600	−21.7	−26
2 slices		BRAM	301		32	320	−6.31	−6.31
2 cores		DSP	0		0	0	0	0

Fig. 11 Scheduling of the best X.265 accelerated solution

There are sensitive performance benefits in the three hardware software solutions defined, until 8.82 fps using two cores and three RRs over software execution (respectively 0.36 fps and 0.7 fps for 1 slice/1 core and 2 slices/2 cores). However this use case is very much affected by the presence of a big computation kernel represented by motion estimation which engages 14,067 slices, that is 65 % of the total accelerator slices. There is no room for sharing and dynamically reconfiguring a single RR able to host all tasks without an important performance penalty (26.8 ms of reconfiguration time). As a result the best solution is based on the use of three RRs, one for each type of function (*Tranf* and *Inv_Transf* functions share the same accelerator IP). There is no possible resource improvement from using dynamic reconfiguration in this example. In fact there is even a slight increase of logic resource due to the inevitable oversizing of a Reconfigurable Region to be able to host a task.

Nevertheless this application study shows the ability of the methodology to easily evaluate the relevance of using dynamic reconfiguration or not from very early development stages. In addition, further investigation can permit to propose and analyze a set of possible improvements. For example, the scheduling details of the 8.82 fps solution reported in Fig. 11 clearly shows that CABAC function becomes a new major bottleneck after global

acceleration. Therefore a hardware implementation for this function, but also for loop filters (sample adaptive offset, deblocking filter) to a lesser extent, can certainly allow to reach real time processing (e.g. 30 fps) and, depending on their size, change the question of dynamic reconfiguration impact. We did not process these functions as we found no hardware implementation reported in the literature yet.

4.6 Design time with FoRTReSS

The design cycle with FoRTReSS can actually be split in two: application specification using FoRTReSS Toolbox and actual execution of the FoRTReSS flow. The graphical specification of the application diagram and FoRTReSS configuration takes less than an hour, which does not include the time required to obtain the resource requirements for every hardware implementation of the tasks and the timing information. This effort is done just the first time an application is defined: diagrams can be imported and reused into multiple projects. Then, each execution of the FoRTReSS flow can take up to a minute, depending on the complexity of the RecoSim simulation and the unknown number of allocation improvements that are performed during exploration. Therefore, it has also been shown that it was possible to estimate the impact of partial reconfiguration on an application in a short amount of time, which cannot be done with current design flows. The H.265 encoder mapping example was processed from scratch in a matter of three days. To do this, we needed descriptions of hardware implementations or at least an estimation of the required resources and the corresponding hardware and software execution times.

5 Future work

In future versions of FoRTReSS, we would like to add some energy considerations. For now, FoRTReSS highly focuses on real-time constraints to automatically identify performance compliant solution(s) under energy minimisation requirement for example. We believe that combining both temporal and energetic considerations for partially reconfigurable systems into a single tool can be very time-saving for the designer interested in power efficiency. For instance, it is possible to develop and evaluate scheduling algorithms with RecoSim that promote energy consumption when considering selecting task implementations and allocation or using blank reconfigurable regions to minimise the global energy cost while potentially considering DVFS oportunies at the processor level for realistic multicore low power execution. Therefore, we would like to integrate these considerations into the reconfigurable region selection and the allocation optimization process.

6 Conclusion

In this paper, we introduced FoRTReSS, a flow enabling design space exploration for partially reconfigurable applications in real-time systems. FoRTReSS provides the designer with a convenient tool for estimating hardware, software and mixed solutions using partial reconfiguration. The hardware design space exploration is automated by FoRTReSS, which infers a set of reconfigurable regions from task resources information. FoRTReSS relies on a SystemC simulator, RecoSim, that allows the designer to develop and evaluate its own scheduling algorithms. We described in depth the features and abilities of FoRTReSS on a H.264 video decoding application targeting a framerate of 30 frames per second on a Zynq-7000 plat-

form. Several implementations of the decoder were evaluated, from a full software solution working on the entire stream to a solution using hardware accelerators and working on a two slice decomposition of the stream. We have shown that a first solution was indentified by FoRTReSS to process a framerate of 34.1 frames per second, but with an important resources overhead. Finally, extending slightly the deadlines permitted to reach a 30 fps solution with very interesting area improvements, saving more than half the slice resources compared to a static implementation and using 44 % less memory resource. We have additionally shown the usefulness of the framework to help analyzing mapping opportunities from early C++ specifications on a H.265/HEVC encoder. The different explorations performed within this work were facilitated by the use of FoRTReSS Toolbox, a GUI for controlling the FoRTReSS flow.

Acknowledgments This work was carried out in the framework of project ARDMAHN [37] sponsored by the French National Research Agency under grant ANR-09-SEGI-001, which aims at developing methodologies for home gateways integrating dynamic and partial reconfiguration. This work is also carried out under the BENEFIC project (CA505), a project labelled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

References

1. Xilinx (2012) EPPs: the ideal solution for a wide range of embedded systems
2. Kao C (2005) Benefits of partial reconfiguration. Xcell J 55:65–67
3. Paulsson K, Hübner M, Bayar S, Becker J (2008) Exploitation of run-time partial reconfiguration for dynamic power management in Xilinx Spartan III-based Systems. In: international conference on field programmable logic and applications (FPL), pp 699–700
4. Manet P, Maufroid D, Tosi L, Gailliard G, Mulertt O, Di Ciano M, Legat JD, Aulagnier D, Gamrat C, Liberati R, La Barba V, Cuvelier P, Rousseau B, Gelineau P (2008) An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications. EURASIP J Embed Syst 2008:1:1–1:11
5. Compton K, Hauck S (2002) Reconfigurable computing: a survey of systems and software. ACM Comput Surv 34(2):171–210. doi:10.1145/508352.508353
6. Tessier R, Burleson W (2001) Reconfigurable computing for digital signal processing: a survey. J VLSI Signal Process Syst 28:7–27
7. Duhem F, Muller F, Aubry W, Le Gal B, Ngru D, Lorenzini P (2013) Design space exploration for partially reconfigurable architectures in real-time systems. J Syst Archit (JSA) 59(8), 571–581 (2013). doi:10.1016/j.sysarc.2013.06.007. URL http://www.sciencedirect.com/science/article/pii/S1383762113001215
8. Belaid I, Muller F, Benjemaa M (2010) New three-level resource management enhancing quality of off-line hardware task placement on fpga. EURASIP Int J Reconfig Comput (IJRC)
9. Belaid I, Muller F, Benjemaa M (2011) Static scheduling of periodic hardware tasks with precedence and deadline constraints on reconfigurable hardware devices. EURASIP Int J Reconfig Comput (IJRC), Article ID 591983
10. Bazargan K, Kastner R, Sarrafzadeh M (2000) Fast template placement for reconfigurable computing systems. IEEE Design Test Comput 17(1):68–83. doi:10.1109/54.825678
11. Lodi A, Martello S, Vigo D (1999) Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In: Proceedings of the Meta-Heuristics. Springer, New York, US, pp 125–139
12. Lodi A, Martello S, Vigo D (1999) Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS J Comput 11(4):345–357
13. Singhal L, Bozorgzadeh E (2007) Multi-layer floorplanning for reconfigurable designs. IET Comput Digital Tech 1(4):276–294
14. Montone A, Santambrogio MD, Redaelli F, Sciuto D (2011) Floorplacement for partial reconfigurable FPGA-based systems. Int J Reconfig Comput 2011:2:1–2:12
15. Vipin K, Fahmy SA (2012) Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In: Proceedings of the 8th international symposium on applied reconfigurable computing (ARC), pp 13–25

16. Kumar R, Gordon-Ross A (2011) Formulation-level design space exploration for partially reconfigurable fpgas. In: international conference on field-programmable technology (FPT), pp 1–6
17. Xilinx (2012) Partial Reconfiguration User Guide
18. Sohanghpurwala AA, Athanas P, Frangieh T, Wood A (2011) OpenPR: an open-source partial-reconfiguration toolkit for Xilinx FPGAs. In: IPDPS Workshops. IEEE, pp 228–235
19. Altera (2012) Quartus II Handbook Version 12.1, Vol 1: design and synthesis—design planning for partial reconfiguration
20. Beckhoff C, Koch D, Torresen J (2012) Go ahead: a partial reconfiguration framework. In: Proceedings of the 2012 IEEE 20th international symposium on field-programmable custom computing machines, FCCM '12 IEEE Computer Society, Washington, DC, USA, pp 37–44
21. Corbetta S, Morandi M, Novati M, Santambrogio MD, Sciuto D, Spoletini P (2009) Internal and external bitstream relocation for partial dynamic reconfiguration. IEEE Trans Very Large Scale Integr Syst 17(11):1650–1654
22. Flynn A, Gordon-Ross A, George AD (2009) Bitstream relocation with local clock domains for partially reconfigurable fpgas. In: Proceedings of the conference on design, automation and test in Europe, DATE '09. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, pp 300–303
23. Duhem F, Muller F, Lorenzini P (2011) Methodology for designing partially reconfigurable systems using transaction-level modeling. In: conference on design and architectures for signal and image processing (DASIP), pp 316–322
24. Duhem F, Muller F, Lorenzini P (2012) Reconfiguration time overhead on field programmable gate arrays: reduction and cost model. IET Comput Digital Tech 6(2):105–113
25. Foucher C, Muller F, Giulieri A (2012) Fast integration of hardware accelerators for dynamically reconfigurable architecture. In: IEEE CAS 7th international workshop on reconfigurable communication-centric systems-on-chip (ReCoSoC 2012). IEEE, York, UK
26. Jozwik K, Tomiyama H, Honda S, Takada H (2010) A novel mechanism for effective hardware task preemption in dynamically reconfigurable systems. In: FPL'10, pp 352–355
27. Bilavarn S, Khan J, Belleudy C, Bhatti MK (2014) Effectiveness of power strategies for video applications: a practical study. J Real-Time Image Process. doi:10.1007/s11554-013-0394-6. URL http://link.springer.com/article/10.1007/s11554-013-0394-6
28. Muller F (2014) FoRTReSS Toolbox (flow for reconfigurable architecture in real-time systems). https://sites.google.com/site/fortresstoolbox/
29. The Eclipse Foundation (2013) Eclipse graphical modeling framework (GMF). http://www.eclipse.org/modeling/gmp/
30. JDOM Project (2012) JDOM. http://www.jdom.org/
31. x265 Project (2014) x265. http://x265.org/
32. Damak T, Werda I, Bilavarn S, Masmoudi N (2013) Fast prototyping h.264 deblocking filter using esl tools. Trans Syst Signals Devices 8(3):345–362
33. ITU-T (2005) ISO/IEC 14496–10, advanced video coding for generic audiovisual services, ITU-T Recommendation H.264, Version 4
34. D'huys TPKC, Momcilovic S, Pratas F, Sousa L (2014) Reconfigurable data flow engine for hevc motion estimation. In: IEEE international conference on image processing (ICIP)
35. Kalali E, Adibelli Y, Hamzaoglu I (2012) A high performance and low energy intra prediction hardware for high efficiency video coding. In: IEEE 22nd international conference on field programmable logic and applications (FPL 2012)
36. Tiago Dias NR, Sousa L (2014) Unified transform architecture for avc, avs, vc-1 and hevc high-performance codecs. In: EURASIP J Adv Signal Process 2014(1):108
37. ARDMAHN consortium (2013) ARDMAHN project. http://ARDMAHN.org/

J Real-Time Image Proc
DOI 10.1007/s11554-013-0394-6

ORIGINAL RESEARCH PAPER

Effectiveness of power strategies for video applications: a practical study

Sébastien Bilavarn · Jabran Khan · Cécile Belleudy · Muhammad Khurram Bhatti

Received: 5 July 2013 / Accepted: 23 December 2013

Abstract This study examinesthe practical effectiveness of power strategies for video applications. Based on real implementations of three power strategies using representative platforms and H.264 applications, we analyse platform and application level parameters affecting the operability and efficiency of power strategies. Results show that, in the same conditions, a strategy might offer highly variable results and sometimes increases energy, depending on the characteristics of the platform. Therefore, we report different measurement results which lead to useful guidelines for succesful power management and show the potential benefits of advanced power strategies over currently available approaches for demanding workloads like video applications.

Keywords Real-time · Embedded systems · Power management · H.264 · Dynamic voltage and frequency scaling (DVFS) · Dynamic power switching (DPS)

1 Introduction

In recent years, there has been a rapid and widespread growth of non traditional computing platforms such as wireless handheld computing devices. Multimedia services, video broadcasting and streaming represent heavy workloads for these systems that critically affect their battery autonomy. Ensuring the best possible use of the limited energy budget is thus essential for this type of application and requires the use of advanced dynamic energy aware processing.

Two main techniques are generally employed to let a computing system adapt its power states. The first one relies on powering down unused resources (dynamic power switching) and the second one, on reducing processor speed (dynamic voltage and frequency scaling). The decision of which power state to use and when is under the control of a dynamic policy (or strategy) which, therefore, determines the actual energy efficiency. Defining advanced power strategies has been an active topic of research for the last couple of decades; however, they also come with strong development cost and complexity that are probably responsible for their relative lack of practical implementation and adoption.

In this study, we investigate the actual efficacy of fully implemented dedicated power strategies, using representative up to date platforms and H.264-based video coding applications. DVFS and DPS energy gains are measured on a real development board (Platform Baseboard ARM1176JZF-S) and virtual SystemC cycle accurate MPSoC platforms (QEMU processor emulator including models of ARM11 and Cortex A9). Detailed analysis of results provides reliable numbers on expectable savings as well as useful conditions, guidelines and perspectives for the practical effectiveness of power strategies, especially for demanding workloads like video processing.

The outline of the paper is the following. First, we review existing efforts in the field of power management and introduce the context and objectives of this work. Then, we present detailed experimental results and analysis of three advanced power strategies applied over applications related to the H.264/AVC coding standard. In Sect. 5,

S. Bilavarn (✉) · C. Belleudy
LEAT, CNRS UMR7248, University of Nice Sophia Antipolis, Nice, France
e-mail: bilavarn@unice.fr

J. Khan
COMSATS Institute of Information Technology, Abbottabad, Pakistan

M. K. Bhatti
COMSATS Institute of Information Technology, CIIT, Lahore, Pakistan

Published online: 19 January 2014

we provide a global discussion of the results in terms of conditions for power management operability and effectiveness. Finally, we present a conclusion and perspectives from this work.

2 Power management challenges

2.1 Overview of general purpose strategies

When it comes to power management, DPS and DVFS are two popular techniques that are broadly employed. In early stages, power switching came first and was under the control of the BIOS Advanced Power Management (APM) developed by Intel and Microsoft [20]. ACPI [19] was later adopted to allow direct power management by the Operating System. ACPI is a hierarchical technique dividing the overall system components into Global G-states, System S-states, Processor P-states, Busses B-states, Links L-states and Devices D-states. In particular, the number of P-states is processor specific, where each state corresponds to a different frequency and power consumption level. In general, power states are defined regardless of the power management technology used, and are controlled by a software power policy. A typical example is OS-directed power management (OSPM) using ACPI, which defines a policy based on different users, application or environmental parameters. The policy manager selects sleep states when there is no workload on the processor, whereas P-states are employed through DVFS when the processor is active to reduce the overall power. In Linux for instance, the CPUFreq infrastructure is used to set a static or dynamic DVFS power policy for the system. In-kernel governors (i.e. strategies) are used and can change the CPU frequency based on different criteria. Each of five possible governors, Performance, Powersave, OnDemand, Conservative and Userspace, has its own unique behavior, purpose, and suitability in terms of workload.

In addition to ACPI or Linux, modern microprocessors come with their own power management hardware and software to handle power consumption. Examples of such processors include Intel SpeedStep [18], AMD Cool'-n'Quiet [1] and PowerNow [2], IBM EnergyScale [10] and ARM IEM [3]. In most cases, existing power management policies rely on the total workload of a system (which is more related to the hardware state), during a certain period to select the right power state. These policies present the advantage of being applicable in all cases (general purpose), but the drawback is a certain level of inefficiency because they lack of advanced and dynamic knowledge of applications. Besides these standards, a lot of academic research has also been carried out on the subject. Surveys have been proposed for example in [6] and [25] from the abundant literature. As some works pointed out recently the benefits of application awareness in power management compared to OS-level and hardware-level schemes [22], we focus in the following on dedicated strategies, especially those which can apply to video processing. These fall into two categories: strategies defined specifically for video applications and deadline (or real time) scheduling.

2.2 Dedicated strategies

2.2.1 Video specific strategies

Video applications are challenging because they represent typical workloads that bring CPUs close to their maximum level of power consumption. However, inherent video properties can be exploited to define efficient DVFS and DPS strategies. Indeed large variations of processing complexity are present in actual video standards that can be used to idle a processor or to lower clock frequencies when decoding less complex frames. This is used for example in [24] to regulate voltage/frequency for individual frames based on a prediction, halfway decoding the frame, of the remainder of the encoded frame. This prediction is based on the complexity ratio observed in the previous frame of the same type (I, P, PB) and results report up to 40 % energy gains. In [13], a DVFS scheme also uses the frame type and a more refined prediction of the frame decoding time to scale down voltage/frequency while meeting the predicted time. This technique provides up to 50 % energy gains for a MPEG decoder on a StrongARM-based platform. A DPS strategy is proposed in [21] for adaptive pipelined MPSoCs executing multimedia applications. This method is able to exploit different power states, based on the application execution history or predictions of the upcoming workload. 40 % energy savings are reported for a H.264 video encoder from cycle accurate simulations of a Tensilica processor.

These works demonstrate undeniable interests in considering application knowledge for critical workloads like video processing. However, it is worth noting the relative low number of works in the direction of dedicated strategies, especially considering multimedia and video services in the current context of multiprocessor system-on-chips and mobile computing.

2.2.2 Deadline scheduling

Energy-efficient deadline scheduling, also referred to as low power scheduling, is part of a more theoretical field of study on real-time scheduling that has attracted a lot of attention. These techniques also apply to video applications as frames are processed under typical constraints of 33 or 40 ms (for 30 and 25 fps). The general principle is to exploit variations of the actual execution time of jobs in

real-time applications to dynamically turn off or change the speed of one or several processors, while ensuring all jobs are complete by their deadlines. Yao et al.'s study [26] is historically the first theoretical study of energy efficient deadline scheduling. They consider a single processor with speed scaling, assuming continuous and infinite variation of processor speed. The method proposes an offline minimum energy schedule and two online heuristics called average rate (AVR) and optimal available (OA), with detailed theoretical models and proofs. An extension to manage temperature is proposed in [4] and more realistic models of bounded speed processors have been considered in [5, 11]. Approaches combining both speed scaling and sleep state energy-efficient deadline scheduling [16], or addressing multiprocessor scheduling without migration [15] have also been proposed.

These works have in common to define advanced theoretical and formal approaches which help to build mathematical proofs that can be used to address scheduling analysis for example. The counterparts for this formalism are important simplifying hardware assumptions, like ignoring state transition latencies or assuming continuous and infinite variation of processor speed. As processor frequencies are discrete and limited in practice, there can be no implementation and verification of energy gains. Another issue is related to the implementation complexity of real-time scheduling that adds to the feasibility and effectiveness issues in these methods.

2.3 Work context and objectives

For previous reasons, this study promotes a practical approach based on concrete experimentation of power strategies to quantify the actual energy gains that can be expected and examine the conditions of efficiency, applicability and variation in representative platforms. Three different types of power strategies based on DVFS and DPS are addressed in the particular context of video processing. These include one video-specific strategy and two deadline scheduling strategies implemented using a prototyping framework based on Linux userspace scheduling [12].

The platforms used for experiments combine a real Platform Baseboard ARM 1176JZF-S development board including built-in resources for power monitoring, and two QEMU-based virtual platforms including models of previous ARM1176JZF-S and for the Cortex A9-based ST-Ericsson Nova A9500 application processor. As QEMU platforms provide SystemC cycle accurate MPSoC simulations supporting power/energy management and estimation [14, 23], they were used profitably in the absence of equivalent real platforms at the start of this work. In the following, we investigate these strategies using both a functional H.264 decoder for the ARM11-based Platform Baseboard and a task model of an H.264 encoder on virtual platforms to be able to process QEMU-based simulations. To be representative, we considered strategies of different types, but falling under the same category of dedicated strategies. Following previous classification, we address first strategies that are specific to video processing (using one custom DVFS-based strategy for video decoding), and then two strategies for real-time scheduling (using a DVFS and a DPS deadline scheduling strategy called DSF and AsDPM), as they also apply considering typical 40 ms frame processing constraints.

3 Analysis of a video specific strategy

3.1 A DVFS strategy for video decoding

3.1.1 Strategy principle

Previous works on a H.264 decoder reported that the frame rate was sensitive to the motion properties in usual video sequences, with variations of about ±20 % [9]. The principle of the power strategy is thus to exploit these variations to smooth the frame rate using DVFS.

The processor might run slower when the frame rate is above the average frame rate to reduce power and energy. This adaptation is based on a frame rate constraint to satisfy (e.g. 9 fps) which defines distinct operating zones in terms of fps for each processor frequency. The thresholds delimiting these zones are determined by the following expression: $\text{thresh}_i = \text{adaptation\_const} \times f_{\text{nom}}/f_i$, where f_{nom} is the processor nominal frequency. For the ARM1176JZF-S, four thresholds (13.50, 10.05, 9.0 and 8.15 fps) can be derived from the four operating points of the processor (160, 215, 240 and 265 MHz) for an adaptation constraint of 9 fps for example. If the decoder frame rate is between 0.0 and 8.15 fps during 250 frames (to minimize frequency switching), the operating point is set to 265 MHz, between 8.15 and 9.0, it is set to 240 MHz, etc. In the following, we investigate the ability of this policy to save power using an H.264 video profile decoder.

3.1.2 Energy gains on PB ARM1176JZF-S

A summary of measurement results on a Platform Baseboard ARM1176JZF-S development board is reported in Table 1, showing the total energy, mean power, total decoding time, and time per frame for different adaptation constraints. Surprisingly, these results show an increase in energy utilization for adaptation constraints of 4, 8 and 9 fps. In each case, the required constraint is under the average performance (11.6 fps at nominal frequency of 240 MHz), so the CPU frequency actually drops but this does not provide energy savings.

Table 1 Energy gains of DVFS video decoding on ARM1176JZF-S

Adaptation constraint (fps)	Total energy (J)	Average power (mW)	Decoding time (s)	Average time/frame (ms)	fps	Energy gain (%)
Not used	52	311	168	86.4	11.6	0
4	64	262	244	126.1	7.9	−23
8	62	270	228	117.6	8.5	−19
9	56	290	196	101.4	9.9	−8
11	52	311	168	86.5	11.6	0

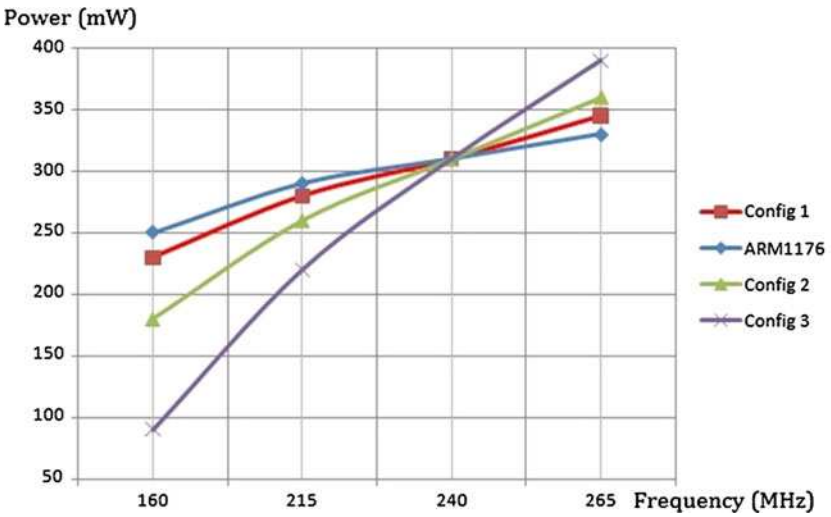


Fig. 1 QEMU operating point configurations

The reason is how downscaling frequency increases the execution time compared to power reduction. As $E = P \times T$, the decrease of power is not enough to compensate the increase of execution time on the Versatile PB1176JZF-S. This adverse effect is due to the characteristics of the operating points on this platform. For example, moving from 240 to 215 MHz raises execution time by 10.4 % for a decrease of 6.45 % in power (plot labelled ARM1176JZF-S in Fig. 1). Therefore, the corresponding product $P \times T$ remains greater after than before the decrease of frequency. In the next section, we further investigate this by experimenting the same strategy on a virtual platform (QEMU ARM1176) that can let us change the power levels of operating points.

3.1.3 Further investigations with QEMU ARM1176

Figure 1 shows the original characteristics of operating points for the ARM1176JZF-S processor. Three configurations labelled $config_1$, $config_2$ and $config_3$ are added to reflect different levels of load power for each processor frequency. The four configurations are set in a way to homogeneously increase the initial power gap between frequencies. For example switching from 240 to 215 MHz implies a reduction of 20, 30, 50 and 90 mW, respectively, for *ARM*1176, $config_1$, $config_2$, $config_3$.

The energy savings reported with these configurations on the H.264 decoder are shown in Fig. 2, for three adaptation constraints of 4, 6 and 8 fps. Since they are below the original nominal frame rate of 11.6 fps, the DVFS strategy actually decreases processor frequency in each case. However, the results indicate that only $config_2$ and $config_3$ succeed at saving energy because for *ARM*1176 and $config_1$, the differences of power consumption in consecutive operating points cannot compensate the increase of execution time.

3.1.4 Main results and analysis

These results raise three main points. First, actual platforms may not always provide energy gains when reducing dynamically the frequency due to an inefficiency of the operating points. This is the case for the Versatile Baseboard ARM1176JZF-S used for the experiments, and also on other platforms for various hardware reasons.

The Versatile Baseboard has a development chip that does not operate at real system speed due to prototyping and debugging constraints. The maximum frequency is

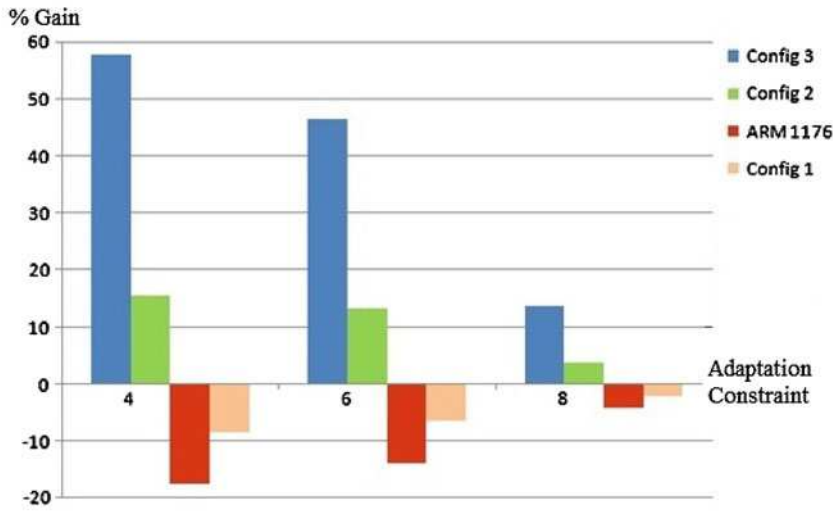


Fig. 2 Energy gains of DVFS video strategy

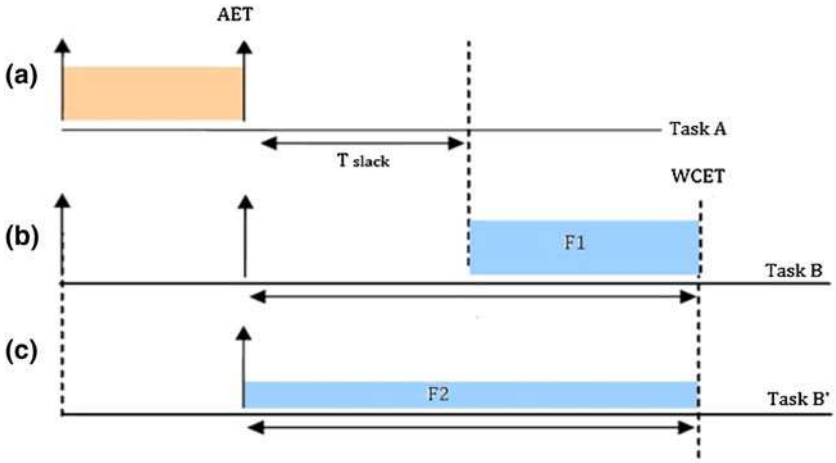


Fig. 3 DSF scheduling principle

limited to 265 MHz while a production device would be able to run at 800 MHz, and this affects the relevance of operating points. Secondly, the characteristics of operating points are key parameters in the efficiency of DVFS. In particular, the differences of power levels between consecutive frequencies have a determining impact on the amount of possible energy savings. Under previous platform conditions of efficiency, variations of energy gains due only to the operating point characteristics can reach several orders of magnitude (e.g. 3.6 times between $config_2$ and $config_3$).

Finally, the results also report more than 50 % possible energy gains, depending on the application driving parameter of the strategy (frame rate constraint). For typically high workloads such as video processing, existing workload-based approaches would make a CPU to switch at maximum frequency for the duration of the application, resulting in maximum power consumption. This points out the additional benefits of dedicated strategies over existing general purpose strategies for demanding applications.

4 Analysis of two deadline scheduling strategies

4.1 A DVFS deadline scheduling strategy: DSF

4.1.1 Scheduling principle

Deterministic stretch-to-fit (DSF) is a type of deadline scheduling algorithm intended to reduce power consumption using DVFS [7]. It exploits the fact that the actual execution time (*AET*) of application tasks is actually less or equal to the worst case execution time (*WCET*). The principle is illustrated in Fig. 3 where the early completion of a current *Task A* produces time slack *Tslack* to be used by the next priority ready *Task B*. As this allocation provides more time for the execution of *Task B*, the processor speed can be decreased as well as the associated power and energy.

To verify its practical efficiency, DSF was implemented with the afore-mentioned Linux userspace scheduling framework (Sect. 2.3). The application use case is a parallelised H.264 encoder supporting dual core execution,

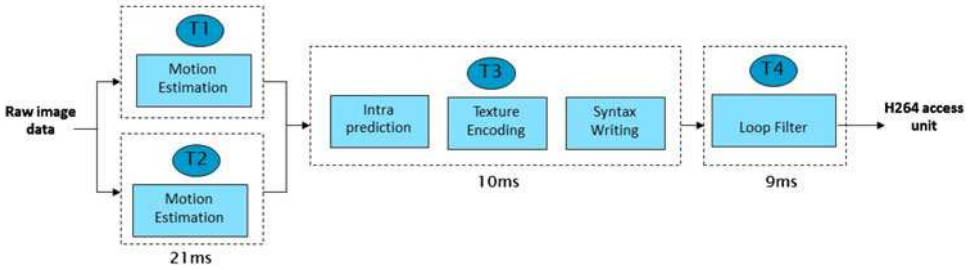


Fig. 4 H.264 encoder block diagram

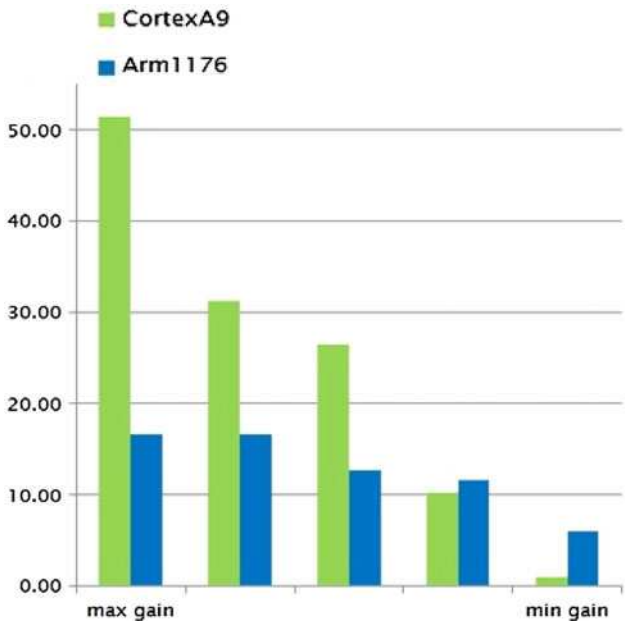


Fig. 5 DSF energy savings for H.264 encoder

corresponding block diagram of which is given in Fig. 4. In this implementation, the motion estimation engine which is the most processing component of the encoder is split into two parts for acceleration. A task model of the encoder is actually used, composed of the same tasks as that of the real H.264 encoder but doing simplified computing to ease virtual platform simulations. Timing values of the task model (*WCET*, *BCET*, deadline and period) are those of a real encoder profiled on a ST-Ericsson Nova A9500 processor (Dual Cortex A9). Thus, varying the slacks from *BCET* to *WCET* for the two motion estimation tasks T_1 and T_2 let us know the range of DSF achievable gains for ARM1176 and Cortex A9 platforms that are both in dual core configuration.

4.1.2 Energy gains on QEMU ARM1176 and QEMU Cortex A9

Figure 5 reports measurement results with energy savings ranging between 5.93 and 16.70 % on QEMU ARM 1176, and between 10.21 and 51.46 % on QEMU Cortex A9. Cortex A9 outperforms ARM1176 in efficiency by 1.8 times in average, despite the same application (slack) and measurement conditions. As frequency is set upon the slack of a previous task and the *WCET* of the next task, the power levels associated with frequencies account for this difference. Indeed, we can check in Table 2 that comparatively, operating points have quite different characteristics on both platforms. For example, when the Cortex A9 is downscaled from maximum (1 GHz) to minimum (300 MHz) frequency, the active (load) power goes 177 mW down. The corresponding power gap is 80 mW on the ARM1176 processor (from 265 to 160 MHz). As the gap in power levels between operating points are more important, more power can be saved when decreasing frequency with the Cortex A9 and net energy gains are improved.

Table 2 Idle versus load power for QEMU ARM1176 and QEMU Cortex A9

Platform	Freq. (MHz)	Idle P. (mW)	Load P. (mW)
ARM1176	160	223	250
	215	238	290
	240	245	310
	265	252	330
Cortex A9	300	38	143
	600	60	215
	1,000	90	320

Another factor impacting the efficiency of DSF is illustrated in Fig. 6. In these measurements, we have used a simple example (2 tasks, 1 CPU) to check how frequency switching latencies affect the time granularity of tasks. Timing values of tasks are set by a scaling factor of 10^{-0}, 10^{-1}, 10^{-2} or 10^{-3}. It is clear from the results that tasks in the order of milliseconds are rapidly loosing their effectiveness. This comes from the fact that, as they get closer to switching delays (typically around hundreds of microseconds), the energy cost of switching frequency becomes increasingly sensitive and alters greatly the efficiency of DSF.

4.1.3 Main results and analysis

These measurements have shown the ability of DSF to save energy for two platforms, with gains that are up to 18 % for

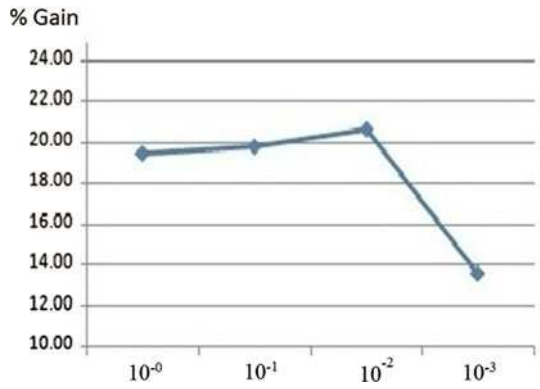


Fig. 6 Energy gain vs. time granularity of tasks (s)

ARM1176 and 52 % for Cortex A9. It emphasizes again the potential of advanced strategies to further reduce energy, whereas a general purpose (workload based) approach would lead to operate at maximum frequency and power for high workloads like video processing. DSF driving parameter is the application dynamic slack. Under the same conditions of slack and measurement, there is a notable factor of 1.6 in energy efficiency between ARM1176 and Cortex A9. Like for previous DVFS video strategy, the level of energy gains is firmly related to the platform characteristics, namely the differences in power levels between operating points. Finally, the execution time of application tasks must also exceed sensitively the latency of frequency switching delays in order to be applicable. In other words, not all applications are compatible with DVFS if this task requirement is not verified.

4.2 A DPS deadline scheduling strategy: AsDPM

4.2.1 Scheduling principle

Assertive dynamic power management is a strategy intended to be used for multiprocessor low-power scheduling [8]. The general principle is based on maximising the execution of application tasks on a minimum number of processors in order to set unused cores into low power sleeping states. The example of Fig. 7a shows the distribution of three tasks on two processors with segmented idle times under a standard EDF schedule. In Fig. 7b, the associated AsDPM schedule groups the execution of the tasks on processor π_1; therefore, substantial energy can be saved by putting processor π_2 into a sleep state for an extended duration. In the following, we examine the effectiveness of this scheduling strategy using the Linux userspace framework mentioned in Sect. 2.3

4.2.2 Energy savings on QEMU ARM1176 and QEMU Cortex A9 platforms

Based on its definition, AsDPM scheduling needs a platform with at least two processors to be applicable. The scheduler is implemented on QEMU ARM1176 and QEMU Cortex A9 in a dual core configuration each, and applied to the previous H.264 encoder example. At the beginning of an hyperperiod, the encoder requires two processors to execute parallel motion estimation tasks T_1 and T_2, while T_3 and T_4 are sequential and need only one processor. The second processor can thus be idled for a period of time that depends on the value of dynamic slacks produced by tasks T_1 and T_2. We have set six different values of *AET*s for T_1 and T_2 corresponding to the six solutions reported in Fig. 8 to fully characterize the range of energy gains. Varying the application slack this way also provides reliable comparison of energy gains on both platforms.

Figure 8 shows that AsDPM results in actual energy gains for both platforms, ranging from 24.05 to 46.73 % on ARM1176 and 15.32 to 42.72 % on Cortex A9. AsDPM is 1.4 times more efficient on ARM1176, whereas DSF was previously better on Cortex A9.

This difference comes again from the characteristics of operating points, but this time in terms of load and idle power levels. Measurements have been carried out at maximum frequency for both platforms, respectively, 265 MHz and 1 GHz for ARM1176 and Cortex A9. It is worth noting from Table 2 that idle power of the ARM1176 (252 mW) is important compared to load power (330 mW), while there is more difference for Cortex A9 (respectively 90 and 320 mW). Therefore, putting a core into a nearly zero Watt sleeping state reduces more power on ARM1176 than on Cortex A9. The important idle power

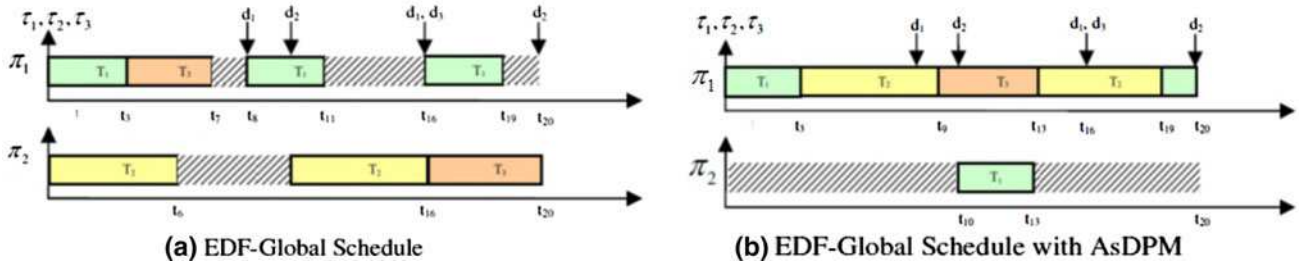


Fig. 7 AsDPM scheduling principle

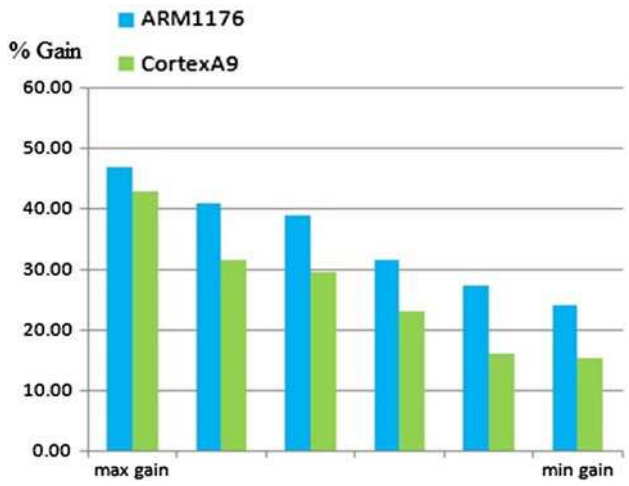


Fig. 8 AsDPM energy savings for H.264 encoder

on ARM1176 results again from the constraints of an early prototyping platform. However, these results show the ability of AsDPM to address systems with an important part of idle power consumption.

4.2.3 Main results and analysis

Practical effectiveness of AsDPM has been shown on the basis of a scheduler prototype operating on two platforms, with net energy gains up to 46 % for ARM1176 and 42 % for Cortex A9 depending on the variability of the application driving parameter (dynamic slack). The amount of energy gains is also determined by the idle power levels of the platform which are operating point dependent. AsDPM is especially suitable for systems with a sensitive share of idle power, which is highly related to standby leakage power and typical of deeply scaled nanometer technologies. Like for DVFS strategies, the latencies of entering/resuming from different power states should be negligible against the minimum best case execution time (*BCET*) of all application tasks. If this condition is not met, the overheads of switching sleep/active states will affect the energy efficiency of the strategy, and also probably lead to improper application execution. As these latencies are usually greater than for changing frequency, AsDPM and more generally DPS-based techniques might be less applicable than DVFS, especially to cope with typical 40ms frame processing constraints of video applications.

5 Effectiveness of power strategies

5.1 Platform level conditions

The majority of DVFS power strategies are based on the hypothesis that decreasing processor frequency results in saving energy. Although this fundamental assumption is verified on most platforms, Sect. 2.3 has shown that there are some cases where it is not due to the operating points. Indeed, their characteristics determine the actual saving ability and to a great extent the amount of energy gains. A determining factor in DVFS efficiency is the relative difference of load power levels compared with that of the corresponding frequencies. Large differences are a prerequisite for meaningful energy gains while too small differences might result in practically inefficient DVFS. DPS strategies are subject to similar platform conditions, a prominent criterion in this case is the power levels of idle states. DPS is more effective when operating on high values of idle power, which is typically the case in recent deep sub-micron process technologies with large static power consumption.

The applicability of power strategies also strongly depends on the transition costs between power states. DVFS policies rely implicitly on the assumption that latencies of changing frequency, usually estimated at around a few hundreds of microseconds, can be neglected, i.e., when the application can afford those waiting times. This does not apply for example if these latencies exceed a few milliseconds regarding typical 40 ms frame processing constraints of video applications. Latencies of DPS state transitions are often higher, due to complex shutdown, wakeup and context saving schemes. This might further restrict the use of DPS compared to DVFS, especially in video applications. Therefore, latencies of state transitions should be analysed thoroughly against the time granularity of tasks before developing DVFS or DPS strategies.

5.2 General purpose vs. domain specific strategies

General purpose workload-based approaches (e.g. Linux OnDemand) would typically set the CPU frequency level to its maximum for H.264 coding/decoding, driving an increased power demand. Advanced DVFS and DPS strategies exploit closer application awareness and can, therefore, perform better. In results of Sects. 2.3, 3.1.4 and 4.2, they show an ability to actually reduce the energy use up to 50 %. The effectiveness of these application driven power strategies relies on their capacity to exploit a form of execution variability. They are based on a driving parameter (e.g. frame rate, dynamic slack) which variations at runtime determine the actual amount of energy gains. A reasonable quantity of energy reduction that can be expected is situated between zero and a maximum of 50 %, based on data-dependent execution. This effectiveness is also subject to the afore-mentioned conditions on time definition of tasks that must be large enough to be able to neglect the cost of processor power state transitions.

5.3 DVFS vs. DPS

A question that often arises is which of DVFS or DPS is more energy efficient. We can provide here a reliable comparison of two related strategies from the results of DSF and AsDPM experimentations on the common H.264 encoder example.

Energy gains have been reported previously in Figs. 5 and 8, respectively, for DSF and AsDPM on both ARM1176 and Cortex A9. They show that AsDPM outperforms DSF energy gains in all cases. However, they also indicate that the difference in efficiency might vary greatly from a platform to another. As stated previously, the influencing factors are the power level gaps of operating points for DVFS, and the idle versus load power levels for DPS strategies. Though it is likely that powering down unused processors (DPS) has potential for higher gains than reducing their speed (DVFS), generalization has to be considered with caution. Indeed the actual characteristics of the platform (operating points, power levels and state transition latencies) can reverse the situation in some cases, or prevent proper application of a strategy. DPS should be more efficient, but more limited in terms of applicability, especially to cope with frame rate-processing constraints of video applications.

6 Conclusion and perspectives

This paper described an analysis of power strategies for video applications based on actual implementations and fine measurements. Three types of power strategies have been investigated with H.264 applications on representative ARM-based platforms, in a way to deal realistically with power strategies and study the conditions influencing high energy gains. The outcome of this experimental study is a better understanding of practical constraints affecting power management efficiency and opportunities for improvement. Actual energy savings have been shown to be strongly dependent on platform conditions that are the levels and transition delays associated with power states for both DVFS and DPS-based strategies. This underlines the importance of a first and foremost analysis of hardware characteristics to define strategies that are successful in practice. If these conditions are met, the efficiency of strategies ultimately depends on their scope and relevance too. The ability of more dedicated policies to exploit fine application knowledge gives room for further energy gains, up to 50 % according to our measures for video applications where general purpose strategies would be unsuited and inefficient.

These results suggest that higher levels of energy gains can be reached using a joint contribution of strategies. As an important concern in the improvement of energy cost is to make an optimal use of power management features, power management can build on finer workload awareness using for example general purpose policies for most applications and dedicated (DVFS and/or DPS based) strategies for power sensitive workloads. We have already started to investigate a cooperation of DSF and AsDPM strategies in this context [8].

While very promising, these perspectives let us expect greater complexity. The near-term prospects of integration technology (below 22 nm FinFET, 3D stacking) and the associated power constraints (power density, thermal management, heterogeneity) will also add significantly to this complexity. In the face of these perspectives, it is quite likely that existing power management solutions, already very complex, reach their limit. More advanced dynamic energy-aware supervision will probably have to be redefined; we have started to investigate this addressing runtime analysis for multiobjective optimization.

Acknowledgements This work was carried out under the COMCAS project (CA501), a project labelled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

References

1. AMD.: AMD Cool'n'Quiet Technology. http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx (2002)
2. AMD.: AMD PowerNow! Technology, Nov (2000)
3. ARM.: Intelligent Energy Manager (IEM) Hardware Control System in the ARM1176JZF-S Development Chip, Nov (2006)
4. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**, 3 (2007)
5. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. In: Proceedings of International Colloquium on Automata, Languages and Programming (ICALP), pp. 409–420 (2008)
6. Benini, L., Bogliolo, A., De Micheli, G.: A survey of design techniques for system-level dynamic power management. IEEE Trans. Very Large Scale Integr. VLSI Syst. **8**(3), 299–316 (2000)
7. Bhatti, K., Belleudy, C., Auguin, M.: An inter-task real time DVFS scheme for multiprocessor embedded systems. In: Proceedings of International Conference on Design and Architectures for Signal and Image Processing (DASIP'10), Edinburgh, Oct (2010)
8. Bhatti, K., Belleudy, C., Auguin, M.: Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques. Real-Time Syst. **47**(2), 143–162 (2011)
9. Bilavarn, S., Belleudy, C., Auguin, M., Dupont, T., Fouilliart, A.: Embedded multicore implementation of a H.264 decoder with power management considerations. In: Proceedings of the 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'08), vol. 50, pp. 124–130. 1 Sep 2008
10. Broyles, M., Francois, C., Geissler, A., Grout, G., Hollinger, M., Rosedahl, T., Silva, G.J., Vanderwiel, M., Van Heuklon, J.,

Veale, B.: IBM energyscale for POWER7 processor-based systems. IBM J. Res. Dev. **55**(3), 220–232 (2011)
11. Chan, H.L., Chan, W.T., Lam T.W., Lee, L.K., Mak, K.S., Wong, P.W.H.: Optimizing throughput and energy in online deadline scheduling. ACM Trans. Algorithms **6**(1), 10 (2009)
12. Chéour, R., Bilavarn, S., Abid, M.: Exploitation of EDF scheduling in wireless sensor networks. Int. J. Meas. Technol. Instrum. Eng. **1**(2), 14–27 (2011)
13. Choi, K., Dantu, K., Cheng, W.C., Pedram, M.: Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD'02), pp. 732–737 (2002)
14. Gligor, M., Pétrot, F.: Handling dynamic frequency changes in statically scheduled cycle-accurate simulation. In: Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASPDAC'11) (2011)
15. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 11–18 (2010)
16. Han, X., Lam, T.W., Lee, L.K., Isaac K.K.T, Prudence W.H.W.: Deadline scheduling and power management for speed bounded processors. J. Theor. Comput. Sci. **411**, 3587–3600 (2010)
17. Intel Corporation.: Enhanced Intel SpeedStep technology for the Intel Pentium M processor, Mar (2004)
18. Intel.: Enhanced Intel SpeedStep technology for the Intel Pentium M processor, Mar (2004)
19. Intel., Microsoft., Toshiba.: Advanced power management (APM) : BIOS interface specification, revision 1.0, Jan (1992)
20. Intel., Microsoft.: Advanced configuration and power interface specification, revision 1.0, Dec (1996)
21. Javaid, H., Shafique, M., Henkel, J., Parameswaran, S.: System-level application-aware dynamic power management in adaptive pipelined MPSoCs for multimedia. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD'11), 2011, San Jose, pp. 616–623 (2011)
22. Liu, X., Shenoy, P., Corner, M.: Chameleon: application level power management with performance isolation. IEEE Trans. Mob. Comput. **7**(8), 995-1010 (2008)
23. Pétrot, F., Fournel, N., Gerin, P., Gligor, M., Hamayun, M.-M., Shen, H.: On MPSoC software execution at the transaction level. IEEE Des. Test **28**, 3 (2011)
24. Pouwelse, J., Langendoen, K., Lagendijk, I., Sips, H.: Power-aware video decoding. In: 22nd Picture Coding Symposium, Seoul, (2001)
25. Singh, P., Chinta, V.: Survey report on dynamic power management. In: Survey report of University of Illinois, Chicago (ECE Department), Chicago, (2008)
26. Yao, F., Demers, A., Shenker. S.: A scheduling model for reduced CPU energy. In: Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS), pp. 374–382 (1995)

Sébastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then, he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a 3 years post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an associate professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis, CNRS. His research interests are in design, exploration and optimisation from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of French, European and international collaborative research projects.

Jabran Khan is an Assistant Professor at the Department of Electronics Engineering, COMSATS Institute of Information Technology Abbottabad. Before that, he was a Ph.D. student in Electronics and Computer Engineering at the Laboratory of Electronics, Antennas and Telecommunications (LEAT) at University of Nice Sophia Antipolis (CNRS) in France. His research interest is in power management for multi-processor computing architectures.

Cécile Belleudy is an associate professor at University of Nice-Sophia Antipolis. She currently leads the MCSOC team (Modelisation and system design of Communicating Objects) at LEAT laboratory and the Master Degree in Electronics, Systems and Telecommunications at University of Nice Sophia Antipolis. Her research interests are in the general field of System-on-Chip design with a specific interest in power optimisation, including power management, low power and real-time scheduling, DVFS, DPM techniques and applications to multiprocessor architectures, multi-bank memories, and operating systems.

Muhammad Khurram Bhatti is currently an assistant professor at the COMSATS Institute of Information Technology, Lahore, Pakistan. Before joining COMSATS, he held the position of ATER (Attaché Temporaire d' Enseignement et de Recherche) at the University of Nice-Sophia Antipolis, France, for 1 year (2010–2011). He has completed his Ph.D. in Real-time Embedded Systems from the University of Nice-Sophia Antipolis, France, in April 2011. He received his Masters Degree in Embedded Systems from University of Nice-Sophia Antipolis, France, in 2007 and Bachelors degree in Industrial Electronics Engineering from NED University of Engineering and Technology, Pakistan, in 2003. Since 2007, he has been doing research and development in real-time embedded systems, energy-aware computing, and QoS-supported operating systems. He is also a visiting researcher at LEAT research laboratory, University of Nice-Sophia Antipolis, Nice, France. His areas of research interest include embedded systems, energy-aware computing, real-time systems, multiprocessor scheduling, scheduling theory, uniform multiprocessors.

An Energy-Aware Scheduler for Dynamically Reconfigurable Multi-Core Systems

Robin Bonamy, Sébastien Bilavarn, Fabrice Muller
LEAT, University of Nice Sophia Antipolis, CNRS, France
robin.bonamy@unice.fr, sebastien.bilavarn@unice.fr, fabrice.muller@unice.fr

***Abstract*—This paper describes an energy-aware scheduling approach intended for use in heterogeneous multiprocessors supporting hardware acceleration with Dynamic and Partial Reconfiguration. Scheduler decisions rely on pragmatic power and energy models to map the load across cores and reconfigurable regions with regards to the actual power costs. Results on a multithreaded H.264/AVC profile decoder with three possible hardware functions on a Xilinx Zynq based platform report energy gains up to 44.1% over full software execution and 49.6% over static hardware / software execution, while ensuring real-time decoding requirement.**

I. Introduction

The use of heterogeneous multiprocessor System-on-Chips has increased because of their potential to address energy efficiency, power and heat density problems. Despite the promises, the increasing level of heterogeneity greatly affects the design and mapping complexity in upcoming systems and applications. Indeed it is no longer simply a question of mapping efficiently concurrent processes to the best cores, but also to consider dynamic aspects such as power management or hardware acceleration and their impact on energy efficiency.

Hardware acceleration is a relatively well-known player in the energy performance equation which is regaining attention today with the advent of Dynamic and Partial Reconfiguration (DPR). Partial reconfiguration is a technique related to FPGAs that can be used to extend their inherent flexibility: it allows to reprogram specific regions with new functionality while other regions continue to run. Drastic reduction of hardware resource utilization results in less static power thus significant better energy efficiency, adding to the inherent benefits of dedicated hardware. However, a variety of parameters such as FPGA partitioning, accelerator parallelism or software execution strongly affect the actual processing efficiency, and other techniques such as blanking or DPR based clock gating can also be used to further decrease power. As a result, the quantity and scope of decisions in a dynamically reconfigurable multi-core system highly complexifies the scheduler's job. It is however critical to provide good support at this level since bad decisions can affect energy efficiency to the point of total ineffectiveness. We address in the following the definition and evaluation of an energy-aware scheduling and mapping approach for multi-processor systems supporting hardware acceleration with DPR, and report representative application results and achievements that are denoting promising prospects in this field.

978-1-4673-7942-7/15/$31.00

The outline of the paper is the following. We first review existing works in the field of heterogeneous multi-core scheduling, pointing out relative novelty in the use of DPR for that matter. In section 3, we introduce pragmatic power and energy models underlying the proposed scheduling procedure. We then describe the decision schemes used in the energy-aware scheduling and mapping process. Detailed results on a multithreaded H.264/AVC decoder on a Xilinx Zynq based platform are analyzed and discussed. Finally, main conclusions from these results are presented and future directions for extension and exploitation are proposed.

II. Overview of heterogeneous multiprocessor strategies

With the rise of multi-core heterogeneous architectures, there has been many works addressing efficient scheduling application workloads with multiple cores, possibly of different types. Early investigations focused primarily on improving load balancing to achieve better performance (throughput, instructions-per-cycle, etc.). Recently the challenge of heterogeneous thread scheduling and global power management switched on delivering higher power-performance levels (performance per Watt). Many works explored the energy efficiency benefits of heterogeneity, that are more extensively discussed for instance in [1]. There is a broad consensus on the fact that exploiting heterogeneity is essential for a better use of energy, the necessary counterpart being that it greatly complexifies the scheduler.

Of the work addressing this problem, [2] proposed a scheduler for a system of processors based on execution prediction to map future processing needs to the most suited processor. Their method applies to single-ISA heterogeneity supporting differing voltages and frequencies. [3] extended a symbiotic scheduling heuristic [4], originally developed to enhance throughput and lower response time, for chip multiprocessors with simultaneous multithreading cores. They report up to 7.4% savings in energy, 10.3% savings in energy-delay product, and 35% savings in power. [5] is another contribution considering both scheduling and power management to address process variations in CMPs. They conducted a design exploration, proposed a number of schedulers to satisfy different objectives, and developed a linear programming solution for power management. The authors in [6] examined the scalability problem for manycores, comparing basic scheduling heuristics and proposed scheduling and power management algorithms for heterogeneous systems scaling up to 256 cores. A recent study [7] addressed a study including the ARM big.LITTLE architecture, associating an energy

efficient processor cluster (Cortex-A7) with a higher performance processor (Cortex-A15). They report that different class of resource allocation heuristics (race-to-idle vs. never-idle) have very different results on different platforms, indicating that the efficiency of a strategy greatly depends on platform characteristics and can go as far as getting inefficient in some cases.

Aside from the question of core specialization, heterogeneity also extends to the aspects arising at run-time due for example to power management [8]. The dynamic use of different power states (P-states, sleep states) available for each core matches the problem of low power scheduling which had a long history of research over the last 20 years. Surveys have been described for example in [9] and [10] from the abundant literature. Our own previous experimental studies in this field [11] came to similar conclusions as [7] and [12] concerning the importance of platform characteristics and application knowledge on the efficiency of a strategy. Likewise, results indicate that custom strategies, i.e. more specialized schedulers dedicated to an application or application domain, can reach significant levels of energy gains (in the 5% to 50% range for video processing applications on representative platforms) compared to existing OS and platform-based strategies.

From this large literature, existing works often consider a specific perspective on the heterogeneous scheduling problem: a type of architecture, one precise objective (manycore scaling, process variability), limited heterogeneity (similar cores, DVFS), etc. In addition, a type of heterogeneity remains uncovered regarding the recent advancement of graphic and reconfigurable processing units: the possibility of hardware (or accelerated) execution of tasks. [1] is thus extending heterogeneity to hardware acceleration enabled with the progresses of Dynamic and Partial Reconfiguration (DPR) and High-Level Synthesis (HLS). Hardware acceleration delivers orders of magnitude performance and energy efficiency compared to software execution, and the flexibility introduced with DPR can improve these benefits. The underlying heterogeneous mapping and scheduling problem have started lately to be re-investigated and this paper extends the conclusions of [13] on the definition of low power scheduling policies able to support efficient execution of dynamic hardware and software tasks.

III. Power modeling

The remainder of this paper addresses a scheduling method capable of taking relevant run-time decisions to reduce the energy use while satisfying performance constraints. As such it requires that the power impact of resource allocation decisions can be realistically predicted. This section describes the underlying formal characterization, which results in great part from previous work reported in [13], and is divided in two main components: a platform and an application model.

A. Platform model

The platform is described by enumerating the execution units (EU) available and providing information on the corresponding power characteristics. To cope with the target reconfigurable multi-core platform, two main execution units are considered : a) processor cores and b) reconfigurable regions (RR).

The power model used for CPU cores is based on the type of core and is defined by the related static power $P^{static}_{core_j}$, the idle power $P^{idle}_{core_j}$ and the run power $P^{run}_{core_j}$, where j is the id of the execution unit.

As a RR needs to be configured before a task can run, which takes time and power that has also to be assessed, the reconfiguration controller can be modeled by its power P^{reconf}. In turn, the reconfiguration time depends on the area of the RR to configure ($T^{reconf}_{RR_j}$).

The power model for the RRs is defined by the static power consumption of the resources (e.g. slices, BRAM, clock tree, etc) in the given RR ($P^{static}_{RR_j}$). The related idle and run power depend on the actual task to be configured on this region. So the corresponding idle and run power are not defined in the platform model, instead they are characterized in the following application model.

B. Application model

An application is characterized by a set of tasks ($\mathcal{G}$) with their data and execution dependencies modeled by a task flow graph. Each task can have one or several implementations available, at least every task has a software implementation. An implementation is a description of how the task is being executed on a software or hardware EU. The model of an implementation reflects the task id (i), the EU id (j), execution time ($T_{i,j}$), and idle / run power consumption for hardware execution (P^{idle}_{i,RR_j}, P^{run}_{i,RR_j}). In case of software execution, the task idle / run power numbers are derived from the core idle and run power described in the platform model.

IV. Scheduler specification

The scheduling procedure involves two steps: a waiting task list is first determined and sorted according to a specific strategy (e.g. Earliest Deadline First), then a task implementation is mapped on a given execution unit according to an energy efficiency criteria. In the following, this process is further referred to as Energy Aware Heterogeneous Scheduler (EAHS).

A. Task Scheduling

Most common task scheduling strategies are a) FIFO (First In First Out) the first task arriving in the waiting state will be the first scheduled, b) EDF (Earliest Deadline First) which consists of sorting tasks to first schedule the one that must finish first and finally c) priority sort, a priority is assigned to each task and the highest priority task is executed first.

However these strict definitions may be alleviated in case of heterogeneous scheduling. Considering a scenario where the top task in the waiting list is blocked, waiting for matching execution unit to be free, then all other tasks in the waiting list are blocked despite the fact that they may be run on other free EUs. This will increase the application delay and execution units under-utilization. Thus we propose a *soft* scheduling approach which consists of using one of the previously mentioned strategies, but letting the possibility to bypass a task blocked because a corresponding mapping is not possible for the moment (e.g. either the reconfiguration controller is busy or compatible execution unit(s) not free).

B. Task Mapping

When managing task allocation on the architecture resources, the choice of an implementation has a major impact on reducing energy consumption. A cost function is thus defined to help identifying the most energy efficient mapping from the different possibilities. This cost is computed each time a task has to be mapped, for all possible implementations, which is based on energy and execution time estimations of the implementation being processed.

Energy estimation is given in equation (1):

$$\forall j \ ; \ E_j^{cost} = \left\{ \begin{array}{ll} E_{i,j}^{run} & \text{when } \rho_{i,j} = 0 \\ E_{i,j}^{run} + E_j^{conf} & \text{when } \rho_{i,j} = 1. \end{array} \right\} \quad (1)$$

where

$$\begin{aligned} E_{i,j}^{run} &= P_{i,j}^{run} \times T_{i,j} \\ E_j^{conf} &= T_{RR_j}^{reconf} \times P^{reconf} \end{aligned}$$

and $\rho_{i,j}$ represents the need to perform a reconfiguration. For instance if the same implementation is already configured, $\rho_{i,j} = 0$ means that RR_j can be used directly to run task i, otherwise $\rho_{i,j} = 1$ and a reconfiguration is required before execution. For all non reconfigurable EUs: $\rho_{i,j} = 0$.

To execute a new task, the scheduler has to check first that the execution unit EU_j currently evaluated is free. If not, the task can be implemented later and this delay must be considered for decision. Task execution time is thus estimated in equation (2):

$$\forall j \ ; \ T_j^{cost} = T_{i,j} + T_{RR_j}^{reconf} \times \rho_{i,j} + T_j^{busy} \quad (2)$$

where T_j^{busy} represents the time during which the EU_j is busy, running another task i'. It is derived from the execution time $T_{i',j}$, the begin time of the task currently running, and the current scheduling time.

The final cost for EU_j is computed in equation (3), where α is a user parameter (real value between 0 and 1) to promote performance (α close to 0) or energy efficiency scheduling (α close to 1).

$$\begin{aligned} &\forall j \ ; \\ &Cost_j \quad = \alpha \ \frac{E_j^{cost}}{max(E^{cost})} + (1-\alpha) \ \frac{T_j^{cost}}{max(T^{cost})} \end{aligned} \quad (3)$$

The cost function is evaluated for all available implementations and execution units for the current top task in the waiting list. The lowest $Cost_j$ value is selected and the task is implemented on EU_j. However, if EU_j is busy the task is kept waiting and will be implemented later.

V. Application Study

The application considered in this case study is a H.264/AVC profile video decoder. The input specification code used is a version derived from the ITU-T reference code [16] to better cope with hardware design constraints. From the code profiling, five major functions are highlighted and define the task set $\mathcal{G}$ as

$$\begin{aligned} \mathcal{G} \quad = \quad & \{Exp\_Golomb, \ MB\_Header, \ Inv\_CAVLC, \\ & Inv\_QTr, \ Inv\_Pred, \ DB\_Filter\}. \end{aligned} \quad (4)$$

An ESL design methodology [15] is used to provide real implementations for the possible hardware functions. The deblocking filter (DB_Filter), inverse CAVLC (Inv_CAVLC), and inverse quantization and transform block (Inv_QTr) contribute together to 76% of the global execution time on a single CPU core. They represent the three functionalities of the decoder that are generated from high level synthesis for hardware execution and can be either software or hardware executed.

This video decoder supports the possibility of slice decomposition of frames as defined in the H.264/AVC standard. A slice represents an independent zone of a frame, therefore decoding one slice (of a frame) is independent from another (slice of the same frame). This way, the decoder can process different slices of a frame in parallel and this application is an interesting case study for multi-core scheduling. We have thus considered a decomposition of the image where four streams process four slices of a same frame.

A. Modeling

The platform used in this case study is a Xilinx ZC702 [17] which is composed of a dual core ARM Cortex-A9 and a Xilinx Artix-7 equivalent FPGA. This development board allows power measurement for both FPGA and CPU core power supply. Thus measurement series are carried out under the following default setup conditions: 1V core voltage, 22°C room temperature, 667MHz ARM core frequency and 100MHz FPGA. The power values exposed in table I are setup from direct measurements on the ZC702 evaluation board.

TABLE I. Model parameters for a Zynq ZC702 platform.

Model	Value	Model	Value
P_{core}^{static}	89.82 mW	P^{reconf}	72 mW
P_{core}^{idle}	55.83 mW	$T_{RR_1}^{reconf}$	1.36 ms
P_{core}^{run}	119.4 mW	$T_{RR_2}^{reconf}$	1.36 ms

The task graph corresponding to the H.264 use case is shown in Figure 1. In this application configuration, reconfigurable regions defined for DPR execution are sized to host all hardware tasks and maximize the use of parallelism, with the methodology described in [18]. The two resulting RRs (RR_1 and RR_2) are then implemented and characterized in terms of

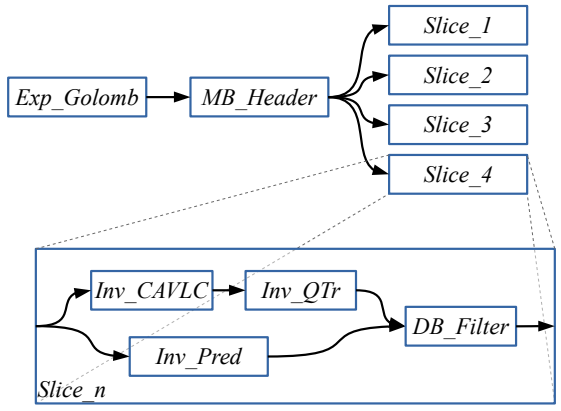


Fig. 1. H.264/AVC profile decoder task graph.

power and execution time on the ZC702 platform, leading to the task model parameters reported in table II.

TABLE II. MODEL PARAMETERS FOR H.264/AVC PROFILE DECODER ON A XILINX ZYNQ PLATFORM.

Task i	EU j	$P^{run}_{i,j}$ (mW)	$P^{idle}_{i,j}$ (mW)	$T_{i,j}$ (ms)
Exp_Golomb	$core_1$ $core_2$	119.4	-	6
MB_Header	$core_1$ $core_2$	119.4	-	5.9
Inv_CAVLC	$core_1$ $core_2$	119.4	-	6.6
	RR_1 RR_2	33.5	55.1	4.5
Inv_QTr	$core_1$ $core_2$	119.4	-	3.1
	RR_1 RR_2	5.8	34.2	1.5
Inv_Pred	$core_1$ $core_2$	119.4	-	3.2
DB_Filter	$core_1$ $core_2$	119.4	-	10.5
	RR_1 RR_2	6.4	44.3	0.9

B. Scheduling Results

The proposed scheduler is compared against four common scheduling strategies. User parameter α is set to 0.8 in a way to promote energy efficiency and to introduce a few performance concerns. Analysis on the H.264 decoder is carried out using a SystemC/TLM simulator [18] and results are reported in Table III and Figure 2. FPGA area is expressed in terms of Xilinx slice resources along with execution time and energy consumed to decode one frame. An energy-delay metric is also computed to provide additional measure and comparison of the energy performance trade-off [14]. Two solutions are highlighted for an EDF full software scheduling execution. The first one is tailored for the proposed platform (two cores) but can not achieve real-time video decoding (58.7ms per frame). As the application hyperperiod should be less than 40ms, the platform model is extended for a total of four ARM Cortex-A9 cores.

Another reference result is the scheduling profile obtained using two cores and static hardware accelerators (EDF Static). Indeed for this application configuration and task parallelism, a very large reconfigurable area would be required (39536 slices), which can be hard to implement on a real FPGA. Therefore another solution is introduced (EDF SmartStatic) which implements only one static accelerator for each possible hardware function (DB_Filter, Inv_CAVLC, Inv_QTr) to sequentially map different task invocation to the unique corresponding accelerator.

Software execution requires a large amount of energy (35mJ) while static implementations benefit from dedicated hardware efficiency with 21.1mJ (EDF Static) and 19.5mJ (EDF SmartStatic). However EDF SmartStatic is slower than EDF Static (35.4ms vs. 19.3ms) because less accelerator instances limit the exploitation of task parallelism. The outcome is visible in the energy-delay product: EDF Static obtains the best score (407) followed by EDF SmartStatic (690) and EDF Software (1350).

As for the DPR solution (two cores, two RRs), EAHS has a score which is close to EDF Static (478). In this case, energy consumption dropped from 21.1mJ to 16mJ compared to EDF Static, but execution time increased due to reconfigurations and fewer hardware resources. In addition, it is worth noting that EAHS provides a solution which consumes 3.4mJ less and is 5.6ms faster than EDF SmartStatic thanks to a better use of reconfigurable resources.

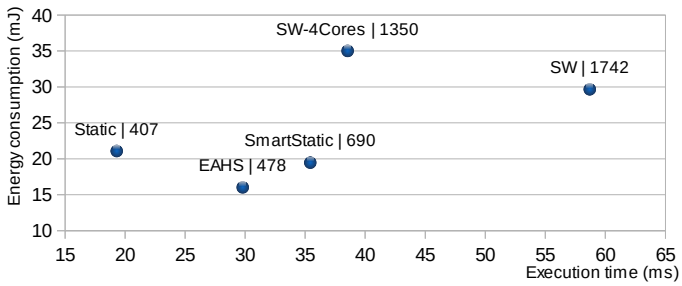


Fig. 2. Energy consumption versus time for different scheduling solutions along with energy-delay product.

Although these results are indicative of energy efficiency, they are not completely rigorous regarding the effective execution constraint which is to process frames periodically, every 40ms typically. In this case, the objective is not so much to minimize the performance energy product but to meet execution deadlines at the lowest energy cost. Therefore, previous energy results are adjusted in Table IV for an execution constraint of 40ms.

The result is that EDF Static consumes more energy than the four other solutions, close to 40mJ. It is due to the large static and idle power consumption arising from the amount of hardware accelerators when they are not in use. This solution which had the best energy-delay previously is in fact the worst if we consider the effect of idle power consumption. EAHS takes advantage of less reconfigurable resources in this case and provides the best energy efficiency with 20.06mJ, which is about 50% better than EDF Static and 6.8% more efficient than EDF SmartStatic.

The corresponding EAHS scheduling profile is presented in Figure 3, showing the usage of each execution unit during time for two 40ms periods. As defined in the platform model, two Cortex-A9 cores are used together with two reconfigurable regions. Reconfigurations are clearly visible (in pink) between different executions of tasks on the two RRs. All four invocations of each hardware accelerator are successively configured. The first four invocations of Inv_CAVLC are implemented using both RRs, so that they can run in parallel by pair. Then one RR is used for four sequential invocations of Inv_QTr and the other is configured to run four sequential invocations of DB_Filter. We see clearly in this application example the ability of the scheduler to maximize the use of hardware execution (more energy efficient), to take advantage of parallel execution, and to minimize the number of reconfigurations (having non-negligible energy overheads), which are essential conditions ensuring DPR energy efficiency in practical applications.

VI. CONCLUSION

This paper described a hardware / software scheduling strategy for energy efficient application execution on hetero-

TABLE III. COMPARISON OF DIFFERENT SCHEDULER RESULTS ON H.264 VIDEO APPLICATION.

Scheduler	N# Cores	FPGA area (eq. slices)	Execution time (ms)	Energy consumption (mJ)	energy-delay product
EDF Software	2	0	**58.7**	29.67	1742
EDF Software	**4**	0	38.54	35.02	1350
EDF Static	2	**39536**	19.31	21.09	407
EDF SmartStatic	2	6372	35.4	19.47	690
EAHS	2	6600	29.8	16.04	478

TABLE IV. COMPARISON OF DIFFERENT SCHEDULER ENERGY CONSUMPTION ON A 40MS PERIOD H.264/AVC PROFILE DECODER.

40ms period energy (mJ)	Energy Gain	EDF SW 4Cores	EDF Static	EDF SmartStatic	EAHS
35.87	EDF SW 4Cores	-	9.9%	-66.7%	-78.8%
39.81	EDF Static	-11.0%	-	-85.0%	-98.5%
21.52	EDF SmartStatic	40.0%	45.9%	-	-7.3%
20.06	EAHS	**44.1%**	**49.6%**	**6.8%**	-

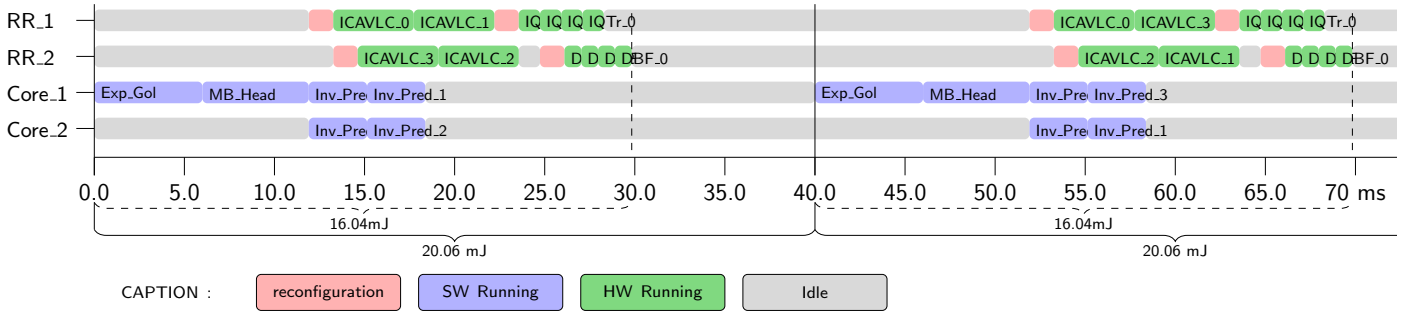


Fig. 3. EAHS scheduling and mapping results on the H.264/AVC profile decoder.

geneous multi-core systems with dynamically reconfigurable accelerators. Results show net improvements in energy savings both against full software execution (44.1%) and static hardware / software execution (49.6%). The global study shows therefore that a great potential lies in the exploitation of DPR assuming scheduling is based on reliable decision support able to clearly identify a low energy solution among the large overall mapping space. It is also likely that the acceleration potential of applications gives room for improving these results as more accelerators would fit in less space. Future works will thus address methods for exploring this potential. Interesting improvements can also be foreseen from extending the proposed EAHS strategy with DVFS in the first place, as a standard multiprocessor technique to reduce power, but also blanking to further exploit DPR in accelerator static power limitation.

ACKNOWLEDGMENTS

This work is carried out under the BENEFIC project (CA505), a project labelled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

REFERENCES

[1] C. Leech, T.J. Kazmierski, Energy Efficient Multi-Core Processing. ELECTRONICS, 18, (1), pp. 3-10, jun 2014. (DOI:10.7251/ELS1418003L).

[2] S. Ghiasi, T. Keller, F. Rawson, Scheduling for Heterogeneous Processors in Server Systems, Proceedings of Computing Frontiers, 2005.

[3] M. DeVuyst, R. Kumar, D.M. Tullsen, Exploiting Unbalanced Thread Scheduling for Energy and Performance on a CMP of SMT Processors, Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 06), 2006, pp. 117.

[4] A. Snavely, D. Tullsen, Symbiotic jobscheduling for asimultaneous multithreading architecture, Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.

[5] R. Teodorescu, J. Torrellas, Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors, Proceedings of the 35th International Symposium on Computer Architecture (ISCA), June 2008, pp. 363-374.

[6] J.A. Winter, D.H. Albonesi, C.A. Shoemaker, Scalable Thread Scheduling and Global Power Management for Heterogeneous Many-Core Architectures, 19th international conference on Parallel architectures and compilation techniques (PACT'10), pp. 29-40, 2010.

[7] C. Imes, H. Hoffmann, Minimizing Energy Under Performance Constraints on Embedded Platforms, 4th Embedded Operating Systems Workshop, EWiLi'14, Lisboa, Portugal, nov 13-14, 2014.

[8] F.A. Bower, D.J. Sorin, L.P. Cox, The Impact of Dynamically Heterogeneous multi-core Processors on Thread Scheduling, IEEE Micro, vol 28, Issue 3, pp. 17-25, jun 2008, (DOI) 10.1109/MM.2008.46

[9] L. Benini, A. Bogliolo, G. De Micheli, A Survey of Design Techniques for System-Level Dynamic Power Management, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 8, 3, pp. 299-316, 2000

[10] P. Singh, V. Chinta, Survey Report on ynamic Power Management, Survey report of University of Illinois, Chicago (ECE Department), Chicago, USA, 2008.

[11] S. Bilavarn, J.J. Khan, C. Belleudy, M.K. Bhatti, Effectiveness of Power Strategies for Video Applications: A Practical Study, Journal of Real-Time Image Processing, Springer, jan 2014. (DOI) 10.1007/s11554-013-0394

[12] H. Javaid, M. Shafique, J. Henkel, S. Parameswaran, System-level application-aware dynamic power management in adaptive pipelined MPSoCs for multimedia, Proceedings of the International Conference on Computer-Aided Design, pp. 616-623, 2011.

[13] R. Bonamy, S. Bilavarn, D. Chillet, O. Sentieys, Power Consumption Models for the Use of Dynamic and Partial Reconfiguration, Microprocessors and Microsystems, Elsevier, feb 2014.

[14] M. Horowitz, T. Indermaur, R. Gonzalez, Low-power digital design, In

Low Power Electronics. Digest of Technical Papers., IEEE Symposium (pp. 8-11), 1994.

[15] T. Damak, I. Werda, S. Bilavarn, N. Masmoudi. 2013, Fast Prototyping H.264 Deblocking Filter Using ESL tools, Transactions on Systems, Signals & Devices, Issues on Communications and Signal Processing, pp. 345362, 2013.

[16] 2005. ISO/IEC 14496-10, Advanced Video Coding for Generic Audio-visual Services, ITU-T Recommendation H.264, Version 4. (2005).

[17] Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit, http://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html (2015).

[18] F. Duhem, F. Muller, R. Bonamy, S. Bilavarn, FoRTReSS: a flow for design space exploration of partially reconfigurable systems, Design Automation for Embedded Systems, Springer Verlag, feb 2015.

A Video Monitoring Application for Wireless Sensor Networks using IEEE 802.15.4.

Abstract

In this paper, we present the development of a visual monitoring application for wireless sensor networks. This application has been designed specifically for low power video processing using IEEE 802.15.4 and power strategies to further extend battery life. We present in detail the implementation of the solution on specific platforms (Beagleboards) and an analysis of oportunities to exploit power strategies at the CPU level based on Dynamic Voltage and Frequency Scaling.

1 Introduction

Recent appearance of very low power and highly extended battery life systems called Wireless Sensor Networks (WSN) implies new levels of optimizations to be reached in many demanding applications. Such systems are already used in different domains ranging from medical to automotive applications. Video processing is a promising but challenging field of application with high CPU workload, thus power consumption requiring more important optimization.

In this paper we present the development of a video monitoring application designed to cope with the constraints of WSNs. We have developed this application with the goal to investigate specific power strategies for video processing and transmission over wireless networks. In the implementation section, we provide a detailed analysis and impacts of the application on the definition of an efficient power strategy. This work has been developed in the scope of the GEODES project which is supported by the Information Technology for European Advancement [1].

The paper is organized as follows: section 1 introduces the context of application, section 2 presents an overview of the solution developed and the implementation choices that have been made. Section 3 focusses on software implementation of the global system, which performances are discussed in section 4. This section will also present an analysis of how to save CPU power using Dynamic Voltage and Frequency Scaling techniques (DVFS). Section 5 concludes the paper.

2 Context

In this paper we focus more especially on video processing. Our scenario of application is to use sensors in environment video monitoring for fire detection. Thus we suppose some sensors are spread in a geographical region of interest (e.g. building), and that the information can travel through the nodes towards an end user (e.g.command center) that can eventually visualize what is happening.

Knowing that video algorithms are especially demanding in terms of CPU processing, that result in high power consumption, we will investigate specific reduction techniques based on DVFS.

In this context, the main requirement is to design a solution that brings maximum lifetime to the network. We target thus low power transmission using an IEEE 802.15.4 protocol and use low power sensors associated with ad hoc power strategies. We chose to use a MPEG2 compression standard in order to be able to run a video encoder on low end processing nodes.

Since we also need a multitude of drivers for different devices (camera, wireless communication, voltage and frequency scaling, ...), we chose to use a Linux operating system. This choice also permit to setup a mobile node with video display capabilities.

As it is very difficult to find currently a platform which able to comply with all these constraints (specific hardware needs and drivers), we setup a solution with the following elements: Beagleboard platform, Xbee, Linux Ubuntu. Hence Linux greatly helps to reuse existing blocks for this kind of application (video standards, drivers, display).

An overview of the solution is presented in **Figure 1**. It represents a basic framework where a capture node streams a monitoring sequence towards a single reception node that displays the video. The developement of this setup is detailed in the following sections.

Figure 1: Application overview

3 Application description

3.1 Hardware

The platform used for video capture is a Beagleboard [2] which is initially intended to be used for mobile phone architectures. The platform is composed of an OMAP

3530 processor from Texas Instruments that can run up to 720MHz. Although this is a bit oversized for our needs, it was not possible to find another platform able to handle all the requirements (video camera, 802.15.4, DVFS, image display, Linux, ...) at the time we started developing this application.
The video is captured on a standard USB webcam, compressed using MPEG2 and transmitted using a 802.15.4 compliant protocol. The RF communication is provided by modules called XBee that are connected through serial RS232 to the Beagleboard. In practice we use an additional FDTI Serial-USB adapter to realize the XBee/Beagleboard connection, so we can keep the RS232 port of the Beagleboard available for debug information access.
Those XBee module series are compliant with the IEEE 802.15.4 specification under certain limitations. One of them is the interface data rate that is limited to 115.2 Kbps. This limitation added to the overhead resulting from the Xbee's API communication mode results in 35 Kbps measured after the implementation of the XBee user space library. This have been obtained under nearly optimal conditions where only two nodes communicates and a channel distance that does not exceed 2 meters. It would be possible to improve the data rate and transmit more frames per second by using another 802.15.4 transceiver that would be less limited by its interface.

3.2 OS requirements

We made the choice of a GNU/Linux OS because it suits many different aspects of the application needs (video, drivers, frequency scaling, ...). Also, the Linux kernel can be tuned to fit with the exact hardware requirement of the nodes for our application. Thus its memory footprint can be reduced to meet the requirements of WSN nodes. In addition, most of the different WSN nodes that can be found target processors that are well supported by large communities of Linux developpers. The following describes the different Linux subsystems that need to be supported in our case.

Video4Linux (V4L)

Video capture is made using a USB camera which is supported by the V4L USB Video Class driver (UVC) included in the linux kernel tree. The use of UVC drivers thus provide much flexibility to work with different kind of USB video cameras (e.g. standard webcams).

XBee modules and USB-Serial FTDI adapter

The video stream is transmitted using a IEEE 802.15.4 protocol which is hold by XBee modules [3]. A XBee module is interfaced with a Beagleboard using a standard serial connection (RS232). For simplicity and convenience, we use USB adapters for this connection, so we can simply connect a XBee module through a USB port of the Beagleboard. The USB adapter for XBee modules uses a FTDI Serial-USB chip, a royalty-free driver is provided by FTDI to support this in the Linux kernel tree.

DVFS support: Linux-OMAP-pm

The Omap processor in the Beagleboards is well supported by Texas Instruments and an active community of Linux kernel developers. Among this community is the omap-pm group which is a branch focussing on power management. They provide appropriate drivers and support that permit to enable usual linux power management based on cpufreq and ACPI standards. In particular cpufreq support will permit to use Dynamic Voltage and Frequency Scaling (DVFS) techniques that brings high potential for CPU power savings.

Linux distributions

The eLinux community [4] is very active to handle the omap platform. Two differents Linux distributions were used during the developement. First, the Angstrom distribution [5] has been used on the capture node for its small operational footprint, where only the minimum services have been enabled (no X server). In a second step, an Ubuntu distribution [6] was tried to develop a video display capability on decoder nodes. Since it supports X display and many installation packages for ARM processors, such as the GNU toolkit, we switched to this environement for coding and debugging convenience.

Video encoding and decoding

The video application used is based on the MPEG2 reference code from the MPEG Software Simulation Group (MSSG) [7]. This code has been used as is, without modifications except the ones needed to stream the compressed bit stream through the Xbee modules. The codec operate on QCIF color images. Many optimizations could be considered in particular considering black and white images which may be sufficient for video monitoring. We chose not to focus on software optimization for the moment because our concern is first on power optimization.
Obviously, a low bit rate coding standard like H.264 would be a better match to maximize frame transmission speed. Nevertheless we have decided to use a MPEG standard for complexity reasons, especially on the encoder. The complexity of a H.264 encoder is something that can not necessarily be processed in typical CPUS of WSN nodes. A real time H.264 encoder could run on an OMAP but we recall this processor has been chosen because of the unavailability of all features needed for our application on other existing WSN development platforms. The OMAP processor is originally intended to mobile phones, so it is not an ideal solution for production WSN devices.

Wireless transmission

We decided to use the XBee modules of the series1, because they provide protocol support up to the MAC layer, leaving the user free to implement the upper layers. This choice allowed us to keep the maximum bandwidth for dedicated video transmission. As stated before, XBee modules are connected to the Beagleboards using FDTI USB adapters. The FTDI driver exposes a Linux character device in the ("/dev") directory which is accessed the same way as a typical Linux serial character device file (named ttyUSB* instead of /dev/ttyS*). Then we can simply use the termios C library to communicate with this character

device.
The XBee modules can be accessed in two ways. The first mode is the AT mode, in which all commands are sent to the xbee module using AT commands. The AT command mode is accessible thanks to the "+++" string followed by a wait of one second. All other strings sent are not interpreted. They are treated like data and are sent to the target node. This communication mode is not convenient for our application, it is easy to understand that a wait of one second is quite prohibitive. The second mode is the API mode. In this mode, all commands and data are included in frames and the previous waiting delay is no longer necessary. However, this implies an overhead when multiple data are sent to the same target node. For this application we want each node of the network to be able to communicate with multiple other nodes. For example, a node must be able to send its captured video to multiple receivers. Thus the choice of the AT command mode it not reasonable and the API mode must be used instead.

4 Application implementation with power management

4.1 Software implementation

The application has been developed to provide much flexibility and re-usability. Each main functionality is implemented in an independent C library. Four libraries are thus used and can be put together in many configurations to provide several behaviours and respond to different application needs.

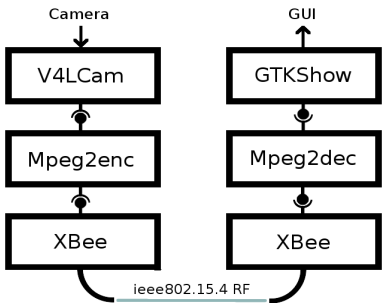


Figure 2: Software implementation

4.1.1 Libraries

Each library exposes a similar type of interface in a way to be used with the same logic. It is composed of two parts: a usual interface offers the user the possibility to call basic functionalities of the library and to handle the basic behaviours of the peripherals, and another interface that can be can be seen as a "component like" interface which is an implementation of the "Listener" design pattern. One can register or unregister the wanted listeners functions. Thanks to a dedicated ID, every time a new data is produced by the library core (that is threaded for most of them), all registered listeners will receive this data or will informed that the data has been updated.

Then it becomes easy to combine different libraries behaviour in any type of stream (list, tree, ...) and produce a more complex behaviour in the main program (**Figure 4.1**).

The first library that have been developed is related to XBee management. It permitted to make first performance measurements on the wireless transmission and most specifically on the behaviour of the XBee modules.

The V4LCam library aims at easing the use of any Video for Linux compatible hardware (video camera). A listener function is used to process new frames one by one at the desireded frame rate and video size (best effort strategy is used here).

The MPEG2 encoder and decoder are also encapsulated in a library. Both of them fit very well within the "component like" paradigm as the streaming sequence to monitor can be send frame by frame to the encoder which will output the compressed bitstream. Then, the compressed stream is sent to the decoder for MPEG2 decompression which will wait until sufficient data are available to produce a new decoded frame.

Finally, we made the choice to use the GIMP Toolkit (GTK) in order to develop a useful graphical user interface displaying the streaming video and differents metrics (e.g. transmission speed). Thus the GTKShow library can handle a video area (the GTKPixbuf of a given GTKWidget) and update it each time a new frame is added through the input interface of the library.

Incidentally, other libraries have been coded in a similar way. One of them intends to read or write data to a file which is useful, for example, to save the video stream independently of the nature of the video stream (encoded or raw).

4.1.2 Global application program

Using this "component like" paradigm, it becomes easy to glue together different library behaviours and build the application. It is possible to do this at run time either (**Figure 4.1**). We can thus easily derive different configurations of the application, e.g. sending the stream without encoding, receiving the video stream and computing some statistics without decoding, no displaying but just saving the video in a file, ... The main program is then only responsible of parsing parameters and combining the different libraries in order to cope as well as possible with the user parameters. Then if the GUI parameter is set, the main program creates the GTK GUI that displays the video and metrics information on the screen.

4.2 Performances analysis

A screen capture showing the demonstrator is provided in **figure 3**. The beagleboard used for encoding is located on the bottom right, where it is connected to the camera and Xbee module through an externally powered USB hub. A second beagleboard is used for decoding (left of the figure). It is also connected to a Xbee module for reception and a DVI screen (behind) to display the monitoring sequence.

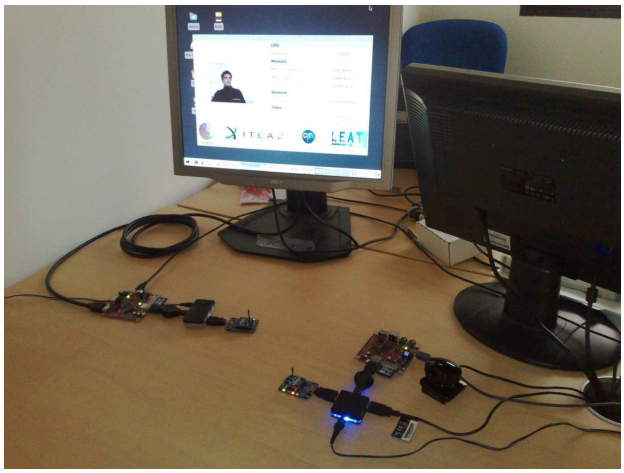


Figure 3: Video monitoring demonstrator

In order to study oportunities for power optimization using DVFS, we have first analyzed the execution of the application in the different configurations of the target platform. We have setup different operating frequencies on the OMAP CPU to run both the encoder and decoder without transmission and measure the pure video processing performances. We used an omap-pm distribution to benefit from a cpufreq support. Results are reported in table 1.

OMAP 3530 (beagleboard)						
Freq. (MHz)	125	250	500	550	600	720
Encoder (fps)	0.95	1.92	3.80	4.16	4.76	5.61
Decoder (fps)	23	43	92	101	111	130

Table 1: MPEG2 encoder and decoder performances for different OMAP frequencies

These results show that the the maximum frame rate is limited to 5.61 fps by the encoder. This imply that the decoder can always operate at its lowest frequency (125MHz), thus saving power knowing that the default clock frequency on the beagleboard is 500MHz. Hence we also expect to be highly limited by the real transmission speed. In order to check this, we equiped the application with a run time measure of the transmission speed that has reported the following results:

- Maximum bit rate: 4KB/sec
- Average compressed frame size: 2KB
- Maximum corresponding frame rate: 2.3 frame/sec

As for the decoder, we can avoid running the encoder at frequencies greater than 250MHz (in this version of the application, without optimizations), since the bit rate provided in this case can not be handled by the wireless modules.

4.3 Impacts on power management

Given previous considerations on the encoder and transmission speed, a natural power strategy would be the following: since the transmission speed can vary a lot depending on the network traffic, we can benefit from several orders of power reduction if we adapt the frequency at which the CPU operates (thus the encoder performance in terms of frame per second) to the instantaneous transmission speed available. Typically:

- if the bit rate is less than 2KB/sec, it is useless to run the CPU at more than 125MHz.
- if the bit rate is less than 4KB/sec, it is useless to run the CPU at more than 250MHz.
- if the bit rate is less than 8KB/sec, it is useless to run the CPU at more than 500MHz.

The potential power reduction can be very sensitive if we consider that without this strategy, we would have to run the CPU at 500MHz to comply with all cases. We recall that dynamic power consumption is proportional to CV^2F where F is the processor frequency and V the associated voltage. So reducing the frequency from a factor of two will result in a similar range of power reduction.

5 Conclusion and perspectives

In this paper, we have presented the design of a low power video monitoring application over wireless sensor networks. We have shown the possibility to use a 802.15.4 protocol to enable this application along with measures that can permit to evaluate the expected performance of such a system. We have also presented how to derive an efficient power strategy to further reduce power by using DVFS techniques, which is based on exploiting the variations that can occur at transmission level and by adapting the processor frequency to cope with the bit rate that can be actually delivered. Future investigations will focus on power management strategies at the decoder level [8] and at the scheduling level [9] on the basis of previous works.

References

[1] GEODES: *http://geodes.ict.tuwien.ac.at/*, Information Technology for European Advancement, 2008-2011.

[2] Beagleboard: *http://beagleboard.org/*, Omap3530 Beagleboard official: systeme reference Manual

[3] XBee Series 1: *http://ftp1.digi.com/support/documentation/90000982_B.pdf* XBee series 1 manual, Digi

[4] eLinux: *http://elinux.org/BeagleBoard*,

[5] Angstrom: *http://code.google.com/p/beagleboard/wiki/HowToGetAngstromRunning*

[6] Ubuntu: *http://elinux.org/BeagleBoardUbuntu*,

[7] MPEG2: *http://www.mpeg.org/MPEG/video/mssg-free-mpeg-software.html*, MPEG Software Simulation Group (MSSG), 1994

[8] S. Bilavarn, C. Belleudy, M. Auguin, T. Dupont, A-M. Fouilliart, *Embedded Multicore Implementation of H.264 Decoder with Power Management Considerations*, 11th Euromicro Conference on Digital System Design, DSD 2008, Parma, Italy, 03/09/08-05/09/08

[9] R. Chéour, S. Bilavarn, M. Abid, *EDF scheduler technique for wireless sensor networks: case study*, 4th International Conference on Sensing Technology, June 3-5, Lecce, Italy

Received: 16 October 2017 | Revised: 19 March 2018 | Accepted: 29 April 2018

DOI: 10.1002/cta.2508

RESEARCH ARTICLE

WILEY

Energy efficient mapping on manycore with dynamic and partial reconfiguration: Application to a smart camera

Robin Bonamy[1] | Sébastien Bilavarn[1] | Fabrice Muller[1] | François Duhem[2] | Simon Heywood[2] | Philippe Millet[2] | Fabrice Lemonnier[2]

[1]Laboratory of Electronics Antennas and Telecommunications, University of Nice Sophia Antipolis, CNRS UMR 7248, Sophia Antipolis, France

[2]High-Performance Computing Lab, Thales Research and Technology, Palaiseau, France

Correspondence
Sébastien Bilavarn, Laboratory of Electronics Antennas and Telecommunications, University of Nice Sophia Antipolis, CNRS UMR 7248, Sophia Antipolis, France.
Email: bilavarn@unice.fr

Summary

This paper describes a methodology to improve the energy efficiency of high-performance multiprocessor architectures with dynamic and partial reconfiguration (DPR), based on a thorough application study in the field of smart camera technology. Field-programmable gate arrays are increasingly being used in cameras owing to their suitability for real-time image processing with intensive, high-performance tasks and to the recent advances in dynamic reconfiguration that further improve energy efficiency. The approach used to best exploit DPR is based on the better coupling of 2 decisive elements in the problem of heterogeneous deployment: design space exploration and advanced scheduling. We show how a tight integration of exploration, energy-aware scheduling, common power models, and decision support in heterogeneous DPR multiprocessor system-on-a-chip mapping can be used to improve the energy efficiency of hardware acceleration. Applying this to a mobile vehicle license-plate tracking and recognition service results in up to a 19-fold improvement in energy efficiency compared with software multiprocessor execution (in terms of energy-delay product) and up to more than a threefold improvement compared with a multiprocessor with static hardware acceleration (ie, without DPR).

KEYWORDS

dynamic partial reconfiguration, energy efficiency, multicore, manycore, smart camera

1 | INTRODUCTION

The use of heterogeneous multiprocessor system-on-a-chip devices (SoCs) has grown because of their potential to address energy efficiency, power, and heat density problems, as we reach the limits of CMOS technology scaling. Although promising, the increasing level of computational power, heterogeneity, and energy efficiency requirements in new systems greatly affects their design and deployment complexity. Indeed, it is no longer simply a question of efficiently mapping concurrent processes to the right cores in typical multicore/manycore platforms, but also taking into account critical dynamic aspects such as power management (Dynamic Voltage and Frequency Scaling (DVFS), clock gating, and power gating) and hardware acceleration (e–field-programmable gate array (FPGA), dynamic and partial reconfiguration (DPR), and Graphics Processing Unit (GPU)) to deliver the best processing efficiency at each instant.

Hardware acceleration is a relatively well-known player in the energy performance equation, which is regaining attention with the advent of DPR. Partial reconfiguration is a technique related to FPGAs that can be used to extend their inherent flexibility: It allows specific FPGA regions to be reprogrammed with new functionalities while other regions

continue running. A significant reduction in hardware resource use results in less static power, and thus better energy efficiency, adding to the inherent benefits of dedicated hardware (ie, statically reconfigurable accelerators). However, a variety of parameters such as FPGA region partitioning, scheduling, accelerator parallelism, and multiprocessor execution opportunities strongly affect the actual processing efficiency and combine with other techniques such as blanking or DPR-based clock gating that can also be used to further decrease power. As a result, the quantity and scope of decisions in a dynamically reconfigurable multicore/manycore system greatly complexifies the scheduler's job. It is, however, critical to provide good support at this level, since bad decisions can affect energy efficiency to the point of total ineffectiveness. In this paper, we describe a methodology better able to address these considerations by further integrating the definition of advanced energy-aware scheduling and exploration or application mapping analysis. This makes for a good combination of the improvements specific to hardware acceleration and DPR, which are investigated in depth through their real-life application to a mobile vehicle license plate tracking and recognition service.

The outline of the paper is as follows. We first review existing works in the field of heterogeneous multicore scheduling, pointing out relative novelty in the use of DPR for that purpose. In Section 3, we introduce the proposed methodology and its underlying power models. We then consider a dedicated power minimization policy defined for heterogeneous deployment and scheduling purposes in Section 4. Detailed results of a license-plate recognition (LPR) application on Xilinx devices are analyzed and discussed in Section 5. Finally, we present our principal conclusions from the detailed case study and future directions for the research and its applications.

2 | RELATED WORK

2.1 | Energy efficiency in heterogeneous multiprocessors

With the rise of multicore heterogeneous architectures, there have been many works addressing the efficient scheduling of application workloads for multiple cores, where the cores can be of different types. Early investigations focused primarily on improving load balancing to achieve better performance (throughput, instructions per cycle, etc). Recently, the challenge of heterogeneous thread scheduling and global power management has hinged on delivering higher power-performance levels (performance per watt). Many works explored the energy efficiency benefits of heterogeneity; these are extensively discussed, for instance, in Leech and Kazmierski.[1] There is a broad consensus that exploiting heterogenity is essential for better use of energy, the inevitable conterpart being that it greatly complexifies the job of the scheduler.

Of the work addressing this problem, Ghiasi et al[2] proposed a scheduler for a system of processors based on execution prediction to map future processing needs to the most suitable processor. Their method applies to single-ISA heterogenity supporting differing voltages and frequencies. DeVuyst et al[3] extended a symbiotic scheduling heuristic,[4] originally developed to enhance throughput and lower response time, for chip multiprocessors with simultaneous multithreading cores. They report up to 7.4% savings in energy, 10.3% savings in energy-delay product (EDP), and 35% savings in power. The study of Teodorescu and Torrellas[5] is another contribution that considers both scheduling and power management to address process variations in Chip Multi-Processors (CMPs). They conducted a design exploration, proposed a number of schedulers to satisfy different objectives, and developed a linear programming solution for power management. Winter et al[6] examined the scalability problem for manycores, comparing basic scheduling heuristics and proposing scheduling and power management algorithms for heterogeneous systems scaling up to 256 cores. Recent studies like Imes and Hoffmann[7] started to address the ARM big.LITTLE architecture that combines an energy-efficient processor cluster (Cortex-A7) with a higher performance processor (Cortex-A15). Specifically, this case study reports that different classes of resource allocation heuristics (race-to-idle vs never-idle) have very different results on different platforms, indicating that the efficiency of a strategy greatly depends on platform characteristics and can go as far as reducing efficiency in some cases.

Aside from the question of core specialization, heterogeneity also extends to the aspects arising at run time due for example to power management.[8] The dynamic use of different power states (P-states and sleep states) available for each core matches the problem of low-power scheduling, which had a long history of research over the last 20 years. Surveys have been described, for example, in Benini et al[9] and Singh and Chinta[10] from the abundant literature. Our own previous experimental studies in this field[11] came to similar conclusions as Imes and Hoffmann[7] and Javaid et al[12] concerning the importance of platform characteristics and application knowledge on the efficiency of a strategy. Likewise, results indicate that custom strategies, ie, more specialised schedulers dedicated to an application or application domain, can reach significant levels of energy gains (in the 5% to 50% range for video-processing applications on representative platforms) compared with existing OS and platform-based strategies.

In this large body of literature, existing works often consider a specific perspective on the heterogeneous scheduling problem: a type of architecture; 1 precise objective (manycore scaling and process variability); limited heterogeneity (similar cores and DVFS); etc. In addition, a type of heterogeneity remains relatively unexplored regarding the recent advancements in graphics processing units and reconfigurable processing units: the possibility of running tasks in hardware (or using hardware acceleration). Leech and Kazmierski[1] thus envision adding heterogeneity to hardware acceleration by taking advantage of progress in DPR and high-level synthesis (HLS). It is however worth noting that Intel is considering, in addition to mixing large and small cores, the use of accelerators, including FPGAs, for heterogeneous architecture research.[13] The following introduces relevant works in this field with an emphasis on their possible use in heterogeneous multicore systems.

2.2 | Hardware acceleration

Hardware acceleration delivers several times better performance and energy efficiency compared with software execution, and the flexibility introduced with DPR can improve these benefits. The underlying heterogeneous mapping and scheduling problems have started lately to be reinvestigated with the possibility of dynamic reconfiguration of hardware tasks, adding new dimensions to heterogeneity, resource management, scheduling, and energy efficiency issues.

Early works like Bazargan et al[14] and Hong et al[15] addressed run-time management for dynamically reconfigurable systems, which were mainly motivated by the search for the best fit/speedup trade-off. The work of Liu et al[16] is a first significant work addressing DPR from a practical energy-efficiency perspective. They show that using DPR to eliminate the power consumption of the accelerator when it is inactive (blanking) can outperform clock gating and reduce the energy consumption by half. But the application study on 64-bit division accelerator makes it difficult to extrapolate conclusions to a more complex application level. Notably, it does not address DPR ability to share and reuse reconfigurable regions (RRs) to further reduce static power nor does it address whether or not DPR improves energy compared with static acceleration.

Energy models of DPR have been previously investigated in depth in Bonamy et al[17] with a real-life application study on an H.264/AVC video profile decoder. Three functions are accelerated using HLS, and an analysis is conducted to identify a DPR scheduling solution and compute the associated energy cost. Results reported 22.5% energy gains for dynamic over static execution. Both previous works and others pointed out the importance of configuration speed as an essential condition for energy efficiency, but various opportunities lie in dynamic reconfiguration.

Aside from blanking, which reduces energy consumption by decreasing the share of static power associated with RRs,[18] some low-level techniques investigated the use of dynamic reconfiguration to reduce clock-related losses. A low-overhead clock-gating implementation based on dynamic reconfiguration has been proposed in Sterpone et al,[19] achieving a 30% power reduction compared with standard FPGA clock-gating techniques based on Lookup Table (LUTs). Another approach has been developed to modify the parameters of clock tree routing during run-time reconfiguration to moderate clock propagation across the whole FPGA and decrease dynamic power.[20] Finally, self-reconfiguration also allows online modification of clock frequency with low resource overhead by acting directly on clock management units from the reconfiguration controller.[21]

Although such DPR-based features are fully effective with the potential to improve the already significant efficiency inherent to hardware processing, there are significant practical problems that remain to be addressed. The most important of them is the need for integrated methodologies able to fully explore their benefits before final implementation. For a long time (and even now), the same reason has prevented the widespread adoption of reconfigurable technologies, as too much hardware expertise is needed to quickly produce fully working accelerators from software programming languages such as C and C++. Indeed, 2 main directions of research stand out in the light of recent works: methodologies and programming models.

There has been a lot of advances since the emergence of HLS in recent years. SDSoC from Xilinx[22] is among the most advanced heterogeneous development tools in this regard. It helps in particular to compensate for the lack of automated system connectivity generation in HLS, the design of which is still critical and very time consuming. However, there are other dimensions to explore at the system level, as far as DPR and power are concerned, that are still not being investigated enough despite a variety of design methodologies available in the literature. ReConOs,[23] for example, is one of these representative efforts to develop approaches targeting heterogeneous CPU/FPGA systems, with a unified programming model, and execution environment for threads running in software and reconfigurable hardware. Among more recent works, Processor Accelerator Architecture Simulator (PAAS)[24] is another design approach for CPU with Application-Specific Integrated Circuit (ASIC)- or FPGA-based accelerators in which the focus is on memory hierarchy.

Other works like PolyPC[25] addressed an OpenCL-based framework to implement a custom hardware platform using HLS. Despite all these necessary contributions, it is worth noting that (1) all of them examine aspects of the problem from a performance perspective; (2) very few address the definition of a consistent DPR methodology, although of course some works like Schwiegelshohn and Hubner[26] do consider DPR, but rather in the scope of a particular application study; and (3) few works investigate power and energy efficiency as an explicit primary focus.

There is currently a profusion of research investigating the hardware/software code compilation problem, from the programming model perspective to the extraction of abstract parallelism, on-chip communications, and platform integration. Indeed, a variety of works address specifically the use of GPU-based (OpenCL[22,27] and Compute Unified Device Architecture (CUDA)[28]), or domain-specifc languages such as HIPAcc[29] or previously RVC-CAL[30] for video and image processing. Aside from the fact that they do not truly address DPR, the primary objective in these works is to automate the generation of statically reconfigurable code to achieve the best performance. Compared with these works, we aim to develop a methodology that fully supports DPR techniques and is explicitly centered on energy efficiency, covering in particular complete power modeling and energy-aware scheduling to identify very highly energy-efficient mapping solutions.

3 | EXPLORATION AND SCHEDULING METHODOLOGY

3.1 | FoRTReSS methodology

3.1.1 | Overview

FoRTReSS stands for flow for reconfigurable architectures in real-time systems.* It is a framework helping the user to explore dynamic and partially reconfigurable multiprocessor systems. Given an application or a set of applications, it basically searches a set of RRs and processor cores and simulates time-accurate mappings of hardware and software application tasks using fully defined scheduling strategies. Each task has a set of possible implementations, corresponding to a resource and performance trade-off. These implementations can be in hardware (to be mapped on an RR) or software (to be mapped on a processor core). A hardware implementation is defined by the actual task resources and performance that can be determined from FPGA synthesis or from estimations (both with the help of HLS). There may be several hardware implementations of the same task to reflect different parallelism trade-offs. A software implementation is mainly characterised by the task execution time on a CPU core, possibly at different operating frequencies. The target architecture is given however by a precise description of FPGA resource organization and a set of processor cores. From application tasks and architecture descriptions, a built-in SystemC/TLM simulator engine called RecoSim is used, first to look for the best RRs that are able to correctly host the tasks. Then, using additional performance and timing characteristics (deadline, period, etc), it simulates the mapping and scheduling of the full application for various combinations of RRs and cores. A detailed description of this methodology is given in publications such as Duhem et al.[31,32] In this work, we focus on our extensions and an original methodology that have been developed on this base so as to provide support for very highly energy efficient mapping on multicore/manycore platforms with DPR.

3.1.2 | Extensions

The goal of the FoRTReSS methodology was initially to ensure a given performance level, potentially under real-time constraints, for a given application. Here, we aim to exploit the same DPR flow, focusing initially on real-time reconfigurable multiprocessors, but we extend the methodology so as to manage power consumption and significantly improve the energy efficiency (targeting real-time reconfigurable multicore/manycore platforms). The founding principles of this approach arise from recognising the very complex heterogeneity implied by software and hardware execution in a modern multiprocessor system (management of cores and accelerators, DVFS, clock gating, power gating, scheduling, etc). Therefore, the methodology is based on better coupling 2 decisive and tightly dependent factors in heterogeneous deployments in general, especially when DPR is concerned: design space exploration (or application mapping analysis) to find the best architecture settings with deployment estimations (eg, how many cores, what types of cores, number, FPGA partitioning, and size and shape of RRs) and scheduling to further improve the actual mapping at run time on the identified architecture. Indeed, the pertinence of mapping analysis depends obviously on the scheduler, which role is essential in the fine-grained exploitation of the architecture resources. Defining dedicated advanced schedulers that better exploit application knowledge at run time is also an important opportunity to grasp, as it can bring up to 50% more energy gains.[11]

*http://fortress-toolbox.unice.fr

Additionally, exploration and scheduling require fast and reliable decision support to (1) allow a large design space to be analyzed and (2) quickly evaluate scheduling choices at run time. Both processes can greatly benefit from using the same estimations to improve the coherence of design and scheduling decisions.

3.2 | Power modeling

The first step to achieving this is to extend the various FoRTReSS resource and performance models with efficient power characterization. This can be beneficially based on our previous studies, which led to define pragmatic, abstract, and overall energy modeling of reconfigurable multiprocessor systems.[17,33] The following describes main features of the models (FPGA, CPU, application, and mapping) and some modifications made to fit our requirements, notably concerning multiprocessor power characterization.

3.2.1 | FPGA

Previous and current work has been conducted on Xilinx devices. To define a consistent formalization hereafter, we use a more general terminology for FPGA resources: logic cells (Xilinx *Slices* or Altera *Logic Elements*), RAM, and DSP blocks.

The power model for RRs is divided into 3 components that reflect their static, idle, and run power. Static power P_j^{static} is the power consumed when region j is empty, and which depends on the configurable resources of the region. We approximate P_j^{static} as a proportion of the full FPGA static power in terms of logic cells only; therefore, P_j^{static} can be more conveniently derived for any size and shape of partition.

Since the idle and run power associated with region j depends on the actual task i configured on it, these values are characterized by an application and mapping model. Idle power $P_{i,j}^{idle}$ is the extra power required when task i is configured on region j but not running. Run power $P_{i,j}^{run}$ is the additional power consumed by task i being executed on region j. Idle and run power can be determined by using postsynthesis estimations (eg, Xilinx power estimator), but in the following, they are determined by direct measurements on the FPGA (to improve accuracy of results) for each hardware task implemented.

Reconfiguration overheads must also be addressed to ensure if there is an actual benefit in using complex dynamic reconfiguration. The reconfiguration controller is modeled with 2 parameters P^{reconf} and T^{reconf} to characterize its power and performance. As with static power, the reconfiguration time T_j^{reconf} and energy E_j^{reconf} required to reconfigure region j are derived from the number of logic cells in the region.

3.2.2 | CPU system

Figure 1 shows power measurements of the dual Cortex-A9 in various configurations (frequency, idle, and running cores). As with the FPGA model, power can be divided into static, idle, and run components. Static power P_{cpu}^{static} has a stable value that can be evaluated at 180.78 mW (intersection of the 3 linear trends at $F = 0$ MHz). Idle and run power depend on the number of cores and frequency. At this stage, we consider a preliminary model at constant nominal frequency (667 MHz). This model will be extended in future work to DVFS (using representative multiprocessor platforms like Exynos) and also

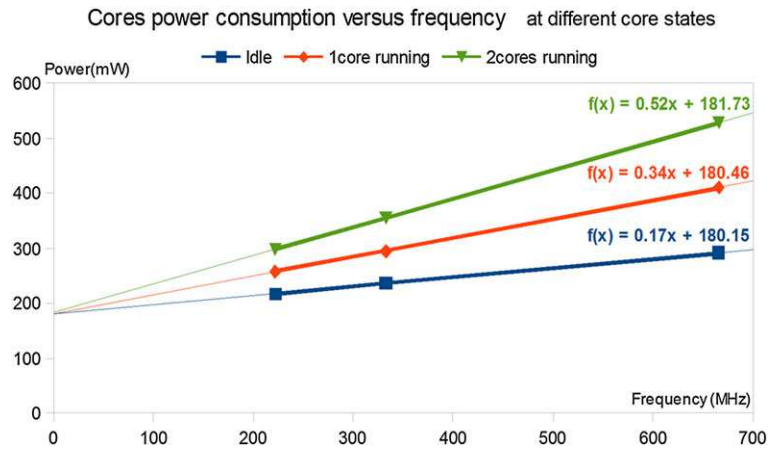


FIGURE 1 Power consumption of Zynq processor measured on the ZC702 platform [Colour figure can be viewed at wileyonlinelibrary.com]

to soft cores (eg, MicroBlaze). Under these conditions, the idle power P_{cpu}^{idle} of the CPU system is a function of the number of cores N_{cores}:

$$P_{cpu}^{idle} = C_{cpu}^{idle} \times N_{cores},$$

where C_{cpu}^{idle} is a coefficient expressing the *idle* contribution for a given type of CPU, whose value is 111.66 mW for a Zynq processor (derived from previous measurements). Run power P_{cpu}^{run} of the CPU system is a function of the number of running cores $N_{running\_cores}$:

$$P_{cpu}^{run} = C_{cpu}^{run} \times N_{running\_cores},$$

where C_{cpu}^{run} is a coefficient expressing the *run* contribution for a given type of CPU, whose value is 119.39 mW for a Zynq processor (derived from previous measurements). Therefore, the total CPU power is

$$P_{cpu} = P_{cpu}^{static} + P_{cpu}^{idle} + P_{cpu}^{run},$$
$$P_{Zynq} = 292.44 + 119.39 \times N_{running\_cores}.$$

3.2.3 | Application and mapping

An application is characterized by a set of nodes (tasks) with data and execution dependencies, usually modeled by a task flow graph ($\mathcal{G}$). To support the modeling of more complex applications, this specification model has been extended to control structures (which means that communications from a node to another can be conditional), hierarchy (a node can be another task flow graph), and multiapplication (several applications can be instantiated).

Each task has 1 or more implementations available, but every task has at least a software implementation. An implementation is a description of how the task is executed on a software or hardware execution unit. The implementation model reflects the task id i, the execution unit j, an execution time $T_{i,j}$, and—for hardware execution—idle power $P_{i,j}^{idle}$ and run power $P_{i,j}^{run}$. For software execution, idle and run power of a task coincide with idle and run power of a core, which can be derived from previous CPU model.

3.2.4 | Model evaluation

We apply our model to a real example so as to evaluate the extent of our formalization and show the setting of model parameters from a full measurement process. The input specification is a LPR software (C++) to be mapped onto Xilinx platforms (ML605, Zynq-7000). A detailed application description is provided in section 5.1. In the following, we show how the various model parameters were determined from physical measurements on a ZC702 platform (containing a dual-core ARM Cortex-A9 MPCore on a 28-nm Xilinx Artix-7 device) to show the applicability and relevance of the models.

In the mapping characterization of section 3.2.3, hardware and software task parameters can be settled by defined implementations and measures (Table 1). Classical profiling can be used to set out software execution time and derive the associated energy cost from previous CPU power P_{Zynq} (eg, $E_{2,1}^{1core\_running} = 412mW * 17.5ms = 7.21mJ$ for the *dilate* function when core 1 is running and core 2 is idle). For the xc7z020 device, *Cores* are the 2 software execution units ($j = [1;2]$) of the platform, and *RRs* are the hardware execution units ($j \geq 3$) that will be automatically defined during the exploration of FPGA partitioning.

In the following, hardware tasks are fully generated using an electronic system level methodology described in Damak et al[34] to provide maximum relevance to our results. Mapping parameters are thus derived from measurements made possible by full accelerator implementation (they could have been defined more easily using estimation tools like Xilinx power estimator). $P_{i,j}^{idle}$ is the consumption measured when hardware task i is configured on RR j but not running. This power is supposed to be independent from RRs in our model. $P_{i,j}^{run}$ is the fraction of dynamic power added when hardware task i is running on RR j (also assumed to be independent of the RR used), which can be determined in practice by subtracting the consumption of a configuration where the task is running from the consumption of a configuration where the task is idle. The total power consumption of hardware task i on RR j is therefore the sum of P_j^{empty} of RR j and $P_{i,j}^{idle}$ when the task is idle, plus an additional contribution $P_{i,j}^{run}$ when the task is running. For example, the hardware implementation of the *dilate* function (task 2) on RR 3 (assuming it is made of 4000 slices) in Table 1 has a total energy cost $E_{2,3}^{run}$ computed

TABLE 1 Hardware/software characterization of LPR application tasks on the ZC702 platform

Function (i)	Execution Unit (j)	$T_{i,j}$, ms	$P^{idle}_{i,j}/P^{run}_{i,j}$, mW	$N^{cell}; N^{bram}; N^{dsp}$
Img_load ($i = 1$)	*Core* ($j = [1;2]$)	31	…	…
dilate ($i = 2$)	*Core* ($j = [1;2]$)	17.5	…	…
	RR ($j \geq 3$)	4.3	38/63	2718; 0; 0
erode_fifo ($i = 3$)	*Core* ($j = [1;2]$)	167	…	…
	RR ($j \geq 3$)	12.1	42.6/46	3554; 0; 0
erode ($i = 4$)	*Core* ($j = [1;2]$)	17.3	…	…
	RR ($j \geq 3$)	4.4	35/64	2681; 0; 0
binarize ($i = 5$)	*Core* ($j = [1;2]$)	24	…	…
Img_preproc ($i = 6$)	*Core* ($j = [1;2]$)	24	…	…
Img_write ($i = 7$)	*Core* ($j = [1;2]$)	15	…	…

Abbreviation: LPR, license plate recognition.

from P_3^{empty} of RR j ($4000 \times P_j^{static} = 4000 \times 0.002180mW/slice = 8.72mW$), $P_{2,3}^{idle}/P_{2,3}^{run}$ of function *dilate* (task 2), and the corresponding execution time $T_{2,3}$:

$$\begin{aligned} E_{2,3}^{run} &= (P_3^{empty} + P_{2,3}^{idle} + P_{2,3}^{run}) \times T_{2,3} \\ &= (8.72mW + 38mW + 63mW) \times 4.3ms \\ &= 0.472mJ. \end{aligned}$$

4 | ENERGY-AWARE HETEROGENEOUS SCHEDULER

This section describes an advanced energy-aware heterogeneous scheduler (EAHS) developed within the above framework and which exploits DPR so as to minimize the energy cost, possibly under performance and deadline constraints. The scheduling procedure involves 2 steps: A waiting task list is first determined and sorted according to specific criteria (eg, earliest deadline first [EDF]), then a task implementation is chosen and mapped to a resource such that the resulting execution cost (including reconfiguration overheads) is minimized. As such, it requires that the power impact of resource allocation decisions can be correctly and quickly predicted, which relies on the models of section 3.2.

4.1 | Task scheduling

Common scheduling strategies fall under 2 broad categories: time sharing and time-critical scheduling. Time-sharing algorithms are based on the principle where each process takes an equal share of CPU time in turn, without priority (ie, round robin). On the other hand, time-critical scheduling is more concerned with classes of real-time systems where processing of jobs must be completed under strict execution constraints.

Deadline scheduling, for example, is often associated with dynamic priorities that increase when a process becomes more critical. However, the strict definition of deadlines may be alleviated for the sake of heterogenous scheduling as power becomes the growing constraint. This provides ways to derive energy-aware strategies under performance constraints. Considering, for example, a scenario where the top task in the waiting list is blocked, waiting for a matching execution unit to be free, then all other tasks in the waiting list are blocked despite the fact that they may be run on other free execution units. This tends to increase the application latency and underuse of resources. Thus, we can devise a *soft* scheduling approach that consists of using a deadline strategy (eg, EDF) while leaving the possibility for a blocked task to be bypassed when its corresponding mapping is momentarily infeasible (eg, either the reconfiguration controller is busy or a compatible execution unit is not free).

4.2 | Task mapping

When processing the allocation of a task to the resources of a heterogeneous platform, the choice of implementation is the key part of reducing energy. A cost function is thus defined to help identify the most energy-efficient mapping from all possibilities. This cost is computed each time a task at the top of the waiting list is eligible for execution, for all its

possible implementations, and this process is based on energy and execution time estimations of the implementations of the task being processed.

The energy estimation is given by Equation 1:

$$\forall j \; ; \; E_j^{cost} = \left\{ \begin{array}{ll} E_{i,j}^{run} & \text{when } \rho_{i,j} = 0 \\ E_{i,j}^{run} + E_j^{reconf} & \text{when } \rho_{i,j} = 1. \end{array} \right\}, \tag{1}$$

where

$$E_{i,j}^{run} = P_{i,j}^{run} \times T_{i,j},$$
$$E_j^{reconf} = T_j^{reconf} \times P^{reconf},$$

and $\rho_{i,j}$ represents the need to perform a reconfiguration or not. For instance, if the same implementation is already configured, $\rho_{i,j} = 0$ means that region j can be used directly to run task i; otherwise, $\rho_{i,j} = 1$, and a reconfiguration is required before execution. For all nonreconfigurable execution units, $\rho_{i,j} = 0$.

To execute a new task, the scheduler has to check first that the execution unit j currently evaluated is free. If not, the task can be implemented later, and this delay must be considered for decision. Execution time of a task is thus estimated as described in Equation 2:

$$\forall j \; ; \; T_j^{cost} = T_{i,j} + T_j^{reconf} \times \rho_{i,j} + T_j^{busy}, \tag{2}$$

where T_j^{busy} represents the time during which execution unit j is busy running another task i', which can be estimated from the execution time $T_{i',j}$, the start time of current task i, and the current scheduling time.

The final cost for execution unit j is computed in Equation 3, where α is a user parameter (a real value between 0 and 1) to promote either performance (α close to 0) or energy (α close to 1).

$$\forall j \; ;$$
$$Cost_j = \alpha \frac{E_j^{cost}}{max(E^{cost})} + (1 - \alpha) \frac{T_j^{cost}}{max(T^{cost})}. \tag{3}$$

The cost function is evaluated for all available implementations and execution units for the current top task in the waiting list. The lowest $Cost_j$ value is selected, and the task is implemented on execution unit j. However, if execution unit j is busy, the task is kept waiting and will be implemented later.

5 | APPLICATION TO A SMART CAMERA SYSTEM

This section illustrates the search for significant energy efficiency improvement in a manycore system with DPR accelerators. This real-life case study concerns a mobile vehicle license plate–tracking and –recognition application developed for the purpose of highway traffic management. Two representative families of devices are used to address realistic DPR implementations with Xilinx-based FPGAs: Virtex-6 with MicroBlaze soft IP cores and Zynq-7000 with ARM Cortex-A9 hard IP cores. All power and performance models used in this application study are therefore based on full prototyping hardware accelerators and real platform measurements (rather than estimations) so as to improve the relevance of the subsequent power analysis. The 2 platforms used for this characterization are the ML605 and ZC702 evaluation boards.

5.1 | LPR application overview

License plate recognition is popular in the transportation industry for its applications in managing traffic congestion and in monitoring, security, and access control systems. In our particular scope of application, the LPR system is designed for automatic detection and identification of vehicles passing highway toll barriers. Cameras (1 per lane) are set up to detect and follow the front plates of cars in each lane. The system can process different lanes in parallel; this is illustrated for 2 lanes in Figure 2. Here, we will consider an LPR system that can be set up to monitor 1, 2, 4, or 8 lanes simultaneously (referred to as 1-, 2-, 4-, and 8-lane LPR).

The lower area of each lane's video feed is used to track the license plate of the current car while the upper area is used to detect the plate of the next car. The 2 regions (*Main task* and *Follow task*) are visible on the right side of Figure 2. Both regions undergo a succession of typical morphological operations and transforms (erosion, dilation, etc) to improve the

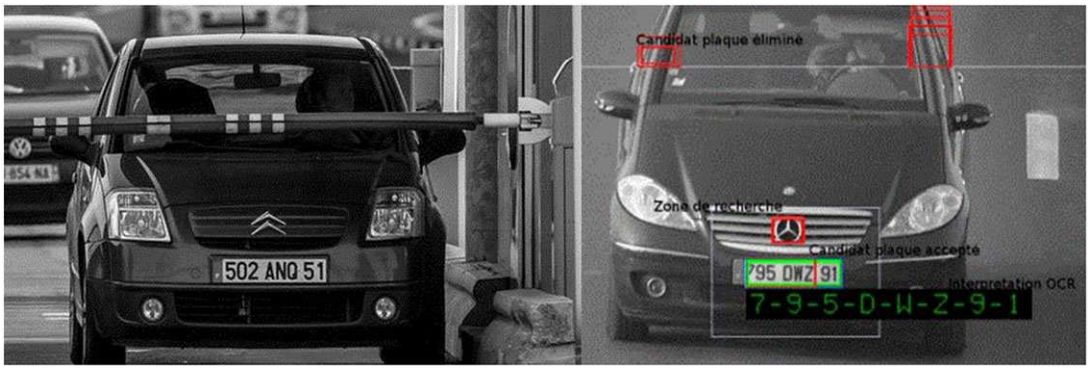


FIGURE 2 License plate detection and recognition in a configuration of 2 monitored toll lanes (2-lane license plate recognition) [Colour figure can be viewed at wileyonlinelibrary.com]

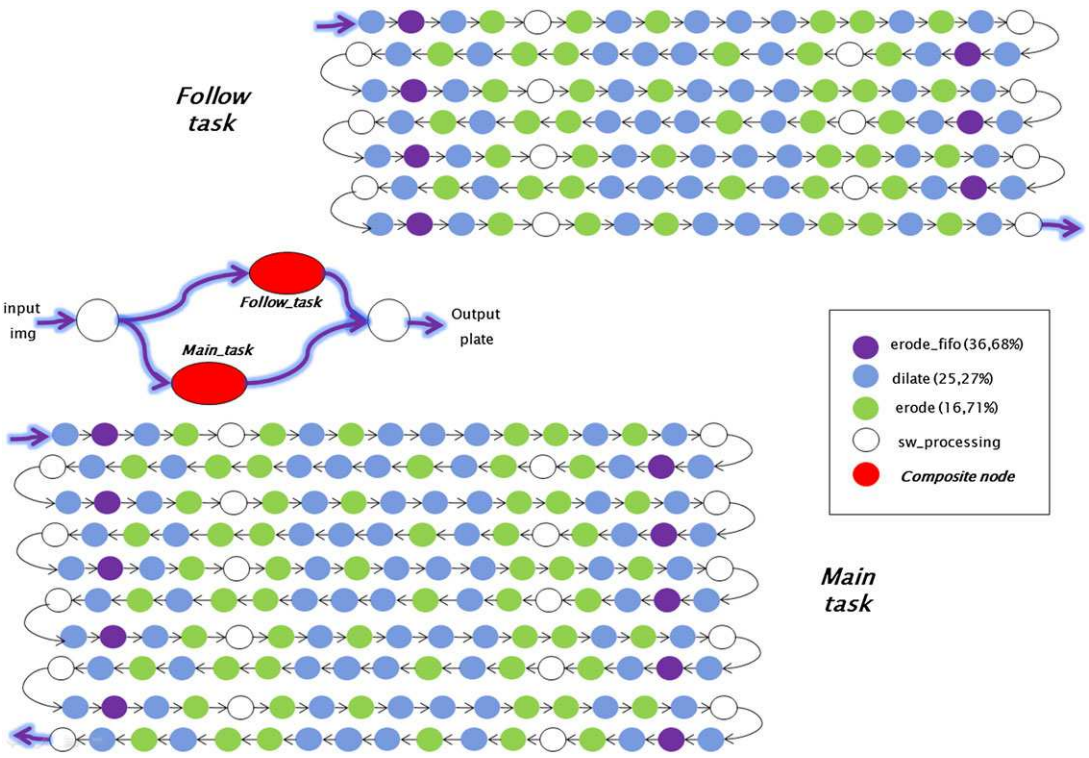


FIGURE 3 Task flow graph of basic LPR processing (1-lane license plate recognition) [Colour figure can be viewed at wileyonlinelibrary.com]

quality of detection and recognition. A sequence of operations (*dilate*, *erode_fifo*, *erode*, *binarize*, and *preprocessing*) is applied 7 times for the detection area and 10 times for the recognition area, as depicted in the task flow graph of Figure 3. This graph, which is composed of 291 nodes, specifies basic LPR process for 1 lane and can be very easily replicated to process multiple tracks using the multiapplication feature of FoRTReSS (section 3.2.3).

In the enhanced detection region, optical character recognition is used to identify alphanumeric characters, with a multilayer artificial neural network.[35] Aside from the learning phase, application profiling shows that the neural network recognition phase can be neglected in comparison with the very computationally intensive morphological operations, and so we will not address multilayer artificial neural network processing acceleration here. More specifically, it turns out that 3 morphological functions account for 79% of the total execution time: *erode_fifo* (36.68%), *dilate* (25.27%), and *erode* (16.71%). Therefore, the maximum theoretical speedup for the whole LPR process is 4.7×, which combines with the energy gains from any hardware implementations of the *erode_fifo*, *dilate*, and *erode* functions and points to significant overall gains in energy efficiency. This energy efficiency is reported below as an EDP reduction compared with software execution (or speedup × energy gain).

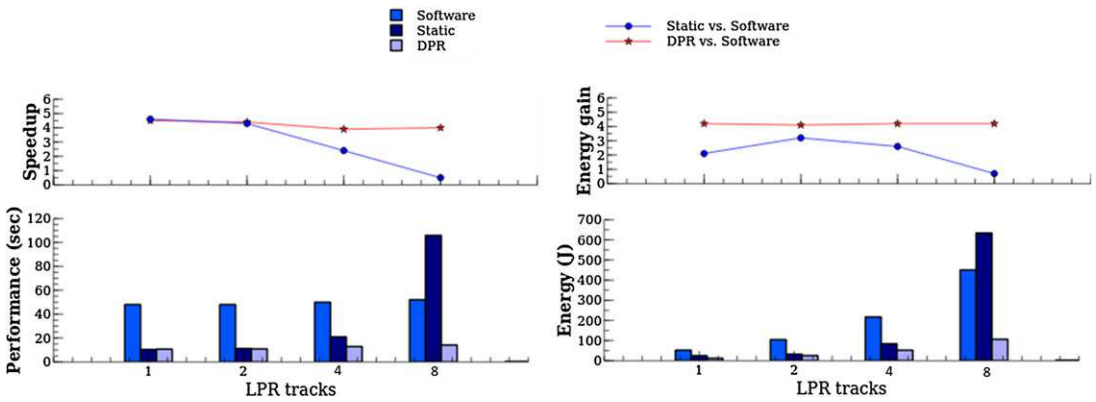


FIGURE 4 Speedup and energy gain of static and dynamic and partial reconfiguration (DPR) acceleration against software execution for XC6VLX240T device. LPR, license plate recognition [Colour figure can be viewed at wileyonlinelibrary.com]

5.2 | Deployment on Virtex-6/MicroBlaze

In this deployment analysis, we report and discuss the performance, energy, and efficiency gains of the LPR application in various configurations (1-, 2-, 4-, and 8-lane LPR). First, we consider a Xilinx Virtex-6 FPGA with MicroBlaze soft IP cores. All necessary power/performance characterization of hardware and software tasks were derived from implementation, execution, and measurements on an ML605 evaluation board, as depicted in section 3.2.4. Each reference software solution (one for each LPR configuration) corresponds to the most energy-efficient multicore execution, ie, the one able to achieve maximum performance with the parallelism of the configuration (ie, 2 cores for 1-lane LPR, 4 cores for 2-lane LPR, 8 cores for 4-lane LPR, and 16 cores for 8-lane LPR).

The lower part of Figure 4 shows stable software performance while scaling application configurations with the number of cores (ie, 1-lane LPR/2 cores and 2-lane LPR/4 cores). Logically, software energy use doubles with each successive LPR configuration (requiring double the number of cores), while DPR solutions always perform better with half the number of cores. For 8-lane LPR, for example, DPR acceleration (8 cores + 6 RRs) achieves much better results than software (16 cores) with, respectively, 4× better performance and 4.2× less energy (top part of Figure 4). It is worth noting that both the speedups and the energy gains of DPR compared with software execution remain very close across all 4 LPR configurations and that energy efficiency (ie, the EDP) improved up to a factor of 19 (4.5× speedup and 4.2× energy gain) for 1-lane LPR.

In the various lane configurations we have considered, static solutions combine a minimum number of cores with 3 accelerators (1 for *dilate*, 1 for *erode_fifo*, and 1 for *erode*). We intentionally limited the number of static accelerators to 3—whatever the LPR configuration—because otherwise the FPGA would rapidly run out of space. This reflects in more performance variability and performs generally worse than DPR. More precisely, the gap is widening as LPR configuration (thus parallelism) grows. For 1-lane LPR, for example, DPR is only 2.6% slower than static acceleration because it is affected little by having less hardware parallelism (1 RR vs 3 static accelerators) and reconfiguration latencies. However, this decrease of logic resources results in 50.3% less energy consumption. For 8-lane LPR, DPR benefits from optimal processing parallelism (8 cores + 6 RRs) to produce a speedup of 8.1× and a 6× energy gain over static acceleration (8 cores + 3 accelerators). A static solution is inefficient in this case, even less than software execution (16 cores), as it is strongly disadvantaged by only 3 accelerators to process 8 LPR lanes in parallel. In the end, static acceleration has improved the energy efficiency from 13.9× (4.3× speedup and 3.2× energy gain) to 0.3× (0.5× speedup and 0.7× energy gain) against reference software executions, while DPR improved from 16.1× (3.9× speedup and 4.2× energy gain) to 19× (4.5× speedup and 4.2× energy gain).

5.3 | Deployment on Zynq-7000

We address the same LPR benchmark and configurations considering now 2 Zynq-7000 platforms based on XC7Z020 and XC7Z045 devices (respectively, 85- and 350-K logic cells) with ARM Cortex-A9 hard IP cores. All necessary power/performance characterization was derived from implementation, execution, and measurements on a ZC702 evaluation board. In the subsequent analysis and exploration, we assume the original dual-core capability can be extended up to 16 Cortex-A9 cores to better cope with the processing requirements of the defined LPR configurations.

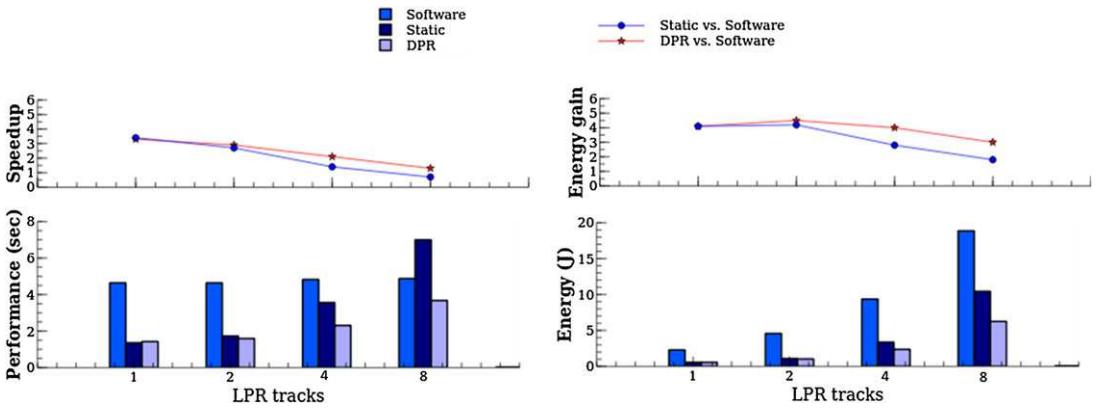


FIGURE 5 Speedup and energy gain of static and dynamic and partial reconfiguration (DPR) acceleration against software execution for the XC7Z020 device. LPR, license plate recognition [Colour figure can be viewed at wileyonlinelibrary.com]

5.3.1 | XC7Z020

We address first the XC7Z020 device of a ZC702 evaluation board, which served for the purpose of hardware/software performance and power characterization. Similar trends can be observed in the results of Figure 5 with regards to previous XC6VLX240T FPGA (Figure 4). Dynamic and partial reconfiguration is still more efficient than static solutions, but the difference is smaller. In addition, DPR provides less benefit with this device when parallelism grows, now with 1.3× speedup and 3× energy gain over software execution (8-lane LPR). The reason relates clearly to the size of the device, which does not allow the definition of more than 3 regions, therefore limiting the maximum gains closer to those of static acceleration (3 accelerators) for high parallelism levels (4- and 8-lane LPR). This limitation in size is also the reason for the decline in speedup/energy gain in these configurations.

However, energy efficiency improvements over software execution are still significant for lower parallelism levels (1- and 2-lane LPR) with, respectively, 13.5× (3.3× speedup and 4.1× energy gain) and 13× (2.9× speedup and 4.5× energy gain). These gains are below the results of Virtex-6 (19×) mainly because MicroBlaze is much slower compared with Cortex-A9 cores, leading to greater hardware versus software acceleration values. The best DPR performance for 2-lane LPR, for instance, is 1.6 seconds compared with 11 seconds for Virtex-6/MicroBlaze. Besides the undeniable improvement of both DPR and static acceleration, this example shows a moderate advantage in using DPR compared with static acceleration, because the potential for processing several hardware functions in parallel for highly parallel configurations exceeds the device capability (3 RRs max). The next study on a larger device will allow to further investigate this.

5.3.2 | XC7Z045

For this device, the energy efficiency of DPR improves from 9.3× (3.0× speedup and 3.1× energy gain) to 11.7× (3.5× speedup to 3.4× energy gain) compared with software execution (Figure 6). The increase in programmable logic resources improves the parallel hardware processing potential with up to 11 RRs (against 3 for the XC7Z020), which raises the speedup around 3× to 3.5×. Speedups and energy gains of DPR over software execution now remain at similar levels across all 4 LPR configurations. Overall performance is stable at around 1.5 seconds, whereas it ranged from 1.4 to 3.7 seconds for the XC7Z020, meaning an ideal performance scaling up to the 8-lane configuration for the XC7Z045.

In comparison, again there are more variations for static solutions with energy efficiency varying from 9.6× (3.4× speedup and 2.8× energy gain) to 1.1× (0.7× speedup and 1.6× energy gain) improvement over software execution. The static speedup follows the same declining trend coming from the limitation to 3 static accelerators. In a high-parallelism solutions (4- and 8-lane LPR), DPR is 3× and 10× more efficient than static acceleration, illustrating the excellent job done by the scheduler in exploiting RRs and cores efficiently (Section 4).

There is an optimal set of RRs for the application/device, which becomes more significant as the parallelism grows. If we focus on 4-lane LPR, for example, Table 2 reports the set of RRs explored by RecoSim from the possible function implementations (section 3.2.4) and from FPGA layout knowledge, with the corresponding performance (*Tex*), energy (*En*), speedup (*Spu*), energy gain (*En.gain*), and energy efficiency (*Spu*×*En.gain*) improvement over software execution.

The simulator defines and considers between 1 and 11 regions to map the application on the XC7Z045. A set of 8 RRs provides the best energy efficiency with an overall 11.7× improvement (3.5× speedup and 3.4× energy gain). Comparison with the XC6VLX240T, for which the best solution is achieved with only 3 RRs, points to the fact that the best RR

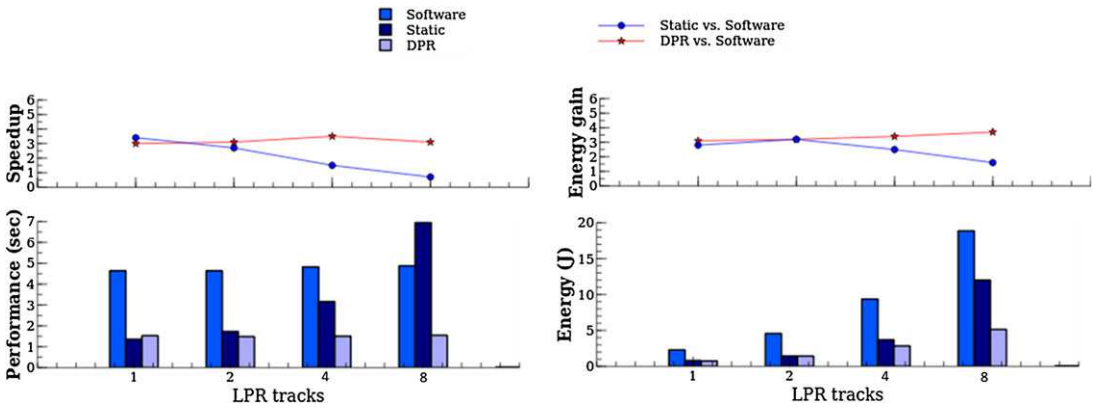


FIGURE 6 Speedup and energy gain of static and dynamic and partial reconfiguration (DPR) acceleration against software execution for the XC7Z045 device. LPR, license plate recognition [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 2 Impact of different RR configurations for 4-lane LPR (XC7Z045, XC6VLX240T)

	XC7Z045					XC6VLX240T				
	Tex(sec)	En(J)	Spu	En.gain	Spu×En.gain	Tex(sec)	En(J)	Spu	En.gain	Spu×En.gain
8*cores*	4.8	9.4	1	1	1	49.9	217	1	1	1
4*cores_1RR*	5.5	4.6	0.9	2	1.8	28.7	78.6	1.7	2.8	4.8
4*cores_2RRs*	2.9	3.1	1.6	3	4.9	15.5	52.9	3.2	4.1	13.2
4*cores_3RRs*	2.3	2.9	2.1	3.2	6.6	12.9	52.2	3.9	4.2	16.1
4*cores_4RRs*	1.9	2.7	2.5	3.4	8.6	12.4	57.9	4	3.7	15
4*cores_5RRs*	1.7	2.7	2.8	3.4	9.8	12.3	65.4	4	3.3	13.4
4*cores_6RRs*	1.5	2.6	3.1	3.6	11.2	11.2	66	4.5	3.3	14.7
4*cores_7RRs*	1.5	2.8	3.2	3.3	10.6	11.5	75.3	4.3	2.9	12.5
4*cores_8RRs*	1.4	2.8	3.5	3.4	11.7	11.1	78.7	4.5	2.8	12.4
4*cores_9RRs*	1.4	2.9	3.5	3.2	11.2	11	84.4	4.5	2.7	11.6
4*cores_10RRs*	1.4	3.1	3.5	3	10.6	…	…	…	…	…
4*cores_11RRs*	1.4	3.3	3.5	2.9	9.9	…	…	…	…	…

Abbreviations: LPR, license plate recognition; RR, reconfigurable region.

configuration is likely to be different from one device to another. Finding an optimal RR partitioning for the application is therefore very important, as scheduling (which largely depends on partitioning) is an essential condition of DPR efficiency, for which we will strive to provide an effective measure in the next section.

5.4 | Overall efficiency analysis

The main outcomes of these simulations are firstly that a variety of conditions influence proper exploitation of the complex mapping heterogeneity and the amount of DPR benefit. Not all kinds of processing suit DPR acceleration. A significant share of hardware functions is the primary necessary (but not sufficient) condition for significant efficiency (over 75% at full application level), and the parallelism of tasks (ie, the ability to run concurrently) will greatly influence the quality of the results. If the benefits of DPR and static hardware against software execution are a given in most cases, the existence of a potential gain of DPR over static is less trivial. To really improve processing efficiency, the platform needs to provide adequate resources to fully exploit the parallelism in the application. From the LPR case study, which has been investigated in detail, DPR can reach up to more than 3 times better energy efficiency than static acceleration under fair parallelism conditions (eg, 4-track LPR).

The essential conditions to achieve this are scheduling and FPGA partitioning. To assess the extent of these aspects, we have set up simulations where a standard real-time scheduler (EDF) is used in place of the EAHS of Section 4 (Figure 7). With 4-lane LPR, results show a drop of 63.4% in energy efficiency for the XC7Z045/Cortex-A9 and 14.6% for the XC6VLX240T/MicroBlaze. These comparisons provide an idea of the potential impact of scheduling and partitioning on which energy efficiency strongly depends. This case study therefore points out the importance of a methodic holistic exploration and mapping methodology to properly exploit the potential of DPR. The benefits when compared with

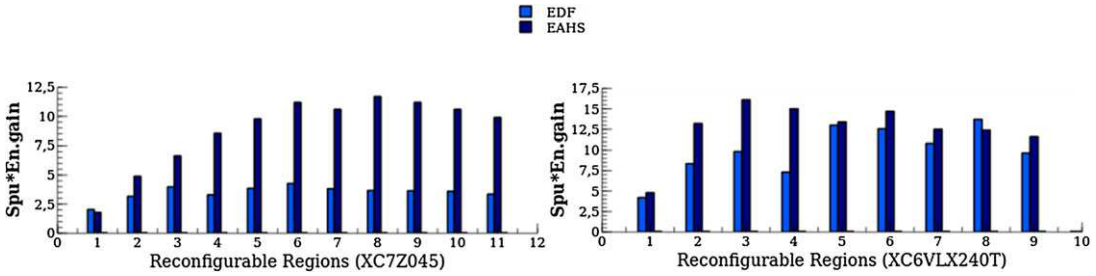


FIGURE 7 Scheduler efficiency on 4-lane license plate recognition (left XC7Z045, right XC6VLX240T). EAHS, energy-aware heterogeneous scheduler; EDF, earliest deadline first [Colour figure can be viewed at wileyonlinelibrary.com]

software execution are a given in most cases, but the real issue is whether DPR is worth the design effort compared with classical (static) FPGA acceleration. With poor static (system sizing and region partitioning) and/or dynamic (scheduling and mapping) choices, the real benefit is likely to be missed because a great deal of static and dynamic variables comes into play as far as energy is concerned. For example, a crucial point is to identify adequate trade-offs in technology (device and IP cores) and partitioning (number, size, and and resources for the programmable logic) because a small programmable logic area will not allow the application parallelism to be fully exploited, whereas a large area will draw a lot of power due to the relative high FPGA static consumption. For the same application, this can lead to significant variations in results from one platform to another. Considering 1-lane LPR, for example, DPR is twice as energy efficient than static on the XC6VLX240T/MicroBlaze, owing to the reduction in size of programmable logic resources (1 RR versus 3 accelerators) and the high static power coming with this large device. On Zynq-7000, DPR and static have a very similar energy efficiency, which probably makes DPR too complex to implement given the benefit.

6 | CONCLUSIONS

This study makes a number of contributions to power analysis designed to make the problems involved in achieving higher energy efficiency more intelligible. Firstly, on better exploiting heterogeneity in complex multiprocessor systems, the methodology it defines brings significant energy efficiency improvements compared with homogeneous multiprocessor execution: 16.1× to 19× (Virtex-6/MicroBlaze), 13.5× to 4× (XC7Z020/Cortex-A9), and 11.7× to 9.3× (XC7Z045/Cortex-A9). These results underline the benefits of a more methodical approach, improving interaction across 3 key areas: design space exploration, scheduling, and decision support. The relevance of this on a practical level relies on a realistic decision-support scheme common to exploration and scheduling, and an EAHS that is effective in getting the best out of heterogeneous resources and techniques.

On the more specific question of exploiting hardware acceleration, the contributions are essentially defined by the determination of the potential of DPR, both in terms of improvement when compared with multiprocessor execution as stated previously but also compared with static acceleration (1.1× to 2.6× for Virtex-6/MicroBlaze, 1× to 2.2× for XC7Z020/Cortex-A9, and 0.9× to 1.7× for XC7Z045/Cortex-A9). These results are driven by concrete power characterisation efforts of DPR, now making it easier to determine mapping solutions that can be markedly more efficient than static acceleration, and—just as importantly—to find out if and how dynamic reconfiguration can be used correctly.

Finally, regarding the general question of advances in low power research, existing approaches are broadly divided in 2 categories: those that build on classical approaches (with power as a secondary requirement after performance, or addressing different techniques separately) and those that explore new techniques and technologies (eg, process nodes and memories). This work uncovers another direction based on integrated and power-dedicated approaches, which is often overlooked. In light of the results, this smarter integration of techniques brings high hopes to extend energy efficiency beyond a factor of 10, which is a major challenge for many upcoming critically power-constrained application domains such as embedded systems (IoT, security, and artificial intelligence); telecommunications (5G); and high-performance computing (Exascale).

ACKNOWLEDGMENTS

This work was conducted under the BENEFIC project (CA505), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

ORCID

Sébastien Bilavarn http://orcid.org/0000-0002-7492-6936

REFERENCES

1. Leech C, Kazmierski TJ. Energy efficient multi-core processing. *ELECTRONICS*. 2014;18(1):3-10.
2. Ghiasi S, Keller T, Rawson F. Scheduling for heterogeneous processors in server systems. In: Proceedings of the 2Nd Conference on Computing Frontiers; 2005; New York, NY, USA. 199-210.
3. DeVuyst M, Kumar R, Tullsen DM. Exploiting unbalanced thread scheduling for energy and performance on a CMP of SMT processors. In: Proceedings of the 20th International Conference on Parallel and Distributed Processing; 2006; Rhodes Island, Greece. 140-140.
4. Snavely A, Tullsen DM. Symbiotic jobscheduling for a simultaneous mutlithreading processor. *SIGPLAN Not*. 2000;35(11):234-244.
5. Teodorescu R, Torrellas J. Variation-aware application scheduling and power management for chip multiprocessors. In: Proceedings of the 35th Annual International Symposium on Computer Architecture; 2008; New York, NY, USA. 363-374.
6. Winter JA, Albonesi DH, Shoemaker CA. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques; 2010; New York. 29-40.
7. Imes C, Hoffmann H. Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-isa heterogeneous multi-cores. *SIGBED Rev*. 2015;11(4):49-54.
8. Bower FA, Sorin DJ, Cox LP. The impact of dynamically heterogeneous multicore processors on thread scheduling. *IEEE Micro*. 2008;28(3):17-25.
9. Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management. *IEEE Trans Very Large Scale Integr VLSI Syst*. August 2002;8(3):299-316.
10. Singh P, Chinta V. Survey report on dynamic power management, Chicago, USA, University of Illinois, Chicago (ECE Department); 2008;26(58).
11. Bilavarn S, Khan JJ, Belleudy C, Bhatti MK. Effectiveness of power strategies for video applications: a practical study. *J Real-Time Image Process*. 2014;12(1):123-132.
12. Javaid H, Shafique M, Henkel J, Parameswaran S. System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia. In: 2011 IEEE/ACM International Conference on Computer-Aided Design, ICCAD; 2011; San Jose, California, USA. 616-623.
13. Chitlur N, Srinivasa G, Hahn S, et al. Quickia: exploring heterogeneous architectures on real prototypes. In: 18th IEEE International Symposium on High Performance Computer Architecture, HPCA; 2012; New Orleans, LA, USA. 433-440.
14. Bazargan K, Kastner R, Sarrafzadeh M. Fast template placement for reconfigurable computing systems. *IEEE Des Test*. 2000;17(1):68-83.
15. Hong C, Benkrid K, Iturbe X, Ebrahim A, Arslan T. Efficient on-chip task scheduler and allocator for reconfigurable operating systems. *Embedded Syst. Lett.* 2011;3(3):85-88.
16. Liu S, Pittman RN, Forin A, Gaudiot J-L. Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems. *ACM Trans Embedded Comput Syst*. April 2013;12(3):72:1-72:21.
17. Bonamy R, Bilavarn S, Chillet D, Sentieys O. Power consumption models for the use of dynamic and partial reconfiguration. *Microprocess Microsyst*. 2014;38(8, Part B):860-872.
18. Tuan T, Lai B. Leakage power analysis of a 90nm FPGA. In: IEEE Custom Integrated Circuits Conference, 2003; 2003; San Jose, CA. 433-440.
19. Sterpone L, Carro L, Matos D, Wong S, Fakhar F. A new reconfigurable clock-gating technique for low power SRAM-based FPGAs. In: Design, Automation & Test in Europe (DATE), 2011; 2011; Grenoble, France. 1-6.
20. Wang Q, Gupta S, Anderson JH. Clock power reduction for virtex-5 FPGAs. In: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays; 2009; Monterey, CA. 13-22.
21. Paulsson K, Hubner M, Becker J. Dynamic power optimization by exploiting self-reconfiguration in Xilinx spartan 3-based systems. *Microprocess Microsyst*. 2009;33(1):46-52.
22. Kathail V, Hwang J, Sun W, Chobe Y, Shui T, Carrillo J. SDSoC: a higher-level programming environment for zynq SoC and Ultrascale+ MPSoC. In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays; 2016; Monterey, CA, USA. 4-4.
23. Agne A, Happe M, Keller A, et al. ReconOS: an operating system approach for reconfigurable computing. *IEEE Micro*. 2014;34(1):60-71.
24. Liang T, Feng L, Sinha S, Zhang W. PAAS: a system level simulator for heterogeneous computing architectures. In: 27th International Conference on Field Programmable Logic and Applications; 2017; Ghent, Belgium. 1-8.
25. Ding H, Huang M. PolyPC: polymorphic parallel computing framework on embedded reconfigurable system. In: 27th International Conference on Field Programmable Logic and Applications; 2017; Ghent, Belgium. 1-8.
26. Schwiegelshohn F, Hubner M. An application scenario for dynamically reconfigurable FPGAs. In: Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip; 2014; Montpellier, France. https://ieeexplore.ieee.org/document/6861352/
27. Weller D, Oboril F, Lukarski D, Becker J, Tahoori M. Energy efficient scientific computing on fpgas using opencl. In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays; 2017; New York, NY. 247-256.

28. Nguyen T, Gurumani S, Rupnow K, Chen D. FCUDA-SoC: platform integration for field-programmable SoC with the CUDA-to-FPGA compiler. In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays; 2016; Monterey, CA, USA. 5-14.
29. Ozkan MA, Reiche O, Hannig F, Teich J. FPGA-based accelerator design from a domain-specific language. In: Proceedings of the 26th International Conference on Field-Programmable Logic and Applications; 2016; Lausanne, Switzerland. 1-9.
30. Wipliez M, Roquier G, Nezan J-F. Software code generation for the RVC-CAL language. *J Signal Process Syst*. 2011;63(2):203-213.
31. Duhem F, Muller F, Aubry W, Gal BL, Négru D, Lorenzini P. Design space exploration for partially reconfigurable architectures in real-time systems. *J Syst Archit*. 2013;59(8):571-581.
32. Duhem F, Muller F, Bonamy R, Bilavarn S. Fortress: a flow for design space exploration of partially reconfigurable systems. *Design Autom Embedded Syst*. 2015;19(3):301-326.
33. Bonamy R, Bilavarn S, Chillet D, Sentieys O. Power modeling and exploration of dynamic and partially reconfigurable systems. *J Low Power Electron*. 2016;12(3):172-185.
34. Damak T, Werda I, Bilavarn S, Masmoudi N. Fast prototyping H.264 Deblocking filter using ESL tools. *Trans Syst, Signals Devices, Issues Commun Signal Process*. 2013;8(3):345-362.
35. Steffen N. Fast artificial neural network library 2013. http://leenissen.dk/fann/wp/

How to cite this article: Bonamy R, Bilavarn S, Muller F, et al. Energy efficient mapping on manycore with dynamic and partial reconfiguration: Application to a smart camera. *Int J Circ Theor Appl*. 2018;1–15. https://doi.org/10.1002/cta.2508

Microprocessors and Microsystems 53 (2017) 68–80

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Efficiency modeling and exploration of 64-bit ARM compute nodes for exascale

J. Wanza Weloli[a,*], S. Bilavarn[b], M. De Vries[a], S. Derradji[a], C. Belleudy[b]

[a] Bull atos technologies, Les Clayes Sous Bois, France
[b] LEAT, CNRS UMR7248, University of Nice Sophia Antipolis, France

ARTICLE INFO

Article history:
Received 28 December 2016
Revised 23 May 2017
Accepted 27 June 2017
Available online 12 July 2017

ABSTRACT

This paper investigates the use of 64-bit ARM cores to improve the processing efficiency of upcoming HPC systems. It describes a set of available tools, models and platforms, and their combination in an efficient methodology for the design space exploration of large manycore computing clusters. Experimentations and results using representative benchmarks allow to set an exploration approach to evaluate essential design options at micro-architectural level while scaling with a large number of cores. We then apply this methodology to examine the validity of SoC partitioning as an alternative to using large SoC designs based on coherent multi-SoC models and the proposed SoC Coherent Interconnect (SCI).

1. Introduction and context

The performance of supercomputers has traditionally grown continuously with the advances of Moore's law and parallel processing, while energy efficiency could be considered as a secondary problem. But it quickly became clear that power consumption was the dominant term in the scaling challenges to reach the next level. It is roughly considered that 20 times energy efficiency improvement is required for exascale computing (10^{18} FLOPS) to cope with the tremendous electrical power and cost incurred by such computational capacity. The idea of using concepts borrowed from embedded technologies has naturally emerged to address this. First prototypes based on large numbers of low power manycore microprocessors (possibly millions of cores) instead of fast complex cores started to be investigated, putting forward a number of proposals for improvement at node level architecture to meet HPC demands.

These works covered a variety of 32-bit RISC cores ranging from ARM Cortex-A8 [1] and Cortex-A9 [2–4] to more recent Cortex-A15 and Cortex-A7 cores [5]. Padoin et al. [2], Rajovic et al. [3] and Ou et al. [4] addressed, for example, dual and quad core systems based on ARM Cortex-A9 cores. The different results indicated various processing limitations to meet HPC performance requirements, in terms of double precision floating point arithmetic, 32-bit memory controllers (limiting the address space), ECC memory (e.g. for scientific and financial computing), and fast interconnect (communication intensive applications). Laurenzano et al. [6] and Maqbool et al. [7] additionally confirmed that the variability in performance and energy could largely be attributed to floating point and SIMD computations, and interactions with the memory subsystem. Other works, which addressed explicit comparison against x86 based systems, also pointed out the need for higher levels of performance to meet HPC demands. Ou et al. [4] conclude that the cost advantage of ARM clusters diminishes progressively for computation-intensive applications (i.e. dynamic Web server application, video transcoding), and other works like Blem et al. [8] conducted on ARM Cortex-A8, Cortex-A9, Intel Sandybridge, and Intel Atom confirmed that ARM and x86 could achieve similar energy efficiency, depending on the suitability of a workload to the microarchitectural features at core level.

Of the works addressing the feasibility of ARM SoCs based HPC systems, efforts focused widely on single-node performance using microbenchmarks. Less studies considered large-scale systems exceeding a few cores even though multi-node cluster performance is an essential aspect of future Exascale systems [11]. Considering further that new generations of cores such as the *ARMv8-A* ISA support features to improve specifically on HPC workloads (64-bit address space, 64-bit arithmetic, high speed interconnects, fast memory hierarchies), this work is one of the first to describe outcomes of research opened up with these perspectives. Therefore we provide an evaluation of available tools, models and platforms able to set the foundations of a methodical system level exploration approach for HPC applications scaling up to 128 64-bit ARM cores

* Corresponding author.
E-mail addresses: joel.wanza-weloli@atos.net (J. Wanza Weloli), sebastien.bilavarn@unice.fr (S. Bilavarn), maarten.de-vries@atos.net (M. De Vries), said.derradji@atos.net (S. Derradji), cecile.belleudy@unice.fr (C. Belleudy).

http://dx.doi.org/10.1016/j.micpro.2017.06.019
0141-9331/

and show how it was used to examine the relevance of SoC partitioning to limit complexity, cost and power consumption.

This work is carried out under a long-term European effort called Mont-Blanc. Mont-Blanc is one of the many H2020+ projects funded by the European Commission to support Exascale research. Started in October 2011, phase 1 (Mont-Blanc 1) investigated the adoption of embedded mobile processors in a HPC system [9] and led to the establishment of a prototype based on Exynos 5 compute cards (ARM 32-bit cores) [10]. The goal of the subsequent project Mont-Blanc 2 was to develop a full software ecosystem along with architecture exploration in a joint co-design (hardware / software) methodology. Finally the last phase and ongoing Mont Blanc 3 project (started in October 2015) aims to exploit the strong knowledge developed to produce the future high-end HPC platform able to realize the level of performance and energy ratio required for exascale class applications. The work presented in this paper is in the background of Mont-Blanc 2 and Mont-Blanc 3 projects. It addresses advanced HPC compute nodes upon the *ARMv8-A* ISA with special attention on the overall on-chip memory consistency, scalability, cost and energy efficiency. This program prefigures features of the upcoming Bull sequana platform based on Cavium ThunderX2 ARMv8-A processors.

The outline of the paper is the following. We present in Section 2 different modeling and simulation tools suited to the analysis of HPC systems with lately available ARM 64-bit based platforms (ARM Juno, AMD Seattle, AppliedMicro X-Gene). Using the defined methodology, Section 3 explores in detail a set of architectural propositions aiming at reducing SoC and cache coherence complexities and examines their impacts from application parallelism perspective in different programming models. Section 4 summarizes the main conclusions from the various results and exposes next directions of research.

2. Methodology

In the following, we investigate the use of available tools, models and platforms to define an exploration approach matching our needs. We then characterize a set of relevant HPC benchmarks on different platform configurations to verify that we meet all conditions for exploration effectiveness given a set of architectural requirements to consider (performance, memory architecture, interconnect, scalability).

2.1. Modeling

2.1.1. ARMv8-A platforms

The Juno ARM Development Platform is one of the first available development platforms for *ARMv8-A*. Our interest goes mainly for the performance cluster (dual Cortex-A57 of the MPCore big.LITTLE processor), the Cache Coherent Interconnect (CCI-400) and the DDR3-1600 dual channel memory controller. In this characterisation effort of real hardware, we also addressed the use of an AMD Seattle board based on four clusters of two Cortex-A57 cores with AMD Coherent Interconnect at 2 GHz and two DDR4-3733 memory controllers. We additionally considered an AppliedMicro (APM) X-Gene1 in which the SoC includes four clusters of the two 64-bit cores running at 2.4 GHz, APM coherent network Interconnect and DDR3 controller (16GB).

We therefore employ virtual platforms to possibly extend exploration perspectives to the support of large scalability (up to 128 cores) and use of upcoming 64-bit cores (e.g. Cortex-A72). ARM Fast Model virtual platforms (AFM) is the largest platform that can be configured in this regard, using only ARM available fast model IPs up to 48 Cortex-A57/A72. The limitation to 48 cores in current releases comes from the Cache Coherent Network CCN-512 interconnect supporting a maximum of 12 coherent clusters while each cluster can contain up to four cores. We finally exploit the Virtual Processing Unit (VPU) and Task Modeling framework which is part of a Synopsys methodology for large scale SoC, coherent interconnect and memory sub-system exploration. This framework provides further abstraction of multicore SoC platforms in SystemC/TLM with an interactive Traffic Generation and Cycle-Accurate TLM Interconnect Models based on software traces of previous ARM Fast Models which can be used to address higher levels of scalability.

Table 1
STREAM kernels.

Functions	Operations
Copy	$a(i) = b(i)$
Scale	$a(i) = q * b(i)$
Sum	$a(i) = b(i) + c(i)$
Triad	$a(i) = b(i) + q * c(i)$
Mean	$(Copy + Scale + Sum + Triad)/4$

2.1.2. Simulation tools

AFM and VPU platforms are considered with Synopsys Platform Architect for performance and power analysis. Fig. 1 provides an overview of the simulation methodology which is based first on a combination of real and virtual platform executions when the system has only a few nodes. Task graphs can then be produced from the results and used to further simulate up to 128 nodes using the VPU platform. In addition to benefiting from both virtual platforms in the same design environment, this framework provides a review on existing ARM models and a mean to improve architectural analysis with either type of complementary platforms AFM and VPU. In the following benchmarking study (Section 2.2), VPU and AFM platforms are configured with features corresponding to the aforementioned real boards in order to verify the correlation in terms of performance (GFLOPS, Cycle Per Instruction, etc.) and memory hierarchy (cache statistics, memory bandwidth).

2.1.3. Applications

Floating point benchmarking is based on SGEMM (Single-precision General Matrix multiply), DGEMM (Double-precision General Matrix multiply [12]) and HPL (High Performance Linpack from Top500 [13]). SGEMM and DGEMM measure the floating point rate of execution of respectively single precision and double precision real matrix-matrix multiplication while HPL measures the floating point rate of execution for solving a linear system of equations. These benchmarks are commonly used in practice to help characterize system performances in terms of floating point operations for HPC systems. In addition, we use the STREAM benchmark [14] to address more memory related aspects (cache stimulation, memory bandwidth) with four types of different kernels reported in Table 1.

2.2. Benchmarking

We analyze the relevance of models and tools against real platforms (up to eight cores), firstly in terms of floating point processing performance and efficiency. We then focus on the memory and cache architecture, analyze the conditions of validity of the results, and extend the methodology to support robust analysis for a larger number of cores (possibly up to 128).

2.2.1. Performance models

We consider two metrics to evaluate the processing efficiency. The first one is based on floating point operations per second (GFLOPS) which is reflective of the processing power for HPC workloads, and the second is the FLOPS efficiency expressing the ratio of actual versus theoretical FLOPS supported by the system.

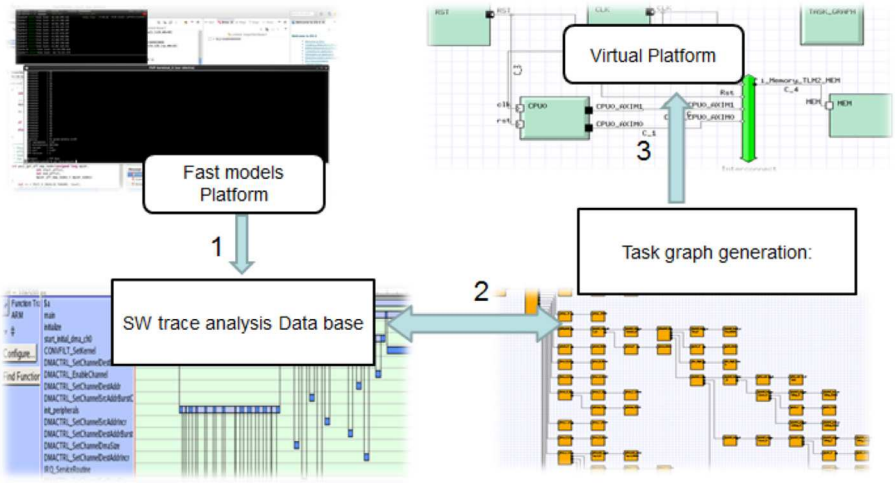


Fig. 1. Task graph based methodology.

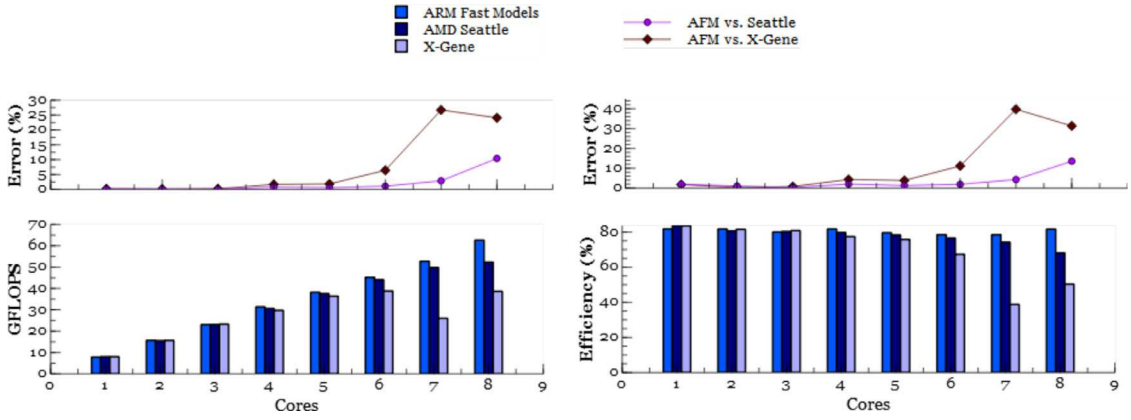


Fig. 2. Performance and efficiency of ARM fast models vs. AMD Seattle and X-Gene platforms on SGEMM benchmark.

ARM Fast models are used as one objective is to examine the organization of efficient clusters, which can well benefit here from an ARM CoreLink CCN-512 interconnect model supporting up to twelve clusters of four A57 cores. Two real platforms (AMD Seattle, AppliedMicro X-Gene) supporting both four clusters of two *ARMv8-A* 64-bit cores with their built-in interconnect are thus used to compare the models with reality as reported in Fig. 2. These two platforms are thus modeled with the afore-mentioned Synopsys virtual platform using A57 AFM models for all cores and the CCN-512 interconnect model in the absence of interconnect models for the AMD and AppliedMicro platforms.

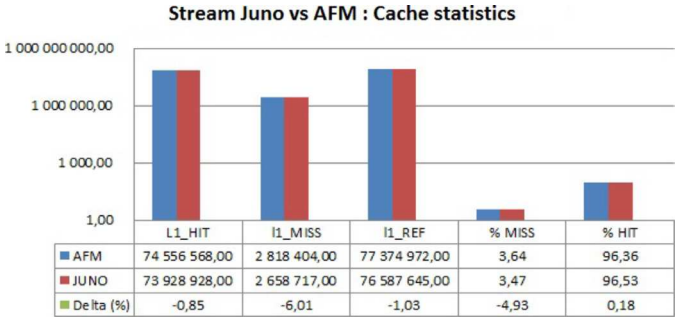
In the following performance measurements, each simulation/execution was reproduced ten times to ensure that variations were negligible. The results indicate an average GFLOPS and efficiency accuracy of respectively 1.1% and 2.5% up to six cores. Then, the disparity of interconnects on the different platforms reflects in deviations that are highly sensitive with the growing number of cores. The results show therefore that the global accuracy of AFM based virtual platforms is very good with less than 1.8% in average using ARM Fast models, but greatly dependent on the relevance of the interconnect model in configurations exceeding six cores. However, simulation times further limit the use of this platform in complex configurations (twenty two cores requires two days on a desktop workstation). This model can therefore be useful to explore core level, interconnect and intra-cluster configurations. Further scalability will thus be addressed in another way as depicted in Section 2.2.3 and the memory hierarchy is addressed in the following section.

2.2.2. Memory and cache architecture

The goal here is to extend previous approach to allow the robust analysis of memory hierarchy performance (execution time and throughput) and performance scalability (considering the possible impacts of cache). Given previous outcome, these metrics and more especially the cache statistics only relates to the cluster level (L1 and L2 cache) and more importantly to the L1 cache which has the most performance impact and should typically have a hit rate above 95% in real world applications. The Juno ARM platform gives access to advanced performance monitoring features of A57 and A53 cores. We can therefore configure and experiment with AFM platforms based on A57 cores to examine the precision of memory models against the Juno platform. The following thus describes simulations of a Cortex-A57 core running the STREAM benchmark where the results (Fig. 3) are plotted against the Juno A57 core in terms of cache statistics (L1 miss, L1 hit) with an average precision of 2.6%.

Fig. 4 confirms that the L1 cache statistics are more meaningful than L2, which is logical because L2 cache traffic comes essentially from L1 misses of the A57 core used here. We can also observe that the cache miss correlation error does not exceed 6%.

It is not possible to compare AFM platforms against real numbers in configurations exceeding two cores since only two A57 are available on the Juno platform. Anyway as pointed out previously, growing deviation is likely to occur due to the difference between interconnects. However, on the 1xA57 reference configuration used to run the five STREAM kernels, AFM provides pinpoint accuracy with 1,9% on memory bandwidth estimation on average



	L1_HIT	I1_MISS	I1_REF	% MISS	% HIT
AFM	74 556 568,00	2 818 404,00	77 374 972,00	3,64	96,36
JUNO	73 928 928,00	2 658 717,00	76 587 645,00	3,47	96,53
Delta (%)	-0,85	-6,01	-1,03	-4,93	0,18

Fig. 3. L1 cache statistics.

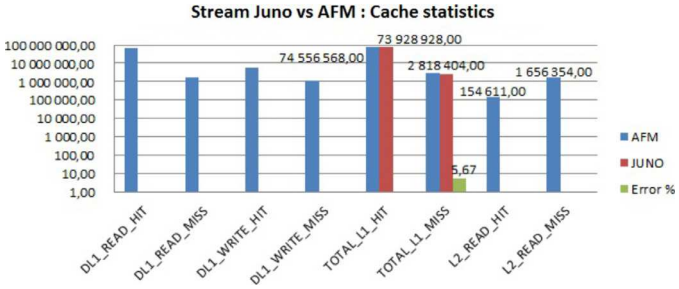
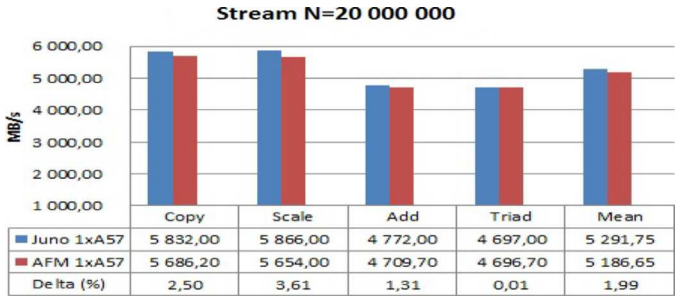


Fig. 4. L1 vs. L2 cache statistics.



	Copy	Scale	Add	Triad	Mean
Juno 1xA57	5 832,00	5 866,00	4 772,00	4 697,00	5 291,75
AFM 1xA57	5 686,20	5 654,00	4 709,70	4 696,70	5 186,65
Delta (%)	2,50	3,61	1,31	0,01	1,99

Fig. 5. Memory bandwidth.

(Fig. 5). Therefore, this setup can be profitably used to identify improvement opportunities at node/cluster level concerning the effect of different cache configurations (size, policy, topology) on system performances. STREAM parameter N is the length of the arrays used in the source code test to cover a large enough portion of the memory. There are three double precision (64 bits = 8 bytes) vectors A, B and C in the test, so the size required in the memory is 3*8*N bytes (457 MB when N = 20,000,000).

2.2.3. Large scale simulation

The scope of this part is to extend the analysis at a larger scale with available ARM fast models IPs. Due to CCN-512 limitations, we target configurations up to 48 cores in the following simulations (Figs. 6 and 7). Fig. 6 reports performance metrics in terms of execution time, number of floating point operations per second and efficiency while increasing the number of threads. The efficiency is defined as the ratio between the measured and the theoretical maximum performance. X-axis represents the number of threads and the values on the y-axis relies on a common scale for performance (seconds and GFLOPS) and efficiency (%).

Inspecting the time and GFLOPS traces, system performance increases until the 22nd thread and then drops, indicating a peak for a 8192 × 8192 configuration (involving 1.5GB of RAM). This means that beyond this peak value, increasing the number of cores is useless for this benchmark configuration. As the parallelism grows, the distribution of workload reaches a point above which there is an heterogeneity of computations caused by desynchronization between threads due to an under-utilization of some cores. This is also the reason for multiple non deterministic variations we can observe after this point.

A larger SGEMM matrix size would be required to reach the peak performance at the 48th thread, but we start to exceed here the limits of AFM abstraction level leading to prohibitive simulation times (more than 2 weeks). The reduction of execution time is exponential as we process the same workload with increasing number of cores. As previously noted, there is a point where thread heterogeneity limits the efficiency of parallelization leading therefore to a performance threshold level.

Fig. 7 reports performance and efficiency analysis of the HPL benchmark using larger AFM platforms configured for 8, 16, 32 and 48 threads. FLOPS efficiency increases gradually with parameter N. Optimized ATLAS libraries (Automatically Tuned Linear Algebra Software) are used in a way to reach the peak performance for 48 cores. However, larger values of N are needed to prevent the system from being under-used as visible in the results. Again this has not been further investigated because of excessive simulation times, but in spite of this, different information can be exploited. SGEMM benchmark show for instance that using more than 22 cores is not relevant for this benchmark configuration (Fig. 6) or that more memory would be needed to exploit up to 48 cores for a more parallel version of this application. This means that the platform is undersized in terms of memory allocated per core to keep a high computing efficiency. This approach represents there-

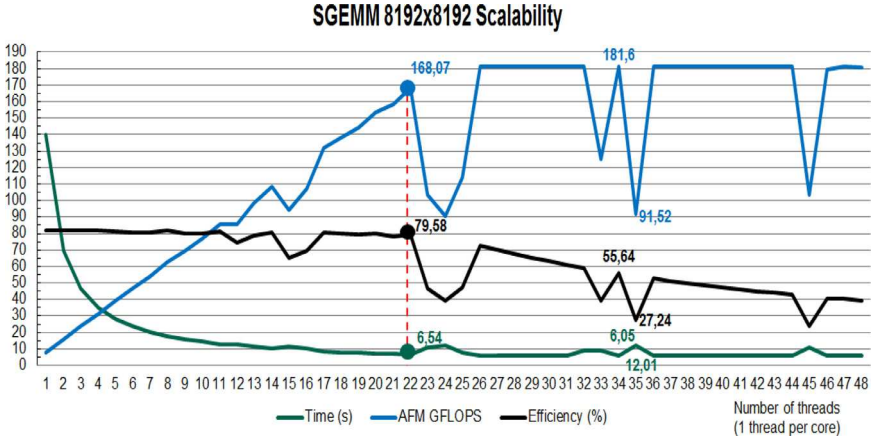
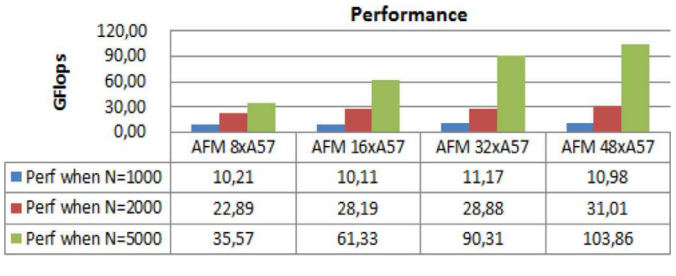
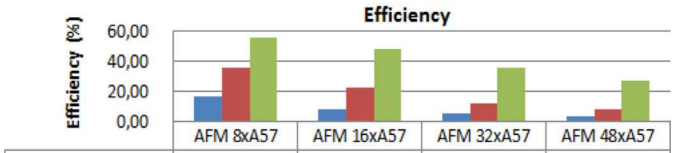


Fig. 6. Scalability on SGEMM benchmark.



	AFM 8xA57	AFM 16xA57	AFM 32xA57	AFM 48xA57
Perf when N=1000	10,21	10,11	11,17	10,98
Perf when N=2000	22,89	28,19	28,88	31,01
Perf when N=5000	35,57	61,33	90,31	103,86



	AFM 8xA57	AFM 16xA57	AFM 32xA57	AFM 48xA57
Eff % when N=1000	15,95	7,90	4,36	2,86
Eff % when N=2000	35,77	22,02	11,28	8,08
Eff % when N=5000	55,58	47,91	35,28	27,05

Fig. 7. Scalability on HPL benchmark with BLAS optimized libraries.

fore valuable feedback in terms of possible hardware and software co-design analysis to find a better balance of the system.

These results set the foundations for proper exploration and evaluation of architecture capabilities in terms of processing efficiency, memory hierarchy, interconnect, topology and scalability. Since the target platform is designed to take advantage of large *ARMv8-A* clusters, communication topology and memory system are key issues to address. In that respect, SoC partitioning becomes an attractive option to consider due to high development and production costs of large monolithic chips in the latest silicon technologies. Next architectural study extends therefore previous exploration approach (Platform Architect, 64-bit ARM cores and a specific interconnect) with necessary hardware requirements, especially regarding inter chip cache coherence support between compute nodes, in a way to study the impacts and efficiency conditions in different partitioning scenarios.

3. Architectural exploration

3.1. SoC and interconnect partitioning

This exploration study addresses the validity of SoC partitioning as an alternative to using large SoC designs. On the basis of a monolithic SoC design integrating 128 *ARMv8-A* cores, high bandwidth memory (HBM/HMC) and a Bull Exascale computing network interface controller [12], the idea is to evaluate coherent multi-SoC models (i.e. two SoCs / 64 cores, four SoCs / 32 cores) communicating through chip-to-chip ports and coherent proxies (top of Fig. 8). Considering the global architecture model (bottom of Fig. 8), a bottleneck lies in the cache coherence management because existing snoop based protocols do not scale with the large number of caches that are commonly found in HPC processing. Therefore we introduce a coherence extension of the SoC interconnect (SCI) required for this partitioning. This aims at reducing the complexity of the coherence protocol and additionally can significantly reduce the energy consumption from the interconnect as well as the tag lookups in the remote caches.

3.2. Coherent interconnect

3.2.1. Overview of coherency protocols

A suitable definition of the coherence is given in [15] as *single-writer-multiple-reader (SWMR)* invariant. That means that for any memory location space, at a given cycle time, there may only be one single writer or a number of cores that may read it. Consequently, implementing a cache coherence mechanism requires avoiding the case where two separated caches contain two different values for the same memory address at the same moment. So a cache coherence protocol addresses the way of maintaining con-

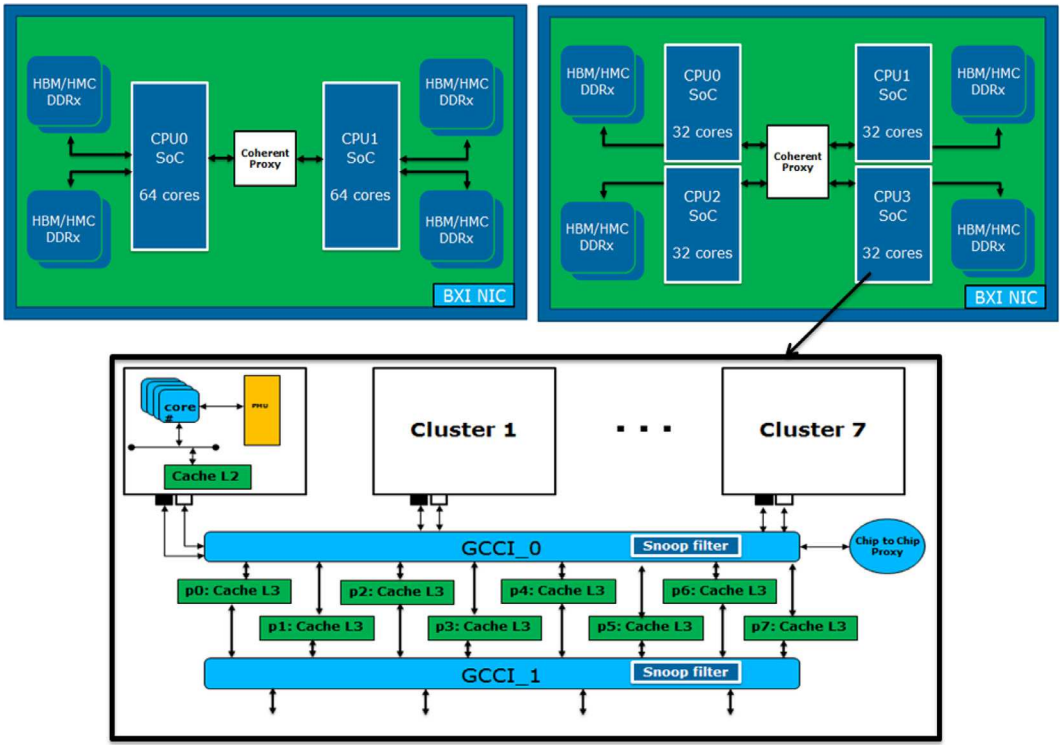


Fig. 8. SoC and cluster internal views (two SoCs / 64 cores, four SoCs / 32 cores).

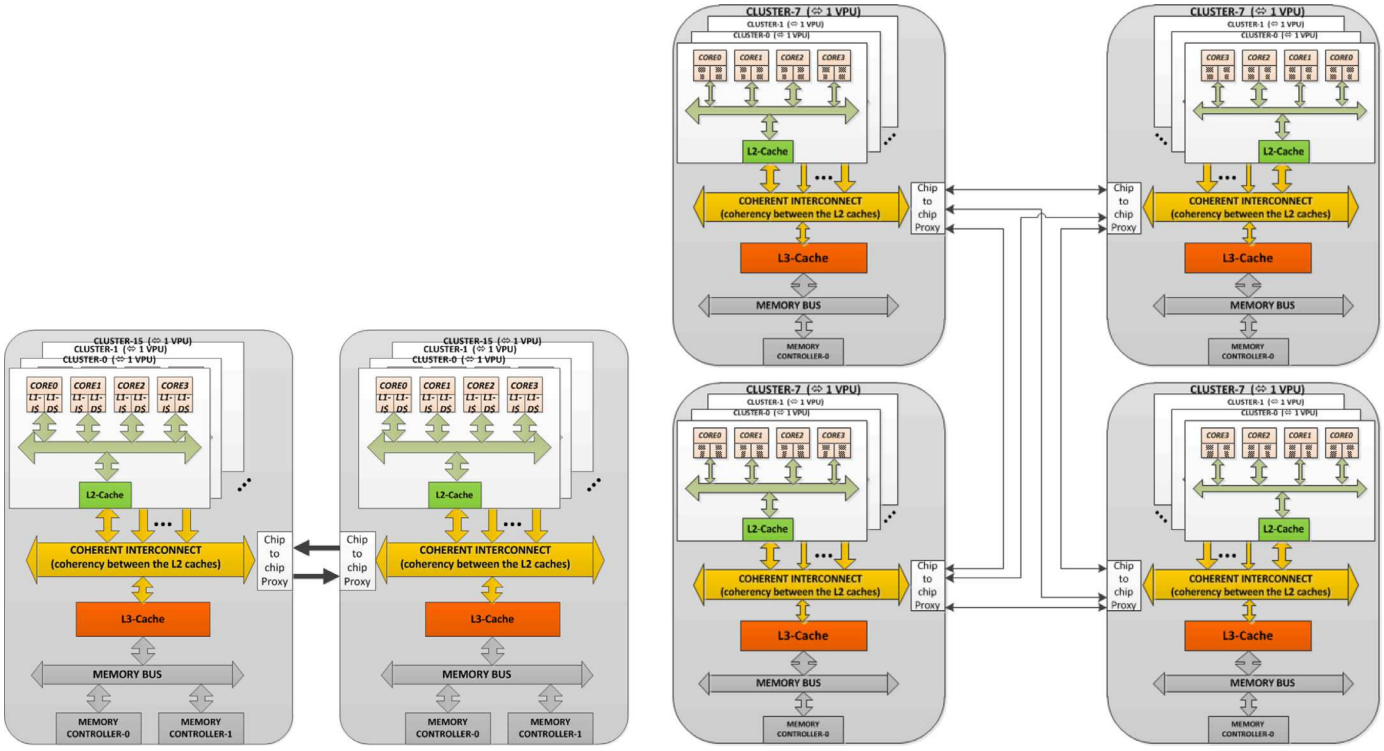


Fig. 9. 2×64 and 4×32 SoC partitioning with coherent proxy ports.

sistency between the multi-level system cache hierarchy and the main memory [16].

In the past, one way to maintain coherency in a multi-cache system was to use software but the required performance challenge became even greater as systems got bigger [16]. This is typically inadequate for HPC systems. Hardware-based cache coherence protocols thus appeared to be more efficient and are commonly used in many computing systems such as servers. A cache coherence protocol specification must define several key elements of the protocol background such as hardware components, coherency granularity, caches sizes, transactions types, the coherence interconnect channels or the impact of transaction on the cache line states. Therefore for the same coherence protocol, there may be several implementations depending on the type of multiprocessor or architecture.

There are many comparison studies between the two main hardware based coherency approaches: snooping cache coherency protocols and directory-based cache coherency protocols [15,17,18]. It always emerges that the first one is logically simple, ideal for ring interconnect topology but does not scale to large numbers of caches because the number of snooping broadcast transactions increases quadratic with $N \times (N-1) = N^2 - N$, where the N is the number of coherent masters [19]. The second one is perfectly scaling but with the drawback that transactions take more time because of the directory filtering complexity where the traffic scales in theory at order $N \times ((N-1)+1) = N^2$, where $+1$ is the request for directory lookup [19]. In reality, this overhead only happens when all caches in a system have a copy of the requested data which is very rare in a large scale system (ten of caches). As it also depends on the workload parallelism, snooping traffic could thus be reduced between caches sharing a copy of the same data. The main counterpart of directory based cache coherency techniques is their implementation cost based upon on-chip SRAM storage whose size depends on the number of cache lines to manage within the system. Therefore, the potential for reducing cache coherence complexity attracts a lot of interest to improve the processing efficiency of large scale compute nodes. We explore in the following the impact of such question on a set of HPC benchmarks based on the previously defined methodology.

3.2.2. Coherence extension

We consider two partitioning scenarios of the SCI resulting from splitting a single-SoC 128 cores topology in two and four partitions. Fig. 9 shows the two block diagrams for the corresponding cluster configurations of 1×128 and 2×64 cores. All the processors in a partitioned scenario must be able to communicate coherently as if they were connected to the same on chip coherent interconnect. All SoCs are thus connected through chip-to-chip coherent proxy ports. The main role of these coherent proxy ports is precisely to enable both coherent transactions with the neighbouring sockets and accesses to external memory areas. The L3 cache is considered to be LLC (Last Level Cache) near each memory controller to save latencies for memory requests.

Fig. 10 depicts exactly the SystemC/TLM2 directory based filtering model as it would be implemented in the SCI. For example, in the scenario where there is an incoming snoop request from L2 cache, the snoop controller sends a request to the directory to locate all copies of the data in the nearby peer caches (in its shareability domain). Then it queries directly the caches identified in the local socket or through the proxy extension if an external transaction is involved. In case of a snoop miss, the request is forwarded to the next cache level (L3).

The proxy component architecture is mainly similar to an empty L2 cache. It repeats incoming snooping requests from a SoC to the other(s). This leads to additional delays and asymmetric waiting response times to the snoop controller requests. The snoop controller is a module attached to each coherent interface (port) in the SCI. It is responsible for processing coherent transactions from a cluster (VPU) to the directory and the (N−1) other peer interfaces. When the SCI manages 32 coherent ports (32 clusters of four cores), each snoop controller must be able to communicate with the other 31 controllers. In the scenario of partitioning in two SoCs,

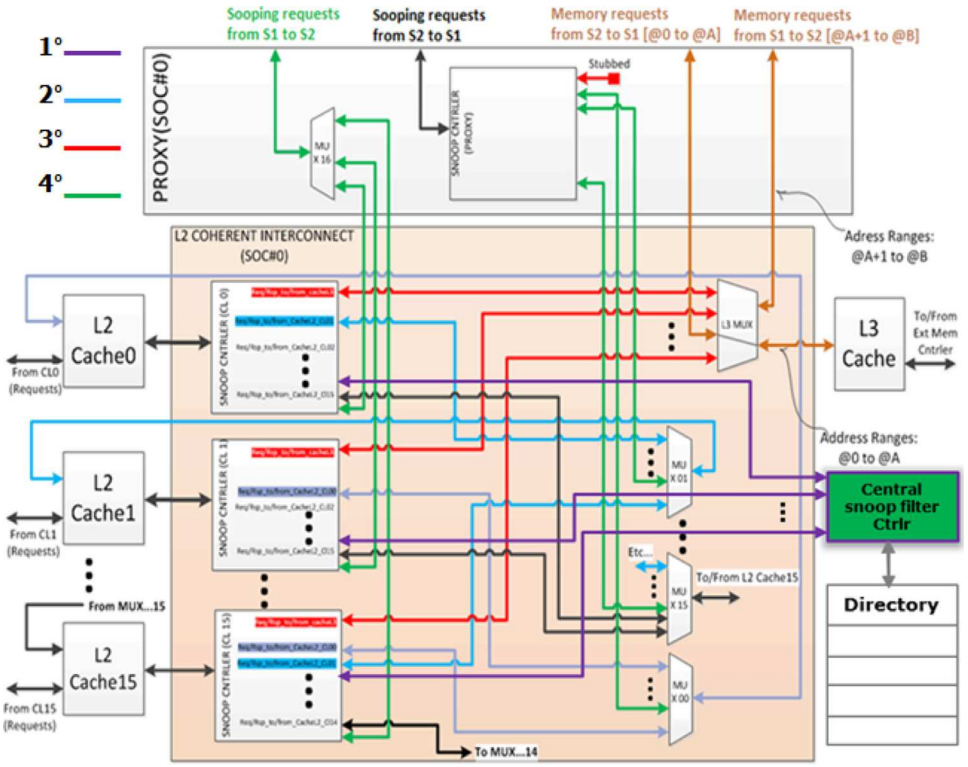


Fig. 10. Chip-to-chip coherence extension through a Proxy component (2×64 SoC partitioning).

it will communicate only with 16 snoop controllers (15 coherent VPU ports and the proxy one) instead of 31 snoop controllers in a large SCI (Fig. 10). This reduces de facto the logic design complexity and the corresponding physical area. The idea remain the same when partitioning in four SoCs, reducing connections from 31 links to 8 (7 peer ports + 1 hub proxy port) in a *one-to-all* topology or from 31 to 10 (7 peers ports + 3 direct proxy port) for an *all-to-all* topology.

3.3. Simulations

In this assessment study, we address first the ability of using previous directory based snoop filtering model in the SCI to reduce the complexity of cache coherence management and the impact on performances considering different types of benchmarks. We then analyse the relevance of two SoC partitioning configurations of two SoCs / 64 cores (2×64) and four SoCs / 32 cores (4×32) against one SoC / 128 cores (1×128), with and without the proposed snoop filtering scheme. Finally we consider more specifically the effect of different parallel programming paradigms on the internal traffic of the coherent interconnect and the corresponding performances.

3.3.1. Directory-based filtering benefits

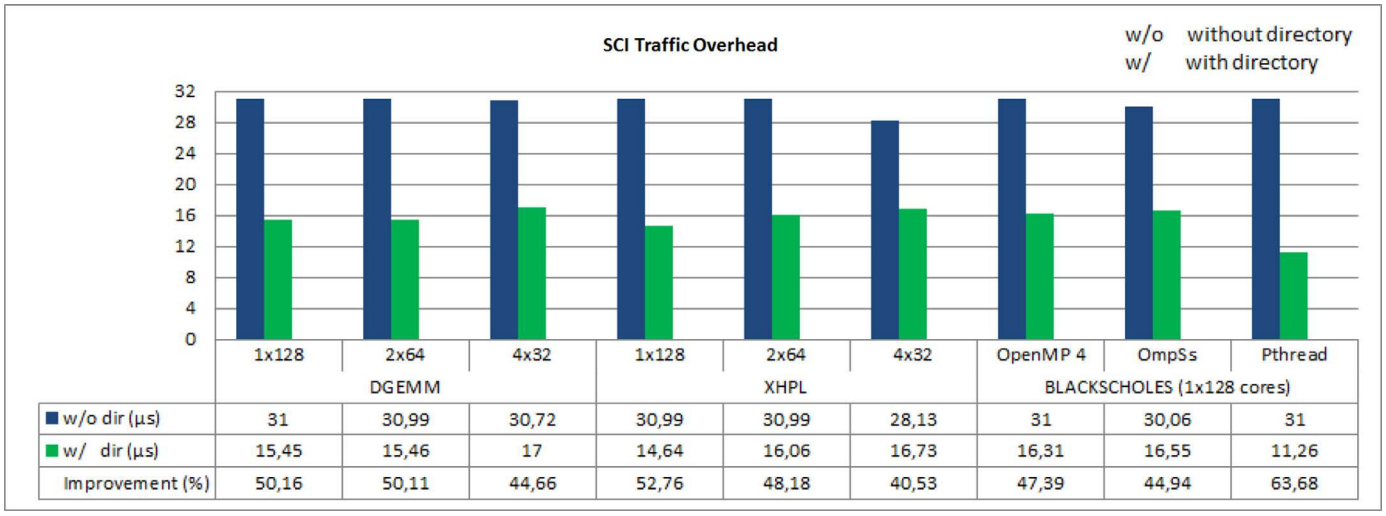
Fig. 11 reports the analysis of snoop transactions generated in each of the three configurations of 32 masters (128 cores). Without directory, 31 transactions need to be generated for each cache miss (or invalidation request) to update every copy of the data. Including a directory reduces the traffic generated in the SCI between 40% and 63%, which is in line with the reduction of size of the snoop controller by a factor of two, from 31 snooping output ports to 16 (Section 3.2.2). In turn, transaction benefits improve benchmark execution times by reducing L2 cache miss penalties. Therefore, Fig. 13 compares execution times of the same benchmarks, with and without directory, to examine these gains. The impact of using a directory in our coherent multi-SoC architecture model is thus an average execution time improvement of 14% (4%–26%). Despite transaction savings of about two, net performance gains are limited by the relative low number of cache misses in practice in real applications (around 2.7% reported for DGEMM in Fig. 12).

However, the consistency of data shared between processors is very complex in large scale computing systems. The use of snoop filtering in the three considered SoC models ensures both data coherency and processing efficiency by reducing transactions to those that are strictly necessary. The overhead for each incoming transaction is thus reduced by more than 40% which is an important matter in consumption and scalability at the memory subsystem level, while performance may be slightly improved at the same time. Memory consistency among clusters is thus ensured with significantly less coherence traffic in light of these results. Despite little benefits in terms of net system performance, this should however bring enough improvement to promote multi-SoC partitioning at low chip-to-chip communication costs, which is the question discussed next.

3.3.2. Partitioning analysis

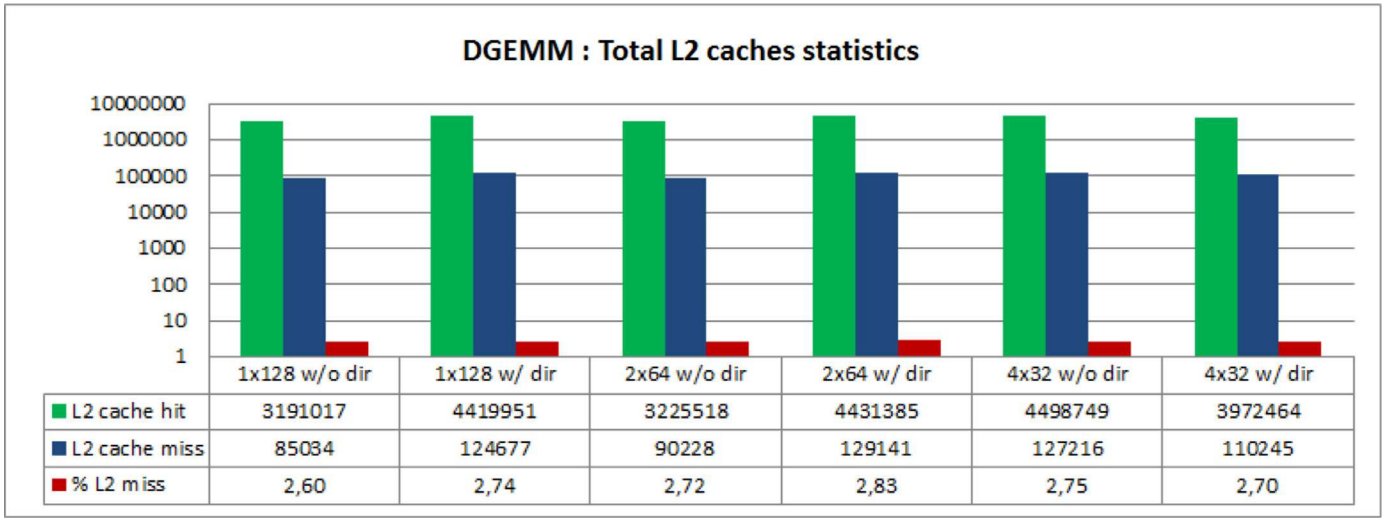
If we focus more specifically on multi-SoC configurations (2×64, 4×32 w/o dir) against single-SoC (1×128 w/o dir) in the results of Fig. 13, we can observe a legitimate deterioration of performances when no directory is used, increasing gradually with SoC partitioning due to the additional latencies coming from chip-to-chip transactions. Considering SoC partitioning in a large scale manycore system does not seem at first to be an effective approach as shown with up to 15% drop in performance for XHPL in configuration 4×32.

However, the presence of a directory improves both single-SoC and multi-SoC performances by respectively 13.7% and 14.33% (in average) since transactions are reduced to what is strictly required for cache coherence. The additional complexity introduced by the directory is widely compensated by the removal of inter-



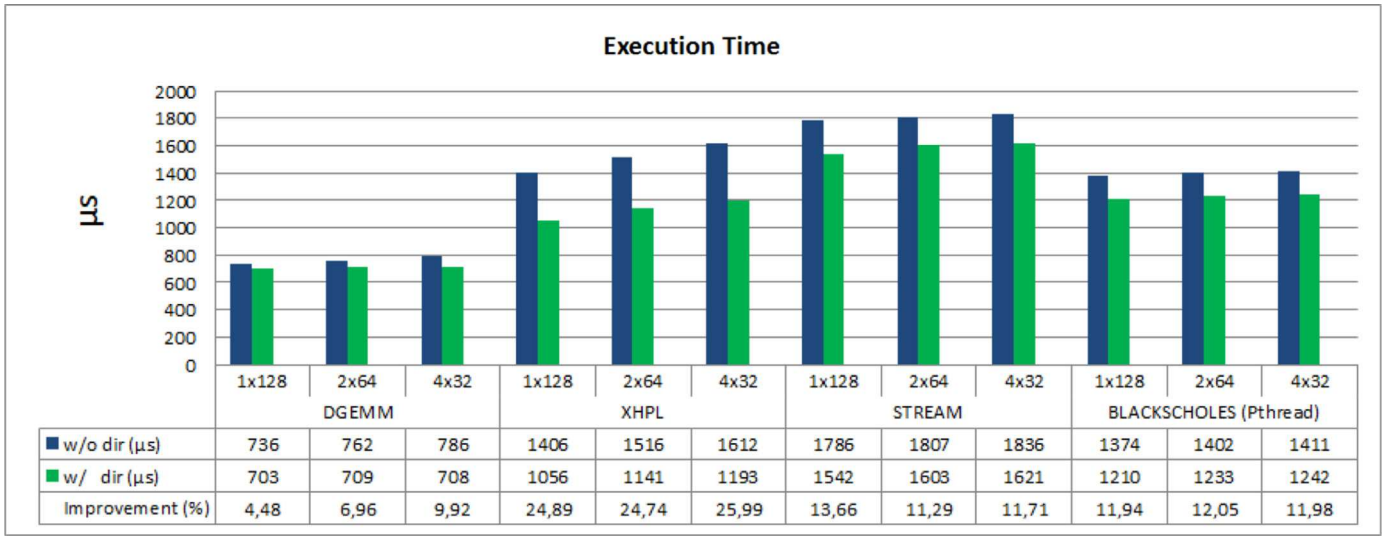
	1x128	2x64	4x32	1x128	2x64	4x32	OpenMP 4	OmpSs	Pthread
	DGEMM			XHPL			BLACKSCHOLES (1x128 cores)		
w/o dir (µs)	31	30,99	30,72	30,99	30,99	28,13	31	30,06	31
w/ dir (µs)	15,45	15,46	17	14,64	16,06	16,73	16,31	16,55	11,26
Improvement (%)	50,16	50,11	44,66	52,76	48,18	40,53	47,39	44,94	63,68

Fig. 11. Number of transactions for different SoC and directory configurations (Overhead = number of transactions generated for one incoming request).



	1x128 w/o dir	1x128 w/ dir	2x64 w/o dir	2x64 w/ dir	4x32 w/o dir	4x32 w/ dir
L2 cache hit	3191017	4419951	3225518	4431385	4498749	3972464
L2 cache miss	85034	124677	90228	129141	127216	110245
% L2 miss	2,60	2,74	2,72	2,83	2,75	2,70

Fig. 12. DGEMM cache statistics.



	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32
	DGEMM			XHPL			STREAM			BLACKSCHOLES (Pthread)		
w/o dir (µs)	736	762	786	1406	1516	1612	1786	1807	1836	1374	1402	1411
w/ dir (µs)	703	709	708	1056	1141	1193	1542	1603	1621	1210	1233	1242
Improvement (%)	4,48	6,96	9,92	24,89	24,74	25,99	13,66	11,29	11,71	11,94	12,05	11,98

Fig. 13. Benchmark performance for different SoC and directory configurations.

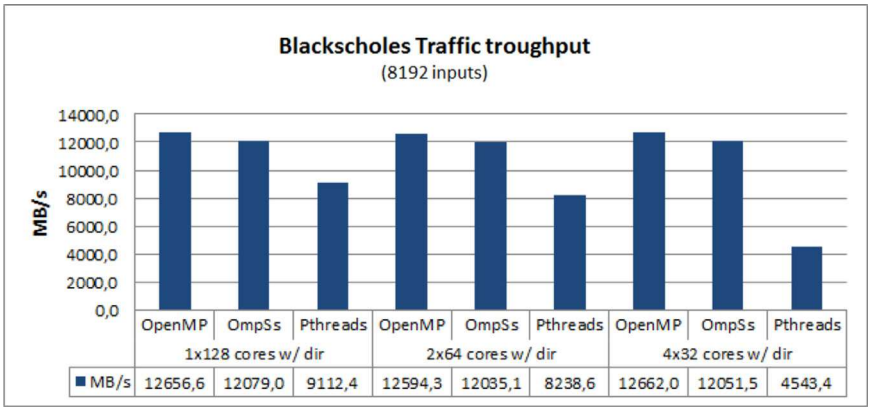


Fig. 14. Impact of programming models on throughput (blackscholes).

nal/external waiting delays. This gain rise up from 4% up to 25% depending on the workload profile. The XHPL benchmark, which employs a large memory space and is very sensitive to latencies, is therefore the ideal example to show the impact of a directory in our on-chip interconnect with 25% performance improvement.

In addition, performances remain stable in the presence of a directory with an average variation of 1.9% across all SoC configurations and benchmarks. If we compare the partitioned (2×64, 4×32 w/ dir) versus single-SoC (1×128 w/ dir) topologies, it can be verified that the impact of partitioning on execution times has been efficiently limited through the use of the directory (4.3% for 2×64 and 5.4% for 4×32). The performance level is more stable in both partitioned SoCs because there is always a copy of shared data in nearby caches resulting in no snoop misses.

These results let us therefore expect an average performance penalty of 5.2% resulting from SoC partitioning (in two and four SoCs for a 128 nodes example). But employing the defined directory-based filtering scheme is efficient enough at reducing chip-to-chip cache coherency transactions and to get rid of this overhead with even a mean improvement of 10.1% (over single-SoC without directory). The coherence extension scheme then promotes interesting opportunities such as the integration of more compute nodes directly on an interposer based System-in-Package (SiP), possibly based on 3D Through Silicon Vias (TSVs) using High Memory Bandwidth (HBM), to approach the processing power and efficiency of Exascale requirements.

3.3.3. Parallel programming efficiency

Another factor which may affect the value of SoC partitioning relates to software parallelism. The issue here is how to best minimize outgoing transactions between the partitioned SoCs at the programming level. We have thus considered three parallel programming models (OpenMP, OmpSs and POSIX Threads) on a *blackscholes* application (part of the PARSEC benchmark suite [20]) to investigate their influence on the efficiency of the directory.

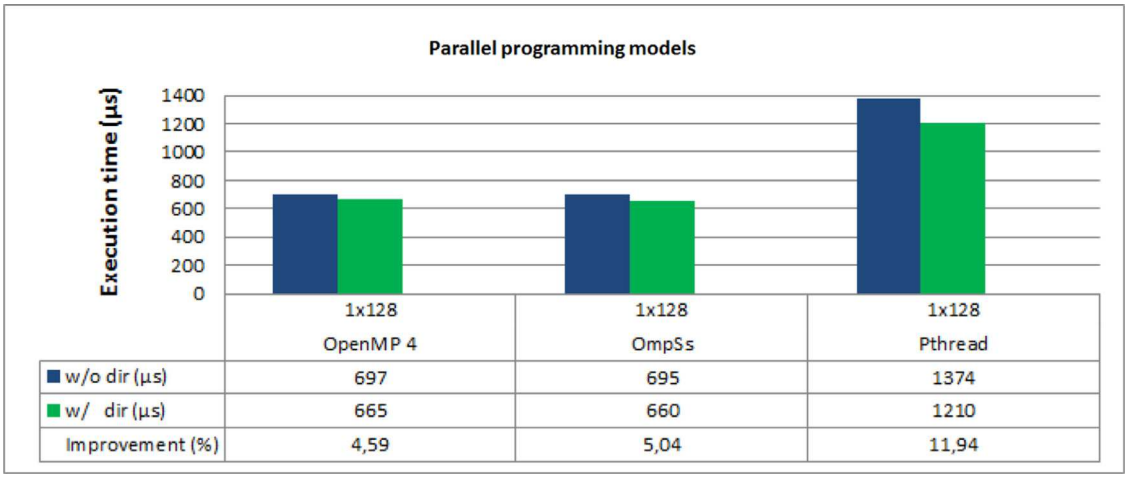
OpenMP, OmpSs and POSIX Threads are application programming interfaces (APIs) for multi-platform shared-memory parallel programming. OpenMP provides high level threading options using code and dataflow annotations that are then used by the runtime system for execution and synchronization. POSIX threads is a lower level API for working with threads offering fine-grained threading-specific code to permit control over threading operations. Unlike OpenMP, the use of Pthreads requires explicit parallelism expression in the source code (e.g. hard-coded number of threads). OmpSs is a mix of OpenMP and StarSs, a programming model developed by the Barcelona Supercomputing Center. It provides a set of OpenMP extensions to enable asynchronous tasks, heterogeneity (accelerators) and exploit more performance out of parallel homogeneous and heterogenous architectures.

OpenMP, OmpSs and POSIX Threads are application programming interfaces (APIs) for multi-platform shared-memory parallel programming. OpenMP provides high level threading options using code and dataflow annotations that are then used by the runtime system for execution and synchronization. POSIX threads is a lower level API for working with threads offering fine-grained threading-specific code to permit control over threading operations. Unlike OpenMP, the use of Pthreads requires explicit parallelism expression in the source code (e.g. hard-coded number of threads). OmpSs is a mix of OpenMP and StarSs, a programming model developed by the Barcelona Supercomputing Center. It provides a set of OpenMP extensions to enable asynchronous tasks, heterogeneity (accelerators) and exploit more performance out of parallel homogeneous and heterogenous architectures.

With previous results showing little influence of partitioning when using a directory, the following analysis is restricted to a single-SoC configuration. Fig. 15 reports execution times of the *blackscholes* benchmark for each programming model. There are comparatively few differences between OpenMP and OmpSs results because of their similarity. The application receives little benefits (4.6% performance improvement) from OmpSs specific features to better exploit the architecture model. However, Figs. 14 and 15 show two clear inflection points with 82.6% less performance and 40.9% fewer application throughput using Pthreads compared to OpenMP and OmpSs. Investigating further shows that Pthreads has 28% less throughput than OpenMP (in configuration 1×128 with directory) for 79% more cache misses (Fig. 16). In turn, cache misses are responsible for generating 23.5% more traffic in the SCI compared to OpenMP (Fig. 17).

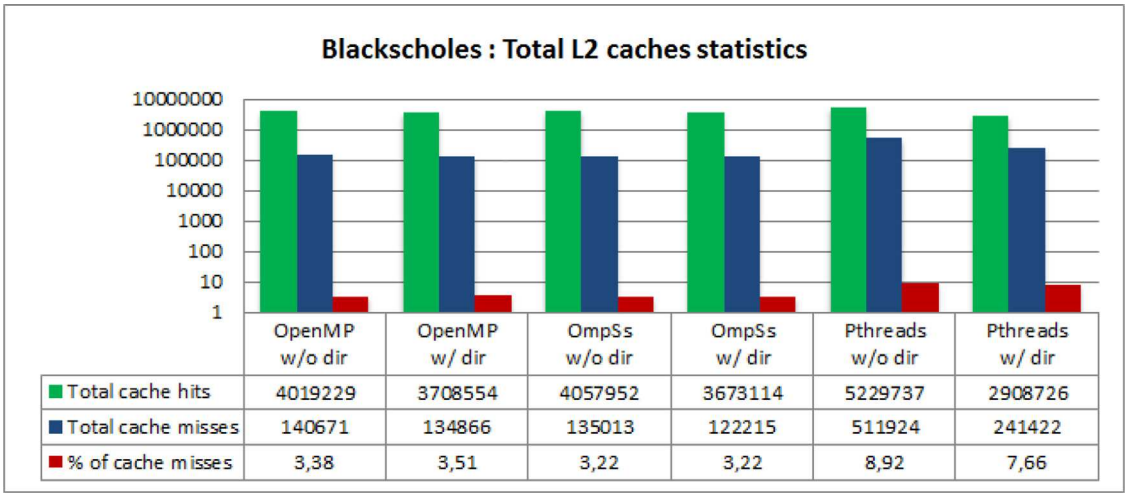
With previous results showing little influence of partitioning when using a directory, the following analysis is restricted to a single-SoC configuration. Fig. 15 reports execution times of the *blackscholes* benchmark for each programming model. There are comparatively few differences between OpenMP and OmpSs results because of their similarity. The application receives little benefits (4.6% performance improvement) from OmpSs specific features to better exploit the architecture model. However, Figs. 14 and 15 show two clear inflection points with 82.6% less performance and 40.9% fewer application throughput using Pthreads compared to OpenMP and OmpSs. Investigating further shows that Pthreads has 28% less throughput than OpenMP (in configuration 1×128 with directory) for 79% more cache misses (Fig. 16). In turn, cache misses are responsible for generating 23.5% more traffic in the SCI compared to OpenMP (Fig. 17).

Fig. 18 confirms that Pthreads has not led to an efficient use of the directory. With 77.37% snoop misses, the on-chip interconnect



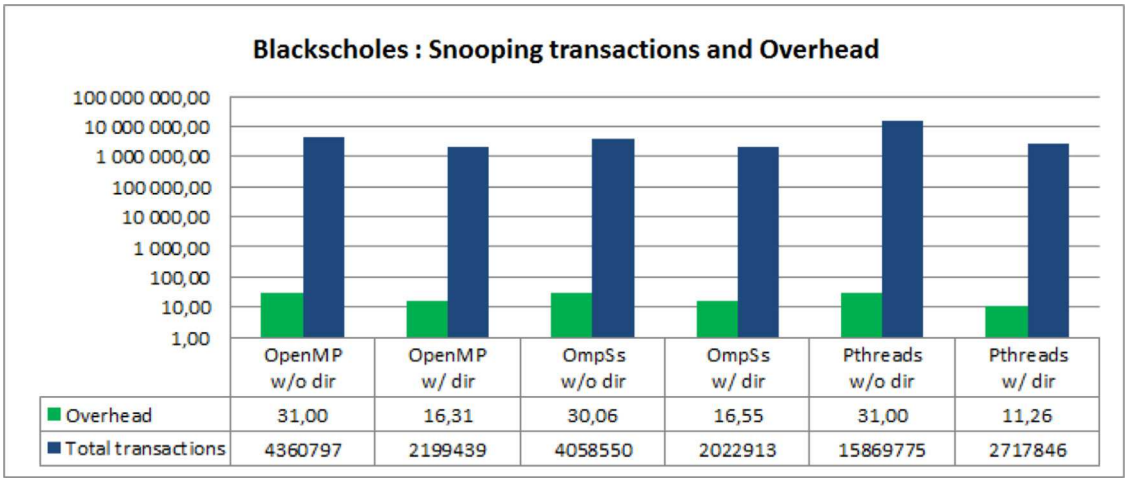
	1x128 OpenMP 4	1x128 OmpSs	1x128 Pthread
w/o dir (µs)	697	695	1374
w/ dir (µs)	665	660	1210
Improvement (%)	4,59	5,04	11,94

Fig. 15. Impact of programming models on performance (blackscholes, 1×128 cores).



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
Total cache hits	4019229	3708554	4057952	3673114	5229737	2908726
Total cache misses	140671	134866	135013	122215	511924	241422
% of cache misses	3,38	3,51	3,22	3,22	8,92	7,66

Fig. 16. Impact of programming model on L2 cache misses (blackscholes, 1×128 cores).



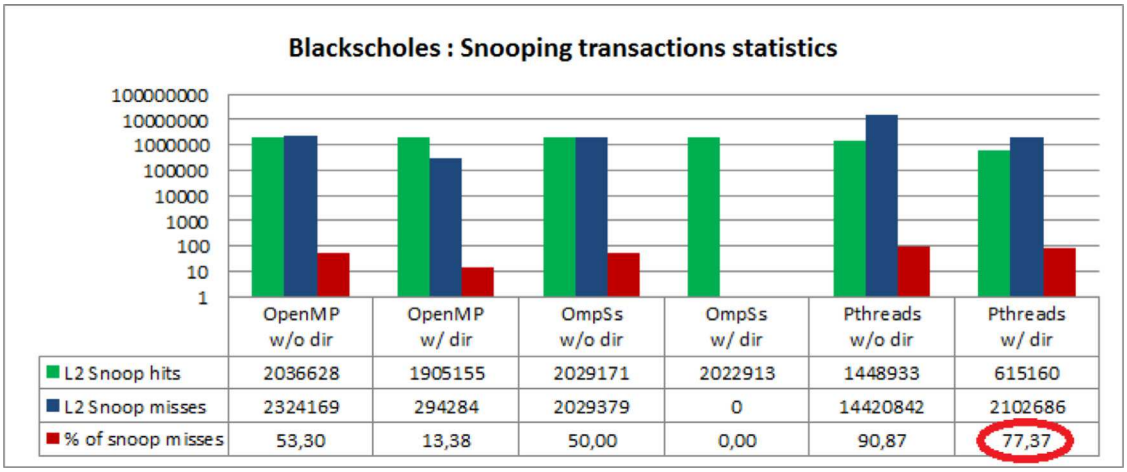
	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
Overhead	31,00	16,31	30,06	16,55	31,00	11,26
Total transactions	4360797	2199439	4058550	2022913	15869775	2717846

Fig. 17. Impact of programming models on the number of snooping transactions and overhead (blackscholes, 1×128 cores).

had to carry out the transport of five times more transactions than OpenMP.

The reasons of these weaknesses lie mainly in the capability of the software model to address efficiently high degrees of parallelism. In the task-graph based parallel versions of *blackscholes* used for OpenMP and OmpSs, the work is better divided into units of a predefined block size which allows having much more task instances and better load balance than Pthreads. The effects from less reliable parallelism exploitation can therefore increase significantly (up to a factor of two) when scaling up to 128 cores.

Besides the fact that limited conclusions can be drawn on the effectiveness of a programming model which depends on how well



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
L2 Snoop hits	2036628	1905155	2029171	2022913	1448933	615160
L2 Snoop misses	2324169	294284	2029379	0	14420842	2102686
% of snoop misses	53,30	13,38	50,00	0,00	90,87	77,37

Fig. 18. Snooping traffic statistics (blackscholes, 1×128 cores).

the software was partitioned and coded, this study shows however the potential for deep analysis of appropriate parallelism exploitation by the application. These results are beneficial to help tuning the architecture and design of algorithms and software, to identify and correct programming shortcomings and further improve parallel processing efficiency.

4. Conclusion and perspectives

In this paper, we have examined in detail how a combined use of relevant models, tools, platforms and benchmarks could be used to define a robust design space exploration approach adapted to the tight processing efficiency constraints of upcoming HPC, especially in the new perspectives offered by 64-bit *ARMv8-A* cores. Proper architectural exploration is decomposed in two steps that allow (i) reliable modeling and simulation at node/cluster level and (ii) scalability analysis of a larger number of nodes using *ARMv8-A* core models. Reported experiments and results have shown the ability of the approach to reliably study central design parameters, namely in terms of FLOPS performance and efficiency, cache and memory hierarchy, and scalability support up to 128 nodes.

Using this methodology, we have explored opportunities for multi-SoC partitioning based on a directory-based coherent interconnect (SCI) defined specifically for this purpose. Exploration of partitioned (2×64, 4×32) versus single-SoC (1×128) topologies with this coherent interconnect have shown to decrease significantly the associated internal traffic (55.3%) and to limit enough the existing partitioning overhead (4.3% for 2×64 and 5.4% for 4×32) such as to permit an average 10.1% execution time saving compared to the situation where no partitioning / directory is used. Additionally, the analysis of parallel programming efficiency on a concrete example confirmed the validity of directory filtering with the ability to identify and correct software weaknesses for better parallel processing efficiency.

The perspectives from this work[1] are to build on the results achieved to combine efficiently all the elements identified for improvement (64-bit *ARMv8-A* cores, SoC and interconnect partitioning, interconnect and cache coherency complexity, parallel programming and 3D integration perspectives) such as to extend significantly the computing efficiency in large scale HPC systems.

[1] The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] and Horizon 2020 under the Mont-Blanc Projects, grant agreement no. 288777, 610402 and 671697. It was also partly funded by a French ANRT CIFRE partnership between Bull (Atos technologies) and LEAT (CNRS UMR7248, University of Nice Sophia Antipolis).

References

[1] K. Fürlinger, C. Klausecker, D. Kranzlmüller, Towards energy efficient parallel computing on consumer electronic device, in: International Conference on Information and Communication on Technology for the Fight Against Global Warming (ICT-GLOW'11), Toulouse, France, 2011.
[2] E.L. Padoin, D.A.G. de Oliveira, P. Velho, P.O.A. Navaux, B. Videau, A. Degomme, J.-F. Mehaut, Scalability and energy efficiency of HPC cluster with ARM MPSoC, Workshop on Parallel and Distributed Processing (WSPPD), Porto Allegre, Brazil, 2013.
[3] N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovik, A. Ramirez, Experiences with mobile processors for energy efficient HPC, in: Design, Automation and Test in Europe Conference and Exhibition (DATE), Grenoble, France, 2013.
[4] Z. Ou, B. Pang, Y. Deng, J.K. Nurminen, A. Yla-Jaaski, P. Hui, Energy and cost-efficiency analysis of ARM based clusters, Symposium on Cluster, Cloud and Grid Computing (CCGRID), Ottawa, Canada, 2012.
[5] M.F. Cloutier, C. Paradis, V.M. Weaver, Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance, Hardware-Software Co-Design for High Performance Computing (Co-HPC), New Orleans, USA, 2014.
[6] M.A. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W.A. Ward Jr, R. Campbell, L. Carrington, Characterizing the performance-energy tradeoff of small ARM cores in HPC computation, in: European Conference on Parallel Processing (Euro-Par), Porto, Portugal, 2014.
[7] J. Maqbool, S. Oh, G.C. Fox, Evaluating energy efficient HPC clusters for scientific workloads, Concurr. Comput. Pract. Exp. 27 (17) (2015).
[8] E.R. Blem, J. Menon, K. Sankaralingam, Power struggles: revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures, Symposium on High Performance Computer Architecture (HPCA), Shenzhen, China, 2013.
[9] N. Rajovic, P. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, Are mobile processors ready for HPC? in: International Supercomputing Conference (SC13), Denver, USA, 2013.
[10] A. Ramirez, European scalable and power efficient HPC platform based on low-power embedded technology, European Exascale Software Initiative (EESI), Barcelona, Spain, 2011.
[11] A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W.D. Gropp, L. Kale, Architectural constraints to attain 1 exaflop/s for three scientific application classes, Parallel & Distributed Processing Symposium (IPDPS), Anchorage, USA, 2011.
[12] C.L. Lawson, R.J. Hanson, D. Kincaid, F.T. Krogh, Basic linear algebra subprograms for FORTRAN usage, ACM Transactions on Mathematical Software (TOMS) 5 (3) (1979).
[13] J. Dongarra, P. Luszczek, A. Petitet, The LINPACK benchmark: past, present, and future, Concurr. Comput. Pract. Exp. 15 (9) (2003).
[14] J.D. McCalpin, Sustainable Memory Bandwidth in Current High Performance Computers, Technical report (1991–2007), University of Virginia, http://www.cs.virginia.edu/stream/.
[15] D.J. Sorin, M.D. Hill, D.A. Wood, A primer on memory consistency and cache coherence, Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2011.
[16] R. Weber, Modeling and verifying cache-coherent protocols, International Symposium on Circuits and Systems (ISCAS), Sydney, Australia, 2001.

[17] A.R. Lebeck, D.A. Wood, Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors, International Symposium on Computer Architecture (ISCA'95), Santa Margherita Ligure, Italy, 1995.
[18] A. Moshovos, G. Memik, B. Falsafi, A. Choudhary, Jetty: filtering snoops for reduced energy consumption in SMP servers, Symposium on High-Performance Computer Architecture (HPCA'01), Nuevo Leone, Mexico, 2001.
[19] A. Stevens, Introduction to AMBA 4 ACE and big.LITTLE processing technology, ARM Holdings, 2013.
[20] C. Bienia, S. Kumar, J.P. Singh, K. Li, The PARSEC benchmark suite: characterization and architectural implications, Parallel Architectures and Compilation Techniques (PACT), Toronto, Canada, 2008.

Joel Wanza Weloli received his M.E. degree in Electronics and Embedded Systems from University of Cote d'Azur, France in 2014. He is currently occupying a Ph.D. student position at Bull (Atos technologies) as member of the hardware architecture team. His research interests are power and energy models and design space exploration especially for ARM based compute nodes and clusters in the field of High Performance Computing.

Sebastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis - CNRS. His research interests are in design, exploration and optimization from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of french, european and international collaborative research projects.

Maarten De Vries received his M.Sc. in Telecommunications engineering from Télécom-ParisTech school (Paris, France), in 1991. As an engineer, he has more than 25 years of experience, through various R&D positions in major companies like Philips, and STMicroelectronics/ST-Ericsson. His expertise covers Hardware and Software developments as well as system architecture including virtual platforms modeling in SystemC. Since 2012, he is in charge of functional verification for ASIC development at Bull (Atos) for the BXI project (Bull Exascale interconnect for HPC). Since 2015, hes the WP7 lead of the MontBlanc2 project.

Said Derradji As a hardware architect at Bull, Said Derradji has been working first on several custom ASIC design interconnecting processors and focusing on cache coherency. He also worked on board design and participated on TERA-100 system delivery in 2010 (ranking 9 in Top500.org). Since 2012, he is working in the hardware architecture team at Bull, which specified recently the open exascale supercomputer, code-named SEQUANA. His areas of expertise are on ASIC/FPGA design, HPC servers architecture and on high performance interconnect technology such as the recently announced BXI (Bull Exascale Interconnect). He is BULLs representative at PCI-SIG (PCI Special Interest Group) and IBTA (InfiniBand®Trade Association) consortiums.

Cecile Belleudy is an Associate professor at University of Côte d'Azur. She currently leads the MCSOC team (Modelisation and system design of Communicating Objects) at LEAT laboratory and the Master Degree in Electronics, Systems and Telecommunications at University of Nice Sophia Antipolis. Her research interests are in the general field of System-on-Chip design with a specific interest in power optimisation, including power management, low power and real time scheduling, DVFS, DPM techniques and applications to multiprocessor architectures, multi-bank memories, operating systems.