Mapping parallel applications on parallel architectures

Kevin J M Martin

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud
ubs

HABILITATION À DIRIGER DES RECHERCHES

UNIVERSITÉ DE BRETAGNE SUD

ÉCOLE DOCTORALE N° 644
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique et Architectures numériques*

Par

Kevin J. M. MARTIN

Mapping parallel applications on parallel architectures

Granularity of parallelism and synchronisation

soutenue à Lorient, le 16 mars 2023
Unité de recherche : Lab-STICC

Composition du Jury :

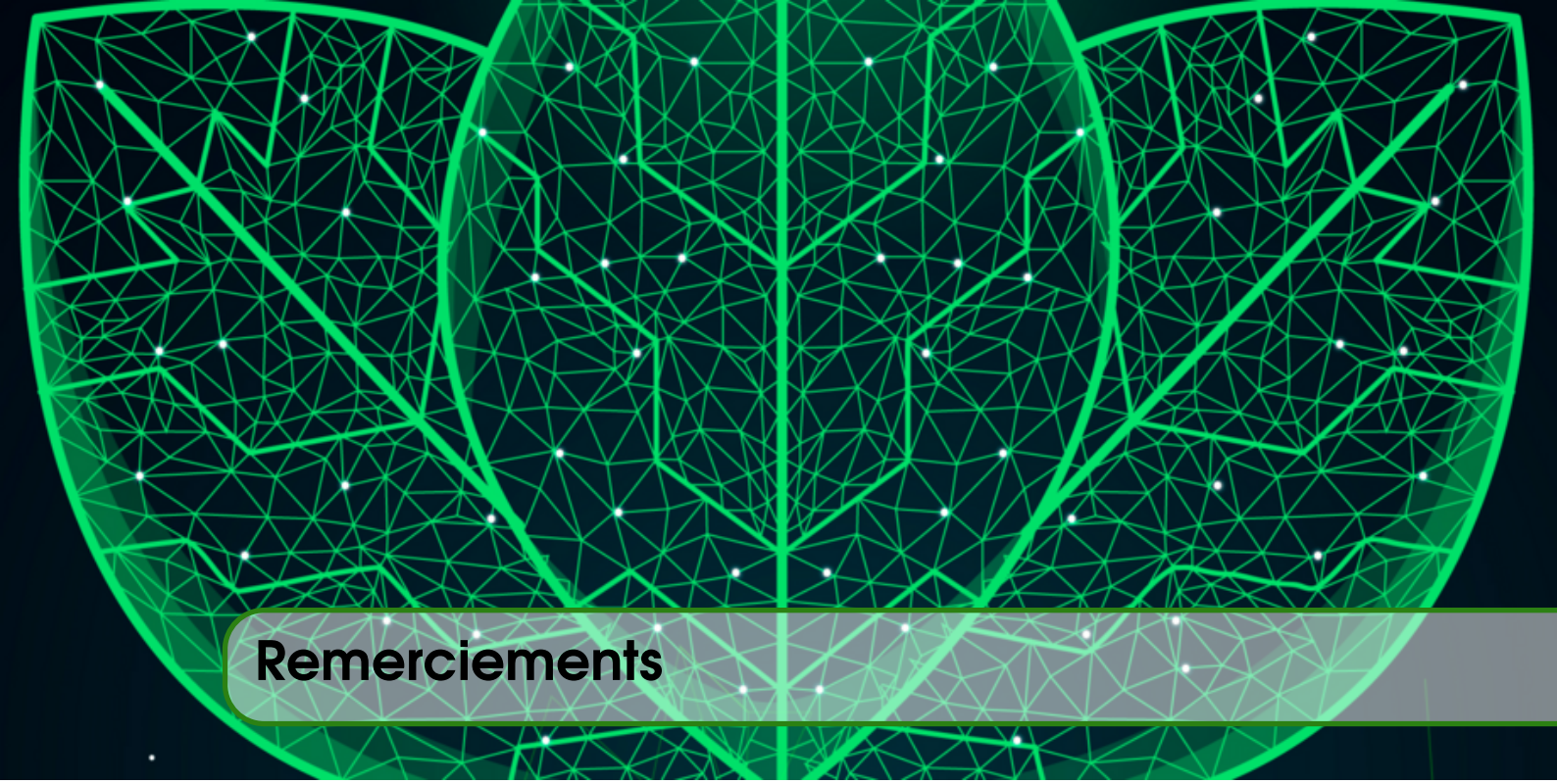
Président :	Olivier SENTIEYS	Professeur - Université de Rennes
Rapporteurs :	Shuvra S. BHATTACHARYYA	Professeur - University of Maryland et Chaire d'excellence - INSA/IETR Rennes
	Frédéric PÉTROT	Professeur - Université Grenoble Alpes
	Claire PAGETTI	Directrice de recherche - ONERA
Examinateurs :	Philippe COUSSY	Professeur - Université de Bretagne-Sud

Made available online, 2023

If you think education is expensive, try ignorance.

Robert Orben
(currently the leading candidate for crafter of this maxim)[1]

[1] https://quoteinvestigator.com/2016/05/03/expense/



Remerciements

Les premiers remerciements vont naturellement vers les membres du jury. Mais au-delà des remerciements de convenance, je tiens à les remercier sincèrement d'avoir pris de leur temps pour lire ce (long) manuscrit et pour avoir assisté à ma soutenance. Je remercie en particulier Prof. Philippe Coussy d'avoir relu ce document avant l'envoi aux rapporteurs. Je remercie Prof. Frédéric Pétrot, qui, après avoir fait l'exercice de rapporter sur ma thèse, a accepté de rapporter sur mon HdR. Je remercie Prof. Olivier Sentieys d'avoir accepté de présider mon jury d'HdR. Il a été mon chef d'équipe pendant ma thèse, et ce n'est pas sans fierté que je lui ai exposé mon parcours depuis que j'ai quitté son équipe. Je remercie DR. Claire Pagetti d'avoir accepté de rapporter sur le document. Je resterai la première HDR sur laquelle elle a rapporté. Enfin, je remercie Prof. Shuvra Bhattacharyya de s'être prêté à cet exercice très français.

Ensuite je remercie les personnes qui ont contribué aux travaux qui sont présentés dans ce document. L'HDR, dans notre domaine, est paradoxalement un diplôme individuel, mais accessible à condition de travailler avec d'autres personnes. Ce document reflète donc un travail collectif, dans lequel j'ai pris ma part, mais dont je ne suis pas le contributeur unique.

Je tiens à remercier les personnes, qui, même si nous n'avons pas eu l'opportunité de travailler ensemble sur des projets de recherche ou l'encadrement de doctorante et doctorant, ont aussi participé à ce travail d'HDR. Par leurs conseils, discussions (pas nécessairement scientifiques), avis et commentaires.

Ce document met l'accent sur mes activités de recherche, mais je tiens à ne pas oublier de remercier mes collègues de l'IUT de Lorient, en particulier mes collègues du département GIM.

La vie d'enseignant-chercheur est un équilibre fragile, qui force à jongler entre enseignement, recherche, administration, et ce "travail-passion" s'invite volontiers dans la sphère privée et s'accapare sans scrupule du temps originellement dédié à d'autres activités personnelles. Sans repère solide, difficile d'avancer sereinement. Ma famille, mes amis et mes proches, ont donc aussi contribué à la réalisation de cette HDR. Lao n'nyarabi, na mansa muso.

Merci donc à toutes les personnes qui ont contribué de près ou loin.

Merci à toi Philippe, Matthieu, Gwenolé, Luca, Davide, Mickaël, Thomas, Satyajit, Rohit, Chilanka, Christie, Remi, Thomas, Jorgiano, Ghizlane, Anju, Chengcong, Ramesh, Baptiste, Majed.

Merci à toi Jean-Philippe, Thanh, Daniel, Yvan, Yaset, Emmanuel, Jean-Philippe, Mickaël, Paola, Mostafa, Johanna, Cataldo, Marcon, Alemeh, Marcelo, Navonil, Hemanta, Jérôme, Serge, Hajar, Hugo, Christina, Khadimou.

Merci à toi Florence, Virginie, Guy, Vianney, Arnaud, Mr Laurent, Marc, Cyrille, Philippe, Florent, Pierre, Dominique, Bertrand, Cédric, Cédric, les collègues du Lab-STICC.

Merci à toi Christophe, Tahar, Géraldine, Pao, Denis, Cédric, Didier, Philippe, Philippe, Philippe, Philippe[2], Michel, Fred, Stéphane, Pascale, Cédric, Maud, Catherine, Morgane, Ludo, Valérie, Aziza, Serge, Yessine, Emilie, Yohan, Gaël, les collègues de l'IUT.

Merci aux communautés logiciels et matériels open-source : ORCC, PREESM, PULP, Gem5, Noxim, Linux, Sniper, QEMU, OVP, RISC-V, OpenPiton.

Merci à toi.

Merci aussi à Rantanplan.

[2] oui, ça fait plein de "Philippe", ce n'est pas une erreur de copier/coller

Contents

General Introduction

THIS document presents my teaching and research activities since September 2011, when I was hired as an associate professor at Université de Bretagne-Sud, to teach at IUT de Lorient and conduct my research in Lab-STICC laboratory, until August 2022. The document is organized in four main parts. The first part contains my curriculum vitae, and a synthesis of my teaching and research activities. The second part is the core of the document, presenting in more details my scientific contributions structured in three main research areas. The third part of the document opens up on research perspectives. Finally, the document ends with the bibliography and some selected publications. Each part is independent of each other.

This document is written in French for the first part (but some sections are likely to be understandable for a reader familiar with English). The second and third part of the document are written in English.

Synthèse

1. Curriculum Vitae

1.1 État civil

Nom	MARTIN
Prénoms	Kevin John Michel
Né à	*removed for public version*
Nationalité	*removed for public version*
	personal situation removed for public version

1.2 Situation Professionnelle

Poste	Maître de conférences
Établissement d'affectation	Université de Bretagne-Sud
Enseignement	IUT Lorient - Département Génie Industriel et Maintenance
Recherche	Lab-STICC - équipe ARCAD
Date de recrutement	01/09/2011
Grade	MCF classe normale
email	kevin.martin@univ-ubs.fr
site web	http://www-labsticc.univ-ubs.fr/~kmartin/
	https://orcid.org/0000-0002-8122-1192
téléphone	+33 2 97 87 46 36

ADRESSE PROFESSIONNELLE

Laboratoire Lab-STICC, UMR CNRS 6285
Centre de recherche, BP 92116 - 56321 LORIENT Cedex

1.3 Parcours

Septembre 2021 – Août 2022	**Délégation CNRS.** Laboratoire Lab-STICC, équipe ARCAD (hardware architectures and computer-aided design tools)
Septembre 2011 – Août 2021	**Maître de conférences** à l'Université de Bretagne Sud, IUT Lorient Laboratoire Lab-STICC, équipe ARCAD depuis 2020, précédemment MOCS (Méthodes, Outils, Circuits, Systèmes) 2011-2019
Septembre 2010 – Août 2011	**ATER** à Université Rennes 1, Irisa, équipe CAIRN
Avril 2007 – Septembre 2010	**Doctorat** en informatique de l'Université de Rennes 1
Avril 2005 – Avril 2007	Formation en **ingénierie logicielle** par voie d'apprentissage à l'ETGL (École des Techniques du Génie Logiciel), CFA AFTI, Saclay **Contrat d'apprentissage** en alternance (2 périodes de 6 mois) en tant qu'ingénieur logiciel, à l'Irisa, INRIA Rennes
2003-2004	**DEA STIR** (Signal, Télécoms, Image, Radar), Université de Rennes 1

1.3.1 Synthèse de la carrière

Après l'obtention d'un DEA en Signal, Télécoms, Image, Radar, option Image de l'université de Rennes 1 en 2004, j'ai suivi une formation en ingénierie logicielle par voie d'apprentissage à partir d'avril 2005. Cette formation était en alternance entre le CFA AFTI (Association pour la Formation aux Techniques Industrielles)[1] à Saclay, et l'INRIA Rennes. À l'INRIA, j'étais intégré dans l'équipe R2D2 (qui est devenu CAIRN puis TARAN en 2021). L'alternance consistait en des périodes de 6 mois. Mon parcours académique m'a donné un bagage électronique quand la formation m'a apporté des connaissances en informatique et m'a initié au développement logiciel. C'est ce double parcours qui me permet aujourd'hui de me situer à l'interface entre le monde matériel et le monde logiciel.

À l'issue de la formation en avril 2007, j'ai directement poursuivi en thèse dans l'équipe CAIRN, thèse dirigée par Christophe Wolinski que j'ai soutenue en septembre 2010. J'ai ensuite effectué une année d'ATER à l'université de Rennes 1 entre septembre 2010 et août 2011. Depuis septembre 2011, je suis maître de conférences à l'université de Bretagne Sud, et affecté à l'IUT de Lorient au département Génie Industriel et Maintenance (GIM) pour l'enseignement. Mes travaux de recherche s'effectuent au laboratoire Lab-STICC, Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance. Le laboratoire est présent sur les villes de Brest, Quimper, Lorient et Vannes. Le Lab-STICC est une unité de recherche inter-établissements (CNRS, IMT Atlantique, ENIB, ENSTA Bretagne, UBO, UBS) comprenant plus de 600 personnes. Mon établissement est l'UBS, et je me trouve physiquement à Lorient.

À mon arrivée à Lorient, j'ai été intégré aux activités de l'équipe Méthodes et Outils pour les Circuits et Systèmes (MOCS) du Lab-STICC, et j'ai eu la chance d'être directement impliqué dans le projet ANR COMPA et l'encadrement de deux thèses (dont une liée au projet). J'ai progressivement développé trois axes de recherches: 1) Architectures de calcul spécialisées, 2) Déploiement de tâches sur architectures parallèles, 3) Synchronisation de tâches sur architectures parallèles.

La thèse qui m'a été proposée de co-encadrer dès mon arrivée est une thèse financée par le CEA à Saclay. Le doctorant a effectué de cours séjours dans notre laboratoire mais est resté majoritairement à Saclay. L'encadrement se présentait donc sous la forme de quelques déplacements à Saclay (6 au total) et

[1] ce CFA n'existe plus aujourd'hui

de réunions par visio-conférences (3 par mois en moyenne). Cette thèse est devenue la première soutenue en tant qu'encadrant. Elle a débouchée sur deux publications en conférences internationales avec actes et comité de lecture (ASAP, GLS-VLSI) et deux dépôts de brevets. *Cette thèse a surtout lancé l'activité autour des CGRAs, le premier des axes de recherche développé dans ce document au chapitre 4, p. 49.*

Le projet COMPA est un projet ANR qui a débuté le 1er octobre 2011 et s'est terminé au 30 juin 2015. Le consortium initial incluait 3 partenaires académiques, l'IETR, l'Irisa (l'équipe dans laquelle j'ai effectué ma thèse), le Lab-STICC, et 3 partenaires industriels, Texas Instrument, CAPS Entreprise, et Modaë. Ce consortium a malheureusement beaucoup évolué car les trois partenaires industriels ont dû successivement se retirer du projet. Dans le cadre du projet, j'ai participé activement à la rédaction des différents livrables. Ce projet a permis de financer une thèse que j'ai co-encadrée. Elle a été soutenue le 19 juin 2015, et donnée lieu à une publication en conférence internationale avec comité de lecture (DASIP), et à un article dans une revue internationale (JSPS). Cette thèse s'est intéressée à la problématique du déploiement d'acteurs flot-de-données sur une architecture multi-processeurs hétérogène. *C'est le deuxième axe de recherche qui sera développé dans ce document au chapitre 5, p. 75.*

Lors de nos recherches pendant le projet COMPA, nous avons été exposé à la problématique de la synchronisation de tâches sur les architectures multi-processeurs. Dans le cas particulier des acteurs flot-de-données, nous pouvons tout simplement nous appuyer sur des techniques existantes, mais celles-ci s'avèrent peu efficaces alors qu'il est possible de s'appuyer sur les caractéristiques des applications flot-de-données pour améliorer la synchronisation entre les acteurs, en particulier par une meilleure gestion des accès mémoire. L'idée est de monitorer l'activité mémoire et déclencher des calculs en fonction de cette activité. Cette idée repose directement sur le patron de conception « *Observer* », bien connu dans le monde de l'ingénierie logiciel. Ma formation en génie logiciel et les nombreuses mises en œvre que j'ai faite de ce patron de conception ont été particulièrement utiles. Un premier prototype virtuel de l'application de ce patron de conception en matériel a été réalisé et fait l'objet d'une publication en conférence internationale (DAC2016). Les très bons résultats obtenus sur une seule application (un décodeur H264) méritaient des investigations plus poussées, c'est pourquoi j'ai déposé un projet ANR JCJC en automne 2016. Ce projet a été accepté et s'est étalé sur la période janvier 2018 - juin 2022. *Cette activité de recherche sur les mécanismes de synchronisation de manière générale, et en particulier sur l'idée originale de la capacité de la mémoire à déclencher des calculs, que j'ai nommé «* Notifying memories *», fait l'objet du troisième axe de recherche développé dans ce document au chapitre 6, p. 99.*

Depuis quelques années maintenant je m'intéresse particulièrement à la problématique mémoire et au lien entre architectures de calcul et mémoire, pour l'exploitation du parallélisme au niveau instructions ou de données avec les CGRAs, ou le parallélisme de tâche sur architectures multi-processeurs. Cette interaction calcul/mémoire est complexe, nécessite du temps, et couvre un large spectre de thématique. Pour pouvoir y consacrer plus de temps et développer ma propre activité sur ce sujet, j'ai sollicité une délégation CNRS d'un an à temps plein, que j'ai obtenue pour l'année universitaire 2021-2022. Cette délégation s'est effectué dans mon équipe de recherche.

Élément de contexte

Un élément de contexte important est la restructuration du Lab-STICC sur la période 2019-2020 au niveau de l'organisation des équipes de recherche, ce qui a conduit à scinder l'ancienne équipe MOCS (plus de 80 personnes), dans laquelle j'étais depuis 2011, en trois nouvelles équipes, dont ARCAD, mon équipe de recherche actuelle, pour une mise en place officielle à l'occasion de l'évaluation HCERES début 2021, en même temps que la mise en place de la nouvelle équipe de direction. À cette occasion, ma mission d'organisateur de séminaires de l'équipe MOCS a pris fin pour être reprise par les chefs d'équipe.

2. Synthèse des activités d'enseignement

Ce chapitre décrit les enseignements réalisés sur la période 2011-2021. Dans un premier temps, les thématiques enseignées sont décrites. Dans un deuxième temps, une présentation synthétique chiffrée sur les volumes est réalisée. Ensuite, les responsabilités liées à l'enseignement sont présentées. Enfin, une réflexion sur le métier d'enseignant est proposée.

2.1 Description des thématiques enseignées depuis 2011

2.1.1 Automatisme et Informatique Industrielle

Niveau : 1ère année de DUT GIM, IUT Lorient, UBS

La thématique « automatisme et informatique industrielle » est organisée autour de trois modules dont l'objectif est de connaître le fonctionnement et la structures des systèmes numériques, en particulier les systèmes automatisés. Je suis en charge de l'enseignement de la partie numération, système binaire, algèbre de Boole, logique combinatoire et logique séquentielle. Cette partie est sanctionnée d'un examen de 2 heures dont je suis responsable de l'élaboration du sujet et de la correction. Au deuxième semestre, j'enseigne la partie acquisition de données, du capteur à l'ordinateur, incluant les capteurs, conditionneurs de signaux, numérisation, filtrage, conversion analogique-numérique et numérique-analogique. Cette partie est sanctionnée d'un examen de 1 heure dont je suis responsable de l'élaboration du sujet et de la correction. J'encadre les séries de TP couvrant l'ensemble du spectre de l'automatisme, avec également la partie automate, programmation ladder et graphcet. J'ai développé sept manipulations de quatre heures pour cette thématique. Le volume d'enseignement associé à cette thématique est d'environ 90 heures eq. TD.

2.1.2 Électronique analogique

Niveau : 1ère année de DUT GIM, IUT Lorient, UBS

Le module d'électronique analogique consiste fournir les bases aux étudiants de premières années autour des notions de filtrage analogique, passif ou actif, d'amplificateur opérationnel, systèmes asservis. Lors des travaux pratiques, il s'agit d'abord de familiariser les étudiants avec les instruments d'observation et de mesures (multimètre, oscilloscopes), et le matériel électronique d'alimentation, GBF, etc. J'aide à

l'encadrement de TP mis en place par mon collègue responsable du module. Le volume d'enseignement associé à cette thématique est d'environ 48 heures eq. TD.

2.1.3 Informatique

Niveau : 2ème année de DUT GIM, IUT Lorient, UBS

L'objectif du module intitulé « INFO2 » dans la spécialité GIM est d'utiliser les outils informatiques nécessaires au traitement de données générales et professionnelles. Les compétences visées incluent l'analyse et la création d'algorithmes, ainsi que la traduction d'un algorithme simple en langage de programmation en respectant une syntaxe imposée. L'outil choisi pour l'analyse et la création d'algorithme est Algobox. Le langage de programmation enseigné est le langage C. J'ai mis en place 6 manipulations de quatre heures pour ce module. Le volume d'enseignement associé est d'environ 30 heures eq. TD.

2.1.4 Acquisition et sécurisation de données

Niveau : Licence professionnelle, IUT Lorient, UBS

Ce module reprend la partie enseignée au niveau DUT 1ère année autour de l'acquisition de données, du capteur à l'ordinateur, incluant les capteurs, conditionneurs de signaux, numérisation, filtrage, conversion analogique-numérique et numérique-analogique. Le module est agrémenté d'une partie sur la vision industrielle, incluant les capteurs (imageurs), l'éclairage, et l'ensemble du système d'acquisition, traitement, analyse, interprétation et communication. La notion de cybersécurité des systèmes industriels est également abordée. J'ai aidé au développement de quatre manipulations de quatre heures pour ce module, dont je suis aujourd'hui responsable. Le volume d'enseignement associé à cette thématique est d'environ 30 heures eq. TD.

2.1.5 Microsystème

Niveau : 3ème année d'école d'ingénieur, filière mécatronique, ENSIBS, UBS

L'objectif du module de microsystème est d'être capable de concevoir un système sur-mesure, avec des composants matériels spécifiques, et d'identifier la relation matérielle-logicielle à travers la couche d'abstraction du matériel. Ces compétences sont développées lors de la conception d'un système numérique composé d'un processeur et de plusieurs périphériques, afin de piloter un base robotique. Les travaux pratiques pour ce module s'articulent autour d'un projet de seize heures, découpé en plusieurs parties avec une complexité graduelle. La plateforme utilisée est une carte DE0-nano, avec la suite d'outils Intel/Altera associée. Le volume d'enseignement associé est d'environ 30 heures eq. TD.

2.1.6 Programmation parallèle

Niveau : Master 2, filières Systèmes Embarqués Systèmes Intégrés, et Cyber-Sécurité des Systèmes Embarqués, UBS

L'objectif du module est de découvrir la programmation parallèle à travers l'approche OpenMP. Les compétences visées sont la capacité d'analyser un code existant et d'y extraire le parallélisme, en utilisant les directives simples d'OpenMP. J'ai mis en place deux manipulations de quatre heures pour ce module. Le volume d'enseignement total est d'environ 18 heures eq. TD. Cette activité, en lien étroit avec mes activités de recherche, me permet d'identifier les étudiants présentant une appétence particulière pour ce sujet.

2.1.7 Mémoires sur puce

Niveau : Master 2, filières Systèmes Embarqués Systèmes Intégrés, et Cyber-Sécurité des Systèmes Embarqués, UBS

L'aspect « mémoires sur puce » est une intervention spécifique dans le module « Système sur puce ». Cette intervention de quatre heures (6 heures eq. TD) permet d'identifier les problématiques de la mémoire sur puces, d'étudier les différentes technologies et organisations existantes, et comprendre la hiérarchie mémoire et son fonctionnement. Cette intervention est directement en lien avec mes activités de recherche.

2.1.8 Algorithmes pour le VLSI

Niveau : Master 2, filières Systèmes Embarqués Systèmes Intégrés, UBS

Le module d'algorithmes pour le VLSI est un module dont j'ai assuré l'enseignement sur la période 2011-2016. L'objectif du module est dans un premier temps de faire connaître les différentes façons de concevoir des circuits, en se basant sur des cellules *full-custom*, *semi-custom*, *Gate Array*, ou FPGA. Le cycle de conception est présenté en identifiant les problèmes spécifiques rencontrées à chaque étape : synthèse, partitionnement, floorplanning, placement, routage, assignation des broches, compaction. Un focus est ensuite mis sur la notion de synthèse logique, avec l'utilisation des BDD (Binary Decision Diagram). Les algorithmes de partitionnement sont également regardés en détail, notamment l'incontournable Kernighan-Lin, et l'algorithme de Fiduccia et Mattheyses.

2.1.9 Description des encadrements d'étudiants

Projets tutorés

Sur la période 2012-2021, j'ai encadré les projets tutorés de première année de DUT. Cela représente quatre à six binômes tous les ans. Les sujets sont proposés par mes soins. Les étudiants peuvent aussi proposer leur sujet. Chaque binôme travaille sur un sujet différent. L'encadrement inclut un suivi régulier tout du long du deuxième semestre, une aide à la rédaction du rapport et l'élaboration de la soutenance, ainsi que la participation en tant que candide à autant de soutenances de projet (8 à 12 soutenances par an).

Exemples de sujets

- Sujets donnés : Logiciels de simulation de circuits numériques, Matrice de LED sous Arduino, Système de déverrouillage d'un coffre par code (ou par badge), l'industrie à l'ère du numérique, ...
- Sujets proposés : arc électrique musical, cloche de protection mobile pour graveuse laser, ampli guitare à lampe, indicateurs lumineux pour motards, ...

La figure 2.1 montre le prototypage réalisé dans le cadre de la cloche de protection pour graveuse laser.

Projets GIM *Période 2012-2021*
41 binômes encadrés, 75 soutenances

Depuis 2017, j'encadre les projets tutorés de la licence professionnelle IMSA (Ingénierie et Maintenance des Systèmes Automatisés). Je propose un sujet tous les ans. J'encadre un seul binôme sur ce sujet. L'encadrement inclut un suivi régulier pendant la période de projet, une aide à la rédaction du rapport et l'élaboration de la soutenance, ainsi que la participation en tant que candide à une autre soutenance de projet.

Exemples de sujets donnés

Maquette de monitoring, vision industrielle.

La figure 2.2 montre la maquette de monitoring réalisé dans le cadre des projets. L'ensemble matériel et logiciel a nécessité plusieurs binômes.

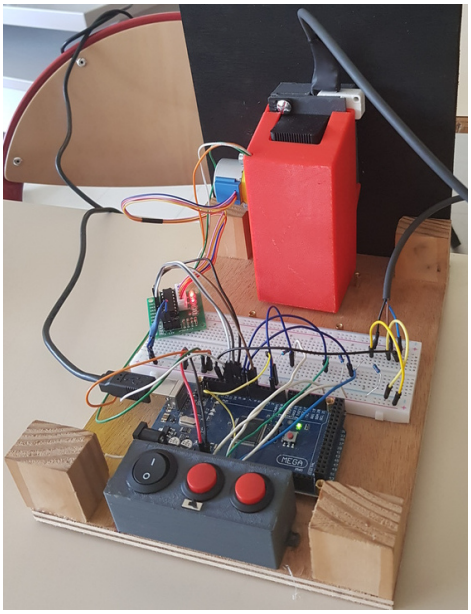


FIGURE 2.1 – Projet tutoré GIM1 - Réalisation d'une cloche de protection pour graveuse laser

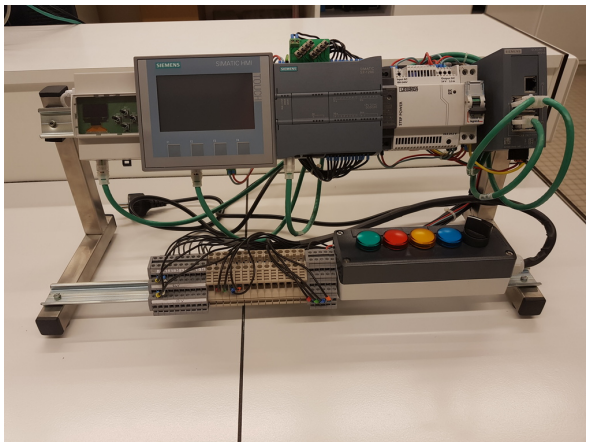


FIGURE 2.2 – Projet tutoré IMSA - Maquette de monitoring

Projets LP IMSA *Période 2017-2021*
4 binômes encadrés, 8 soutenances

Encadrement de stages

L'encadrement des stages est équitablement réparti dans l'équipe pédagogique, ce qui représente trois à quatre stagiaires à encadrer tous les ans. L'encadrement inclut une visite en entreprise, une aide à la rédaction du rapport et l'élaboration de la soutenance, ainsi que la participation en tant que candide à autant de soutenances de stage (donc 6 à 8 soutenances par an). Il est à noter que la promotion 2020 était en pleine période de stage lors du confinement de mars 2020. Tous les stagiaires que j'encadrais ont eu la chance de pouvoir continuer leur stage, soit en présentiel (domaine de l'eau, activité essentielle), soit en distanciel. Seule la visite sur site n'a pas pu avoir lieu. Les soutenances ont eu lieu en distanciel.

Stages GIM2 *Période 2011-2021*
31 stagiaires encadrés, 62 soutenances

Encadrement d'alternants

Je participe à l'encadrement de deux types d'alternants : les apprentis du DUT GIM, ainsi que des contrats de professionnalisation de la licence pro IMSA. Bien que les niveaux soient différents, l'encadrement se concrétise par les mêmes actions : une à deux visites sur site par an, interactions avec le tuteur entreprise, point d'activité après chaque période en entreprise, aide à la rédaction du rapport et l'élaboration de la soutenance, participation en tant que candide à autant de soutenances que d'étudiants encadrés. Cela représente deux soutenances (une en tant que tuteur, une en tant que candide). Un étudiant en contrat de professionnalisation est amené à présenter deux fois : une fois début avril, et une autre fois fin août. Cela représente donc 4 soutenances par an. Lors du confinement de mars 2020, l'alternant GIM a continué son activité sur site (domaine agroalimentaire, activité essentielle). L'alternant IMSA a connu une période d'arrêt d'activité, sans possibilité de travailler à distance. L'encadrement a consisté également à rassurer l'étudiant sur la non remise en cause du diplôme.

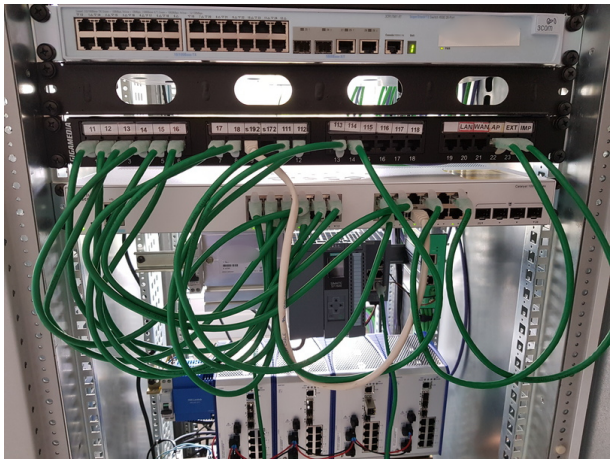


FIGURE 2.3 – Photo de l'armoire contenant une partie du matériel pour l'enseignement de la cybersécurité dans le département GIM.

Alternants *Période 2011-2021*
- 4 alternants GIM (4 × 2 ans), 32 soutenances
- 9 alternants IMSA (9 × 1 an), 36 soutenances

2.1.10 Montée en compétence « cybersécurité des systèmes industriels »

Côté enseignement, le fait notoire des trois dernières années est une volonté de montée en compétence sur l'aspect « cybersécurité des systèmes industriels », insufflée à la fois par une décision politique de l'établissement, mais surtout par une préoccupation de nos partenaires industriels. Cette montée en compétence s'est traduite par le suivi de formations et l'intégration de la notion de cybersécurité des systèmes industriels pour la licence professionnelle IMSA (Ingénierie et Maintenance des Systèmes Automatisés) dans le cadre du module « acquisition et sécurisation de données ». L'IUT a investi récemment dans du matériel pour la mise en place de manipulations spécifiques sur cette thématique. Cet investissement important a permis au département GIM de s'équiper de plusieurs automates certifiés, d'un firewall, et de plusieurs switches ethernet industriels administrables et non administrables. Ce matériel était nécessaire pour la mise en place de nouvelles manipulations relatives à l'enseignement de la cybersécurité. Six maquettes sont en cours d'élaboration pour une série des six nouveaux TP. La figure 2.3 montre l'armoire qui accueille les automates, les switches administrables, et le firewall.

Par ailleurs, l'IUT de Lorient m'a nommé « référant cybersécurité ». Mes contacts quotidiens avec mes collègues de recherche de l'équipe ARCAD, dont certains sont spécialistes de cybersécurité, me confèrent une connaissance globale de cette problématique.

2.2 Présentation synthétique des enseignements

Le tableau 2.1 présente l'enseignement effectué sur la période septembre 2011 - août 2021. Il est à noter que la somme des volumes de chaque enseignement n'est pas égale au service réalisé (heures payées), pour cause de proratisation des heures TP au delà du service dû, sauf pour la formation continue. Le volume pour chaque enseignement est donné en heure équivalent TD, avec 1h TP = 1h TD, et 1h CM = 1,5h TD.

Le tableau 2.2 présente les modules enseignés sur la période 2011-2021, le niveau, le type de formation (initiale, continue), la nature (cours, TD, TP, encadrement), les effectifs moyens des promotions concernées, ainsi que le volume cumulé sur la période. La figure 2.4 propose une autre vue de ces enseignements et montre clairement que la partie automatisme et l'informatique industrielle est la plus importante.

TABLE 2.1 – Synthèse du volume d'enseignement réalisé sur la période septembre 2011 - août 2021

Année universitaire	Volume (service réalisé)
2011 - 2012	229
2012 - 2013	247
2013 - 2014	338
2014 - 2015	282
2015 - 2016	337
2016 - 2017	340
2017 - 2018	301
2018 - 2019	288
2019 - 2020	279
2020 - 2021	292
Total	**2 933**

TABLE 2.2 – Synthèse des modules enseignés sur la période septembre 2011 - août 2021

Enseignement	Niveau	Type de formation	Nature	Effectif moyen	Volume cumulé
Automatisme et informatique industrielle	DUT1	Initiale et continue	CM/TD/TP	80	1 081
Électronique analogique	DUT1	Initiale et continue	TD/TP	81	360
Informatique	DUT2	Initiale et continue	CM/TP	53	279
Acquisition et sécurisation de données	LP	Continue	TD/TP	25	154
Microsystème	M2	Initiale	CM/TD/TP	35	248
Programmation parallèle	M2	Initiale	CM/TD/TP	20	136
Mémoires sur puce	M2	Initiale	CM	20	12
Algorithmes pour le VLSI	M2	Initiale	CM	20	36
Projets tutorés	DUT1, LP	Initiale et continue	Encadrement	12	108
Encadrement de stages	DUT2	Initiale	Encadrement	3	152
Encadrement d'alternants	DUT1, DUT2, LP	Continue	Encadrement	2	264

La figure 2.5 montre la répartition des enseignements effectués par niveau. Étant affecté à l'IUT de Lorient, il est logique qu'une écrasante majorité de mes enseignements concerne les niveaux L1 (DUT1), L2 (DUT2), et L3 (LP), qui représentent près de 85% du volume. L'enseignement au niveau M2 (UFR, ENSIBS) occupe les 15% restants. On remarque que je n'ai effectué aucun enseignement au niveau M1 (BAC+4).

La figure 2.6 montre la répartition de mes enseignements par type. L'enseignement à l'IUT étant principalement axé sur l'apprentissage par la pratique, la majorité de mon activité (plus de 50%) concerne les travaux pratiques et l'encadrement de projets. L'encadrement lors de période en entreprise lors des alternances à l'occasion des stages représente 15%. Finalement, seulement 35% de mon activité se fait sous la forme de cours magistraux ou de travaux dirigés (et encore, les heures sont données en equivalent TD, les heures de cours en présentiel sont donc artificiellement gonflées de moitié).

2.3 Responsabilités et autres activités liées à l'enseignement

2.3.1 Programme pédagogique national

Le programme pédagogique enseigné à l'IUT est défini nationalement. J'ai participé à la mise en jour en 2015 du programme pédagogique qui concerne la partie automatisme et informatique industrielle. J'ai participé à la rédaction du nouveau programme lors du passage au BUT, notamment les modules (appelés « ressources ») informatique et certains liés à l'automatisme.

2.3.2 Commission pédagogique

J'ai été membre de la commission pédagogique de l'IUT de Lorient sur la période 2017-2021. L'objectif principal de cette commission est d'organiser des « cafés pédagogiques », événements rassemblant

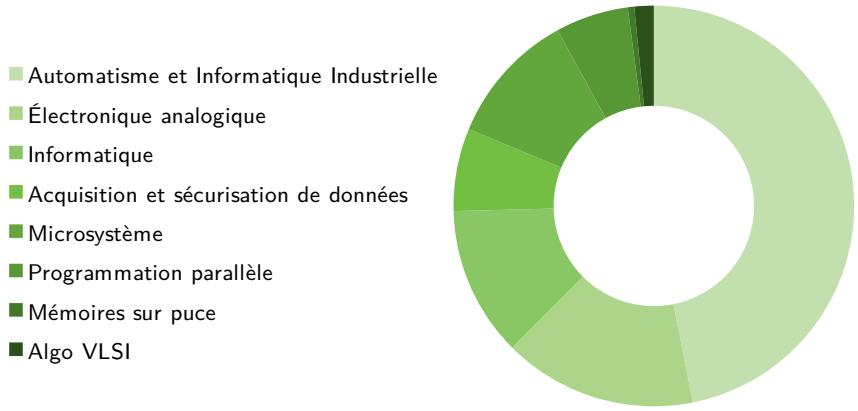


FIGURE 2.4 – Répartition des modules enseignés

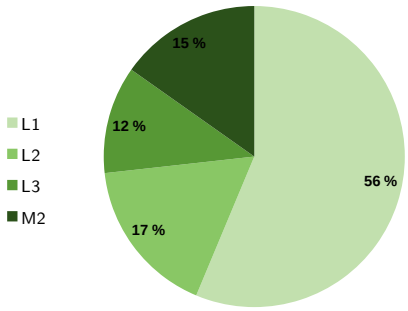


FIGURE 2.5 – Répartition des enseignements effectués par niveau (en heure eq. TD)

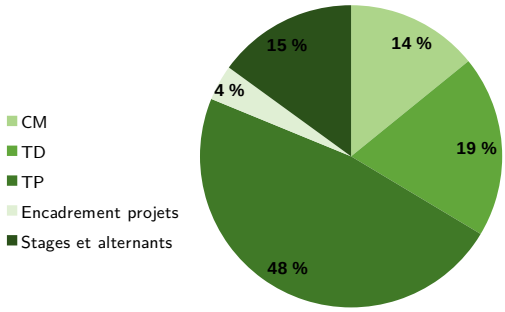


FIGURE 2.6 – Répartition des enseignements effectués par type (en heure eq. TD)

l'ensemble des enseignants de l'IUT de Lorient sur les thèmes des pratiques pédagogiques innovantes. Nous avons organisé des événements dédiés sur les sujets suivants : les quiz, les capsules vidéo, classe inversée, cours en ligne. Il faut noter que l'action « cours en ligne » s'est déroulée avant le confinement et s'est révélée très utile à cette occasion.

2.3.3 Référant cybersécurité

Depuis 2019, je suis référant cybersécurité de l'IUT de Lorient, rôle qui consiste à :

- Être le référent de l'IUT de Lorient pour tout ce qui concerne la cybersécurité, et en particulier sur la cybersécurité des systèmes industriels.
- Être force de proposition pour introduire ou développer la cybersécurité au sein des formations de l'IUT de Lorient, et en particulier dans la licence « IMSA ».
- Conseiller la direction de l'IUT sur le thème de la cybersécurité et permettre à l'IUT de développer des compétences et donc des offres de formation en cyber (particulièrement dans le cadre de la cybersécurité des systèmes industriels) aussi bien en formation initiale qu'en formation continue.
- Représenter l'IUT de Lorient au bureau de l'institut cyber de l'UBS
- Suivre des formations pour monter en compétence dans le domaine de la cybersécurité et partager en retour avec les collègues des formations concernées.
- Assister aux différentes conférences organisées autour de la cybersécurité au sein de l'UBS

2.4 Synthèse et réflexions sur le métier d'enseignant

Après 10 ans d'expérience en enseignement, je me pose encore les mêmes questions : comment transmettre la connaissance de manière efficace ? Comment améliorer mes enseignements ?

L'enjeu n'est pas le même selon le niveau enseigné. La maîtrise du contenu au niveau DUT n'est pas

une difficulté. Il s'agit d'enseigner les bases des différentes disciplines, qui par définition n'évoluent pas ou très peu. Le défi est au niveau pédagogique. Comment faire comprendre ces notions de base à des générations d'étudiants en constante évolution ? Les générations successives arrivent avec des bagages différents alors que le programme pédagogique reste le même. Dans le domaine technologique que j'enseigne, ces différences sont de manière intéressante liées à un certain contexte sociétal d'utilisation des outils numériques. Par exemple, il était quasiment acquis il y a quelques années que les étudiants connaissent une arborescence de fichiers dans un ordinateur. Avec l'utilisation massive des smartphones et la gestion transparente du stockage de l'information notamment dans le cloud, la notion d'arborescence de fichiers n'est plus acquise, et doit être revue.

L'enseignement au niveau master ou école d'ingénieur présente un autre défi. La maîtrise du contenu doit être consolidée, même au delà du périmètre défini dans le cadre du cours, pour mieux positionner le module dans un contexte plus général, et être capable de répondre aux éventuelles questions qui sortent du cadre.[1]

Du niveau BAC+1 à BAC+5, j'ai expérimenté différentes techniques, pour les CM, les TD, ou les TP, souvent après discussions et échanges avec les collègues. Pour les cours magistraux, au niveau DUT1 et DUT2, j'ai proposé des mini quiz, d'une dizaine de minutes, en fin de cours. Cela permet d'avoir un retour instantanée des notions comprises ou non en sortant du cours (mémoire court terme). J'ai même donné exactement le même quiz deux semaines après le premier (et après les TD sur ces notions, en prévenant les étudiants) pour évaluer la mémoire à long terme. Les résultats sur quelques promotions montrent que la moyenne du deuxième quiz est moins bonne que la moyenne du premier. Pour le niveau BAC+5, il m'arrive de proposer des cours très interactifs (lorsque le nombre d'étudiants le permet). Il s'agit d'échanger avec les étudiants, pour s'appuyer sur les bases déjà acquises pour faire connaître les nouvelles notions. Cette méthode marche généralement bien pour le premier cours d'un module.

Pour les TD, j'ai également utilisé la technique très classique d'envoyer une étudiante ou un étudiant au tableau, soit pour une correction interactive, soit pour écrire simplement la correction pour tout le monde pendant que j'explique individuellement. J'ai également essayé de faire travailler en petit groupe de 4/5 étudiants. Il n'y a finalement (sans surprise) pas de recette miracle. En fonction du contenu à transmettre, de l'autonomie des étudiants, et des acquis du cours magistral qui a précédé, le travail en groupe TD doit s'adapter.

Pour les TP, nous[2] avons proposé de constituer des binômes tournants. La constitution des binômes d'étudiants pour les séances de travaux pratiques est souvent laissée à la discrétion des étudiants eux-mêmes. Par ailleurs, le binôme formé lors de la première séance est inchangé pour le reste de la série de TP. Ce schéma fonctionne généralement dans le cadre de TP parallèle, chaque binôme effectuant le même sujet de TP. Pour des raisons d'encombrement et de budget, les TP dans les ateliers mécaniques, ou les TP que j'encadre en automatisme, sont des TP « tournants », chaque binôme se trouvant sur une machine différente. Par ailleurs, pour des raisons pédagogiques, il est préférable que des binômes différents soient constitués à chaque séance. On pourrait même pousser la réflexion jusqu'à affirmer que ceci est une bonne reconstitution d'un contexte professionnel, puisqu'on ne choisit pas forcément les collègues avec lesquels on travaille. Différentes contraintes doivent être respectées, avec des possibilités de variation de taille de groupes, d'ordre à respecter pour certains TP, etc. Bien évidement, un étudiant ne doit pas faire le même TP plusieurs fois, et si possible, un binôme déjà créé ne doit pas être reformé, un monôme doit être unique pour la série de TP, etc. Il peut exister des cas où (et c'est généralement le cas), en cours de série de TP, la planification est changée (par l'enseignant). Ceci arrive typiquement lorsque plusieurs étudiants sont absents. Il est alors préférable de voir un même binôme reconstitué plusieurs fois que deux monômes (bien que cette affirmation ait fait l'objet de plusieurs débats : vaut-il mieux apprendre seul qu'à deux ? Qu'en est-il de la qualité de l'encadrement lorsqu'il y a trop de monômes ?). Lorsque les classes sont grandes, l'affectation des binômes et des TP devient très vite un vrai casse-tête.

[1] ceci dit, cela m'est arrivé plusieurs fois de dire simplement « *je ne sais pas* », et de revenir au cours suivant avec la réponse à la question.

[2] sur l'idée originale de mon collègue Philippe Corfa

Le problème d'affectation de binômes s'apparente à un problème de planification d'événements sportifs, les élèves étant alors les équipes, et les manipulations les terrains de sport. Le problème de planification d'événements sportifs (SSP pour *sport scheduling problem*) est un problème classique de la littérature. Il est intéressant de noter qu'au delà de la modélisation classique d'événements sportifs, le problème d'affectation de binômes requiert des contraintes particulières. J'ai mis à profit mes compétences en programmation par contraintes pour résoudre de manière automatique le problème d'affectation de binômes. Cette méthode a même fait l'objet d'une publication en conférence (ROADEF).

On parle aujourd'hui beaucoup de pédagogie innovante, avec des approches comme la pédagogie inversée, la pédagogie innovante collaborative, l'approche par projet, les CTF (*Catch The Flag*), les *serious games*, le travail de groupe. Les sciences de l'éducation évoluent aussi, et abondent régulièrement de nouvelles théories d'apprentissage. Personnellement, je les accueille avec beaucoup d'intérêt et de curiosité. On peut se questionner sur l'organisation classique CM/TD/TP, et notamment sur l'efficacité du cours magistral. Dans son article, Philippe Meirieu se pose même la question de le supprimer, sans préconiser de supprimer toute forme de magistralité [3]. En effet, le cours magistral s'est imposé dans un temps où l'accès à la connaissance n'était possible qu'à travers les livres, et l'explication n'était possible qu'en présentiel et de manière synchronisée. Aujourd'hui, la connaissance est disponible n'importe quand sur Internet, et les différentes plateformes de diffusion foisonnent d'explications sous forme de vidéos. Il est donc possible d'accéder à la connaissance à la demande depuis chez soi, en suivant son propre rythme d'apprentissage. Cela peut-il sonner le glas du métier d'enseignant ? Au contraire. Je pense que cela renforce son utilité et sa responsabilité. En effet, Internet, c'est aussi « l'illusion de la connaissance » [1]. Internet, c'est l'information et la désinformation, la connaissance et la méconnaissance à la fois. Le rôle de l'enseignant dans ce contexte est non seulement d'alimenter le contenu (ce que beaucoup de collègues font déjà très bien par ailleurs), mais aussi, et surtout, d'être capable de diriger les étudiants vers les bonnes ressources, de trier l'information, et de maintenir l'intégrité du socle de connaissances pour les transmettre aux générations futures.

Pour améliorer la qualité de son enseignement, on peut demander un retour des étudiants. L'évaluation des enseignements est un sujet qui revient régulièrement sur la table et suscite souvent un débat dans notre communauté. Il est intéressant de regarder son impact dans les pays qui la pratique depuis plusieurs années. Ainsi, cette évaluation peut s'accompagner d'effets indésirables, comme en témoigne la mise en garde proposée dans [2]. Le risque est de se concentrer sur la satisfaction des étudiants, qui peut passer par de l'indulgence voire du laxisme vis à vis des efforts à fournir, au détriment de la transmission de la connaissance.

Références

[1] Matthieu RICARD et Bruno PATINO. *Internet, c'est l'illusion de la connaissance*. fr. URL : https://www.parismatch.com/Actu/Societe/Matthieu-Ricard-et-Bruno-Patino-Internet-c-est-l-illusion-de-la-connaissance-1788572 (visité le 21/07/2022) (cf. page 17).

[2] Nina Powell Rebekah WANIC. *Student-centred education : a philosophy most unkind*. en. Avr. 2022. URL : https://www.timeshighereducation.com/depth/student-centred-education-philosophy-most-unkind (visité le 21/07/2022) (cf. page 17).

[3] Philippe MEIRIEU. "Faut-il supprimer le cours magistral ?" In : *Cahiers pédagogiques* 424 (mai 2004), p.7-9 (cf. page 17).

3. Synthèse des activités de recherche

Ce chapitre présente une synthèse de mes activités de recherche réalisées sur la période septembre 2011 à juin 2022. La première section présente de manière synthétique trois axes de recherche qui seront ensuite développé en partie II. La suite du chapitre énumère l'ensemble de mes activités liées à la recherche : encadrement doctoral et scientifique, diffusion et rayonnement, outils logiciels, et responsabilités. Le chapitre se termine par une une réflexion sur le métier de chercheur et la liste de mes publications.

3.1 Présentation synthétique des thématiques de recherche

Mes différentes activités et contributions en recherche peuvent se réunir sous une bannière « Méthodes, outils et supports matériels pour le déploiement d'applications parallèles sur architectures parallèles ». Trois grands axes de recherche peuvent ensuite se dégager et sont présentés de manière synthétique ici. Les trois axes de recherche sont :

1. Architectures de calcul spécialisées
2. Déploiement de tâches sur architectures parallèles
3. Synchronisation de tâches sur architectures parallèles

Ces trois axes de recherche sont ensuite présentés en détails dans trois chapitres différents qui constituent le cœur de ce document.

3.1.1 Architectures de calcul spécialisées

Les architectures de calcul spécialisées que j'étudie exploitent le parallélisme au niveau instructions et au niveau données, et s'inscrivent dans la catégorie des architectures reconfigurables à gros grain (CGRA pour Coarse Grained Reconfigurable Architecture). Dans le spectre des architectures de calcul, ces solutions proposent un compromis intéressant entre flexibilité et performance. Dans nos travaux, nous proposons des architectures très faible consommation qui respectent des budgets énergétiques très serrés de l'ordre de 3 mW, pour des systèmes embarqués ou des nœuds IoT. La figure 3.1 montre une vue schématique du CGRA que nous avons développé, appelé IPA pour *Integrated Programmable Array*. La flexibilité est apportée par le côté programmable de ces architectures. Elles doivent donc être accompagnées de méthodes et outils de projection d'application (compilation), incluant les structures de contrôle, conditionnelles et itératives, ce qui fait principalement l'objet de mes travaux. La question de la projection d'application peut se formuler ainsi : comment ordonnancer dans le temps et placer dans l'espace les opérations de l'application sur les

opérateurs de l'architecture, tout en respectant les dépendances de données et de contrôle, dans un temps raisonnable, de sorte que l'application soit la plus performante possible (temps d'exécution, efficacité énergétique) ? Trouver la réponse consiste à résoudre plusieurs problèmes NP-complets, ce qui rend le sujet assez unique dans le domaine des architectures numériques de calcul.

Nos principales contributions se situent autour de notre outil logiciel de projection d'application et de l'architecture du CGRA (conception des éléments de calcul, interconnexion, etc.). Un exemple de flot de compilation est donné figue 3.2, qui prend en entrée l'application spécifiée en langage C et un modèle de l'architecture CGRA cible. L'application est ensuite représentée sur forme de CDFG (*Control Data Flow Graph*), une représentation interne sous forme de graphe permettant de regrouper le flot de contrôle et le flot de données de l'application. La compilation sur CGRA consiste à « projeter » le CDFG sur un modèle du CGRA, également représenté sous forme de graphe. Il s'agit ensuite de résoudre un problème de placement et d'ordonnancement, que nous avons choisi de résoudre conjointement. Si ce problème n'a pas de solution, il est possible de transformer le graphe de l'application à la volée (par un jeu de mouvement ou de recalcul) pour tenter à nouveau de résoudre le problème [37].

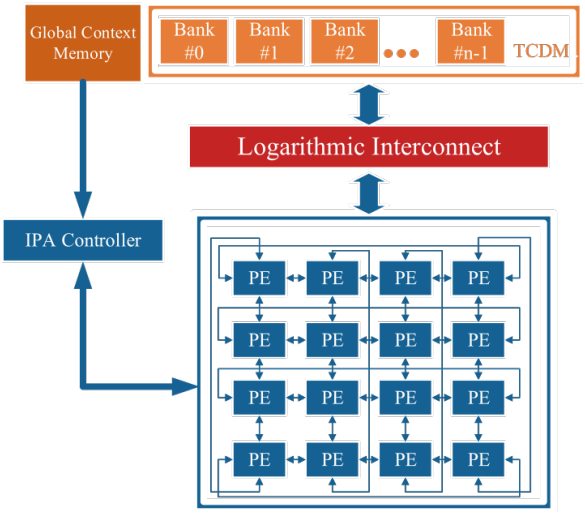


FIGURE 3.1 – IPA, *Integrated Programmable Array*, le CGRA que nous avons proposé

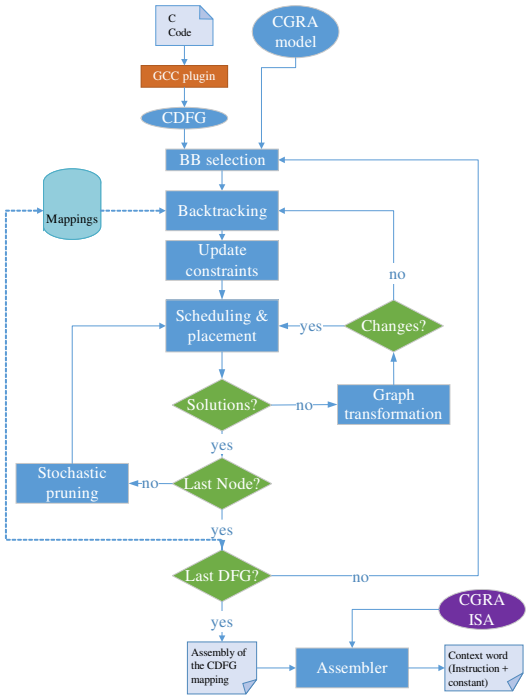


FIGURE 3.2 – Flot de projection d'une application spécifiée en langage C sur une architecture type CGRA

Une difficulté majeure est la prise en compte du flot de contrôle lors de la projection de l'application. Initialement étudié pour exécuter seulement le flot de données d'une application, aucun CGRA de l'état de l'art ne permettait de supporter le flot de contrôle (incluant le contrôle de boucles). Le support actuel du flot de contrôle (en plus du flot de données) permet d'obtenir une projection efficace. Nous avons proposé une solution pour le support de tout type de flot de contrôle (structures conditionnelles, alternatives, itératives) permettant de réduire les accès mémoire entre le CGRA et la mémoire partagée avec un processeur [25]. Nos résultats montrent un facteur d'accélération de 4 par rapport à un processeur généraliste pour un surcoût de 1.6 en surface, et un facteur de 1.8 par rapport aux techniques de l'état de l'art, sur les performances en nombre de cycles [7]. Cette projection efficace couplée à une faible consommation de l'architecture permettent un gain énergétique d'environ un ordre de grandeur par rapport à un processeur généraliste.

Dans notre approche, une projection complète est construite itérativement à partir de solutions partielles.

Mais toutes les solutions partielles n'aboutissent pas forcément à une solution complète. La principale difficulté lors du processus de projection est alors la maîtrise du nombre de solutions partielles permettant d'aboutir à une solution complète. Pour être sûr de trouver une solution, il faudrait conserver toutes les solutions partielles, ce qui en pratique n'est pas possible. En effet, il s'agit de résoudre conjointement un problème d'ordonnancement et un problème de placement qui sont tout deux des problèmes NP-complets. Le nombre de solutions partielles explose de manière exponentielle, et ce nombre devient très vite ingérable même pour de petites instances. Nous avons développé une technique de sélection intelligente pour conserver le bon nombre de solutions partielles [1, 28]. Ce nombre doit être suffisamment grand pour conserver une chance d'aboutir à une solution complète à partir d'une solution partielle, mais également suffisamment petit pour garder la maîtrise sur le nombre. Il s'agit également de s'assurer de conserver une certaine diversité dans les solutions partielles conservées. Notre technique permet de trouver des solutions pour des instances que d'autres approches de l'état de l'art ne permettent tout simplement pas d'obtenir.

Nous avons en complément proposé une approche permettant de répartir le plus uniformément possible les différents calculs sur les différentes tuiles, afin de diminuer la taille de la mémoire de configuration de chaque tuile [19]. Nous avons également proposé d'inclure les opérateurs permettant d'effectuer du calcul en virgule flottante [6], et des opérateurs pour la *transprécision* [18]. L'idée de la *transprécision* est de s'autoriser une perte de précision dans les calculs tout en gardant la dynamique des nombres, afin de réduire la consommation énergétique et d'établir un compromis précision/qualité du résultat. Nos résultats montrent un gain d'un ordre de grandeur en consommation énergétique pour un dégradation au pire de 9% de la qualité de résultats. Cet axe sera présenté dans le chapitre 4, p. 49.

Architectures de calcul spécialisées *Période 2011-2022*
Cette activité de recherche sur les CGRA et leur flot de compilation associé représente en tout :
- 3 thèses soutenues, 2 en cours
- 1 post-doc
- 4 articles de revue [1, 4, 6, 7]
- 2 brevets [10, 11]
- 10 publications en conférences [13, 16, 17, 18, 21, 25, 26, 28, 36, 37]
- plusieurs communications orales [48, 52, 27, 35]

3.1.2 Déploiement de tâches sur architectures multi-processeurs

Le parallélisme niveau tâches est exploité par des architectures multi ou many-core. Nous nous intéressons aux architectures présentant quelques cœurs à plusieurs dizaines, et plus particulièrement à l'organisation du sous-système mémoire accompagnant ces architectures. Une exploitation efficace du parallélisme inhérent aux architectures multi-processeurs peut être particulièrement laborieuse d'un point de vu logiciel, surtout lorsque les particularités de la machine sont visibles du code applicatif. Une approche alors généralement adoptée consiste à considérer une seule mémoire partagée et cohérente, agrémentée de mécanismes de synchronisation adéquats, pour limiter l'adhérence du code applicatif à la machine cible. La mémoire est alors organisée autour d'une hiérarchie mémoire, constituée de la mémoire principale et de plusieurs niveaux de caches. Un ensemble de mécanismes matériels et logiciels complexes garantissent la consistance et la cohérence des données. Nous cherchons à répondre à une première question autour de ces architectures : comment déployer une application type flot-de-données ?

Le modèle de calcul flot-de-données possède des propriétés spécifiques qu'il est intéressant d'exploiter dans le cadre du parallélisme. Une application dite flot-de-données permet d'exprimer de manière explicite plusieurs niveaux de parallélisme : parallélisme de tâches, parallélisme de données, parallélisme temporel et parallélisme spatial. Un exemple de spécification d'une application sous cette forme est présenté figure 3.3. Si le déploiement d'applications flot-de-données sur une architecture multi-processeurs paraît alors évident, il n'en est pas de même pour le déploiement des données en mémoire. En réalité, de nombreuses complications viennent contre-carrer le déploiement, et en particulier, l'hétérogénéité du

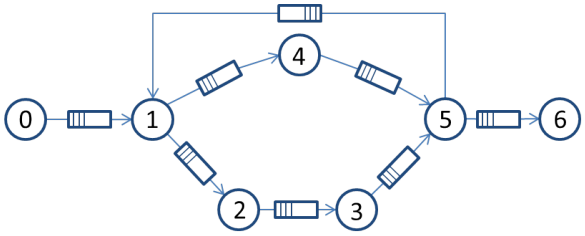


FIGURE 3.3 – Exemple d'application flot-de-données constituée de 7 tâches

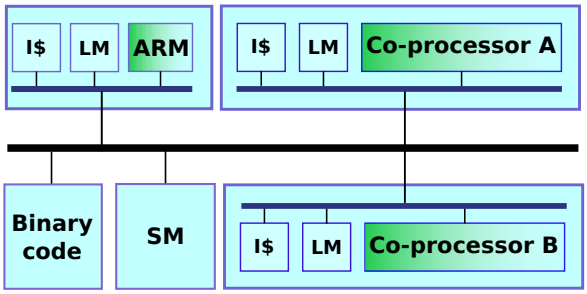


FIGURE 3.4 – Exemple d'architecture multi-processeurs hétérogène. SM : Shared Memory. LM : Local Memory. I$: Instruction Cache

temps d'exécution des tâches et des temps d'accès à la mémoire, surtout dans le cadre d'applications flot-de-données dynamiques, et l'utilisation de buffers de données suivant un accès de type FIFO (First In First Out) qui n'exploitent pas favorablement les caractéristiques des caches présents aujourd'hui dans les machines. Le déploiement de tâches flot-de-données sur architectures multi-processeurs peut se modéliser sous la forme d'un problème de partitionnement de graphe. Ce problème peut ensuite être résolu à l'aide des méthodes bien connues du domaine, comme l'algorithme de Kernighan-Lin. Nous avons proposé un algorithme, basé sur l'algorithme de Fiduccia-Mattheyses, pour le déploiement à la volée d'applications flot-de-données dynamiques [8]. Le côté dynamique des applications étudiées impose de modifier à la volée le déploiement afin de répartir équitablement la charge de travail sur les processeurs à disposition. L'approche proposée, étudiée dans le cadre d'une application de décodage vidéo, permet de maintenir la qualité de service malgré les fortes variations dans l'application. L'idée défendue dans nos travaux est que le changement d'affectation s'accompagne d'un surcoût causé par les mouvements de données et d'instructions en mémoire. Nous avons montré qu'il n'est pas toujours rentable de changer complètement le placement, comme cela est proposé dans la littérature, dans un article accepté récemment [3], et fourni dans ce document en annexe B.

Déploiement de tâches sur architectures parallèles *Période 2011-2022*
Cette activité de recherche sur le déploiement d'applications flot-de-données sur architecture multi-processeurs représente en tout :
- 1 thèse
- 1 post-doc
- 1 projet ANR PRCI (COMPA)
- 1 projet Jeunes Chercheurs du GdR ISIS (MORDRED)
- 2 articles de revue [3, 8]
- 4 publications en conférences [23, 30, 33, 34]
- plusieurs communications orales ou demos [55, 56, 57, 58, 59]

3.1.3 Synchronisation de tâches sur architectures multiprocesseurs

Nous cherchons ici à répondre à une deuxième question autour des architectures multiprocesseurs : comment garantir la synchronisation ?

Deux verrous principaux sont identifiés aujourd'hui dans le déploiement des applications parallèles : 1) l'accès aux données, 2) la synchronisation entre les tâches. La figure 3.5 montre un exemple de déploiement du graphe de la figure 3.3 sur une architecture constituée de trois processeurs, selon un algorithme d'affectation élémentaire de type First-fit. Les traits noirs représentent les dépendances de données, telles qu'elles sont imposées par la structure du graphe. Le temps étant représenté en ordonné vers le bas, il paraît évident que les tâches 0 (T0) et 1 (T1) ne peuvent en réalité pas être ordonnancées

en même temps, T1 ayant une dépendance de données sur T0. Même si dans le cadre d'une application flot-de-données, le parallélisme serait possible entre la deuxième itération de T0 et la première de T1 (parallélisme temporel), nous considérons pour plus de simplicité une unique itération du graphe. La synchronisation entre T0 et T1 est alors explicite, T1 doit attendre que T0 ait terminé (produit les données) pour pouvoir s'exécuter. L'autre type de synchronisation à observer est la synchronisation entre T0 et T3, qui sont affectés au même processeur. Même si T0 a terminé, T3 ne peut être exécutée qu'à condition que la tâche 2 (T2) ait terminée. Il faut donc ajouter de manière explicite dans le code applicatif un mécanisme de synchronisation. Dans cet exemple, finalement seul la dépendance entre T1 et T4 (symbolisée en vert) ne nécessite de synchronisation particulière. Dans tous les autres cas, une vérification préalable par un mécanisme de synchronisation est requise.

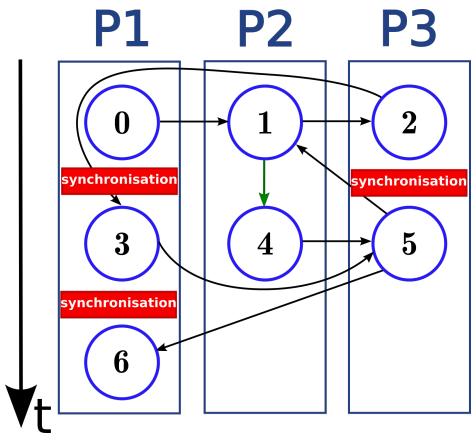


FIGURE 3.5 – Exemple d'ordonnancement des tâches de la figure 3.3 sur une architecture à 3 processeurs

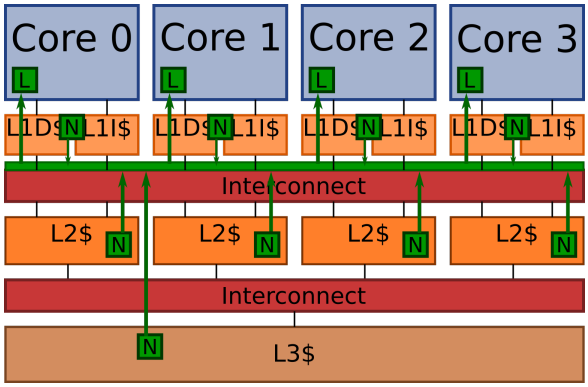


FIGURE 3.6 – Exemple de plateforme constituée de quatre processeurs avec une intégration des *notifying memories*

Nous proposons des supports matériels pour les mécanismes de synchronisation, en dehors du processeur, permettant de répondre aux problèmes d'accès aux données et de synchronisation. Premièrement, une solution logicielle/matérielle a été proposée pour distribuer les mécanismes de synchronisation dans le réseau sur puce, tout en restant compatible avec le code source originel [20]. Cette solution est composée d'un matériel spécialisé dans l'accélération des opérations de synchronisation, une mémoire privée, un pilote de système d'exploitation et une bibliothèque personnalisée. Nous ciblons la bibliothèque POSIX Threads (PThreads), largement utilisée comme bibliothèque de synchronisation native et en interne par d'autres bibliothèques telles que OpenMP ou TBB. Nous fournissons aussi des extensions destinées à accélérer encore davantage les applications dans deux cas: (i) plusieurs applications dans un contexte d'exécution fortement disputé; et (ii) sérialisation d'accès pour les variables condition dans Pthreads. Les résultats expérimentaux sur quatre applications du benchmark PARSEC fonctionnant sur un MPSoC à 64 cœurs montrent une accélération moyenne des applications de 1,57 par rapport à des solutions purement logicielles. Une accélération de 5% en plus est obtenue en utilisant notre politique d'ordonnancement *Critical Section-aware* comparée à un ordonnanceur *Round-Robin* de base.

Deuxièmement, un concept complètement nouveau baptisé « notifying memories », a été proposé pour améliorer spécifiquement la synchronisation de tâche flot-de-données [29]. L'idée s'inspire du patron de conception *Observer*, dans lequel un observateur monitore une activité et informe le sujet de tout changement. Ce patron de conception est largement connu et utilisé dans la communauté du génie logiciel. Mes compétences doubles en architecture et génie logiciel m'ont permis d'en proposer une mise en œuvre matérielle. Cette approche permet de supprimer les accès inutiles à la mémoire. En effet, il existe deux principales approches pour la synchronisation : 1) les interruptions, 2) le *polling* (une scrutation continuelle). Les interruptions ne sont pas applicables dans notre cas car une caractéristique formelle du modèle flot-de-données impose la non préemption, c'est-à-dire qu'une tâche ne peut (ne doit) pas

être interrompue (pour exécuter une autre tâche de l'application). L'approche utilisée classiquement est donc basée sur une scrutation continuelle des valeurs en mémoire pour savoir si certaines données ont changé. J'ai mené une campagne expérimentale pour déterminer le coût en accès mémoire de ces scrutations et les résultats montrent que jusqu'à 45% des accès mémoires sont inutiles. L'utilisation de notre approche permet de supprimer ces accès inutiles pour garder la mémoire disponible pour des accès « utiles », c'est-à-dire ceux qui sont nécessaires pour effectuer un calcul. Nous avons mené une première étude en guise de preuve de concept pour une architecture à mémoires distribuées dénuées de cache. Les gains étaient attendus sur le débit ou le temps d'exécution de l'application mais les meilleurs résultats sont apparus au niveau de la consommation énergétique. En effet, un premier prototype matériel pire cas (en surface) du composant a été proposé et les résultats préliminaires affichent un gain en consommation énergétique de 50%.

Dans la foulée de ces bons résultats, nous avons étudié, dans le cadre du projet ANR JCJC Nooman dont je suis porteur, l'intérêt d'une telle approche pour des architectures à mémoire partagée centralisée avec une hiérarchie de caches, telle que présentée par la figure 3.6. Ce genre d'architecture supporte classiquement la cohérence de cache, et les mécanismes de synchronisation peuvent en tirer deux bénéfices : 1) le processus de scrutation continuelle s'opère localement au niveau du cache et ne requiert pas d'accès global à la mémoire, 2) la localité temporelle habituellement observée qui veut qu'un processeur ayant accédé au mécanisme de synchronisation récemment est amené à le ré-utiliser dans un futur proche, ce qui réduit grandement le coût d'accès à la prochaine tentative. Il n'en reste pas moins que ces accès continuels occupent inutilement des lignes du cache, et ramènent parfois des données « inutiles » dans le cache au détriment d'autres données « utiles ». La figure 3.6 montre également l'intégration des composants pour l'implémentation des *Notifying memories* : le *Notifier* (N) côté mémoire, et le *Listener* (L) côté processeur.

Les différents résultats obtenus me confortent dans ma direction scientifique : il existe une large place pour l'optimisation des transferts mémoire.

Synchronisation de tâches sur architectures parallèles *Période 2011-2022*

Cette activité de recherche sur la synchronisation de tâches représente en tout :

- 2 thèses
- 2 post-doc
- 1 projet ANR JCJC
- 2 articles de revue [2, 5]
- 4 publications en conférences [15, 20, 24, 29]
- plusieurs communications orales [50, 53, 54]

3.2 Encadrement doctoral et scientifique

3.2.1 Thèses soutenues entre 2011 et 2022

Nom	PEYRET Thomas
Dates	01/11/2011 - 2/12/2014
Titre	*Architecture matérielle et flot de programmation associé pour la conception de systèmes numériques tolérants aux fautes*
Financement	bourse CEA
Situation actuelle	System Architect at Alkalee
Mon encadrement	30%
Autres encadrants	Coussy (20%), Thevenin (30%) Corre (20%)

Publications

- 2 brevets [10, 11]
- 2 conférences internationales avec actes et comité de lecture [36, 37]
- 1 conférence nationale avec comité de lecture et actes [35]

Résumé

Que ce soit dans l'automobile avec des contraintes thermiques ou dans l'aérospatial et le nucléaire soumis à des rayonnements ionisants, l'environnement entraîne l'apparition de fautes dans les systèmes électroniques. Ces fautes peuvent être transitoires ou permanentes et vont induire des résultats erronés inacceptables dans certains contextes applicatifs. L'utilisation de composants dits « rad-hard » est parfois compromise par leurs coûts élevés ou les difficultés d'approvisionnement liés aux règles d'exportation. Cette thèse propose une approche conjointe matérielle et logicielle indépendante de la technologie d'intégration permettant d'utiliser des composants numériques programmables dans des environnements susceptibles de générer des fautes. Notre proposition comporte la définition d'une Architecture Reconfigurable à Gros Grains (CGRA) capable d'exécuter des codes applicatifs complets mais aussi l'ensemble des mécanismes matériels et logiciels permettant de rendre cette architecture tolérante aux fautes. Ce résultat est obtenu par l'association de redondance et de reconfiguration dynamique du CGRA en s'appuyant sur une banque de configurations générée par une chaîne de programmation complète. Cette chaîne outillée repose sur un flot permettant de porter un code sous forme de Control and Data Flow Graph (CDFG) sur l'architecture en obtenant un grand nombre de configurations différentes et qui permet d'exploiter au mieux le potentiel de l'architecture. Les travaux, qui ont été validés aux travers d'expériences sur des applications du domaine du traitement du signal et de l'image, ont fait l'objet de publications en conférences internationales et de dépôts de brevets.

Abstract

Whether in automotive with heat stress or in aerospace and nuclear field subjected to cosmic, neutron and gamma radiation, the environment can lead to the development of faults in electronic systems. These faults, which can be transient or permanent, will lead to erroneous results that are unacceptable in some application contexts. The use of so-called rad-hard components is sometimes compromised due to their high costs and supply problems associated with export rules. This thesis proposes a joint hardware and software approach independent of integration technology for using digital programmable devices in environments that generate faults. Our approach includes the definition of a Coarse Grained Reconfigurable Architecture (CGRA) able to execute entire application code but also all the hardware and software mechanisms to make it tolerant to transient and permanent faults. This is achieved by the combination of redundancy and dynamic reconfiguration of the CGRA based on a library of configurations generated by a complete conception flow. This implemented flow relies on a flow to map a code represented as a Control and Data Flow Graph (CDFG) on the CGRA architecture by obtaining directly a large number of different configurations and allows to exploit the full potential of architecture. This work, which has been validated through experiments with applications in the field of signal and image processing, has been the subject of two publications in international conferences and of two patents.

Nom	NGO Thanh Dinh
Dates	01/02/2012 - 19/06/2015
Titre	*Runtime mapping of dynamic dataflow applications on heterogeneous multiprocessor platforms*
Financement	projet ANR COMPA
Situation actuelle	Assistant Professor at Danang University, Vietnam
Mon encadrement	60%
Autres encadrants	Diguet (40%)

Publications

- 1 article de revue d'audience internationale indexée JCR [8]
- 1 conférence internationale avec actes et comité de lecture [34]
- 1 communication par affiche [58]

Résumé

La complexité et le nombre toujours plus grandissant des applications, notamment les standards vidéo, nécessitent d'étudier des méthodes et outils pour leur déploiement sur des architectures elles aussi toujours

plus complexes. En effet, afin d'atteindre les performances requises en matière de temps d'exécution ou consommation énergétique, les architectures modernes proposent des éléments de calculs hétérogènes, où chacun est spécialisé pour une fonction précise. Cette thèse s'appuie sur le modèle flot de données pour la spécification de l'application. Ce modèle permet d'exposer explicitement le parallélisme spatial et temporel de l'application à travers un réseau d'acteurs interconnectés par des canaux de type FIFO. Les acteurs, en charge du calcul, peuvent exhiber un comportement statique ou dynamique. Les derniers standards vidéo contraignent à s'appuyer sur les modèles dynamiques pour obtenir une spécification fonctionnelle. Les besoins de calcul sont alors dépendants des données à traiter. Le déploiement d'une application dynamique ne peut donc se faire à l'aide des approches statiques existantes dans la littérature. L'objectif de cette thèse est de proposer des algorithmes efficaces permettant de déployer à la volée une application flot de données dynamique sur une architecture multiprocesseurs hétérogène. La première contribution est un algorithme qui permet de trouver rapidement une solution de déploiement de l'application. La deuxième contribution est un algorithme basé sur les mouvements pour adapter en cours d'exécution le déploiement, en réponse aux aspects dynamiques de l l'application.

Abstract

Modern multimedia applications are subject to an increasing complexity with widespread standards. This has led to the interest in dataflow approach that offers a powerful perspective on parallel computations at high level. In the meantime, the emergence of massively parallel architectures has revealed the trend towards heterogeneous Multi-Processor System-on-Chips (MPSoCs) to offer a better performance and energy tradeoff than their homogeneous counterparts. However, this also imposes challenges to the mapping of multimedia applications on such complex architectures. This thesis presents an adaptive methodology for mapping dataflow applications on heterogeneous MPSoCs. This thesis focuses on video decoders specified in RVC-CAL language, a dedicated dataflow language for video applications. Existing static approaches cannot capture all behaviors in dynamic dataflow applications. Thus, this requires to adapt the mapping according to the input data. The algorithm offers some adaptive parameters combined with our analytical communication model to improve a performance while considering load balancing. We evaluate our algorithms on a set of randomly generated benchmarks and real video decoders like MPEG4-SP and HEVC. Experimental results reveal that our mapping methodology is fast enough (in milliseconds) and the runtime remapping significantly improves the initial mapping. In the remapping process, we take the migration cost into account because the reconfiguration time also contributes to the overall performance.

Nom	VALLEJO Paola
Dates	01/10/2012 - 15/12/2015
Titre	*Réutilisation de composants logiciels pour l'outillage de DSML dans le contexte des MPSoC*
Financement	CDE UBO (Université Bretagne Occidentale)
Situation actuelle	Assistant Professor at Universidad EAFIT, Colombie
Mon encadrement	33%
Autres encadrants	Babau (33%), Kerboeuf (33%)

Publications

- 1 conférence internationale avec actes et comité de lecture [31]

Résumé

La conception d'un langage de modélisation spécifique à un domaine (DSML) implique la conception d'un outillage dédié qui met en œuvre des fonctionnalités de traitement et d'analyse pour ce langage. Dans bien des cas, les fonctionnalités à mettre en œuvre existent déjà, mais elles s'appliquent à des portions ou à des variantes du DSML que le concepteur manipule. Réutiliser ces fonctionnalités existantes est un moyen de simplifier la production de l'outillage d'un nouveau DSML. La réutilisation implique que les données du DSML soient adaptées afin de les rendre valides du point de vue de la fonctionnalité à réutiliser. Si l'adaptation est faite et les données sont placées dans le contexte de la fonctionnalité, elle

peut être réutilisée. Le résultat produit par l'outil reste dans le contexte de l'outil et il doit être adapté afin de le placer dans le contexte du DSML (migration inverse). Dans ce cadre, la réutilisation n'a de sens que si les deux adaptations de données sont peu coûteuses. L'objectif de cette thèse est de proposer un mécanisme qui intègre la migration, la réutilisation et la migration inverse. La principale contribution est une approche qui facilite la réutilisation de fonctionnalités existantes via des migrations de modèles. Cette approche facilite la production de l'outillage d'un DSML. Elle permet de faire des migrations réversibles entre deux DSMLs sémantiquement proches. L'utilisateur est guidé lors du processus de réutilisation pour fournir rapidement l'outillage complet et efficace d'un DSML. L'approche a été formalisée et appliquée à un DSML (ORCC) dans le contexte des compilateurs pour les systèmes multiprocesseur intégrés sur puce (MPSoC).

Abstract

Designers of domain specific modeling languages (DSML) must provide all the tooling of these languages. In many cases, the features to be developed already exist, but it applies to portions or variants of the DSML. One way to simplify the implementation of these features is by reusing the existing functionalities. Reuse means that DSML data must be adapted to be valid according to the functionality to be reused. If the adaptation is done and the data are placed in the context of the functionality, it can be reused. The result produced by the tool remains in the context of the tool and it must be adapted to be placed in the context of the DSML (reverse migration). In this context, reuse makes sense only if the migration and the reverse migration are not very expensive. The main objective of this thesis is to provide a mechanism to integrate the migration, the reuse and the reverse migration and apply them efficiently. The main contribution is an approach that facilitates the reuse of existing functionalities by means of model migrations. This approach facilitates the production of the tooling for a DSML. It allows reversible migration between two DSMLs semantically close. The user is guided during the reuse process to quickly provide the tooling of his DSML. The approach has been formalized et applied to a DSML (ORCC) in the context of the the compilers for multiprocessor System-on-Chip (MPSoC).

Nom	DAS Satyajit
Dates	01/10/2014 - 4/06/2018
Titre	*Architecture and Programming Model Support For Reconfigurable Accelerators in Multi-Core Embedded Systems*
Financement	thèse en co-tutelle, 50% CDE Univ. Bretagne-Sud, 50% Univ. Bologne (Italie)
Situation actuelle	Assistant Professor at IIT Palakkad, Inde
Mon encadrement	25%
Autres encadrants	Coussy (25%), Rossi (25%), Benini (25%)

Publications

- 1 article de revue d'audience internationale indexée JCR [7]
- 5 conférences internationales avec actes et comité de lecture [21, 25, 26, 28]
- 1 conférence nationale avec comité de lecture sans actes [27]

Résumé

La complexité des systèmes embarqués et des applications impose des besoins croissants en puissance de calcul et de consommation énergétique. Couplé au rendement en baisse de la technologie, le monde académique et industriel est toujours en quête d'accélérateurs matériels efficaces en énergie. L'inconvénient d'un accélérateur matériel est qu'il est non programmable, le rendant ainsi dédié à une fonction particulière. La multiplication des accélérateurs dédiés dans les systèmes sur puce conduit à une faible efficacité en surface et pose des problèmes de passage à l'échelle et d'interconnexion. Les accélérateurs programmables fournissent le bon compromis efficacité et flexibilité. Les architectures reconfigurables à gros grains (CGRA) sont composées d'éléments de calcul au niveau mot et constituent un choix prometteur d'accélérateurs programmables. Cette thèse propose d'exploiter le potentiel des architectures reconfigurables à gros grains et de pousser le matériel aux limites énergétiques dans un

flot de conception complet. Les contributions de cette thèse sont une architecture de type CGRA, appelé IPA pour Integrated Programmable Array, sa mise en œuvre et son intégration dans un système sur puce, avec le flot de compilation associé qui permet d'exploiter les caractéristiques uniques du nouveau composant, notamment sa capacité à supporter du flot de contrôle. L'efficacité de l'approche est éprouvée à travers le déploiement de plusieurs applications de traitement intensif. L'accélérateur proposé est enfin intégré à PULP, a Parallel Ultra-Low-Power Processing-Platform, pour explorer le bénéfice de ce genre de plate-forme hétérogène ultra basse consommation.

Abstract

Emerging trends in embedded systems and applications need high throughput and low power consumption. Due to the increasing demand for low power computing and diminishing returns from technology scaling, industry and academia are turning with renewed interest toward energy efficient hardware accelerators. The main drawback of hardware accelerators is that they are not programmable. Therefore, their utilization can be low as they perform one specific function and increasing the number of the accelerators in a system on chip (SoC) causes scalability issues. Programmable accelerators provide flexibility and solve the scalability issues. Coarse-Grained Reconfigurable Array (CGRA) architecture consisting of several processing elements with word level granularity is a promising choice for programmable accelerator. Inspired by the promising characteristics of programmable accelerators, potentials of CGRAs in near threshold computing platforms are studied and an end-to-end CGRA research framework is developed in this thesis. The major contributions of this framework are: CGRA design, implementation, integration in a computing system, and compilation for CGRA. First, the design and implementation of a CGRA named Integrated Programmable Array (IPA) is presented. Next, the problem of mapping applications with control and data flow onto CGRA is formulated. From this formulation, several efficient algorithms are developed using internal resources of a CGRA, with a vision for low power acceleration. The algorithms are integrated into an automated compilation flow. Finally, the IPA accelerator is augmented in PULP - a Parallel Ultra-Low-Power Processing-Platform to explore heterogeneous computing.

Nom	CATALDO Rodrigo Cadore
Dates	01/10/2016 - 16/12/2019
Titre	*SUBUTAI: Distributed synchronization primitives for legacy and novel parallel applications*
Financement	thèse en co-tutelle, 50% CDE Univ. Bretagne-Sud, 50% PUCRS (Brésil)
Situation actuelle	Ingénieur chez Huawei, Paris
Mon encadrement	33%
Autres encadrants	Diguet (33%), Marcon (33%)

Publications

- 1 article de revue d'audience internationale indexée JCR [5]
- 2 conférences internationales avec actes et comité de lecture [20, 24]
- 1 communication sans actes [54]

Résumé

Les applications parallèles sont essentielles pour utiliser efficacement la puissance de calcul des systèmes multi-processeurs (MPSoC). Cependant, ces applications ne s'adaptent pas sans effort au nombre de cœurs à cause des opérations de synchronisation qui limitent les gains de parallélisation. Les solutions existantes soit se restreignent à un sous-ensemble de primitives de synchronisation, soit nécessitent de modifier le code source de l'application, ou les deux. Nous présentons Subutai, une solution logiciel/matériel conçue pour distribuer les mécanismes de synchronisation sur le réseau sur puce, tout en restant compatible avec le code source originel. Subutai est composé d'un matériel spécialisé dans l'accélération des opérations de synchronisation, une mémoire privée, un pilote de système d'exploitation et une bibliothèque personnalisée. Nous ciblons la bibliothèque POSIX Threads (PThreads), largement utilisée comme bibliothèque de synchronisation native et en interne par d'autres bibliothèques telles que OpenMP ou TBB. Nous fournissons aussi des extensions à Subutai destinées à accélérer encore davantage

les applications dans deux cas: (i) plusieurs applications dans un contexte d'exécution fortement disputé; et (ii) sérialisation d'accès pour les variables condition dans PThreads. Les résultats expérimentaux sur quatre applications du benchmark PARSEC fonctionnant sur un MPSoC à 64 cœurs montrent une accélération moyenne des applications de 1,57× par rapport à des solutions purement logicielles. Une accélération de 5% en plus est obtenue en utilisant notre politique d'ordonnancement Critical Section-aware comparée à un ordonnanceur Round-Robin de base.

Abstract

Parallel applications are essential for efficiently using the computational power of a MultiProcessor System- on-Chip (MPSoC). Unfortunately, these applications do not scale effortlessly with the number of cores because of synchronization operations that take away valuable computational time and restrict the parallelization gains. The existing solutions either restrict the application to a subset of synchronization primitives, require refactoring the source code of it, or both. We introduce Subutai, a hardware/software architecture designed to distribute the synchronization mechanisms over the Network-on-Chip. Subutai is comprised of novel hardware specialized in accelerating synchronization operations, a small private memory for recording events, an operating system driver, and a user space custom library that supports legacy and novel parallel applications. We target the POSIX Threads (PThreads) library as it is widely used as a synchronization library, and internally by other libraries such as OpenMP and Threading Building Blocks. We also provide extensions to Subutai intended to further accelerate parallel applications in two scenarios: (i) multiple applications running in a highly-contended scheduling scenario; (ii) remove the access serialization to condition variables in PThreads. Experimental results with four applications from the PARSEC benchmark running on a 64-core MPSoC show an average application speedup of 1.57× compared with the legacy software solutions. The same applications are further sped up to 5% using our proposed Critical Section-aware scheduling policy compared to a baseline Round-Robin scheduler without any changes in the application source code.

Nom	PRASAD Rohit
Dates	15/11/2017 - 20/01/2022
Titre	*Integrated Programmable-Array accelerators to design heterogeneous ultra-low power manycore architectures*
Financement	thèse en co-tutelle, 50% CDE Univ. Bretagne-Sud, 50% Univ. Bologne (Italie)
Situation actuelle	Ingénieur au CEA, Paris Saclay
Mon encadrement	25%
Autres encadrants	Coussy (25%), Rossi (25%), Benini (25%)

Publications

- 1 article de revue d'audience internationale indexée JCR [6]
- 2 conférences internationales avec actes et comité de lecture [17, 18]

Résumé

La demande sans cesse croissante d'efficacité énergétique (EE) dans les nœuds de l'Internet des objets pousse les chercheurs et les ingénieurs à développer des solutions architecturales qui offrent à la fois une flexibilité de programmation et des performances en temps d'exécution. L'une de ces solutions est une architecture reconfigurable à gros grains (CGRA). Au cours des dernières décennies, les CGRA ont évolué et rivalisent pour devenir des accélérateurs matériels grand public, en particulier pour accélérer les applications de traitement du signal numérique. Dans le cadre de ces travaux de recherche, l'accent est mis sur l'intégration de calculs sur nombres flottants (FP) dans les CGRA. Le calcul utilisant la représentation FP nécessite de nombreux encodages et conduit à des circuits complexes pour les opérateurs FP, diminuant l'EE de l'ensemble du système. Cette thèse présente la conception d'un CGRA ultra-basse consommation avec un support natif pour le calcul FP en tirant parti d'un paradigme émergent de calcul approximatif appelé calcul de *transprécision*. Nous présentons également les contributions dans la chaîne d'outils de compilation et l'intégration du CGRA dans un système sur puce, pour envisager le CGRA proposé

comme un accélérateur matériel. Enfin, une campagne d'expérimentations utilisant des algorithmes du monde réel employés dans des applications de traitement proches capteurs sont effectués, et les résultats sont comparés avec des architectures existantes. Il est démontré empiriquement que le CGRA que nous proposons fournit de meilleurs résultats par rapport aux solutions existantes en termes de consommation, de performances et de surface.

Abstract

There is an ever-increasing demand for energy efficiency (EE) in rapidly evolving Internet-of-Things end nodes. This pushes researchers and engineers to develop solutions that provide both Application-Specific Integrated Circuit-like EE and Field-Programmable Gate Array-like flexibility. One such solution is Coarse Grain Reconfigurable Array (CGRA). Over the past decades, CGRAs have evolved and are competing to become mainstream hardware accelerators, especially for accelerating Digital Signal Processing (DSP) applications. Due to the over-specialization of computing architectures, the focus is shifting towards fitting an extensive data representation range into fewer bits, e.g., a 32-bit space can represent a more extensive data range with floating-point (FP) representation than an integer representation. Computation using FP representation requires numerous encodings and leads to complex circuits for the FP operators, decreasing the EE of the entire system. This thesis presents the design of an EE ultra-low-power CGRA with native support for FP computation by leveraging an emerging paradigm of approximate computing called *transprecision* computing. We also present the contributions in the compilation toolchain and system-level integration of CGRA in a System-on-Chip, to envision the proposed CGRA as an EE hardware accelerator. Finally, an extensive set of experiments using real-world algorithms employed in near-sensor processing applications are performed, and results are compared with state-of-the-art (SoA) architectures. It is empirically shown that our proposed CGRA provides better results w.r.t. SoA architectures in terms of power, performance, and area.

Nom	GHASEMI Alemeh
Dates	01/10/2018 - 18/05/2022
Titre	*Notifying Memories for Dataflow Applications on Shared-Memory Parallel Computer*
Financement	projet ANR NOOMAN
Situation actuelle	Ingénieure chez SMARTNVY, Grenoble
Mon encadrement	50%
Autres encadrants	Diguet (50%)

Publications

- 1 article de revue d'audience internationale indexée JCR [2]
- 1 conférence internationale avec actes et comité de lecture [15]
- 1 communication par affiche [50]

Résumé

Les machines parallèles à mémoire partagée (SMP) constituent une solution pratique pour mettre en œuvre des architectures multiprocesseurs puisqu'elles proposent une vue unifiée de la mémoire aux programmeurs ce qui facilite le développement des applications, au prix d'un mécanisme coûteux de cohérence de cache. Par ailleurs, les modèles de calcul flux-de-données offrent aux développeurs l'expressivité pour spécifier des applications complexes, en explicitant le parallélisme, permettant ainsi d'exploiter les ressources disponibles. Cependant, une implémentation d'une application flux-de-données sur SMP nécessite de nombreuses synchronisations qui impliquent la cohérence de cache et pénalisent les performances. Cette thèse s'intéresse à la compréhension des sources d'inefficacité dans l'exécution de ces applications et propose des techniques qui s'appuient sur la synchronisation exprimée dans le modèle pour en améliorer les performances. Tout d'abord, nous avons extrait les caractéristiques des applications selon plusieurs métriques, puis nous avons évalué deux techniques de gestion mémoire, Copy-on-Write et Non-Temporal Memory, pour soulager la pression sur la mémoire. Enfin, en contribution principale, nous proposons une unité matérielle spécialisée, proche de la mémoire, appelée NM4SMP (Notifying

Memory for SMP) permettant d'accélérer les applications flux-de-données en y intégrant les règles de déclenchement des calculs. L'approche est validée sur des applications dites statiques et reconfigurables. Les résultats montrent une accélération de 1,23 et une économie d'énergie de 15% pour une plateforme basée sur des processeurs Intel et plusieurs applications réelles.

Abstract

Symmetric Shared-memory multiprocessor (SMP) is the most widely used implementation of high-performance multi-core processors. It offers a uniform shared memory view that eases the development of parallel applications, but it requires cache-coherency management among the cores. Besides, dataflow Model of Computation helps the developers to specify complex applications with explicit parallelism to efficiently exploit the parallel resources of SMP. However, a dataflow application running on SMP requires high synchronization for data communication that stresses the cache memory and penalizes performance. Existing techniques for synchronization are not suited to dataflow as they are not aware of the model of computation. This thesis aims to deeply study dataflow applications' behavior on SMP and proposes novel techniques to speed them up. First, we evaluate dataflow application behavior based on several statistics. Second, we evaluate two memory techniques called Copy-on-Write and Non-Temporal Memory Transfer, to alleviate the memory footprint of dataflow applications on caches. Third, as our main contribution, we introduce an optimized hardware logic implemented near memory, Notifying Memory for SMP (NM4SMP) designed to speed up dataflow applications. Our solution improves synchronization of shared data by considering dataflow firing rules within the logic. A HW-SW co-design platform integrating NM4SMP is presented to support static and reconfigurable dataflow applications. Overall results show an average speedup of 1.23× and an average energy saving about 15%, assuming Intel SMP baseline system and real dataflow applications.

3.2.2 Thèses en cours au 01/07/2022

Nom	SUNNY Chilankamol
Date de début	30/12/2019
Titre	*Energy Efficient Loop Acceleration on CGRAs*
Financement	thèse en co-tutelle, 100% MHRD Govt of India PHD Scholarship
Mon encadrement	33%
Autres encadrants	Coussy (33%), Das (33%)

Publications

- 1 article de revue d'audience internationale indexée JCR [4]
- 1 conférence internationale avec actes et comité de lecture [16]

Nom	SAJAN Christie
Date de début	22/02/2022
Titre	*Energy Efficient Multi-core Programmable Accelerator for ULP massive edge computing*
Financement	thèse en co-tutelle, 50% ARED (Région Bretagne), 50% IIT Palakkad (Inde)
Mon encadrement	33%
Autres encadrants	Coussy (33%), Das (33%)

3.2.3 Encadrement de post-docs

Nom	RIZK Mostafa
Dates	Novembre 2014 - Juin 2016
Titre	*Projection et exécution d'applications flot-de-données sur architectures multi-processeurs basés sur des réseaux sur puce*
Financement	mixte région Bretagne/projet ANR COMPA
Situation actuelle	Post-doc at IMT Atlantique, Brest

Publications

- 1 article de revue d'audience internationale indexée JCR [3]
- 1 conférence internationale avec actes et comité de lecture [29]

Nom	VIDAL Jorgiano
Dates	Octobre 2015 - Septembre 2016
Titre	*Optimisation de code pour la conception d'accélérateurs matériels*
Financement	projet FUI SPICA
Situation actuelle	Professor at Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Brésil

Nom	DAS Satyajit
Dates	Avril 2018 - Avril 2019
Titre	*Near-Sensor Ultra-Low Power Secured Processing in IoT devices*
Financement	mixte région Bretagne/projet ANR NOOMAN
Situation actuelle	Assistant Professor at IIT Palakkad, India

Publications

- 1 article de revue d'audience internationale indexée JCR [1]
- 1 conférence internationale avec actes et comité de lecture [19]

Nom	CATALDO Rodrigo Cadore
Dates	Février 2020 - Novembre 2020
Titre	*Efficient memory strategies*
Financement	projet ANR NOOMAN
Situation actuelle	Ingénieur

Publications

- 1 article de revue d'audience internationale indexée JCR [2]
- 1 conférence internationale avec actes et comité de lecture [15]

Nom	CHATTERJEE Navonil
Dates	Décembre 2020 - Juin 2022
Titre	*Broadcast and multicast in Wireless NoC*
Financement	projet ANR RAKES
Situation actuelle	Ingénieur

Publications

- 1 article de revue d'audience internationale indexée JCR, en révision majeure
- 1 conférence internationale avec actes et comité de lecture [12]
- 1 communication par affiche [49]

Nom	RUARO Marcelo
Dates	Janvier 2021 - Février 2022
Titre	*Communication, Memory, and Energy Profiling on Many-core architectures*
Financement	projet ANR NOOMAN
Situation actuelle	Ingénieur
Publications	

- 1 article de revue d'audience internationale indexée JCR [2]
- 1 conférence internationale avec actes et comité de lecture [14]
- 1 communication par affiche [51]

3.2.4 Encadrement d'ingénieurs

Nom	EUSTACHE Yvan
Dates	mars 2015 - juin 2015
Titre	*Développement du démonstrateur du projet COMPA*
Financement	projet ANR COMPA
Situation actuelle	Ingénieur chez RTsys

Nom	LEBRETON Ghizlane
Dates	janvier 2015 - janvier 2017
Titre	*GAUT : Outil de synthèse de haut-niveau*
Financement	projet ANR SPICA
Situation actuelle	Software and Embedded System Engineer chez Marport

3.2.5 Encadrements de stage de niveau M2

- Khadimou Kassoul Diop, 2021, *Development of Parallel Applications with Reconfigurable Dataflow Graph on a Manycore Architecture*, encadrement 50%
- Hugo Miomandre, 2017, *Portage de gestionnaire d'exécution pour graphe flux-de-donnée reconfigurable sur architecture massivement parallèle Kalray MPPA*, encadrement 50%, actuellement en thèse à l'IETR INSA Rennes.
- Majed Aiaida, 2017, *Génération de configurations d'exécution d'applications sur CGRA tolérante aux fautes*, encadrement 60% (Coussy 40%)
- Merhej Christina, 2015, *Design of smart memory on multiprocessor architectures for data-flow applications*, encadrement 50% (Diguet 50%)
- Goupille-Lescar Baptiste, 2015, *Design and programmation of reconfigurable accelerators in shared-memory many-cores*, encadrement 50% (Coussy 50%)
- Sureshbabu Ramesh, 2014, *Génération des configurations d'exécution d'applications sur CGRA tolérante aux fautes*, encadrement 60% (Coussy 40%)
- El Rhomri Hajar, 2014, *Conception et simulation d'accélérateurs matériels pour décodeur vidéo MPEG4*, encadrement 100%
- Bao Chengcong, 2013, *Implementation of GPU on FPGA*, encadrement 100%

3.3 Diffusion et rayonnement

***Expertise ANR* - (3)**

- Projet JCJC AAPG2020
- Projet PRCI AAPG2021
- Projet PRCI AAPG2022

***CSI - Comité de Suivi Individuel* - (4)**

- Université de Rennes 1, Irisa/INRIA, équipe CAIRN (2017-2020)
- INSA de Rennes, IETR, équipe SYSCOM (2019-2022)
- Université de Rennes 1, Irisa/INRIA, équipe CAIRN (2020-2023)
- Université Grenoble-Alpes, TIMA, équipe SLS (2020-2023)

***Comité de sélection MCF* - (7)**

- Université de Nantes, IUT St-Nazaire (2013, 2014)
- INSA Rennes, 2015, 2018
- Université de Nantes, 2018
- Université de Rennes 1, 2022
- ENSEIRB-MATMECA Bordeaux, 2022

***Participation jurys de thèse (hors établissement)* - (2)**

- Rapporteur de la thèse de Arthur Stoutchinin sous la direction du PR. Luca Benini, Université de Bologne, Italie, décembre 2018. L'habilitation à dirigé des recherches n'est pas requise pour être rapporteur en Italie.
- Examinateur de la thèse de Thomas Baumela, sous la direction de Frédéric Pétrot et Olivier Gruber, Université de Grenoble Alpes, laboratoires TIMA et LIG, soutenue le 24/02/2021

***Cours lors d'écoles thématiques* - (2)**

- ARCHI2019 : « Mémoires sur puce : architecture et organisation »
- AMLE2022 : « On-chip memories at the edge: the edge of memories »

Comités de programme de conférences nationales

- Compas (conférence francophone en parallélisme, architecture et système) : 2014, 2016, 2018, 2019

Comités de programme de conférences internationales

- IEEE DASIP (Design and Architectures for Signal and Image Processing) : 2016, 2018, 2019, 2021, 2022
- BEC (Baltic Electronics Conference) : 2016, 2018
- SBCCI (Symposium on Integrated Circuits and System Design) : 2018, 2020, 2021, 2022
- SAMOS (International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation) : 2019, 2020, 2021, 2022
- IEEE SiPS (Workshop on Signal Processing Systems) : 2022
- RISC-V week : 2019, 2021, 2022

Relecteur de conférences internationales

- IEEE DASIP (2012, 2015, 2016, 2018, 2019, 2021, 2022), IEEE SiPS (2012, 2013, 2015, 2016, 2021, 2022), IEEE DATE (2013, 2014, 2018, 2019, 2020, 2021, 2022), ACM/IEEE DAC (2021, 2022), IEEE VLSI-SOC (2018), ACM GLSVLSI (2013, 2014, 2015), IEEE ICASSP (2016, 2019, 2020, 2021), ISVLSI (2013), FPT (2013), IEEE FCCM (2014), IEEE ASAP (2015, 2016), FPL (2015, 2016, 2017, 2018, 2021, 2022), SBCCI (2018, 2020, 2021), SAMOS (2020, 2021)

Relecteur de journaux internationaux

- ACM Transactions on Reconfigurable Technology and Systems (TRETS)
- ACM Transactions on Embedded Computing System (ACM TECS)
- Journal of Computers (JCP)
- Elsevier Microprocessors and Microsystems
- Springer Journal of Signal Processing and Systems (JSPS)
- Elsevier Journal of System Architecture (JSA)
- IEEE MICRO
- IEEE Transactions on Computers (TC)
- IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)
- Elsevier Future Generation Computer Systems (FGCS)
- Springer International Journal of Parallel Programming (IJPP)

Modération dans des conférences internationales

- IEEE DASIP 2014

- IEEE ISCAS 2018

3.3.1 Responsabilités et activités au sein des sociétés savantes

Responsabilités et activités au sein du GdR SOC2 [depuis 2017]

- Co-responsable de l'axe « *Méthodes et outils de conception, simulation, évaluation et vérification des systèmes et systèmes de systèmes* » du GdR SOC2
  - ▷ invitation des orateurs lors des colloques
  - ▷ supervision de l'organisation de journées thématiques
  - ▷ rédaction des rapports d'activité
  - ▷ co-rédaction des rapports de conjoncture
  - ▷ co-rédaction des rapports de prospective
  - ▷ relecture des posters soumis pour les colloques (20-25 posters à relire par an)
  - ▷ co-rédaction du dossier de renouvellement du GdR SOC2 2023-2027
- Co-animateur de l'axe « *Méthodes et outils de conception, simulation, évaluation et vérification des systèmes et systèmes de systèmes* » du GdR SOC2. Organisation ou co-organisation de six journées thématiques
  - ▷ Journée LLVM pour les nuls, 01/04/2022, en ligne
  - ▷ Outils pour la Synthèse de Haut Niveau, 08/04/2021, en ligne
  - ▷ Outils de prototypage virtuel de plates-formes multi/many-core, 15/05/2019, Paris
  - ▷ Journée Thématique Commune 2019 des GDR SOC et RO, 28/11/2019, Paris
  - ▷ Journée Thématique Commune 2018 des GDR SOC et RO, 10/10/2018, Paris
  - ▷ Conception basée Modèles des Systèmes de Traitement du Signal et de l'Information, 08/06/2018, Rennes
- Co-animateur du « thème de l'année » du GdR SOC2 sur le « Near-sensor computing ». Co-organisation de deux journées thématiques
  - ▷ Near Sensor Computing, 08/11/2017, Paris
  - ▷ Near Image Sensor Computing, 09/11/2018, Paris, en collaboration avec le GdR ISIS
- Point de contact avec d'autres GdR en interaction avec SOC2 : groupe OSI (Optimisation des Systèmes Intégrés) du GdR RO, groupe compilation (puis CLAP) du GdR GPL

Responsabilités et activités au sein du GdR GPL [depuis 2017]

- Co-responsable du groupe compilation du GdR GPL : supervision de l'organisation des « journées de la compilation » 2019, invitation des orateurs pour le colloque 2019, rédaction du rapport d'activité 2016-2019
- Co-porteur de nouveau groupe de travail CLAP (Compilation, Langages, Analyses, Parallélisme) : rédaction du projet de GT, cartographie des équipes liées au groupe de travail.
- Co-organisateur de séminaires en ligne
  - ▷ Compilation et vérification, 23/09/2021
  - ▷ Compiling pattern matching and reactive programming langages, 21/10/2021
  - ▷ Machine learning compiler and MLIR, 18/11/21
  - ▷ Recherche par types et Coq SydPaCC, 16/12/21

3.3.2 Organisation de colloques, conférences

Comités de pilotage de conférences nationales

- Compas (conférence francophone en parallélisme, architecture et système) depuis 2018, en tant que représentant du GdR SOC2
- Comité RISC-V depuis début 2019

Comités d'organisation de conférences nationales

- Compas (conférence francophone en parallélisme, architecture et système) 2016 : responsable du site web
- RISC-V week 2019

Comités d'organisation de conférences internationales

- DASIP 2016 : Demo Night Co-Chair
- SiPS 2017 : responsable du site web
- RISC-V week 2022 : poster chair

***Organisation d'écoles thématiques* - (1)**

- AMLE2022 : Adaptive Maching Learning at the network Edge, du 13 au 16 juin 2022, Lorient, dans le cadre de la chair Cominlabs *Design Methodologies and Tools for Adaptive Machine Learning at the Network Edge* de Shuvra S. Bhattacharyya

Organisation de sessions spéciales dans des conférences internationales

- IEEE ISCAS 2018 « Near-sensor computing »

3.3.3 Collaborations internationales

- Université de Bologne, Italie
  - ▷ Thèse en co-tutelle, Satyajit Das, co-direction du PR. Luca Benini, soutenue le 4 juin 2018.
    - ⋆ Luca Benini a reçu un *Honoris Causa* en 2015 de l'Université de Bretagne Sud dans le cadre de cette collaboration
  - ▷ Thèse en co-tutelle, Rohit Prasad, co-direction du PR. Luca Benini, soutenue le 16 janvier 2022.
- PUCRS, Brésil
  - ▷ Thèse en co-tutelle,Rodrigo Cadore Cataldo, co-direction du PR. Cesar Marcon, soutenue le 16 décembre 2019.
- University of Manchester, Royaume-Uni
  - ▷ Soumission d'un projet de mobilité H2020 MSCA-IF avec le Dr. Antoniu Pop.
- Lebanese International University, Liban
  - ▷ Accueil en tant que cherche invité de Mostafa Rizk, début décembre 2019.
- IIT Palakkad, Inde
  - ▷ Thèses en co-tutelle de Chilankamol Sunny sous la co-direction de S. Das
  - ▷ Thèses en co-tutelle de Christie Sajan sous la co-direction de S. Das

3.4 Outils logiciels

Participation active (actuelle ou passée) au développement des outils logiciels suivants :

- CGRA compiler : A tool for automatic mapping of applications onto a Coarse Grain Reconfigurable Architecture
  - ▷ Java, Model-driven engineering, design patterns, Eclipse Modeling Framework
  - ▷ Encadrement technique
- GAUT High Level Synthesis tool
  - ▷ Java, Model-driven engineering, design patterns, Eclipse Modeling Framework
  - ▷ Encadrement technique
- Gecos : Generic Compiler Suite
  - ▷ Java, Model-driven engineering, design patterns, Eclipse Modeling Framework
  - ▷ Dans le cadre de ma thèse, implication dans le développement de l'infrastructure, développement du plug-in pour ASIP (compilation, génération de code)
- PolyGraphy : Model-to-model software tools for graphs (intermediate representation of applications)
  - ▷ Java, C to Java binding
  - ▷ Développement d'une librairie dynamique d'interfaçage entre deux outils logiciels

3.5 Responsabilités scientifiques

Responsabilités au sein du laboratoire Lab-STICC

- Co-organisateur des séminaires d'équipe MOCS (Méthodes et Outils pour les Circuits et Systèmes), 2013-2020
  - ▷ 4 à 8 séminaires par an
  - ▷ L'équipe MOCS était composée d'environ 80 membres dont une quarantaine de permanents, répartie sur 3 sites (UBS Lorient, UBO Brest, ENSTA Brest)
  - ▷ Organisateur des « séminaires posters » de l'équipe, 1 fois par an, 3 éditions (2018, 2019, 2020)
    - ⋆ collecte des propositions de posters (15 à 20 posters par édition), relecture
  - ▷ Mise à jour de la page web des séminaires
- Organisateur des séminaires ponctuels de chercheurs invités

Contrats de recherche publics

- ANR PRC RAKES : responsable du partenaire, 635 k€ (213 k€ pour le Lab-STICC), 3 partenaires, 2020-2024
- ANR JCJC Nooman : porteur, 272 k€, 2018-2022
- Projet « Jeunes Chercheurs » du GdR ISIS, MORDRED, responsable du partenaire, 6 k€, 2 partenaires, 2016-2018
- Projet ANR PRCI HPeC : membre, 715 k€ (196 k€ pour le Lab-STICC), 6 partenaires, 2015-2019
- Projet « BoostEurope », région Bretagne, 1 000 €, mise en place de la collaboration avec Antoniu Pop, 2018, projet de mobilité H2020 MSCA-IF (non retenu).
- Projet ANR PRCI COMPA : membre, 800 k€ (178 k€ pour le Lab-STICC), initialement 6 partenaires, 2011-2015

3.6 Synthèse et réflexions sur le métier de chercheur

Tout part d'une idée. Une idée plus ou moins originale. Une idée plus ou moins disruptive. Une idée basée sur des hypothèses, elles-mêmes fondées sur des phénomènes ou observations connus, issus de travaux précédents. Chercher, c'est chercher à valider cette idée. Valider ou invalider cette idée, c'est toujours produire de la connaissance, et repousser les limites de la connaissance. Trouver, c'est aboutir à une conclusion. Un chercheur ne s'en contente pas. Il s'en sert pour aller plus loin. Alors oui un chercheur trouve[1], mais il n'arrête pas de chercher.

Le chercheur a souvent besoin de moyens pour développer son idée : des moyens matériels, des moyens humains. Il faut donc trouver des moyens, soit par nécessité, soit par devoir. Par nécessité pour l'achat de matériel. Par devoir car un chercheur forme par et pour la recherche, cela fait partie du métier. Une fois les moyens humains trouvés, il faut alors expliquer l'idée aux étudiantes et étudiants qui vont concrétiser les choses, et diriger les travaux pour une mise en œuvre effective. La compréhension de l'idée et son appropriation par les étudiants deviennent alors prépondérantes dans le degré d'avancement des travaux. Il en résulte des conclusions parfois tout autre que celles initialement prévues, donnant naissance à d'autres idées. C'est sans fin. À chaque découverte, des nouvelles connaissances amènent à de nouvelles questions. Inlassablement, de nouveaux moyens sont nécessaires pour chercher. Il faut donc continuellement déposer des projets, demander des financements de thèse, etc. Les demandes n'étant pas toujours fructueuses, il faut redoubler d'effort, et resoumettre des versions retravaillées des projets, dans l'espoir d'un financement. Ce mode de financement n'aide pas à la continuité des travaux.

Le recrutement des bonnes personnes pour développer le projet de recherche devient la clé de la réussite. Le contexte actuel fait qu'il est difficile de recruter. Trouver les moyens financiers pour développer un projet n'est qu'un première étape. La seconde est de trouver les forces vives qui vont pouvoir travailler à plein temps sur le projet et le concrétiser. On se pose alors la question de la formation. Il nous faut former les personnes avec le profil qui correspond aux besoins de recherche.

Depuis 2011, j'exerce ce métier de chercheur à mi-temps[2]. C'est un travail de veille scientifique et technique, de formation, de recherche de financement, d'encadrements de travaux, d'évaluation, de

[1] en réponse à la question qui m'est régulièrement posée

[2] le métier d'enseignant-chercheur étant vu administrativement comme la somme de deux métiers à mi-temps

rédaction, de relecture, d'organisation d'événements scientifiques, de vulgarisation, . . .Lors de l'année universitaire 2021-2022, j'ai été en délégation CNRS. C'était une délégation complète sur l'année, que j'ai effectuée dans mon laboratoire. J'ai donc pu découvrir le métier de chercheur à temps plein. Cette délégation a d'abord été l'occasion de finaliser et concrétiser plusieurs actions entamées avant la délégation, démarrer de nouvelles activités, mais aussi l'occasion de saisir des opportunités. Par exemple, j'ai pu soumettre en tant que porteur un projet Cominlabs qui a été retenu, j'ai organisé une école d'été à l'attention des jeunes chercheurs et jeunes chercheuses, . . .

Il y a finalement une part d'opportunisme dans le métier de chercheur. Il faut savoir saisir une occasion quand elle se présente, tout en gardant une ligne directrice sur le long terme.

3.7 Liste des publications

Cette section fournit la liste exhaustive de toutes les publications dont je suis co-auteur jusqu'à la date du 31 août 2022, y compris les publications avant ma prise de fonction à l'Université de Bretagne-Sud. La liste contient également ma thèse.

La liste exhaustive et actualisée de mes publications est disponible en ligne sur HAL avec l'idHal suivant : `kevin-martin`.

Journaux

[1] Satyajit DAS, Kevin J. M. MARTIN, Thomas PEYRET et Philippe COUSSY. "An Efficient and Flexible Stochastic CGRA Mapping Approach". In : *ACM Trans. Embed. Comput. Syst.* 8 (oct. 2022), page 24. ISSN : 1539-9087. DOI : 10.1145/3550071. URL : https://doi.org/10.1145/3550071 (cf. pages 21, 32).

[2] Alemeh GHASEMI, Marcelo RUARO, Rodrigo CATALDO, Jean-Philippe DIGUET et Kevin MARTIN. "The Impact of Cache and Dynamic Memory Management in Static Dataflow Applications". In : *Journal of Signal Processing Systems* (2022). DOI : 10.1007/s11265-021-01730-7. URL : https://hal.archives-ouvertes.fr/hal-03606524 (cf. pages 24, 30, 32, 33).

[3] Mostafa RIZK, Kevin J. M. MARTIN et Jean-Philippe DIGUET. "Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs". In : *IEEE Transactions on Parallel and Distributed Systems* (2022), pages 1-1. DOI : 10.1109/TPDS.2022.3177957 (cf. pages 22, 32).

[4] Chilankamol SUNNY, Satyajit DAS, Kevin J. M. MARTIN et Philippe COUSSY. "Energy Efficient Hardware Loop Based Optimization for CGRAs". In : *Journal of Signal Processing Systems* (mai 2022). ISSN : 1939-8115. DOI : 10.1007/s11265-022-01760-9. URL : https://doi.org/10.1007/s11265-022-01760-9 (cf. pages 21, 31).

[5] R. CATALDO, R. FERNANDES, K. J. M. MARTIN, J. SILVEIRA, G. SANCHEZ, J. SEPÚLVEDA, C. MARCON et J.-P. DIGUET. "Subutai : Speeding Up Legacy Parallel Applications Through Data Synchronization". In : *IEEE Transactions on Parallel and Distributed Systems* 32.5 (mai 2021). Journal Name : IEEE Transactions on Parallel and Distributed Systems, pages 1102-1116. ISSN : 1558-2183. DOI : 10.1109/TPDS.2020.3040066 (cf. pages 24, 28).

[6] Rohit PRASAD, Satyajit DAS, Kevin J. M. MARTIN et Philippe COUSSY. "Floating Point CGRA based Ultra-Low Power DSP Accelerator". In : *Journal of Signal Processing Systems* (jan. 2021). ISSN : 1939-8115. DOI : 10.1007/s11265-020-01630-2. URL : https://doi.org/10.1007/s11265-020-01630-2 (cf. pages 21, 29).

[7] Satyajit DAS, Kevin J. M. MARTIN, Davide ROSSI, Philippe COUSSY et Luca BENINI. "An Energy-Efficient Integrated Programmable Array Accelerator and Compilation Flow for Near-Sensor Ultralow Power Processing". In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.6 (2019), pages 1095-1108. DOI : 10.1109/TCAD.2018.2834397 (cf. pages 20, 21, 27).

[8] Thanh Dinh NGO, Kevin J. M. MARTIN et Jean-Philippe DIGUET. "Move Based Algorithm for Runtime Mapping of Dataflow Actors on Heterogeneous MPSoCs". en. In : *Journal of Signal Processing Systems* (2017), pages 1-18. ISSN : 1939-8018, 1939-8115. DOI : 10.1007/s11265-015-1088-z. (Visité le 02/12/2015) (cf. pages 22, 25).

[9] Kevin MARTIN, Christophe WOLINSKI, Krzysztof KUCHCINSKI, Antoine FLOCH et François CHAROT. "Constraint Programming Approach to Reconfigurable Processor Extension Generation and Application Compilation". In : *ACM Trans. Reconfigurable Technol. Syst.* 5.2 (juin 2012). ISSN : 1936-7406. DOI : 10.1145/2209285.2209289. URL : https://doi.org/10.1145/2209285.2209289.

Brevets

[10] Thomas PEYRET, Mathieu THEVENIN, Gwenole CORRE, Philippe COUSSY et Kevin MARTIN. “Procédé et dispositif de tolérance aux fautes sur des composants électroniques”. FR1460633 (France). Mar. 2015. URL : https://hal.archives-ouvertes.fr/hal-01166874 (visité le 13/10/2015) (cf. pages 21, 24).

[11] Thomas PEYRET, Mathieu THEVENIN, Gwenole CORRE, Kevin MARTIN et Philippe COUSSY. “Procédé et dispositif d’architecture configurable à gros grains pour exécuter l’intégralité d’un code”. FR1460631 (France). Mar. 2015. URL : https://hal.archives-ouvertes.fr/hal-01166875 (visité le 13/10/2015) (cf. pages 21, 24).

Conférences

[12] Navonil CHATTERJEE, Marcelo RUARO, Kevin J. M. MARTIN et Jean-Philippe DIGUET. “Mitigating Transceiver and Token Controller Permanent Faults in Wireless Network-on-Chip”. In : *Euromicro International Conference on Parallel, Distributed and Network-based Processing*. Special Session on On-chip parallel and network-based systems. Valladolid, Spain, mar. 2022. URL : https://hal.archives-ouvertes.fr/hal-03609150 (cf. page 32).

[13] Kevin J M MARTIN. “Twenty Years of Automated Methods for Mapping Applications on CGRA”. In : *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Lyon, France, mai 2022. DOI : 10.1109/IPDPSW55747.2022.00118. URL : https://hal.archives-ouvertes.fr/hal-03704256 (cf. page 21).

[14] Marcelo RUARO et Kevin J. M. MARTIN. “ManyGUI : A Graphical Tool to Accelerate Many-Core Debugging Through Communication, Memory, and Energy Profiling”. In : *System Engineering for Constrained Embedded Systems*. DroneSE and RAPIDO. Budapest, Hungary : Association for Computing Machinery, 2022, pages 39-46. ISBN : 9781450395663. DOI : 10.1145/3522784.3522791. URL : https://doi.org/10.1145/3522784.3522791 (cf. page 33).

[15] Alemeh GHASEMI, Rodrigo CATALDO, Jean-Philippe DIGUET et Kevin J. M. MARTIN. “On Cache Limits for Dataflow Applications and Related Efficient Memory Management Strategies”. In : *Workshop on Design and Architectures for Signal and Image Processing (14th edition)*. DASIP ’21. New York, NY, USA : Association for Computing Machinery, jan. 2021, pages 68-76. ISBN : 978-1-4503-8901-3. DOI : 10.1145/3441110.3441573. (Visité le 02/02/2021) (cf. pages 24, 30, 32).

[16] Chilankamol SUNNY, Satyajit DAS, Kevin J. M. MARTIN et Philippe COUSSY. “Hardware Based Loop Optimization for CGRA Architectures”. In : *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Sous la direction de Steven DERRIEN, Frank HANNIG, Pedro C. DINIZ et Daniel CHILLET. Cham : Springer International Publishing, 2021, pages 65-80. ISBN : 978-3-030-79025-7 (cf. pages 21, 31).

[17] Satyajit DAS, Rohit PRASAD, Kevin J. M. MARTIN et Philippe COUSSY. “Energy Efficient Acceleration Of Floating Point Applications Onto CGRA”. In : *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pages 1563-1567. DOI : 10.1109/ICASSP40776.2020.9054613 (cf. pages 21, 29).

[18] Rohit PRASAD, Satyajit DAS, Kevin MARTIN, Giuseppe TAGLIAVINI, Philippe COUSSY, Luca BENINI et Davide ROSSI. “TRANSPIRE : An energy-efficient TRANSprecision floating-point Programmable archItectuRE”. In : *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. Grenoble, France, mar. 2020, pages 1067-1072. DOI : 10.23919/DATE48585.2020.9116408. URL : https://hal.archives-ouvertes.fr/hal-02510931 (visité le 17/06/2020) (cf. pages 21, 29).

[19] S. DAS, K. J. M. MARTIN et P. COUSSY. “Context-memory Aware Mapping for Energy Efficient Acceleration with CGRAs”. In : *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2019, pages 336-341. DOI : 10.23919/DATE.2019.8715288 (cf. pages 21, 32).

[20] Rodrigo CATALDO, Ramon FERNANDES, Kevin J. M. MARTIN, Johanna SEPULVEDA, Altamiro SUSIN, César MARCON et Jean-Philippe DIGUET. “Subutai : Distributed Synchronization Primitives in NoC Interfaces for Legacy Parallel-applications”. In : *Proceedings of the 55th Annual Design Automation Conference*. DAC ’18. New York, NY, USA : ACM, 2018, 83 :1-83 :6. ISBN : 978-1-4503-5700-5. DOI : 10.1145/3195970.3196124. (Visité le 27/06/2018) (cf. pages 23, 24, 28).

[21] Satyajit DAS, Kevin J. M. MARTIN, Philippe COUSSY et Davide ROSSI. “A Heterogeneous Cluster with Reconfigurable Accelerator for Energy Efficient Near-Sensor Data Analytics”. In : *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. Mai 2018, pages 1-5. DOI : 10.1109/ISCAS.2018.8351749 (cf. pages 21, 27).

[22] Kevin MARTIN et Alban DERRIEN. “Une approche basée sur la programmation par contraintes pour résoudre le problème d’affectation de binômes”. In : *ROADEF*. Lorient, France, fév. 2018. URL : https://hal.archives-ouvertes.fr/hal-01724696 (visité le 11/06/2018).

[23] Hugo MIOMANDRE, Julien HASCOËT, Karol DESNOS, Kevin J. M. MARTIN, Benoît Dupont de DINECHIN et Jean-François NEZAN. "Embedded Runtime for Reconfigurable Dataflow Graphs on Manycore Architectures". In : *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*. PARMA-DITAM '18. New York, NY, USA : ACM, 2018, pages 51-56. ISBN : 978-1-4503-6444-7. DOI : 10.1145/3183767.3183780. (Visité le 11/06/2018) (cf. page 22).

[24] H. K. MONDAL, R. C. CATALDO, C. A. Missio MARCON, K. MARTIN, S. DEB et J. DIGUET. "Broadcast- and Power-Aware Wireless NoC for Barrier Synchronization in Parallel Computing". In : *2018 31st IEEE International System-on-Chip Conference (SOCC)*. Sept. 2018, pages 1-6. DOI : 10.1109/SOCC.2018.8618541 (cf. pages 24, 28).

[25] Satyajit DAS, Kevin J. M. MARTIN, Philippe COUSSY, Davide ROSSI et Luca BENINI. "Efficient mapping of CDFG onto coarse-grained reconfigurable array architectures". In : *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2017, pages 127-132. DOI : 10.1109/ASPDAC.2017.7858308 (cf. pages 20, 21, 27).

[26] Satyajit DAS, Davide ROSSI, Kevin J. M. MARTIN, Philippe COUSSY et Luca BENINI. "A 142MOPS/mW integrated programmable array accelerator for smart visual processing". In : *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. Mai 2017, pages 1-4. DOI : 10.1109/ISCAS.2017.8050238 (cf. pages 21, 27).

[27] Satyajit DAS, Kevin MARTIN, Thomas PEYRET et Philippe COUSSY. "Introduction d'aléas dans le processus de projection d'applications sur CGRA". In : *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS 2016)*. Lorient, France : Lab-STICC (UMR 6285), juil. 2016. URL : https://hal.archives-ouvertes.fr/hal-01347737 (visité le 03/11/2016) (cf. pages 21, 27).

[28] Satyajit DAS, Thomas PEYRET, Kevin MARTIN, Gwenolé CORRE, Mathieu THEVENIN et Philippe COUSSY. "A Scalable Design Approach to Efficiently Map Applications on CGRAs". In : *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2016, pages 655-660. DOI : 10.1109/ISVLSI.2016.54 (cf. pages 21, 27).

[29] Kevin J. M. MARTIN, Mostafa RIZK, Martha Johanna SEPULVEDA et Jean-Philippe DIGUET. "Notifying Memories : A Case-Study on Data-Flow Applications with NoC Interfaces Implementation". In : *Proceedings of the 53rd Annual Design Automation Conference*. DAC '16. Austin, Texas : Association for Computing Machinery, 2016, 35 :1-35 :6. ISBN : 9781450342360. DOI : 10.1145/2897937.2898051. URL : https://doi.org/10.1145/2897937.2898051 (visité le 11/07/2016) (cf. pages 23, 24, 32).

[30] Yaset OLIVA, Emmanuel CASSEAU, Kevin MARTIN, Jean-Philippe DIGUET, Thanh Dinh NGO et Yvan EUSTACHE. "COMPA backend : Runtime dynamique pour l'exécution de programmes flot de données sur plates-formes multiprocesseurs". In : *COMPAS 2015 : - Conférence d'informatique en Parallélisme, Architecture et Système*. Lille, France, juin 2015, pages 1-9. URL : https://hal.archives-ouvertes.fr/hal-01167037 (cf. page 22).

[31] P. VALLEJO, M. KERBOEUF, K. J. M. MARTIN et J. P. BABAU. "Improving reuse by means of asymmetrical model migrations : An application to the Orcc case study". In : *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Sept. 2015, pages 358-367. DOI : 10.1109/MODELS.2015.7338267 (cf. page 26).

[32] P. BOMEL, K. MARTIN et J.-P. DIGUET. "Virtual Devices for Hot-Pluggable Processors". In : *2014 17th Euromicro Conference on Digital System Design (DSD)*. Août 2014, pages 58-65. DOI : 10.1109/DSD.2014.84.

[33] Kevin MARTIN, Jean-Philippe DIGUET, Emmanuel CASSEAU et Yaset OLIVA. "Dataflow program implementation onto a heterogeneous multiprocessor platform". In : *METODO*. Madrid, France, oct. 2014. URL : https://hal.archives-ouvertes.fr/hal-01075481 (cf. page 22).

[34] Thanh Dinh NGO, Daniel SEPULVEDA, Kevin J. M. MARTIN et Jean-Philippe DIGUET. "Communication-model based embedded mapping of dataflow actors on heterogeneous MPSoC". In : *2014 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Oct. 2014, pages 1-8. DOI : 10.1109/DASIP.2014.7115629 (cf. pages 22, 25).

[35] Thomas PEYRET, Gwenole CORRE, Mathieu THEVENIN, Kevin MARTIN et Philippe COUSSY. "Ordonnancement, assignation et transformations dynamiques de graphe simultanés pour projeter efficacement des applications sur CGRAs". In : *ComPAS 2014 : conférence en parallélisme, architecture et systèmes*. Neuchatel, Switzerland, avr. 2014. URL : https://hal.archives-ouvertes.fr/hal-00985815 (visité le 27/02/2015) (cf. pages 21, 24).

[36] Thomas PEYRET, Gwenolé CORRE, Mathieu THEVENIN, Kevin MARTIN et Philippe COUSSY. "An Automated Design Approach to Map Applications on CGRAs". In : *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*. GLSVLSI '14. Houston, Texas, USA : Association for Computing Machinery, 2014, pages 229-230. ISBN : 9781450328166. DOI : 10.1145/2591513.2591552. URL : https://doi.org/10.1145/2591513.2591552 (cf. pages 21, 24).

[37] Thomas PEYRET, Gwenolé CORRE, Mathieu THEVENIN, Kevin MARTIN et Philippe COUSSY. "Efficient application mapping on CGRAs based on backward simultaneous scheduling/binding and dynamic graph transformations". In : *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. Juin 2014, pages 169-172. DOI : 10.1109/ASAP.2014.6868652 (cf. pages 20, 21, 24).

[38] P. BOMEL, K. MARTIN et J.-P. DIGUET. “Virtual UARTs for Reconfigurable Multi-processor Architectures”. In : *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*. Mai 2013, pages 252-259. DOI : 10.1109/IPDPSW.2013.25.

[39] Antoine FLOC’H, Tomofumi YUKI, Ali EL-MOUSSAWI, Antoine MORVAN, Kevin MARTIN, Maxime NAULLET, Mythri ALLE, Ludovic L’HOURS, Nicolas SIMON, Steven DERRIEN, François CHAROT, Christophe WOLINSKI et Olivier SENTIEYS. “GeCoS : A framework for prototyping custom hardware design flows”. In : *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2013, pages 100-105. DOI : 10.1109/SCAM.2013.6648190.

[40] Antoine FLOCH, François CHAROT, Steven DERRIEN, Kevin MARTIN, Antoine MORVAN et Christophe WOLINSKI. “Sélection d’instructions et ordonnancement parallèle simultanés pour la conception de processeurs spécialisés”. fr. In : mai 2011. URL : https://hal.inria.fr/hal-00640999/document (visité le 27/02/2015).

[41] Christophe WOLINSKI, Krzysztof KUCHCINSKI, Kevin MARTIN, Antoine FLOCH, Erwan RAFFIN et François CHAROT. “Graph Constraints in Embedded System Design”. en. In : *Worshop on Combinatorial Optimization for Embedded System Design (COESD 2010)*. Juin 2010. URL : https://hal.inria.fr/inria-00481135 (visité le 27/02/2015).

[42] Kevin MARTIN, Christophe WOLINSKI, K KUCHCINSKI, François CHAROT et Antoine FLOC’H. “Constraint-Driven Identification of Application Specific Instructions in the DURASE system”. In : *SAMOS ’09 : Proceedings of the 9th International Workshop on Embedded Computer Systems : Architectures, Modeling, and Simulation*. Samos, Greece : Springer-Verlag, 2009, pages 194-203.

[43] Kevin MARTIN, Christophe WOLINSKI, K KUCHCINSKI, François CHAROT et Antoine FLOC’H. “Constraint-Driven Instructions Selection and Application Scheduling in the DURASE system”. In : *ASAP ’09 : Proceedings of the 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. Boston, USA : IEEE Computer Society, 2009, pages 145-152. DOI : 10.1109/ASAP.2009.19.

[44] Kevin MARTIN, Christophe WOLINSKI, K KUCHCINSKI, François CHAROT et Antoine FLOC’H. “Design of Processor Accelerators with Constraints”. In : *SweConsNet Workshop*. Linköping, Sweden, 2009.

[45] Kevin MARTIN, Christophe WOLINSKI, Krzysztof KUCHCINSKI, Antoine FLOC’H et François CHAROT. “Sélection automatique d’instructions et ordonnancement d’applications basés sur la programmation par contraintes”. Français. In : *13ème Symposium en Architecture de machines (SympA’13)*. Toulouse, France, 2009. URL : http://hal.archives-ouvertes.fr/inria-00449670/en/.

[46] Christophe WOLINSKI, Krzysztof KUCHCINSKI, Kevin MARTIN, Erwan RAFFIN et François CHAROT. “How Constraints Programming Can Help You in the Generation of Optimized Application Specific Reconfigurable Processor Extensions”. Anglais. In : *International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA 2009)*. Las Vegas États-Unis d’Amérique, 2009. URL : http://hal.inria.fr/inria-00449775/en/.

[47] Kevin MARTIN et Olivier DÉFORGES. “Méthode LAR : de l’algorithme à la synthèse pour un codeur d’image embarqué faible coût”. In : *JFAAA Journées Francophones sur l’Adéquation Algorithme Architecture*. 2005.

Autres publications: posters, communications orales, etc.

[48] Kevin J. M. MARTIN et Philippe COUSSY. *Ultra-low Power Computing with CGRAs : an architecture, compilation, and application triptych*. Budapest, Hungary, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03704282 (cf. page 21).

[49] Navonil CHATTERJEE et Kevin J. M. MARTIN. *Broadcast Communication in Wireless Network-on-Chip*. Colloque du GdR SOC2. Poster. Juin 2021 (cf. page 32).

[50] Alemeh GHASEMI, Marcelo RUARO et Kevin J. M. MARTIN. *Notifying Memories for Static Dataflow Applications on Shared-Memory Processors*. Colloque du GdR SOC2. Poster. Juin 2021 (cf. pages 24, 30).

[51] Marcelo RUARO, Kevin J. M. MARTIN et Fernando G MORAES. *Software-Defined Networking for Many-cores*. Colloque du GdR SOC2. Poster. Juin 2021. URL : https://hal.archives-ouvertes.fr/hal-03294530 (cf. page 33).

[52] Satyajit DAS, Kevin J. M. MARTIN et Philippe COUSSY. *Prise en compte de la contrainte de mémoire de programme dans un flot de compilation pour CGRA*. Juil. 2019. URL : https://hal.archives-ouvertes.fr/hal-02171262 (cf. page 21).

[53] Kevin MARTIN. *Compiling for notifying memories : issues and challenges*. Présentation lors des journées nationales du GDR GPL 2019. Juin 2019. URL : https://hal.archives-ouvertes.fr/hal-02494373 (cf. page 24).

[54] Rodrigo CATALDO, Kevin MARTIN et Jean-Philippe DIGUET. *Subutai : Implantation de primitives de synchronisation au sein d’interfaces NoCs sans modification du code source*. Colloque du GdR SOC2. Poster. Juin 2018. URL : https://hal.archives-ouvertes.fr/hal-01828617 (cf. pages 24, 28).

[55] Kevin MARTIN, Thanh Dinh NGO et Jean-Philippe DIGUET. *Move Based Algorithm for Runtime Mapping of Dataflow Actors on Heterogeneous MPSoCs*. Journée du Groupe de travail Optimisation des Systèmes Intégrés (OSI) du GDR RO, Paris, France, 2017. Oct. 2017. URL : https://hal.archives-ouvertes.fr/hal-01638863 (cf. page 22).

[56] Hugo MIOMANDRE, Julien HASCOËT, Karol DESNOS, Kevin MARTIN, Benoît DUPONT DE DINECHIN et Jean-François NEZAN. *Demonstrating the SPIDER Runtime for Reconfigurable Dataflow Graphs Execution onto a DMA-based Manycore Processor*. IEEE International Workshop on Signal Processing Systems. Published : IEEE International Workshop on Signal Processing Systems. Lorient, France, oct. 2017. URL : https://hal.archives-ouvertes.fr/hal-01637300 (visité le 11/06/2018) (cf. page 22).

[57] K. J. M. MARTIN, Y. EUSTACHE, J. P. DIGUET, T. Dinh NGO, E. CASSEAU et Y. OLIVA. *Compa backend : A dynamic runtime for the execution of dataflow programs onto multi-core platforms*. Sept. 2015. DOI : 10.1109/DASIP.2015.7367246 (cf. page 22).

[58] Thanh Dinh NGO, Kevin MARTIN et Jean-Philippe DIGUET. *Déploiement à la volée de réseaux d'acteurs dataflow dynamiques sur plateforme multiprocesseurs hétérogène*. SoCSiP. Poster. Juin 2014. URL : https://hal.archives-ouvertes.fr/hal-01078215 (cf. pages 22, 25).

[59] Yaset OLIVA, Emmanuel CASSEAU, Kevin MARTIN, Pierre BOMEL, Jean-Philippe DIGUET, Hervé YVIQUEL, Mickael RAULET, Erwan RAFFIN et Laurent MORIN. *Orcc's Compa-Backend demonstration*. en. Oct. 2014. URL : https://hal.inria.fr/hal-01059858/document (visité le 27/02/2015) (cf. page 22).

[60] Kevin MARTIN, Christophe WOLINSKI, K KUCHCINSKI, François CHAROT et Antoine FLOC'H. *Extraction automatique d'instructions spécialisées en utilisant la programmation par contraintes*. GDR SoCSiP. 2009.

[61] Kevin MARTIN et François CHAROT. *Utilisation combinée d'approches statique et dynamique pour la génération d'instructions spécialisées*. GDR SoCSiP. 2008.

Cours

[62] Kevin J. M. MARTIN. "On-chip memories at the edge". Doctoral. Lecture. France, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03710634.

[63] Kevin MARTIN. "Mémoires sur puce : architecture et organisation". Doctoral. Lecture. France, mai 2019. URL : https://hal.archives-ouvertes.fr/hal-03294440.

Ma thèse

[64] Kevin MARTIN. "Génération automatique d'extensions de jeux d'instructions de processeurs". Theses. Université Rennes 1, sept. 2010. URL : https://tel.archives-ouvertes.fr/tel-00526133.

II Mapping parallel applications on parallel architectures

Introduction

How to design efficient yet flexible computing devices? This question is fuelling the hardware architecture community for decades. Few years after the first computers, made of bulky thermionic valves, the invention of the transistor revolutionised the design of computers. More efficient, smaller and affordable computers could be made, bringing more money to invest in the next technology node: Moore's law was born. During a prosperous period, the hardware architecture community, especially the processor micro-architects, designed still ever more performant circuits by increasing the frequency. But Moore's law is not a mathematical or physical law, it is an economical law, a self-made prophecy, a planning elaborated from the initial outcomes of the first products. The semi-conductor industry has been caught up by Dennard's scaling, which is a truly physical law and states that as the density of transistors increases, the consumption per transistor decreases, and the consumption per mm^2 of silicon remains constant. At constant area, computing capacities increase, improving energy efficiency. Around 2005, this nice property stopped, and power and thermal issues came into play. Dissipating the heat produced by the chip as the frequency increases became too complex. The industry thus moved to multicore processors to increase the raw performance [178].

For the last 70 years, the technology is the driving force for the hardware community, and major architectural advances went along with still ever smaller and more efficient transistors. Several architectural advances made use of parallelism. There are two main physical dimensions in parallelism: *temporal parallelism* and *spatial parallelism*. Besides, there are four types of parallelism: *Bit-Level Parallelism*, *Instruction-Level Parallelism* (ILP), *Data-Level Parallelism* (DLP), *Task-Level Parallelism* (TLP). Several architectural concepts have been proposed to make use of these types of parallelism in both dimensions, and table 1 gathers the most famous ones. At bit-level for instance, the data-path of a processor which usually corresponds to the size of the arithmetic operators has continuously spatially grown, from 4-bit to the mainstream 64-bit today. During the frequency race, designing faster processors with higher frequencies made them exploiting ILP through pipelining. In simple words, processors are made of several stages, each stage is dedicated to a simple job, the instruction goes through each stage, like in a production line. The faster the stages, the higher the throughput. This technique is thus the meeting point between *temporal parallelism* and ILP. It appears that there are conditional, alternative, or iterative instructions that insert hasards in the pipeline, and several architectural countermeasures were introduced to maintain efficiency, like data forwarding or branch prediction to name just a few.

Table 1 – Hardware techniques to make use of the different types of parallelism

	Bit-level	Instruction-Level Parallelism (ILP)	Data-Level Parallelism (DLP)	Task-Level Parallelism (TLP)
Temporal parallelism		Pipeline	MISD	Simultaneous Multi-Threading (SMT)
Spatial parallelism	Word-wise operators	VLIW, Out-of-Order (OoO), Superscalar	SIMD, SIMT, Vector processing	MIMD

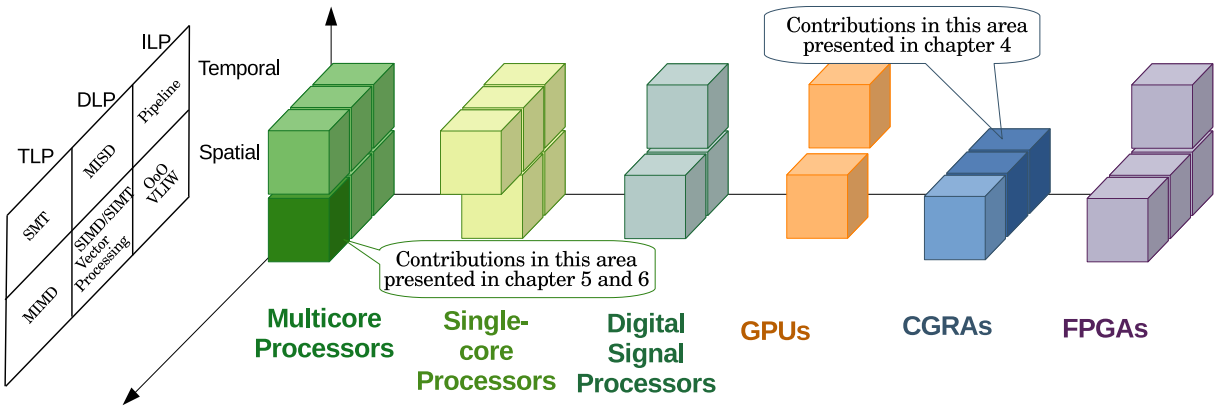


Figure 1 – Computing devices supporting several types of parallelism

The other dimension of instruction level parallelism is by spatially multiplying the issues inside a processor. This is done by VLIW (Very Long Instruction Word) processors. This technique is exploiting *spatial parallelism*. Superscalar processors and out-of-order techniques are other examples of spatial parallelism and ILP, where several instructions can be effectively executed at the same time.

The third type of parallelism is Data-Level Parallelism (DLP). An application may need to repeat the same sequence of instructions on a data set. This type of parallelism can be exploited by vector processing, or Single Instruction Multiple Data (SIMD) techniques. The widest datapath can go up to 1024 bits in leading-edge processors [12]. Graphics Processing Units (GPUs) are a good example of an architecture that intensively make use of DLP. The execution model goes beyond SIMD with the Single Instruction Multiple Threads (SIMT) style, where several instances (threads) of the same programme are run synchronously on several cores (with one thread per core), which works perfectly when the threads evolve at the same pace, but hits the limits of imbalanced branches in control-flow programmes. Pipelined SIMD operators could be the meeting point between temporal parallelism and DLP.

The higher level of parallelism is Task-Level Parallelism (TLP), which can conveniently also stand for *Thread-Level Parallelism*. Multicore processors, providing spatially several cores, are today the widest solution to propose *spatial parallelism* at task level. Simultaneous MultiThreading (SMT), with the Hyperthreading technique from Intel, can be considered as a *temporal parallelism* technique for TLP.

Figure 1 shows how existing computing devices make use of the different types of parallelism. The figure also highlights in dark colors the area which we have focused on during our work. The figure shows that our contributions mainly focus on spacial parallelism for two kinds of architectures:

1. Coarse-Grained Reconfigurable Architectures (CGRAs)
2. Multicore processors

As their names suggest, CGRAs are reconfigurable architectures at a “coarse-grained” level, meaning

that they are built upon word-wise arithmetic operators as opposed to FPGAs which are reconfigurable at bit-level. A CGRA is usually composed of several processing units (which contain the operators), tightly interconnected so that they can share a register or a register file to communicate data. A systolic array can be seen as an instance of a CGRA. CGRAs can offer a high level of spatial parallelism with their high number of elementary processing units, as evidenced by the other names found in the literature like "Reconfigurable Dataflow Accelerator" or "Reconfigurable Dataflow Architecture". CGRAs initially emerged as hardware accelerators, and had limited capabilities compared to a general purpose processor. For instance, the CGRA is not meant to run an operating system, and does not support convenient software features like dynamic memory allocation or function calls.

CGRAs can exploit ILP and DLP, and our contributions in this domain are presented in chapter 4.

Multicore processors can exploit parallelism at all level: bit-level by their word-wise arithmetic operators, ILP by pipelining and superscalar techniques, DLP by vector processing or SIMD units, and TLP by SMT or spatially distributed threads. In our work, we focused on how to spatially distribute threads or tasks on the parallel cores.

Our contributions in the domain of mapping parallel applications, especially specified through the dataflow model of computation, are presented in chapter 5.

Whatever the parallelism offered by the hardware, a theoretical limit is given by the intrinsic property of the application. As early as 1967, a formula known as Amdahl's law (named after Gene Amdahl) was proposed to determine the theoretical speedup of an application given its sequential portion. This law highlights the great impact of the sequential portion of the programme on the whole execution time. This sequential portion includes the costs of communication between the cores and coordination, and the gains obtained on these parts are expected to have a great impact on the raw performance [23]. The question is then how to express parallelism and coordination through a programming language, and how to parallelise (automatically) an application. As a complement to the mapping contributions presented in chapter 5, we studied the scalability of dataflow application and proposed synchronisation solutions to mitigate the sequential part of the programme and accelerate the execution time of parallel applications on multicore processors.

Our contributions in synchronisation solutions for multicore processors are presented in chapter 6.

Organisation of Part II

The part II of this document focuses on my scientific contributions in three main areas. It is organized in three chapters:

1. Chapter 4 presents how to exploit ILP and DLP with CGRAs.
2. Chapter 5 presents how to exploit DLP and TLP of dataflow applications on multicore processors.
3. Chapter 6 presents how to improve scalability and synchronisation in multicore processors.

A conclusion recaps the main scientific contributions presented.

4. Exploiting ILP and DLP with CGRAs

Coarse-Grained Reconfigurable Architectures (CGRAs) emerged about 30 years ago. The very first CGRAs were programmed manually. Fortunately, some compilation approaches appeared rapidly to automate the mapping process. Numerous surveys on these architectures exist. Other surveys also gather the tools and methods, and only few of them focuses on the mapping process only. This chapter presents briefly both architectures and compilation tools for CGRA. This chapter then presents our work on a fault-tolerant CGRA with its compilation tool, and an ultra-low power version for embedded systems. The third part describs how to integrate floating-point capabilities and tune the precision.

4.1 A warm-up on CGRAs

Despite three decades of constant study, Coarse Grained Reconfigurable Architectures (CGRAs) are still in 2022 the ever promising solution that did not yet meet the expected commercial success. Computer architecture is entering a new golden age [56] and CGRAs might eventually go beyond promise. CGRAs are seen as good compromise between the necessary flexibility and computing power needed by next-generation applications and the energy-efficiency required by all systems, not only the embedded ones. CGRAs gather together a huge set of possible architectures, ranging from simple organisations to complex ones [61, 208]. One may even consider also GPGPUs as part of this big family [82]. Indeed, the design space is huge and includes several architectural dimensions: processing elements and their homogeneity, interconnection network, context frame, partial reconfiguration, orchestration mechanism, design of memory hierarchy, and host-CGRA coupling just to name a few. The number of proposed architectures is simply tremendous and undoubtedly, CGRAs still keep a wide unexplored area. From its reconfigurable features, CGRAs are a key member of the reconfigurable computing family. Figure 4.1 shows the ideal trade-off between flexibility, performance, and energy efficiency that CGRAs offer compaired with other architectures.

Making an inventory of existing CGRAs is a complex and time consuming task that has been successfully

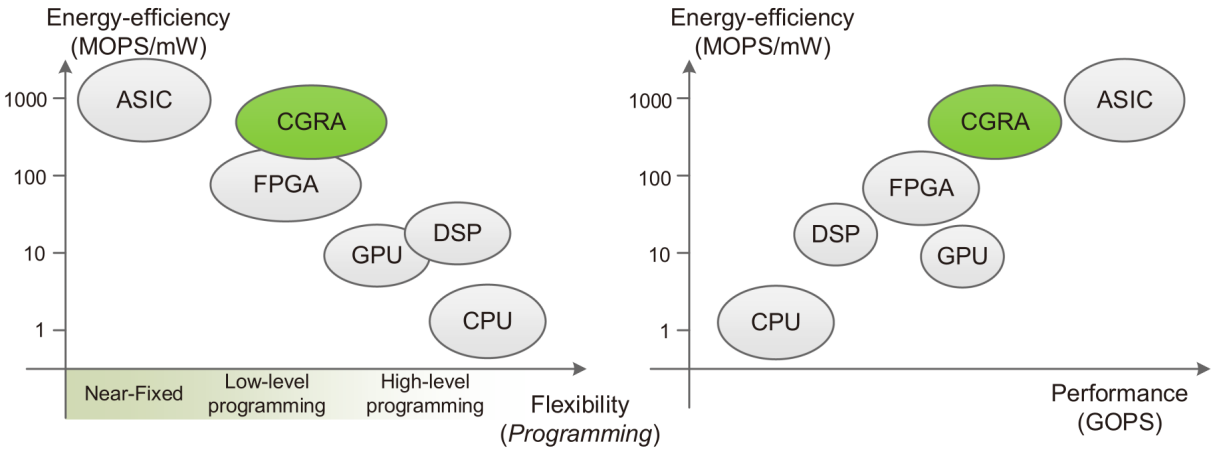


Figure 4.1 – Architecture comparison proposed in [61]

done in the past [45, 61, 105, 161, 190, 208]. Fortunately, the newcomer in the domain can restrict to reading only few papers to acquire a good overview. The Hartenstein's paper surveys the first decade of reconfigurable computing [208]. In 2010, De Sutter et al. published a book chapter detailing the architecture features of a CGRA [171]. Wijtvliet et al. review 25 years of CGRAs in 2016 [105]. The most recent surveys are provided by Liu et al. [61] and Podebas et al. [45]. Liu et al. [61] suggest another classification, complementary to the ones proposed in the previous surveys. Podebas et al. interestingly gather the published CGRAs from a performance perspective [45], and highlights by figures what is commonly accepted: CGRAs are serious competitors to GPGPUs[1]. These two last surveys point out the severe limitations that CGRAs meet like the inadequate programming model.

The abovementioned papers focus on the architectures. In order to make use of the abondant number of processing elements available, a CGRA must come along with a compiler. The very early CGRAs were programmed manually, i.e. at assembly level [201]. These first steps were important to understand how to program such an architecture and describe a systematic method that can then be automated. The automated process of programming a CGRA from a high level language falls in the compilation category. The backend part, responsible for defining the use of the hardware resources is called application mapping.

An inventory of existing mapping techniques and their associated CAD tools has also been done in the past [161, 169, 190]. Theodoridis et al. present the CAD tools along with the CGRAs up to 2007 [190]. In 2011, Choi [161] wrote a survey that combines both architecture and application mapping. These papers present first the architecture, and their associated mapping flow individually. In [169], a survey on compiling for reconfigurable computing architectures covers the broad range of reconfigurable computing, including FPGAs, up to 2010. The common features in the compiler are described, and then some dedicated compilers are presented. I have identified a lack in the literature on papers focusing on automated methods for mapping on CGRAs only, and including the last decade (2010-2020) of research on that topic. This is why I wrote the paper "Twenty Years of Automated Methods for Mapping Applications on CGRA" [11], which content is largely used in the beginning of this chapter. From the early first papers [202, 207, 212] to the latest publications on the topic [22, 24, 27], the paper paints a picture of two decades of CGRA mapping techniques, and proposes a classification. It also proposes a terminology to clearly state the problem, and extracts a general problem formulation. The paper concludes with the research challenges to be taken up.

4.1.1 Architectures

Figure 4.2, taken from [45], presents a simple CGRA which contains the minimal components of a basic CGRA. A CGRA is a set of processing elements (PEs), also called reconfigurable cells (RCs), or *tile*, or

[1] provided that we do not consider GPGPUs as part of the big CGRA family

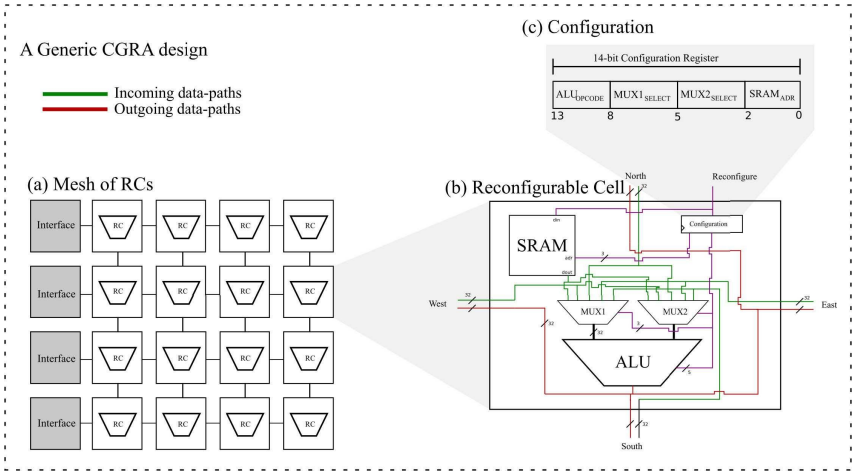


Figure 4.2 – Illustration of a simple CGRA taken from [45], showing the mesh topology (a), the internal architecture of the reconfigurable Cell, RC (b), and an example of the configuration register (c)

functional unit (FU). The term cell might be more generic, as in some CGRAs, the cells are heterogeneous, composed of computation unit, or memory units. This set of cells is usually placed as a two dimensions array, where the cells are interconnected through point to point connections, or more complex topologies. The interested reader is invited to read the dedicated papers for more details [16, 45, 61, 190]. The key element to highlight is that a CGRA exposes both spatial and temporal parallelism.

? Array or Architecture?

In the literature, the 'A' of CGRA sometimes stands for "Array", sometimes for "Architecture". Both cases make sense. A CGRA is typically organized around an *array* of cells, but the word *architecture* encompasses all kind of organisations, not only array-based.

A CGRA is a *reconfigurable* architecture. As such, it relies on *configurations*. The term *context* or *control* are also commonly found in the literature to mean a configuration. Some authors may even use the term *instruction*. A newcomer might wonder what is the difference between a *configuration*, a *context*, and an *instruction* of a CGRA. The difference lies in the hardware that allows to reconfigure the architecture. A *configuration* must store in a memory all the values of a set of signals that select the correct input of a multiplexer. A *context* is such a structure that contains all the *raw* values. An *instruction* can be seen as a *condensed* representation of a context. An instruction needs to go through a decoder whose outputs drive the multiplexers. Deducing that a processor is a reconfigurable architecture is a precocious conclusion that we cannot draw though. But whether it be a *context* or an *instruction*, the importance from the compilation point of view is to know what to produce as the format defines the contract between the hardware and the software to reach a valid execution.

Coupling a CGRA with a host CPU

The way a CGRA is coupled with a CPU can follow different schemes. A CGRA can be a fully standalone computing device, or tightly or loosely coupled with a host CPU. When tightly coupled, the CGRA and the CPU share a register file, like in ADRES [187]. When loosely coupled, the CGRA and the CPU have communication means. The on-chip memory, cache or scratch-pad memory, can be shared between the CGRA and the CPU. A CGRA can also have access to the off-chip memory through a controller. The degree of coupling has a direct impact on the reconfigurability (how often a reconfiguration takes place), the expected performance, and the compiler.

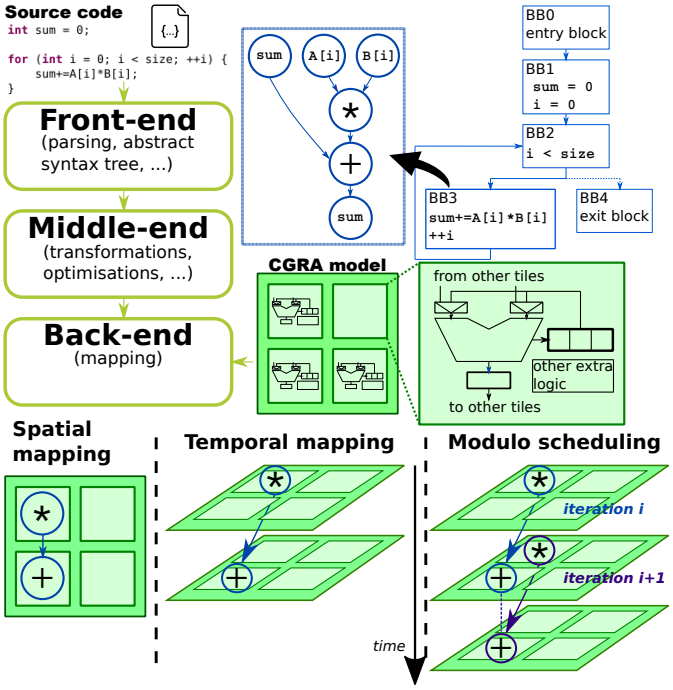


Figure 4.3 – Classical compilation flow for CGRAs

Address Generation Units

In traditional signal processing applications, the data are arranged and accessed through arrays. A given element of the array is pointed to with an address. Usually accessed through the index of a loop, the address of a given element in the array is obtained after a set of additions/subtractions and/or multiplications/divisions. In [66], the authors observed that a substantial percentage of the cells (and computation time) is dedicated to compute the address, from 20% to 80%. An Address Generation Unit (AGU) is a component embedded in the cell of the CGRA dedicated to generating the address and acts as a hardware accelerator. In [66, 176], the authors propose to embed such an component in the tiles of the CGRA, reaching a 3× speed-up.

Like any other hardware resources available in the CGRA, a compiler is needed to automatically make use of the AGU.

4.1.2 Compilers

Compilation is an automated process that takes an input source code and transforms it into an equivalent *binary* code, executable by a given architecture. Figure 4.3 shows a typical compilation flow for CGRAs. A compiler is conceptually composed of three main steps: (1) the front-end, in charge of parsing the source code and producing an equivalent intermediate representation (IR), (2) the middle-end, where some optimization passes may occur on the IR[2], and (3) the back-end, responsible for producing the binary code from the IR. Thus the back-end must *know* the target architecture. The specific features of the early CGRAs were *hardcoded* in their own compiler, some techniques being specific to a very particular hardware and hardly reusable. This is why the previous surveys present individually the compilers [161, 169, 190], as they are all tailored to a specific target. Designing a *retargetable* compiler for CGRAs is still an open issue today.

The internal (or intermediate) representation (IR) of a compiler is usually in the form of a graph. A Data Flow Graph (DFG) is a graph whose nodes represent operations and whose edges are the data dependencies between the operations. A DFG is embedded in a basic block, such that a basic block has a single entry and single exit. Figure 4.3 shows an example of a DFG inside a basic block (BB3). A Control

[2] in real life there are multiple IRs according to the optimisation to perform

Flow Graph (CFG) is a graph whose nodes are basic blocks and whose edges are the control dependencies between the basic blocks. The combination of the two forms a CDFG (Control Data Flow Graph). An application specified in a given language can thus be represented in the form of a graph, where the nodes are the operations, and the edges are the dependencies (control or data).

The *mapping* is the main step in the back-end. The word *mapping* can designate both the process and the output of the process. For a spatial CGRA, the mapping process amounts to solving the binding problem. For temporal CGRA, the mapping process must solve both binding and scheduling problems. When the problem is solved, the output of the process is a *valid mapping*, i.e. a binding (and scheduling) of operations of the application on the hardware resources while guaranteeing the dependencies. Figure 4.3 shows a *spatial mapping* and a *temporal mapping* of a simple dot-product input source code. Spatial mapping is also sometimes referred to as *straight forward mapping*.

Historically, CGRA mapping is the meeting point between VLIW compilation, and FPGA place-and-route. The difference with VLIW compilation is the direct communication possibilities offered by the CGRAs between the different cells. VLIW processors share data through a register file only. The difference with FPGA place-and-route is the granularity of the processing elements and a usually less flexible interconnect.

? Binding or placing?

The word *binding* holds the idea of "tying" things together (e.g. an operation to a computation cell), whereas *placing* let think a little freedom about the spatial location. Both terms are equally used in the literature for the same meaning.

Software pipelining is a general technique for overlapping loop iterations, and allows for exposing spacial parallelism available at loop level. *Modulo scheduling* is the most commonly used technique for software pipelining, especially in the CGRA domain. In [185], the authors define clearly the goal: "*The objective of modulo-scheduling is to engineer a schedule for one iteration of a loop such that this same schedule can be initiated at regular, as short as possible, intervals, taking into account data dependencies and resource constraints. This interval in terms of cycles is termed initiation interval (II)*". Since the II directly defines the performance, the quest of the minimum II became the main motivation of many works. Figure 4.3 shows an example of modulo scheduling for the dot-product. The II in the example is one (the best reachable II), and the figure clearly shows that two different iterations of the loop are being processed at the same time. Most of the existing mapping approaches focused only on modulo scheduling and did not consider the whole application, including its control flow.

The mapping problem

The mapping problem can be modeled in several ways. It can be seen as the "multi-processor scheduling problem" which is NP-complete [220, SS13], or through graph representation by solving yet again NP-complete problems like subgraph isomorphism [220, GT48] or largest common subgraph [220, GT49].

The mapping problem combines thus several NP-complete problems: scheduling and binding, and potentially also the register allocation problem. This raises CGRA compilation as a unique scientific problem and main challenge, because mapping might fail [73, 100, 181], which is of course inconceivable from the user point of view. To this end, for instance, HiMap [34] is an iterative algorithm that terminates when a valid mapping is found. A single formalisation of the mapping problem is not possible, as it is specific to the architecture model and the execution model considered. The interested reader can refer to other papers where the authors clearly formalized their problem [116, 181].

A nice definition is given in [61]: "*the mapping of a CGRA is actually equivalent to identifying the spatial and temporal coordinates of every node and arc in the control/data flow graph (CDFG). Compilers are responsible for making this arrangement.*" We may add that the temporal coordinate system is often called the *time extended CGRA* (TEC) [152], or the *time-space graph* [184]. The challenge is reminded by

Table 4.1 – A review of binding and scheduling techniques for automated spatial and temporal mapping of applications on CGRAs.

	Approximate methods			Exact methods	
	Heuristics	Meta-heuristics		ILP/B&B	CSP
		Population-based	Local search		
Spatial mapping	[26, 181, 189]	GA [43]	SA [21, 50]	ILP [69, 140, 181]	
Temporal mapping	[18, 24, 34, 51, 72, 74, 186, 212]	GA [148]	SA [206]	ILP [192] B&B [76]	CP [173] SAT [27] SMT [55]
Binding	[100, 115, 126, 137, 152, 202]	QEA [164]	SA [130, 179, 189]	ILP [22, 164]	
Scheduling	[37, 51, 100, 130, 137, 152, 164, 199]			ILP [22, 28]	

Chen et al. [116]: to provide *high quality solution with fast compilation time*. The mapping problem can be summarized as follows:

Mapping problem definition

Bind in place and schedule in time operations of the application on the CGRA while guaranteeing the dependencies and in a short time, such that the application executes as fast as possible.

Compilation techniques

This section presents a round-trip of proposed methods to solving the CGRA mapping problem. As the application is composed of data-flow parts, and control-flow parts, some methods have been devised specifically for each part. The section ends with an overview of the scientific production of the last two decades.

Data-flow mapping

All the papers about CGRA mapping include a technique to map the data-flow part of the application. Some methods follow a place-and-route similar to what is done in FPGAs. Some other methods formalized the problem to delegate to a solver. Since the mapping problem is an NP-complete problem, researchers naturally looked after techniques provided by the operational research or graph theory domains. These have been extensively used to solve the data-flow mapping problem.

Table 4.1 gathers all the techniques used to solve the mapping problem, for spatial or temporal architectures. The different techniques used are presented in four main columns: (1) heuristics, (2) meta-heuristics, (3) ILP or Branch and Bound (B&B) methods, (4) Constraint Satisfaction Problems (CSP). The heuristics encompass all the techniques specifically designed for the given problem. The meta-heuristics form a family of optimisation algorithms, and the table further divides it into two families: population-based techniques like Genetic Algorithms (GA) or quantum-inspired evolutionary algorithm (QEA), and local search techniques like Simulated Annealing (SA). The exact methods include Integer Linear Programming (ILP) and branch and bound on one side, and techniques that model the mapping problem as a constraint satisfaction problem. This problem is then solved through constraint programming (CP), SAT (Boolean satisfiability), or SMT (Satisfiability Modulo Theories). Please note that all the papers cited do not appear in the table, as some of them rely on already referenced papers. For instance, the approach presented by De Sutter et al. [185] relies on DRESC compiler [206], which already appears in the table.

The table shows that the topic has been widely covered by the researchers, with higher efforts in heuristics.

Control-flow mapping

Mapping the control-flow graph raises another difficulty. A solution adopted in many cases is to let the control flow managed by a host processor. But this reduces greatly the possibilities to use the CGRA and

Table 4.2 – Summary of existing approaches to manage control flow in CGRAs

Techniques	Conditionals		Loops	
	Balanced	Imbalanced	Single	Nested
Partial predication [183]	√	√	×	×
State based full predication [153]	√	√	×	×
Dual issue single execution [172]	√	×	×	×
TLIA [102]	√	√	√	×
Software pipelining [206]	×	×	√	×
Our work: Register allocation [86]	√	√	√	√

increases the communication overhead, loosing sometimes the benefit of the acceleration provided by the CGRA. Another approach is to provide the CGRA with extra hardware features to support the control flow. Two structures are distinguished: 1) Conditional & alternative structures, and 2) iterative structures.

1) Conditional & alternative structures

Conditional & alternative structures are if-then-else (ITE) constructs. As clearly presented in [35], there are four basic methods to map applications with ITE onto CGRAs: (1) Full predication [204], (2) Partial predication [183], (3) Dual-issue single execution [35, 59, 119, 172], (4) Direct CDFG mapping [86], our work originally named *register allocation approach*, and presented in section 4.3.2. In [181], if-statements are transformed into predicate statements. Supporting ITE constructs efficiently is still a hot topic, as witnessed by recent publications [35, 59], motivated by imbalanced branches that further raise the efficiency challenge. In [59], the mutual exclusive dataflows can be mapped on the same hardware resources. Then, at runtime, the correct datapath is selected according to the branch outcome. When the branches are unbalanced, Yuan et al. [35] propose a dynamic scheme to execute directly the correct branch and not wait.

Note that in a CGRA, branch prediction is usually not needed. Branch prediction is needed because of pipelining and knowing the result of the taken branch is several cycles behind the current fetch or decode steps. In the case of a single-cycle elementary processing unit like in classical CGRAs, the result is known in one cycle, enabling fetching the right configuration or instruction directly.

2) Iterative structures

Iterative structures are defined by an initialisation phase, an iteration condition, and an iteration step. Most of the works focus on *for loops*. Loops have been the primary care since the early days of CGRAs [212]. Since loops concentrate the most important computing part of the application, researches naturally focused on this specific case, and the topic has been intensively studied during the last two decades. Most of the works consider the loop body, letting the control flow managed by a host processor. When the loop body contains conditional or alternative structures, the techniques presented in 4.1.2 can be used. Mapping loops on CGRA is so intensively studied that it would certainly deserve a survey on its own.

Modulo scheduling. Modulo scheduling is the most widely used technique to map loops on the CGRA [37, 184, 189, 203]. It can rely on a modulo routing resource graph (MRRG) [59, 203]. It can also be solved through graph-based approaches [72, 186].

Hardware loops. Hardware loops consist of extra logic inside the CGRA to manage the iterations of the loop in order to reduce the overhead of loop control by the processor [32, 68, 98].

Table 4.2 summarizes the different techniques to manage control flow in CGRAs. The table shows that our work contributes in managing nested loops.

Graph transformations

A DFG can be reshaped to exhibit a better mapping with the CGRA. Figure 4.4 shows a simple DFG with four possible transformations:

- "Operation Splitting" duplicates an operation node by keeping its same inputs and distributing output

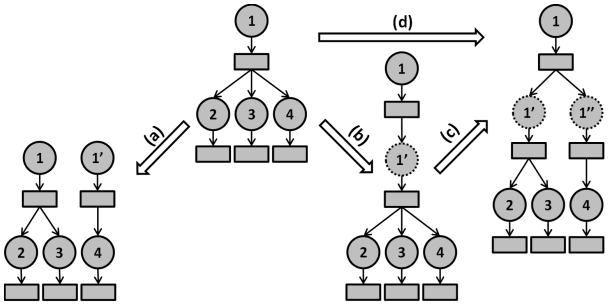


Figure 4.4 – Graph transformations: (a) Operation Splitting, (b) Simple Routing, (c) Memorization Splitting, (d) Routing and Splitting, figure taken from [100]

edges to reduce the number of successors of the original operation node as shown in figure 4.4(a). "Operation Splitting" is also called "recomputing" in the literature [152]. When a node has several successors, it leads to high contraints in the routing. Operation splitting is thus useful to relax such constraints.

- "Simple Routing" adds a memorization node and its associated data node to delay one operation and to keep data dependencies as shown in figure 4.4(b). This is typically interesting to add an explicit move operation, to move a data from one tile to another one further than the immediate neighbour.
- "Memorization Splitting" shown in figure 4.4(c) is equivalent to "Operation Splitting" but applied to memorization nodes. It adds another memorization node with the same parents at the current cycle and distributes edges to reduce the number of successors of the original node.
- "Routing and Splitting" is the combination of "Simple Routing" and "Memorization Splitting" as shown in figure 4.4(d). It delays the schedule of an operation and reduces the number of successors of the generated memorization node.

These transformations help the mapping step in finding a solution. In the literature, these transformations are usually applied *a priori*, before the mapping process [37, 152]. Some transformations can be applied a priori, e.g. when the number of fanout nodes is greater than the number of neighbors, but in the general case, it is hard to know statically what transformation is interesting to apply before the mapping process. In our work, we consider these transformations during the mapping process. We call this "dynamic" transformations because they are applied at the same time as the mapping solution is built. The transformation to apply is relevantly chosen when no solution is found. Our approach is discussed in more details in section 4.3.2.

Data mapping

The interaction between the CGRA and the memory is also of utmost importance as it defines the efficiency of the whole execution of the application. Various parameters of the memory can be considered for an efficient mapping: number of banks, communication bandwidth, and memory size [25, 83, 99, 130, 162].

The internal memory resources of the CGRA should also be used efficiently. Register allocation is presented in [137, 184], for a rotating register file [184], or for a unified register file [73].

Timeline

Figure 4.5 presents the evolution of scientific production around CGRA mapping the last two decades. The number of publications per year is not accurate, as it considers the papers focusing on CGRA mapping only, and a subset of selected papers, but still it shows that the community has intensified the efforts in the last decade, with a clear increase in 2021. The figure also shows that modulo scheduling was considered since the beginning of the studies, that supporting branches started in the early 2000s, and that memory-aware methods gained interest around 2010.

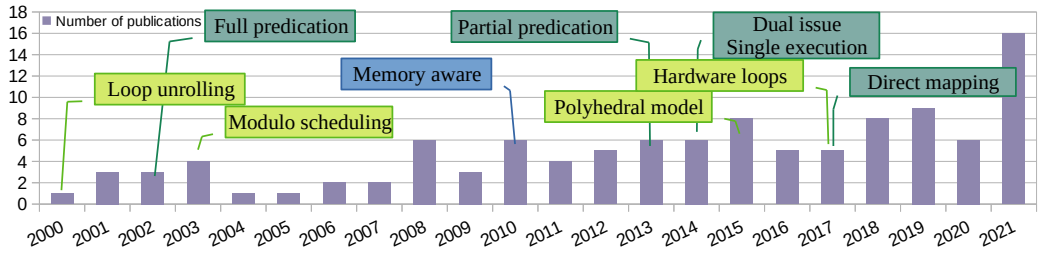


Figure 4.5 – Number of publications related to CGRA mapping from 2000 to 2021. Note that this timeline does not include papers about CGRA architectures, and is not comprehensive.

4.1.3 Execution model

A CGRA may follow different execution models, according to the way the configurations are first loaded and then run in the CGRA. The order which the configurations are loaded in can be statically defined at compile time, or dynamically decided. A second level defines whether the operations are scheduled and executed following an order defined by the compiler, or following a dataflow scheme, as soon as the operands are ready. In [61], the authors suggest four main categories as the combination of the possibilities: (1) static-scheduling sequential execution (SSE), (2) static-scheduling static-dataflow (SSD) execution, (3) dynamic-scheduling static-dataflow (DSD) execution, and (4) dynamic-scheduling dynamic-dataflow (DDD) execution.

In our work, we focus on the first category: static-scheduling sequential execution (SSE), meaning that the order of the configurations can be decided at compile time (most of the applications considered fit in the CGRA though), and the scheduling of the operations is statically defined by the compiler.

Spatial computation vs. temporal computation.

One of the crucial hardware feature that the compiler must *know* is if the CGRA supports spatial computations or temporal computations [61]. Spatial computation is very similar to FPGAs. Along with spatial computations that all CGRAs support, temporal computations allow to share in time the hardware resources leading to more flexibility, but are often criticized to reduce the energy efficiency [43].

4.2 Contribution 1: From a fault-tolerant reconfigurable standalone architecture...

The abundant number of (homogeneous) resources in a CGRA has made it an interesting architecture to study from the fault-tolerance point of view. Indeed, if one cell is out of order, it can be substituted by a cell nearby.

This was studied by Thomas Peyret during his PhD thesis [125]. The goal was to provide a fault-tolerant and flexible standalone digital signal processing architecture, able to respect throughput and latency constraints. With the constraint of technological independency to not fall under the International Traffic in Arms Regulation (ITAR), the fault tolerance shall be integrated at architectural level. A new architecture along with its compilation tools have been proposed. The flow allowed to program the fault-tolerant architecture from a high level language. The proposed system was fully autonomous as to not depend on the reliability of any other component like a host processor for the execution of the application.

4.2.1 Architectural fault-tolerant techniques

Faults can be merely classified in two categories: transient faults, and permanent faults.

Transient faults

For the computing ressources, the basic hardware technique to recover from a transient fault is by using Triple Modular Redundancy (TMR) [91]. The idea is to perform three times the same computation by three different ressources, and then apply a majority function to get the correct result. For the memory ressources, the basic technique is to use simple parity check or more complex error code correctors (ECC).

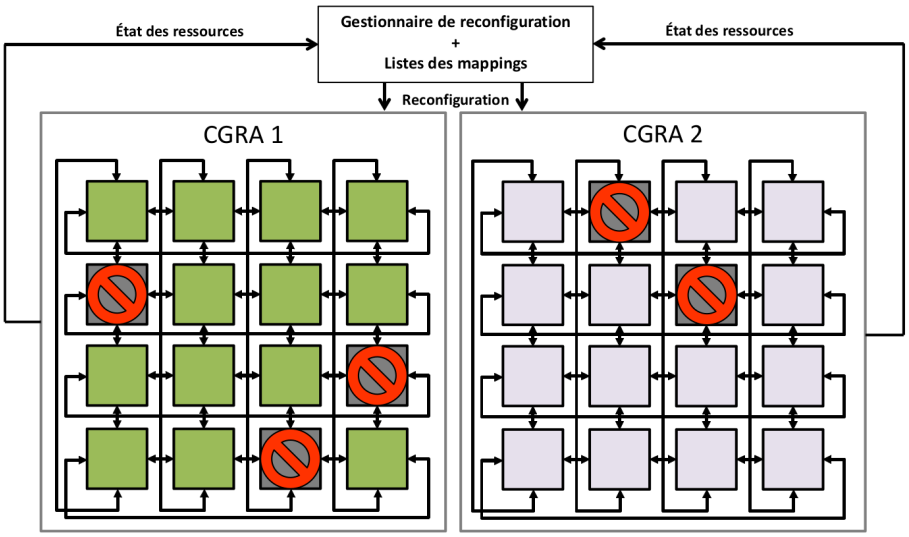


Figure 4.6 – The "Ping pong" system [125], showing the CGRA 1 with three faulty tiles, and the CGRA 2 with two faulty tiles. *État des ressources*: status of the ressources, *Gestionnaire de reconfiguration*: Reconfiguration engine, *Listes des mappings*: list of mappings.

We did not address these memory aspects in our work, so they are no further developed in this document. Note that using ECC for the memories (e.g. caches) is now mainstream in today's consumer electronics.

Permanent faults

When a permanent fault occurs, the tile of the CGRA becomes faulty, and cannot be used anymore. The issue is to be able to detect when the tile is permanently affected. One strategy is to rely on built-in self-test techniques like proposed by Rakossy et al. [142], which present the drawbacks of extra hardware overhead, the limited reactivity and unavailability during the tests. The other approach, that we follow, is to rely on the existing transient faults correction mechanisms inside the tile to detect permanent faults. The basic idea is to count over a sliding window the number of times a ressource provides a wrong result.

The "Ping pong" system

One part of the work done was to pave the way to a fault-tolerant CGRA-based system. Figure 4.6 shows a picture of such a system, composed of two CGRAs, working intermittently. Each CGRA is composed of tiles able to support transient faults, and detect faulty tiles. The reconfiguration engine comes in action to find a new configuration that do not use the faulty tiles. During reconfiguration, the second CGRA takes over to guarantee the operating continuity. The assumption made is that the reconfiguration engine (typically a simple and lightweight CPU) is itself radiation-hardened, and can run the reconfiguration steps.

The execution model considered follows one configuration loaded for the full application. When the CGRA is configured, it can run continuously the application until a fault appears. The CGRA is assumed to be large enough to hold the full program. The application targeted is a streaming application in a signal processing chain, with latency and throughput constraints. This "Ping pong system" has been the subject of the patent described in [110].

4.2.2 Limitations identified

If fault-tolerant mechanisms were already existing at architectural levels, one of the biggest lack in the literature was a method to compile a complete application, not only the kernel, on a CGRA. Indeed, the very first CGRAs were meant to act as co-processors to speed up the execution time. Most of the proposed approaches focused, rightly from the performance point of view, on kernels of applications, offloaded by the host processor on the CGRA; the host processor executing the rest of the application,

Table 4.3 – Summary of operations of an application executed on a host CPU or the CGRA

References	[187][165] [175][188] [197][215] [211]	[139]	[210][198] [209][150]	Our work
Memory operations	CPU	CGRA	CPU	CGRA
Innermost loop	CGRA	CGRA	CGRA	CGRA
Outer loop	CPU	CPU	CGRA	CGRA
Reconfiguration	CPU	CPU	CPU	CPU
Overhead				

including the loop control. The basic techniques available in the literature, back in 2011, were modulo scheduling of dataflow only loop bodies. But compiling an application needs to take into account all control structures: conditional and alternatives structures (if-then-else), and iterative structures (loops). The classical predication techniques for conditionals were early used: full predication [204], partial predication [183]. After 2011, a more elaborated technique called dual-issue single execution was proposed [35, 59, 119]. But methods supporting the loop control along with the data flow of an application were lacking.

Table 4.3 shows a summary of the operations of an application, including the memory operations and the loop control, executed either on a host CPU, or the CGRA.

The table clearly shows a lack in existing solutions to execute a full application on a CGRA.

Mapping control flow on CGRAs

How to map a complete application, including its control flow (i.e. loop control), on a CGRA?

The work of Thomas Peyret thus focused first on how to make a CGRA standalone, able to compute both control and data flow parts of an application, including the loop control. This challenge has been taken up by a two-fold action: (1) adding some hardware features to the CGRA, (2) developing a new mapping technique.

4.2.3 Hardware support and compilation tool for a standalone CGRA

Hardware features for a standalone CGRA

One of the key element to design first is a tile (a cell) that a standalone CGRA can be built on. Such a tile should be able to support control flow, meaning comparison and jump instructions, and a global synchronisation mechanism so that all tiles run the instructions in a synchronized way.

Figure 4.7 shows the proposed tile. As a reminder, a basic cell is composed of the ALU, the register file (RF), the output register, and sometimes the Load-Store Unit (LSU). The main component added to support the control flow is the control unit, and compare and jump instructions in the instruction set architecture. The hardware components added to support the global synchronisation scheme are: the status register, the blocking load, the timestamps associated to the instructions.

Blocking load

When accessing a data in the memory, the response time varies for reasons linked to the location of the data in the memory, memory conflicts, or the refresh time in the case of DRAM memories. When a tile

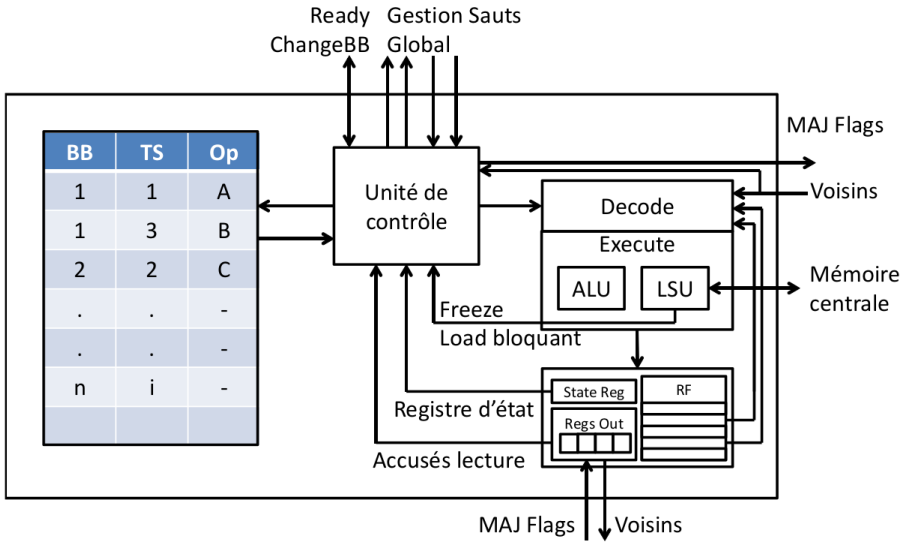


Figure 4.7 – A tile with internal configuration memory, control support, and memory blocking features [125]. BB: Basic Block, TS: Time Stamp, Op: Operation, RF: Register File. *Gestion Sauts Global*: Global jump management, *Unité de contrôle*: Control Unit, *Registre d'état*: Status Register, *Accusés lecture*: Read acknowledgment, *MAJ (Mise à jour) Flags*: Flags update, *Mémoire centrale*: Main memory, *Voisins*: Neighbors, *Load bloquant*: Blocking load.

is blocked while waiting for the data to be available, a safe strategy to guarantee the correctness of the execution is to force the other (concurrent) tiles to also wait, maintaining thus a global evolution of the program for each tile, each synchronized with a timestamp. The timestamp evolves similarly for each tile. This execution model corresponds to the "sequential execution" as described in section 4.1.3. When a tile is blocked, a "freeze" signal is then triggered to freeze all the tiles of the CGRA. The freeze signal is released when the data is available.

Mapping control flow on CGRAs

- Adding minimal architectural support: a comparator for comparison instructions
- Adding a jump instruction with a label
- Adding a global hardware synchronisation mechanism

The full application support mechanism in the CGRA has been the subject of the patent described in [111].

A compilation approach for a standalone CGRA

We identified three features to support a CDFG compared to a simple DFG:

1. ability to go from one basic block to another through jumps,
2. possibility to have shared variables between the basic blocks,
3. possibility to have a specific node (a PHI node), that selects the good value according to a condition.

These three key features should be considered while solving the mapping problem. Jump operations are explicitly available in the intermediate representation. Unconditional and conditional jumps are possible. Conditional jumps are specifically interesting to control the iterations of a loop. An instruction is needed to compare the current index of the loop with the condition. The result of the comparison tells which basic block the CGRA shall jump into.

The systematic load-store approach

In a CDFG, a single basic block is executing at a time. The first and straightforward strategy is to load all the data needed to execute a basic block, and store back all the data when the basic block is finished. This leads to an overhead for the shared variables. From a fault-tolerant point of view, it eases the ping-pong mechanism, or checkpointing techniques. At worst, only the current basic block needs to be re-run in case of failure, the data being systematically stored back in the main memory.

For the shared variables, i.e. the ones that could be kept inside the registers of the CGRA, the compiler automatically transforms their accesses to an explicit load and store node in the DFG, with the corresponding data dependency. During the mapping process, these nodes are simply mapped like other load/store nodes. This technique is called the "systematic load-store approach".

Diversification of mappings

The goal of the reliable system is to run as long as possible. Given an initial database with all the possible reconfigurations, the idea is to fill the database with mappings as diverse as possible. The mappings should not be different just by one or two tiles, but also by different shapes in their footprint. When no constraint is given, the mapping tool uses all possible tiles, in all possible directions. In order to force diversification, some constraints are imposed, in the number of tiles used and the maximum number of neighbors, mimicking some faulty tiles. Following this strategy, the mapping process ends up with several solutions, all similar from the latency point of view, but with different caracteristics and different footprints on the grid.

Figure 4.8 shows the average number of different mappings obtained on different kernels from the signal processing domain. The proposed approach is compared with three other methods:

- "Method 1" that solves the scheduling and the binding problem separately as proposed in [164]. Graphs are transformed during scheduling by applying "Simple Route" transformation only. A forward list-based scheduling algorithm and the original Levi's binding algorithm are used.
- "Method 2" that traverses the graph forwardly, schedules nodes by applying a priori transformations and tries to find a mapping by using the original Levi's algorithm as proposed in [152].
- "Method 3" that backward traverses the graph, schedules and binds nodes simultaneously with dynamic transformations as proposed in this paper and removes redundant partial mappings to prune the solution space.

"Method 3" is actually one approach that we also proposed, but suffers from scalability issues, which are discussed later in section 4.3.2.

The results show that our approach can better explore the solution space and thus find solutions with high diversity. This approach offered a solid baseline for further studies.

4.3 Contribution 2: ... to an ultra-low power programmable array

One of the main outcome of the PhD thesis of Thomas Peyret was a new standalone CGRA, able to execute a full application, along with its mapping tool. The results obtained were also interesting from the execution time point of view, although not the primary goal. The question then arose how this kind of CGRA would behave in an embedded system context, with a stringent power budget.

Energy efficient embedded systems

Could a standalone CGRA offer a good energy efficiency compared to the traditional CPU+CGRA coupling scheme?

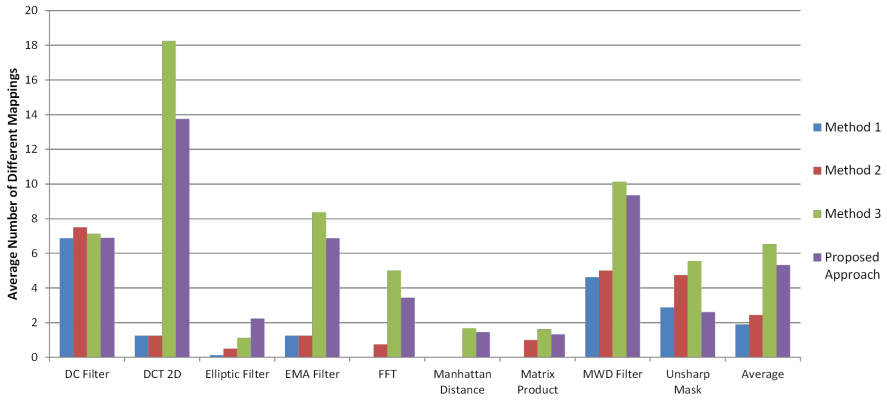


Figure 4.8 – Average number of different mappings.

This question was the main underlying motivation of the PhD thesis of Satyajit Das [70], a joint collaboration between université de Bretagne-Sud and university of Bologna. In order to answer the question, some actions were needed both on the hardware side and the software side.

4.3.1 Architecure

The first action was performed on the architecture of the CGRA. Two main features are detailed here: 1) a minimalist instruction-set, 2) clock gating technique.

A minimalist instruction-set

The fault-tolerant CGRA proposed by Thomas Peyret was based on a 44-bit instruction-set, including specific redundancies and instructions (e.g. voting) for fault-tolerant reasons. The first action was to rethink the instruction-set to a minimalist version, to minimize the memory needed to store the instructions, the size of the instruction decoder, and finally reach the smallest energy footprint. This work led to a 20-bit instruction-set.

Clock gating

The second feature relies on the well-known clock gating technique. Clock gating consists of adding an AND gate to the clock signal, to avoid the clock signal to propagate throughout a component. Indeed, most of the energy spent in a circuit is based on the switching activity. When this activity is not needed, simply shutting down the clock signal brings interesting energy savings. In the context of a CGRA, all the cells of the grid are not all used every clock cycle. The idea is to clock-gate the cells that are not in use at a given time.

IPA: Integrated Programmable Array

The design step ended in a CGRA that we called Integrated Programmable Array (IPA). The architecture comprises a PE array, a global context memory, a DMA controller (DMAC), a tightly coupled data memory (TCDM) with multiple banks and a logarithmic interconnect. Figure 4.9 shows the organization of the IPA.

PE Array: The array consists of a parametric number of PEs, connected with mesh torus network. Figure 4.10 describes the components of a PE. Two input muxes select two operands (OpA and OpB). The sources are the neighboring PEs or the register file. A 32-bits ALU and a 16-bit x 16-bit = 32-bit multiplier are employed in this block. The Load Store Unit (LSU) is optional for the PEs (the optimal number of LSU is a parameter that we studied). The Control unit is responsible for fetching the instruction from the corresponding address of the instruction memory and managing program flow. The Regular Register File (RRF) and Output Register (OR) store the temporary variables, while constants are stored in Constant Register File (CRF). The Condition Register (CR) contains 0 for all the normal operations and true conditions, and 1 for false conditions. The boolean OR of all the control bits from all PEs gives the

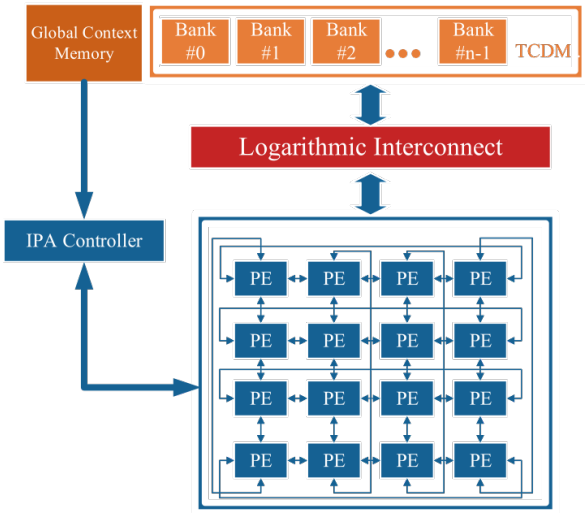


Figure 4.9 – IPA computing system [87]. TCDM: Tightly Coupled Data Memory, PE: Processing Element

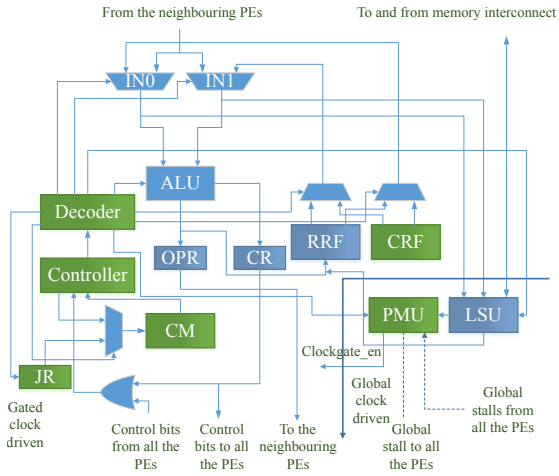


Figure 4.10 – Components of the PE (Processing Element) of IPA [87]. PMU: Power Management Unit, RRF: *Regular* Register file, CRF: Constant Register File, CM: Context Memory, OPR: Output Register

indication that one PE has executed false condition in the previous cycles. So next, the offset address of the false path must be fetched. The Jump Register (JR) contains the address to be jumped at.

Global Context Memory (GCM): The Global Context Memory stores configuration data (instruction and constants) for each PE in the PE Array.

DMA controller (DMAC): The DMA controller identifies configuration data for the corresponding PE and transfers it in the *load context* stage. It also initiates the execution phase after loading all the contexts.

TCDM and logarithmic interconnect: The TCDM (Tightly Coupled Data Memory) has a number of ports equal to the number of memory banks providing concurrent access to different memory locations. Load store operations in the PEs are based on a high bandwidth low-latency interconnect, implementing a word-level interleaving scheme to reduce access contention.

The organisation of the global context memory and the mechanism to load the context (instructions) in the grid are described in [87].

Integration in the PULP platform

The Parallel Ultra Low Power (PULP) Platform started as a joint effort between the Integrated Systems Laboratory (IIS) of ETH Zürich and Energy-efficient Embedded Systems (EEES) group of the University of Bologna in 2013 to explore new and efficient architectures for ultra-low-power processing [6], at the instigation of Prof Luca Benini.

The thesis of Satyajit Das was a joint collaboration between université de Bretagne-Sud and university of Bologna. Satyajit, during his stay in Bologna, benefited from the ideal conditions to integrate the IPA into a PULP cluster.

The PULP Cluster

The PULP cluster features 8 32-bit RISC-V cores based on a four pipeline stages micro-architecture optimized for energy-efficient operations [88] sharing a 64KB multi-banked scratchpad memory through a low-latency interconnect [166]. The ISA of the cores is extended with instructions targeting energy efficient digital signal processing such as hardware loops, load/store with pre/post increment, SIMD operations. The cores share a 4KB private instruction cache to boost performance and energy efficiency for tightly coupled clusters of processors typically relying on data parallel computational models [92]. Off-cluster data transfers are managed by a lightweight multi-channel DMA optimized for energy-efficient

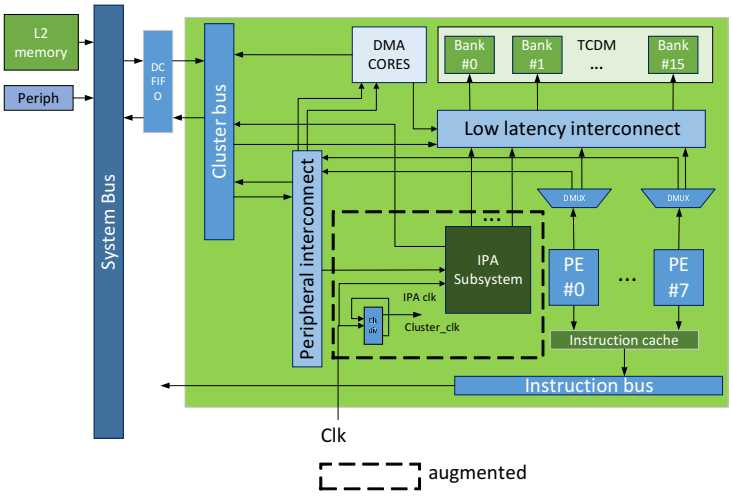


Figure 4.11 – PULP cluster augmented with IPA subsystem

operation [128]. Both the instruction cache and DMA are connected to an AXI4 cluster bus. A peripheral interconnect is used to communicate with on-cluster peripherals such as a timer, a hardware synchronizer and other memory mapped peripherals such as application-specific accelerators. To operate at the best operating point for a given workload the cluster can be integrated in an independent voltage and frequency domain, featuring dual-clock FIFOs and level shifters at its boundary.

Heterogeneous PULP-IPA Cluster

We extended the PULP cluster with the IPA, as shown in figure 4.11. As opposed to many CGRA architectures, the IPA can access a multi-banked shared memory through 8 master ports connected to the low-latency interconnect. This eases data sharing with the other processors of the cluster, following the computational model described in [133]. The optimal number of port has been chosen to optimize the trade-off between the size of the interconnect and the bandwidth requirements of the IPA. Following the analysis we conducted in [86], which shows that the IPA can operate $2\times$ faster than the processors, we have extended the architecture of the cluster in a way that the IPA can work at twice the frequency of rest of the cluster. The integration is detailed in [71].

4.3.2 Compilation

The second action was performed on the compiler, specifically on the mapping approach. This section actually presents the outcome of the work initiated by Thomas Peyret and extended by Satyajit Das.

Architecture and application models

The compiler takes two inputs. The first is the model of the CGRA, and the second is the ANSI-C code of the application.

The IPA is modelled by a bipartite directed graph with two types of nodes: operators and registers. Timing is implicitly represented by connections between registers and operators, which is referred to as the *time extended model* of the PEA [152]. Two types of operator nodes are defined for the PEs. The first type is the computing operator that represents the physical implementation of an arithmetic and logical operation (+, $\times$, -, OR, AND) and/or memory access (e.g. load/store). The second type of operator is the memorization operator, which is associated with the output register and represents the operation of keeping a value in a local register explicitly.

The application is modelled as a control and data flow graph (CDFG). A CDFG is depicted as $G = (V, E)$ where V is the set of basic blocks and $E \subseteq V \times V$ is the set of directed edges representing control flow. A Basic Block (BB) is represented as a data flow graph (DFG) or $BB = (D, O, A)$ where D is the set of data nodes, O is the set of operation nodes and A is the set of arcs representing dependencies. The control flow

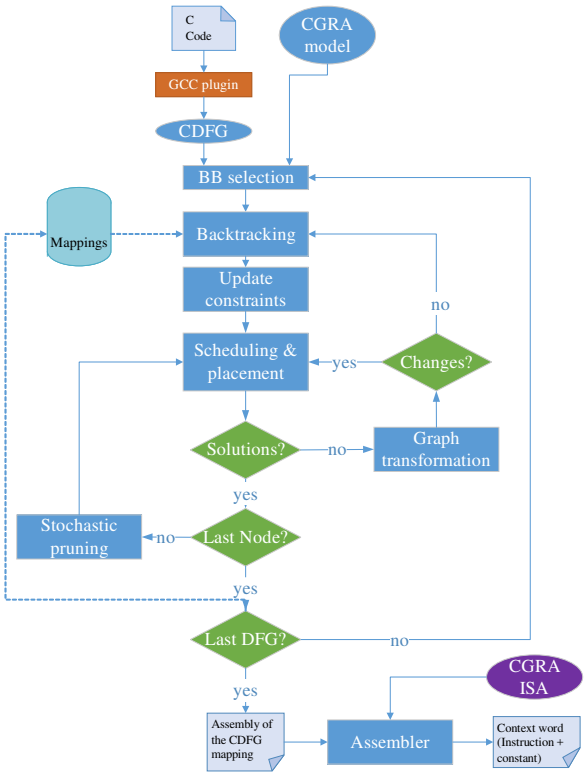


Figure 4.12 – Our proposed compilation flow to find a mapping for a CDFG on the IPA, presented in [54]

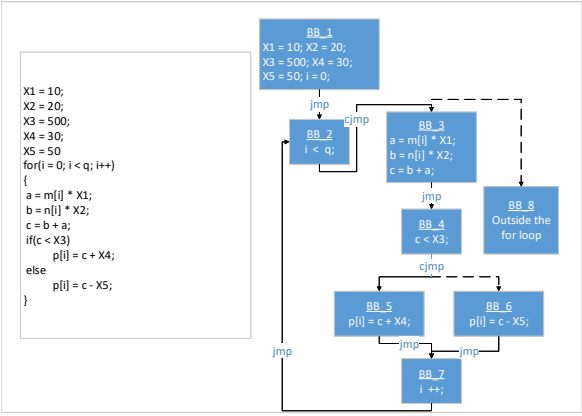


Figure 4.13 – Sample program and corresponding CDFG

from one basic block to another is supported with jump (*jmp*) and conditional jump (*cjmp*) instructions. Figure 4.13 shows an example of a kernel with its corresponding CDFG. The basic blocks in the CDFG are composed of data nodes, operation nodes, and data dependencies.

Homomorphism

Three equivalences between the basic block DFGs and nodes of the model of the IPA are defined: (1) data and registers; (2) computation and computing operators; (3) data dependencies and connections between the time extended PE components. As the two models are homomorphic, the mapping of a DFG onto the PEA is therefore a problem equivalent to finding a DFG in the IPA graph.

The compilation flow step by step

Figure 4.12 shows the steps followed by our compilation flow for mapping CDFGs onto the IPA. A CDFG mapping is a set of DFG mappings that are compatible with each other. To be compatible, the DFGs must access the data that remain in the PEs in the same location, which is ensured by the register allocation approach described later in 4.3.2.

The full compilation flow is composed of six interdependent stages: basic block selection, backtracking, update constraints, scheduling and placement, graph transformation and stochastic pruning, which are now described.

Scheduling and placement

A *full* DFG mapping is built up cycle by cycle. At each cycle, there are several nodes to be mapped, which number might be greater than the number of available resources at the current cycle. First, priority is given to nodes with a high fanout, assuming that they are more difficult to place, then the mobility is considered [221].

The proposed approach uses a backward traversal list scheduling algorithm, originally presented by

Thomas Peyret in [127], to schedule the DFG of each basic block. It relies on a heuristic in which the schedulable operations are listed by priority order. In backward traversal, a node is schedulable if and only if all its children are already scheduled. It is possible to process memorization nodes and conventional nodes differently. As soon as the highest priority node has been defined, the compiler tries to find a placement in the CGRA model. If a placement solution exists, the node is scheduled else the graph is transformed.

The proposed placement uses an incremental version of Levi's algorithm [226]. The proposed algorithm adds the newly scheduled operation node and its associated data node to the sub-graph composed of already scheduled and placed nodes. If no solution is found, there is absolutely no possibility to bind this couple in all the previous partial solutions because Levi's algorithm provides a complete exploration of the solution space. In that case, graph transformation is required.

Graph transformation

The graph transformations that we used are the ones described in section 4.1.2, p. 54.

Stochastic pruning

The comprehensive nature of the placement step usually leads to a high number of partial mappings (depending on the data dependencies and architectural constraints). This prevents to use complex DFGs or CGRAs with large number of tiles or register files. To reduce the number of partial mappings generated by the placement step, we introduce a stochastic pruning step. For each partial mapping, the selection step generates a random number between 0 and 1 which is compared to a *threshold*. If the generated number is less than or equal to the threshold value, the partial mapping is kept otherwise the solution is discarded. Since the number of partial mappings depends on the current step, and grows exponentially, choosing a fixed threshold value is not an option. Typically, there are only few partial mappings after the first cycle, so we must keep most of them, but there are quickly thousands of them after few cycles, and many can be discarded. So, the threshold must adapt to the current number of partial mappings. The ultimate goal of defining the threshold is to have control over the number of partial mappings selected or passed. This number should be low enough to scale up and high enough to allow finding at least one valid solution as very few of the selected partial mappings result in a valid mapping. In other words, success rate highly depends on the choice of the threshold function.

The threshold function should be a decreasing function. We studied different functions, like an exponential function which is widely applied in simulated annealing based algorithms, hyperbolic, or inverse functions. Our experiments show that the inverse function works better than the other one studied. The full explanations with the experiments are available in [8].

Finding a valid mapping solution depends on the number of partial mappings. Although the inverse threshold helps to control memory footprint by reducing the number of partial mappings, some heuristics is necessary to ensure a good number of partial mappings is maintained throughout the mapping process. We propose to introduce bounds as control mechanisms: *LB (Lower Bound)* and *UB (Upper Bound)*. While randomly selecting the partial mappings from a set, it might so happen that the pruning function did not not select any one of the partial solutions failing to find any valid solution. Hence, it is absolutely necessary to set a minimum number (lower bound) which the pruning function must select from the solution space. This gives an opportunity to find a valid mapping in the end. If the selected partial solutions does not reach to the lower bound, the pruning function will iterate through the solution space until the number is reached. In this iterative selection process, the function might select huge number of solutions which will increase the compilation time. To get the compilation time scalable and success rate high, we introduce both *upper* and *lower* bound. However, the upper bound may impact the quality of mapping by over-constraining the selection process. We investigated the impacts of the different bound-based pruning approaches for the best trade-off between quality of mapping and compilation time:

- *RED* (**Redundant** deletion) [127] removes redundant partial mappings to prune the solution space in the baseline approach
- *SNoB* (**Stochastic** selection with **No Bounds**) uses stochastic method without any bounds to prune

the solution space.
- *SLUB* (**Stochastic** selection with **Lower** and **Upper Bounds** (LB&UB)) uses stochastic method with upper and lower bounds for pruning the solution space
- *SLoB* (**Stochastic** selection with **Lower only Bound** (LB)) uses stochastic method with only lower bound to prune the solution space

Basic block selection

Once all the nodes of the basic block have been scheduled and bound, the compiler selects one mapping among the several mappings generated, and selects the next basic block to be mapped. Data integrity must be maintained over several basic block mappings for the shared variables. This is guaranteed by the register allocation approach explained later.

Backtracking

For a basic block to be mapped (except the first one), this stage selects the first mapping out of several mappings generated for the last basic block mapped. If the compiler is unable to find a mapping solution compatible with the current mapping, the next one in the list is selected. The process continues up to the first basic block mapped until a valid mapping is found for the current basic block.

Register allocation

The idea of the register allocation approach is to use the internal registers of the PEs to hold scalar variables that could be alive accross different basic blocks.

The register allocation approach is the centerpiece of our CDFG mapping approach, which ensures to obtain a set of consistent and compatible DFG mappings, able to share variables kept inside the registers of the PEs of the CGRA.

The approach relies on two kinds of constraints: the Reserved Location Constraints (RLC), and the Target Location Constraints (TLC).

The RLC are used to prevent the usage of a register that holds an alive variable. In that case, the register cannot be used and is called “reserved”. The RLCs keep track of all the registers that cannot be used to find a valid mapping.

The TLCs are used to force the usage of a given register. When a scalar variable is used in different basic blocks (for instance, typically the variable `i`, the index of the loop), this variable is stored in a register and the mappings should use this same location when accessing the value. It is called a “target location”.

The full explanations are available in appendix A.

Context-memory aware mapping

The area and energy efficiency of our standalone CGRA are bottlenecked by the configuration/context memory. The size of these context memories is of prime importance due to their high area and impact on the power consumption. For instance, a 64-word context memory typically represents 40% of a processing element area. Our first mapping approach did not take the size of the context memory into account, and the standalone CGRA became oversized which strongly degrades its performance and interest. This is why we propose a context memory aware mapping. The size of the context memory inside the (PE) needs to be managed for ultra low power acceleration. In [53], we describe the mapping approach which tries to find at least one mapping solution for a given set of constraints defined by the size of the context memories of the PEs. These constraints force to better share the operations over the PEs of the CGRA. Experiments show that our proposed solution achieves an average of 2.3× energy gain (with a maximum of 3.1× and a minimum of 1.4×) compared to the mapping approach without the memory constraints, while using 2× less context memory. When compared to the CPU, the proposed mapping achieves an average of 14× (with a maximum of 23× and minimum of 5×) energy gain.

Table 4.4 – Synthesized area information for the PULP heterogeneous cluster

Components		Area (μm^2)	% of cluster area
CORES (x8)		160,352	18
ICACHE (4 kB)		190,089	22
DMA_CORE		41,406	5
IPA	PE ARRAY	154,515	**18**
	IPAC	861	
	GCM_INTCNCT	359	
	OTHERS	588	
	Total	156,323	
DMA_IPA		32,636	4
GCM		18,704	2
TCDM (16x4 kB)		149,638	17
CLUSTER_INTCNCT		63,126	7
CLUSTER_PERIPHERALS		21,610	2
OTHERS		37,932	4
Total		871,816	100

4.3.3 Results

This section recaps a few results obtained.

Experimental setup

Several versions of the IPA have been designed. The results presented here are obtained for a design synthesized with Synopsys design compiler 2014.09-SP4 using STMicroelectronics 28nm UTBB FD-SOI technology libraries. Synopsys PrimePower 2013.12-SP3 was used for timing and power analysis at the supply of 0.6V, 25°C temperature, in typical process conditions. The cycle information was achieved by simulating the RTL with Mentor Questa Sim-64 10.5c. The IPA is built upon a 4×4 array with 16 PEs, each one including 20×32-bit instruction register file, a 32×8-bit regular register file and 32×16-bit constant register file.

The PULP cluster consists of 8 cores featuring 4 kB of shared instruction cache. The TCDM is composed of 16 banks of 4 kB each, leading to an overall TCDM size of 64 kB. Table 4.4 presents the area information of the components in the cluster. Although, the total area of the IPA with 16 PEs is almost similar to the area of the 8 cores combined, the area occupied by the GCM is much less than the total cache memory, which in turn provides better area efficiency while running applications in IPA.

The IPA competes with an 8-core PULP cluster

Table 4.5 reports the execution time in nano seconds for different benchmarks running on a single-core, on 8 cores and on the IPA. The IPA execution time includes the time taken for loading the context into the PEs. Comparing to the performance of execution in single-core, the accelerator achieves a maximum of 8× (with a minimum of 2.49× and an average of 5.4×) speed-up. The control intensive kernel like GCD does not exhibit parallelism, hence parallel software execution does improve performance of the homogeneous cluster. On the other hand, the execution on the IPA improves the performance by almost 5×, exploiting also instruction-level parallelism rather than data-level parallelism only. The performance gain in the accelerator for the compute intensive kernels like matrix multiplication, convolution, FIR and separable filters is limited if compared to the performance of parallel-cores. However, the relatively similar performance from the purely execution time point of view is compensated by the gain in energy consumption (Table 4.6) due to the simpler nature of the compute units of the IPA with respect to full processors, to the smaller number of power-hungry load/store operations, and to the fine-grained power management architecture that applies clock gating on the inactive PEs during execution.

Table 4.6 shows the evaluation of the energy consumption for each configuration of the heterogeneous

Table 4.5 – Performance evaluation in execution time (ns) for different configurations in the heterogeneous platform

Kernels	Single-core (ns)	Multi-core (ns)	Speed-up in multi-core	IPA (ns)	Speed-up in IPA
MatMul	3,358,740	435,180	7.72x	432,630	7.76x
Conv	9,733,380	1,520,840	6.40x	1,494,860	6.51x
FFT	767,640	142,720	5.38x	94,510	8.12x
FIR	182,500	33,460	5.45x	33,410	5.46x
Sep Filter	39,870,420	6,404,160	6.23x	6,334,700	6.29x
Sobel Filter	117,024,880	40,894,260	2.86x	28,865,890	4.05x
GCD	2,951,160	2,951,160	1.00x	61,1300	4.83x
Cordic	9,000	7,000	1.29x	3,610	2.49x
Manh Dist	244,640	164,640	1.49x	70,300	3.48x

Table 4.6 – Energy consumption evaluation in μJ for different configurations in the heterogeneous platform

Kernels	Single-core	Multi-core	IPA	
			Energy	% of Active PEs/cycle
MatMul	1.247	0.313	0.208	58.5
Convolution	2.876	1.095	0.658	59.2
FFT	0.292	0.087	0.042	59.7
FIR	0.08	0.026	0.026	46.1
Separable filter	16.663	4.611	4.28	55.5
Sobel Filter	51.491	29.444	12.701	51.2
GCD	1.151	1.151	0.257	6.25
Cordic	0.004	0.003	0.001	50
ManhDistance	0.1	0.095	0.03	48.5

cluster. The table also shows the average usage of the PEs per cycle.

The results show that the IPA consumes much less energy than the multi-core version.

The stochasticity helps in better exploring the solution space

Figure 4.14 presents the results for ten runs of the DCT benchmark on different CGRA configurations with three different methods, *RED, SLoB and SLoBS*. Each point in the figure corresponds to one run by a method on the corresponding CGRA configuration. The *x* axis of the graph represents latency normalized to ASAP length and the *y* axis represents the number of transformed nodes normalized to the number of operation nodes in the original graph. So, each point in the graph is basically the outcome latency and number of transformed nodes of each run by a certain method. The points corresponding to the method *RED* and *SLoB* (**Stochastic** selection with **Lower only Bound**) show that they found similar latencies with almost similar number of transformations. The wide varieties of latencies and wide varieties of transformations of method *SLoBS* (SLoB with Stochastic Scheduling) prove that this method can better explore the solution space. The nodes with similar priority are scheduled in different order for each run in the case of SLoBS.

Stochasticity in placement in scheduling helps in exploring different possibilities in the mapping latency and DFG transformations.

Hence, the method *SLoBS* found the best latency with least number of transformations.

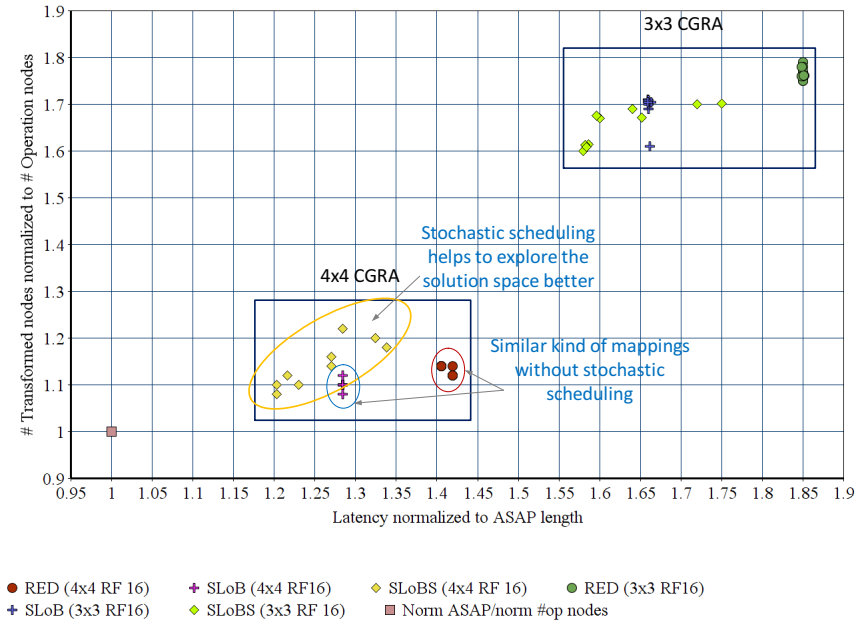


Figure 4.14 – Architectural coverage between methods for ten runs of the DCT benchmark on different CGRA configurations

4.4 Contribution 3: Adding floating-point capabilities

All the kernels considered up to that point were on integer numbers only. But many applications need some floating point numbers. Adding some floating-point capabilities to the standalone CGRA was the main goal of the PhD thesis of Rohit Prasad [13].

4.4.1 Adding FPU

Due to the area overhead induced by floating point capability, typical CGRAs become less attractive if all the PEs contain floating point computational units (which is confirmed by our results). Similarly, the operators on floating-point numbers are much more complex than their integer counterparts. In order to not degrade the performances on integer numbers, floating-point operations are performed in several cycles. Besides, the heterogeneity in the PEs often fails to transport data synchronously to the PEs containing floating point units, disturbing parallel float computations resulting in a performance bottleneck. Transporting data synchronously to the processing elements can be guaranteed by decoupling address generation of the data structures from the computation flow.

Adding floating point capabilities to CGRAs

Adding floating point capabilities to CGRAs raises new challenges:

- Multi-cycle operations
- Address generation unit
- Heterogeneous set of computing cells
- New instructions in the ISA

Hardware

The first contribution was to extend the IPA with IEEE 754 compliant floating point operations units with the associated compilation flow to efficiently exploit parallelism between the floating point operations at

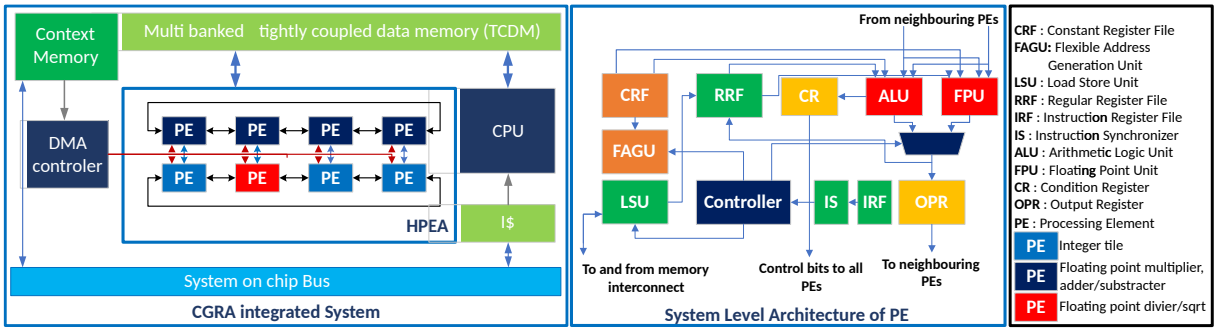


Figure 4.15 – The IPA extended with the FAGU (Flexible Address Generation Unit) and FPU (Floating-Point Unit). One PE embeds a hardware divider and square root unit.

Table 4.7 – Performance comparison with RISC-V 1-core. PC: Principle Component

Kernels	Execution time in cycles				Energy consumption in μJoule		
	CGRA without FAGU	CGRA with FAGU	RISC-V	Gain in CGRA with FAGU	CGRA with FAGU	RISC-V	Gain in CGRA with FAGU
mean covariance	734,934	300,050	732,377	2.44x	3.24	5.71	1.8x
accumulate	94,018	48,877	66,334	1.36x	0.38	0.52	1.4x
householder	211,113	125,375	132,870	1.06x	1.02	1.04	1x
diagonalize	257,488	84,186	175,092	2.08x	0.8	1.4	1.8x
PC	268,809	105,993	488,252	4.61x	1.14	3.8	3.3x

instruction level. The PEs of the IPA are also extended with a Flexible Address Generation Unit (FAGU) which decouples addresses generation from the computation flow.

Figure 4.15 shows a diagram of the new CGRA with floating point capabilities. The main differences with the original IPA are:

- 8 PEs only instead of 16
- 4 PEs are augmented with both FPU and FAGU
- 1 PE embeds a floating-point hardware divider and square root unit
- an instruction synchronizer (IS) that manages multi-cycle operations

The new CGRA system ends up with an area of 204,067 μm^2, which is nearly twice the size of a PULP cluster with a single core.

Compilation

From the compilation point of view, taking into account floating-point operations needs new features. First, the compilation flow identifies multi-cycle operations (floating point computations) in the whole CDFG. The operation nodes are then transformed by adding dummy nodes equal to the number of the total cycles needed to perform the operation. The dummy nodes are of course placed on the same node, using thus the resource during the whole computation. Second, the consecutive multi-cycle operations should be mapped onto the same PE, to not further increase the latency with extra move operations. This is encouraged by a new priority given to the schedulable nodes during the mapping process.

Results

Table 4.7 shows the performance comparison between the CGRA with FPU and a PULP cluster with a single RISC-V core [88], both on execution time in cycles and energy consumption in μJoule, for executing the kernels computing PCA (Principle Component Analysis) in the EEG (electroencephalogramme) analysis. The table shows also the results obtained without the FAGU. The results clearly show that without the FAGU, the CGRA performs similarly with the RISC-V core.

With the FAGU, a maximum speed-up of 4.61x is obtained, and an average of 1.86x gain in energy consumption.

The complete set of results, with comparison with 2, 4, and 8 RISC-V cores is available in [47].

4.4.2 *Transprecision*

An emerging approach to reduce the power consumption of floating-point operations while preserving the dynamic required by applications without manual adjustments is *transprecision computing* [81]. This paradigm aims at designing systems which deliver the required precision for intermediate computations given an accuracy bound specified by the user, and leverages automated tools to associate reduced-precision types to program variables [81]. An attempt towards transprecision computing was made by introducing two new custom FP data-types (*binary16alt* and *binary8*) and a hardware unit called smallFloat Unit (SFU) [77], which employs IEEE-754 *binary32*, *binary16*, and two new data-types, namely *binary16alt* (featuring a higher dynamic range vs. *binary16*) and *binary8*. Exploiting these data types leads to significant improvement in terms of performance and energy efficiency [77].

TRANSPIRE

The second contribution of Rohit Prasad's PhD thesis was to combine the principles of transprecision computing with the flexibility of CGRA in exploiting multi-datapath for high Instruction Level Parallelism (ILP) and Data Parallelism (DP), to propose a high energy efficiency low power FP-CGRA architecture called TRANSprecision floating-point Programmable archItectuRE (TRANSPIRE). TRANSPIRE gains 10.06× performance and consumes 12.91× less energy over a RISC-V CPU extended with SIMD-style vectorization and executing same kernels using same FP data-types as of the proposed CGRA. On this part, the efforts were concentrated on the hardware architecture, by integration the SFU inside the PE of the CGRA, and on the application to tune it for the CGRA.

The SFU (SmallFloat Unit) is a dedicated units for the so-called "small floats", float numbers on a reduced number of bits. It is designed following slices, 1x32 bits, 2x16 bits, and 4x8bits, shared between several cores. For the integration in the CGRA, Rohit Prasad designed the mSFU (mini-SFU), which includes 2 slices of *binary16alt* units and 4 slices of *binary8* units. The datapath is 32-bits wide, which enables TRANSPIRE to perform SIMD operations for custom FP data types. The operators in the mSFU are non-blocking and non-pipelined. Float-absolute and float-less-than operators support the IEEE-754 *binary32* format and are shared among these slices in mSFU.

Results

The TRANSPIRE architecture has been compared with RI5CY_SFU, a 4-stage RISC-V CPU with an enhanced ISA supporting SIMD-style vectorization that includes SFU [77]. Table 4.8 summarizes the results obtained for three applications. CONV implements a 5×5 convolution kernel, used for image and audio processing applications. DWT computes the Discrete Wavelet Transform, used for Electrocardiography (ECG) analysis applications. SVM is the prediction stage of a Support Vector Machine.

The results show that TRANSPIRE can bring an interesting 10x improvement on execution time, and up to 12x in energy.

Table 4.8 shows the results for binary8 datatypes only. Other results on other datatypes are available in [47].

Kernel	Average deviation (%)	Data-type	TRANSPIRE binary8 (cycles)	RI5CY_SFU binary8 (cycles)	Gain	TRANSPIRE binary8 (μJ)	RI5CY_SFU binary8 (μJ)	Gain
CONV	2.32	*binary8*	268,179	1 455,097	5.43×	3.036	21.506	7.08×
DWT	6.98	*binary8*	11,140	16,912	1.52×	0.124	0.256	2.07×
SVM	7.11	*binary8*	11,408	114,747	10.06×	0.123	1.588	12.91×

Table 4.8 – Accuracy performance of TRANSPIRE, with cycles and energy consumption

4.5 Summary

This chapter presented how to exploit ILP and DLP with CGRAs. The chapter starts by reminding the research done the last two decades, particularly on the compilation side, with the CGRA mapping problem. The review of existing methods highlighted that CGRAs have been mainly studied as co-processors in charge of accelerating the dataflow parts of an application, the main limitation identified being that CGRAs were not able to also handle the loop control.

Mapping control flow on CGRAs

How to map a complete application, including its control-flow, on a CGRA?

Back in 2012, there was no such existing method, that can support the control flow of an application including the loop control and not only if-then-else constructs.

Mapping control flow on CGRAs

- Adding minimal architectural support
- Propose a new mapping method

The first step is to consider a hardware support for control flow. The hardware proposal stands upon a global synchronisation scheme to guarantee the correct execution of the application in all cells of the CGRA.

Mapping control flow on CGRAs

We proposed two techniques to map the control flow of an application

- the systematic load-store with architectural support
- a *register allocation based approach*, renamed *direct CDFG mapping* in the literature

We proposed a CGRA named IPA for Integrated Programmable Array, integrated into a PULP cluster, to act as a standalone programmable hardware accelerator. We also proposed a version able to support floating-point operations, and a *transprecision* version, that allows to tune between accuracy and energy efficiency.

The compilation method proposed relies on several key original features:

- backward traversal of the nodes of the DFG
- a cycle by cycle mapping construction with simultaneous scheduling and placement
- graph transformations during the mapping construction
- stochastic pruning with adaptive threshold that allows to empirically always find a solution
- support for multi-cycle operations and address generation unit

The results show that a CGRA can effectively make use of ILP and DLP of an application, and can bring up to 10× in performance gain on execution time or energy efficiency, compared to a RISC-V based CPU.

The contributions presented in this chapter open up new research directions. On hardware organisation side, a cluster (or island) based standalone CGRA can be studied. On compiler side, compilation techniques that mix modulo scheduling with our control-flow approach are needed. Runtime compilation techniques can also be studied to share the ressources of the CGRA between several applications. The perspectives, particularly with artificial intelligence applications, are discussed in chapter 7.

5. Exploiting DLP and TLP with multicore processors

A single processor can make use of ILP and DLP. Parallel processors can furthermore exploit Task Level Parallelism (TLP). The still ever increasing density of transistors in a single chip make it possible to have a multicore processor on a single die, sometimes referred to as Chip Multi-Processors (CMP). At the programming level, languages and models are needed to specify an application that takes the advantage of the available parallelism exposed by parallel machines. This chapter presents our contributions on mapping an application specified through the dataflow model of computation onto a multicore processor.

5.1 A warm-up on multicore processors and dataflow model of computation

5.1.1 Multicore processors

Parallel processor is a general term that includes all kinds of machines that can process data at the same time in the context of a single application. Parallel processors were historically built on several chips, each chip containing one central processing unit (CPU), like the one in figure 5.1. Since 2005, it is common to have several processors inside the same chip, as illustrated in figure 5.2, leading to the term *core* to be used for processor, and the term *multicore processor*, to name a processor composed of several processing units (or processing elements). In this context, the term CPU started to become outdated [154]. Since then, several terms emerged in the literature to designate basically the same thing: several processing cores inside the same chip. The word "Chip Multi-Processors" (CMP) explicitly holds the idea. The term "Multi-Processor System on Chip" (MPSoC), which includes heterogeneous architectures, is also very common. Finally, the word "many-core" appeared to designate architectures composed of several tens or hundreds of (implicitly homogeneous) cores. Today's parallel processors are still built on several chips, each chip containing now several cores. We did not target in our work such architectures. We focused on hardware architecture and organisation inside a single chip. In the rest of this chapter, and without loosing generality, we use the term "multicore processors" to designate several processing cores (no matter the number) inside the same chip. This section does not present a comprehensive view of

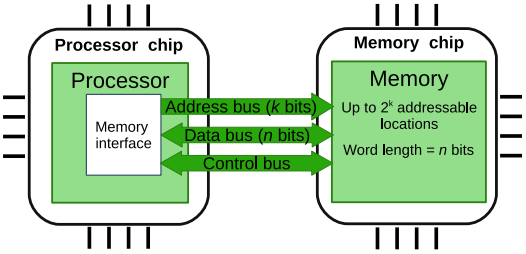


Figure 5.1 – Connection between only one processor with the memory

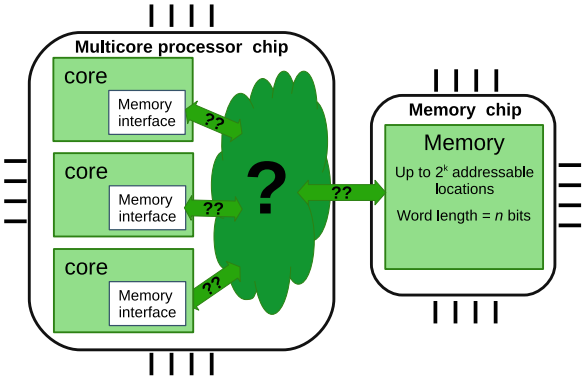


Figure 5.2 – How to connect the memory with a multicore processor?

existing multicore processors but presents the ones we used in the context of our work, and emphasizes on the key architectural features that are unavoidable when mapping a dataflow application. The interested reader is invited to read reference books [154, 196] or surveys on the topic [178].

Memory system

One of the biggest impact of moving from a single CPU (uniprocessor) design to a multicore processor is the interaction with the memory. Figure 5.1 shows a very simple connection between a single processor and the memory that contains both instructions and data, known as Von Neumann architecture. As the processor and the memory are on two different chips, and the communication bandwidth is low, some cache memories are added on processor side to improve the performances. If this same memory is now shared between several processors (or cores), some questions arise. How to interconnect them physically? What is the memory consistency model? Where to place the cache memories? What is the cache coherence support? These questions are symbolised in figure 5.2. Lots of multicore processor designs have been proposed differing from the way these questions are answered, which in turns, impact the programmability and the overall performances. This section briefly covers only two dimensions: the on-chip interconnect and memory consistency model.

On-chip interconnect

On-chip interconnect is in charge of the physical connections which allow communications between the processing elements and the memory. Many solutions have been proposed, such as bus, crossbar, ring, or Network-On-Chip (NoC), each having advantages and drawbacks. In our work, we considered multicore processors offering two widely used interconnects which are bus and NoC. The bus presents the advantage of simplicity, but is rapidly limited in bandwidth as the number of connected elements increases. In order to scale up to tens of connected element, the NoC is the right option, but comes at the price of design complexity, higher latencies, along with area and energy overhead.

Memory consistency

In the case of an uniprocessor interacting with the memory, the order in which the accesses to the memory are made is sequential. In the case of multicore processors interacting with a shared memory, memory accesses can be parallel (i.e. at the same time), and in different orders. The main issue is to find an order that leads to the correct execution of the application. A memory consistency model (or a memory model for short) defines *how the memory operations may be reordered when code is executing* [178]. Another definition is given in [62]: *the set of rules that defines the interaction between the processors to keep consistency of the value of the data.*

Two memory models are discussed here: the sequential model and the weak model. In a sequential memory model, all reads and writes to all addresses appear in the same order. This is ensured by the pair processing element-memory system (through atomic operations) which defines a global ordering of

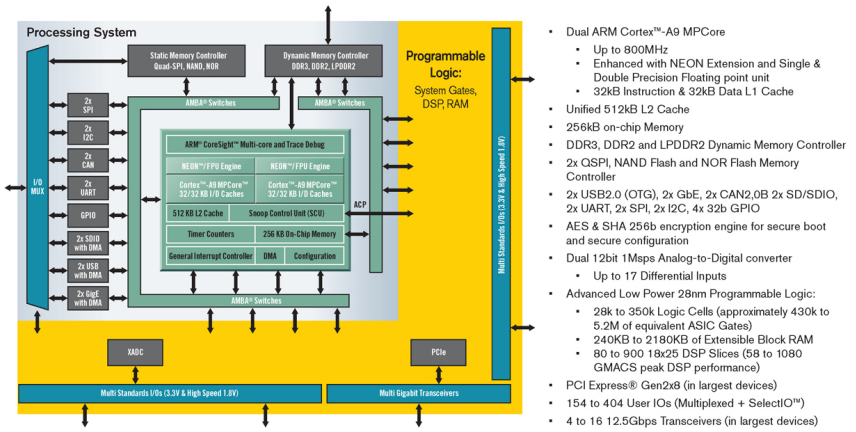


Figure 5.3 – Zynq-7000 Extensible Processing Platform (EPP) [2]

the accesses, usually involving also the interconnect. A sequential memory model makes programming easier but offload the burden to the hardware making the memory system more complex (and slower) and prevents from any potential performance improvements coming from another possible, yet consistent, sequence.

In a weak consistency model, there is no specific hardware that prevents a processor from accessing to the memory. It is the responsibility of the programmer to arrange the memory accesses such that no memory conflicts or inconsistencies occur. The weak model pushes the complexity to the software, but opens the door to optimized interleaved memory accesses that lead to better performances.

Homogeneous vs heterogeneous multicore processors

A multicore processor is composed of several *cores* or *processing elements* (PE), which can be all of same type, building then a homogeneous platform, or of different types, building then a heterogeneous platform. Heterogeneity can come from several features: the instruction-set architecure (ISA), custom instructions in the cores, or dedicated hardware accelerators. Figure 5.4 shows such an example with different co-processors based on another ISA than the ARM processor.

Examples of commercial platforms

A (multicore) processor needs to interact with the external world, additionnally to external memory, through peripherals and IOs (Input/Output). A “platform” is a multicore processor with its surrounding external connections.

Zynq platform

The Xilinx® Zynq™-7000 Extensible Processing Platform (EPP) combines an industry-standard ARM processor-based system with Xilinx 28nm programmable logic (FPGA) in a single device. The processor boots first, prior to configuration of the programmable logic. This platforms offers the flexibility to system and software architects and developers to design new solutions [2]. Figure 5.3 shows a schematic view of the Zynq platform, with the hard-wired “Processing System” part, composed of the ARM processors, peripherals, local buses, memory controllers, and the reconfigurable “Programmable Logic” part which is essentially an FPGA. The FPGA can be configured to support custom hardware accelerators, or *soft-core* processors. A *soft-core* is a processor that can be built from the reconfigurable resources of the FPGA. They are usually configurable in several ways: hardware multiplier/divider, number of pipeline stages, size of the cache, etc. A *soft-core* can also be extended by custom instructions. The Microblaze is the proprietary soft-core from Xilinx. The NiosII is the proprietary soft-core for Intel FPGA (formerly Altera). Since the thundering advent of RISC-V ISA, several open-source RISC-V based soft-cores are available for an implementation on any FPGA device. The Zynq platform offers the ideal playground to prototype some heterogeneous multicore processor platforms.

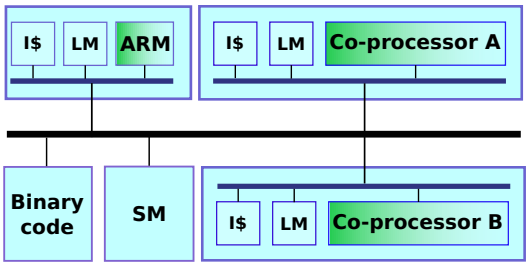


Figure 5.4 – Bus-based heterogeneous multicore processor platform

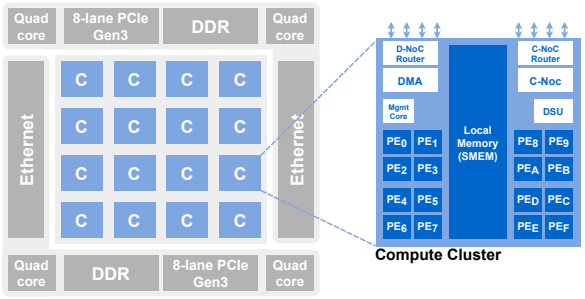


Figure 5.5 – A NoC-based many-core architecture such as the MPPA from Kalray

Kalray's MPPA®

Multi-Purpose Processor Array (MPPA®) is an example of a homogeneous architecture, and implements hierarchical computing resources featuring 18 NUMA nodes, 16 compute clusters and 2 IOs subsystems, interconnected with a NoC. Each compute cluster consists of 16 VLIW cores and 2 MB of *scratchpad-only memory*. The MPPA is a DMA-based manycore architecture. All computations are driven by DMA data transfers over the NoC and the software runtime is in charge of configuring the DMA NoC interface. Indeed the compute clusters can access the main memory (DDR) and the memory of other compute clusters *only* through the NoC and explicitly by software, making the MPPA follow a weak memory consistency model. Because of these features, programming efficiently the MPPA is challenging as all communications have to be managed explicitly by the software, and thus written by the developer. Figure 5.5 shows a schematic view of the MPPA.

This section briefly introduced multicore processors. In our contributions described later in this chapter, we considered both heterogeneous and homogeneous platforms, bus-based and NoC-based architectures.

5.1.2 A warm-up on dataflow model of computation

From a general point of view, a model of computation (MoC) provides a theoretical and formalized framework that defines how output data is produced from input data, including how data is processed, stored, and communicated. Models of computation are very interesting and useful from a theoretical point of view as they allow to formally proof some properties on a programme that respects the model, independently of the implementation or the technology. Many models of computations exist, and classifications have already been proposed [129]. The dataflow model of computation is such a class where data *flows* through computational units. A usual way to represent a dataflow application is through a graph, where nodes represent the computational units, called *actors*, and edges represent the connections between the actors through unbounded (theoretically infinite) FIFO (First-In First-Out) *buffers*. Actors exchange data samples, called *tokens*. When the required number of tokens is present as input, the actor can execute, or *fire*, producing then some output tokens stored in the output buffer. The set of rules that defines when an actor can fire is called *firing rules*. Note that in the formal model, only input tokens are needed to fire an actor. On real devices, the buffers are bounded, and the space available in the output buffer also becomes part of the firing rule in the implementation.

As a dataflow programme is usually in the form of a graph, a convenient way is to represent it graphically, through a network of actors, like the example provided in figure 5.6. The figure shows a network of seven actors, starting from actor `0` to actor `6`, with a *flow* going from left to right. An actor may have any number of input or output FIFOs. A dataflow application naturally expresses temporal and spatial parallelism of an application, making this approach an ideal candidate for specifying parallel applications.

There exists many dataflow models of computation that can be classified into two main categories of dataflow MoC: static and dynamic. This section puts the emphasize on three examples of MoC, that we

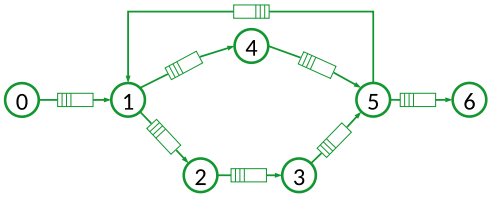


Figure 5.6 – An example of a dataflow application, presented through a network of actors

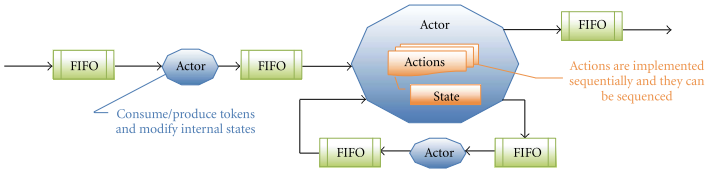


Figure 5.7 – The DPN actor model

used for our experiments. The interested reader can refer to existing surveys for a comprehensive overview of either static MoCs [136] or dynamic MoCs [85, 131].

Synchronous DataFlow (SDF)

The family of static dataflow MoCs can be characterized by the common feature of a constant rate in the consumption and production of tokens, which offers interesting properties like determinism, decidability, and compile-time optimizations. Synchronous DataFlow (SDF) [223] is certainly the most used and studied static dataflow model. The *Synchronous* term relates to the fixed value of the rate of generation and consumption of tokens, i.e., all the actors produce/consume a fixed number of tokens in the whole graph. This model has significant advantages. First, high analyzability, which means it can be analysed at compile time. Second, the determinism of the firing rules provides the opportunity to compute statically the memory usage (size of the buffers), and optimally map and schedule the application [194]. However, SDF, as a static model, cannot capture the dynamic behaviour of some applications, e.g. data-dependent video coding and decoding.

Dataflow Process Network (DPN)

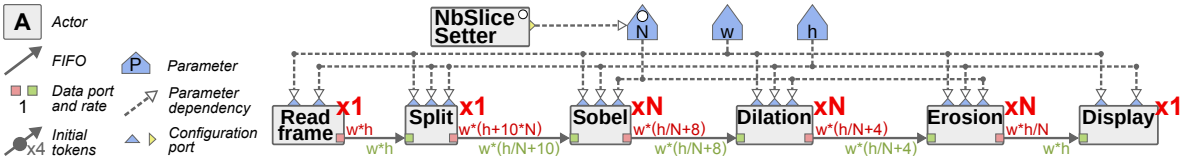
In contrast to static model, dynamic dataflow models are able to capture the behaviours of dynamic applications. Lee et al. were pioneers in a theory of dataflow process network (DPN) [218]. DPN is related to Kahn Process Networks (KPN) [225] but it can be used to model the most general form of dataflow MoCs. In a KPN model, reading is a *blocking* operation. An absence of tokens forces the actor to wait. The DPN model adds *non-determinism* to the KPN model, by allowing actors to test the presence or absence of tokens in their input buffers. A DPN actor is thus never blocked. It fires, or not, an action according to the number of tokens in its input buffers.

Figure 5.7 illustrates a network of actors that follows DPN semantics, and zooms inside an actor showing that it can implement several actions, and can have an internal state. At runtime, according to the number of tokens in the input FIFOs, the internal state, or even priorities between actions, one of the actions is fired. With such a model, it is impossible to know the production and consumption rate of actors at compile time since each actor may have a set of actions, each having its own set of firing rules, and can be fired if one of them is satisfied.

Parameterized and Interfaced Synchronous Dataflow (PiSDF)

In our work, we also used the PiSDF MoC [134], a reconfigurable dataflow MoC whose semantics is depicted in Figure 5.8. Reconfiguration in the PiSDF MoC is based on parameters, which are nodes of the graph associated to rate configuration parameters. These production and consumption rates of actors can be specified with expressions depending on these parameters. Following PiSDF execution rules [134], an actor may trigger a reconfiguration of the graph topology and intrinsic parallelism by setting a new parameter value at runtime.

Figure 5.8 depicts the graphical elements of the PiSDF semantics and gives an example of a graph implementing a video filtering algorithm. At each iteration of the graph, which corresponds to the processing of a new frame, the *SetNbSlice* actor triggers a reconfiguration of the data rates by assigning a new value to parameter N. Reconfigurations enable a dynamic variation of the number of parallel



(a) PiSDF MoC semantics (b) PiSDF graph of an image filtering application

Figure 5.8 – PiSDF MoC semantics and example

executions of the *Sobel*, *Dilation* and *Erosion* actors.

Dataflow frameworks

A dataflow framework provides a full environnement to specify, program, compile, synthesis software or hardware, and sometimes functionally simulate a dataflow application by gathering the set of tools, compilers, editors and methodologies. There exists numerous frameworks that are not listed here. Only the two frameworks used in the context of our work are presented.

Open RVC-CAL Compiler (ORCC)

The Moving Picture Experts Group (MPEG) has introduced the Reconfigurable Video Coding framework (RVC) [158], which offers reconfiguration, reusability and platform independent dataflow models. An RVC codec is described by using a domain-specific language, called CAL Actor Language (CAL) [200]. ORCC is an open-source toolkit dedicated to develop RVC-CAL applications [5, 177]. ORCC is a complete Eclipse based Integrated Development Environment, which aims at providing a compiler infrastructure to allow software/hardware code to be generated from dataflow descriptions. The compiler is able to translate RVC-CAL applications into an equivalent description not only in software but also in hardware languages for various platforms (FPGA, DSP, GPP, etc). In consequence, there are numerous back-ends in ORCC that target different languages (C, C++, LLVM, VHDL, Verilog, etc.). In our work, we mainly used the C backend, which produces an application described in portable ANSI C (Windows, Linux, Mac) like Pthreads with multi-core ability.

Parallel and Real-time Embedded Executives Scheduling Method (PREESM)

The Parallel and Real-time Embedded Executives Scheduling Method (Preesm) is a rapid prototyping framework that provides methods to study the deployment of SDF, IBSDF, and PiSDF applications onto multicore processors. Preesm is a set of open-source plugins for the Eclipse Integrated Development Environment (IDE) [117, 124]. In Preesm, the code of the actors is directly written in C language. The framework provides the graphical user interface to define the network of actors. Like ORCC, Preesm provides several back-ends for different target architectures (x86, TI Keystone). The C backend produces an application described in portable ANSI C (Windows, Linux, Mac) like Pthreads with multi-core ability.

SPIDER *Runtime*

SPIDER was originally introduced in [120] as a runtime manager for the execution of PiSDF graphs on heterogeneous MPSoC. The internal structure is built on two types of processes, each responsible for managing the cores which they are mapped on, and adopting a master/slaves model. The GRT (Global RunTime) is the master of the system: it manages the PiSDF graph topology and takes mapping and scheduling decisions. It is usually implemented over a general purpose core. The GRT can also process actors. The LRT (Local RunTime) are lightweight slave processes that execute actors. LRT can be implemented over heterogeneous types of PE: general purpose or specialized processors, accelerators.

SPIDER executes the following steps to run an actor. First, the GRT schedules an actor on a PE of the architecture, and sends the execution order through the dedicated *job queue* of the LRT of this PE. A *job* is a message that embeds all data required to execute one instance of an actor: a job ID, location of actor data and code, and which are the preceding actors in graph execution. When an LRT starts an actor

execution, it waits for data tokens to be available in the input FIFO specified in the job message, among a pool of data FIFO. On actor completion, data tokens are written in output FIFO, and the LRT sends new parameter values, if any, and execution traces back to the GRT for reconfiguration, monitoring and debugging purposes. Each LRT is associated with a *job counter* that stores the integer job ID of the last executed job. As the job IDs increase monotonically both with scheduling order and data dependencies between jobs, these job counters can be used for synchronization purposes between LRT, to check whether an LRT already executed a given job.

Open-source implementations of the SPIDER runtime have been proposed for general purpose x86 architectures, Texas Instruments' Keystone digital signal processor architectures, and Xilinx's Zynq heterogeneous platforms [120].

This section briefly introduced the dataflow model of computation, which is used as the way to specify the applications we consider in the rest of this chapter. We used three models of computation: SDF, PiSDF, and DPN.

5.2 The mapping problem

Given a multicore processor and a dataflow application presented in previous sections, the main contribution presented in this chapter is about how to map the application on the target platform.

? Mapping dataflow applications on a multicore processor

How to map a dataflow application on a multicore processor?

Basically, mapping a task-based parallel application on a multicore processor amounts to placing the tasks on the processors, such that a given cost function is optimized. The mapping problem can be formalized as a graph partitioning problem, where the partitions are the processors, and the tasks the nodes to be placed, and as such is an NP-complete problem [220]. The edge-cut, the number of links between two partitions, can model the communication between the processors. Existing methods on graph partitioning from the literature can be used, such as the ones implemented in the graph partitioning tool METIS [4].

METIS - Graph partitioning tool

METIS [4] is a well-known graph partitioning tool, developed at the University of Minnesota and distributed open source. The algorithms implemented in METIS are based on the recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes [214]. This tool quickly produces high quality partitions for a wide variety of irregular graphs [213]. Since METIS supports multi-constraint partitioning and allows for target partition weights, it can be adapted to compute partitions that balance the computations on heterogeneous architectures. Furthermore, the partitioning objective in METIS is to minimize the edge-cut so the communications among the processors are also minimized.

Some generalities about application mapping are given before a focus specifically on mapping dataflow applications. A problem formulation is then given.

5.2.1 Generalities

The question of mapping parallel applications on multi or many-core architectures is a very wide problem, with a large number of dimensions, including the programming model, the target architecture (homogeneous or heterogeneous, bus-based or NoC-based, etc.), and the optimization goal (throughput, execution

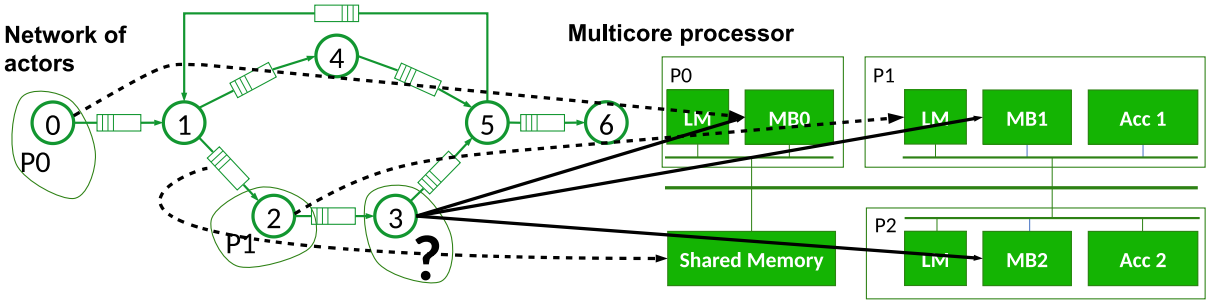


Figure 5.9 – Illustration of the dataflow mapping problem

time, energy, etc.) [145]. The interested reader can refer to the paper gathering different mapping strategies for NoC-based architectures [143]. Following the taxonomy proposed in [145], the mapping problem can be solved based on two main strategies: design-time, and run-time. When solved at design-time, the mapping is called *static* since it's computed offline and does not change while the application runs. This approach allows for exact methods to find an optimal solution [26, 57, 101], but suffers from a lack of flexibility since it cannot capture the dynamic behaviour of some applications. Moreover, even in the case of deterministic execution times of actors in a static context, the paper [30] interestingly shows the difference between the optimal mapping obtained from a well-formalized problem and the real execution trace, due to execution variabilities coming from the hardware.

The dynamic workload should be handled using run-time techniques. The run-time mapping strategies can themselves be divided into two categories: *on-the-fly* mapping, or *hybrid* mapping. On-the-fly mapping techniques are application- and platform-agnostic and solve the problem online. Very simple and efficient heuristics should be used to shorten the response time. For NoC-based MPSoCs, various fast heuristics targeting the reduction of communications under constraints have been already proposed [118, 170, 174]. These approaches consider one task per core. Allowing multiple tasks on one core is considered in [107]. Heuristics are fast but can be far from optimal solutions, so hybrid approaches have been introduced. They are based on pre-computed optimal solutions for a set of cases. The job is split into two phases: (1) at design-time, a set of solutions is computed, and (2) one solution is selected at run-time. A wide variety of approaches can then be cited: based on traces in [104], on priority in [168], on scenario in [141], on previously identified design points in [146], or on WCET and scheduling in [97]. The proposed real-time mapping reconfiguration method in [75] requires to suspend the currently running application and the manager remaps the tasks at run-time according to scenarios previously defined at design-time based on the evaluation of multiple mappings, optimizing for their resource requirements and power consumption. Finally, a last approach can fall into the family of hybrid mappings, which considers to recompute (partially) the mapping problem at run-time. This is called *run-time remapping*. The work presented in this chapter presents two flavours of such an approach for dataflow applications. Section 5.4.1 presents a runtime manager to map reconfigurable dataflow application (PiSDF) on the MPPA. Section 5.4.2 presents a move-based algorithm to map a dynamic dataflow application (DPN) on a NoC-based heterogeneous many-core platform.

5.2.2 Focus on the dataflow model of computation

Figure 5.9 illustrates the dataflow mapping problem, with a network of seven actors to be mapped on a three-core processor with a shared memory. In the figure, Actor `0` is mapped on `P0`, and actor `2` is mapped on `P1`. The question is where to place actor `3` (and the other actors)? In the figure, all FIFOs go in the shared memory.

This section presents the existing methods for mapping dataflow applications. In [147], the mapping is modeled as a graph partitioning problem, and the problem is solved at run-time by METIS tool, based on profiling information obtained by a first run. Though the migration cost of the actors is not taken into account, the results are promising and could be improved if the mapping does not change completely at

Table 5.1 – Various approaches for mapping dataflow applications. IPC: Inter Processor Communication

Reference	Year	MoC	Platform		Communication	Mapping
Stuijk et al. [195]	2006	SDF	Fixed	Homogeneous	N/A	Static
Singh et al. [146]	2013	SDF	Generic	Heterogeneous	NoC	Hybrid
Lin et al. [155]	2012	SDF	Generic	Heterogeneous	IPC	Hybrid
Lee et al. [138]	2013	SDF	Generic	Homogeneous	NoC	Hybrid+R
Schor et al. [156]	2012	KPN	Fixed	Homogeneous	NoC	Hybrid
Castrillon et al. [149]	2012	KPN	Fixed	Heterogeneous	Yes	N/A
Castrillon et al. [132]	2013	KPN	Generic	Heterogeneous	Yes	Hybrid
Quan et al. [112]	2015	KPN	Generic	Heterogeneous	Constant	Hybrid+R
Quan et al. [141]	2013	KPN	Generic	Homogeneous	Constant	Hybrid
Stuijk et al. [167]	2011	SADF	Fixed	Heterogeneous	N/A	N/A
Yviquel et al. [147]	2013	DPN	Generic	Homogeneous	Constant	Hybrid
Our work [93, 122]	2015	DPN	Generic	Heterogeneous	Yes	Hybrid+R
Our work [14]	2022	DPN	Fixed	Heterogeneous	NoC	Hybrid+R
Our work [78]	2018	PiSDF	Fixed	Homogeneous	NoC	On-the-fly

each iteration. Some work proposed to take into account the communication cost between actors. In [155], additional actors, namely send and receive, are bound on the buses in addition to original computation actors that are bound on processors. The approach relies on an ILP formulation. Our goal is to embed the application to be typically executed by an embedded processor and the solving time is not compatible with these constraints. In [146, 155], the delay for a token to be transmitted is constant.

In [112], the application is specified with KPN (Kahn Process Network) and the target architecture is a shared-memory based MPSoC, with also a model of the communication channel (bus or NoC). The approach proposes to rely on three main steps: the two usual design-time preparation and run-time mapping steps plus a new customization step. The design time step computes a set of candidates and fills a database. The run-time mapping initialization derives from the candidates a new initial mapping for the given workload. Finally, the run-time customization step incorporates a Scenario-based run-time Task Mapping (STM) algorithm that is applied to find new mapping of tasks when the system detects that an objective is unsatisfied. It first detects the so-called *critical task* and then identifies why it misses its objectives: either poor locality or load imbalance. In case of poor locality, an algorithm that considers the communication between tasks is used to find a new mapping. In case of load imbalance, a load balancing strategy based on computational demands of the tasks is used. This step produces a new mapping that may move several tasks, which leads to a (re-)mapping overhead.

Table 5.1 lists the approaches reported in literature about the mapping of dataflow applications over different kinds of platforms. The approaches consider either static models (SDF), KPN or scenario aware dataflow (SADF) applications. The table reports the kind of platform. We call “fixed” platforms when the algorithms embed some specific features of the platform and make them tailored to it. On the contrary, a “generic” solution usually relies on an architectural model or an implicit simple model. A homogeneous platform is composed of processing elements of exactly the same kind, meaning that all actors run equally on each of the processing elements. A heterogeneous platform contains specific hardware or custom extensions which accelerates the execution of some actors. The “communication” column shows the approaches that consider communication cost into account. Some works assume an NoC, others a constant communication time. The last column shows the mapping category: static, hybrid, or hybrid with remapping (Hybrid+R).

Mapping dataflow applications on a multicore processor

How to map *at runtime* a dataflow application on a multicore processor?

5.2.3 Problem formulation

Solving the problem of mapping dataflow actors on a multicore processor relies on two models: (1) the application model, (2) the architecture model. Other constraints or parameters can complete the problem formulation.

The application model

In the context of dataflow application, the application model is clearly formalized in the framework of the model of computation. The dataflow application holds lots of interesting information like the number of actors, the number of FIFOs, the size of the tokens, the number of tokens in the case of a static model, etc., along with the explicit parallelism.

The architecture model

In its simplest form, an architecture model can be reduced to just a number of processors sharing a single and shared memory. In PREESM, the system level architectural model (S-LAM) is made available to the developers to specify their target architecture [180].

Complementary constraints or parameters

The complementary constraints and parameters that are useful when solving the mapping problem include the execution time of the actors, the code size, the memory, profiling data, energy profile, etc. In PREESM, the "scenario" model allows to specify the execution time for each actor, to force or forbid some bindings, to add an energy profile.

Mapping problem definition

Mapping problem definition

Given an application specified in known dataflow model of computation, and a multicore processor, place the actors on computational resources and the FIFOs into memory resources, such that the application executes as fast as possible or respects some quality of service (e.g. throughput).

Two main contributions are now presented in this chapter. The first one concerns model-based designs and mapping of dataflow applications. The second one concerns algorithms for runtime mapping of dataflow applications.

5.3 Model-based design and mapping of dataflow applications

This section presents our contribution in methodologies and tools for mapping dataflow applications. Methodologies and tools are needed in all steps of the application design and the mapping process. Model-based approaches allows to raise the level of abstraction to the designer to ease the development and improve the software productivity. We used model-driven engineering techniques to develop software tools and frameworks. We also used architectural models to abstract some hardware features of our target platforms. These two usages are now described.

5.3.1 Model-based design of dataflow applications

In our work, and for the experiments, we used several frameworks, all based on the Eclipse Modeling Framework (EMF) (see section 5.1.2). For dynamic dataflow applications, we used the Open RVC-CAL Compiler (ORCC) framework [5]. For static and reconfigurable dataflow applications, we used PREESM framework [124][1]. When languages, tools, or frameworks are designed following a model-based approach, the question of evolution and reusability of software components arises. Designers of domain specific

[1]Note that the CGRA compiler presented in the previous chapter relies on GAUT HLS tool, also developed with EMF technologies.

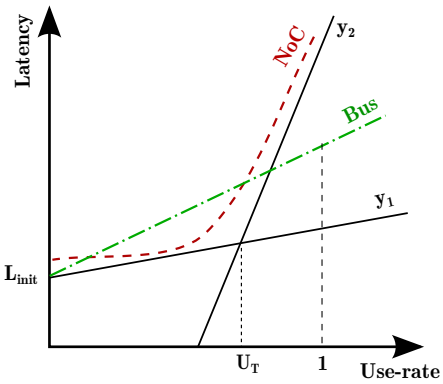


Figure 5.10 – Communication model

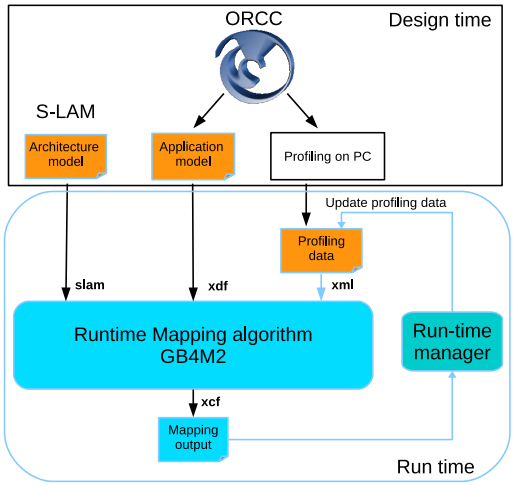


Figure 5.11 – Overall flow of our heuristic mapping algorithm

modeling languages (DSML) must provide all the tooling of these languages. In many cases, the features to be developed already exist, but it applies to portions or variants of the DSML. One way to simplify the implementation of these features is by reusing the existing functionalities. Reuse means that DSML data must be adapted to be valid according to the functionality to be reused. If the adaptation is done and the data are placed in the context of the functionality, it can be reused. The result produced by the tool remains in the context of the tool and it must be adapted to be placed in the context of the DSML (reverse migration). The thesis of Paola Vallejo [114] stood in this context, when reuse makes sense only if the migration and the reverse migration are not very expensive. The main objective of the thesis is to provide a mechanism to integrate the migration, the reuse and the reverse migration and apply them efficiently. The main contribution is an approach that facilitates the reuse of existing functionalities by means of model migrations. This approach facilitates the production of the tooling for a DSML. It allows reversible migration between two DSMLs semantically close. The user is guided during the reuse process to quickly provide the tooling of his DSML. The approach has been formalized and applied to ORCC, the RVC-CAL dataflow framework.

As the migration round-trip has been formally defined [106], it is possible to state and prove formal properties about the correctness of reuse, through a set of tests. For instance, we noticed that the existing specific ORCC flattener did not pass all the tests because of groundless remaining object, namely ports related to *composite structures*. The details about the full methodology and tools are available in [113].

5.3.2 Model-based mapping algorithm

Back in 2012, early works on actor mapping focused on computational features, while systematically ignoring the communication cost between two actors that are mapped on two different processors. We believed that solving the mapping problem under these assumptions ignores most of the execution time which is due to data movement and memory issues. Solving the mapping problem at runtime while considering memory issues and migration costs for heterogeneous platforms was the main topic of the thesis of Dinh Thanh Ngo [108].

Communication-model based run-time algorithm

In our approach, we consider that the communication delay between two actors mapped on two different processors depends on the traffic on the interconnect. Indeed, the latency of the bus increases with the number of processors connected to it [193], but we also estimate the delay for data to be transmitted. It is likely that this delay increases with the traffic. This is typically observed in NoC [135], where the latency increases with the injection rate.

In our approach, the communication model gives the relationship between use-rate and latency since the

use-rate can be estimated at run-time according to mapping decisions or estimates. We propose a generic and parametric communication model as shown in figure 5.10.

The aim is to have a unique model that can fit with different communication standards, which are supported in the platform.

A model is based on two linear functions y_1 and y_2, the intersection between the two curves correspond to the saturation threshold (U_T). These functions can express the communications of a NoC, i.e. the red curve in Fig. 5.10, or a bus, i.e. the green line where $y_1 = y_2$. In practice the parameters of y_1 and y_2 can be adapted online according to the monitored data as depicted in figure. 5.11.

Figure 5.11 presents an overview of our mapping methodology and flow. It relies on the information from the application model, architecture model, and profiling data. The goal is to find in a (very) short time a mapping solution, that can be refined while the application is running based on the updated profiling data. The algorithm follows three main steps that are now described :

1. Initialization phase: a processor budget is first computed to coarsely bound the execution time allocated to each processor (to avoid the solution where all the actors are mapped on the same processor) for a theoretically balanced solution when communication cost is not considered.
2. Computation phase: with the aim to obtain a good balance between the processors, we introduce a factor α ($0 < \alpha \leqslant 1$), which represents a ratio of the processor budget. The actors are sorted with a fast sorting algorithm (a bubble sort in the current version) in a descending order based on the value of their total computation cost. Then, for each actor from the list, the algorithm selects the best processor according to the minimum execution time, until the processing use of all processors is greater than the allowed budget defined by α.
3. Communication phase: this step of the algorithm deals with the remaining actors. These are now sorted according to their amount of incoming and outgoing data. The idea is to take the communication cost into account in this phase. Hence, the algorithm takes the first actor in the second sorting list and considers all the connections with other actors in the network. If this actor and its adjacent actors are mapped on the same processor, the communication time is zero. In order to deal with unknown information when we estimate the use-rate of the bus (% usage of available bandwidth), we introduce a factor β that represents the ratio of remaining data transfers associated to unmapped actors that will use the communication media (e.g. bus). Then the latency is estimated with the communication model (Fig. 5.10) and the algorithm makes a decision of mapping. This procedure is applied to all the remaining actors.

Need for Speed: Payback

The algorithm was implemented for Xilinx Zynq-7000 and first simulated through the Cadence Virtual System Platform (VSP) [7]. VSP allows to virtually design a functional platform. The algorithm has been experimented for synthetic graphs, and for two real-life video decoding applications, MPEG4 Part 2 (or MPEG4-SP Simple Profile), and HEVC (High Efficiency Video Coding). These applications, coming from the ORCC application repository, are specified following the DPN model with the ORCC [5] tool as in [147]. We used the C back-end for software synthesis. In order to evaluate our heuristic approach more thoroughly, we make the comparison with METIS tool [4] both in terms of throughput and in terms of solving time. Only the results on the video decoders for one video sequence each are presented here. The full set of results is available in [122].

Table 5.2 describes the properties of MPEG-4 and HEVC, as well as the number of actors and FIFO channels. Options in the tool can allow for generating additional code for intrusive profiling in order to perform the design-time profiling step presented in figure 5.11. For the profiling, we specify the size of the FIFO channels used for communication between 512 and 8192 bytes. We introduce some usual hardware accelerators that can provide significant speed-up when associated with Microblaze (MB) softcore on Xilinx FPGAs. Our objective is to test the flexibility and efficiency of our mapping algorithm for different

Table 5.2 – Properties of different tested MPEG video decoders

Decoder	Profile	YUV	#Actors	#FIFOs
MPEG-4 Part 2	SP	yes	41	104
HEVC/ H.265	Main	no	27	185

Table 5.3 – Accelerators used in different platforms

Platform	MB1	MB2	MB3	MB4	MB5	MB6	MB7
7.1	Merger	IDCT	Parser	Inter	IQ+IAP	Add	IDCT
7.2	IQ+IAP	IDCT	Parser	Inter	IDCT	Merger	IDCT
7.3	Parser	xIT	Intra	Inter	Merger	DPB	xIT
7.4	Merger	xIT	Intra	Inter	xIT	Parser	xIT

heterogeneous architectures corresponding to different possible terminals, so we consider in this example four different platforms based on 1 ARM and 7 MB processors. By using S-LAM modelling, we specify 4 different heterogeneous platforms detailed in Table 5.3.

Table 5.4 shows the results obtained by our algorithm called GB4M2 compared with METIS partitioning tool, on the average throughput obtained for the videos and the solving time needed by the mapping algorithms.

The results show that our algorithm performs similarly or better regarding the throughput, for a significantly better solving time, offering thus a better trade-off between performance (throughput) of the application and the solving time.

Need for Speed: Shift

The runtime mapping algorithm has also been tested on a hardware prototype. Figure 5.12 shows a bus-based heterogeneous MPSoC that has been implemented in the Zynq platform [109, 123]. The cluster with the ARM processor is a *hard-core*, while the co-processor clusters are based on Microblaze *soft-core*. Some performance monitors (PMs) and timers are added to monitor the running application. The microblaze processors have local memories (LM) to store the instructions and temporary data. The clusters share an *on-chip* memory built on the memory resources of the FPGA. The ARM processor is used as the Master PE, while two clusters of 8 Microblazes each are synthesized to play the role of slaves. On the PS zone, a DDR memory is split into two sections. The first 512Mb are private to the ARM and the last 512Mb are accessible to the Microblazes. On the FPGA, one Microblaze (MB0) is connected to 512Kb of private memory through the Local Memory Bus (LMB). The other slaves are connected to 16Kb. In addition, 4Kb of BRAM are shared between each Microblaze and the ARM. Also, 128Kb of BRAM are accessible to all the slaves. The accesses to shared memories are done via AXI Interconnect elements which are probed by Performance Monitors (PM).

Table 5.4 – Results with MPEG4-SP and HEVC mapped on 8 processors in the heterogeneous platform

Video CODEC	Input video	Platform	**Throughput**			**Solving time**		
			GB4M2	METIS	Speed-up	GB4M2	METIS	Speed-up
MPEG4	Hit	7.1	49.26 fps	46.70 fps	1.05x	4.25 ms	190 ms	44.66x
		7.2	48.00 fps	46.52 fps	1.03x	3.63 ms	190 ms	52.34x
HEVC	Kristen	7.3	10.38 fps	6.84 fps	1.52x	5.33 ms	130 ms	24.39x
		7.4	9.71 fps	9.78 fps	0.99x	4.99 ms	120 ms	24.05x

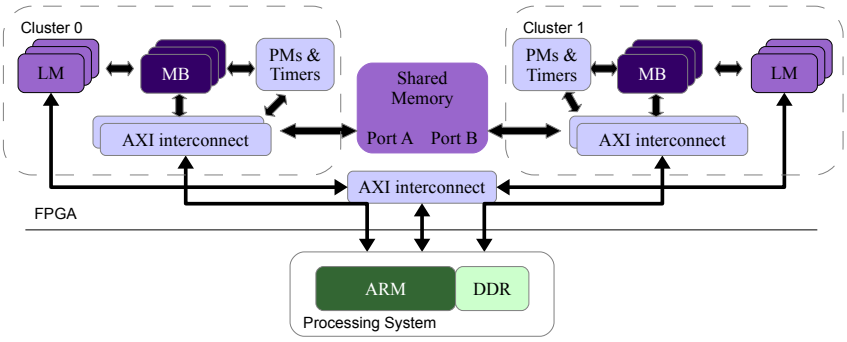


Figure 5.12 – A cluster-based architecture for the Zynq platform. MB: MicroBlaze, LM: Local Memory, PM: Performance Monitor.

Runtime kernel

The runtime is responsible for operating the system. It provides routines to initialise the system's components (i.e. timer, performance monitors) and to establish the communications between the Master and the Slaves. In our runtime, Master - Slaves communications are done using messages. Messages travel through software FIFOs that are implemented into the 4Kb shared memories. The runtime defines the messages that can be exchanged as well as the routines to interface with the control FIFOs (e.g. send a message, send data, read data, etc). Other functionalities are specific to the role of the processor.

Results

Table 5.5 shows the hardware resources used when implementing the prototype on a Zynq ZC706, which holds a XC7Z045 with 350 000 Logic Cells, 218 600 LUTs, 2 180 Ko of BRAM, and 900 DSP Blocks. This Zynq target can support the 16 microblaze processors and there is still room for dedicated hardware accelerators. The application tested was the MPEG4-SP decoder, with several video sequences. Two strategies for the runtime profiling were tested. One with the average time over the full video sequence (AT), and another one with a sliding window (SWT). Data is collected every ten frames. Figures 5.13 and 5.14 show the results for Foreman video sequence. The curves show the evolution in throughput for the two profiling strategies, and when no remapping is applied, compared to our runtime mapping algorithm.

We observe that remapping systematically improves the performances.

Considering the observation time, the results on Foreman tend to show that the sliding window strategy offers an interesting options. This trend is not confirmed by the other videos tested (not shown in this chapter) where the average time strategy performs slightly better.

We measured the execution time of the algorithm running on the ARM processor. The algorithm is run every 10 frames. Table 5.6 gives the average execution time. Recall that the complexity of the algorithm (thus the execution time) depends on the number of actors mapped on the processor to relief, and on the number of processors. We wanted to know if the algorithm always completes in a reasonable time. Table 5.6 also gives the variance of the execution time, showing that it is very low. The results show that this kind of algorithm is suited for embedded systems with soft real-time constraints. More details and results are available in [109, 123].

Need for runtime adaptivity

Based on the profiling information, we observed the variations of the workloads of each actor. The workload of an actor is defined as the ratio of its computation time over a given time window. Table 5.7 and 5.8 show the variations (as a percentage) between two different input video sequences for MPEG-4 decoder and HEVC decoder respectively. These tables just pick up a few actors in the decoder, which

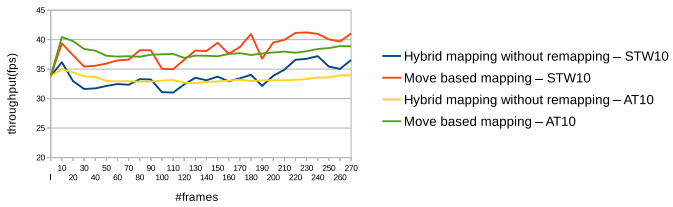


Figure 5.13 – Comparison of throughout for different mapping strategies for decoding Foreman sequence with MPEG4-SP

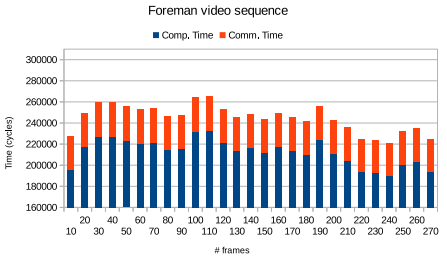


Figure 5.14 – Computation and communication time variations while decoding Foreman video sequence

Table 5.5 – Resources used on the Zynq

Resource	Utilisation	%
Slices LUTs + Registers	316 618	90
BRAM	391	71
DSP	105	11
Cortex A9	1	100
DDR 1Go	1	100

Table 5.6 – Execution time of the mapping algorithm and variance measured for different video sequences.

Sequence	Time (ms)	Variance
Foreman	43,54	0,001
Stefan	54,06	0,004
Coastguard	54,01	0,004

present the highest variations of the workloads. The results reveal up to 50% of variations for the same actor (though the IDCT actor is a purely static actor), showing that some actors are unequally requested according to the video sequence.

Besides, we also compared the resulting mapping solutions for different video sequences, and for the same target platform. The results show that the binding of some actors on the processors differs. This is true both for MPEG-4 and HEVC decoders. For MPEG4-SP, about 10% of the actors are mapped differently between *Foreman* and *Hit* videos. With HEVC decoder, the difference in actor mapping is 33.33% between *Kristen and Sara* and Four People videos, and 25.93% between *Four People* and *Johnny* videos. This difference can be explained from the *dynamicity* of some actors both in MPEG4-SP and HEVC decoders.

These results show that the same mapping shall not be used for all video sequences of a given decoder on a given platform.

Runtime adaptivity is needed to find the good solution of a three involved members problem (application, architecture, and profiling).

Table 5.7 – Example of variations in terms of workload for two different MPEG4-SP video sequences

Actor	% difference
decoder_texture_U_idct2d	32.73
decoder_texture_V_IQ	25.29
decoder_texture_V_idct2d	57.94

Table 5.8 – Example of variations in terms of workload for two different HEVC video sequences

Actor	% difference
HevcDecoder_xIT	11.89
HevcDecoder_InterPred_Inter_y	16.48
HevcDecoder_SAOFilter_Sao_U	10.32

5.4 Runtime mapping of dataflow applications on NoC-based multiprocessor architectures

This section now presents our contribution in runtime algorithms for mapping dataflow applications. Specifically, in the two following works, we consider NoC-based many-core platforms, and two different models of computation.

5.4.1 SPIDER on MPPA®

Recall that SPIDER was originally implemented on shared-memory based MPSoC [120]. The objective of this work was to demonstrate the feasibility and show the potential and flexibility of implementing a runtime for reconfigurable dataflow on a manycore architecture. This was the main task assigned to Hugo Miomandre during his internship, done in the context of a young researcher project called MORDRED, in collaboration with Karol Desnos from INSA Rennes.

Porting SPIDER on the MPPA®

Three main actions were needed to be performed in order to adapt the original code of SPIDER to run on the MPPA®:

1. Software explicit NoC communications
2. Runtime scheduling for a large number of processing elements
3. Distributed scratchpad memory allocation

The three actions are synthesized here, and the full details are available in [78].

Software explicit NoC communications

SPIDER was originally designed for shared memory MPSoC. Shared memory models are easy to use thanks to the global address space and the provided hardware synchronization mechanisms (atomics). As this is not the case for MPPA®, a new synchronization scheme was needed. We opted for a distributed synchronization scheme which limits the traffic overhead over the NoC. The proposed algorithm builds on the observer design pattern, where the *observers* are the LRT waiting for completion of a preceding actor, and the *notifier* is the GRT executing this actor.

Runtime scheduling for a large number of processing elements

The original scheduler implemented in SPIDER is a LIST scheduling heuristic. The issue with the LIST scheduler is that its complexity becomes prohibitively large when targeting a processor with hundreds of PE. Indeed, its complexity is given by $O(A.log(A)+P.(A+E))$, where A and E are the number of actors and dependencies in the DAG, and P is the number of PE. Manycore architectures implement hundreds of PE and require a lot of application parallelism. Therefore, the number of DAG actors to be scheduled in parallel increases roughly linearly with the number of PE. Consequently, the complexity of the LIST scheduling tends to increase quadratically with the number of PE, making it a bottleneck for runtime scheduling. We replaced the original LIST scheduler with a less complex scheduling algorithm based on a specialized round robin heuristic.

Distributed scratchpad memory allocation

On a scratchpad memory based clustered manycore architecture like MPPA®, the memory in the cluster needs to be allocated by the software. We implemented a thread-safe scratchpad memory allocator for managing the memory allocation in the scratchpad memory of a clustered manycore architecture. This allocation procedure ensures that all job scheduled on an LRT running in a cluster will succeed in allocating the required memory, as long as their required memory does not exceed the maximum capacity of the cluster memory space. When all firing conditions of a mapped actor are fulfilled, the LRT attempts to allocate memory using this algorithm.

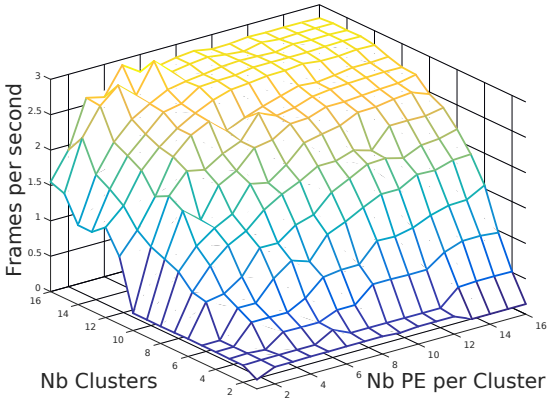


Figure 5.15 – Performance of the image filtering application for a 4K video on MPPA®

Results

Figure 5.15 shows the performance obtained when executing the image filtering PiSDF graph (Figure 5.8) on the MPPA® for 4K images. Application performance, expressed as the number of processed fps (frames per second), is plotted for a varying number of active clusters, and a varying number of active PE per cluster. The sequential performance on a single PE is 0.13 fps. When using the 256 PE of the compute clusters, a throughput of 2.81 fps is reached, which represents a speed-up of 22 compared to the sequential execution. During the processing of each frame (0.36s), only 8% of this latency is due exclusively to GRT computations. Hence, actor computations and NoC communications are responsible for 92% of the latency. In [89], the authors evaluate the performance of a static version of the PiSDF graph from Figure 5.8. In the static version, N is fixed and all mapping and scheduling are done at compile time for VGA videos (640x480). The top performance obtained for the static execution is 217 fps. For an identical video resolution, the reconfigurable PiSDF graph executed with SPIDER peaks at 47 fps. In addition to the SPIDER runtime overhead, the difference between the performance of the static and reconfigurable executions is mostly due to the lack of memory optimization in the reconfigurable implementation. In the reconfigurable version, many `memcpy` calls are issued to create the image slices in the *Split* actor and to merge processed slices into a contiguous buffer before *Display*. Thanks to compile time optimizations, these `memcpy` calls are replaced with pointer operations in the static version reducing the memory bandwidth drastically by a factor of 3. When using a standard thread-based implementation of SPIDER on an Intel Xeon E5-1650 with 6 hyper-threaded x86 cores clocked at 3.60GHz, the processing of a 4K video with the same PiSDF graph reaches 11.40 fps, using 95% of all CPU time. Although the performance on the Xeon processor is almost 4 times better, this processor dissipates on average 10 times more power than the MPPA®. Hence, the execution on the MPPA® is approximately 2.5 more energy efficient than on the Xeon.

5.4.2 Move-based mapping

This work stands in the context of ANR project COMPA, and was realised by Mostafa Rizk during his post-doc.

We observed in section 5.3.2 the variations between two video sequences for a given video decoder. In this section, we study this variation inside one video sequence. Indeed, according to the scene (fixed plan, moving background, object tracking, etc.), the needs from the decoder might change.

We thus proposed a *remapping* algorithm, an algorithm which starting point is a currently mapped and running application, and goal is to seamlessly refine the current mapping to improve the performances.

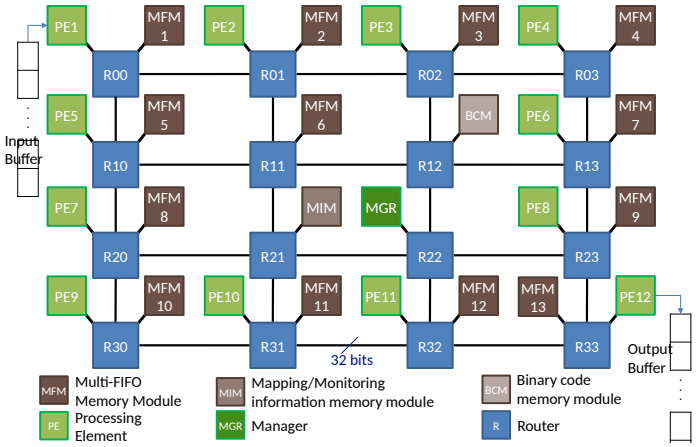


Figure 5.16 – The structure of the used NoC-based architecture

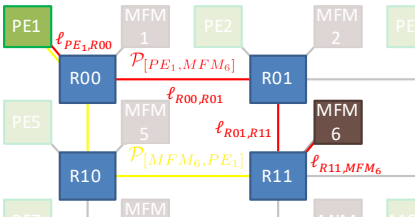


Figure 5.17 – Example on path declaration in the NoC

We believe that moving from one mapping to another comes with a migration cost, such that the overhead penalizes the expected gains. We proposed such an algorithm that takes into account the migration cost and allows only one actor to move at a time. The move-based algorithm has been studied with an analytical simulation and model for the interconnect [93]. This section presents a summary of our work on our move-based algorithm on a NoC-based platform with accurate simulation at NoC level. The full paper is available as appendix B.

Runtime remapping of dataflow applications

- take into account the migration cost
- only one actor can move at a time

NoC-based platform

The target architecture is a heterogeneous Multi-Processor System on Chip (HMPSoC) containing several different PEs and distributed shared memories connected through a Network-on-Chip (NoC). Fig. 5.16 presents the structure of the adopted NoC-based architecture. The target architecture is a 4×4 mesh-based NoC with 32-bit links composed of 15 memory modules, 12 PEs, and a processing element that acts as a manager (MGR). The PEs and memory modules communicate through the network using a network interface (NI). We consider a simple NoC model that employs the wormhole packet switching mode, the deterministic XY routing algorithm, and a flow control policy without virtual channels. The implemented routers have one buffer of 3 flits per input port and use distributed arbitration logic (one arbiter per port). The back-end part of the NI is typical and includes a packet maker/un-maker, which are used to assemble and disassemble the packets, and a priority manager to synchronize packet transmission and reception.

In this work, it is assumed that $PE1$ imports the incoming streamed data from an Input buffer and $PE12$ outputs the processed data, as illustrated in figure 5.16. It is assumed that any actor can be mapped on any PE. The used PEs can all work in parallel according to dataflow firing rules. However, some PEs are enhanced by hardware accelerators dedicated to certain functionalities in order to perform them more efficiently, and the PEs run with different frequencies. Table 5.10 shows the list of accelerators adopted in the simulation platform, and table 5.9 shows the frequencies for the PEs, relatively to the NoC operating frequency f. The shared memories are distributed in memory blocks which have a unique NI. From an NoC perspective, the novelty is the introduction of new command packets used as instructions to manage FIFO accesses, broadcast mapping information, collect monitoring data, and the transfer of binary codes.

Table 5.9 – Processing element operating frequency

PE ID	Operating Frequency
PE1, PE12, MGR	f
PE2, PE6, PE10	$2f$
PE3, PE7, PE11	$3f$
PE4, PE8	$4f$
PE5, PE9	$5f$

Table 5.10 – hardware accelerators used in the simulation platform

PE ID	Accelerated Function	Acceleration Ratio
PE3 & PE6	IDCT	1/0.3
PE4	IQ + IAP	1/0.75
PE10	Add	1/0.57
PE11	Interpolation	1/0.4

The manager is a PE dedicated to the following five tasks: (1) map initially the actors on the available PEs, (2) parse the feedback collected data from all modules (memories and PEs), (3) apply the run-time remapping algorithm and selects the actor to be moved (if any), (4) notify the corresponding PEs about the updated mapping and (5) manage the transferring of the binary code corresponding to the moved actor from the shared memory into the cache of the gainer processor. As the number of PEs is smaller than the number of actors, each PE is considered to run more than one actor. Hence, an actor scheduler is required to manage the order of execution of actors. In this work, the well-known round-robin scheduling technique has been adopted in all PEs. The actors are given the attempt to be executed in a circular order without priority. The PE checks the firing rules of an actor and keeps executing it while it can. When no action can be fired, the PE checks the rules for the next actor.

The platform integrates three types of memory modules:

1. Binary code memory module (BCM) which contains the binary code of the application
2. Mapping/Monitoring information memory module (MIM), which stores the mapping information (what actor to execute), and the monitoring information, collected by the PEs during the execution of the actors
3. Multi-FIFO memory module (MFM) which stores all the FIFOs of the application

Execution model

The simulator follows six steps to run the application:

1. Initial mapping
2. Monitoring actor execution
3. Collecting monitoring information
4. Estimating NoC communication time delay
5. Applying RR algorithm
6. Moving the actor to the gainer processor

The initial mapping step can be a random strategy or the exact method presented in [57]. FIFOs are mapped randomly and are approximately equally distributed on all memory blocks. The manager then informs the PEs, by means of dedicated packets not detailed here, to retrieve the mapping information and start the execution. While running, each PE monitors the actors, in particular computation time, communication time, and total number of tokens received to each input port. When the number of the processed frames meets the observed window, each PE node generates its own monitoring information packet and sends it to the MIM module, which centralizes the monitoring data for the manager. The next three steps are now more detailed.

Estimating NoC communication time delay

We believe that not only the communication time over the NoC is important, but also an accurate estimation of this time is needed. In the context of an NoC, with different paths between the different communicating elements (PEs, memories), and the XY routing algorithm assumed, the links between the routers might have varying workloads. In this work, two novel methods have been proposed to estimate

the communication time delay for transferring one token in the NoC. The first method is called the average-path token delay and it is based on finding the average delay for transferring one token depending on the path delays between all routers of the NoC. The second is called the average-link token delay and considers the time-delay of the token according to the used physical links connecting the NoC components while transferring the token.

Average-path token delay (APTD)

In this approach, a path is considered to be formed from the set of the interconnections between two specific nodes. As an example, Fig. 5.17 illustrates in red the path $\mathscr{P}_{[PE_1,MFM_6]}$ between processing element PE_1 to memory module MFM_6. As the deterministic XY routing is assumed, the packets always use the same path between the source node and the destination node. Recall that following the XY routing strategy, the packets transferred from a processing element p to a memory module m do not follow the same path used in transferring packets from the memory module m to the processing element p. Fig. 5.17 illustrates in red the followed path to transfer packets from PE_1 to MFM_6 and in yellow the followed path to transfer packets from MFM_6 to PE_1. The APTD strategy considers the same average time for all tokens through both paths (red or yellow).

Average-link token delay (ALTD)

A link is defined as the interconnection between two consecutive components of the NoC. A path is a set of links. Because the same physical link is shared, an average path delay might hide a heterogeneous workloads in each link. The ALTD strategy goes down to the link level, the communication time on the path being the sum of the communication time of each link constituting the path.

Move-based Runtime Remapping (RR) algorithm

The move-based algorithm is inspired from the Fiduccia and Mattheyses algorithm (FM) [224], well known for solving the balanced circuit bipartitioning problem. In the original algorithm, the algorithm starts with an existing partitioning (usually randomly obtained), and tries to refine it by evaluating the gain (or loss) obtained for each node placed in a partition when it is moved to the other partition. The one with the maximum gain is chosen to move to the other partition, and the algorithm iterates until the moves offer no more gain. We adapted the actor moves and gain concept to apply to our context.

The cost of remapping tries to balance between migration cost and performance improvement.

For each observation window (N_F frames), the manager executes at run-time the algorithm, which is divided into two main steps. The first step identifies the PE with the highest workload. All actors mapped on this PE are possible candidates for a move. Then, for each actor, the manager estimates the total gain achieved if the actor is moved to another PE, by estimating the new workload for each PE, including the computation time of course, but also the communication time (based on ALTD or APTD). The second step sets a tradeoff between the migration cost and the predicted improvement of the performance. The migration cost of an actor is the required time to transfer its binary code into the local memory of the new hosting processing element. It depends on the size of the binary programme required to be transferred and the communication-time delay in the network. Finally, the expected gain is estimated for all mapping combinations by finding the difference between the estimated performance gain and the estimated migration cost of the actor. The move that leads to the maximum gain is selected, and the corresponding actor is decided to move. When there is no gain, no move is made. The complete formalization of the problem and the detailed structures of the packets are available in appendix B.

Results

Experimental setup

The algorithm has been experimented for MPEG4 Part 2 (or MPEG4-SP Simple Profile) video decoder, implemented with ORCC [5]. The adopted NoC-based architecture is implemented in an in-house NoC simulator described in SystemC TLM model [205]. In order to accurately model the adopted application, all involved actions are functionally simulated to generate the real data exchanged by actors during video decoding. The SystemC model adopted in the simulation platform is cycle accurate at the level of the NoC and the network interfaces. The timing of all corresponding action executions on PE is compensated in the simulation according to the profiling data extracted while running the application on a reference computer. Profiling data provides, for each involved action, the mean value of the number of cycles required to execute it. In this work, profiling data has been extracted using on a desktop computer (i7-2620M CPU@2.7 GHz and 8GB memory). We consider that the NoC operating frequency f is 500 MHz. The clock cycle in each PE is determined according to Table 5.9.

Comparison of different mapping strategies

In order to determine the relevancy of the devised algorithm, it is compared to the STM method introduced in [112]. To achieve fair comparison, the STM method has been modeled and implemented on our devised NoC-based architecture. We have also implemented the exact method presented in [57] for the initial mapping, with two differences: we have used constraint programming instead of ILP, and the objective function is the maximum period as it is our optimization goal. The workload used for the computation time of the actors is based on the profiling of Foreman video. Simulations have been conducted while running the MPEG4 decoder to process real-life videos. The observation window considered is 10 frames.

Fig. 5.18 presents the achieved throughput in terms of frames per second (FPS) when decoding Foreman video (CIF format) for seven different mapping strategies:

1. *No Remapping*: initial random mapping kept all along the video sequence
2. *Remapping STM*: the method presented in [112]
3. *No Remapping Optimal*: initial mapping obtained from [57] kept all along the video sequence
4. *Remapping MB ALTD*: our move-based algorithm using ALTD starting from a random initial mapping
5. *Remapping MB APTD*: our move-based algorithm using APTD starting from a random initial mapping
6. *Remapping MB ALTD Optimal*: our move-based algorithm using ALTD starting from the optimal solution
7. *Remapping MB APTD Optimal*: our move-based algorithm using APTD starting from the optimal solution

Note that the "optimal" mapping corresponds to the best mapping found based on the profiling of Foreman video after a time out of one hour (like the original paper), and the optimality is not proven. The letter "M" shown on the curves represents when an actor move occurs. The results show that the MB algorithm, starting from a random mapping (without significant initial delay), performs better that the optimal with no remapping for Foreman video sequence in CIF format. As the optimality is searched for the Foreman profile, we used the optimal mapping as a starting point for the MB algorithm, and the results show that it further improves the throughput. We used the optimal mapping obtained for Foreman to decode other video sequence. The figures are not shown here but are available in appendix B. As expected, the mapping obtained for Foreman does not perform good for the Ice video sequence in 4CIF format and Grandma video sequence in QCIF format. But surprisingly, it performs good for the Bus video in QCIF format and Bus video in CIF format.

The results show once more that one does not fit all and runtime adaptivity is needed to find the good solution of a three involved members problem (application, architecture, and profiling).

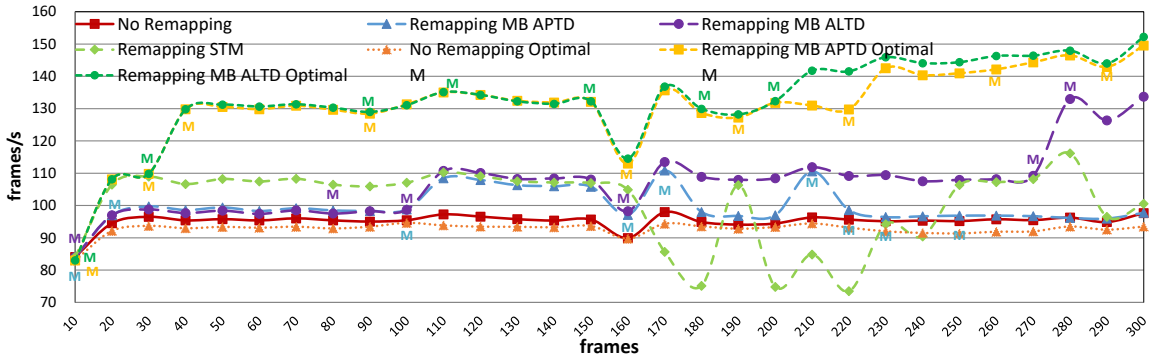


Figure 5.18 – Throughput in terms of FPS when decoding Foreman video sequence [1] using MB and STM remapping algorithms

Table 5.11 – Improvement in throughput (fps) for different remapping techniques

Video		Remapping Algorithm		
Sequence	Format	MB-ALTD	MB-APTD	STM
Foreman	CIF	11.4%	5%	4.1%
Bus	CIF	5.4%	5.4%	−17.7%
Ice	4CIF	26.1%	2%	−13.04%
Bus	QCIF	9%	8%	−20%
Grandma	QCIF	14.91%	14.11%	NA

Our move-based algorithm, restricting to move only one actor at a time if it is worth, offers the appropriate adaptivity.

Table 5.11 summarizes the comparison of average FPS achieved when processing multitude video sequencing while adopting different remapping techniques. The table shows that the MB algorithm achieves the maximum average performance enhancements of 26% and 14.11% when adopting ALTD and APTD respectively compared to the achieved throughput of processing the frames without remapping unlike remapping using STM algorithm which achieves a maximum average enhancement of 4% (and performs worse than no remappin on average). Our results show firstly that, as assumed, migration cost affects the performances, so the mapping cannot be fully changed at once (like considered in the STM approach). Secondly, the ALTD strategy performs generally better than the APTD strategy, showing that a fine-grained monitoring of the traffic on the NoC leads to system-level better mapping decisions.

5.5 Summary

This chapter presented how to exploit DLP and TLP from dataflow applications on multicore processors. The chapter started by reminding the background in multicore processors and dataflow models of computation. Then, the specific problem of mapping dataflow applications on these architectures was presented. The review of existing methods highlighted that: (1) the communication cost between the actors was not considered, (2) static methods are not adapted to dynamic workloads, which are becoming prevalent in leading-edge applications like video codecs.

Locks in mapping dataflow applications on a multicore processor

- take into account communication cost
- take into account dynamic workloads

Our work focused first on means to consider the communication cost by modeling the interconnect and estimating the delay based on the traffic. Then we proposed a greedy heuristic to find at runtime a first mapping solution. To refine the mapping while the application is running, we proposed a runtime remapping algorithm that seamlessly improves the performances. The runtime techniques works thanks to monitoring information.

mapping dataflow applications on a multicore processor

- An analytical model for the interconnect
- Runtime monitoring of the hardware platform
- A runtime remapping algorithm
- A move-based algorithm to limit the migration cost

We evaluated our algorithms through different simulators: Cadence virtual system platform, in-house analytical simulator, in-house SystemC-based cycle accurate simulator for the NoC. We also designed a hardware prototype on the Zynq platform. The SPIDER runtime was directly evaluated on the MPPA® platform.

Contributions in mapping dataflow applications on a multicore processor

We proposed:

- Two algorithms to map dataflow applications on a multicore processor
  - ▷ A greedy algorithm that finds a first mapping in a very short time
  - ▷ A move-based runtime remapping algorithm that relies on the monitoring of the platform to adapt the mapping to the workload
- A runtime manager for PiSDF applications on MPPA
- A demonstrator based on the Zynq platform

The contributions presented in this chapter shows the importance of communications weight between tasks (actors), and their placement and migration during execution. With the emergence of new interconnect technologies, like wireless presented in chapter 7 or optical, these contributions can feed the more general field of interconnect-aware mapping.

6. Exploiting DLP and TLP: scalability and synchronisation issues

Two main issues are identified in multicore processing: data access and synchronisation [222]. This chapter starts by reminding the current issues with scalability and synchronisation in multicore processors. The chapter then presents our scalability study on dataflow applications, followed by our concept of notifying memories, a synchronisation mechanism initiated by the memory. The chapter presents also Subutai, our solution for Pthread synchronisation on NoC-based many-core processors.

6.1 A warm-up on scalability and synchronisation

This chapter starts with a short introduction on one of the most essential part of any modern multicore processor: the memory hierarchy. It completes the section on the memory system initiated in section 5.1.1 on p. 76.

Memory is a vital part of any digital system. It is used to store the programs and the data on which these programs operate. Recall that the speed at which a program runs depends directly on the speed at which instructions and data can be transferred between a processor and memory. The memory must therefore be (very) fast, but ideally also (very) large (to store all the programs and the flow of data in today's data science context), and inexpensive. The impossible meeting between these three requirements has led to proposing solutions based on a memory hierarchy by relying on various memory technologies. Figure 6.2 shows such a hierarchy, usually presented in the form of a pyramid. On top of the pyramid are the registers, that are fully part of the datapath of the processor. At the bottom of the pyramid lies the massive storage, like hard disk/flash drives. In between, the main memory and the cache memory. Each layer of the pyramid is built on different technologies (DRAM for the main memory, SRAM for the cache), which speed and cost per bit increase as moving up the levels, while the capacity decreases. The different technologies also explain why there are different chips. Cache memory, based on SRAM technology which is compatible with CMOS, embedded as close as possible to the processor on the same chip, is a key element of this hierarchy, giving the programmer the illusion of both fast and large memory.

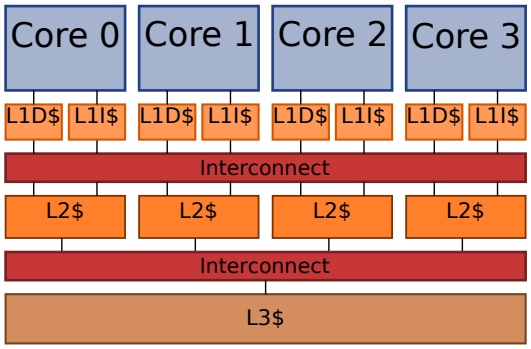


Figure 6.1 – Example of a SMP architecture with four cores, each having a private L1 cache (split in data cache and instruction cache), and a shared L3 cache.

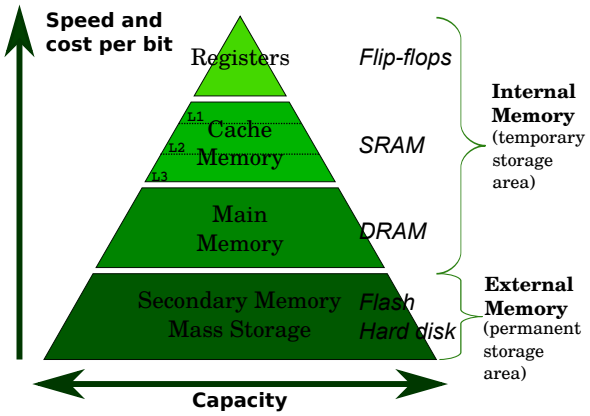


Figure 6.2 – The memory hierarchy, presented as a pyramid

6.1.1 Caches and cache coherence

The cache memory exploits the principle of locality, both spatial and temporal. Spatial locality dictates that if a data is used at a given time, then it is highly likely that the data stored next to it will be used soon. An example is the pixels of an image. Time locality dictates that if a data is used at a given instant, it is very likely to be used again soon. Time locality is, for example, especially adapted for a set of instructions in a loop body.

In the context of an uniprocessor system, the cache increased the pace which a processor could access the data to, and compensated a relatively slow, low-bandwidth, off-chip memory. In the context of multicore processors, the pressure on the memory system is even higher, and the number of cache levels has been increasing as the number of cores also increases. The problem arises that a data in the main memory can have multiple copies simultaneously alive in the different caches. When the value of the data changes, all the copies must be updated. The cache coherence is the mechanism which ensures that all copies (spatially distributed) of a data have the same value. Since all the copies of a data spread out throughout the chip cannot be updated instantly, the cache coherence is tightly linked to the memory consistency model. The memory consistency is the mechanism which ensures that all copies of a data follow the same sequence of values.

Caches, from a hardware point of view, and cache coherence protocols have been intensively studied the last decades. This section is not going to detail the cache coherence protocols, the cache organisation and policies. The feature to highlight is that cache coherent protocols lead to a high number of transactions on the interconnect, along with energy consumption. As the number of cache memories increases, the traffic also grows up to saturating the interconnect, not to talk about the complexity of the protocols. The key message is the ambivalent role of the cache in multicore processor design. It is both the unavoidable component to increase the performances and a major obstacle in the scalability of multicore design and synchronisation of parallel applications.

6.1.2 Scalability

The scalability of a multicore architecture is the property to provide (at least) linearly increasing performances as the number of cores grows. However, as the number of cores grows, the centralized memory access becomes a bottleneck. Connecting the cores on the same bus is not likely to scale over eight cores [154]. Scalability shall be considered at system-level, including the memory system and I/O.

The classical shortcut is to oppose a non-scalable bus-based architecture and a scalable NoC-based architecture. But lots of hybrid organisations lie in between. For instance, the MPPA presented in figure 5.5 on p. 78 shows a cluster-based architecture, where eight cores are interconnected with a bus in

the cluster, and the clusters are interconnected through an NoC. An L1 cache is usually private and as close as possible to the core, but a shared cache (at L2 or L3 level) could be logically shared but physically centralized or distributed.

6.1.3 Synchronisation

Synchronisation mechanisms

Synchronization primitives are means provided to developers to guarantee data integrity when developing parallel applications. The problem of synchronisation dates back long before multicore processors, in the context of concurrent programming, when several processes running on the same computer were competing for shared resources. The *mutual exclusion* problem is certainly one of the most classical problems.

> **Mutual exclusion problem**
> *The problem of making sure that only one thread at a time can execute a particular block of code* [23].

The standard way to solve the mutual exclusion problem is by using a *lock*, a structure that protects the block of code (or a *section*). A *critical section* is the block of code where the mutual exclusion property must be guaranteed. Only one thread can hold the lock. A thread is said to *acquire* the lock when it holds it. The thread *releases* the lock when done. The question to answer then is what to do when the thread cannot acquire the lock? There are only two options: 1) trying continuously until the threads eventually acquires the lock, 2) suspending the thread (meaning freeing the core to do something else) and letting it try again later. The first option is called *spinning*, or *busy-waiting*. The second is called *blocking*. Both options have advantages and drawbacks. Spinning uselessly occupies the processor, but avoids an expensive context-switching. This strategy is interesting if the lock is expected to be released in a short time. Blocking is interesting if the lock delay is expected to be long. The thread frees the processor, but at the cost of a context switching. Both techniques are important, and should be relevantly used according to the context. The mutual exclusion problem involves two threads, and the lock is usually named a *mutex*.

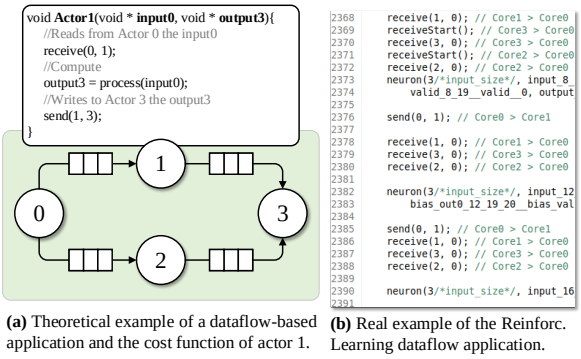
The lock is the most basic synchronisation mechanism. Other mechanisms exist, like the *semaphore* which is the generalisation of the mutex to n different threads. A semaphore has a capacity c and lets up to c threads at the same time in the critical section. Another one is the *barrier* which forces all threads to wait for each others at a given point in the programme. There are also *monitors*, *queues*, *conditions*, *transactional memories*, which are not detailed here, but can be found in [23].

The hardware fundation of any synchronisation mechanism is an atomic memory operation (AMO), which ensures that consecutive read and write operations at a given address are performed in a raw, without any possibility of interruption or other operation in between, as an indivisible step. This operation is defined in the ISA. The ISA is the hardware/software contract: as soon as the hardware implementation respects the atomicity of the memory operation, then the software stack can safely build on. Intel, AMD and SPARC architectures provide the *Compare-And-Swap* (CAS) instruction. Another hardware synchronisation primitive is the pair *load-linked* and *store-conditional* (LL/SC) which is used in several architectures like IBM PowerPC, MIPS, ARM, or Alpha. The RISC-V provides the "A" standard extension for atomic instructions [65], relying on the *Load-Reserved Store-Conditional* (LR/SC) instruction[1].

Synchronisation and caches

The understanding of synchronisation and caches is certainly one of the most complicated feature of a shared-memory multicore processor. It implies a thorough understanding of the microarchitecture of the processor, of the memory system including complex cache coherence protocols, and knowledge in low-level software programming. This section is not going to explain in details this complex mechanism,

[1] which is the same principle as LL/SC



(a) Theoretical example of a dataflow-based application and the cost function of actor 1.

(b) Real example of the Reinforc. Learning dataflow application.

Figure 6.3 – Code snippet of actor synchronisation for an SDF application following a blocking scheme

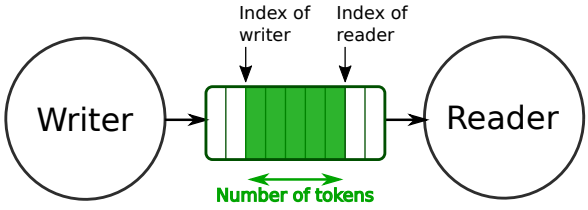


Figure 6.4 – Lock-free synchronisation based on the indexes of the reader and writer for a DPN application

which actually shall be mentionned as plural: mechanisms, since there are several of them, working functionally similarly, but very differently effectively. The interested reader can refer to reference books in the domain to learn more [23, 144, 151]. Basically, on a cache-based architecture, spinning is expected to perform interestingly [95]. Indeed, at the very first tentative to acquire a lock, the processor encounters a cache miss, which eventually loads the content in the cache. The processor then continuously reads again the value in its cache (but without trying to acquire the lock with a write operation), and benefits from the cache coherence protocol to know if the value has changed. When the value has changed, the processor can have a try. Another advantage is to rely (again) on the property of locality. It is likely that the processor tries to acquire again the lock in the near future. Even if evicted from cache L1, it certainly still resides in cache L2.

Synchronisation in dataflow applications

Let's focus now on how to synchronise dataflow applications. From an implementation point of view, a dataflow application follows the well known and overstudied producer/consumer model [23]. A source actor produces tokens that are consumed by a destination actor. The implementation of a dataflow application, e.g. the software synthesis, can simply and safely rely on existing solutions.

This section only presents the two techniques used in the context of our work and does not pretend to be exhaustive over all dataflow implementations.

Semaphores for SDF or PiSDF models

In PREESM, the C code generated from SDF applications relies on semaphores. A *matrix* of semaphores, of the size of the number of cores to cover all possible communications between all the cores, is used to synchronise between a sender and a receiver. The matrix is square, and there is one useless semaphore per line (recall that all actors mapped on one core are wrapped together inside a single thread, with one thread per core, so a given core does not need to synchronise with itself).

Figure 6.3(a) shows the example of a four-actor network, with an insight on the code for actor 1. Actor 1 *consumes* tokens from actor 0 in buffer `input0`, and *produces* tokens for actor 3 in buffer `output3`. Since the buffers `input0` and `output3` are shared between the actors, synchronisation is required. The default implementation in PREESM uses the matrix of semaphores. The function `sem_post` (through *send(senderID,receiverID)*) is used to increment the value of the semaphore when writing in the output buffer is done, and the function `sem_wait` (through *receive(senderID,receiverID)*) to decrement the value of the semaphore. As such, the `sem_wait` function is a *blocking* read, the programme *spin-locks* on the semaphore until it has been incremented by the producer. Figure 6.3(b) shows a snippet of the generated code for the reinforcement learning application (see 6.3.3). It clearly shows the high number of *send* and *receive* functions.

For an SDF application, a blocking synchronisation scheme is justified since there is no chance to fire

the actor until the preceding actor has finished. The number of tokens is known and constant, recall that the scheduling is static and solved at compile-time.

Lock-free synchronisation for DPN model

In the case of a DPN model of computation, where reading is non-blocking, and the number of tokens needed to fire an action is not resolved at compile-time, a blocking synchronisation scheme is not adapted. The following explanation takes as example the implementation proposed in ORCC. Figure 6.4 highlights the FIFO shared between a writer and a reader. The management of the tokens is done through indexes. At the beginning, both indexes are at the starting address (`address 0`). The index of the reader shall never be ahead of the index of the writer. The index of the writer shall never catch up the index of the reader (the FIFOs are circular buffers). Both writer and reader need to read the indexes of each other. But it is important to recall explicitly that only the writer changes the index of the writer, and only the reader changes the index of the reader, and that the indexes can never go back. This means that, in the case that the actors are mapped on two different cores, each having in own cache, it may happen that the value of the index is not the *real* one (we need to wait for the cache coherence protocol to proceed), which is not an issue as the value will be updated soon. It can be called a *conservative* synchronisation scheme since reading the not-updated value will not do any harm to the system.

The number of tokens in the FIFO is simply known by computing the difference between the index of the writer and the index of the reader. Since the FIFOs are circular buffers, some modulo operations are sometimes needed to compute this difference. However, as the modulo operations are costly, some efficient software synthesis techniques to avoid them are proposed and detailed in [147, 177].

Synchronisation is thus non blocking. If the number of tokens available does not fire any action, the actor continuously polls on the indexes waiting for an updated value. When several actors are mapped on the same core (which happens usually), a local round-robin scheduler simply gives a chance to the next actor. We can note here that this approach leads to lots of, sometimes useless, memory accesses. This analysis motivated our work and is discussed more thoroughly in section 6.3.2.

6.1.4 Questions and challenges taken up

In light of the scalability and synchronisation issues studied in a general view, we contributed in a better understanding in the context of dataflow applications. Beyond the hardware complexity of designing scalable multicore processors, we wondered how dataflow applications, naturally presenting explicit parallelism, scale with the number of cores.

? How does a dataflow application scales with the increasing number of cores?

Does the execution time of a dataflow application reduces linearly as the number of cores increases?

The synchronisation mechanisms are studied from a theoretical point of view, and from a practical point of view, both in the general case: there is no possibility from the application to know when a specific event is about to happen. In the context of dataflow applications, where the sequence of events is explicitly available in the network of actors, we wondered if there would be yet another way to synchronise the actors more efficiently.

Synchronizing dataflow applications on a multicore processor

Can we think of a more efficient hardware approach to synchronise dataflow applications on a multicore processor?

The two next sections explain our contribution in the scalability study of dataflow applications, and a hardware based synchronisation mechanism. Section 6.4 presents a hardware/software proposal for Pthread applications.

6.2 On scalability of dataflow applications

Given the explicit parallelism available in dataflow applications, by actors spatially and temporally placed in parallel, it seems obvious that a dataflow application should scale well with the increasing number of cores, bigger caches, higher number of levels of cache, and cache shared between several cores. But, this is without considering communication time, memory and synchronisation issues. As mentionned in [178]: *Bigger caches are better for performance but show diminishing returns as caches sizes grow. Large caches may also not be of use to applications where data is only used once, such as video decoding*. This feature is the very nature of the dataflow model of computation: the data is used only once.

If these trends are qualitatively known, there was surprisingly no study that quantitatively evaluates the behaviour of dataflow applications as the cache memory grows, and number of cores increases.

We have thus carried out a set of experiments based on static dataflow applications and SMP architectures. The results show that bigger is not always better, and the foreseen future of more cores and bigger caches do not guarantee software-free better performance for dataflow applications.

We used PREESM framework for dataflow applications, and we used the tool Sniper to simulate the SMP architectures, including leading-edge hardware configurations available commercially, e.g. a 32-cores with 256 MiB L3 cache. The Sniper multi-core simulator [160] includes the description of the Nehalem cores as well as cache, memory controller, and DRAM.

Sniper is a system simulator for multi-core architectures, used to evaluate application's performance including power and energy consumption [48, 90]. Sniper adopts an interval-based core model simulation, which allows fast and accurate simulation. The Nehalem cores are by default provided within Sniper distribution. Sniper core model and cache hierarchy are validated against actual Xeon processor using Splash2 benchmarks. Sniper takes as input configuration files that allow the user to set parameters as cache sizes, cache sharing, number of cores, core frequency, among many others. The following sections extracts some results from our study. The full paper is available online [10].

6.2.1 Higher number of cores

Figure 6.5 shows the application iteration time (time for the application to complete the execution of one loop), for Stabilization (a), Stereo (b), and SIFT (c). The x-axis groups the results for one configuration, and each bar is the number of cores. For instance, the first configuration is a private L2 (x1) with no L3 cache. The last configuration is a shared L2 between two cores (x2), and a shared L3 between four cores (x4).

The results show that only Stabilization scales reasonably, reducing its execution time on average by 46% from 4 to 8 cores, 43% from 8 to 16 cores, and 39% from 16 to 32 cores. However, the same does not occur to Stereo and SIFT, which have a moderate or even worst improvement, with Stereo presenting an execution time of -22%, -1.3%, +2.6%, for an increase in the number of cores of 4 to 8, 8 to 16, and 16 to 32, respectively.

6.2.2 Bigger L2 cache

Figure 6.6 shows the results for an L2 cache increasing in size (256KB, 512KB, and 1MB) on the x-axis. The left y-axis represents the application iteration time, and the right y-axis represents the cache miss rate. Each plot represents one application, with each one having 3 sets of results representing different L3 sizes.

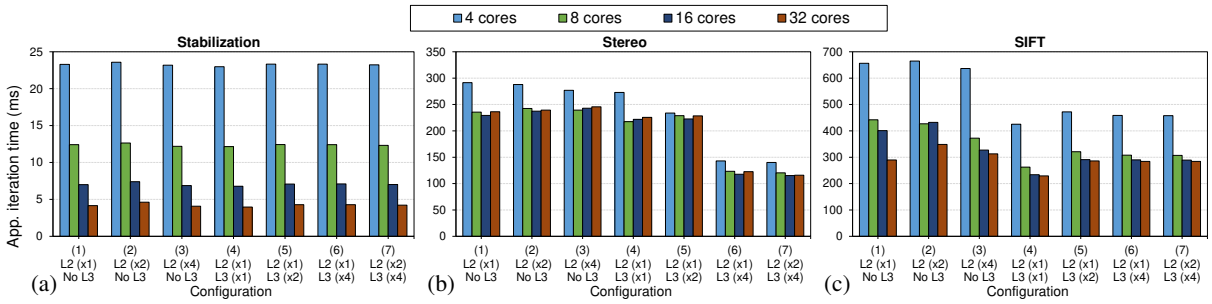


Figure 6.5 – Application iteration time over different number of cores for three applications: (a) Stabilization, (b) Stereo, (c) SIFT.

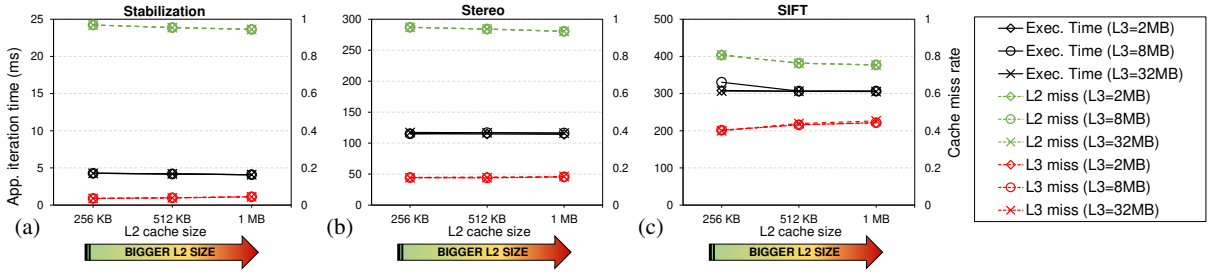


Figure 6.6 – Results for an increasing size of L2 cache (a) Stabilization, (b) Stereo, (c) SIFT.

The results show that increasing the L2 and L3 size has a low influence on the L2 and L3 miss rate for all applications. The execution time has a small reduction according to higher L2 sizes. However, this value is insignificant, representing an average reduction from the lower L2 size (256KB) to the higher L2 size (1MB), of -0,49% for Stereo, -1.76% for SIFT, and -4.62% for Stabilization.

6.2.3 Bigger L3 cache

The results for bigger L3 cache follow the same trend observed for L2. Figure 6.7 shows an example with the L2 size fixed in 512KB (bigger L2 sizes present very similar behaviour). It is possible to see that both L2 and L3 cache misses remain stable, and with an insignificant reduction in the execution time (not better than -0.26% for all applications).

6.2.4 Summary and findings on the cache study

The study on the cache sizes and configurations for dataflow applications confirms empirically what was commonly accepted: dataflow applications do not benefit effortlessly from higher number of cores and bigger caches. One interesting finding is that private L2 and L3 shared by all cores was the configuration

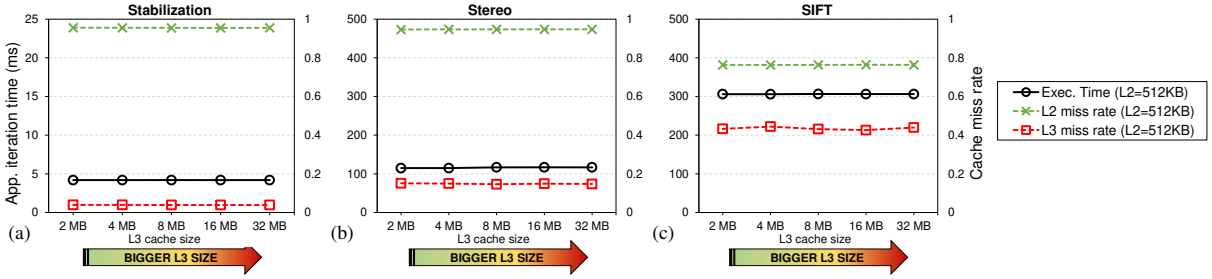


Figure 6.7 – Results for an increasing size of L3 cache with a private L2 of 512KB. (a) Stabilization, (b) Stereo, (c) SIFT.

that presented the best results related to application speed-up and L2/L3 miss rate. While this conclusion can sound similar as Intel had reached some years ago, justifying its current cache organization with L3 shared by all cores, it was not so obvious from our point of view. First, our focus was to evaluate the impact specifically for dataflow applications, research that was not sufficiently addressed, with not enough data supporting the claim. Secondly, our initial hypothesis was that when two actors – sharing the same FIFO – are mapped on two cores sharing the same L2 cache, the performance would improve due to the reduction in the coherence traffic and the L2 miss rate reduction. This behaviour is supported by our results (figures not shown in this document). However, this leads to a higher miss rate for L3, which has higher penalties than L2, and consequently, has a higher influence on the execution time.

As an answer to the bad cache utilisation for dataflow applications, we proposed to use two dynamic memory management strategies: Copy-on-Write (CoW) and Non-Temporal Memory (NTM) copying.

They are not novel in their principle, CoW is a well-known approach supported by Linux OS by the mmap() syscall [191], and NTM is essentially a direct RAM-to-RAM copy, supported in some Intel processors [42]. The novelty here is to exploit opportunities of using such techniques in the dataflow context, and quantify their improvements in the applications execution time and system energy by saving memory transfers.

The results, not shown here but available online in [10], show that the NTM technique can contribute modestly in the improvement of the execution time. The CoW technique presents interesting results, especially in the case of transfers bigger than the 400 KiB, achieving a reduction of 15% in the execution time and 20% in energy consumption.

6.3 The notifying memories concept

Though identified already 30 years ago [216, 219], the memory is one of the most important issue in today's multicore processor designs to build faster systems. We thus studied new architectural concepts, especially to improve synchronisation and communication between several cores, by reducing the data movements in the cache hierarchy.

Our concept is called "Notifying memories" because it provides memories with *notification* capabilities, meaning that they can initiate a transaction with the processor, instead of simply replying to requests.

This concept is particularly adapted to dataflow applications and emerged during our work in COMPA ANR project. Mostafa Rizk, during his post-doc, implemented the proof-of-concept. Then, during Nooman ANR project, the concept was further studied during the PhD of Alemeh Ghasemi.

6.3.1 The observer design pattern

Our idea is inspired from the observer design pattern, widely used in software engineering. A brief description of this software design pattern is now given.

The observer design pattern

The observer design pattern *defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically* [217]. That behaviour is exactly what we expect when the content of a FIFO changes: notify the concerned processors that execute the actors connected to that FIFO. Figure 6.8 shows the UML class

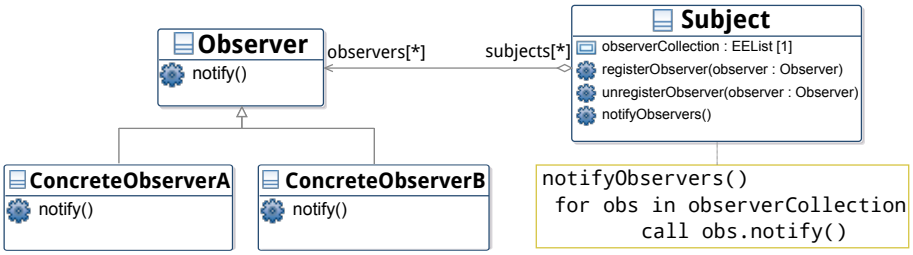


Figure 6.8 – UML Diagram of Observer Design Pattern for software implementation

diagram of the observer design pattern. The subject is the element to watch. The observer is the element that should react whenever a change occurs from the subject. The subject notifies a change to their observers by means of a method.

Porting the observer design pattern to our context, the memory is the subject and the processor is the observer. Implementing such a pattern means breaking the conventional way of connecting a processor to the memory. In a Von Neumann architecture, the memory replies only to the requests initiated by the processor. For instance, when designing a bus-based multicore processor, the processors are connected as *master* on the bus, meaning that they are allowed to initiate a transaction, whereas the memories (memory controller or scratch pad memories) are connected as *slave* component on the bus, meaning that they are allowed to reply to a request, but not to initiate a transaction. The notifying memories concept implies that the memories can initiate a transaction, so connected as master on the interconnect. In a NoC-based architecture, the network interface (NI) is the component between the routers and the processor or the memory. An NI naturally has the possibility to initiate a transaction. We thus first implemented the concept for an NoC-based architecture, so that the NI component changes only, and we can keep unchanged the other components like the processor or memories. Everything happens in the NI. The next section explains our proof-of-concept for a DPN dataflow model.

6.3.2 A case-study on Data-Flow Applications with NoC Interfaces Implementation

Motivational example

Recall the DPN the actor model presented on Figure 5.7 p. 79. An actor can contain several *actions*. An action is executed (*fired*) when a set of conditions is satisfied. This so-called *firing rule* consists of checking that the number of tokens in the input FIFO is greater than the number needed to compute, and that the output FIFO is empty enough to store the produced tokens.

The main idea of notifying memories is to give the capabilities to the FIFO to notify directly the processor about its content.

We use as a case study an MPEG-4 Simple Profile decoder (MPEG4-SP) specified in RVC-CAL [5]. This decoder is specified with heterogeneous MoC and contains up do 40% of dynamic actors [157]. The ORCC tool is used for compiling and software synthesis [5]. Our work relies on the C-backend that generates C code for multi-core platforms. Since the number of actors (41) is greater than the number of processing cores, several actors are mapped on the same core, and an *actor scheduler* is required. Different policies have been proposed, one of the most efficient one remains the round-robin (the default in ORCC). They all have the same major drawback related to dataflow principles, which is the inefficient polling that leads to useless accesses to the memory when a scheduling attempt is not successful.

Figure 6.9 presents the structure of the software FIFO generated by ORCC [5]. It is composed of five parts: i) size of the FIFO; ii) FIFO content, where memory allocation is done according to the FIFO size

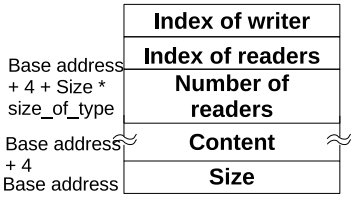


Figure 6.9 – Structure of a software FIFO generated by ORCC

```
if (numTokens_FIFO_IN1 >= 64
 && numTokens_FIFO_IN2 >= 1){
   if(SIZE_FIFO_OUT -
    numTokensFIFO_OUT > 64){
        // fire action 1
   }
}
```

Listing 6.1 – Example of a firing rule

and size of data type; iii) number of readers, since one actor can write in a FIFO but there might be several readers; iv) index of readers, each reader has its own index; and v) index of writer. Synchronisation is handled through indexes. The difference between the reader index and writer index determines the number of tokens inside the FIFO. When multiple readers occur, the minimum index is used. It might happen that one slow reader blocks the other readers. Many papers deal with FIFO sizing and FIFO handling but it is orthogonal to this work. Given this FIFO structure, the processor that executes an actor has to read the values of the different indexes in order to determine the number of tokens in the FIFO. This leads to a set of memory requests for each FIFO. Taking the example of the firing rule given in Listing 6.1, to compute `numTokens_FIFO_IN1`, the number of tokens in the first FIFO, the processor emits two requests, one for the index of the writer, and one for the index of the reader (one for each reader actually). For the second FIFO, the processor emits another set of requests. Then, in order to check for the output FIFO, other requests are emitted on the NoC. In C language, if the first condition is not satisfied, the whole test is stopped. The worst case occurs when the input conditions are satisfied but not the output condition, which would lead to six transactions for no action firing. Of course, next scheduling attempt will test again these conditions although the conditions on the input FIFO are satisfied. It has to be noticed that true conditions cannot become false on the next trial. Our contribution also relies on this property.

We have carried out some experiments using the C backend of ORCC and executed the MPEG4-SP decoder on a desktop computer. We have traced the number of firings of each actor during one scheduling attempt. We have counted the number of zero firings, i.e. no action could be fired, out of the total number of scheduling. Table 6.1 shows the percentage of unsuccessful scheduling for different video sequences from [1]. There are two reasons why no action can be fired: 1) one of the input FIFO is *empty* (i.e. does not contain enough tokens); or 2) one of the output FIFO is *full* (i.e. does not contain enough space). Table 6.1 also shows the distribution between empty and full FIFO. These results show that at least 20% of scheduling attempts are unsuccessful. Although the lack of tokens in the input FIFO seams to be the major reason, the disparity among the different video sequences prevents from drawing any clear conclusion.

This observation motivates the integration of mechanisms able to monitor the FIFO status and to emit notifications.

This mechanism can be integrated in the NI, close to the FIFO implementation.

The motivation is to find a solution to delete useless memory requests, independently of the processors, memories, NoC parameters and scheduling policy[2]. The main issue is to stop the polling on the NoC that: 1) is useless when no action can be fired; and 2) consumes bandwidth that would be useful for effective transactions. The current situation is that memories are subjected to processor requests. The idea is to give new capabilities to memories, so that, they can inform the concerned processors that their (FIFO) content has changed.

[2] although combining our approach with a new scheduling policy is relevant

Table 6.1 – Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video Sequence	Format	Useless attempt	Empty input FIFO	Full output FIFO
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

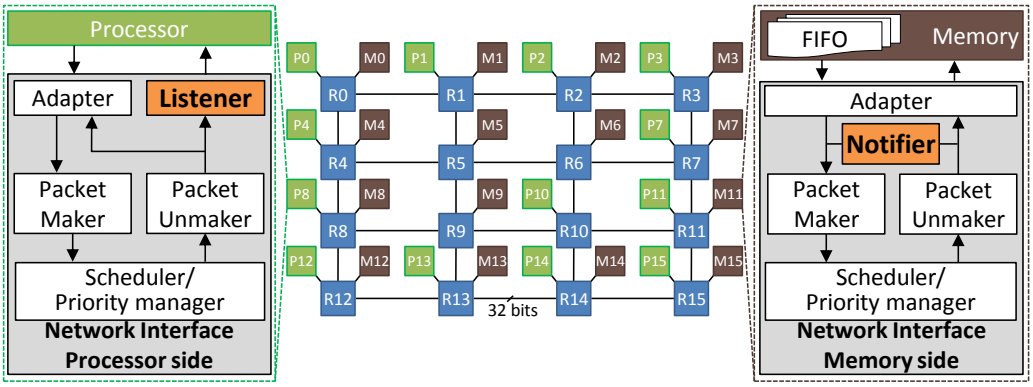


Figure 6.10 – The structure of the used NoC implementing the notifying memories concept

NoC Implementation

The implementation in the NoC was performed by Mostafa Rizk, during his post-doc [103]. There are two elements to be added in the NoC platform: the notifier and the listener. In order to remain compliant with any existing processor or memory and to be independent from NoC parameters such as topology, router buffer depth and routing policy, we have decided to integrate these elements into the NI of the NoC. Besides, we need a master component that can send packets through the network and a component that can monitor requests, the NI is the ideal component to offer such features.

Figure 6.10 illustrates the structure of the NoC used to demonstrate the notifying memory concept. The NoC is a 4×4 mesh-based network which interconnects 28 IP cores (13 processing and 15 memory nodes). It is based on a wormhole packet switching mode, deterministic XY routing algorithm to avoid deadlocks, and flow control policy without virtual channels. The routers have one arbiter per port and one buffer per input port. Our approach is actually generic and can be applied to NoC with different features. For instance, N-flit buffers can be used to improve performance at the cost of more memory, in that case all transactions including notifications will take advantage of it. The back-end part of the NI includes a packet maker and packet un-maker to assemble and disassemble the packets, a scheduler/priority manager to synchronize packet transmission and reception. The modifications lie in the front-end of the network interface by either implementing the notifier or the listener. The notifier is implemented in the NI of a memory. The listener is added in the NI of a processor. The structures of the additional components are detailed in the following subsections.

Notifier

The notifier is a hardware module that transmits the status of all FIFOs allocated in a node hosting memory. For each FIFO, the notifier generates a notification signal that is passed to the FIFO's writer whenever it contains enough space to save new tokens, or to the FIFO's readers whenever enough tokens are available. The notifier functionality can be divided into three phases: the configuration phase, the checking phase, and the notification phase.

1. *Configuration phase:* it corresponds to the `registerObserver` method in the design pattern. The memory is configured with the processors to notify.
2. *Checking phase:* it corresponds to a "monitoring" phase, where each new write operation to the memory is scanned, and each update on an index is checked if it triggers a firing rule.
3. *Notification phase:* when a firing rule becomes valid, a notification is sent through a packet to the concerned processor.

Listener

The listener stores the information included in the notification packets sent by notifiers. The coupled listener to each processor specifies the validation statuses of all firing rules related to all actions of all actors mapped to the processor. The processor accesses the validation status of all firing rules corresponding to an action before it is fired. In addition to the status, the listener stores the value of the available tokens at the input FIFO and the free space at the output FIFO for each input and output firing rules respectively. The information is thus available locally to the processor and no memory requests through the NoC are needed. The listener functionality is divided into two phases: the configuration phase and the execution phase.

1. *Configuration phase:* a table storing the firing rules is initialized.
2. *Execution phase:* the notification packets are caught in the NI on processor side, and the information about the firing rule is stored in a table. The processor reads in this table instead of polling on the cache.

Results

Experimental Setup

In order to check the relevancy of the proposed approach, the adopted NoC implementing notifying memories has been described in SystemC TLM model as a proof of concept. The devised model executes an MPEG4-SP decoder with 41 actors and 70 FIFOs specified in RVC-CAL. The number of FIFOs are approximately equally distributed on all nodes accommodating memories. The actors are then mapped manually such that the number of hops is minimized between one actor and its FIFOs. The SystemC model is cycle accurate at the NoC level and network interface; whereas actions are functionally simulated. We strive to accurately reproduce the timing features of actions executions in addition to their functionalities. The mean values of execution time of all actions are imported from profiling data on a desktop computer. Multiple simulations have been conducted to decode ten frames for several video sequences from [1]. To evaluate the efficiency of the notifying memories concept, the obtained results are compared with the ones of ordinary memories. Both models use identical NoC features (e.g. 500 MHz frequency, switching mode, routing algorithm), processing elements features, and mapping strategy.

Results and Comparison

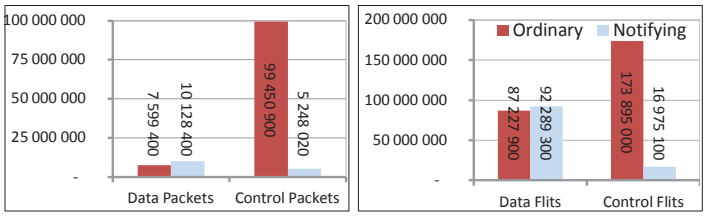
Table 6.2 shows the results obtained after decoding 10 frames of ice video sequence in 4CIF format in terms of throughput, latency[3], injection rate, switch conflicts, total number of transported flits, and power consumption. The comparison between the two cases, ordinary and notifying, shows significant reductions in terms of latency, injection rate, switch conflicts, and number of transported flits and packets. Also the throughput is improved such that it is compatible with the 25 frames per second standard without using additional hardware accelerators or processing speedups.

The analysis of the results reveals an additional traffic overhead in case of ordinary memories which increases significantly the injection rate and switch conflicts. To track its cause, packets are classified according to their functionality into two categories: data and control packets. Packets holding tokens or requests for reading tokens are data packets, while control packets category includes all other packets that are used to transport mapping information, set FIFO indexes, request or retrieve FIFO indexes, and notification signals produced by notifiers. This classification is also applied to flits. Figure 6.11

[3] the time between the first token consumed and the first token produced by the application

Table 6.2 – Results after decoding 10 frames of ice video sequence in 4CIF format

Parameter	Notifying memory	Ordinary memory	gain
Latency (μs)	143.42	665.06	-78.44%
Throughput (frames/s)	27.53	23.29	+15.41%
Injection rate(flits/s)	60 167 732	121 635 294	-50.53%
Switch conflicts	71 182 509	288 574 519	-75.33%
Transported flits	109 264 000	261 123 000	-58.16%
Transported packets	15 376 400	107 050 000	-85.64%



(a) (b)

Figure 6.11 – Classification of packets and flits transported after 10 decoded frames of ice-4CIF

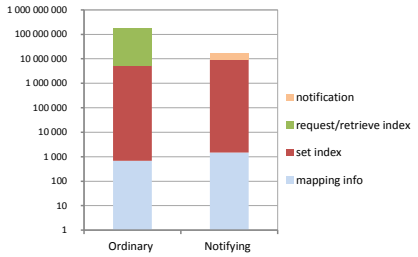


Figure 6.12 – Classification of control flits according to their types after 10 decoded frames of ice-4CIF

presents a comparison between notifying and ordinary memories in terms of control and data categories for transported packets and flits for the ice-4CIF video.

The figure shows that ordinary memories induce 19 times more control packets and 10 times more control flits than notifying memories.

The number of data packets and flits is approximately the same in the two cases. The simulation is stopped after 10 fully decoded frames while the network contains partially decoded frames. This explains the traffic overhead, due to the remaining control packets injected to the NoC. Investigating thoroughly the types of transported control flits shows on one hand that more flits are needed for mapping information when notifying memories is adopted since the manager has to send additional information for all notifiers. Also, additional notification flits are transported from notifiers to the listeners. On the other hand, the adoption of notifying memories eliminates the use of flits to request and retrieve FIFO indexes. Figure 6.12 shows in logarithmic scale the number of each type of control flits transported while decoding 10 frames of ice video sequence in 4CIF format for the case of notification memories and ordinary memories. It shows that the added flits for notification and extra mapping in the case of notification memories are negligible (4.58%) compared to the required flits to request and retrieve FIFOs indexes in ordinary memories.

The comparisons for other video sequences are summarized in Table 6.3. These average results confirm the efficiency of the notifying memories concept which leads to great reductions reaching 78% for latency, 60% for injection rate, 67% for transported flits, and 85% for switch conflicts. Also the throughput enhancement is improved by up to 16%.

Preliminary synthesis results

We have implemented a *worst-case* design, where a single notifier, which is implemented in all memory NIs, and a single listener, which is implemented in all processor NIs, can manage all firing rules of all actors. The number of firing rules of in the MPEG4-SP application is 145. Hence, 145 registers are required in all banks of the 12 listeners (one per processor) and 15 notifiers (one per memory). The area and power results are obtained with the Cadence Encounter RTL Compiler RC12.24 tool. The synthesis

Table 6.3 – Notification memory gain for decoding 10 frames of different video sequences

Video Sequence	Format	Throughput	Latency	Injection rate	Switch conflicts	Flits number
Bridgefar	QCIF	+15.53%	-73,96%	-45,80%	-71,38%	-54,22%
bus	CIF	+2.84%	-73,79%	-53,40%	-72,90%	-54,73%
grandma	QCIF	+16.79%	-68,96%	-60,78%	-85,50%	-67,36%
foreman	CIF	+14.26%	-78,41%	-46,81%	-72,86%	-54,39%
ice	4CIF	+15.41%	-78,44%	-50,53%	-75,33%	-58,16%

targets the 65nm process technology at 500 MHz operating frequency and 25°C. Total power was obtained by using the leakage and dynamic power of the NoC components relying on the switching activity traced by the SystemC simulation of 10 decoded frames of the ice-4CIF video sequence.

The results show that the NoC adopting notifying memories saves up to 49.1% of power consumption compared to the reference NoC.

Besides, the power overhead of the interfaces of the proposed NoC presents a modest value of 16.3%. Regarding the area, the proposed NoC presents an overhead of 12.4%, when compared to reference NoC.

The case study focusing on the NoC called for further investigations. First, the simulation was accurate at the NoC level only. A system-level simulation is needed to know the impact on the full system. Particularly, the processors, along with their caches, were not accurately simulated. The results are obtained for a single application, and other dynamic applications may behave differently. The target considered assumed distributed memory modules. More experiments are required on other types of application and on other types of multicore processor architectures.

6.3.3 NM4SMP: Notifying Memories for Symmetric shared-Memory Processors

Given the identified limitations of the proof-of-concept of notifying memories, we wanted to perform the study on centralized shared-memory architecture, also referred to as "Symmetric shared-memory processors" (SMP). This work has been done by Alemeh Ghasemi during her PhD thesis [9]. The implementation led to a proposal called NM4SMP, Notifying Memories for Symmetric shared-Memory Processors.

Implementation

There are two main differences in the implementation between NM4SMP and the proof-of-concept. First, the NM4SMP module gathers both the Notifier and Listener near the L1 cache. Secondly, the module was designed for lock-based synchronisation technique as an alternative to semaphores, in the context of SDF applications.

Figure 6.13 shows an example of the implementation of notifying memories in a 4-core SMP. Figure 6.14 shows a closer look to the integration of the notifier and listener.

The NM4SMP module monitors the activity between the core and the L1 data cache, and preempts the memory accesses for specific addresses.

The addresses to look at are stored in the BAR (Base Address Register). There are then two cases:

1. a write means a change in the value which should set a firing rule to true, and will trigger a notification,
2. a read means a check for a firing rule and the listener holds the correct information

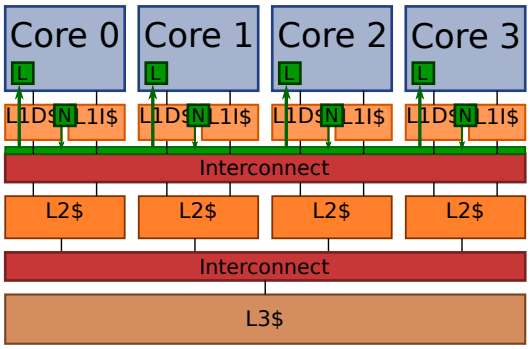


Figure 6.13 – Example of the implementation of notifying memories in a 4 core SMP. N: Notifier, L: Listener

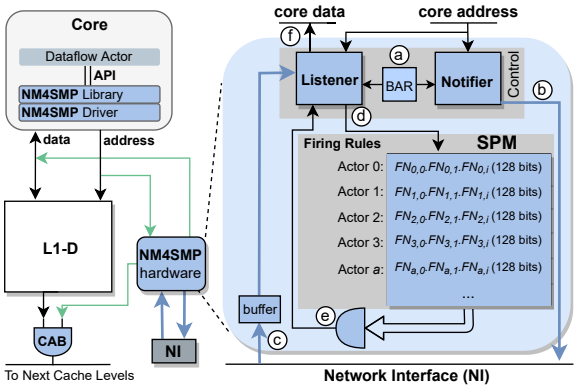


Figure 6.14 – Overview of the NM4SMP hardware module tightly coupled to the L1 cache. BAR: Base Address Register, SPM: Scratch Pad Memory, CAB: Cache Access Blocker

The NM4SMP module has an access to the interconnect in order to send notifications to other NM4SMP modules, and to catch the notifications coming from the other modules. A Scratch Pad Memory (SPM) stores the firing rules for each actor. The considered implementation is provided for 255 actors, each having the possibility to have up to 128 firing rules, leading to a 2KiB SPM. Though oversized compared to the applications we tested, the preliminary hardware estimations for area and energy show insignificant overhead.

The execution model follows the same steps as presented in section 6.3.2. For the notifier, there are three phases: 1) the configuration phase, 2) the checking phase (monitoring), 3) the notification phase. For the listener, there are two phases: 1) the configuration phase, 2) the execution phase.

Methodology

Figure 6.15 shows the methodology used for the experiments on NM4SMP. We used PREESM dataflow framework and a set of applications available in the repository. We support two models of computation, the SDF model and the PiSDF model. This section focuses on the SDF case. The details on the PiSDF case are available in the thesis [9]. We take the C code generated by PREESM and adapt it to use the NM4SMP library. The application is then compiled with a classical compilation toolchain. The binary code is simulated with Sniper, which provides the performance results.

We evaluate four dataflow applications implemented as SDF graph:

- Reinforcement Learning application - Training phase (RLT)
- Reinforcement Learning application - Prediction phase (RLP)
- *Stabilization*, a filter to compensate the movements of a video recorded with a shaky camera
- *Stereo*, a computer vision application that processes a pair of images to produce a disparity map corresponding to the depth of the captured scene

Results

We focus here on only three sets of results: the execution time, the impact on the cache and the scalability.

Execution time

First, we show the results on the execution time. Table 6.4 shows the performance of the four applications considered on a Xeon SMP with 16 cores. The results show interesting speed-up for the RLT application. However, there is no improvement for stabilization of stereo applications, meaning that these applications are not bound by synchronisation.

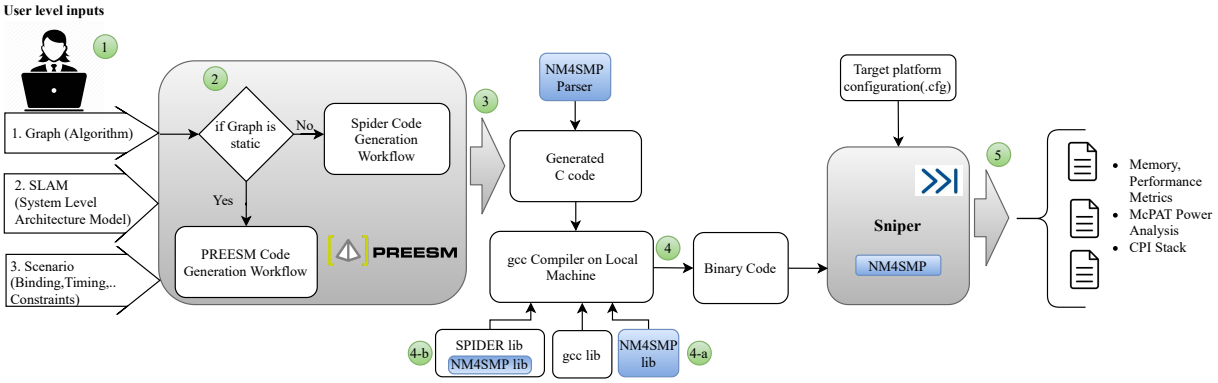


Figure 6.15 – Methodology followed to experiments NM4SMP

Table 6.4 – Performance and synchronization metrics for a 16-cores SMP

Applications	RLT	RLP	Stabilization	Stereo
		Baseline		
Execution time (ns)	3.15E+08	4.07E+08	3.49E+08	1.72E+09
Synchronization time (ns)	1.62E+08	8.02E+07	4.35E+07	2.15E+08
		NM4SMP		
Execution time (ns)	1.60E+08	3.26E+08	3.45E+08	1.72E+09
Synchronization time (ns)	2.52E+06	2.44E+06	3.35E+07	2.15E+08
Average synchronisation speedup	**64.48**×	**32.93**×	**1.3**×	**1**×
Application speedup	**1.97**×	**1.25**×	**1.03**×	**1**×
Executed instructions saved	**17.85%**	**4.51%**	**0.02%**	**0.00%**

Impact on the cache

Second, we show the impact on the cache. Figure 6.16 shows the reduction in the number of cache accesses and the miss rates obtained for the Xeon and the Atom architectures. The results on the cache help in understanding the results on the execution time. Figure 6.16 clearly shows that in the case of RLT application, we can reduce around 20% of the number of L1 accesses, reducing further the miss rate, which explains the speed-up obtained on the execution time. For stabilization and stereo, the impact is insignificant, meaning that the cache is not polluted by the semaphores for these applications.

Scalability

Third, we show how the applications scale with or without NM4SMP. Figure 6.17 shows the results on the execution time for a varying number of cores from 2 to 16. It is clear from the figures that stabilisation scales pretty well with the number of cores, this is why there is no room for improvement from the synchronisation point of view. Stereo shows an unexpected behaviour, with better execution time for 8 cores than 16 cores. The curves for RLT and RLP clearly show that these applications do not scale well and that our NM4SMP solution helps in making these applications scale.

6.3.4 Notifying memories: *the juice*

We introduced a new concept called “Notifying Memories” as an alternative to synchronisation mechanism for dataflow applications. We implemented it for two different cases: 1) lock-free synchronisation in a NoC-based architecture adapted to a dynamic model of computation (DPN), 2) lock-based synchronisation in a bus-based architecture adapted to a static model of computation (SDF). The experiments show interesting results for the dynamic case. The results on the static case show that this solution helps in making scalable a non scaling application. The concept is thus better adapted to dynamic models, and further studies should go into that direction. The concept can also be combined with emerging approaches like *near-memory*

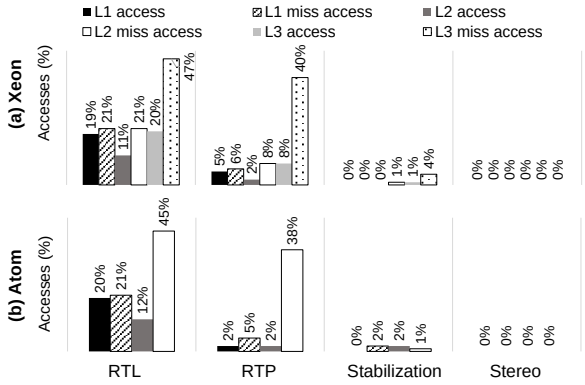


Figure 6.16 – Cache access reduction using NM4SMP for Xeon (a) and Atom (b) processors.

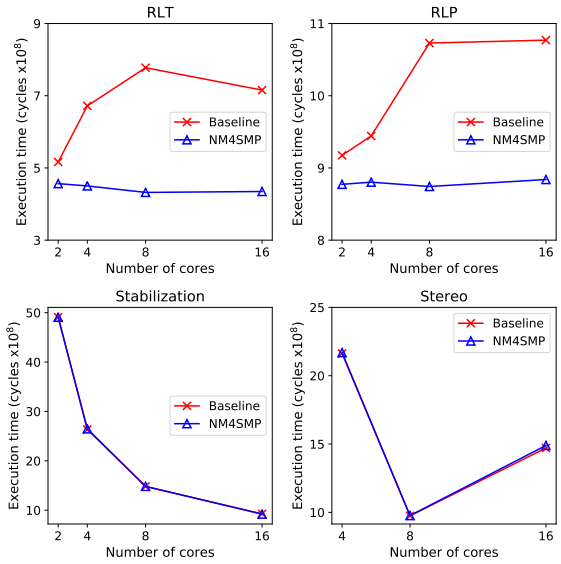


Figure 6.17 – Scalability of the static applications

processing or *computing in network* presented as perspectives in chapter 7.

6.4 Subutai: synchronisation primitives spread throughout the NoC

The last section of this chapter deals with imperative parallel programming models, like Pthread with C language, and synchronisation primitives. Recall that synchronization primitives are means provided to developers to guarantee data integrity when developing parallel applications. While multiple novel solutions have been proposed to speed up parallel applications through handling one type of data synchronization primitive, exceptionally few works support multiple types of synchronization primitives and legacy code. During his PhD thesis, Rodrigo Cataldo proposed Subutai [52], in a joint collaboration with PUCRS, Porto Alegre, Brazil.

Subutai is a hardware/software co-design solution for accelerating multiple synchronization primitives without modifying the application source code.

By providing a new user library, while retaining an existing synchronization API, legacy and novel applications can benefit from our solution. Our experimental evaluation, which provides a POSIX Threads implementation, demonstrates Subutai speeds up to 2.71× and 4.61× the execution of single- and multiple-application executions, respectively. The paper "Subutai: Speeding Up Legacy Parallel Applications Through Data Synchronization" is added as appendix C and is summarized here.

The main proposal is a novel synchronization solution that accelerates parallel applications without modifying the application source code. The proposed hardware/software solution, called Subutai, tackles the synchronization problem within a low-level Network-on-Chip (NoC) Interface (NI).

Fig. 6.18 depicts the Subutai solution with a general-purpose computing stack, highlighting the components required for its operation. The figure shows the four major actions needed to implement Subutai: 1) a new implementation of the Pthread library respecting the API, 2) a device driver, 3) a new scheduling policy, 4) a new hardware inside the NI of the NoC.

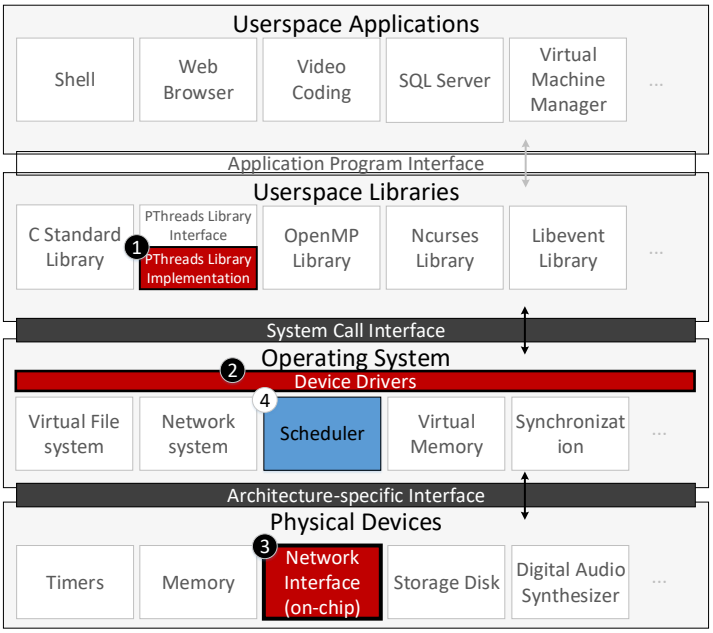


Figure 6.18 – Subutai components are highlighted in red (1, 2, 3) in the computing stack. Subutai only requires changes in the (1) PThreads implementation, (2) OS NI driver, and (3) on-chip NI. Additionally, (4) a new scheduling policy (in blue) is explored in this work as an optional optimization.

6.4.1 Subutai overview

Software-wise (Subutai-SW), the POSIX Threads (PThreads) was implemented according to the IEEE Std 1003.1 standard. Thus, any application employing the PThreads API (i.e., pthread.h) is compatible with Subutai. The PThreads compatibility restricts a multitude of optimizations since we cannot inject the source code with extra synchronization metadata or change the application communication model. In addition to interfacing with the application, our software must work with new functionalities on the hardware-side; hence, we provide an Operating System (OS) driver responsible for the latter activity.

Hardware-wise (Subutai-HW), we extended an existing on-chip NI to support, in a distributed way, the following synchronization primitives: mutex, barrier, and condition. NI handles new types of packets and requires access to a small (less or equal to 1KiB) memory to record synchronization events and metadata. Figure 6.19 presents an internal view of the architecture considered and the added elements inside the NI of the NoC.

The left-hand side of Fig. 6.19 shows that Subutai-HW employs double-linked queues to record events. As an alternative to statically allocating for the worst case, the double-linked queues allow Subutai-HW to employ a dynamic allocator for reducing memory consumption to the minimum, at the cost of additional pointer arithmetic logic. Besides, condition variables are dealt more efficiently with such structure. The queue manipulation is based on the futex implementation of the Linux kernel.

6.4.2 Experiments

Experimental setup

The performance of Subutai is evaluated through the widely used PARSEC benchmark, as it provides a wide range of application domains, parallelization models and data sharing [182]. From the application set, we employ Bodytrack, Streamcluster, and x264. We employ the Gem5 simulator [159] to produce synchronization points of the applications; next, we feed this information into an in-house SystemC simulator, which enables us to collect experimental results. We run applications with and without Subutai: the former will henceforth be called Subutai, and the latter SW-only (i.e., Linux Kernel).

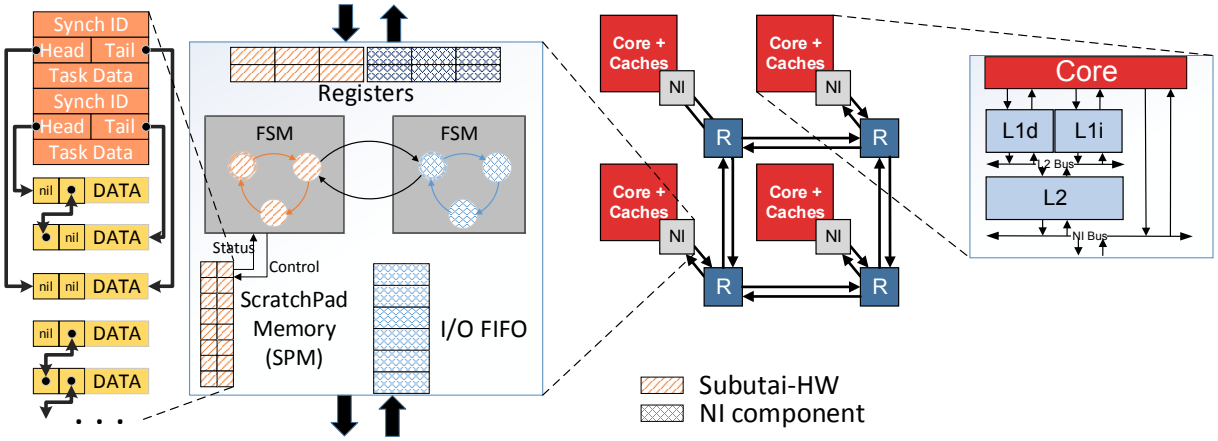


Figure 6.19 – Subutai-HW organization - specific Subutai-HW elements are highlighted in orange, whereas standard NI components are in blue.

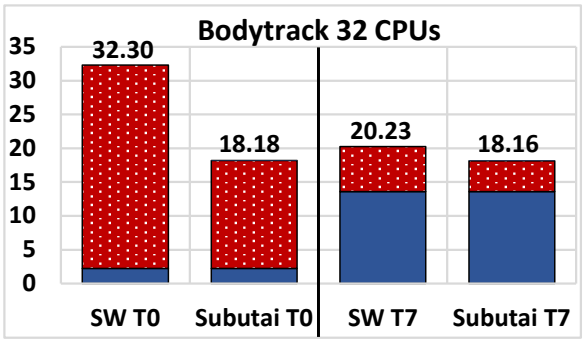


Figure 6.20 – Experimental results showing acceleration for a single parallel application. Values are in seconds of execution; the dotted red color is the sum of synchronization operations; the flat blue color is processing time.

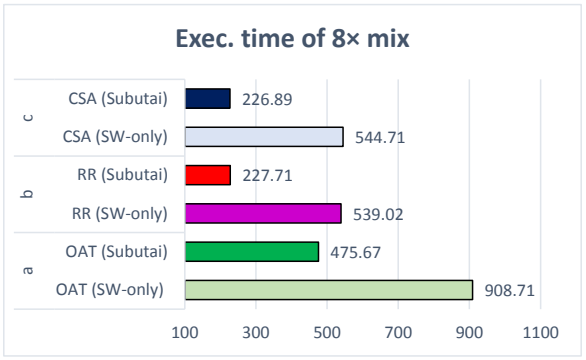


Figure 6.21 – Execution in seconds for multiple applications (lower is better) - (Exec = Execution). CSA: Critical Section Aware, RR: Round Robin, OAT: One At a Time.

Results

A full set of results is available in appendix C, including area, acceleration of a single application and multiple applications, and synthetic benchmarks. This paragraph presents only two figures. Figure 6.20 shows the results obtained for Bodytrack application. We analyze the entire application execution but plot the results for two threads for each application: the master thread (T_0), responsible for global synchronization, and a worker thread instance (T_7). Besides, the results are divided into two: synchronization operations and processing. The former aggregates all calls to PThreads (e.g., mutex lock), while the latter collects the processing needed by the application. NoC communication and Subutai-HW latencies did not contribute significantly to the execution time; thus, they are not visually perceivable on the figure, although they are present. Nonetheless, the figure shows that our solution reduces the application total time by handling synchronization faster.

Figure 6.21 displays the experimental results for a mix of multiple applications: a combination of 3, 2 and 3 instances of Bodytrack, x264, and Streamcluster, respectively. All applications have been set to use 64 threads and cores without restriction regarding mapping threads to cores. The execution time has been measured for three different scheduling strategies for executing this combination of applications. The first one, starting from the bottom, is the One At a Time (OAT) strategy. Each application is simply executed alone in a sequential manner. The figure show the results for SW-only (the original Linux), and the Subutai solution. The second strategy is the Round Robin (RR) scheduling policy. The third strategy

is the proposed policy, the Critical Section Aware (CSA) policy, which gives priority to threads that hold a mutex. When several threads hold a mutex or no threads hold any mutex, the RR policy decides. The results show that Subutai systematically improves the execution time.

The best configuration is Subutai with the CSA scheduling policy, reaching a 2.4× speed-up over the original Pthread implementation with CSA policy.

6.5 Summary

This chapter presented our contribution in the study of scalability of dataflow applications on SMP architectures. It also presented our contributions on hardware-based solutions for synchronisation. The concept of notifying memories is specifically introduced to improve the synchronisation of dataflow applications. The chapter ends with Subutai, a solution for Pthread based applications.

How does a dataflow application scales with the increasing number of cores?

Does the execution time of a dataflow application reduces linearly as the number of cores grows?

We performed an empirical study on the scalability of SDF applications on SMP architectures. The results show that the foreseen future of still ever more cores and bigger caches does not guarantee software-free better performances.

Synchronizing dataflow applications on a multicore processor

Is there any other more efficient hardware approach to synchronise dataflow applications on a multicore processor?

We studied the two big families of synchronisation: lock-based and lock-free. We proposed a new architectural approach that breaks the conventional Von Neumann architecture by giving the possibility to the memory to initiate transactions on the communication medium.

Synchronizing parallel applications on a multicore processor

- A new hardware module embedded in the network interface of the network-on-chip
- A new capacity given to the memory to notify a processor when a specific value is written at a given address

The experiments are performed through system-level simulators. The notifying memories reduce greatly the number of useless memory accesses in the case of lock-free configuration. The new synchronisation mechanisms proposed help in the scalability of the applications in the lock-based configuration.

Synchronizing parallel applications on a multicore processor

We proposed:

- the notifying memories concept for dataflow applications
- Subutai for Pthread applications

Though providing good performance, the notifying memories or Subutai suffer from the same limitation as all hardware synchronisation mechanisms: eventually it runs out of space [151]. The question of sizing becomes thus a major concern. Rollback methods using software-based solution is the good option that we did not explore in our work.

The results show that interesting speedups can be reached by enhancing the synchronisation of parallel applications, without any dedicated hardware accelerators or bigger caches or wider interconnects, which lead to reducing the data movements in the cache hierarchy.

Contributions wrap-up

With the slowdown of Moore's law and the end of Dennard's scaling, the performance of application is coming from parallelism [38]. Hardware architectures can make use of the different types of parallelism in two dimensions, the temporal dimension and the spacial dimension. In our work, we focused on three types of parallelism: ILP (Instruction Level Parallelism), DLP (Data Level Parallelism), and TLP (Task Level Parallelism).

We first proposed a programmable hardware accelerator, falling in the category of Coarse Grained Reconfigurable Architectures (CGRA), that can make use of ILP and DLP, presented in chapter 4. The accelerator is a standalone component, able to execute a full application, including its control flow, and is studied in the context of embedded systems. The energy efficiency is better than an embedded processor.

Takeaway from exploiting ILP and DLP with CGRAs

- Architectural support and mapping approach to execute full application (including control flow and loop control) on a CGRA
- IPA (Integrated Programmable Array) for integer operations
  - ▷ maximum of 8× speed-up, with a minimum of 2.49× and an average of 5.4×
- TRANSPIRE to support transprecision
  - ▷ 10× improvement on execution time, and up to 12× in energy

The parallelism of an application should be expressed through languages that help in explicitly specifying the types of parallelism. The dataflow model of computation offers an interesting and convenient way to express spacial and temporal parallelism, which helps in the mapping of such applications on parallel architectures. But the natural dynamic behaviour of some applications forces to adapt the mapping at runtime to improve the performance.

We proposed two mapping algorithms. A first algorithm (runtime mapping) is proposed to find rapidly a first solution. A second algorithm (runtime remapping) is proposed to adapt at runtime the mapping while the application runs.

Takeaway from exploiting ILP and DLP with multicore processors

- Runtime mapping algorithm for dataflow applications on a multicore processor
  - A fast algorithm to find a first solution at runtime, taking into account communication cost
  - 50× faster solving time for similar quality of results
- Runtime remapping algorithm for dataflow applications on a multicore processor
  - Adaptation at runtime to the dynamic behaviour of dataflow applications
  - Move-based algorithm to not fully change the mapping at once
  - up to 26% improvement in throughput compared to the baseline (no remapping)

One of the major issue in exploiting parallelism is synchronisation. When several tasks share the same data, some mechanisms are needed to guarantee the correct results of the executed application.

We proposed an original mechanism to synchronise dataflow applications, but making use of the intrinsic availability of their firing rules to improve the synchronisation between the actors. We also proposed a Pthread compatible solution to improve the execution time of legacy parallel applications.

Takeaway from synchronisation

- Notifying memories
  - Synchronizing dataflow applications on a multicore processor
  - Lock-free synchronisation for DPN application
    - Deletion of useless memory accesses (-78% latency, -60% injection rate)
    - +15% in throughput for a video decoder
  - Lock-based synchronisation for SF applications
    - Making scalable a non scaling application
    - 2× speedup for a synchronisation bound application on 16 cores
- Subutai: a hardware/software solution for synchronizing Pthread applications
  - Management of synchronisation mechanisms inside the network interface of the network on chip
  - Pthread compatible
  - 2.71× speedup over a single application
  - 4.61× speedup over multiple applications

III Perspectives and conclusion

7. Perspectives

This chapter presents three main perspectives identified in the continuity of the work presented in the core of the document. Firstly, CGRAs are knowing a new momentum with the boom of artificial intelligence (AI) applications, as they are the relevant (if not the only credible) solution to take up the energy-efficient challenge raised by this application domain. The question about how AI can help in the design and programming of CGRAs is also addressed.

The second perspective is related to emerging technology coming to the rescue (once again). Particularly, the wireless communication capabilities are now foreseen inside a chip. This solution might be an interesting way to solve the congestion problems in current wired interconnects. A project is currently ongoing on the topic.

The third perspective proposed pushes the computing capabilities inside the network on chip. The routers in current networks become not only simple forwarders of data. Augmented with simple computing operators, routers can operate seamlessly on the data that goes through to perform simple operations.

7.1 CGRAs for AI, AI for CGRAs

The first wave of CGRA was fuelled by signal processing applications, especially multimedia applications like image, audio, and video, for embedded systems, constrained by stringent power and energy budget. The Samsung Reconfigurable Processor (SRP) [121], an ADRES-like CGRA, integrated in the past in the Exynos SoC, is an example of a commercial use of CGRAs. The choice of Samsung to discontinue the use of SRP in favour of more conventional processors is the sign of a mitigated success [3].

CGRAs and AI

What can CGRAs can do for AI? What AI can do for CGRAs?

7.1.1 Trends

CGRAs know a new momentum as they get carried away by artificial intelligence (AI) applications. The massive need of high-performance computing, coupled with the slowdown of Moore's law and end of Dennard scaling, and the mismatch between AI workloads and conventional Von Neumann architectures, drives the efforts towards a multitude of AI-accelerators, which fall in the category of CGRAs. The diversity of names in the literature also shows that the domain is buzzing: Xilinx AI-engine [41], Reconfigurable Dataflow Architecture [31], Reconfigurable Dataflow Accelerator [36]. These "modern" CGRAs differ from the legacy ones in the number of cells that are available, which causes a serious scalability issue that is discussed in the challenge section. Another difference is the coupling with a host CPU. Modern CGRAs tend to be standalone, similarly to a GPU, and they require a full system integration. CGRAs are the relevant (if not the only credible) solution to take up the challenge of energy-efficient AI applications. The second wave of CGRAs might eventually be the one that meets an industrial success.

The mapping problem is complex, and needs sophisticated algorithms that are time consuming to understand, and to formalize. The methods based on artificial intelligence and machine learning are clearly interesting trails [60]. AI-based methods help in concentrating within a single model some functional but also non functional contraints, that are hard to formalize through traditional methods. AI for electronic design automation in general is at its early beginnings, and AI for CGRA specifically will be part of this global trend.

From the architectural point of view, some evolutions will obviously impact the compilation. The CGRA coupling can also be further explored: near the memory, or directly integrated within the memory array in a *processing-in-memory* manner [61]. The emerging memory technologies will also be game changers.

Finally, some open-source frameworks recently appeared [17, 33, 46, 50] to share the technical efforts and provide a ready for use tool to democratize the CGRAs and make them widely adopted for energy-efficient or high-performance computing. These frameworks are also part of a wider trend about open source hardware.

7.1.2 Challenges and opportunities

Programming model

The inadequate programming model used up to now for CGRAs is the main limitation identified in two recent surveys [45, 61]. Other programming models needs to be considered, more adapted to CGRAs, able to specify the data-level parallelism, like OpenMP, SYCL, CUDA or OpenCL. The dataflow model of computation could also be interesting to look at. This kind of streaming model can fit with CGRAs [26, 94].

Scalability

Scalability is clearly one of the biggest challenge to be taken up. Some techniques are already proposed do deal with scalability. In [34], the repetitive patterns of loops are detected and are mapped in a hierarchical way. In our work [100], the partial solutions are stochastically pruned to keep under control their number. But while legacy CGRAs are composed of tens of cells, with a use rate quite low limited by the instruction level parallelism available in the applications, the more recent and modern CGRAs, the most capable of crunching AI workloads, contains hundreds to thousands of cells. The issue is to effectively make use of the massive number of cells. The standalone feature of modern CGRAs is another game changer for mapping methods. The mapping problem is intractable, scalability further raises the challenge, and the number of cells involved takes it in yet another dimension. The application should be considered as a whole, not with intensive kernels to be offload to the CGRA and letting the host processor interact with the system. A holistic approach is thus needed to first analyse the input application, and then relevantly partition it for finally an efficient complete mapping. SARA [36] is such a recent approach. It relies on a hierarchical pipelining to extract spatial parallelism and temporal parallelism. For instance, at loop level, two sibling loops might be executed in parallel. When there are data dependencies across the loops,

the memory consistency is managed by the compiler, and the instructions are ordered to guarantee the correct execution. The iterations of the loops are overlapped at all levels as an advanced implementation of software pipelining (not only modulo scheduling). In other words, the new generation of compilers for CGRAs must be able to make use of all levels of parallelism: instruction level, data level, and loop level.

7.2 Wireless Network-On-Chip

The wired communication inside a chip shows limitations, especially when it comes to send multiple times the same packet (broadcast/multicast) which lead to an increase in latency and energy consumption. Other physical communication means are studied like optical communication or wireless communication. We focus here on wireless communication. It is possible today to embed an antenna inside a chip, with CMOS compatible technology, and benefit from the huge knowledge gathered on wireless communication to adapt it to intra-chip communication. Wireless communication holds a natural broadcast capability and is foreseen to be a solution to reduce the latency of communication.

Wireless Network-On-Chip
Use radio-frequency techniques to communicate inside a chip.

7.2.1 Trends

In a NoC, several types of communications are needed: one-to-one communication (unicast), one-to-many communication (multicast), and one-to-all communication (broadcast). A straightforward solution to implement multicast or broadcast in a wired NoC consists in making the sender send n times the same packet. More efficient solutions can embed in hardware the multicast and broadcast capabilities [163]. But as the number of cores (or more generally communicating elements) increases, a wired NoC faces serious communication bottlenecks. WiNoC (Wireless Network on Chip) has emerged as an alternative for long range communication and a viable solution for the implementation of multicast and broadcast [67, 79] communications on large manycore.

Figures 7.1 and 7.2 show two illustrations of the integration of wireless technology inside a manycore architecture. In [19], the authors proposed the illustration of figure 7.1, where each core has an antenna and RF communication possibilities. In [20], the authors proposed the illustration of figure 7.2, a cluster-based implementation where the communication in the cluster is based on wired communication and the intercluster communication is based on wireless.

7.2.2 Challenges and opportunities

Several challenges come along with the WiNoC approach. First, a wireless interface is very costly in terms of power consumption and area. In [19], the authors claim that the RF part occupies 0.4 mm^2 at 65 nm technology. According to [64], at 22 nm, it is possible to expect a 0.14 mm^2 footprint for the antenna. In [15], a cluster is composed of 9 RISC-V processors, and occupies a surface of 1.48 mm^2 at 22 nm technology, leading to an average surface of 0.16 mm^2 per core. The size of the antenna at 22 nm is thus similar to the size of a single core. Solutions planning one antenna per core thus double the area of the chip. One antenna per cluster at 22 nm would lead to a reasonable 10%area overhead.

Another difficulty is the passage from the digital world to the analog world for transmission, and vice-versa for reception. For instance, in [20], an Orthogonal Frequency Division Multiplexing (OFDM)-based

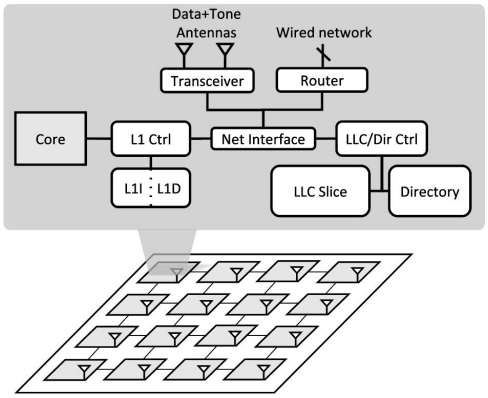


Figure 7.1 – Illustration of the WiNoC adopted in [19]

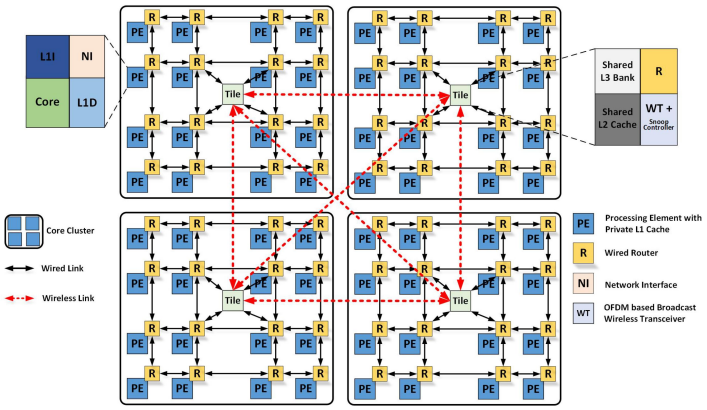


Figure 7.2 – Illustration of the WiNoC adopted in [20]

transceiver is used for inter-cluster communication, which needs to go through an FFT (Fast Fourier Transform) step. The time needed to go through the steps should be carefully compared with the time in the wired domain. It is clear that for one-hop communications, the wireless path is less interesting. For long distance communications, the wireless approach should be considered.

Wireless communications are studied for enhancing cache coherence in shared memory multicore processors, for invalidation-based directory cache coherence protocol [19] or snooping based protocols [20]. Invalidate protocols are the option opted in today's wired-based multicore processors. But thanks to its natural broadcast capabilities, wireless communications could be the interesting means to implement efficient write update protocols, weakly studied in the wired domain. Indeed, in a write update protocol, all writes to a shared cache line must be broadcasted, consuming high bandwidth in the wired domain [95], but naturally supported in wireless.

Lastly, application mapping is an important aspect of WiNoC based multi-core platform, especially in the case of cluster-based architecture with mixed wired/wireless communications, when the two paths are possible. The challenge is to fully make use of the available bandwidth. Applications have different profiles and varying communication requirements regarding broadcast and unicast communications. The communicating tasks in the application need to be mapped close to one another to reduce long distance communication. Dynamic behaviours of applications further raise the challenge.

7.3 Computing in network

Finding the ideal meeting point between software models and hardware implementations is a recurring issue since the early days of computer engineering. The software community can provide nice abstractions and formalisms that are not suited for hardware implementation. On the other side, the hardware community can provide very efficient and dedicated hardware components that are difficult to efficiently program because of a shallow abstraction model. The good-old combination between the processor and Von Neumann architecture with the imperative programming model is smashing into pieces because of memory reasons and energy efficiency drops, especially for machine learning applications. The perspective is to tackle both problems and finally propose a new holistic software/hardware model by integrating processing capabilities all along the path from the main memory to the processor.

Computing in network

Compute the data on its way in the network on chip.

7.3.1 Trends

Assigning a processor to compute a low number of very simple and basic operations is extremely inefficient from an energy point of view. Consider an example of a simple arithmetic operation like an addition. It requires 2 read memory accesses (to load the two operands) and one write access (to store the result). This represents 3 memory accesses for a single, simple arithmetic operation. In a current many-core architecture interconnected through a network-on-chip (NoC), the data needs to take a long path from the main memory to the cores: AXI Fabric, DMA (Direct Memory Access), NoC Router, L2 cache, L1 cache.

To avoid the incessant transfers of data, an idea, quite old but topical, is to perform the calculations where the data is located: in the memory. In the example of a simple addition, it would certainly be more relevant and less expensive to "bring the computation into memory". There are two types of approach: 1) Near-Memory Computing (NMC), 2) In-Memory Computing (IMC). Near memory computing (NMC) consists of adding computing capacities (typically processors) as close as possible to the main memory of the computer, thus bypassing the hierarchy of caches, and limiting its numerous data transfers [63]. In-Memory Computing (IMC) goes even further by integrating computing capabilities directly into the memory technology [84]. The two approaches, Near-memory Computing and In-Memory Computing, find many obstacles for an effective implementation. From the technological aspects of physical integration up to the application programming interface available to the programmer, all layers of the system are impacted: memory technology (DRAM or NVM), interconnection architecture, programmability (virtual addresses, languages, models), not to talk about the associated tool chains.

The idea of *data-stream processing* is to integrate very simple processing elements inside the routers of the network on chip to compute on the data as long as it comes before transferring it to the next router [29, 49, 96]. In [96], the processing part embedded in the routers is actually limited to multiplication with pre-loaded constants and addition, which is typically needed for filters in digital signal processing. The authors focus on an FPGA implementation, and do not study the execution time of an application. In [49], the authors designed a NoC with computing capabilities inside the routers. This very interesting architectural idea allows for 6× speed-up for the considered linear algebra kernels. The main limitation of this approach is a need for transforming (i.e. rewriting) the applications in order to express a producer-consumer data model. We believe that the dataflow approach naturally follows this pattern, and this direction is also explored in [29]. The authors use a KPN model of computation and claim a speed-up of 5× for a single task. [49] and [29] are the only two works that are close to the idea of this project, and we believe there is a wide unexplored area around this concept, including the model of computations, the model of architecture, the memory model, and compilation and computer-aided design (CAD) tools. Moreover, none of these papers discuss energy results whereas such a solution should have a major impact.

Several other approaches distort the routers to implement interesting functionalities [39, 40, 58]. In [39] and [40], the authors use the buffers of the routers to keep the evicted data as much as possible inside the many-core chip, avoiding thus the expensive off-chip transfers. The authors claim an average speed-up of 7% to 12% but do not discuss the impact on the energy. In [58], the authors use the routers to implement a TLB (Translation Look-aside Buffer), the component involved in the translation of a virtual address to the physical address. The authors are more interested in plugging processors that do not embed TLB rather than the impact on the memory transfers or the energy.

Some processing capabilities have also been integrated into a DMA [80]. The reconfigurability is actually limited as the component supports four operating modes, which is usually referred in the literature as a *multi-mode* component. This approach is also questionable from an implementation point of view since a DMA component manages data transfers but the data does not travel *through* the DMA, but rather through the interconnection network. The DMA supervises the transfer between two existing components,

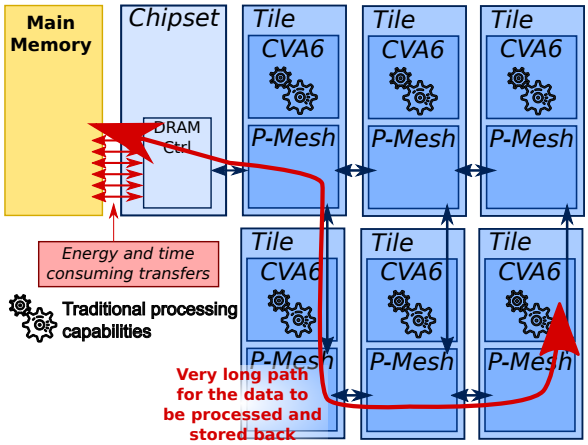


Figure 7.3 – A typical NoC-based many-core architecture showing the long path between the main memory and a processing element.

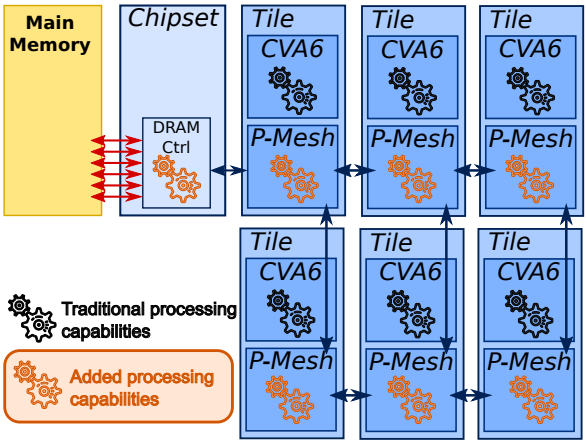


Figure 7.4 – The architecture with added processing capabilities inside the routers of the network-on-chip

deals with addresses of the data but not the values of the data. In this project, we intend to go further by designing truly programmable components, driven by the needs of the application, inside the different components involved in data transfer: DRAM controller, DMA, and routers.

7.3.2 Challenges and proposals

Architectural challenge

Figure 7.3 shows a typical many-core architecture based on a NoC and the long path that the data needs to go through before being processed. The figure also shows the connection with the main memory, which is typically designed based on another technology that the many-core architecture and lies in another chip. The red links that connects the two chips are very high energy and time consuming. Reducing as much as possible the transfers between the two chips lead to high energy gains, and caches inside the many-core chip contribute to the gains.

The architectural proposition is to embed some computing capabilities all along the path between the main memory and the processor. The challenge to be taken up in such an approach is to add the new processing capabilities without degrading the performance of the original functionality of the routers. The performance of the NoC is critical, and degrading its original performance is unacceptable. The area overhead and frequency should be carefully monitored. A trade-off between the area footprint, power consumption, and utilisation rate of the component is expected. A clock gating technique is foreseen to manage the power, which is not considered in previous works [29, 49]. The design space to explore also includes the programmability feature of the component. In [49], an ordered instruction buffer is used to store the instruction of the operations to perform on the data. The instructions are provided by a *Central Packet Manager*, a global controller of the platform. In [96], the instruction is directly embedded inside the packet of the data, which limits the number of operations that can be executed.

Figure 7.4 shows the same figure as figure 7.3 with the proposed added processing capabilities. These new capabilities will allow to pre-process or post-process some data while it's travelling between the main memory and the processor, possibly also through the caches of other tiles. The processing capabilities shall be based on a very simple programmable or reconfigurable architecture such as a CGRA.

Programming challenge

The challenge is to seemlessly programme this kind of architecture. Figure 7.5 illustrates the architecture supporting computing in network programmed from a dataflow application. The network of actors shown is actually a very little subset of the Squeezenet application, a deep neural network for computer vision, designed for small networks and lower number of parameters, while achieving the same level of accuracy

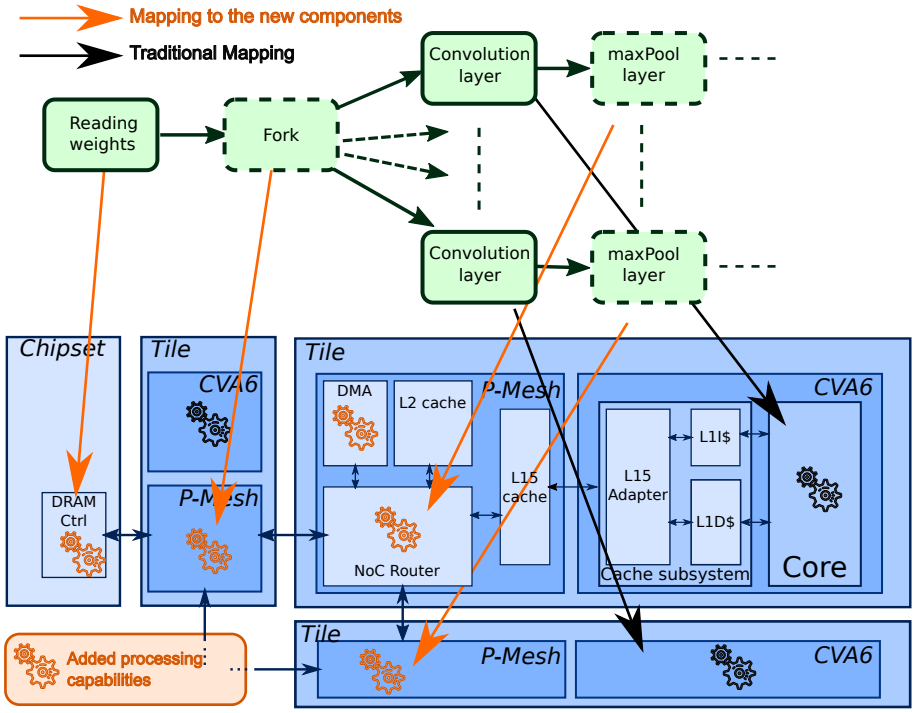


Figure 7.5 – Illustration of the concept of computing in network, realised from a dataflow application

than bigger networks.

A representative example in the figure is the *Fork* actor, which is usually responsible for (naively and inefficiently) replicating data from a producer to several consumers, leading to numerous memory transfers with no computations. Some dataflow models of computation like PAFG (passive-active flow graph) [44] allows designers to express such a behaviour much more efficiently and opens the way to design tools and methods for a consistent hardware/software model. Following a relevantly scheduled data stream, the convolution layer can be efficiently mapped on the processor.

Another representative example is the *maxPool* actor, which keeps the maximum value over a set of values. Finding the maximum value is a simple comparison from a computation point of view, and is a typical case of lots of memory accesses for a very few computations. Furthermore, the if-condition leads to branches that are poorly compatible with processor pipelines. This *maxPool* actor is also known as a “reduction”, as the output produces only one value among N. But the tremendous amount of data to manage makes this simple step a memory bottleneck. For instance, the first layer of maxPool can reach an amount of several megabytes, far beyond the capacities of the L1 or L2 caches. But the *reduced* data, after finding the maximum value drops down to hundreds of kilobytes, which can fit in the cache. The challenge is to be able to seamlessly map these kinds of computation into the NoC router when the processor commits the results of the convolution layers.

8. Closing chapter

This document presented eleven years of teaching and research activities since I was hired as associate professor at Université de Bretagne-Sud. The first part of the document presented my CV, my teaching activity, and a summary of my scientific contributions, along with contributions to the research operation. The second part presented my research organised around three main areas. The research perspectives are presented in the third part of the document.

The document continues with the acronyms used throughout the description of my activities, and the references cited. Three appendixes are added at the end of the document, giving an emphasis on one contribution per research area.

Acronymes

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
BB	Basic Block
CAD	Computer-Aided Design
CAL	CAL Actor Language
CDFG	Control Data Flow Graph
CFG	Control Flow Graph
CGRA	Coarse Grained Reconfigurable Architecture
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multi-Processors
CP	Constraint Programming
CPU	Central Processing Unit
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DDR	Double Data Rate
DFT	Data Flow Tree
DLP	Data Level Parallelism
DMA	Direct Memory Access
DPN	Dataflow Process Network
DSP	Digital Signal Processor
DRAM	Dynamic Random Access Memory
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
GA	Genetic Algorithm
GPP	General Purpose Processor
GPU	Graphics Processing Unit

GRT	Global RunTime
HLS	High Level Synthesis
ILP	Integer Linear Programming
ILP	Instruction Level Parallelism
IP	Intellectual Property
IR	Intermediate Representation
ISA	Instruction Set Architecture
ITE	If-Then-Else
KPN	Kahn Process Network
LE	Logic Element
LLVM	Low Level Virtual Machine
LRT	Local RunTime
LUT	Look-Up Table
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MMU	Memory Management Unit
MoC	Model of Computation
MPPA	Multi-Purpose Processor Array
MPSoC	Multi-Processor System on Chip
NI	Network Interface
NoC	Network on Chip
NUMA	Non Uniform Memory Access
ORCC	Open RVC-CAL Compiler
PE	Processing Element
PiSDF	Parameterized and Interfaced Synchronous Dataflow
PREESM	Parallel and Real-time Embedded Executives Scheduling Method
QEA	Quantum-inspired Evolutionary Algorithm
RISC	Reduced Instruction Set Processor
RTL	Register Transfer Level
RVC	Reconfigurable Video Coding
SA	Simulated Annealing
SAT	Boolean satisfiability
SDF	Synchronous DataFlow
SHA	Secure Hash Algorithm
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
S-LAM	System Level Architectural Model
SMP	Symmetric MultiProcessor
	Symmetric shared-Memory Processors
SMT	Simultaneous MultiThreading
SMT	Satisfiability Modulo Theories
SPM	Scratch Pad Memory
SRAM	Static Random Access Memory
TCDM	Tightly Coupled Data Memory
TLM	Transaction Level Modeling
TLP	Task Level Parallelism
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Long Instruction Word
WiNoC	Wireless Network on Chip

IV Bibliography and selected publications

Bibliography

[1] *Xiph.org Video Test Media*. URL: http://media.xiph.org/video/derf/ (cited on pages 96, 108, 110).

[2] *Zynq, http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html*. URL: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (cited on page 77).

[3] Andrei Frumusanu. *The Samsung Exynos 7420 Deep Dive - Inside A Modern 14nm SoC*. URL: http://www.anandtech.com/show/9330/exynos-7420-deep-dive (visited on 03/15/2016) (cited on page 125).

[4] METIS. *Serial Graph Partitioning and Fill-reducing Matrix Ordering*. URL: http://glaros.dtc.umn.edu/gkhome/views/metis (cited on pages 81, 86).

[5] ORCC. *The Open RVC-CAL Compiler : A Development Framework for Dataflow Programs*. URL: http://orcc.sourceforge.net (cited on pages 80, 84, 86, 95, 107).

[6] PULP. *Parallel Ultra-Low Power platform*. https://pulp-platform.org (cited on page 63).

[7] VSP. *Cadence Virtual System Platform for the Xilinx Zynq-7000 All Programmable SoC*. URL: http://www.cadence.com/products/sd/virtual%5C_system (cited on page 86).

[8] Satyajit Das, Kevin J. M. Martin, Thomas Peyret, and Philippe Coussy. "An Efficient and Flexible Stochastic CGRA Mapping Approach". In: *ACM Trans. Embed. Comput. Syst.* 8 (Oct. 2022), page 24. ISSN: 1539-9087. DOI: 10.1145/3550071. URL: https://doi.org/10.1145/3550071 (cited on page 66).

[9] Alemeh Ghasemi. "Notifying Memories for Dataflow Applications on Shared-Memory Parallel Computer". Theses. Université Bretagne-Sud, May 2022. URL: https://tel.archives-ouvertes.fr/tel-03704297 (cited on pages 112, 113).

[10] Alemeh Ghasemi, Marcelo Ruaro, Rodrigo Cataldo, Jean-Philippe Diguet, and Kevin Martin. "The Impact of Cache and Dynamic Memory Management in Static Dataflow Applications". In: *Journal of Signal Processing Systems* (2022). DOI: 10.1007/s11265-021-01730-7. URL: https://hal.archives-ouvertes.fr/hal-03606524 (cited on pages 104, 106).

[11] Kevin J M Martin. "Twenty Years of Automated Methods for Mapping Applications on CGRA". In: *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Lyon, France, May 2022. DOI: 10.1109/IPDPSW55747.2022.00118. URL: https://hal.archives-ouvertes.fr/hal-03704256 (cited on page 50).

[12] RISC-V. Community News. *Andes Announces RISC-V Multicore 1024-Bit Vector Processor: AX45MPV*. en-US. Dec. 2022. URL: https://riscv.org/news/2022/12/andes-announces-risc-v-multicore-1024-bit-vector-processor-ax45mpv/ (visited on 12/12/2022) (cited on page 46).

[13] Rohit Prasad. "Integrated Programmable-Array accelerator to design heterogeneous ultra-low power manycore architectures". Theses. Université de Bretagne Sud ; Università degli studi (Bologne, Italie). Facoltà di Ingegneria, Jan. 2022. URL: https://tel.archives-ouvertes.fr/tel-03701879 (cited on page 70).

[14] Mostafa Rizk, Kevin J. M. Martin, and Jean-Philippe Diguet. "Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs". In: *IEEE Transactions on Parallel and Distributed Systems* (2022), pages 1–1. DOI: 10.1109/TPDS.2022.3177957 (cited on page 83).

[15] Davide Rossi, Francesco Conti, Manuel Eggiman, Alfio Di Mauro, Giuseppe Tagliavini, Stefan Mach, Marco Guermandi, Antonio Pullini, Igor Loi, Jie Chen, Eric Flamand, and Luca Benini. "Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode". In: *IEEE Journal of Solid-State Circuits* 57.1 (Jan. 2022), pages 127–139. DOI: 10.1109/jssc.2021.3114881. URL: https://doi.org/10.1109%2Fjssc.2021.3114881 (cited on page 127).

[16] Mark Wijtvliet, Henk Corporaal, and Akash Kumar. “Architectural Model”. In: *Blocks, Towards Energy-efficient, Coarse-grained Reconfigurable Architectures*. Cham: Springer International Publishing, 2022, pages 151–180. ISBN: 978-3-030-79774-4. DOI: 10.1007/978-3-030-79774-4_6. URL: https://doi.org/10.1007/978-3-030-79774-4%5C_6 (cited on page 51).

[17] Jason Anderson, Rami Beidas, Vimal Chacko, Hsuan Hsiao, Xiaoyi Ling, Omar Ragheb, Xinyuan Wang, and Tianyi Yu. “CGRA-ME: An Open-Source Framework for CGRA Architecture and CAD Research : (Invited Paper)”. In: *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2021, pages 156–162. DOI: 10.1109/ASAP52443.2021.00030 (cited on page 126).

[18] Michael Canesche, Marcelo Menezes, Westerley Carvalho, Frank Sill Torres, Peter Jamieson, José Augusto Nacif, and Ricardo Ferreira. “TRAVERSAL: A Fast and Adaptive Graph-Based Placement and Routing for CGRAs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.8 (2021), pages 1600–1612. DOI: 10.1109/TCAD.2020.3025513 (cited on page 54).

[19] Antonio Franques, Apostolos Kokolis, Sergi Abadal, Vimuth Fernando, Sasa Misailovic, and Josep Torrellas. “WiDir: A Wireless-Enabled Directory Cache Coherence Protocol”. In: *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2021. DOI: 10.1109/HPCA51647.2021.00034 (cited on pages 127, 128).

[20] Sri Harsha Gade and Sujay Deb. “A Novel Hybrid Cache Coherence with Global Snooping for Many-Core Architectures”. In: *ACM Trans. Des. Autom. Electron. Syst.* 27.1 (Sept. 2021). ISSN: 1084-4309. DOI: 10.1145/3462775. URL: https://%doi.org/10.1145/3462775 (cited on pages 127, 128).

[21] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. “Snafu: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pages 1027–1040. DOI: 10.1109/ISCA52012.2021.00084 (cited on page 54).

[22] Yijiang Guo, Jiarui Wang, Jiaxi Zhang, and Guojie Luo. “Formulating Data-arrival Synchronizers in Integer Linear Programming for CGRA Mapping”. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pages 943–948. DOI: 10.1109/DAC18074.2021.9586267 (cited on pages 50, 54).

[23] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. *The Art of Multiprocessor Programming (Second Edition)*. Edited by Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. Second Edition. Morgan Kaufmann, 2021. DOI: 10.1016/C2011-0-06993-4. URL: https://doi.org/10.1016/C2011-0-06993-4 (cited on pages 47, 101, 102).

[24] Jinho Lee and Trevor E. Carlson. “Ultra-Fast CGRA Scheduling to Enable Run Time, Programmable CGRAs”. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pages 1207–1212. DOI: 10.1109/DAC18074.2021.9586255 (cited on pages 50, 54).

[25] Cheng Li, Jiangyuan Gu, Shouyi Yin, Leibo Liu, and Shaojun Wei. “Combining Memory Partitioning and Subtask Generation for Parallel Data Access on CGRAs”. In: *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2021, pages 204–209. DOI: 10.1145/3394885.3431414. URL: https://doi.org/10.1145/3394885.3431414 (cited on page 56).

[26] Zhaoying Li, Dhananjaya Wijerathne, Xianzhang Chen, Anuj Pathania, and Tulika Mitra. “ChordMap: Automated Mapping of Streaming Applications onto CGRA”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), pages 1–1. DOI: 10.1109/TCAD.2021.3058313 (cited on pages 54, 82, 126).

[27] Yukio Miyasaka, Masahiro Fujita, Alan Mishchenko, and John Wawrzynek. “SAT-Based Mapping of Data-Flow Graphs onto Coarse-Grained Reconfigurable Arrays”. In: *VLSI-SoC: Design Trends*. Edited by Andrea Calimera, Pierre-Emmanuel Gaillardon, Kunal Korgaonkar, Shahar Kvatinsky, and Ricardo Reis. Cham: Springer International Publishing, 2021, pages 113–131. ISBN: 978-3-030-81641-4 (cited on pages 50, 54).

[28] Song Mu, Yi Zeng, and Bo Wang. “Routability-Enhanced Scheduling for Application Mapping on CGRAs”. In: *IEEE Access* 9 (2021), pages 92358–92366. DOI: 10.1109/ACCESS.2021.3092781 (cited on page 54).

[29] Jens Rettkowski and Diana Göhringer. “Wormhole Computing in Networks-on-Chip”. In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 2021, pages 273–274. DOI: 10.1109/FPL53798.2021.00054 (cited on pages 129, 130).

[30] Claudion Rubattu, Francesca Palumbo, Shuvra Bhattacharyya, and Maxime Pelcat. “PathTracing: Raising the Level of Understanding of Processing Latency in Heterogeneous MPSoCs”. In: *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*. DroneSE and RAPIDO ’21. Budapest, Hungary: ACM, 2021, pages 46–50. ISBN: 9781450389525. DOI: 10.1145/3444950.3447282. URL: https://doi.org/10.1145/3444950.3447282 (cited on page 82).

[31] SambaNova. *Accelerated Computing with a Reconfigurable Dataflow Architecture*. SambaNova systems. June 2021. URL: https://sambanova.ai/wp-content/uploads/2021/06/SambaNova%5C_RDA%5C_Whitepaper%5C_English.pdf (cited on page 126).

[32] Chilankamol Sunny, Satyajit Das, Kevin J. M. Martin, and Philippe Coussy. “Hardware Based Loop Optimization for CGRA Architectures”. In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Edited by Steven Derrien, Frank Hannig, Pedro C. Diniz, and Daniel Chillet. Cham: Springer International Publishing, 2021, pages 65–80. ISBN: 978-3-030-79025-7 (cited on page 55).

[33] Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, and Antonino Tumeo. “AURORA: Automated Refinement of Coarse-Grained Reconfigurable Accelerators”. In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2021, pages 1388–1393. DOI: 10.23919/DATE51398.2021.9473955 (cited on page 126).

[34] Dhananiaya Wijerathne, Zhaoying Li, Anuj Pathania, Tulika Mitra, and Lothar Thiele. “HiMap: Fast and Scalable High-Quality Mapping on CGRA via Hierarchical Abstraction”. In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2021, pages 1192–1197. DOI: 10.23919/DATE51398.2021.9473916 (cited on pages 53, 54, 126).

[35] Baofen Yuan, Jianfeng Zhu, Xingchen Man, Zijiao Ma, Shouyi Yin, Shaojun Wei, and Leibo Liu. “Dynamic-II Pipeline: Compiling Loops with Irregular Branches on Static-Scheduling CGRA”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), pages 1–1. DOI: 10.1109/TCAD.2021.3121346 (cited on pages 55, 59).

[36] Yaqi Zhang, Nathan Zhang, Tian Zhao, Matt Vilim, Muhammad Shahbaz, and Kunle Olukotun. “SARA: Scaling a Reconfigurable Dataflow Accelerator”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pages 1041–1054. DOI: 10.1109/ISCA52012.2021.00085 (cited on page 126).

[37] Mahesh Balasubramanian and Aviral Shrivastava. “CRIMSON: Compute-Intensive Loop Acceleration by Randomized Iterative Modulo Scheduling and Optimized Mapping on CGRAs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pages 3300–3310. DOI: 10.1109/TCAD.2020.3022015 (cited on pages 54–56).

[38] William J. Dally, Yatish Turakhia, and Song Han. “Domain-Specific Hardware Accelerators”. In: *Commun. ACM* 63.7 (June 2020), pages 48–57. ISSN: 0001-0782. DOI: 10.1145/3361682. URL: https://doi.org/10.1145/3361682 (cited on page 121).

[39] Abhijit Das, Abhishek Kumar, and John Jose. “Reducing Off-Chip Miss Penalty by Exploiting Underutilised On-Chip Router Buffers”. In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 2020, pages 230–238. DOI: 10.1109/ICCD50377.2020.00049 (cited on page 129).

[40] Abhijit Das, Abhishek Kumar, John Jose, and Maurizio Palesi. “Exploiting On-Chip Routers to Store Dirty Cache Blocks in Tiled Chip Multi-processors”. In: *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2020, pages 147–152. DOI: 10.1109/ISVLSI49217.2020.00036 (cited on page 129).

[41] Alok G. “Architecture Apocalypse Dream Architecture for Deep Learning Inference and Compute-VERSAL AI Core”. In: *Embedded World*. 2020 (cited on page 126).

[42] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes*. English. Version 325462-072US. Intel Corporation. 2020. 5052 pages. May, 2020 (cited on page 106).

[43] Takuya Kojima, Nguyen Anh Vu Doan, and Hideharu Amano. “GenMap: A Genetic Algorithmic Approach for Optimizing Spatial Mapping of Coarse-Grained Reconfigurable Architectures”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.11 (2020), pages 2383–2396. DOI: 10.1109/TVLSI.2020.3009225 (cited on pages 54, 57).

[44] Yaesop Lee, Yanzhou Liu, Karol Desnos, Lee Barford, and Shuvra S. Bhattacharyya. “Passive-Active Flowgraphs for Efficient Modeling and Design of Signal Processing Systems”. In: *Journal of Signal Processing Systems* 92.10 (Oct. 2020), pages 1133–1151. ISSN: 1939-8115. DOI: 10.1007/s11265-020-01581-8. URL: https://doi.org/10.1007/s11265-020-01581-8 (cited on page 131).

[45] A. Podobas, K. Sano, and S. Matsuoka. “A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective”. In: *IEEE Access* 8 (2020), pages 146719–146743. DOI: 10.1109/ACCESS.2020.3012084 (cited on pages 50, 51, 126).

[46] A. Podobas, K. Sano, and S. Matsuoka. “A Template-based Framework for Exploring Coarse-Grained Reconfigurable Architectures”. In: *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2020, pages 1–8. DOI: 10.1109/ASAP49362.2020.00010 (cited on page 126).

[47] Rohit Prasad, Satyajit Das, Kevin Martin, Giuseppe Tagliavini, Philippe Coussy, Luca Benini, and Davide Rossi. “TRANSPIRE: An energy-efficient TRANSprecision floating-point Programmable archItectuRE”. In: *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. Grenoble, France, Mar. 2020, pages 1067–1072. DOI: 10.23919/DATE48585.2020.9116408. URL: https://hal.archives-ouvertes.fr/hal-02510931 (visited on 06/17/2020) (cited on page 72).

[48] V. Rathore, V. Chaturvedi, A. Singh, T. Srikanthan, and M. Shafique. “Longevity Framework: Leveraging Online Integrated Aging-Aware Hierarchical Mapping and VF-Selection for Lifetime Reliability Optimization in Manycore Processors”. In: *IEEE Transactions on Computers* (2020), pages 1–1. DOI: 10.1109/TC.2020.3006571 (cited on page 104).

[49] Karthik Sangaiah, Michael Lui, Ragh Kuttappa, Baris Taskin, and Mark Hempstead. “SnackNoC: Processing in the Communication Layer”. In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2020, pages 461–473. DOI: 10.1109/HPCA47549.2020.00045 (cited on pages 129, 130).

[50] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. “DSAGEN: Synthesizing Programmable Spatial Accelerators”. In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 2020, pages 268–281. DOI: 10.1109/ISCA45697.2020.00032 (cited on pages 54, 126).

[51] Zhongyuan Zhao, Weiguang Sheng, Qin Wang, Wenzhi Yin, Pengfei Ye, Jinchao Li, and Zhigang Mao. “Towards Higher Performance and Robust Compilation for CGRA Modulo Scheduling”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020), pages 2201–2219. DOI: 10.1109/TPDS.2020.2989149 (cited on page 54).

[52] Rodrigo Cadore Cataldo. “Subutai : Distributed synchronization primitives for legacy and novel parallel applications”. Theses. Université de Bretagne Sud ; Pontifícia universidade católica do Rio Grande do Sul, Dec. 2019. URL: https://tel.archives-ouvertes.fr/tel-02865408 (cited on page 115).

[53] S. Das, K. J. M. Martin, and P. Coussy. “Context-memory Aware Mapping for Energy Efficient Acceleration with CGRAs”. In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2019, pages 336–341. DOI: 10.23919/DATE.2019.8715288 (cited on page 67).

[54] Satyajit Das, Kevin J. M. Martin, Davide Rossi, Philippe Coussy, and Luca Benini. “An Energy-Efficient Integrated Programmable Array Accelerator and Compilation Flow for Near-Sensor Ultralow Power Processing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.6 (2019), pages 1095–1108. DOI: 10.1109/TCAD.2018.2834397 (cited on page 65).

[55] Caleb Donovick, Makai Mann, Clark Barrett, and Pat Hanrahan. “Agile SMT-Based Mapping for CGRAs with Restricted Routing Networks”. In: *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 2019, pages 1–8. DOI: 10.1109/ReConFig48160.2019.8994781 (cited on page 54).

[56] John L. Hennessy and David A. Patterson. “A New Golden Age for Computer Architecture”. In: *Commun. ACM* 62.2 (Jan. 2019), pages 48–60. ISSN: 0001-0782. DOI: 10.1145/3282307. URL: https://doi.org/10.1145/3282307 (cited on page 49).

[57] Kai Huang, Xiaomeng Zhang, Dandan Zheng, Min Yu, Xiaowen Jiang, Xiaolang Yan, Lisane B. de Brisolara, and Ahmed Amine Jerraya. "A Scalable and Adaptable ILP-Based Approach for Task Mapping on MPSoC Considering Load Balance and Communication Optimization". In: *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 38.9 (Sept. 2019), pages 1744–1757. ISSN: 1937-4151. DOI: 10.1109/TCAD.2018.2859400 (cited on pages 82, 93, 95).

[58] Hyeonguk Jang, Kyuseung Han, Sukho Lee, Jae-Jin Lee, and Woojoo Lee. "MMNoC: Embedding Memory Management Units into Network-on-Chip for Lightweight Embedded Systems". In: *IEEE Access* 7 (2019), pages 80011–80019. DOI: 10.1109/ACCESS.2019.2923219 (cited on page 129).

[59] Manupa Karunaratne, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. "4D-CGRA: Introducing Branch Dimension to Spatio-Temporal Application Mapping on CGRAs". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, pages 1–8. DOI: 10.1109/ICCAD45719.2019.8942148 (cited on pages 55, 59).

[60] Dajiang Liu, Shouyi Yin, Guojie Luo, Jiaxing Shang, Leibo Liu, Shaojun Wei, Yong Feng, and Shangbo Zhou. "Data-Flow Graph Mapping Optimization for CGRA With Deep Reinforcement Learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.12 (2019), pages 2271–2283. DOI: 10.1109/TCAD.2018.2878183 (cited on page 126).

[61] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications". In: *ACM Comput. Surv.* 52.6 (Oct. 2019). ISSN: 0360-0300. DOI: 10.1145/3357375. URL: https://doi.org/10.1145/3357375 (cited on pages 49–51, 53, 57, 126).

[62] Timothy G. Mattson, Yun (Helen) He, and Alice E. Koniges. *The OpenMP Common Core - Making OpenMP Simple Again*. Scientific and Engineering Computation. The MIT Press, 2019. ISBN: 9780262538862 (cited on page 76).

[63] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. "Near-memory computing: Past, present, and future". In: *Microprocessors and Microsystems* 71 (Nov. 2019), page 102868. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2019.102868. URL: http://www.sciencedirect.com/science/article/pii/S0141933119300389 (visited on 09/24/2019) (cited on page 129).

[64] Joel Ortiz Sosa. "Design of a Digital Baseband Transceiver for Wireless Network-on- Chip Architectures". PhD thesis. University of Rennes, France, Dec. 2019. URL: https://tel.archives-ouvertes.fr/tel-03120129/document (cited on page 127).

[65] *The RISC-V Instruction Set Manual*. Andrew Waterman and Krste Asanovi'c. 2019. URL: https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf (cited on page 101).

[66] Dhananjaya Wijerathne, Zhaoying Li, Manupa Karunarathne, Anuj Pathania, and Tulika Mitra. "CASCADE: High Throughput Data Streaming via Decoupled Access-Execute CGRA". In: *ACM Trans. Embed. Comput. Syst.* 18.5s (Oct. 2019). ISSN: 1539-9087. DOI: 10.1145/3358177. URL: https://doi.org/10.1145/3358177 (cited on page 52).

[67] S. Abadal, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio. "OrthoNoC: A Broadcast-Oriented Dual-Plane Wireless Network-on-Chip Architecture". In: *IEEE Trans. on Parallel and Distributed Systems* 29.3 (Mar. 2018). ISSN: 1045-9219. DOI: 10.1109/TPDS.2017.2764901 (cited on page 127).

[68] Mahesh Balasubramanian, Shail Dave, Aviral Shrivastava, and Reiley Jeyapaul. "LASER: A hardware/software approach to accelerate complicated loops on CGRAs". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pages 1069–1074. DOI: 10.23919/DATE.2018.8342170 (cited on page 55).

[69] S. Alexander Chin and Jason H. Anderson. "An Architecture-Agnostic Integer Linear Programming Approach to CGRA Mapping". In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. San Francisco, California: Association for Computing Machinery, 2018. ISBN: 9781450357005. DOI: 10.1145/3195970.3195986. URL: https://doi.org/10.1145/3195970.3195986 (cited on page 54).

[70] Satyajit Das. "Architecture and Programming Model Support for Reconfigurable Accelerators in Multi-Core Embedded Systems". Theses. Université de Bretagne Sud, June 2018. URL: https://tel.archives-ouvertes.fr/tel-01989827 (cited on page 62).

[71] Satyajit Das, Kevin J. M. Martin, Philippe Coussy, and Davide Rossi. "A Heterogeneous Cluster with Reconfigurable Accelerator for Energy Efficient Near-Sensor Data Analytics". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2018, pages 1–5. DOI: 10.1109/ISCAS.2018.8351749 (cited on page 64).

[72] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. "RAMP: Resource-Aware Mapping for CGRAs". In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. San Francisco, California: Association for Computing Machinery, 2018. ISBN: 9781450357005. DOI: 10.1145/3195970.3196101. URL: https://doi.org/10.1145/3195970.3196101 (cited on pages 54, 55).

[73] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. "URECA: Unified register file for CGRAs". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pages 1081–1086. DOI: 10.23919/DATE.2018.8342172 (cited on pages 53, 56).

[74] Jiangyuan Gu, Shouyi Yin, Leibo liu, and Shaojun Wei. "Stress-Aware Loops Mapping on CGRAs with Dynamic Multi-Map Reconfiguration". In: *IEEE Transactions on Parallel and Distributed Systems* 29.9 (2018), pages 2105–2120. DOI: 10.1109/TPDS.2018.2816955 (cited on page 54).

[75] Jörg Henkel, Jürgen Teich, Stefan Wildermann, and Hussam Amrouch. "Dynamic Resource Management for Heterogeneous Many-Cores". In: *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*. 2018, pages 1–6 (cited on page 82).

[76] Manupa Karunaratne, Cheng Tan, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. "Dnestmap: Mapping Deeply-Nested Loops on Ultra-Low Power CGRAs". In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. San Francisco, California: Association for Computing Machinery, 2018. ISBN: 9781450357005. DOI: 10.1145/3195970.3196027. URL: https://doi.org/10.1145/3195970.3196027 (cited on page 54).

[77] Stefan Mach, Davide Rossi, Giuseppe Tagliavini, Andrea Marongiu, and Luca Benini. “A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing”. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018, pages 1–5. DOI: 10.1109/ISCAS.2018.8351816 (cited on page 72).

[78] Hugo Miomandre, Julien Hascoët, Karol Desnos, Kevin J. M. Martin, Benoît Dupont de Dinechin, and Jean-François Nezan. “Embedded Runtime for Reconfigurable Dataflow Graphs on Manycore Architectures”. In: *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*. PARMA-DITAM ’18. New York, NY, USA: ACM, 2018, pages 51–56. ISBN: 978-1-4503-6444-7. DOI: 10.1145/3183767.3183780. (Visited on 06/11/2018) (cited on pages 83, 90).

[79] H. K. Mondal, R. C. Cataldo, C. A. Missio Marcon, K. Martin, S. Deb, and J. Diguet. “Broadcast- and Power-Aware Wireless NoC for Barrier Synchronization in Parallel Computing”. In: *2018 31st IEEE International System-on-Chip Conference (SOCC)*. Sept. 2018, pages 1–6. DOI: 10.1109/SOCC.2018.8618541 (cited on page 127).

[80] Hung K. Nguyen, Khoi P. Dong, and Xuan-Tu Tran. “A Reconfigurable Multi-function DMA Controller for High-Performance Computing Systems”. In: *2018 5th NAFOSTED Conference on Information and Computer Science (NICS)*. 2018, pages 344–349. DOI: 10.1109/NICS.2018.8606841 (cited on page 129).

[81] Giuseppe Tagliavini, Stefan Mach, Davide Rossi, Andrea Marongiu, and Luca Benini. “A transprecision floating-point platform for ultra-low power computing”. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pages 1051–1056. DOI: 10.23919/DATE.2018.8342167 (cited on page 72).

[82] Dani Voitsechov, Oron Port, and Yoav Etsion. “Inter-Thread Communication in Multithreaded, Reconfigurable Coarse-Grain Arrays”. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, pages 42–54. DOI: 10.1109/MICRO.2018.00013 (cited on page 49).

[83] Zhongyuan Zhao, Yantao Liu, Weiguang Sheng, Tushar Krishna, Qin Wang, and Zhigang Mao. “Optimizing the data placement and transformation for multi-bank CGRA computing system”. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pages 1087–1092. DOI: 10.23919/DATE.2018.8342173 (cited on page 56).

[84] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. “Compute Caches”. In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, pages 481–492. DOI: 10.1109/HPCA.2017.21 (cited on page 129).

[85] Adnan Bouakaz, Pascal Fradet, and Alain Girault. “A Survey of Parametric Dataflow Models of Computation”. In: *ACM Transactions on Design Automation of Electronic Systems* (Jan. 2017). URL: https://hal.inria.fr/hal-01417126 (cited on page 79).

[86] S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini. “Efficient mapping of CDFG onto coarse-grained reconfigurable array architectures”. In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2017, pages 127–132. DOI: 10.1109/ASPDAC.2017.7858308 (cited on pages 55, 64).

[87] Satyajit Das, Davide Rossi, Kevin J. M. Martin, Philippe Coussy, and Luca Benini. “A 142MOPS/mW integrated programmable array accelerator for smart visual processing”. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2017, pages 1–4. DOI: 10.1109/ISCAS.2017.8050238 (cited on page 63).

[88] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. “Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP.99 (2017), pages 1–14. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2017.2654506 (cited on pages 63, 71).

[89] J. Hascoët, K. Desnos, J.-F. Nezan, and B. D. de Dinechin. “Hierarchical Dataflow Model for Efficient Programming of Clustered Manycore Processors”. In: *Application-specific Systems, Architectures and Processors (ASAP)*. 2017 (cited on page 91).

[90] T. Kim, Z. Sun, H. Chen, H. Wang, and S. X. -. Tan. “Energy and Lifetime Optimizations for Dark Silicon Manycore Microprocessor Considering Both Hard and Soft Errors”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.9 (2017), pages 2561–2574. DOI: 10.1109/TVLSI.2017.2707401 (cited on page 104).

[91] Ganghee Lee, Ediz Cetin, and Oliver Diessel. “Fault Recovery Time Analysis for Coarse-Grained Reconfigurable Architectures”. In: *ACM Trans. Embed. Comput. Syst.* 17.2 (Nov. 2017). ISSN: 1539-9087. DOI: 10.1145/3140944. URL: https://doi.org/10.1145/3140944 (cited on page 57).

[92] I. Loi, A. Capotondi, D. Rossi, A. Marongiu, and L. Benini. “The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores”. In: *IEEE Transactions on Multi-Scale Computing Systems* PP.99 (2017), pages 1–1. DOI: 10.1109/TMSCS.2017.2769046 (cited on page 63).

[93] Thanh Dinh Ngo, Kevin J. M. Martin, and Jean-Philippe Diguet. “Move Based Algorithm for Runtime Mapping of Dataflow Actors on Heterogeneous MPSoCs”. en. In: *Journal of Signal Processing Systems* (2017), pages 1–18. ISSN: 1939-8018, 1939-8115. DOI: 10.1007/s11265-015-1088-z. (Visited on 12/02/2015) (cited on pages 83, 92).

[94] Chris Nicol. “A coarse grain reconfigurable array (cgra) for statically scheduled data flow computing”. In: *Wave Computing White Paper* (2017) (cited on page 126).

[95] David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Edited by Elsevier Science & Technology. Morgan Kaufmann, 2017 (cited on pages 102, 128).

[96] Jens Rettkowski and Diana Göhringer. “Data Stream Processing in Networks-on-Chip”. In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2017, pages 633–638. DOI: 10.1109/ISVLSI.2017.125 (cited on pages 129, 130).

[97] S. Skalistis and A. Simalatsar. “Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees”. In: *Proc. of the IEEE Design, Automation Test in Europe Conf. Exhibition (DATE)*. 2017, pages 752–757 (cited on page 82).

[98] Kanishkan Vadivel, Mark Wijtvliet, Roel Jordans, and Henk Corporaal. “Loop Overhead Reduction Techniques for Coarse Grained Reconfigurable Architectures”. In: *2017 Euromicro Conference on Digital System Design (DSD)*. 2017, pages 14–21. DOI: 10.1109/DSD.2017.83 (cited on page 55).

[99] Shouyi Yin, Xianqing Yao, Tianyi Lu, Dajiang Liu, Jiangyuan Gu, Leibo Liu, and Shaojun Wei. "Conflict-Free Loop Mapping for Coarse-Grained Reconfigurable Architecture with Multi-Bank Memory". In: *IEEE Transactions on Parallel and Distributed Systems* 28.9 (2017), pages 2471–2485. DOI: 10.1109/TPDS.2017.2682241 (cited on page 56).

[100] Satyajit Das, Thomas Peyret, Kevin J. M. Martin, Gwenolé Corre, Mathieu Thevenin, and Philippe Coussy. "A Scalable Design Approach to Efficiently Map Applications on CGRAs". In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. July 2016, pages 655–660. DOI: 10.1109/ISVLSI.2016.54 (cited on pages 53, 54, 56, 126).

[101] Y. Lesparre, A. Munier-Kordon, and J. Delosme. "Evaluation of Synchronous Dataflow Graph Mappings onto Distributed Memory Architectures". In: *Proc. of Euromicro Conf. on Digital System Design (DSD)*. Aug. 2016, pages 146–153 (cited on page 82).

[102] L. Liu, J. Wang, J. Zhu, C. Deng, S. Yin, and S. Wei. "TLIA: Efficient Reconfigurable Architecture for Control-Intensive Kernels with Triggered-Long-Instructions". In: *IEEE Transactions on Parallel and Distributed Systems* 27.7 (July 2016), pages 2143–2154. ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2477841 (cited on page 55).

[103] Kevin J. M. Martin, Mostafa Rizk, Martha Johanna Sepulveda, and Jean-Philippe Diguet. "Notifying Memories: A Case-Study on Data-Flow Applications with NoC Interfaces Implementation". In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC '16. Austin, Texas: Association for Computing Machinery, 2016, 35:1–35:6. ISBN: 9781450342360. DOI: 10.1145/2897937.2898051. URL: https://doi.org/10.1145/2897937.2898051 (visited on 07/11/2016) (cited on page 109).

[104] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. "Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs". In: *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 35.1 (2016), pages 72–85 (cited on page 82).

[105] M. Wijtvliet, L. Waeijen, and H. Corporaal. "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification". In: *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. 2016, pages 235–244. DOI: 10.1109/SAMOS.2016.7818353 (cited on page 50).

[106] M. Kerboeuf, P. Vallejo, and J. P. Babau. *Formal framework of recontextualization by means of dependency graphs*. Technical report. Université Bretagne Occidentale, 2015 (cited on page 85).

[107] Tahir Maqsood, Sabeen Ali, Saif U.R. Malik, and Sajjad A. Madani. "Dynamic task mapping for Network-on-Chip based systems". In: *J. of Systems Architecture* 61.7 (2015), pages 293–306. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2015.06.001 (cited on page 82).

[108] Dinh Thanh Ngo. "Runtime mapping of dynamic dataflow applications on heterogeneous multiprocessor platforms". Theses. Université de Bretagne Sud, June 2015. URL: https://hal.archives-ouvertes.fr/tel-01167316 (cited on page 85).

[109] Yaset Oliva, Emmanuel Casseau, Kevin Martin, Jean-Philippe Diguet, Thanh Dinh Ngo, and Yvan Eustache. "COMPA backend : Runtime dynamique pour l'exécution de programmes flot de données sur plates-formes multiprocesseurs". In: *COMPAS 2015 : - Conférence d'informatique en Parallélisme, Architecture et Système*. Lille, France, June 2015, pages 1–9. URL: https://hal.archives-ouvertes.fr/hal-01167037 (cited on pages 87, 88).

[110] Thomas Peyret, Mathieu Thevenin, Gwenole Corre, Philippe Coussy, and Kevin Martin. "Procédé et dispositif de tolérance aux fautes sur des composants électroniques". FR1460633 (France). Mar. 2015. URL: https://hal.archives-ouvertes.fr/hal-01166874 (visited on 10/13/2015) (cited on page 58).

[111] Thomas Peyret, Mathieu Thevenin, Gwenole Corre, Kevin Martin, and Philippe Coussy. "Procédé et dispositif d'architecture configurable à gros grains pour exécuter l'intégralité d'un code". FR1460631 (France). Mar. 2015. URL: https://hal.archives-ouvertes.fr/hal-01166875 (visited on 10/13/2015) (cited on page 60).

[112] Wei Quan and Andy D. Pimentel. "A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs". In: *ACM Trans. on Embedded Computing Systems (TECS)* 14.1 (Jan. 2015), 14:1–14:25. ISSN: 1539-9087. DOI: 10.1145/2680542 (cited on pages 83, 95).

[113] P. Vallejo, M. Kerboeuf, K. J. M. Martin, and J. P. Babau. "Improving reuse by means of asymmetrical model migrations: An application to the Orcc case study". In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Sept. 2015, pages 358–367. DOI: 10.1109/MODELS.2015.7338267 (cited on page 85).

[114] Paola Vallejo. "Réutilisation de composants logiciels pour l'outillage de DSML dans le contexte des MPSoC". Theses. Université de Bretagne occidentale - Brest, Dec. 2015. URL: https://hal.univ-brest.fr/tel-01260937 (cited on page 85).

[115] Shouyi Yin, Dajiang Liu, Leibo Liu, Shaojun Wei, and Yike Guo. "Joint affine transformation and loop pipelining for mapping nested loop on CGRAs". In: *DATE*. Mar. 2015, pages 115–120 (cited on page 54).

[116] Liang Chen and Tulika Mitra. "Graph Minor Approach for Application Mapping on CGRAs". In: *ACM Trans. Reconfigurable Technol. Syst.* 7.3 (Sept. 2014). ISSN: 1936-7406. DOI: 10.1145/2655242. URL: https://doi.org/10.1145/2655242 (cited on pages 53, 54).

[117] K. Desnos. "Memory Study and Dataflow Representations for Rapid Prototyping of Signal Processing Applications on MPSoCs". Theses. INSA de Rennes, Sept. 2014 (cited on page 80).

[118] M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen. "Adjustable contiguity of run-time task allocation in networked many-core systems". In: *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC)*. 2014, pages 349–354 (cited on page 82).

[119] Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. "Branch-Aware Loop Mapping on CGRAs". In: *Proceedings of the 51st Annual Design Automation Conference*. DAC '14. San Francisco, CA, USA: Association for Computing Machinery, 2014, pages 1–6. ISBN: 9781450327305. DOI: 10.1145/2593069.2593100. URL: https://doi.org/10.1145/2593069.2593100 (cited on pages 55, 59).

[120] J. Heulot, M. Pelcat, K. Desnos, J. F. Nezan, and S. Aridhi. "Spider: A Synchronous Parameterized and Interfaced Dataflow-based RTOS for multicore DSPS". In: *Embedded Design in Education and Research Conference (EDERC)*. Sept. 2014, pages 167–171 (cited on pages 80, 81, 90).

[121] Seunghun Jin, Woong Seo, Yeon-Gon Cho, and Soojung Ryu. “Low-power reconfigurable audio processor for mobile devices”. In: *2014 IEEE International Conference on Consumer Electronics (ICCE)*. 2014, pages 369–370. DOI: 10.1109/ICCE.2014.6776045 (cited on page 125).

[122] Thanh Dinh Ngo, Daniel Sepulveda, Kevin J. M. Martin, and Jean-Philippe Diguet. “Communication-model based embedded mapping of dataflow actors on heterogeneous MPSoC”. In: *2014 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Oct. 2014, pages 1–8. DOI: 10.1109/DASIP.2014.7115629 (cited on pages 83, 86).

[123] Yaset Oliva, Emmanuel Casseau, Kevin Martin, Pierre Bomel, Jean-Philippe Diguet, Hervé Yviquel, Mickael Raulet, Erwan Raffin, and Laurent Morin. *Orcc’s Compa-Backend demonstration*. en. Oct. 2014. URL: https://hal.inria.fr/hal-01059858/document (visited on 02/27/2015) (cited on pages 87, 88).

[124] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J. Nezan, and S. Aridhi. “Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming”. In: *European Embedded Design in Education and Research Conference (EDERC)*. 2014, pages 36–40. DOI: 10.1109/EDERC.2014.6924354 (cited on pages 80, 84).

[125] Thomas Peyret. “Architecture matérielle et flot de programmation associé pour la conception de systèmes numériques tolérants aux fautes”. Theses. Université de Bretagne Sud, Dec. 2014. URL: https://tel.archives-ouvertes.fr/tel-01217584 (cited on pages 57, 58, 60).

[126] Thomas Peyret, Gwenolé Corre, Mathieu Thevenin, Kevin J. M. Martin, and Philippe Coussy. “Efficient application mapping on CGRAs based on backward simultaneous scheduling/binding and dynamic graph transformations”. In: *ASAP*. June 2014, pages 169–172. DOI: 10.1109/ASAP.2014.6868652 (cited on page 54).

[127] Thomas Peyret, Gwenolé Corre, Mathieu Thevenin, Kevin Martin, and Philippe Coussy. “Efficient application mapping on CGRAs based on backward simultaneous scheduling/binding and dynamic graph transformations”. In: *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. June 2014, pages 169–172. DOI: 10.1109/ASAP.2014.6868652 (cited on page 66).

[128] Davide Rossi, Igor Loi, Germain Haugou, and Luca Benini. “Ultra-low-latency lightweight DMA for tightly coupled multi-core clusters”. In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM. 2014, page 15 (cited on page 64).

[129] John E Savage. “Models of computation”. In: *Early Years* 4.1.1 (2014), page 2 (cited on page 78).

[130] Simon Schulz, Oliver Bringmann, Thomas Schweizer, and Wolfgang Rosenstiel. “Rotated parallel mapping: A novel approach for mapping data parallel applications on CGRAs”. In: *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. 2014, pages 1–6. DOI: 10.1109/ReConFig.2014.7032554 (cited on pages 54, 56).

[131] Shuvra S. Bhattacharyya, Ed F. Deprettere, and Bart D. Theelen. “Dynamic Dataflow Graphs”. In: *Handbook of Signal Processing Systems*. Edited by Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala. New York, NY: Springer New York, 2013, pages 905–944. ISBN: 978-1-4614-6859-2. DOI: 10.1007/978-1-4614-6859-2_28. URL: https://doi.org/10.1007/978-1-4614-6859-2_28 (cited on page 79).

[132] J. Castrillon, R. Leupers, and G. Ascheid. “MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs”. In: *Industrial Informatics, IEEE Transactions on* 9.1 (Feb. 2013), pages 527–545. ISSN: 1551-3203 (cited on page 83).

[133] Francesco Conti, Andrea Marongiu, and Luca Benini. “Synthesis-friendly Techniques for Tightly-coupled Integration of Hardware Accelerators into Shared-memory Multi-core Clusters”. In: *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS ’13. Montreal, Quebec, Canada: IEEE Press, 2013, 5:1–5:10. ISBN: 978-1-4799-1417-3. URL: http://dl.acm.org/citation.cfm?id=2555692.2555697 (cited on page 64).

[134] K. Desnos, M. Pelcat, J.-F. Nezan, S.S. Bhattacharyya, and S. Aridhi. “PiMM: Parameterized and Interfaced Dataflow Meta-Model for MPSoCs Runtime Reconfiguration”. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE. 2013, pages 41–48 (cited on page 79).

[135] S. Foroutan, Y. Thonnart, and F. Petrot. “An Iterative Computational Technique for Performance Evaluation of Networks-on-Chip”. In: *Computers, IEEE Transactions on* 62.8 (Aug. 2013), pages 1641–1655. ISSN: 0018-9340. DOI: 10.1109/TC.2012.85 (cited on page 85).

[136] Soonhoi Ha and Hyunok Oh. “Decidable Dataflow Models for Signal Processing: Synchronous Dataflow and Its Extensions”. In: *Handbook of Signal Processing Systems*. Edited by Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala. New York, NY: Springer New York, 2013, pages 1083–1109. ISBN: 978-1-4614-6859-2. DOI: 10.1007/978-1-4614-6859-2_33. URL: https://doi.org/10.1007/978-1-4614-6859-2_33 (cited on page 79).

[137] Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. “REGIMap: register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)”. In: *DAC*. ACM, 2013, 18:1–18:10. ISBN: 978-1-4503-2071-9. DOI: 10.1145/2463209.2488756 (cited on pages 54, 56).

[138] Chanhee Lee, Sungchan Kim, and Soonhoi Ha. “Efficient run-time resource management of a manycore accelerator for stream-based applications”. In: *Embedded Systems for Real-time Multimedia (ESTIMedia), 2013 IEEE 11th Symposium on*. Oct. 2013, pages 51–60 (cited on page 83).

[139] Dajiang Liu, Shouyi Yin, Leibo Liu, and Shaojun Wei. “Polyhedral model based mapping optimization of loop nests for CGRAs”. In: *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE. 2013, pages 1–8 (cited on page 59).

[140] Tony Nowatzki, Michael Sartin-Tarm, Lorenzo De Carli, Karthikeyan Sankaralingam, Cristian Estan, and Behnam Robatmili. “A General Constraint-Centric Scheduling Framework for Spatial Architectures”. In: *SIGPLAN Not.* 48.6 (June 2013), pages 495–506. ISSN: 0362-1340. DOI: 10.1145/2499370.2462163. URL: https://doi.org/10.1145/2499370.2462163 (cited on page 54).

[141] Wei Quan and Andy D. Pimentel. “A Scenario-Based Run-Time Task Mapping Algorithm for MPSoCs”. In: *Proc. of the Annual Design Automation Conf. (DAC)*. Austin, Texas, 2013 (cited on pages 82, 83).

[142] Zoltán Endre Rákossy, Masayuki Hiromoto, Hiroshi Tsutsui, Takashi Sato, Yukihiro Nakamura, and Hiroyuki Ochi. "Hot-swapping architecture with back-biased testing for mitigation of permanent faults in functional unit array". In: *2013 Design, Automation and Test in Europe Conference (DATE)*. 2013, pages 535–540. DOI: 10.7873/DATE.2013.120 (cited on page 58).

[143] Pradip Kumar Sahu and Santanu Chattopadhyay. "A survey on application mapping strategies for Network-on-Chip design". In: *J. of Systems Architecture* 59.1 (2013), pages 60–76. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2012.10.004 (cited on page 82).

[144] Michael Scott. *Shared-Memory Synchronization*. Edited by Mark D. Hill. Volume 8. Morgan & Claypool, June 2013, pages 1–221. DOI: 10.2200/S00499ED1V01Y201304CAC023 (cited on page 102).

[145] A.K. Singh, M. Shafique, A. Kumar, and J. Henkel. "Mapping on multi/many-core systems: Survey of current and emerging trends". In: *Proc. of the Design Automation Conf. (DAC)*. May 2013, pages 1–10 (cited on page 82).

[146] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. "Accelerating Throughput-Aware Runtime Mapping for Heterogeneous MPSoCs". In: *ACM Trans. Des. Autom. Electron. Syst.* 18.1 (Jan. 2013) (cited on pages 82, 83).

[147] Hervé Yviquel, Emmanuel Casseau, Mickaël Raulet, Pekka Jääskeläinen, and Jarmo Takala. "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms". In: *Proc. of the Int. Symposium on Image and Signal Processing and Analysis (ISPA)*. 2013, pages 732–737 (cited on pages 82, 83, 86, 103).

[148] Li Zhou, Dongpei Liu, Min Tang, and Hengzhu Liu. "Mapping Loops onto Coarse-Grained Reconfigurable Array Using Genetic Algorithm". In: *Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BICTA)*. Edited by Zhixiang Yin, Zhixiang Yin, and Xianwen Fang. 2013. DOI: 10.1007/978-3-642-37502-6_95. URL: https://doi.org/10.1007/978-3-642-37502-6_95 (cited on page 54).

[149] Jeronimo Castrillon, Andreas Tretter, Rainer Leupers, and Gerd Ascheid. "Communication-aware Mapping of KPN Applications Onto Heterogeneous MPSoCs". In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. San Francisco, California: ACM, 2012, pages 1266–1271. ISBN: 978-1-4503-1199-1 (cited on page 83).

[150] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, and Glenn Reinman. "Charm: A composable heterogeneous accelerator-rich microprocessor". In: *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM. 2012, pages 379–384 (cited on page 59).

[151] Michel Dubois, Murali Annavaram, and Per Stenstrm. *Parallel Computer Organization and Design*. USA: Cambridge University Press, 2012. ISBN: 0521886759 (cited on pages 102, 119).

[152] Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. "EPIMap: using epimorphism to map applications on CGRAs". In: *DAC*. ACM, 2012, pages 1284–1291. ISBN: 978-1-4503-1199-1. DOI: 10.1145/2228360.2228600 (cited on pages 53, 54, 56, 61, 64).

[153] K. Han, S. Park, and K. Choi. "State-based full predication for low power coarse-grained reconfigurable architecture". In: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2012. DOI: 10.1109/DATE.2012.6176704 (cited on page 55).

[154] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 5th edition. Amsterdam: Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8 (cited on pages 75, 76, 100).

[155] Jing Lin, Andreas Gerstlauer, and Brian L. Evans. "Communication-aware Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems". en. In: *Journal of Signal Processing Systems* 69.3 (Dec. 2012), pages 279–291. ISSN: 1939-8018, 1939-8115. DOI: 10.1007/s11265-012-0674-6. (Visited on 10/22/2013) (cited on page 83).

[156] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. "Scenario-based Design Flow for Mapping Streaming Applications Onto On-chip Many-core Systems". In: *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES '12. Tampere, Finland: ACM, 2012. ISBN: 978-1-4503-1424-4. DOI: 10.1145/2380403.2380422 (cited on page 83).

[157] Matthieu Wipliez and Mickaël Raulet. "Classification of Dataflow Actors with Satisfiability and Abstract Interpretation". In: *Int. J. of Embedded and Real-Time Communication Systems (IJERTCS)* 3.1 (2012), pages 49–69 (cited on page 107).

[158] Shuvra S. Bhattacharyya, Johan Eker, Jörn W. Janneck, Christophe Lucarz, Marco Mattavelli, and Mickaël Raulet. "Overview of the MPEG Reconfigurable Video Coding Framework". In: *J. Signal Process. Syst.* 63.2 (May 2011), pages 251–263. ISSN: 1939-8018. DOI: 10.1007/s11265-009-0399-3. URL: http://dx.doi.org/10.1007/s11265-009-0399-3 (cited on page 80).

[159] N. Binkert et al. "The Gem5 Simulator". In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pages 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718 (cited on page 116).

[160] T. E. Carlson, W. Heirman, and L. Eeckhout. "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2011, pages 1–12. DOI: 10.1145/2063384.2063454 (cited on page 104).

[161] Kiyoung Choi. "Coarse-Grained Reconfigurable Array: Architecture and Application Mapping". In: *IPSJ Transactions on System LSI Design Methodology* 4 (2011), pages 31–46. DOI: 10.2197/ipsjtsldm.4.31 (cited on pages 50, 52).

[162] Yongjoo Kim, Jongeun Lee, Aviral Shrivastava, and Yunheung Paek. "Memory Access Optimization in Compilation for Coarse-grained Reconfigurable Architectures". In: *ACM Trans. Des. Autom. Electron. Syst.* 16.4 (Oct. 2011), 42:1–42:27. ISSN: 1084-4309. DOI: 10.1145/2003695.2003702. URL: http://doi.acm.org/10.1145/2003695.2003702 (visited on 11/04/2016) (cited on page 56).

[163] Tushar Krishna, Li-Shiuan Peh, Bradford M. Beckmann, and Steven K. Reinhardt. "Towards the Ideal On-chip Fabric for 1-to-many and Many-to-1 Communication". In: *IEEE/ACM International Symposium on Microarchitecture (44th MICRO)*. Porto Alegre, Brazil: ACM, 2011, pages 71–82. ISBN: 978-1-4503-1053-6. DOI: 10.1145/2155620.2155630 (cited on page 127).

[164] Ganghee Lee, Kiyoung Choi, and N.D. Dutt. "Mapping Multi-Domain Applications Onto Coarse-Grained Reconfigurable Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.5 (May 2011), pages 637–650. ISSN: 0278-0070. DOI: 10.1109/TCAD.2010.2098571 (cited on pages 54, 61).

[165] Nobuaki Ozaki, Y Yoshihiro, Yoshiki Saito, Daisuke Ikebuchi, Masayuki Kimura, Hideharu Amano, Hiroshi Nakamura, Kimiyoshi Usami, Mitaro Namiki, and Masaaki Kondo. "Cool Mega-Array: A highly energy efficient reconfigurable accelerator". In: *Field-Programmable Technology (FPT), 2011 International Conference on*. Dec. 2011, pages 1–8. DOI: 10.1109/FPT.2011.6132668 (cited on page 59).

[166] Abbas Rahimi, Igor Loi, Mohammad Reza Kakoee, and Luca Benini. "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters". In: *2011 Design, Automation & Test in Europe*. IEEE. 2011, pages 1–6 (cited on page 63).

[167] S. Stuijk, M. Geilen, B. Theelen, and T. Basten. "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications". In: *Int. Conf on Embedded Computer Systems (SAMOS)*. 2011 (cited on page 83).

[168] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration". In: *IET Computers & Digital Techniques* 5 (2 Mar. 2011), 123–135(12) (cited on page 82).

[169] João M. P. Cardoso, Pedro C. Diniz, and Markus Weinhardt. "Compiling for Reconfigurable Computing: A Survey". In: *ACM Comput. Surv.* 42.4 (June 2010). ISSN: 0360-0300. DOI: 10.1145/1749603.1749604. URL: https://doi.org/10.1145/1749603.1749604 (cited on pages 50, 52).

[170] C. Chou and R. Marculescu. "Run-Time Task Allocation Considering User Behavior in Embedded Multiprocessor Networks-on-Chip". In: *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 29.1 (2010), pages 78–91 (cited on page 82).

[171] Bjorn De Sutter, Praveen Raghavan, and Andy Lambrechts. "Coarse-Grained Reconfigurable Array Architectures". In: *Handbook of Signal Processing Systems*. Edited by Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala. Boston, MA: Springer US, 2010, pages 449–484. ISBN: 978-1-4419-6345-1. DOI: 10.1007/978-1-4419-6345-1_17. URL: https://doi.org/10.1007/978-1-4419-6345-1%5C_17 (cited on page 50).

[172] K. Han, J. K. Paek, and K. Choi. "Acceleration of control flow on CGRA using advanced predicated execution". In: *Field-Programmable Technology (FPT), 2010 International Conference on*. Dec. 2010. DOI: 10.1109/FPT.2010.5681452 (cited on page 55).

[173] E. Raffin, C. Wolinski, F. Charot, K. Kuchcinski, S. Guyetant, S. Chevobbe, and E. Casseau. "Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture". In: *DASIP*. 2010, pages 168–175. DOI: 10.1109/DASIP.2010.5706261 (cited on page 54).

[174] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes. "Dynamic Task Mapping for MPSoCs". In: *IEEE Design Test of Computers* 27.5 (2010), pages 26–35 (cited on page 82).

[175] Yoshiki Saito, Toru Sano, Masaru Kato, Vasutan Tunbunheng, Yoshihiro Yasuda, Masayuki Kimura, and Hideharu Amano. "MuCCRA-3: a low power dynamically reconfigurable processor array". In: *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. IEEE Press. 2010, pages 377–378 (cited on page 59).

[176] Muhammad Ali Shami and Ahmed Hemani. "Control Scheme for a CGRA". In: *2010 22nd International Symposium on Computer Architecture and High Performance Computing*. 2010, pages 17–24. DOI: 10.1109/SBAC-PAD.2010.12 (cited on page 52).

[177] Matthieu Wipliez. "Compilation infrastructure for dataflow programs". PhD thesis. INSA of Rennes / IETR, 2010 (cited on pages 80, 103).

[178] Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. "A survey of multicore processors". In: *IEEE Signal Processing Magazine* 26.6 (2009), pages 26–37. DOI: 10.1109/MSP.2009.934110 (cited on pages 45, 76, 104).

[179] Stephen Friedman, Allan Carroll, Brian Van Essen, Benjamin Ylvisaker, Carl Ebeling, and Scott Hauck. "SPR: An Architecture-adaptive CGRA Mapping Tool". In: *FPGA*. ACM, 2009, pages 191–200. ISBN: 978-1-60558-410-2. DOI: 10.1145/1508128.1508158 (cited on page 54).

[180] Maxime Pelcat, Jean François Nezan, Jonathan Piat, Jerome Croizer, and Slaheddine Aridhi. "A System-Level Architecture Model for Rapid Prototyping of Heterogeneous Multicore Embedded Systems". In: *Proc. of the Conf. on Design and Architectures for Signal and Image Processing (DASIP)*. Nice, France, Sept. 2009, 8 pages. URL: https://hal.archives-ouvertes.fr/hal-00429397 (cited on page 84).

[181] Jonghee W. Yoon, Aviral Shrivastava, Sanghyun Park, Minwook Ahn, and Yunheung Paek. "A Graph Drawing Based Spatial Mapping Algorithm for Coarse-Grained Reconfigurable Architectures". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.11 (2009), pages 1565–1578. DOI: 10.1109/TVLSI.2008.2001746 (cited on pages 53–55).

[182] C. Bienia, S. Kumar, J. Singh, and K. Li. *The PARSEC Benchmark Suite: Characterization and Architectural Implications*. Technical report. Princeton University, Jan. 2008 (cited on page 116).

[183] Kyungwook Chang and K. Choi. "Mapping control intensive kernels onto coarse-grained reconfigurable array architecture". In: *2008 International SoC Design Conference*. Volume 01. Nov. 2008, pages I-362-I–365. DOI: 10.1109/SOCDC.2008.4815647 (cited on pages 55, 59).

[184] Bjorn De Sutter, Paul Coene, Tom Vander Aa, and Bingfeng Mei. "Placement-and-Routing-Based Register Allocation for Coarse-Grained Reconfigurable Arrays". In: *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '08. Tucson, AZ, USA: Association for Computing Machinery, 2008, pages 151–160. ISBN: 9781605581040. DOI: 10.1145/1375657.1375678. URL: https://doi.org/10.1145/1375657.1375678 (cited on pages 53, 55, 56).

[185] Bjorn De Sutter, Paul Coene, Tom Vander Aa, and Bingfeng Mei. "Placement-and-routing-based Register Allocation for Coarse-grained Reconfigurable Arrays". In: *SIGPLAN Not.* 43.7 (June 2008), pages 151–160. ISSN: 0362-1340 (cited on pages 53, 54).

[186] Hyunchul Park, Kevin Fan, Scott A. Mahlke, Taewook Oh, Heeseok Kim, and Hong-seok Kim. "Edge-centric Modulo Scheduling for Coarse-grained Reconfigurable Architectures". In: *PACT*. ACM, 2008. ISBN: 978-1-60558-282-5. DOI: 10.1145/1454115.1454140 (cited on pages 54, 55).

[187] Frank Bouwens, Mladen Berekovic, Andreas Kanstein, and Georgi Gaydadjiev. “Architectural Exploration of the ADRES Coarse-grained Reconfigurable Array”. In: *Proceedings of the 3rd International Conference on Reconfigurable Computing: Architectures, Tools and Applications*. ARC’07. Mangaratiba, Brazil, 2007. ISBN: 978-3-540-71430-9. URL: http://dl.acm.org/citation.cfm?id=1764631.1764633 (cited on pages 51, 59).

[188] Gregory W Donohoe, David M Buehler, K Joseph Hass, William Walker, and Pen-Shu Yeh. “Field Programmable Processor Array: Reconfigurable Computing for Space”. In: *2007 IEEE Aerospace Conference*. IEEE. 2007, pages 1–6 (cited on page 59).

[189] Akira Hatanaka and Nader Bagherzadeh. “A Modulo Scheduling Algorithm for a Coarse-Grain Reconfigurable Array Template”. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007, pages 1–8. DOI: 10.1109/IPDPS.2007.370371 (cited on pages 54, 55).

[190] G. Theodoridis, D. Soudris, and S. Vassiliadis. “A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools”. In: *Fine- and Coarse-Grain Reconfigurable Computing*. Edited by Stamatis Vassiliadis and Dimitrios Soudris. Dordrecht: Springer Netherlands, 2007, pages 89–149. ISBN: 978-1-4020-6505-7. DOI: 10.1007/978-1-4020-6505-7_2. URL: https://doi.org/10.1007/978-1-4020-6505-7%5C_2 (cited on pages 50–52).

[191] Daniel P. Bovet and Marco Cesati. “Understanding the Linux kernel”. In: *Understanding the Linux kernel*. 3rd. O’Reilly, 2006. Chapter 10, page 295 (cited on page 106).

[192] Janina A Brenner, JC Van Der Veen, Sándor P Fekete, J Oliveira Filho, and Wolfgang Rosenstiel. “Optimal Simultaneous Scheduling, Binding and Routing for Processor-Like Reconfigurable Architectures”. In: *FPL*. Aug. 2006, pages 1–6. DOI: 10.1109/FPL.2006.311262 (cited on page 54).

[193] Young-Sin Cho, Eun-Ju Choi, and Kyoung-Rok Cho. “Modeling and Analysis of the System Bus Latency on the SoC Platform”. In: *Proceedings of the 2006 International Workshop on System-level Interconnect Prediction*. SLIP ’06. Munich, Germany: ACM, 2006, pages 67–74. ISBN: 1-59593-255-0. DOI: 10.1145/1117278.1117293 (cited on page 85).

[194] A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A.J.M. Moonen, M.J.G. Bekooij, B.D. Theelen, and M.R. Mousavi. “Throughput Analysis of Synchronous Data Flow Graphs”. In: *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*. June 2006, pages 25–36. DOI: 10.1109/ACSD.2006.33 (cited on page 79).

[195] S. Stuijk, M. Geilen, and T. Basten. “Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs”. In: *43rd ACM/IEEE DAC*. 2006, pages 899–904. DOI: 10.1109/DAC.2006.229409 (cited on page 83).

[196] Wayne Wolf. *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN: 012369485X (cited on page 76).

[197] Gregory Donohoe. “Reconfigurable data path processor”. 6,883,084. US Patent 6,883,084. Apr. 2005 (cited on page 59).

[198] Yoonjin Kim, Mary Kiemb, Chulsoo Park, Jinyong Jung, and Kiyoung Choi. “Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization”. In: *Design, Automation and Test in Europe*. IEEE. 2005, pages 12–17 (cited on page 59).

[199] Nikhil Bansal, Sumit Gupta, Nikil Dutt, and Alexandru Nicolau. “Analysis of the Performance of Coarse-Grain Reconfigurable Architectures with Different Processing Element Configurations”. In: *Workshop on Application Specific Processors, held in conjunction with the International Symposium on Microarchitecture (MICRO), 2003*. 2003 (cited on page 54).

[200] Johan Eker and Jörn W. Janneck. *CAL Language Report: Specification of the CAL Actor Language*. Technical report. University of California, Berkeley, 2003 (cited on page 80).

[201] P. M. Heysters and G. J. M. Smit. “Mapping of DSP algorithms on the MONTIUM architecture”. In: *Proceedings International Parallel and Distributed Processing Symposium*. 2003. DOI: 10.1109/IPDPS.2003.1213333 (cited on page 50).

[202] Jong-eun Lee, Kiyoung Choi, and N. D. Dutt. “Compilation approach for coarse-grained reconfigurable architectures”. In: *IEEE Design Test of Computers* 20.1 (2003), pages 26–33. DOI: 10.1109/MDT.2003.1173050 (cited on pages 50, 54).

[203] Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. “Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling”. In: *2003 Design, Automation and Test in Europe Conference and Exhibition*. 2003, pages 296–301. DOI: 10.1109/DATE.2003.1253623 (cited on page 55).

[204] M. L. Anido, A. Paar, and N. Bagherzadeh. “Improving the operation autonomy of SIMD processing elements by using guarded instructions and pseudo branches”. In: *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*. 2002, pages 148–155. DOI: 10.1109/DSD.2002.1115363 (cited on pages 55, 59).

[205] T. Grotker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Springer, 2002 (cited on page 95).

[206] Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. “DRESC: a retargetable compiler for coarse-grained reconfigurable architectures”. In: *FPT*. Dec. 2002, pages 166–173. DOI: 10.1109/FPT.2002.1188678 (cited on pages 54, 55).

[207] K. Bondalapati. “Parallelizing DSP nested loops on reconfigurable architectures using data context switching”. In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. 2001, pages 273–276. DOI: 10.1145/378239.378483 (cited on page 50).

[208] R. Hartenstein. “A Decade of Reconfigurable Computing: A Visionary Retrospective”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE ’01. Munich, Germany: IEEE Press, 2001, pages 642–649. ISBN: 0769509932 (cited on pages 49, 50).

[209] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matthew Moe, and R Reed Taylor. “PipeRench: A reconfigurable architecture and compiler”. In: *Computer* 33.4 (2000), pages 70–77 (cited on page 59).

[210] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J Kurdahi, Nader Bagherzadeh, and Eliseu M Chaves Filho. “MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications”. In: *IEEE Transactions on Computers* 49.5 (May 2000), pages 465–481. ISSN: 0018-9340. DOI: 10.1109/12.859540 (cited on page 59).

[211] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, and Brad Hutchings. “A reconfigurable arithmetic array for multimedia applications”. In: *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*. ACM. 1999, pages 135–143 (cited on page 59).

[212] Kiran Bondalapati and Viktor K. Prasanna. “Mapping loops onto reconfigurable architectures”. In: *Field-Programmable Logic and Applications From FPGAs to Computing Paradigm*. Edited by Reiner W. Hartenstein and Andres Keevallik. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pages 268–277. ISBN: 978-3-540-68066-6 (cited on pages 50, 54, 55).

[213] George Karypis and Vipin Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM J. Sci. Comput.* 20.1 (Dec. 1998). ISSN: 1064-8275 (cited on page 81).

[214] George Karypis and Vipin Kumar. “Multilevel Algorithms for Multi-constraint Graph Partitioning”. In: *ACM/IEEE Conf. on Supercomputing*. Washington, DC, USA, 1998 (cited on page 81).

[215] Ethan Mirsky, Andre DeHon, et al. “MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources.” In: *FCCM*. Volume 96. 1996, pages 17–19 (cited on page 59).

[216] Richard Sites. “It’s the Memory, Stupid!” In: *Architects Look to Processors of Future*. Microdesign resources, 1996. URL: https://www.ardent-tool.com/CPU/docs/MPR/101006.pdf (cited on page 106).

[217] Erich Gamma et al. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2 (cited on page 106).

[218] E.A. Lee and T.M. Parks. “Dataflow process networks”. In: *Proceedings of the IEEE* 83.5 (1995), pages 773–801. DOI: 10.1109/5.381846 (cited on page 79).

[219] Wm. A. Wulf and Sally A. McKee. “Hitting the Memory Wall: Implications of the Obvious”. In: *SIGARCH Comput. Archit. News* 23.1 (Mar. 1995), pages 20–24. ISSN: 0163-5964. DOI: 10.1145/216585.216588. URL: http://doi.acm.org/10.1145/216585.216588 (cited on page 106).

[220] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455 (cited on pages 53, 81).

[221] Pierre G Paulin and John P Knight. “Force-directed scheduling for the behavioral synthesis of ASICs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8.6 (1989), pages 661–679 (cited on page 65).

[222] Arvind and Robert A. Iannucci. “Two Fundamental Issues in Multiprocessing”. In: *4th International DFVLR Seminar on Foundations of Engineering Sciences on Parallel Computing in Science and Engineering*. Bonn, Germany: Springer-Verlag, 1988, pages 61–88. ISBN: 0387189238 (cited on page 99).

[223] E.A. Lee and D.G. Messerschmitt. “Synchronous data flow”. In: *Proceedings of the IEEE* 75.9 (Sept. 1987), pages 1235–1245. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13876 (cited on page 79).

[224] C. M. Fiduccia and R. M. Mattheyses. “A Linear-Time Heuristic for Improving Network Partitions”. In: *Proceedings of the 19th Design Automation Conference*. DAC ’82. IEEE Press, 1982, pages 175–181. ISBN: 0897910206 (cited on page 94).

[225] Gilles Kahn. “The Semantics of a Simple Language for Parallel Programming”. In: *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*. Edited by Jack L. Rosenfeld. North-Holland, 1974, pages 471–475 (cited on page 79).

[226] Giorgio Levi. “A note on the derivation of maximal common subgraphs of two directed or undirected graphs”. en. In: *CALCOLO* 9.4 (Dec. 1973), pages 341–352. ISSN: 0008-0624, 1126-5434. DOI: 10.1007/BF02575586 (cited on page 66).

A. Selected publication on CGRA

This appendix provides the author version of the paper "An Energy-Efficient Integrated Programmable Array Accelerator and Compilation flow for Near-Sensor Ultra-low Power Processing", as a complement to the work presented in chapter 4.

Publication details

Title	*An Energy-Efficient Integrated Programmable Array Accelerator and Compilation flow for Near-Sensor Ultra-low Power Processing*
Authors	Satyajit DAS, Kevin J. M. MARTIN, Davide ROSSI, Philippe COUSSY and Luca BENINI
In	IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems vol. 38, no. 6, pp. 1095-1108, June 2019
DOI	10.1109/TCAD.2018.2834397
URL	`https://hal.archives-ouvertes.fr/hal-01828604`

An Energy-Efficient Integrated Programmable Array Accelerator and Compilation flow for Near-Sensor Ultra-low Power Processing

Satyajit Das*†, Kevin J. M. Martin*, Davide Rossi†, Philippe Coussy*, and Luca Benini†‡

*Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France, [firstname].[lastname]@univ-ubs.fr
†Department of Electrical, Electronic and Information Engineering, University of Bologna, Italy, [firstname].[lastname]@unibo.it
‡Integrated Systems Laboratory, ETH Zurich, Switzerland, [first-initial][last name]@iis.ee.ethz.ch

***Abstract*—In this paper we give a fresh look to Coarse Grained Reconfigurable Arrays (CGRAs) as ultra-low power accelerators for near-sensor processing. We present a general-purpose Integrated Programmable-Array accelerator (IPA) exploiting a novel architecture, execution model, and compilation flow for application mapping that can handle kernels containing complex control flow, without the significant energy overhead incurred by state of the art predication approaches. To optimize the performance and energy efficiency, we explore the IPA architecture with special focus on shared memory access, with the help of the flexible compilation flow presented in this paper. We achieve a maximum energy gain of 2×, and performance gain of 1.33× and 1.8× compared with state of the art partial and full predication techniques, respectively. The proposed accelerator achieves an average energy efficiency of 1617 MOPS/mW operating at 100MHz, 0.6V in 28nm UTBB FD-SOI technology, over a wide range of near-sensor processing kernels, leading to an improvement up to 18×, with an average of 9.23× (as well as a speed-up up to 20.3×, with an average of 9.7×) compared to a core specialized for ultra-low power near-sensor processing.**

***Index Terms*—CGRA, compilation, control flow, CDFG, ultra-low power accelerator, computer architecture**

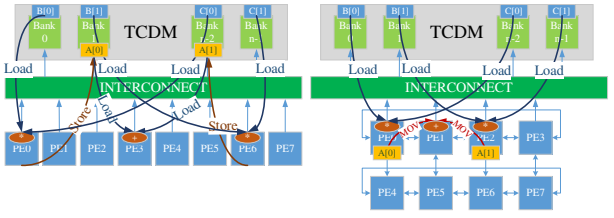
I. Introduction

DUE to the increasing complexity of near-sensor data analytics algorithms, low power embedded applications such as Wireless Sensor Networks (WSN), Internet of Things (IoT), and wearable sensors combine the requirement of high performance and extreme energy efficiency in a mW power envelope [2]. While traditional ultra-low power sensor processing circuits rely on hardwired Application Specific Integrated Circuit (ASIC) architectures [12], near-threshold parallel computing is emerging as a promising solution [47]. Even though this approach provides maximum flexibility, a dominating majority of the power consumed during processing is linked to the typical overheads of instruction processors [16], such as instruction fetching and decoding, control and data-path pipeline overheads (up to 40%), load/store overhead (up to 30%). In this work, we make significant step forward in parallel near-threshold computing toward the goal of achieving the energy efficiency of application-specific data-paths, by exploiting the Coarse Grain Reconfigurable Array (CGRA) architectural template, and revisiting it to fit within an ultra-low power (mW) power envelope.

CGRAs have been intensely investigated in the past for applications with power consumption profiles ranging from mobile (hundreds of mW) [11] to high performance (hundreds of W) [34]. In this paper, we focus on a CGRA architecture in the mW range (and below). Very few CGRA architectures have been pushed in this ultra-low power mission profile [36] [48] [13]. Our CGRA is designed to work as an accelerator of an ultra-low power PULP processor cluster [47], sharing L1 memory with the processors. Hence, another major challenge in this context is achieving efficient L1 memory sharing [47]. To reduce memory access contention, it is necessary to have enough banks in the shared memory. On the other hand, the number of ports into the multi-banked shared-L1 memory logarithmic interconnect must be tightly constrained to avoid significant area and power overheads [44].

To cope with the ultra-low power profile and memory sharing challenges we build upon the Integrate Programmable-Array accelerator (IPA) concept proposed in [10] involving a multi-bank Tightly Coupled Data Memory (TCDM) coupled with a flexible and configurable memory hierarchy for data storage. As shown in Fig. 1, from an architectural viewpoint, point-to-point data communication between processing elements (PEs) during kernel execution, represents a key advantage over energy-hungry data sharing over shared memory that is required when using a traditional processor-cluster architecture for parallel processing. The IPA cluster performs a lower number of memory operations on the sample program presented in the Fig. 1(c), which in turn gives and energy improvement of 1.3× over the clustered multi-core architecture, which performs data sharing through the TCDM. In this comparison, we even ignore the barrier synchronization overheads in the many-core cluster for the sake of simplicity.

The IPA approach allows to significantly reduce the pressure on L1 memory, and hence the complexity of the interconnect between the PEs and the memory banks, since it requires a smaller number of banks to achieve low contention [47]. On the other hand, as opposed to clustered multi-core architectures, where data-exchange among cores is managed through




```
for(i=0; i<1; i++)
{
A[i] = B[i] * C[i]
}
for(i=0; i<1; i++)
{
sum = sum + A[i];
}
```

(c)

Average energy consumption in pJ/operation					
	Load-Store operations	Arithmetic operations	MOV operation		
	4.2	3.4	3.1		
	Total #LS	Total #Arithmetic	Total #MOV	Energy (pJ)	Gain
Many-Core	8	3	0	43.8	-
IPA	4	3	2	33.2	1.3x

(d)

Figure 1: (a) Cluster of processors; (b) Cluster with IPA accelerator (c) Sample program running in both clusters; (d) Energy consumption comparison between the two clusters

shared data structures and OpenMP parallel constructs, in CGRAs the compiler must take care of data-exchange among PEs, exploiting as much as possible point-to-point connections among PEs to minimize accesses to the shared memory.

Another major compiler challenge towards achieving high energy efficiency in CGRAs is the management of loop-carried data dependencies and control dependencies. State of the art compilers [37] [19] [40] [6] for CGRAs deal only with the straight-line code sequence (basic block) of the innermost loop of a kernel. In case of conditional present in the innermost loop, the compilers use predication [45] techniques to convert the control flows into data flow structures. Indeed, these compilers can only generate code to execute a single loop, as a set of pipelined stages is repeatedly executed up to a certain number ($loop\ boundary\ /\ number\ of\ pipelined\ stages$) specified by the compiler. In case of nested loops, only the innermost loop is accelerated using a CGRA, leaving the outer loops for the host processor. However, this approach requires several offloads by the host, which implies additional memory-mapped I/O operations for synchronization and communication with the CGRA. Hence, it causes significant overhead, especially when the innermost loop has a very small number of iterations, which is a typical scenario for near-sensor processing applications [50].

On the other hand, large CGRA architectures for high performance computing have frequently resorted to predication techniques to expose parallelism across control dependencies, such as conditionals [45] [5]. Unfortunately, predication leads to waste of resources and it is hard to justify in an extremely power and area constrained scenario [52]. In this paper, we address the above challenges by proposing a novel compilation flow tailored for our ultra-low power IPA architecture. This flow enables the execution of multiple loops and conditionals starting from ANSI C code, relying on the energy efficient *register allocation approach* presented in [8].

In a nutshell, this paper contributes to the two critical aspects of energy-efficient application mapping onto CGRAs. First, we carry out an architectural exploration, based on the IPA proposed in [10], for optimizing performance and energy efficiency. The IPA features full support for conditional operations, exploits the internal registers of the PEs for intermediate data exchange and relies on a multi-bank TCDM only for accesses to input/output buffers, significantly improving energy efficiency. Second, we describe a complete compilation flow to map kernels with multiple loop nests and conditionals onto the IPA. The flow helps releasing the host processor from performing the computation of the outer loops, significantly improving performance of the IPA. It also achieves high energy efficiency by minimizing the number of memory operations exploiting the features of the architecture.

To quantify the efficiency of IPA architecture and compilation flow, we compare the performance and energy consumption with the state of the art predication methods running on the IPA. Experimental results on a benchmark set of control intensive kernels show that the register allocation approach achieves a maximum of 1.33× (with minimum of 1.04× and average of 1.13×) and 1.8× (with minimum of 1.37× and average of 1.59×) performance gain compared to partial predication and full predication techniques. For what concerns shared-L1 memory access optimization, our exploration shows that a banking factor of 0.5 (i.e. 8 LSUs, 4 TCDM banks) provides the optimal configuration in terms of performance and energy for a IPA configuration with 4x4 PEs. Moreover, the IPA features a very regular control and data-path structure, which is suitable for fine-grained power management. We exploit this architectural regularity to design a fine-grained clock gating mechanism, which turns into an average 2× energy efficiency boost with respect to the non-clock-gated IPA. Results show that the IPA achieves a maximum speed up of 20.3×, with an average of 9.7× compared to one or1k processor [17][1], with an area ratio of just 1.6×. The average energy efficiency achieved by the IPA operating at 0.6V is 1617 MOPS/mW, which is up to 18× and on average 9.23× better than what is achieved by the processor.

The rest of this paper is organized as follows. In Section II, the background and related work are discussed. In section III, the target architecture, memory hierarchy and the execution model are described. Section IV focuses on presenting the full compilation flow, with the support of required definitions, and models. Section V presents the implementation and experimental results. Finally, the paper is concluded in Section VI.

II. Background and Related Work

Much research has been done to evaluate the performance, power, and cost of CGRAs [11]. In this paper, we focus on the energy efficiency aspects of both architecture and compiler.

A. Architecture

While targeting low power execution, data and context management is of utmost importance. Integration of CGRAs as accelerators with the data and instruction memory has seen several solutions over the past years [11].

In many low-power targeted CGRAs [3][39][48][23], memory operations are managed by the host processor. Among

[1]This processor is optimized for low power execution in the context of near threshold near-sensor processing

these architectures, Ultra-Low-Power Samsung Reconfigurable Processor (ULP-SRP) and Cool Mega Array (CMA) operate in ultra-low-power (up to 3 mW) range. In these architectures, PEs can only access the data once prearranged in the shared register file by the processor. For an energy efficient implementation, the main challenge for these designs is to balance the performance of the data distribution managed by the CPU, and the computation in the PE array. However, in several cases, the computational performance of the PE array is compromised by the CPU, due to large synchronization overheads. For example, in ADRES [3] the power overhead of the VLIW processor used to handle the data memory access is up to 20%. In CMA [39] the host CPU feeds the data into the PEs through a shared fetch register (FR) file. This is very inefficient in terms of flexibility. The key feature of this architecture is the possibility to apply independent DVFS [51] or body biasing [36] to balance array and controlling processor parameters to adjust performance and bandwidth requirements of the applications. The highest reported energy efficiency for CMA is 743 MOPS/mW on 8-bit kernels, not considering the overhead of the controlling processor, which is not reported. With respect to this work, which only deals with DFG described with a customized language, we target 32-bit data and application kernels described in C language, which are mapped onto the array using an end-to-end C-to-CGRA compilation flow.

In architectures such as, MorphoSys [49], RSPA[24], SmartCell [29], PipeRench [18], SIMD-CGRA [15], load-store operations are managed explicitly by the PEs. Data elements in these architectures are stored in a shared memory with one memory port per PE row. The main disadvantages of such data access architecture are: (a) lots of contention between the PEs on the same row to access the memory banks, and (b) expensive data exchange between rows through complex interconnect networks within the array. With respect to these architectures, our approach minimizes contention by exploiting a multi-banked shared memory with word-level interleaving. In this way data-exchange among tiles can be performed either through the much simpler point-to-point communication infrastructure or fully flexible shared TCDM.

NASA's Reconfigurable Data-Path Processor (RDPP) [13], and Field Programmable Processor Array (FPPA) [14] are targeted for low-power stream data processing for spacecrafts. These architectures rely on control switching [13] of data streams, and synchronous data flow computational model avoiding investment on memories and control. On the contrary, the IPA is tailored to achieve energy-efficient near sensor processing of data with workloads very different from the stream data processing.

Table I summarizes an overview of the jobs managed by CGRA and the host processor for different architectural approaches. Acceleration of the kernels involves memory operations, innermost loop computation, outer loop computation, offload and synchronization with the CPU. As shown in the table, IPA manages to execute both the innermost and outer loops, and the memory operations of a kernel imposing least communication and memory operation overhead while synchronizing with the CPU execution.

With respect to these state of the art reconfigurable arrays and array of processors, this paper introduces a highly energy efficient, general-purpose IPA accelerator where PEs have random access to the local memory, and execute full control and data flow of kernels on the array starting from a generic ANSI C representation of applications [8]. This paper also focuses on the architectural exploration of the proposed IPA accelerator [10], with the goal to determine the optimal configuration of number of LSUs and number of banks for the shared L1 memory. Moreover, we employ a fine-grained power management architecture to eliminate dynamic power consumption of idle tiles during kernels execution which provides 2× improvement of energy efficiency, on average. The globally synchronized execution model, low cost but full-flexible programmability, tightly coupled data memory organization, and fine-grained power management architecture define the suitability of the proposed architecture as an accelerator for ultra-low power embedded computing platforms.

B. Compilation

To map the loops, state of the art compilers for CGRA mostly rely on software pipelining [19] [37] [40]. This approach can manage to map the innermost loop body in a pipelined manner. On the other hand, for the outer loops, CPU must initiate each iteration in the CGRA, which causes significant overhead in the synchronization between the CGRA and CPU execution. Liu et al in [31] pinpointed this issue and proposed to map maximum of two levels of loops using polyhedral transformation on the loops. However, this approach is not generic as it cannot scale to an arbitrary number of loops. Some approaches [30] [27] use loop unrolling for the kernels. The basic assumption for these implementations is that the innermost loops trip count is not large. Hence, the solutions end up doing partial unroll of the innermost loops. The outer loops remain to be executed by the host processor. As most of the proposed compilers handle innermost loop of

Table I: Comparison between different architectural approaches

References	[3][39][48] [14][13] [38][35]	[31]	[49][24] [18][7]	IPA This paper
Memory ops	CPU	CGRA	CPU	CGRA
Innermost loop	CGRA	CGRA	CGRA	CGRA
Outer loop	CPU	CPU	CGRA	CGRA
Offload + Sync	CPU	CPU	CPU	CPU
Overhead				

Table II: Comparison between different approaches to manage control flow in CGRA

Techniques	**Conditionals**		**Loops**	
	Balanced	**Imbalanced**	**Single**	**Nested**
Partial predication [5]	√	√	×	×
Full predication [1]	√	√	×	×
State based full predication [21]	√	√	×	×
Dual issue single execution [20]	√	×	×	×
TLIA [32]	√	√	√	×
Software pipelining [37]	×	×	√	×
Loop unrolling [27]	×	×	√	NA
Register allocation [8]	√	√	√	√

the kernels, they mostly bank upon the partial predication [5] and full predication [1] techniques to map the conditionals inside the loop body.

Partial predication maps instructions of both if-part and else-part on different PEs. If both the if-part and the else-part update the same variable, the result is computed by selecting the output from the path that must have been executed based on the evaluation of the branch condition. This technique increases the utilization of the PEs, at the cost of higher energy consumption due to execution of both paths in a conditional. Unlike partial predication, in full predication all instructions are predicated. Instructions on each path of a control flow, which are sequentially configured onto PEs, will be executed if the predicate value of the instruction is similar with the flag in the PEs. Hence, the instructions in the false path do not get executed. The sequential arrangement of the paths degrades the latency and energy efficiency of this technique.

Full predication is upgraded in state based full predication [21]. This scheme prevents the wasted instruction issues from false conditional path by introducing sleep and awake mechanisms, but fails to improve performance. Dual issue scheme [20] targets energy efficiency by issuing two instructions to a PE simultaneously, one from the if-path, another from the else-path. In this mechanism, the latency remains similar to that of the partial predication with improved energy efficiency. However, this approach is too restrictive, as far as imbalanced and nested conditionals are concerned. To map nested, imbalanced conditionals and single loop onto CGRA, the triggered long instruction set architecture (TLIA) is presented in [32]. This approach merges all the conditionals present in kernels into triggered instructions, and creates instruction pool for each triggered instruction. As the depth of the nested conditionals increases the performance of this approach decreases. As far as the loop nests are concerned, the TLIA approach reaches bottleneck to accommodate the large set of triggered instructions into the limited set of PEs.

The compilation flow we propose, uses the register allocation approach [8] to map CDFGs onto the CGRA. This allows to map both loops and conditionals of any depth. In our case, the only limitation in the mapping of kernels onto the CGRA is given by the size of instruction memory of the PEs, and not by the structure of the application (i.e. number of loops, and branches). Also, one can increase the size of code segment to be executed in the CGRA as much as possible, minimizing the control and synchronization overheads with the core, which is not negligible in the other approaches. Table II presents a comprehensive comparison between several techniques to manage control flow in the kernels. Software pipelining and loop unrolling are mostly used for the mapping of the innermost loop, while branches inside the loop are managed by one of the described predication techniques. Hence, the existing compilers use combined solutions for branches and innermost loop mapping. This requires exhaustive exploration to find out the most suitable combination for the target architecture and application domain. On the contrary, our proposed compilation flow can handle both conditionals and loops efficiently.

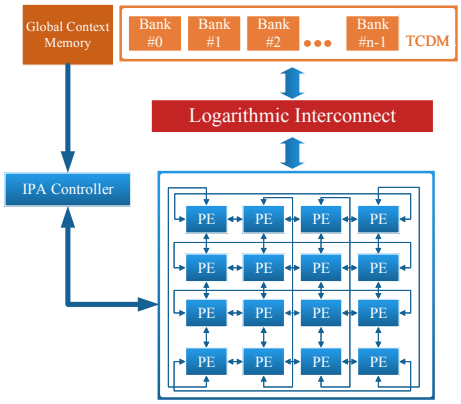


Figure 2: Integrated Programmable-Array Accelerator

III. IPA Architecture and Execution Model

In this section, we present the Integrated Programmable-Array Accelerator (IPA) architecture, supporting standalone execution of complete control and data flow applications.

A. Integrated Programmable-Array Accelerator (IPA)

IPA is the integration of a PE Array (PEA) and a tightly coupled data memory (TCDM) through a low-latency logarithmic interconnect. An IPA controller loads the context into the PEs from a pre-loaded Global Context Memory (GCM). Fig. 2 shows the organization of the IPA.

The PEA consists of a parametric number of PEs connected with a 2-dimensional tours network. The PE Array follows the multiple instruction, multiple data (MIMD) model of computation. All PEs operate on different set of instructions. A bus based interconnect network is implemented to load instructions and constants (i.e. context) from the GCM into the PEs, whereas the torus network is used during execution phase for low power data communication between the PEs. The details of the load context protocol are discussed in [10]. Targeting low power execution, the instruction set architecture [10] was designed from scratch resulting 20-bit long instruction. We took the advantage of the visibility of the micro-architecture to the compiler and moved the immediate data to constant register file in the PEs (discussed later) which eases the compression of the instruction, imposing low pressure on the decoder.

Fig. 3 describes the components of a PE. The Load Store Unit (LSU) is optional for the PEs (the optimal number of LSU is a parameter studied in this paper). Two operands (IN0 and IN1) define the inputs of each PE. The input sources are the neighbouring PEs and the register file. A 32-bit ALU and a 16-bit $\times$ 16-bit $\rightarrow$ 32-bit multiplier are employed in this block. The Constant Register File (CRF) stores the immediate values of the instructions, while the Regular Register File (RRF) and Output Register (OPR) store the temporary variables. The Controller fetches the instructions from the Instruction Register File (IRF). If the decoded instruction is a *jump*, the target address of the *jump* is stored in the Jump Register (JR). The *cjump* (conditional jump) instruction contains two target addresses. The true path is evaluated in the JR by the Boolean "OR" of the Condition Register (CR) bits of the PEs.

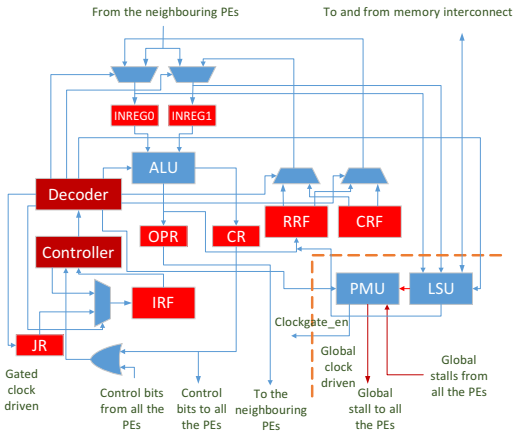


Figure 3: Components of the PE

The TCDM acts as L1 memory for the IPA. Featuring a number of ports equal to the number of memory banks, the TCDM provides concurrent access to different memory locations. The TCDM is interfaced with the LSUs of the PE array through a low latency, logarithmic interconnect [44], implementing a word level interleaving scheme to minimize access contention.

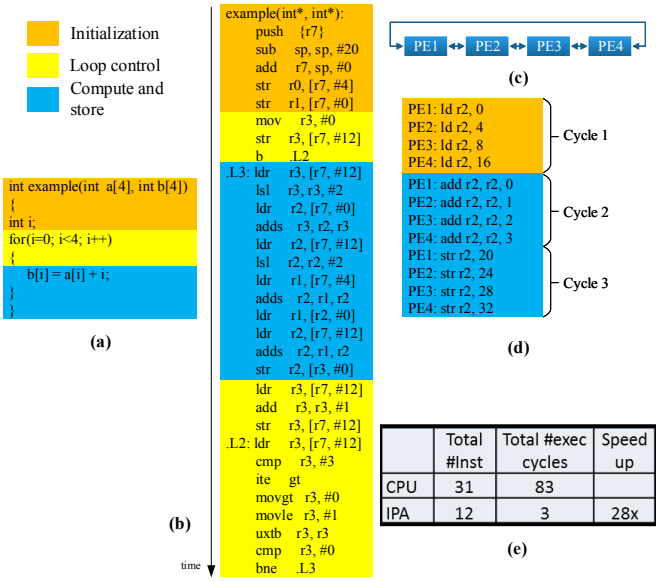
B. Power Management Unit (PMU)

To reduce dynamic power consumption in idle mode, each PE contains a tiny Power Management Unit (PMU) which clock gates the PEs when idle. An idle condition for a PE arises from three situations: (i) Unused PE: when a PE is not used during mapping; (ii) Load Store stall: In case of TCDM banking conflict the PMU generates a *global stall*, which is broadcast to all the PEs. Until the global stall is resolved, all the PEs are clock gated by their corresponding PMUs. LSUs are placed in the global clock region (Fig. 3) to avoid deadlocks; (iii) Multiple NOP operations: a NOP instruction contains the number of successive NOPs. When a NOP instruction is fetched, the decoder loads this number into a counter within the PMU. The *clockgate_en* remains low until the count reaches zero. The counter gets halted when it encounters a global stall and resumes the count after the stall is resolved, synchronizing the execution flow among PEs.

Due to the fine-grained nature of the power management, more aggressive power gating is not implemented, since it imposes large area penalty without remarkable benefits. The leakage power of each tile is so small that it does not change notably the energy efficiency when rest of the system is active.

C. Overview of the execution model

After compiling a kernel (see section IV), the compiler generates the assembly and the addresses for the input and output data in the local shared memory. The assembler takes the assembly and the Instruction Set Architecture (ISA) of the IPA, to generate the context (i.e. the program to be stored into the IRF) for each PE, which is pre-loaded in the GCM. The context contains instructions and constants for each PE in the array. Prior to the execution start, the context is loaded into the



	Total #Inst	Total #exec cycles	Speed up
CPU	31	83	
IPA	12	3	28x

Figure 4: (a) Sample program (b) Execution in CPU (c) Example PEA (d) Execution in IPA (e) Execution metrics in CPU and IPA

corresponding IRF and CRF of the PEs. We assume that the code fits in the local memory. Larger execution contexts can be handled using the IPA controller and overlays. Details on this process are omitted for the sake of conciseness[2]. In each cycle, the PEs fetch 20-bit instruction from the local IRF. The immediate data are shifted to constant register file which eases the compression of the instruction. Hence, the pressure on the decoder is quite low.

Fig. 4 shows the execution of a sample program in a traditional CPU and the IPA. The total number of instructions for the sample program in the CPU and the IPA are 31 and 12 respectively. Also, the IPA achieves 28× performance gain compared to that of the CPU while executing the sample program. The decrease in the number of instructions in the IPA in this specific example is mainly due to the much lower number of memory operations and the fact that the small loop can be completely unrolled without code size blown-up.

IV. Compilation Flow

The compilation generates a mapping of the program for the corresponding PEA. This section presents the models adopted for the architecture and the application and the full compilation flow to map control and data flow onto the PEA. We also discuss the register allocation approach to exploit the register files of the PEs while preserving control-carried dependencies.

A. Architecture, application model and homomorphism

The compiler takes two inputs. The first is the PEA model, and the second is the ANSI-C code of the application. The PEA is modelled by a bipartite directed graph with two types of nodes: operators and registers. Timing is implicitly represented by connections between registers and operators, which

[2]Note that the context loading and setup cost are accounted for in the experimental results.

is referred to as the *time extended model* of the PEA [19]. Two types of operator nodes are defined for the PEAs. The first type is the computing operator (functional unit (FU) nodes in Fig. 5(a)) that represents the physical implementation of an arithmetic and logical operation (+, ×, -, OR, AND) and/or memory access (e.g. load/store). The second type of operator is the memorization operator (circular nodes in Fig. 5(b)). It is associated with the output register and represents the operation of keeping a value in a local register explicitly.

Fig. 5 (a) shows a sample PEA with two PEs connected by a torus network. Each PE has 3 registers in the distributed register file, and a single output register. Fig. 5 (b) represents the *time extended model* of the PEA shown in Fig. 5 (a). In this model, one can vary the interconnect network, the distribution and size of the register file, and the type of the PE, to explore different PEA architectures.

The application is modelled as a control and data flow graph (CDFG). Supporting control flow gives the opportunity to accelerate a kernel without any intervention of the host processor. A CDFG is depicted as $G = (V, E)$ where V is the set of basic blocks and $E \subseteq V \times V$ is the set of directed edges representing control flow. A Basic Block (BB) is represented as a data flow graph (DFG) or $BB = (D, O, A)$ where D is the set of data nodes, O is the set of operation nodes and A is the set of arcs representing dependencies. The control flow from one basic block to another is supported with jump (*jmp*) and conditional jump (*cjmp*) instructions.

Fig. 6 presents a sample program and the corresponding CDFG. In this figure, basic blocks are represented as blue rectangles. The flow from one basic block to another basic block is represented by black arrows and managed by simple branch (*jmp*) operation. The true and false paths of a conditional managed by *cjmp*, are shown by solid and dashed arrows respectively. The execution flow of the CDFG is presented as: $BB\_1 \rightarrow BB\_2 \rightarrow$ (either BB_3 or BB_8) if $BB_3 \rightarrow BB\_4$ $\rightarrow$ (either BB_5 or BB_6) $\rightarrow BB\_7 \rightarrow BB\_2 \cdots$. In order to maintain the execution flow, it is necessary to synchronize all the PEs in the array, to the execution of the same basic block. When the execution flow jumps from one basic block to another, all the PEs in the PEA must be synchronized to the current basic block execution. This allows to use all the PEs concurrently or sequentially, while executing a single basic block. Dually, several basic blocks can use the same PE. The synchronized execution allows the compiler to map several operations and data onto the same PE. Next, we present the homomorphism of the CDFG model with the application model, to support different stages in the compilation flow.

The basic blocks in the CDFG, presented in Fig. 5(c), are composed of data nodes, operation nodes, and data dependencies. Three equivalences between the basic block DFGs and PEA model nodes are defined: (1) data and registers; (2) computation and computing operators; (3) data dependences and connection between the time extended PE components. As the two models are homomorphic, the mapping of a DFG onto the PEA is therefore a problem equivalent to finding a DFG in the PEA graph.

Fig. 5(b) represents a possible mapping of the sample CDFG in Fig. 5(c) onto the PEA in Fig. 5(a) over 4 cycles. Following, we discuss the full compilation flow for CDFG mapping.

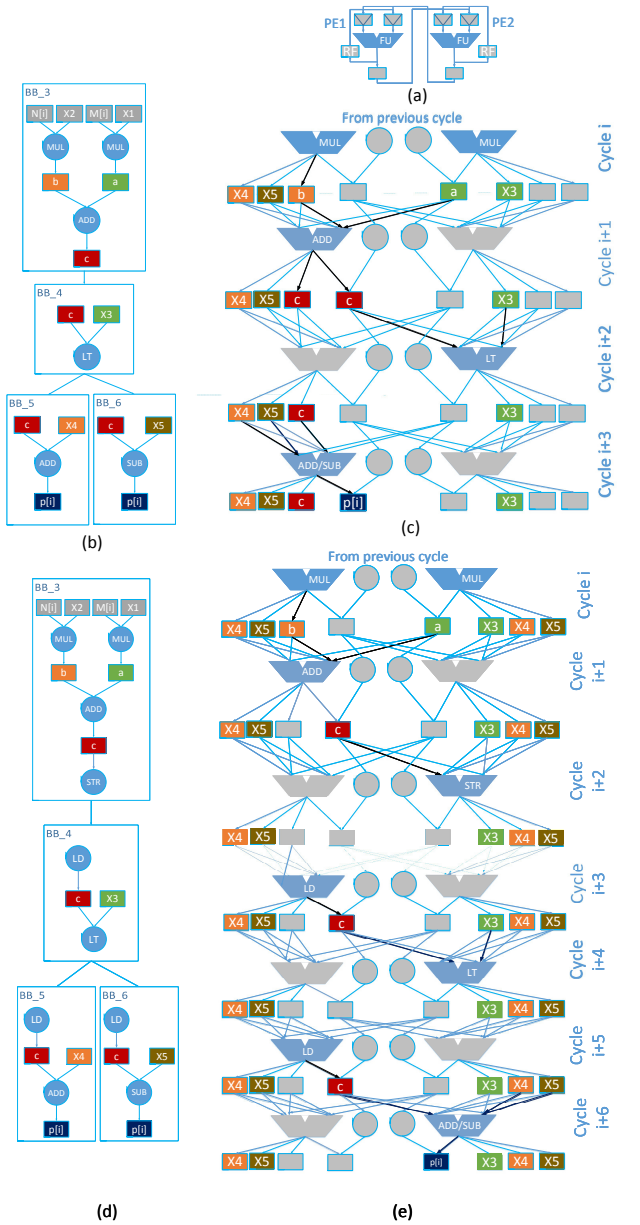


Figure 5: (a) A 2×1 PEA with 3 registers in RF and one output register (b) CDFG model (c) A possible mapping of (b) onto the PEA over 4 cycles using register allocation based approach. (d) The transformed CDFG of (b) for systematic load store based approach (e) A possible mapping of (d) onto the PEA over 7 cycles using systematic load store based approach

B. The compilation flow step by step

Fig. 7 shows a schematic representation of the compilation flow for mapping CDFGs onto the PEA. A CDFG mapping is a set of DFG mappings that are compatible with each other. To be compatible, the DFGs must access the data that remain in the PEs (see symbol variables (see definition IV.1)) in the same location. This is ensured by the register allocation approach.

To map the basic blocks, we rely on the highly scalable and efficient mapping approach for DFGs described in [9].

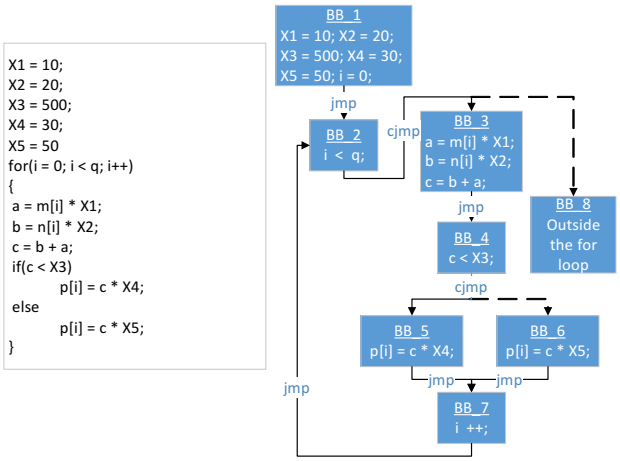


Figure 6: Sample program and corresponding CDFG

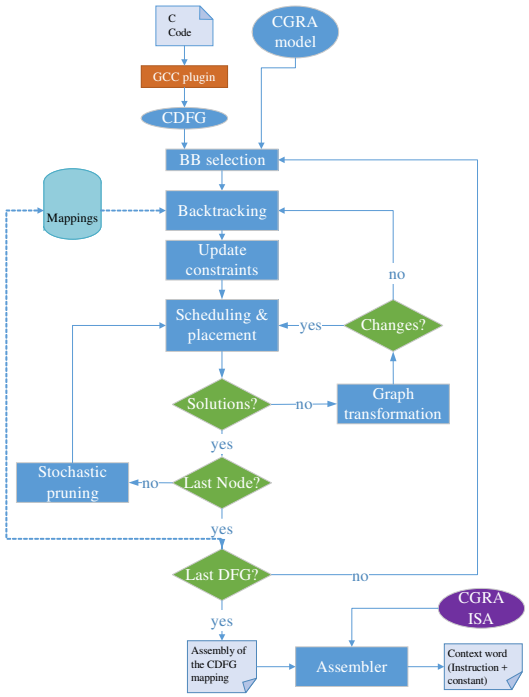


Figure 7: Compilation flow

The compilation flow proposed in this paper, extends the DFG mapping to accommodate the register allocation approach to map a full CDFG onto the PE array. As presented in Fig. 7, the full compilation flow is composed of six interdependent stages: BB selection, backtracking, update constraints, scheduling and placement, graph transformation and a stochastic pruning. These tasks are described in detail in the next sub-sections.

1) Scheduling and placement: The proposed approach uses a backward traversal [43] list scheduling algorithm to schedule the DFG of each basic block. It relies on a heuristic in which the schedulable operations are listed by priority order. In backward traversal, a node is schedulable if and only if all its children are already scheduled. The priority of nodes depends on their mobility [42]. It is possible to process memorization nodes and conventional nodes differently. Also, when several nodes have the same mobility, their respective number of

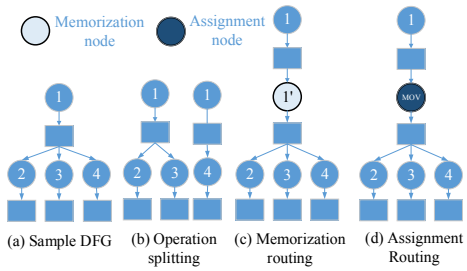


Figure 8: Graph transformation

successors is used as a second priority criterion. The higher is the number of successors, the higher the priority is. Indeed, a node with a higher number of successors is more difficult to map due to the routing constraint coming from the limited amount of connections between tiles. Thus, scheduling these nodes at first usually allows to decrease the application's latency [43]. As soon as the highest priority node has been defined, the compiler tries to find a placement in the PE array model. If a placement solution exists, the node is scheduled else the graph is transformed.

The proposed placement uses an incremental version of Levi's algorithm [28]. The proposed algorithm adds the newly scheduled operation node and its associated data node to the sub-graph composed of already scheduled and placed nodes. Only the previous set of solutions that have been kept, location constraints (RLC (see definition IV.3) and TLC (see definition IV.2)) are used to find every possibility to add this couple of nodes without considering the non-yet scheduled nodes. If no solution is found, there is absolutely no possibility to bind this couple in all the previous partial solutions because Levi's algorithm provides a complete exploration of the solution space. In that case, graph transformation is required.

2) Graph transformation: DFG is transformed dynamically when no binding solution is found. The three graph transformations are used in our compilation flow (Fig. 8).

- *Operation splitting* duplicates an operation node by keeping its same inputs and distributing output edges to reduce the number of successors of the original operation node.
- *Memorization routing* adds a memorization node and its associated data node to delay one operation and to keep data dependencies
- *Assignment routing* adds an assignment node (*mov* operation node) to increase the physical distance between the source and the sink of symbol variables by one. Due to TLC or RLC, when the physical distance between the source and sink of the symbol variable becomes more than one, the compiler dynamically adds one *mov* operation node to the DFG.

3) Stochastic pruning: The exactness of the placement approach leads to very large number of partial mappings. And it grows exponentially if not pruned. Hence, we use the stochastic pruning approach described in [9].

4) Basic block selection: Once all the nodes of the basic block have been scheduled and bound, the compiler selects one mapping among the several mappings generated, and selects the next basic block to be mapped. As discussed previously, the data integrity must be maintained over several basic block

mappings. The data mapping problem for CDFG mapping is now described before detailing the basic block selection step.

4.a: Definition and problem formulation

Data in an application is separated into two categories. (i) The standard input and output data (mostly the array inputs and outputs) are mapped as memory operands. The inputs and outputs are allotted by load-store operations. In our sample program in Fig. 6, m, n are the input arrays and p is the output array, which are managed by load and store operations. (ii) The internal variables of a program are mapped onto the registers of the processing elements, and managed by the register allocation based approach [8]. Following, we introduce several definitions concerning register allocation approach:

Definition IV.1. [Symbol Variables and location constraints] In compilation, the recurring variables (repeatedly written and read) are managed in local register files of the PEs to avoid multiple access of local memory. The recurring variables which have occurrences in multiple basic blocks need special attention since the integrity of these variables must be kept intact throughout the mapping process for different basic blocks. These variables are defined as *Symbol variables*. The register locations for the symbol variables are referred to as *location constraints*. For instance, variable c in the CDFG (Fig. 6) is written in BB_3, and read in BB_4, BB_5 and BB_6. The register location for c must be same for all the mappings of these basic blocks. Similarly, $X1$, $X2$, $X3$, $X4$, $X5$, i, a and b must be location constrained. In the rest of the paper, the locations for such *symbol variables* are denoted with an overline, as $\overline{variable\_name}$.

Depending on the order of the basic blocks mapped (i.e. traversing the CDFG), some location constrains may be reused in the mapping process or may be kept reserved for later use. These two types of location constraints are now detailed.

Definition IV.2. [Target Location Constraints (TLC)] We consider a scenario *scenario_1*, where BB_6 is mapped first, BB_3 is mapped next and so on. While mapping BB_6, variables c and $X5$ are placed at $\bar{c}$ and $\overline{X5}$. While mapping BB_3, $\bar{c}$ and $\overline{X5}$ which are already mapped in BB_6, must be considered because $\bar{c}$ will be used to map c in BB_3. In other words, the placement of the variables in the registers must be respected. Also, $\bar{a}$, $\bar{b}$, $\overline{X1}$ and $\overline{X2}$ must not reuse $\overline{X5}$. Otherwise, $X5$ will have wrong value when executing BB_6. Let's consider *scenario_2* with another order of basic blocks mapped, like first BB_3 and then BB_6 and so on. In this order of mapping, it is necessary to pass $\bar{c}$ and $\overline{X5}$ from BB_3 to BB_6 mapping. To keep c and $X5$ alive in BB_6 both $\bar{c}$ and $\overline{X5}$ must be used in mapping of BB_6. The placement or binding information which are passed from the mapping of one basic block to the mapping of the other basic block is referred to as *constraint* (e.g. *scenario_1*: $\bar{c}$ and $\overline{X5}$ passed from BB_6 to BB_3). The location constraints related to the data that are used within a basic block mapping phase (e.g. *scenario_1*: $\bar{c}$ in BB_3 mapping) are referred to as *target location constraints* (TLC).

Definition IV.3. [Reserved Location Constraints (RLC)] As we have seen in the previous examples, some of the location constraints must be reserved in the mapping of basic blocks for the sake of data integrity. To keep the symbol variables alive, it is necessary to exclude the memory elements from placement. Accordingly, these resources will not override while mapping the basic block (e.g. *scenario_1*: $\overline{X5}$ in BB_3 mapping). These are referred to as *reserved location constraints* (RLC).

4.b: Selection approach

If the number of RLC and TLC is high, mapping becomes complex. As TLC will force to use resources, and RLC will force to exclude resources from placement. Hence, the primary goal for our compiler is to minimize the number of TLCs and RLCs by choosing an efficient traversal of the CDFG.

The basic solution to deal with the symbol variables is to introduce memory operations. The symbol variables are stored in the memory where they are written and are loaded from the memory when read. In the rest of the paper this basic solution is referred to as *systematic load-store based approach*. This method is presented in the Fig. 5(d). For the symbol variable c in the CDFG shown in Fig. 5(c), it stores variable c in the memory in BB_3, and loads in BB_4, BB_5 and BB_6. Fig. 5 refers to the mapping of the transformed CDFG in this approach. This basic solution reduces the complexity of the mapping as there are no constraints to be dealt with while mapping the basic block. However, it requires a huge memory bandwidth, significantly reducing the energy efficiency of the system. As the compilation is built on *register allocation approach*, the symbol variables are stored in the register files when they are produced, and retrieved from the registers when used as operands. While doing so, the effects of the constraints in mapping are unavoidable. RLC restrict the use of some resources, and TLC force to reuse some resources. If there is only a single TLC in a basic block mapping, it becomes easier to start mapping from the known place. But several TLC and RLC complicates the mapping. Forced and blocked placements by these constraints induce extra routing effort (dynamically transforming the graph in compilation).

As the selection of the basic blocks during the mapping is important, we compare the number of TLC and RLC for several CDFG traversal in this section. Table III presents the comparison between the number of different constraints in the forward and backward CDFG traversal for Breadth First Search (BFS) and Depth First Search (DFS) strategies. As the trend is similar for other kernels we present the results for sobel and seperable 2D filter only. The numbers show that DFS strategy generates a lower number of RLC than the BFS in both forward and backward traversal. The number of RLC for sobel filter is much higher in BFS due to several sequential loops present in the kernel. The numbers of TLC are similar in both the strategies for different traversal mechanisms. Also for the different search strategies forward and backward traversal perform similarly. The DFS strategy is thus used.

5) Backtracking: For a basic block to be mapped (except the first one), this stage selects the first map out of several mappings generated for the last basic block mapped. The selected map updates the constraints for the current basic block mapping. If the compiler is unable to find a mapping solution for the basic block due to the constraints, this stage selects the second map from previous basic block to update the constraints and restart mapping of the new basic block.

Table III: Comparison of RLC and TLC numbers between different CDFG traversal

Kernels	Forward Traversal				Backward Traversal			
	Breadth First Search		DepthFirst Search		Breadth First Search		DepthFirst Search	
	# RLC	# TLC	# RLC	# TLC	# RLC	# TLC	# RLC	# TLC
Seperable 2D Filter	22	35	17	35	22	35	17	35
sobel Filter	64	85	35	85	69	85	35	85

The process continues up to the first basic block mapped until a valid mapping is found for the current basic block.

6) Update Constraints: In this stage, the compiler creates and updates a constraint database. This database is used in the placement algorithm, to place the data nodes and corresponding operation nodes according to the TLC and RLC. When mapping a current basic block, new variables cannot be placed in RLCs, while TLCs are used to map the symbol variables. If the symbol variables in the current basic block mapping are not present in the constraint database, then the variables are mapped using available resources, and the respective placements are used to update the constraint database prior to mapping of the next basic block. Once all the basic blocks are mapped the compiler generates the assembly containing a single map for the whole CDFG.

C. Assembler

Assembler holds the key to differentiate from the PEA model used in the compiler and the actual hardware implementation. The assembler takes the ASCII text assembly generated by the compiler and the instruction set architecture (ISA) and produces machine code, which can then be used to configure the PEs in the hardware. The ISA provides the added hardware information to the PEA model used in the compiler. As an example, the PEs in the IPA use an added constant register file (CRF) for storing the constants. The introduction of the CRF in the PEA model minimizes the instruction length by storing the immediates of the instruction into the internal registers, giving a low power solution. That is how the assembler separates the model used in the compiler from the actual implementation of the hardware. One can define their own PEA model and derive an architecture from that for actual implementation. Thus, the compiler can be used for a wide range of PEA variations.

V. Experimental Results

This section analyses the implementation results, providing performance, area, and energy consumption on several signal processing kernels. We carry out experiments to show the efficiency of the register allocation approach compared to the state of the art predication techniques, considering a wide range of control dominated kernels. An architectural exploration is also performed to find the optimal configuration in terms of number of load-store units and TCDM banks for an IPA with 4x4 PE array. Performance, area and energy efficiency are also compared with that of the or1k CPU [26].

A. Implementation Results

This section describes the implementation results for the IPA accelerator, providing a comparison with the or1k CPU. Both the designs were synthesized with Synopsys design compiler 2014.09-SP4 using STMicroelectronics 28nm UTBB FD-SOI technology libraries. Synopsys PrimePower 2013.12-SP3 was used for timing and power analysis at the supply of 0.6V, 25°C temperature, in typical process conditions. The cycle information was achieved simulating the RTL with Mentor Questa Sim-64 10.5c. The code-sizes (instructions and constants) of all the kernels used in the experiments are presented in Table IV. In the following, the exploration considers a 4×4 array with 16 PEs, each one including 20×32-bit instruction register file, a 32×8-bit regular register file and 32×16-bit constant register file, as shown in Table V. For area comparison, the CPU includes 32kB of data memory, 4kB of instruction memory, and 1 kB of instruction cache, which is equivalent to the design parameters of the IPA. The cost of the IRF is considered both in size and power. Thanks to the simpler architecture and tiny processing elements, at the target operating voltage of 0.6V, the IPA runs at 100 MHz while or1k can only reach 45MHz in the same operating point.

Fig. 9 shows the area of the whole array and memory with different numbers of TCDM banks, where the total amount of memory is kept constant at 32kB. As the area of LSUs is negligible if compared to the overall system area, we show the area results for the worst-case scenario with maximum number of LSUs present in the PE array (i.e. 16). As shown in Fig. 9, in the minimal configuration with 4 TCDM banks, the IPA area is dominated by that of the array (60%) and by the local data storage (35%), while the remaining 5% is consumed by the interconnect. Increasing the number of TCDM banks imposes a significant area overhead on the size of the interconnect. Also, the area of the TCDM increases as well due to the higher area/bit of small SRAM cuts necessary to implement 32kB of memory with several banks. Hence, it is fundamental to properly balance the number of LSUs and TCDM banks with the bandwidth requirements of applications.

B. Comparison of the proposed compilation approach with state of the art predication techniques

To evaluate the efficiency of the register allocation approach to handle the control flow we compare the execution of six control intensive kernels compared to the state of the art partial and full predication techniques. The results, presented in Table VI, show that the register based approach achieves a maximum of 1.33× (with minimum of 1.04× and average of 1.13×) and 1.8× (with minimum of 1.37× and average of 1.59×) performance gain compared to partial predication and full predication techniques. The maximum gain achieved over existing methods are highlighted in bold in the table. The smaller number of executed instructions allows the register allocation approach to outperform the partial and full predication techniques by an average of 1.54× (with min 1.35×, max 2×) and 1.71× (with min 1.44×, max 2×) respectively in terms of energy efficiency. The table also presents a comparison with respect to or1k CPU and C64 DSP processor [22] from TI. The

Table IV: Code size and the maximum depth of loop nests for the different kernels in the IPA

Kernels	FIR	MatM	Conv	Sep Filter	Non Sep Filter	FFT	DC Filter	cordic	sobel	gcd	sad	deblock	manh-dist
Code size (KB)	0.568	0.704	0.704	0.720	0.784	0.696	1.16	0.496	0.336	1.448	0.600	2.016	0.624
Max depth loop nests	2	3	3	3	4	2	2	1	1	1	2	3	2

Table V: Specifications of memories used in the IPA

Name	Type	Size
Global context memory	SRAM	8KB
TCDM	SRAM	32KB
Instruction Register File (IRF)	Registers	0.08KB
Regular register file (RRF)	Registers	0.032KB
Constant register fie(CRF)	Registers	0.128KB

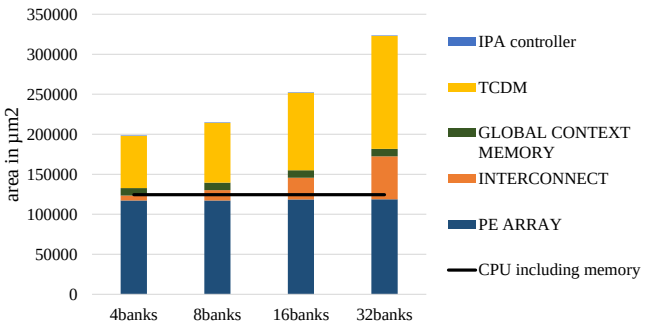


Figure 9: Synthesized area of IPA for different number of TCDM banks

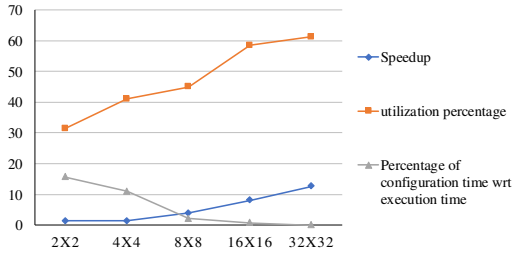


Figure 10: Performance of IPA executing matrix multiplication of different size

register allocation approach achieves a maximum of 3.94×, 15.8× performance gain and 7.52×, 32.77× energy gain over or1k and C64 processor, respectively. Due to the abundance of branches in these kernels, the DSP processor performs worst. Finally, we compare with the basic systematic load-store (SLS) based approach for control mapping. It is depicted from the Table VI that the register allocation approach performs an average of 1.16× (with max of 1.46×, min of 1.05×) better than the SLS based approach, while gaining an average of 1.31× energy efficiency with a maximum gain of 2× and minimum gain of 1.07×.

C. Architectural Exploration

This section provides an extensive comparison with respect to the CPU computational model and an evaluation of the performance of the IPA while varying the number of LSUs and TCDM banks, a critical parameter for data-hungry accelerators. To carry out the exploration, we selected 7 compute intensive signal processing kernels featuring a high bandwidth towards the TCDM.

1) Performance: Generally speaking, the IPA performs well when significant parallelism can be extracted from a kernel. This concept is well shown in Fig. 10, which compares the performance of the IPA with that of the or1k processor on a matrix multiplication when growing the size of the matrices from 2×2 to 32×32. It is possible to note that the increase of the kernel size increases the average utilization of the PEs as well, which in turn helps to enhance performance. It also demonstrates that the initial configuration time, which is dominant for small kernel size is well amortized for larger kernels, further contributing to improve performance.

Fig. 11 presents the total execution time (clock cycles) of seven compute-intensive kernels. The execution time is normalized with respect to that of or1k processor, where the kernels are compiled with -O3 optimization flag. The IPA outperforms the CPU by up to 20.3×, with an average speed-up of 9.7×. A quantitative performance comparison with respect to the CPU is presented in Table VII. The table presents the configuration and execution cycles in the IPA for different kernels. It also presents the average utilization of PEs over the total execution period and total number of instructions executed in the IPA. The instruction count includes the instructions that are replicated on all the active PEs for keeping the PE in synch across conditionals and jumps. It also includes NOPs that are used when some PEs are stalled due to manipulation of index variables. However, during NOP execution PEs are clock gated and do not consume dynamic power. The IPA achieves a maximum of 18× and an average of 9.23× energy gain over the CPU.

To establish the impact of the memory bandwidth over performance and energy efficiency, we vary the number of LSUs in the PE array from 4 to 16 and the number of TCDM banks from 4 to 32. The number of LSUs defines the available bandwidth from the TCDM to the array, while increasing the number of TCDM banks reduces the banking conflict probability, improving performance. To perform the exploration without any bias towards configurations, the innermost loops of the kernels are unrolled to get a maximum of 16 load-store operations in one cycle (as the highest number of LSUs considered is 16, in the exploration). In Fig. 11, each configuration is represented as a 2-dimensional number, where the first one represents the number of LSUs, and the second one represents the number of TCDM banks.

Results show that, as opposed to tightly coupled clusters of processors which require a banking factor of 2 (i.e. number of TCDM banks is twice the number of cores) [47], IPA performance is almost insensitive to the number of TCDM banks, and a configuration with a banking factor of 0.5 is sufficient to minimize the impact of contention on the shared memory banks for most applications. Indeed, while the typical processor execution requires several load/store operations for variables exceeding the size of the register file, direct CDFG

Table VI: Performance comparison between the register allocation approach and the state of the art approaches

Kernels	# Loops	# Condi tionals	Performance (cycles)						Energy (μJ)					
			reg based	SLS based [9][43]	partial [5]	full [1]	CPU	C64 DSP	reg based	SLS based [9][43]	partial [5]	full [1]	CPU	C64 DSP
cordic	1	2	328	408	396	542	513	286	0.001	**0.002**	**0.002**	**0.002**	0.004	0.002
sobel	4	11	179 617	**262 282**	188 253	245 583	454 028	669 794	0.736	1.102	1	1.058	3.531	5.656
gcd	1	1	55 312	58 596	**73 747**	92 852	67 545	92 184	0.227	0.246	0.392	0.4	0.525	0.778
sad	2	1	15 962	16 824	16 573	**28 776**	**62 932**	**252 193**	0.065	0.071	0.088	0.124	**0.489**	**2.13**
deblocking	5	7	472 258	495 081	518 722	727 243	834 683	1 310 220	1.936	2.079	2.754	3.134	6.492	11.064
manh-dist	1	1	6 288	6 826	6 738	9 522	15 394	55 317	0.026	0.029	0.036	0.041	0.12	0.467
	max gain			1.46×	1.33×	1.8×	3.94×	15.8×		2×	2×	2×	7.52×	32.77×

Table VII: Overall instructions executed and energy consumption in IPA vs CPU

Kernels		FIR	MatM (16×16)	Convolution	SepFilter	NonSepFilter	FFT	DC Filter
IPA	**Configuration cycles**	71	88	88	90	98	87	145
	Execution cycles	6 071	11 940	56 241	827 685	1 852 382	8 076	4 748
	Total number of instructions executed	44 294	110 946	531 815	7 349 843	17 486 486	76 310	28 868
	Active PEs/cycle (%)	46.1	58.5	59.2	55.5	59	59.7	39.5
	Energy (μJ)	0.022	0.043	0.202	2.98	6.669	0.032	0.017
	Energy (μJ) in non-clock-gated IPA	0.047	0.077	0.479	7.152	11.704	0.063	0.045
CPU	**Execution cycles**	37 677	96 256	616 805	5 982 730	9 084 101	164 480	50 085
	Energy (μJ)	0.132	0.337	2.159	20.94	31.794	0.576	0.175
	Speed-up	6.21x	8.06x	10.97x	7.23x	4.9x	20.3x	10.55x
	Energy-gain	6x	7.84x	10.69x	7.03x	4.77x	18x	10.29x

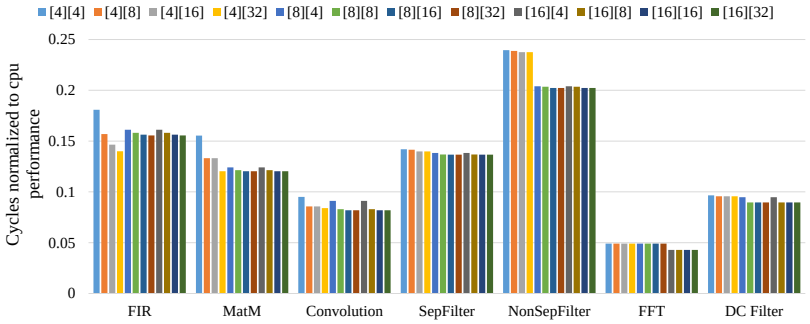


Figure 11: Latency performance in different configurations ([#LSUs][#TCDM Banks])

mapping on the IPA does not add extra memory operations except primary inputs and outputs, since all the temporary variables are stored in the register file of the PEs. Moreover, flexible point-to-point connections within the array allow to efficiently exchange data among PEs, further reducing the pressure on the TCDM. This concept is well explained in Fig. 4 and Fig. 1, which show the typical mapping of an application on the IPA.

2) Energy Efficiency: Fig. 12 shows the average power consumption breakdown for various configurations of the IPA. As expected, the PE array is the most dominant power consumer for all the configurations. The configurations with 4 TCDM banks achieve the best power advantages in each group, as increasing the number of banks increases the interconnect complexity, causing timing pressure on the array, which increases the sizing of the cells, hence power consumption.

Fig. 13 shows the average energy efficiency (MOPS/mW) for different configurations. Million Operations Per Second (MOPS) only considers the active PEs during execution, since a PE may be idle due to TCDM bank access conflicts, consecutive NOPs, or not mapped (not used in the application execution). Executions with high number of active PEs/cycle achieve large MOPS. As depicted in Fig. 13, for different number of LSUs in the PE array, the configuration with 4 TCDM banks achieves the best energy efficiency, since this is the least number of banks in each configuration, it

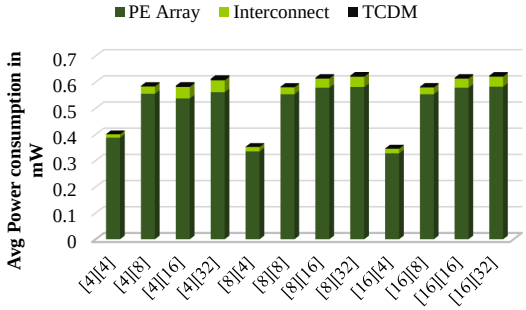


Figure 12: Average power breakdown in different configurations ([#LSUs][#TCDM Banks])

causes lowest power consumption. At the same time, the active number of PEs/cycle does not get significantly impacted due to the least memory access policy of the compilation. As a result, the best efficiency is achieved at 2306 MOPS/mW for matrix multiplication, in a configuration with 8 LSUs and 4 TCDM banks. The minimum energy efficiency is achieved at 1112 MOPS/mW for separable filter in a configuration with 4 LSUs and 16 TCDM banks.

To investigate the power gain in the fine-grained clock gating we present the energy consumption of the clock gated IPA and the non clock gated IPA in Table VII. The clock gated design consumes an average of 2× less power compared to that

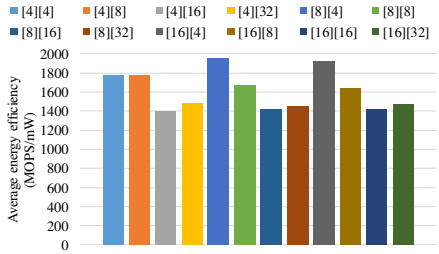


Figure 13: Average energy efficiency for different configurations ([#LSUs][#TCDM Banks])

of the non clock gated design. Due to the regular architecture of the PE array, fine grained power management is much more suitable to implement than a processor. Moreover, thanks to the efficient execution of CDFG on the array, the smaller energy required to execute an instruction in the IPA with respect to a CPU (5.6E-07 μJ vs 3.49E-06 μJ), and the effectiveness of the fine-grained power management the IPA outperforms the or1k CPU's energy efficiency by up to 18× (Table VII). The energy per instruction execution in the IPA is much less than that of the CPU due to its simple instruction set architecture. Also, the lower number of memory operations executed in the IPA helps reducing on the average energy consumption.

D. Comparison with low-power CGRA architectures

Table VIII shows a comparison with existing CGRAs. For some papers, energy efficiency figures could not be extracted, so 'NA' is put in the corresponding cell. The energy efficiency figures are provided both in the original manufacturing technology node and scaled to the 28nm technology, according to the power scaling factor $C*V^2$. C and V represent the effective capacitance (approximated with the channel length of the technology) and the supply voltage of the designs, normalized to the nominal parameters of the 28nm technology node. It should be noted that this simplified scaling factor penalizes our design, since deep-submicron technologies such as 28nm, where the load capacitance of gates is typically dominated by wires require much more buffering than mature technology nodes, which penalizes energy efficiency. Nevertheless, IPA provides leading-edge energy efficiency, surpassing by more than one order of magnitude other architectures featuring a C based mapping flow. The driving factors for this gain are (a) architectural simplicity with less complex interconnect network, (b) low power instruction processing, (c) lowest possible number of memory operations in application execution, (d) fine grained power management architecture. Compared to ultra-low power targets (that fit in a power envelope of 3mW), the IPA presents a better energy efficiency than [33] and [36] for which information could be extracted from the papers. One distinguishing characteristic of the proposed accelerator is the flexible execution model capable of implementing CDFG on the array without the need of a host processor, coupled with a fully automated mapping flow that starts from a plain ANSI C description of the application. Moreover, the memory architecture, based on a shared multi-banked TCDM enables easy integration within ultra-low-power tightly coupled clusters of processors, while fine-grained power management allows improving energy efficiency by up to 2×. The average power consumption of the IPA is 0.49mW, which is compatible with the ultra-low power target.

VI. Conclusion

This work presents an ultra-low power coarse grained reconfigurable array accelerator for near-sensor processing. The proposed *Integrated Programmable-Array* (IPA) is a 2-D array of NxN processing elements (a 4x4 configuration is used in this paper), and leverages a multi-banked tightly coupled data memory for data storage, to ease the integration in clustered multi-core architectures. We present a compilation flow targeting the mapping of both control and data flow portions of kernels onto the array of processing elements, aimed at reducing the pressure on the shared data memory, along with an architectural exploration of the memory architecture parameters. The results of the exploration show that a configuration of the IPA with 8 load-store units and 4 TCDM banks achieves the optimal performance/energy trade-off featuring an average speed-up of 9.7× (max 20.3×, min 4.9×) compared to a general-purpose processor. With respect to state of the art partial and full predication techniques, the proposed compilation flow improves performance by 1.54× on average (min 1.35×, max 2×) and energy efficiency by 1.71× on average (min 1.44×, max 2×). Thanks to the optimized architecture and mapping flow, the proposed accelerator achieves an average energy efficiency of 1617 MOPS/mW over a wide range of sensor signal processing kernels, surpassing other CGRA architectures featuring a C based mapping flow by more than one order of magnitude.

Acknowledgements

This work was funded by the ERC MultiTherman Project (ERC-AdG-291125), and by the OPRECOMP Project funded from the European Unions Horizon 2020 Research and Innovation Programme under grant agreement No. 732631. We also thank STMicroelectronics for granting access to the FDSOI 28nm technology libraries.

References

[1] M. L. Anido, A. Paar, and N. Bagherzadeh. Improving the operation autonomy of simd processing elements by using guarded instructions and pseudo branches. In *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, pages 148–155, 2002.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[3] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev. Architectural exploration of the adres coarse-grained reconfigurable array. In *Proceedings of the 3rd International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ARC'07, 2007.

[4] F. Campi, R. König, M. Dreschmann, M. Neukirchner, D. Picard, M. Jüttner, E. Schler, A. Deledda, D. Rossi, A. Pasini, M. Hübner, J. Becker, and R. Guerrieri. RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip. In *2009 International Symposium on System-on-Chip*, pages 110–113, Oct 2009.

[5] K. Chang and K. Choi. Mapping control intensive kernels onto coarse-grained reconfigurable array architecture. In *2008 International SoC Design Conference*, volume 01, pages I–362–I–365, Nov 2008.

Table VIII: Comparison with the state of the art low power targets

Ref	Arch	Maps	Source	Access local memory	Tech [nm]	Supply voltage	Area [mm^2]	Power [mW]	Freq [MHz]	Area eff [MOPS /mm^2]	Energy eff [MOPS /mW]	Energy eff scaled to 28nm [MOPS/mW]	Perf [MOPS]
Low power targets													
[25]	TCPA	CDFG	Customized	VLIW	90	1.0V	15	12.48	200	106	112.00	360	1587
[46]	Layers	CDFG	NA	PE	65	1.0	0.35	44.45	488	2786	21.94	72	975
[29]	SmartCell	CDFG	Customized	PE	130	1.0V	8.2	160	100	13.04	37.8	176	6048
[18]	PipeRench	DFG	Customized	PE	180	1.8V	55.5	675	120	NA	NA	NA	NA
[41]	SYSCORE	CDFG	NA	PE	90	1.0V	5.73	18.5	100	NA	NA	NA	NA
[3]	ADRES	DFG	ANSI C	VLIW	90	1.0V	15	80	100	94	17.51	56	1409
[4]	XPP	CDFG	ANSI C	PE	90	1.0V	42	93	150	310	10.00	32	13000
[53]	AsAP	CDFG	ANSI C	PE	180	1.8V	23.76	84	116	40	11.00	229	942
[48]	MUCCRA-3	DFG	Customized	VLIW	65	1.2V	8.82	11	41.4	NA	NA	NA	NA
Ultra-low power targets													
[33]	Lopes et al	DFG	NA	PE	90	1.0V	0.45	3.47	100	222	28.8	92.6	100
[36]	CMA	DFG	Customized	μC	65	0.5V	25	1.6	85	3	186[a]	430[a]	74[a]
[15]	SIMD-CGRA	DFG	ANSI C	PE	65	0.9	0.59	NA	1	NA	NA	NA	NA
[23]	ULP-SRP	DFG	ANSI C	VLIW	40	0.5V	NA	0.21	7	NA	NA	NA	NA
This paper	IPA	CDFG	ANSI C	PE	28	0.6V	0.25	0.49	100	3036	1617	1617	759

[a] PEs perform 8-bit operations, hence energy efficiency is normalized to equivalent 32-bit operations, does not include the power of controlling processor

[6] L. Chen and T. Mitra. Graph minor approach for application mapping on cgras. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(3):21, 2014.

[7] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Charm: A composable heterogeneous accelerator-rich microprocessor. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 379–384. ACM, 2012.

[8] S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini. Efficient mapping of CDFG onto coarse-grained reconfigurable array architectures. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 127–132, Jan 2017.

[9] S. Das, T. Peyret, K. Martin, G. Corre, M. Thevenin, and P. Coussy. A scalable design approach to efficiently map applications on CGRAs. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 655–660, July 2016.

[10] S. Das, D. Rossi, K. Martin, P. Coussy, and L. Benini. A 142 mops/mw integrated programmable array accelerator for smart visual processing. In *2017 IEEE International Symposium of Circuits and Systems (ISCAS)*, page Accepted, 2017.

[11] B. De Sutter, P. Raghavan, and A. Lambrechts. *Coarse-Grained Reconfigurable Array Architectures*, pages 449–484. Springer US, Boston, MA, 2010.

[12] M. Dehyadegari, A. Marongiu, M. R. Kakoee, S. Mohammadi, N. Yazdani, and L. Benini. Architecture support for tightly-coupled multi-core clusters with shared-memory hw accelerators. *IEEE Transactions on Computers*, 64(8):2132–2144, 2015.

[13] G. Donohoe. Reconfigurable data path processor, Apr. 19 2005. US Patent 6,883,084.

[14] G. W. Donohoe, D. M. Buehler, K. J. Hass, W. Walker, and P.-S. Yeh. Field programmable processor array: Reconfigurable computing for space. In *2007 IEEE Aerospace Conference*, pages 1–6. IEEE, 2007.

[15] L. Duch, S. Basu, R. Braojos, G. Ansaloni, L. Pozzi, and D. Atienza. Heal-wear: An ultra-low power heterogeneous system for bio-signal analysis. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2448–2461, 2017.

[16] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[17] M. Gautschi, A. Traber, A. Pullini, L. Benini, M. Scandale, A. Di Federico, M. Beretta, and G. Agosta. Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of openrisc cores. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 25–30, Oct 2015.

[18] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor. Piperench: A reconfigurable architecture and compiler. *Computer*, 33(4):70–77, 2000.

[19] M. Hamzeh, A. Shrivastava, and S. Vrudhula. Regimap: register-aware application mapping on coarse-grained reconfigurable architectures (cgras). In *Proceedings of the 50th Annual Design Automation Conference*, page 18. ACM, 2013.

[20] K. Han, J. K. Paek, and K. Choi. Acceleration of control flow on cgra using advanced predicated execution. In *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec 2010.

[21] K. Han, S. Park, and K. Choi. State-based full predication for low power coarse-grained reconfigurable architecture. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2012.

[22] T. Instruments. Tms320c64x/c64x+ dsp cpu and instruction set reference guide. *Texas Instruments, User manual SPRU732C*, 2005.

[23] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim. Ulp-srp: Ultra low-power samsung reconfigurable processor for biomedical applications. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(3):22, 2014.

[24] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi. Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization. In *Design, Automation and Test in Europe*, pages 12–17. IEEE, 2005.

[25] D. Kissler, A. Strawetz, F. Hannig, and J. Teich. Power-efficient reconfiguration control in coarse-grained dynamically reconfigurable architectures. *Journal of Low Power Electronics*, 5(1):96–105, 2009.

[26] D. Lampret, C.-M. Chen, M. Mlinar, J. Rydberg, M. Ziv-Av, C. Ziomkowski, G. McGary, B. Gardner, R. Mathur, and M. Bolado. *Openrisc 1000 architecture manual*. Opencores, January 2003.

[27] J. Lee, S. Seo, H. Lee, and H. U. Sim. Flattening-based mapping of imperfect loop nests for cgras? In *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Oct 2014.

[28] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, 1973.

[29] C. Liang and X. Huang. Smartcell: An energy efficient coarse-grained reconfigurable architecture for stream-based applications. *EURASIP Journal on Embedded Systems*, 2009(1):518659, Jun 2009.

[30] C. Liu, H. Ng, and H. K. So. Automatic nested loop acceleration on fpgas using soft CGRA overlay. In *Proceedings of the Second International Workshop on FPGAs for Software Programmers (FSP)*, pages 13–18, 2015.

[31] D. Liu, S. Yin, L. Liu, and S. Wei. Polyhedral model based mapping optimization of loop nests for cgras. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–8. IEEE, 2013.

[32] L. Liu, J. Wang, J. Zhu, C. Deng, S. Yin, and S. Wei. Tlia: Efficient reconfigurable architecture for control-intensive kernels with triggered-long-instructions. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):2143–2154, July 2016.

[33] J. Lopes, D. Sousa, and J. C. Ferreira. Evaluation of cgra architecture for real-time processing of biological signals on wearable devices. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–7, Dec 2017.

[34] K. T. Madhu, S. Das, N. Sivanandan, S. K. Nandy, and R. Narayan. Compiling HPC Kernels for the REDEFINE CGRA. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 405–410, Aug 2015.

[35] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings. A reconfigurable arithmetic array for multimedia applications. In

Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays, pages 135–143. ACM, 1999.

[36] K. Masuyama, Y. Fujita, H. Okuhara, and H. Amano. A 297mops/0.4 mw ultra low power coarse-grained reconfigurable accelerator CMA-SOTB-2. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2015.

[37] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. Dresc: A retargetable compiler for coarse-grained reconfigurable architectures. In *Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on*, pages 166–173. IEEE, 2002.

[38] E. Mirsky, A. DeHon, et al. Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *FCCM*, volume 96, pages 17–19, 1996.

[39] N. Ozaki, Y. Yoshihiro, Y. Saito, D. Ikebuchi, M. Kimura, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo. Cool mega-array: A highly energy efficient reconfigurable accelerator. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pages 1–8, Dec 2011.

[40] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 166–176. ACM, 2008.

[41] K. Patel, S. McGettrick, and C. J. Bleakley. Syscore: A coarse grained reconfigurable array architecture for low energy biosignal processing. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 109–112. IEEE, 2011.

[42] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 1989.

[43] T. Peyret, G. Corre, M. Thevenin, K. Martin, and P. Coussy. Efficient application mapping on cgras based on backward simultaneous scheduling/binding and dynamic graph transformations. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, pages 169–172. IEEE, 2014.

[44] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini. A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[45] Z. E. Rakossy, A. Acosta-Aponte, T. G. Noll, G. Ascheid, R. Leupers, and A. Chattopadhyay. Design and synthesis of reconfigurable control-flow structures for cgra. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, Dec 2015.

[46] Z. E. Rákossy, D. Stengele, G. Ascheid, R. Leupers, and A. Chattopadhyay. Exploiting scalable cgra mapping of lu for energy efficiency using the layers architecture. In *Very Large Scale Integration (VLSI-SoC), 2015 IFIP/IEEE International Conference on*, pages 337–342. IEEE, 2015.

[47] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Bartolini, P. Flatresse, and L. Benini. A 60 GOPS/W, -1.8 V to 0.9 V body bias ULP cluster in 28 nm UTBB fd-soi technology. *Solid-State Electronics*, 117:170 – 184, 2016.

[48] Y. Saito, T. Sano, M. Kato, V. Tunbunheng, Y. Yasuda, M. Kimura, and H. Amano. Muccra-3: a low power dynamically reconfigurable processor array. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 377–378. IEEE Press, 2010.

[49] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers*, 49(5):465–481, May 2000.

[50] Y. Song and Y. Lin. Unroll-and-jam for imperfectly-nested loops in dsp applications. In *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '00, pages 148–156, New York, NY, USA, 2000. ACM.

[51] H. Su, Y. Fujita, and H. Amano. Body bias control for a coarse grained reconfigurable accelerator implemented with silicon on thin box technology. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, Sept 2014.

[52] S. Yin, P. Zhou, L. Liu, and S. Wei. Acceleration of nested conditionals on cgras via trigger scheme. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2015.

[53] Z. Yu, M. J. Meeuwsen, R. W. Apperson, O. Sattari, M. Lai, J. W. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas. AsAP: An asynchronous array of simple processors. *IEEE Journal of Solid-State Circuits*, 43(3):695–705, 2008.

Satyajit Das is a PhD student at the Université de Bretagne-Sud, France collaborated with the University of Bologna, Italy. His research interests include reconfigurable computing, low power system design, and embedded systems.

Kevin J. M. Martin received a M.S. degree in electrical and computer engineering in 2004 and a PhD in computer science in 2010 from the Université de Rennes, France. He is since 2011 an associate professor at Université de Bretagne-Sud in Lorient, France, in the Lab-STICC. His research interests stand at the crossing point between architecture, methods and tools for embedded systems, including custom processors, CGRAs, multi/many cores, high-level synthesis, computer-aided design tools, compilers and software engineering.

Davide Rossi received the PhD from the University of Bologna, Italy, in 2012. He has been a post doc researcher in the Department of Electrical, Electronic and Information Engineering Guglielmo Marconi at the University of Bologna since 2015, where he currently holds an assistant professor position. His research interests focus on energy efficient digital architectures in the domain of heterogeneous and reconfigurable multi and many-core systems on a chip.

Philippe Coussy is a full professor in the Lab-STICC (UMR CNRS) at the Université de Bretagne-Sud, France, where he leads the Communications Architectures Circuits Systems department. He was graduated from Université Pierre et Marie Curie (MSc, 1999), Université de Bretagne-Sud (Ph.D., 2003 and Habilitation 2011). He is a member of the technical committee of the IEEE Signal Processing Society, Design and Implementation of Signal Processing Systems (DISPS) since 2011.

Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Universita di Bologna. He received a PhD degree from Stanford University in 1997. Dr. Benini's research interests are in energy-efficient system design, from embedded to high-performance computing. He is also active in the design of smart sensing microsystems and ultra-low power VLSI circuits. He a Fellow the ACM and a member of the Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg award.

B. Selected publication on mapping dataflow applications

This appendix provides the author version of the paper "Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs", as a complement to the work presented in chapter 5.

Publication details

Title	*Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs*
Authors	Mostafa RIZK, Kevin J. M. MARTIN and Jean-Philippe DIGUET
In	IEEE Transactions on Parallel and Distributed Systems
DOI	10.1109/TPDS.2022.3177957
	vol. 33, no. 12, pp. 3959-3976, 1 Dec. 2022
URL	`https://hal.archives-ouvertes.fr/hal-03702259`

Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs

Mostafa Rizk, Kevin J. M. Martin, and Jean-Philippe Diguet

IEEE Transactions on Parallel and Distributed Systems
Print ISSN: 1045-9219
Online ISSN: 1045-9219
Digital Object Identifier: 10.1109/TPDS.2022.3177957

***Abstract*—Multiprocessor system-on-chip (MPSoC) platforms have been emerging as the main solution to cope with processor frequency ceiling and power density issues while still improving performances. Then, network-on-chip (NoC) has been adopted to provide the increasing number of processors with the required communication bandwidth as well as with the necessary flexibility. Video processing and streaming applications are adopting dynamic dataflow model of computation as the need for high performance parallel computing is growing. Dataflow applications executed on modern MPSoC-based architectures are becoming increasingly dynamic and more data-dependent. Different tasks execute concurrently with significant modifications in their workloads and resource demanding over time depending on the input data. Hence, adopting any static or offline dynamic scheduling for mapping tasks will not cope with the computation variations. This paper introduces an original run-time mapping algorithm based on the Move Based (MB) method targeting a dedicated heterogeneous NoC-based MPSoC architecture to achieve workload balancing and optimized communication traffic. The performance of the proposed algorithm is verified by conducting cycle-accurate SystemC simulations of the adopted NoC implementing a real MPEG4-SP decoder. The obtained results reveal the effectiveness of our proposed algorithm. For various real-life videos, the proposed algorithm systematically succeeded to enhance significantly the performance.**

***Index Terms*—NoC, Heterogeneous MPSoC, Run-time remapping, Dataflow actor, Move-based algorithm.**

I. Introduction

MULTIPROCESSOR system-on-chip (MPSoC) platforms have been emerging as the main solution to cope with processor frequency ceiling and power density issues while still improving performances. Then, networks-on-chip (NoCs) have been adopted to provide the increasing number of processors with the required communication bandwidth as well as with the necessary flexibility. But legacy code for instance, mainly designed for single or few core architectures, does not scale well with manycore architectures and fails to fully benefit from the available parallelism. However, as discussed decades ago [1], dataflow programming can address the limitations of conventional approaches regarding synchronization and shared memory issues. With the rise of massively parallel architectures, we can reconsider the use of dataflow programming as a solution to efficiently exploit the resources of parallel architectures for computing intensive application domains such as video coding, computer vision, machine learning and physics simulation for instance.

A dataflow application can be specified as a graph where nodes, called actors, process data called token(s). The computational models are based on First-In First-Out (FIFO) buffers and respect their formalized read and write rules. Each FIFO holds a set of tokens. Fig. 1(a) illustrates a network of actors, which exchange tokens through defined FIFO channels [2]. Fig. 1(b) presents an example of a structure of the software FIFO generated with the tool ORCC [3]. A network of actors holds specific features that make it different from a generic task graph. First, an actor is non-preemptive. Once started, an actor ends its execution. Second, the actor can start if and only if there are enough tokens as input, and enough space in the output FIFOs. The FIFOs are considered updated (i.e. tokens consumed and produced) at the end of the execution of the actor, establishing a conservative synchronization scheme, and preventing from any data race.

When the number of actors is larger than the number of processing elements (PEs), then the main design challenge is the mapping of actors on the network of PEs. In the case of static dataflow [4], where the number of tokens produced and consumed by the actors is known, an optimal solution can be computed offline [5]. However, an increasing number of applications cannot be specified with a static graph since the performance improvement of complex applications usually lead to context and data-dependent optimizations. This evolution is, for instance, significant in the domain of video coding. Dynamic models are then used to express data-dependent behavior of some applications [6]. Dynamic dataflow is a useful model of computation (MoC) for handling streaming data and video processing applications.

As the workload of an actor may change according to the input data set, adapting the mapping while the application runs is required to optimize the use of the computing and communication resources. The mapping problem is known

M. Rizk is with CNRS and member of Lab-STICC UMR CNRS 6285, Brest, France also with the School of Engineering, International University of Beirut, Lebanon, and with the Physics and Electronics Department, Lebanese University, Lebanon. e-mail: mostafa.rizk@imt-atlantique.fr

K. J. M. Martin is with Universtité de Bretagne-Sud (UBS) and member of Lab-STICC UMR CNRS 6285, Lorient, France

J-P. Diguet is with CNRS and member of CROSSING, IRL CNRS 2010, Adelaide, Australia

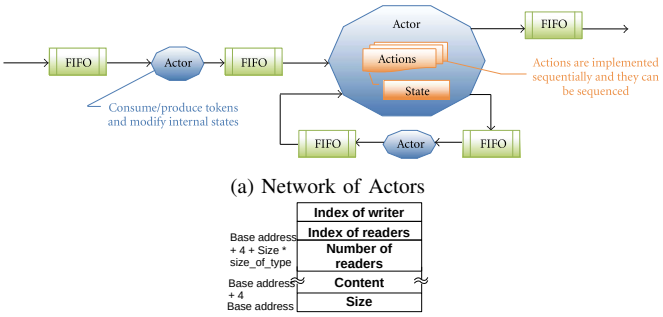
as NP-complete. Heuristic methods, for a fast response time, are thus required to address manycore architectures. Run-time adaptation relies on system observation, decisions and configurations. Several previous works have addressed the problem of task mapping at run-time. In [7], the authors have proposed a dynamic resource balance algorithm targeting NoC-based Many-core homogenous platforms to enhance the system performance by balancing the utilization of on-chip computing resources and communication resources. In [8], the authors have introduced a hybrid application mapping that combines design-time analysis with run-time mapping in the context of dynamic thermal and reliability-aware resource management. Most of the available methods focus on determining the suitable mapping of tasks before starting the execution of the application [9], [10], [11], [12], [13], [14]. The mapping of actors is also an active topic for other target platforms like Coarse-Grained Reconfigurable Arrays (CGRA) [15] or Field Programmable Gate Array (FPGA) [16].

This research work addresses the problem of reconfiguring at run-time and at the application level the mapping of dataflow actors on heterogeneous processors. In this work, heterogeneous means that processors share the same instruction set architecture (ISA) while having different coprocessors and different clock domains. The proposed method, which is called *run-time remapping*, relies on continuous monitoring of exact performance metrics such as the computational time and communication time during real-time execution of the application. Accordingly, a new mapping of the involved actors is determined at run-time targeting the enhancement of the overall performance. This approach is sequentially repeated while the application is running. The application is neither suspended nor modified. The proposed remapping method meets with the dynamic behavior of dataflow applications. Static or offline mapping methods cannot capture the dynamic behavior and thus may not lead to optimal solutions. Also, on-the-fly and hybrid mapping methods suffer from a lack of means to monitor the performance and remap the actors accordingly. In order to apply the proposed method, the architecture of NoCs must be augmented to efficiently provide new services of monitoring performance metrics and remapping the actors, which are not available in conventional networks.

Adopting the devised remapping method and NoC-based architecture leads to balancing the workload. The obtained results show that the adoption of the remapping method reduces the standard deviations of the computational times and communication times of involved processors by 38.58% and 69% respectively. Thus, the variation of the use rate of processors is reduced compared to running the application without remapping. In addition, a reduction of 8.6% in the total execution time has been achieved as well as a reduction of 21% in the number of packets' hops is recorded when comparing to the execution without remapping.

In this paper we introduce three contributions:

- First, we optimize for a NoC-based architecture with heterogeneous processors, a new run-time remapping (RR) algorithm based on the Move Based (MB) method [17], which allows only one actor to move at a time from one processor to another. Our solution is then compared with state of the art methods for dataflow architectures.
- Second, we present new NoC services that allow to implement the observation and adaptation mechanisms.
- Finally, we demonstrate our solution with a full implementation of MPEG4-SP, which is available as a reference of a typical dynamic dataflow application. It is also complex enough to exhibit data-dependent execution and communication times. We consider a SystemC packet cycle-accurate NoC simulator to fully decode reference videos and demonstrate the effectiveness of the adaptation mechanism with a real-life dataflow application.



(a) Network of Actors

(b) Structure of the software FIFO

Fig. 1. Actors and software FIFO Models [2]

The rest of the paper is organized as follows. Section II presents the related work. Section III illustrates the adopted architecture model. Section IV describes the processing flow. Section V details the conducted experiments and presents the obtained results. Finally, Section VI concludes the paper.

II. Related Work

The question of mapping parallel applications on multi or many-core architectures is a very wide problem, with a large number of dimensions, including the programming model, the target architecture (homogeneous or heterogeneous, bus-based or NoC-based, etc.), and the optimization goal (throughput, execution time, energy, etc.) [18]. The interested reader can refer to the paper gathering different mapping strategies for NoC-based architectures [19]. Following the taxonomy proposed in [18], the mapping problem can be solved based on two main strategies: design-time, and run-time. When solved at design-time, the mapping is called *static* since it's computed offline and does not change while the application runs. This approach allows for exact methods to find an optimal solution [20] [5] [15], but suffers from a lack of flexibility since it cannot capture the dynamic behavior of some applications. Moreover, even in the case of deterministic execution times of actors in a static context, the paper [21] interestingly shows the difference between the optimal mapping obtained from a well-formalized problem and the real execution trace, due to execution variabilities coming from the hardware.

The dynamic workload should be handled using run-time techniques. The run-time mapping strategies can themselves be divided into two categories: on-the-fly mapping, or hybrid mapping. On-the-fly mapping techniques are application- and

platform-agnostic and solve the problem online. Very simple and efficient heuristics should be used to shorten the response time. For NoC-based MPSoCs, various fast heuristics targeting the reduction of communications under constraints have been already proposed [22] [23] [24]. These approaches consider one task per core. Allowing multiple tasks on one core is considered in [10]. Heuristics are fast but can be far from optimal solutions, so hybrid approaches have been introduced. They are based on pre-computed optimal solutions for a set of cases. The job is split into two phases: (1) at design-time, a set of solutions is computed, and (2) one solution is selected at run-time. A wide variety of approaches can then be cited: based on traces in [25], on priority in [26], on scenario in [27], on previously identified design points in [28], or on WCET and scheduling in [29]. None of these studies demonstrates its efficiency with real video applications running reference sequences. The proposed real-time mapping reconfiguration method in [8] requires to suspend the currently running application and the manager remaps the tasks at run-time according to scenarios previously defined at design-time based on the evaluation of multiple mappings, optimizing for their resource requirements and power consumption. Finally, a last approach can fall into the family of hybrid mappings, which considers to recompute partially the mapping problem at run-time. This is called *run-time remapping*. The work presented in this paper follows such an approach for dataflow applications and leverages design-time analysis profiling results to find at run-time a first mapping. The application is then monitored to update the profiling results and a run-time remapping algorithm runs regularly to check if a new mapping would be better than the current one.

Among the innumerable papers dealing with task mapping, we consider run-time methods for dataflow tasks, and have identified a limited number of solutions. In [30], the mapping is modeled as a graph partitioning problem, and the problem is solved at run-time by METIS tool, based on profiling information obtained by a first run. Though the migration cost of the actors is not taken into account, the results are promising and could be improved if the mapping does not change completely at each iteration. The approach in [17] allows to successively refine the mapping according to the dynamic behavior of the application, by allowing only one actor to move at a time from one processor to the other. This approach assumes dynamic dataflow application and the target architecture is composed of several heterogeneous cores interconnected by a bus or a NoC. The communication cost is computed based on a rough analytical model of the interconnection network, with the loss of accuracy that comes with it, whereas in our work, we consider profiled values gathered automatically by the system, with a finer grain down to the link. In [31], the application is specified with KPN (Kahn Process Network) and the target architecture is a shared-memory based MPSoC, with also a model of the communication channel (bus or NoC). The approach proposes to rely on three main steps: the two usual design-time preparation and run-time mapping steps plus a new customization step. The design time step computes a set of candidates and populates a database. The run-time mapping initialization derives from the candidates a new initial mapping for the given workload. Finally, the run-time customization step incorporates a Scenario-based run-time Task Mapping (STM) algorithm that is applied to find new mapping of tasks when the system detects that an objective is unsatisfied. It first detects the so-called *critical task* and then identifies why it misses its objectives: either poor locality or load imbalance. In case of poor locality, an algorithm that considers the communication between tasks is used to find a new mapping. In case of load imbalance, a load balancing strategy based on computational demands of the tasks is used. This step produces a new mapping that may move several tasks, which leads to a (re-)mapping overhead.

When focusing on the small subset of the existing work around hybrid and run-time (re-)mapping of dataflow applications on NoC-based architectures, we consider the work presented in [31] for comparison.

III. Architecture Model

The target architecture is a heterogeneous Multi-Processor System on Chip (HMPSoC) containing several different PEs and shared memories connected with a Network-on-Chip (NoC). Fig. 2 presents the structure of the adopted NoC-based architecture. Our method is scalable and without loss of generality we consider a specific model of architecture which is required for a data-accurate functional simulation with a packet-level time accuracy. The target architecture is a 4×4 mesh-based NoC with 32-bit links that interconnects 28 intellectual property (IP) cores including 15 memory modules, 12 PEs and a processing element that acts as a manager (MGR). The PEs and memory modules are technologically independent of the structure of the NoC. They communicate through the network using a network interface (NI). We consider a simple NoC model that employs the wormhole packet switching mode, the deterministic XY routing algorithm, and a flow control policy without virtual channels. The implemented routers have one buffer of 3 flits per input port and use distributed arbitration logic (one arbiter per port). The back-end part of the NI is typical and includes a packet maker/un-maker, which are used to assemble and disassemble the packets, and a priority manager to synchronize packet transmission and reception.

In this work, it is assumed that $PE1$ imports the incoming streamed data from an Input buffer and $PE12$ outputs the processed data. Fig. 2 illustrates the buffers in order to communicate with external systems. Each PE has its local memory. It is assumed that there are no restrictions to map any MPEG4-SP application actor to any PE. The used PEs can all work in parallel according to dataflow firing rules. However, some PEs are enhanced by hardware accelerators dedicated to certain functionalities in order to perform them more efficiently. The shared memories are distributed in memory blocks which have a unique NI. From an NoC perspective, the novelty is the introduction of new command packets used as instructions to manage FIFO accesses, broadcast mapping information, collect monitoring data, and the transfer of binary codes. In order to cope with the command packet and associated notification packet concepts, the NIs implement some additional

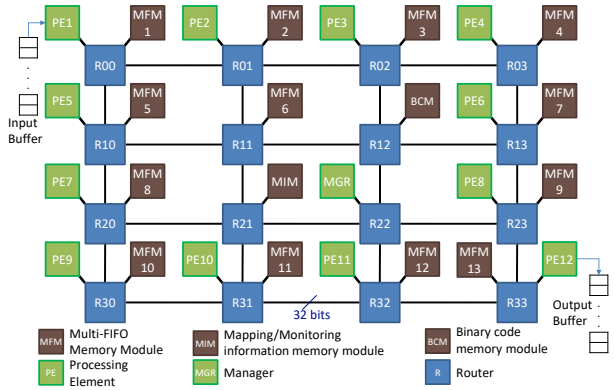


Fig. 2. The structure of the used NoC-based architecture

TABLE I
HARDWARE ACCELERATORS USED IN THE SIMULATION PLATFORM

PE ID	Accelerated Function	Acceleration Ratio
PE3 & PE6	IDCT	1/0.3
PE4	IQ + IAP	1/0.75
PE10	Add	1/0.57
PE11	Interpolation	1/0.4

TABLE II
PROCESSING ELEMENT OPERATING FREQUENCY

PE ID	Operating Frequency
PE1, PE12, MGR	f
PE2, PE6, PE10	$2f$
PE3, PE7, PE11	$3f$
PE4, PE8	$4f$
PE5, PE9	$5f$

logic modules. The command packets were already proposed in [32] but only produced by the manager and for a specific application.

A. Manager

The manager is a PE dedicated to the following five tasks: (1) map initially the actors on the available PEs, (2) parse the feedback collected data from all modules (memories and PEs), (3) apply the run-time remapping algorithm and selects the actor to be moved (if any), (4) notify the corresponding PEs (*looser*, *gainer*, etc.) about the updated mapping and (5) manage the transferring of the binary code corresponding to the moved actor from the shared memory into the cache of the gainer processor.

B. Processing Elements

The target platform includes twelve PEs. All PEs are supposed to be able to execute any of the forty-one actors involved in the MPEG4-SP application. As the number of PEs is smaller than the number of actors, each PE is considered to run more than one actor. Hence, an actor scheduler is required to manage the order of execution of actors. Mainly, in dataflow applications, all schedulers suffer from inefficient polling which leads to useless memory accesses when a scheduling attempt fails. In this work, the well-known round-robin scheduling technique has been adopted in all PEs. The actors are given the attempt to be executed in a circular order without priority. The PE will execute the allocated actor if there are enough input tokens and enough space in the output FIFOs as specified in dataflow applications.

Furthermore, some PEs are augmented with hardware accelerators in order to perform special functions more efficiently. In this work, we adopt one of the hardware accelerator specification described in previous similar work [17]. Table I shows the list of accelerators adopted in the simulation platform. In addition, the PEs have been specified randomly to operate on different frequencies. Table II shows the randomly chosen operating frequency of all PEs in terms of the NoC operating frequency f.

C. Memory Modules

The tailored platform integrates three types of memory modules. Each module includes a memory block that returns the data allocated at its specified address. Since the PEs and the manager do not recognize the local mapping of stored data in each memory module and in order to remain compliant with any available memory, the typical NI is extended to accommodate the services for managing the addressing and arranging the retrieved output bits into flits. These new functionalities are implemented as additional components in the front-end of the NI corresponding to each memory module type in order to be independent of NoC parameters. In the following the functionalities of each memory type is described.

1) Binary code memory module (BCM): It contains the binary codes of all actors. The manager sends a specific packet request to BCM to forward the binary code of the moved actor to a given PE according to the decision taken after executing the RR algorithm. A simple module, so-called memory address mapper (MAM), is integrated into the NI of the BCM in order to find the correct memory address. For a specific actor, MAM determines the starting address of the binary code and its corresponding size based on the actor's ID and by the means of simple look-up-tables that include the starting addresses and the size of the binary codes of all actors. Furthermore, MAM manages the extraction of data from the memory and delivers it to the packet maker unit.

2) Mapping/Monitoring information memory module (MIM): This memory module accommodates twelve memory blocks. Each block is dedicated to a specific PE and is supposed to store two types of data. The first type is the mapping information, which is generated by the manager and indicates which actors are to be executed by each PE in addition to their supplementary information about input and output FIFOs and the reading orders for each input FIFO (III-D1c). The second type is the monitoring information (III-D1e), which is collected by the PEs during processing a specified number of video frames. Storing the monitoring information overwrites the mapping information, which is not needed by the PEs anymore.

When a packet holding either mapping or monitoring information is received, the MIM module first identifies the

TABLE III
ADDRESSES DETERMINED BY THE MFM-NI CONTROLLER

Packet Type	Starting Address	Offset
Request/Set writing index	$FIFOsize$	0
Request/Set reading index	$FIFOsize + 1$	reading order
Reading Request packet	Reading address	incremented till
Data packet	Writing address	reaching data size

corresponding PE. Accordingly, it dissembles the packet and stores the data found in the packet payload into the memory block assigned to the identified PE.

Moreover, the MIM informs the manager about the availability of new monitoring information and the corresponding PE about the availability of new mapping information. To do so, the MIM sends notification packets (III-D1a) as per the concept of notifying memory concept demonstrated in [2]. In addition, the MIM responds to reading requests (III-D1b) sent from the manager to acquire the stored monitoring information from the PEs or to get the new mapping information.

3) Multi-FIFO memory module (MFM): This type of memory module is dedicated to store the data which is either imported to the system or processed by the PEs. Each MFM accommodates a specific number of FIFOs. Only one FIFO is used at once. The inputs of all FIFOs are connected to the module's inputs using demultiplexers whereas the FIFOs' output data ports are multiplexed. This signal is buffered from the value of FIFO address which is specified in the payload of the arriving packets (see Fig. 4). The multiplexer and demultiplexers are added to the adapter in the NI.

Moreover, the MFMs receive the following types of packets: (1) FIFO Index packet (III-D1f) that aims either to retrieve or to set the writing and reading indexes, (2) Data Reading Request packet (III-D1g) that demands to read data from a specified FIFO and (3) Data packet (III-D2a) that is used to write data in a specified FIFO.

A simple circuit is integrated into the adapter of the NI in all MFM modules in order to manage the memory addressing for all listed-above packet types. It is composed of a simple controller and two 4-to-1 multiplexers and an adder in order to generate the appropriate address values to be given to the MFM FIFOs. After disassembling the arriving packet, the packet un-maker delivers the packet type and the data size to the controller. Accordingly, the controller generates the control signals to configure the two multiplexers, which are dedicated to select the values of starting address and the offset as listed in Table III. These two values are then added to compute the memory address. In addition, the controller determines the number and type of the required memory accesses. It incorporates a simple comparator and an address counter which is incremented for each required access.

D. Packets' structure

The developed NoC architecture considers two categories of packets: (1) command packets and (2) data packets.

1) Command packets: Command packets are initiated by the cores and processed by the NIs of destination nodes. Several command packets, described hereafter, have been opted in order to manage FIFO accesses, send mapping information, collect monitoring data, and manage the transferring of binary codes.

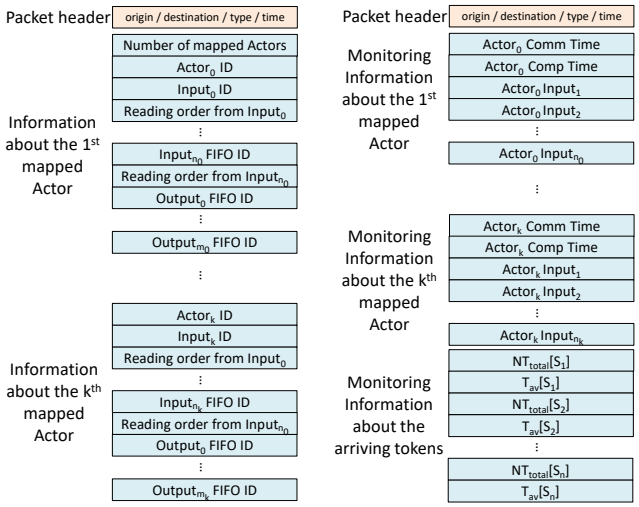


Fig. 3. Packets structure for mapping (left) and monitoring (right) information

a) Notification packets (NP): The NPs aim to inform the PEs that new information is ready to be requested. This technique is inherited from the notifying memories (NM) concept presented in [2]. When receiving a NP, the PE will send a reading request to retrieve the available data at the corresponding notifying memory. In this work, notification packets are used either to inform an ordinary PE that new mapping information is available or to notify the manager that updated monitoring information has been generated and stored. The NP has empty payload and aims to trigger the manager and PEs to request data when it is ready rather than frequent inefficient polling.

b) Monitoring/Mapping information reading request packets (MRP): This type of packet is used to request the information stored in the MIM module as a response to the NP. It is either generated by the manager to acquire the new monitoring information sent from a definite PE or by one of the PEs to get the new mapping information provided by manager. For both information types, monitoring or mapping information, the request packet does not include any payload.

c) Mapping information packets (M_pIP): The manager uses a M_pIP to inform all involved PEs after determining or modifying the actor mapping strategy. Its payload includes the following: (1) the number of actors which are mapped to the PE, (2) the IDs of the mapped actors, (3) the IDs of the input and output FIFOs, and (4) the actor reading order in each input FIFO. Fig. 3 illustrates the structure of the packet holding the mapping information.

d) Mapping Confirmation packets (MCP): A MCP aims to inform the manager that the new mapping information is well received by both the former and the new owner of the actor. The MCP payload is also empty.

e) Monitoring information packets (M_nIP): This type of packet holds the feedback information needed by the manager to perform the RR algorithm. Fig. 3 presents the structure of the monitoring information packet.

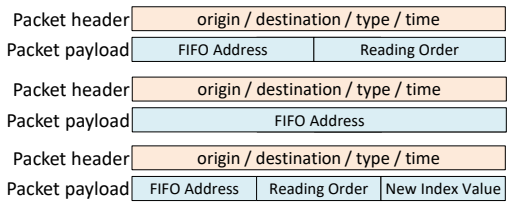


Fig. 4. Packet structures for holding reading (top) / writing (middle) index requests and set index (bottom)

f) FIFO index packets (FIP): The FIPs are designed to hold the writing indexes or reading indexes of FIFOs. As mentioned before, DF applications rely on a large number of requests to memories for firing rule checking. So, these indexes are used to determine either the number of available tokens corresponding to each reader actor or the free space in a FIFO. If data is required to be read from input FIFOs, the firing rule is satisfied by checking if the number of available tokens in all input FIFOs is equal or greater than the required number during computation. Whereas, if data has to be written to output FIFOs, the firing rule is satisfied by checking if all output FIFOs have sufficient empty room to accommodate the produced tokens. Hence, before processing an action, a PE has to request the reading and/or writing indexes of input and output FIFOs. When the PE receives the value of demanded reading/writing index, it will check the satisfaction of the firing rule. After reading/writing data from/to a FIFO, the reading/writing index has to be incremented by the size of the transferred data. The PE, which consumes/produces data, has to set the new reading/writing index in the targeted FIFO after reading/writing operation is performed. Accordingly, four types of packets are utilized: (1) Request read index, (2) Request write index, (3) Setting read/write index, and (4) Holding read/write index.

As the FIFO may have several reading indexes corresponding to different reader actors, the PE has to determine the reading order of the actor and sends it in the payload of the packet. However, a FIFO has only one writer actor; hence, to attain the value of its writer index the PE has to send the FIFO address in the destination memory module. In both packets, the packet type, given in the packet's header, is used by the NI at the destination memory module to decode the request type. Fig. 4 depicts the structure of the FIP packets holding the requests of a reading index and writing index.

On the other side, whenever a memory module receives a request of reading/writing index it will retrieve its value from the specified FIFO and sends it back to the PE. The NI in the memory module will assemble a 1-flit payload packet as shown in Fig. 4.

In order to set the reading/writing index after finalizing the data transfer operations from/to a FIFO, the PE sends a control packet that notifies the FIFO about its new reading/writing index. It includes one flit that contains the FIFO address in the destination memory module, the reading order of the actor, and the new value of the reading index. Since the FIFOs have only one writer actor, the writing packet payload simply includes the address of the targeted FIFO in the destination memory module and the new value of the writing index.

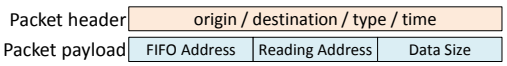


Fig. 5. The structure of the packets holding the reading requests

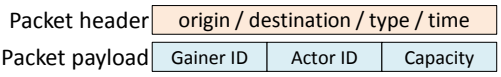


Fig. 6. Manager command requesting the transfer of the moved actor code

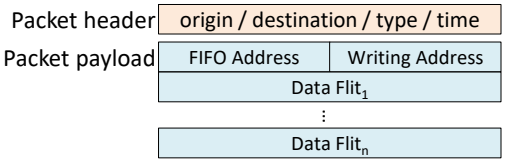


Fig. 7. The structure of packets carrying processed data

g) Data reading request packets (DRP): Fig. 5 presents the packet holding the reading request of data from PE to memory module. Its payload consists of one flit that includes the address of the FIFO in the destination memory module, the starting address of reading, and the size of required data.

h) Code transferring packets (CTP): Actor binary codes are stored in a shared memory. When updating the actor mapping, the binary code referring to the moved actor should be transferred from the shared memory to the cache of the new PE. The manager sends a command of transferring the binary code in the form of a reading request packet. The sent request includes the actor ID, the address of the new PE, and the size of transferred data per packet (see Fig. 6).

2) Data packets: The second category of packets refers to the ordinary flow of data between PEs and memory modules. These packets, described hereafter, carry data that is either processed in a PE and written in a memory module or sent from a memory module as a response to a PE reading request.

a) Dataflow packets (DFP): The DFPs encompass all packets transferred between PEs and the FIFOs distributed in the memory modules. They carry data that is either processed in a PE and will be stored in a FIFO or sent from a FIFO as a response to a PE reading request. Fig. 7 presents the structure of packets carrying processed data in their payloads.

b) Binary code packets (BCP): The BCPs aim to transfer the binary code from the shared memory to the cache memory of the new PE. Note that the binary code is divided into sections of reasonable sizes which are transferred consequently. The size of the transferred data (payload capacity) is specified by the manager according to the monitored traffic in the network and based on the required cache lines to be filled before launching the actor on the new PE. For example, the packet including in its payload 64 flits of 32-bitwidth transfers 256 bytes which form 4 lines of L1 cache.

IV. Processing Flow

A. Initial mapping

Initially, the actors are mapped randomly to the PEs, or can be mapped using the exact method presented in [20]. FIFOs are

TABLE IV
PARAMETERS AND VARIABLES USED FOR THE MAPPING ALGORITHM

Parameter	Definition
DPN application graph (DPNapp)	
$\|\mathbb{A}\|$	Number (Nb) of actors
$\|\mathbb{F}\|$	Nb of FIFO channels
$\|\mathbb{K}\|$	Nb of data packets
$\|\mathbb{I}_c\|$	Nb of input ports of actor A_c
Architecture graph (arch)	
$\|\mathbb{P}\|$	Nb of processing elements
$\|\mathbb{M}\|$	Nb of memory modules
Profiling data (profile)	
R_i	Mean number of firings of actor i
W^i	Total computation cost of actor i
Cs^i	Instruction code size of actor i

mapped randomly and are approximately equally distributed on all memory blocks. The manager informs by means of packets all involved PEs. For each PE in charge of executing actors, the manager generates and sends its corresponding mapping information in a separate packet (M_pIP). Packets holding the mapping information are stored in a predefined location in MIM. Then, the involved PEs are notified to retrieve their mapping information from the shared memory using notifying packets. At this stage, the manager waits the PEs, which are incorporated in processing a specific number of video frames N_F to send their monitoring information. Note that N_F is set originally to a default value and may be changed dynamically by the manager.

Before receiving the notification packet about initial mapping of actors, all PEs are in idle state. Once it receives the notification packet, the PE sends a request to retrieve the mapping information which includes IDs of actors to be executed, IDs of input and output FIFOs for each actor, and the reading order of each input FIFO. The mapped actors are scheduled according to the order sent from the manager and the PE begins to execute them in round-robin manner.

B. Monitoring actor execution

The execution of actors continues until receiving a new notification packet about changing the mapping information. All involved PEs monitor their running actors during the processing of N_F video frames, which determine the observation window. Precisely, each PE node accumulates for every mapped actor A_c its communication time $T_{cm}[A_c]$, computation time $T_{cp}[A_c]$, and total number of tokens received to each input port $NT_{total}[A_{c_{[I_j]}}]$ where $c \in \{1, ..., |\mathbb{A}|\}$ and $j \in \{1, ..., |\mathbb{I}_c|\}$. In addition, the adapter, which is embedded in the NI of each node n (processor or memory), extracts from each received packet carrying processed data, the following information for each source S_i: (1) the total number of transferred tokens from S_i to n: $NT^n_{total}[S_i, n]$ and (2) the average time delay consumed per token to reach the node n from source S_i: $T_{av}[S_i, n]$.Table IV gathers the variables and parameters used to formalize our mapping approach.

The total number of transferred tokens is simply determined. First, input packets are classified according to their sources S_i. Then, their corresponding sizes $size_{P_k}[S_i]$, which reflect the number of data-flits, are accumulated.

$$NT^n_{total}[S_i, n] = \sum_{k=1}^{\mathbb{K}} size_{P_k}[S_i, n] \tag{1}$$

where $i \in \{1, ..., |\mathbb{P}| + |\mathbb{M}|\}$.

The average time delay per token per each source $T_{av}[S_i, n]$ is calculated by dividing the time delay of each token transferred from S_i by $NT_{total}[S_i, n]$.

$$T_{av}[S_i, n] = \frac{\sum_{k=1}^{\mathbb{K}} size_{P_k}[S_i, n] \times D_{P_k}[S_i, n]}{NT^n_{total}[S_i, n]} \tag{2}$$

where $k \in \{1, ..., |\mathbb{K}|\}$.

$D_{P_k}[S_i]$ is determined by embedding, at the source node, for each packet P_k its sending time-stamp $T_s[P_k]$ in its header then subtracting it from the reception time $T_r[P_k]$ at the destination node. All tokens in a packet are considered to have the same delay.

$$D[P_k] = T_r[P_k] - T_s[P_k] \tag{3}$$

C. Collecting monitoring information

When the number of the processed frames meets the observed window, each PE node generates its own monitoring information packet. The packet is then sent to the MIM module (presented in III-C). Directly, the accumulated values are reset with the beginning of the new observation window. Then, the PE continues executing the previously mapped actors according to the adopted circular order. This guarantees that the remapping does not impose any additional overhead in terms of latency. The MIM module notifies in its turn the manager when new monitoring data is available corresponding to a specific processor throughout a notification packet (III-D1a). Whenever a new notification packet is received by the manager, the latter directly requests to retrieve the new available monitoring data. Also, the manager requests using command packets from all memory modules to send their monitoring information. Note that memory modules respond to the manager and send the requested data directly without any notification process since the adopted MoC allows the direct communication between a memory and a processor. All received monitoring packets are disassembled and their contents are parsed and saved in the manager local registers.

When the feedback data is collected from all modules incorporated in processing the video frames, the manager applies the run-time remapping algorithm. At this stage, the manager owns locally the following data: (1) the communication time of each actor: $T_{cm}[A_c]$, (2) the computation time of each actor: $T_{cp}[A_c]$, (3) the number of input tokens corresponding to every input port of all actors: $NT^a_{total}[A_{c_{[I_j]}}]$, (4) the number of incoming tokens to each processor module from each memory module m: $NT^n_{total}[S_m, p]$, (5) the average communication delay of received tokens to each processor p from each memory module m: $T_{av}[S_m, p]$, (6) the number of incoming tokens to each memory module m from each processor module p: $NT^n_{total}[S_p, m]$, and (7) the average communication delay of received tokens to each memory module m from each processor module p: $T_{av}[S_p, m]$ where

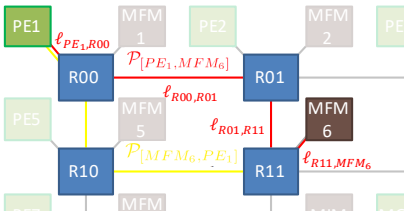


Fig. 8. Example on path declaration in the NoC

$c \in \{1, ..., |\mathbb{A}|\}$, $j \in \{1, ..., |\mathbb{I}_c|\}$, $m \in \{1, ..., |\mathbb{M}|\}$ and $p \in \{1, ..., |\mathbb{P}|\}$.

D. Estimating NoC communication time delay

Communication time delay is a critical factor in HMPSoC platforms using NoCs. The communication time of the moved actor is affected by the location of the new hosting PE in the network. NoC time-delay estimation impacts directly the prediction process of the communication time of the moved actor. Hence, the accuracy level in estimating the delay latency changes the decision on the actor move in the RR algorithm. In this work, two novel methods have been proposed to estimate the communication time delay for transferring one token in the NoC. The first method is called the average-path token delay and it is based on finding the average delay for transferring one token depending on the path delays between all nodes of the NoC. The second is called the average-link token delay and considers the time-delay of the token according to the used physical links connecting the NoC components while transferring the token. Both proposed methods make use of the monitoring data, which is collected while processing N_F video frames in the previous observation window. The techniques used in estimating the NoC communication time-delay are described in the following subsections.

1) Average-path token delay (APTD): In this approach, a path is considered to be formed from the set of the interconnections between two specific nodes. As an example, Fig. 8 illustrates in red the path $\mathcal{P}_{[PE_1,MFM_6]}$ between processing element PE_1 to memory module MFM_6. As a deterministic routing is applied in this work, the packets always use the same path between the source node and the destination node. Since the adopted MoC forbids the transfer of packets in between memory modules and in between PEs, the active paths are those connecting either memory modules to PEs or PEs to memory modules. Note that the packets transferred from a processing element p to a memory module m do not follow the same path used in transferring packets from the memory module m to the processing element p. Fig. 8 illustrates in red the followed path to transfer packets from PE_1 to MFM_6 and in yellow the followed path to transfer packets from MFM_6 to PE_1. In APTD, the manager calculates the average path delay per token T_{av} in several steps as shown in Algo. 1. T_{av} refers to the average time delay required to transfer one token from the source node to the destination node, regardless of the path between the source and destination nodes. As an example, the average time delay of all tokens transferred through either the path $\mathcal{P}_{[PE_1,MFM_6]}$ or the path $\mathcal{P}_{[MFM_6,PE_1]}$ (Fig. 8) is considered equal regardless of the number of links constituting each path and the corresponding traffic in each link and the switch conflicts in the connecting routers. T_{av} is computed by dividing the sum of the communication-time delays D_{total} by the total number of transferred tokens in the network NT_{total}:

$$T_{av} = \frac{D_{total}}{NT_{total}} \tag{4}$$

The manager benefits from the collected monitoring data. It makes use of the number of input tokens $NT^n_{total}[S_i, n]$ transferred to each destination node n from each source node S_i to determine the total number of all transferred tokens in the network (NT_{total}) as presented in (5):

$$NT_{total} = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} NT^n_{total}[S_i, n] \tag{5}$$

Also, the communication-time delays for all tokens transferred in the network are accumulated. The sum of the communication delays D_{total} is determined according to (6):

$$D_{total} = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} T_{av}[S_i, n] \times NT^n_{total}[S_i, n] \tag{6}$$

where $T_{av}[S_i, n]$ is the collected average time delay required to transfer one token from the source node S_i to the destination node n.

2) Average-link token delay (ALTD): A link is defined as the interconnection between two consecutive components of the NoC: Router, Memory and PE. As an example, Fig. 8 shows the links constituting the path $\mathcal{P}_{[PE_1,MFM_6]}$. In this approach, the average communication time delay per token is determined for each link as shown in Algo. 2. The total communication-time delay in a path $\mathcal{P}_{[S_i,n]}$ connecting the source node S_i and the destination node n is determined from the monitored data as shown in (7):

$$D^{\mathcal{P}}_{total}[S_i, n] = T_{av}[S_i, n] \times NT^n_{total}[S_i, n] \tag{7}$$

Each path is segmented into a set of links $\mathbb{L}_{\mathcal{P}_{[S_i,n]}}$. The average communication time delay per link $D^{\mathcal{L}}_{av}[S_i, n]$ in the path $\mathcal{P}_{[S_i,n]}$ is determined as follows:

$$D^{\mathcal{L}}_{av}[S_i, n] = \frac{D^{\mathcal{P}}_{total}[S_i, n]}{NL[S_i, n]} \tag{8}$$

where $NL[S_i, n]$ is the number of links constructing the path $\mathcal{P}_{[S_i,n]}$. Here, the links constructing a path are assumed to have similar contribution in the total communication time delay monitored in the path. As a link l is shared among different paths, the total link communication-time delay $D_{total}[l]$ is the sum of all average communication-time delay per link computed in all paths in which link l constitutes one of their interconnections:

$$D_{total}[l] = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} D^{\mathcal{L}}_{av}[S_i, n] \ni l \in \mathbb{L}_{\mathcal{P}_{[S_i,n]}} \tag{9}$$

Algorithm 1 Average-path token delay (APTD)

Step 1: *Find the sum of the communication delays* D_{total}
Step 2: *Find the total number of all tokens* NT_{total}
Step 3: *Calculate the average time delay per token* T_{av}

Algorithm 2 Average-link token delay (ALTD)

Step 1:
for each path $\mathcal{P}_{[S_i,n]}$ **do**
a- *Find the total communication-time delay* $D^{\mathcal{P}}_{total}[S_i,n]$
b- *Calculate average communication time delay per link* $D^{\mathcal{L}}_{av}[S_i,n]$
end for
Step 2:
for each link l **do**
a- *Find the total link communication-time delay* $D_{total}[l]$
b- *Find the total number of tokens* $NT_{total}[l]$
b- *Calculate the average communication time delay per token* $T_{av}[l]$
end for

On the other hand, the tokens passing through a path are definitely passing through all links constructing the path. Hence, the total number of tokens $NT_{total}[l]$ passing through a link l is the sum of all tokens passing through all paths, which link l constitutes one of their interconnections:

$$NT_{total}[l] = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} NT_{total}[S_i,n] \ni l \in \mathbb{L}_{\mathcal{P}_{[S_i,n]}} \tag{10}$$

The average communication-time delay per token $T_{av}[l]$ for each link l is determined by dividing the accumulated communication-time delay $D_{total}[l]$ by the number of tokens $NT_{total}[l]$ passing through this link.

$$T_{av}[l] = \frac{D_{total}[l]}{NT_{total}[l]} \tag{11}$$

E. Applying RR algorithm

For each observation window (N_F frames), the manager executes at run-time the RR algorithm, which is divided into two main steps. The first step is dedicated to find all possible candidate actors which their moves would enhance the overall throughput. The second step sets a tradeoff between the cost of migration and the predicted improvement of the performance.

1) Specify the possible candidate actors: In this work, the definitions of the terms period of each processor p ($Period_p$), maximum period ($Period_{max}$) and throughput (Th) have been adopted as introduced in [17]. $Period_p$ is the sum of total computation time $compT_p$ and total communication time $commT_p$ recorded during N_F video frames:

$$Period_p = compT_p + commT_p \qquad \forall p \in \mathbb{P} \tag{12}$$

where $compT_p$ and $commT_p$ of processor p are the sums of the computation times and of the communication times respectively of all actors which are mapped on this processor:

$$compT_p = \sum_{k:\mathbb{P}[k]=p} T_{cp}[A_k] \qquad \forall p \in \mathbb{P} \tag{13}$$

$$commT_p = \sum_{k:\mathbb{P}[k]=p} T_{cm}[A_k] \qquad \forall p \in \mathbb{P} \tag{14}$$

The throughput is defined as the inverse of the maximum period over all processors.

Hence, the first task is to find the PE with the maximum period. The manager computes the periods of all PEs during the current observation window of N_F video frames. Later, a simple comparison between all obtained period values is performed in order to specify the processor with the maximum period. The processor with the maximum period ($Period_{max}$) is nominated as *looser* processor. The algorithm used to determine the *looser* processor is outlined in Algo. 3. The set of candidate actors to be moved $\mathbb{C}$ includes the actors that have been previously executed by the *looser* processor. Fig. 9 demonstrates an example of $Period_p$ and $Period_{max}$. The figure shows three PEs (PE_1, PE_2 and PE_3) that run six actors (A_1, A_2, A_3, A_4, A_5 and A_6). In this example, PE_1 has the largest period, thus it is selected as the *looser* processor.

Algorithm 3 Finding processor with maximum period

$Period_{max} \leftarrow 0$
$looser \leftarrow \phi$
for $p \in \mathbb{P}$ **do**
if $Period_{max} < Period_p$ **then**
$Period_{max} \leftarrow Period_p$
$looser \leftarrow p$
end if
end for

2) Decision of the actor move: The actor selected to be moved should have a maximum total gain. According to the collected monitoring values, the manager estimates the total gain achieved for all combinations of mapping the actors which belongs to the candidate list $\mathbb{C}$ onto all available PEs. The estimated total gain $Gain^e_{total}[C_{A_c,p}]$ of a mapping combination $C_{A_c,p}$, which corresponds to moving A_c to p, is computed by finding the difference between the estimated performance gain $Gain^e_{per}[C_{A_c,p}]$ and the estimated migration cost of the actor $Cost^e_{mig}[C_{A_c,p}]$. The mapping combination that leads to the maximum estimated total gain is then selected. The engaged processor and actor are specified and so-called the *gainer* processor and *moved-actor* respectively.

a) Estimated performance gain: For each actor A_c in the candidate list $\mathbb{C}$, the manager considers it is moved virtually to all PEs except the *looser* processor. For each virtual-move combination, the manager estimates the achieved period of

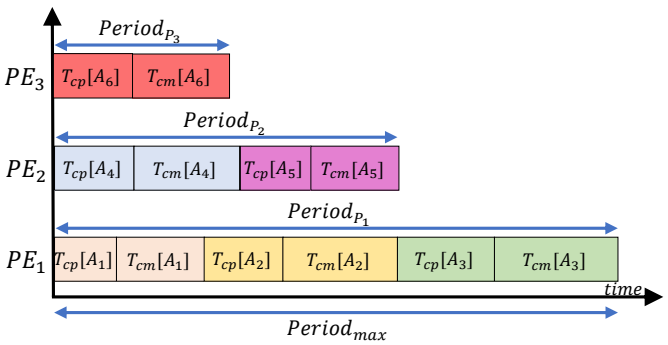


Fig. 9. An example of $Period_p$ and $Period_{max}$

each processing element $Period^e_p[C_{A_c,p}]$. The new period of processor p is estimated by adding to the processor period $Period_p$ the estimated communication time $T^e_{cm}[C_{A_c,p}]$ and the estimated computation time $T^e_{cp}[C_{A_c,p}]$ of the moved actor A_c as shown in the following expression:

$$Period^e_p[C_{A_c,p}] = Period_p + T^e_{cp}[C_{A_c,p}] + T^e_{cm}[C_{A_c,p}] \quad (15)$$

Note that the tokens, which are consumed by a certain reader actor running on a processing element PE_R, are imported from a FIFO f. These tokens are previously generated by another actor running on another processing element PE_W. The generated tokens are first stored in a FIFO f and then transferred once requested to the processing element PE_R where the reader actor is executed. Hence, the tokens pass through two paths. The first path $\mathcal{P}_{[PE_W,MFM_f]}$ connects the processing element PE_W, which executes the writer actor, and the memory module that accommodates the FIFO f. On the other hand, the second path $\mathcal{P}_{[MFM_f,PE_R]}$ connects the memory module that accommodates the FIFO f and the processing element PE_R which executes the reader actor. The communication-time delays in both paths are considered when estimating the communication time of the moved actor.

When adopting APTD method for determining the communication delay in the NoC, the estimated communication-time delay per input j for each actor A_c is equal to the total number of input tokens $NT_{total}S[A_{c_{[I_j]}}]$ transferred to the actor at this input multiplied by the double of the calculated average path communication-time delay per token T_{av} (4). The average path delay per token is doubled to compensate the time delay of the two paths $\mathcal{P}_{[PE_W,MFM_f]}$ and $\mathcal{P}_{[MFM_f,PE_R]}$. The total estimated communication time is the sum of all estimated communication-time delays of all inputs:

$$T^e_{cm}[C_{A_c,p}] = \sum_{j=1}^{|\mathbb{I}_c|} 2 \times T_{av} \times NT_{total}[A_{c_{[I_j]}}] \quad (16)$$

Note that the adopted model of computation forbids the transfer of tokens in between actors (running on PEs) directly without passing through a FIFO (allocated in a memory module MFM_f). Hence, tokens produced by the writer actor (running on PE_W) will pass through two paths ($\mathcal{P}_{[PE_W,MFM_f]}$ and $\mathcal{P}_{[MFM_f,PE_R]}$) before arriving to the reader actor (running on PE_R). The exact number of tokens passes through both paths while considering same average path delay per token T_{av}. So, the average path delay per token is doubled in (16).

When adopting ALTD method, the estimated communication-time per input j is equal to the total number of input tokens $NT_{total}[A_{c_{[I_j]}}]$ transferred to the actor A_c through this input multiplied by the sum of all average communication-time delay per token $T_{av}[l]$ for each link l constructing the paths which the input tokens use to reach the processing element running the actor A_c. The total estimated communication time will be the sum of all estimated communication-time delays of all inputs:

$$T^e_{cm}[A_c] = \sum_{j=1}^{|\mathbb{I}_c|} \left(\sum_{i=1} T_{av}[l_i] \right) \times NT_{total}[A_{c_{[I_j]}}] \quad \ni l_i \in \left\{ \mathbb{L}_{\mathcal{P}_{[PE_W,MFM_f]}} \cup \mathbb{L}_{\mathcal{P}_{[MFM_f,PE_R]}} \right\} \quad (17)$$

In addition, the estimated computation time $T^e_{cp}[C_{A_c,p}]$ of the moved actor A_c is determined depending on the recorded computation time of the moved actor A_c during the previous mapping $T_{cp}[A_c]$ and the estimated total speed-up ratio $SU^e_{total}[C_{A_c,p}]$, which is achieved when moving A_c to p:

$$T^e_{cp}[C_{A_c,p}] = T_{cp}[A_c] \times SU^e_{total}[C_{A_c,p}] \quad (18)$$

such that

$$SU^e_{total}[C_{A_c,p}] = \frac{\mathcal{A}_{A_c}[p]}{\mathcal{A}_{A_c}[looser]} \times \frac{f[looser]}{f[p]} \quad (19)$$

where $f[p]$ is the operating frequency of processor p (Table II) and $\mathcal{A}_{A_c}[p]$ is the acceleration enhancement ratio of the moved actor A_c when running on processor p (Table I).

Note that for all mapping combinations, the period of the *looser* processor is modified when an actor A_c is supposed to be mapped to another processor p. Hence, it is updated by subtracting the actual communication time $T_{cm}[A_c]$ and the actual computation time $T_{cp}[A_c]$ of the moved actor A_c:

$$Period^e_{looser}[C_{A_c,p}] = Period_{max} - T_{cm}[A_c] - T_{cp}[A_c] \quad (20)$$

For each mapping combination, the manager determines the maximum estimated period $Period^e_{max}[C_{A_c,p}]$ which denotes the maximum period among all processors when actor A_c is mapped to processor p. Fig. 10 demonstrates an example of finding $Period^e_{max}[C_{A_c,p}]$. The figure considers the example illustrated in Fig. 9. Three actors are mapped to the *looser* processor PE_1. The candidate list $\mathbb{C}$ includes three actors: A_1, A_2 and A_3. Six mapping combinations are illustrated: C_{A_1,PE_2}, C_{A_1,PE_3}, C_{A_2,PE_2}, C_{A_2,PE_3}, C_{A_3,PE_2} and C_{A_3,PE_3}. The figure shows how to find the maximum estimated period $Period^e_{max}[C_{A_c,p}]$ for each mapping combination. It is shown in the figure that both the estimated communication time and estimated computation time of the same actor differ when mapped to different PEs.

These computed new periods are then used to find the performance gain related to each mapping combination:

$$Gain^e_{per}[C_{A_c,p}] = Period^e_p[C_{A_c,p}] - Period_{max} \quad (21)$$

b) Estimated migration cost: The migration cost of an actor is the required time to transfer its binary code into the local memory of the new hosting processing element. It depends on the size of the binary data required to be transferred and the communication-time delay in the network. The sizes of the binary codes of all actors are considered to be known by the manager in terms of number of flits. Accordingly, the migration cost of the moved actor is estimated by the manager using the estimated NoC communication-time delay. When adopting APTD method, the estimated migration cost related to the moving of actor A_c to processor p is calculated as expressed in (22):

$$Cost^e_{mig}[C_{A_c,p}] = size_{bin}[A_c] \times T_{av} \quad (22)$$

where $size_{bin}[A_c]$ is the size of the binary code of actor A_c and T_{av} is the average path communication-time delay per token (4). When ALTD method is adopted, the migration cost of the moved actor A_c is determined by (23):

$$Cost^e_{mig}[C_{A_c,p}] = size_{bin}[A_c] \times \left(\sum_{i=1} T_{av}[l_i] \right) \quad \ni l_i \in \mathbb{L}_{\mathcal{P}_{[BCM,p]}} \quad (23)$$

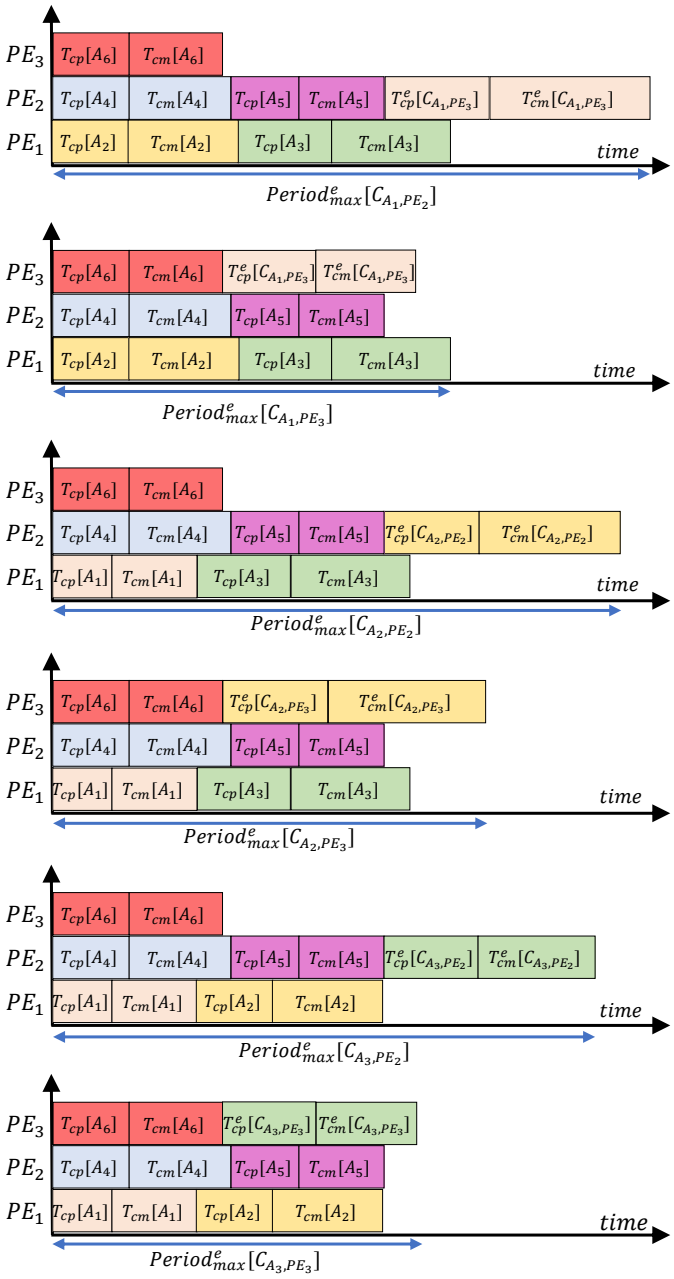


Fig. 10. An example of finding the maximum periods for each mapping combination

c) Estimated total gain: The manager computes the total gain estimated to be achieved for all mapping combinations by finding the difference between the estimated performance gain $Gain^e_{per}[C_{A_c,p}]$ and the estimated migration cost of the actor $Cost^e_{mig}[C_{A_c,p}]$.

$$Gain^e_{total}[C_{A_c,p}] = Gain^e_{per}[C_{A_c,p}] - Cost^e_{mig}[C_{A_c,p}] \quad (24)$$

The moving of an actor would lead to permanent performance gain and the migration cost is paid once. However, the estimated performance gain takes the cost of migration into account in order to aggravate the probability of enhancing the overall performance directly after applying the move (in the next observation window). In fact, the variation of the input data and its corresponding effects on executing the involved actors incites to consider worst case (severe) decision where the performance enhancement should be guaranteed once moving the actor.

Then, the manager finds the maximum achieved total gain among all mapping combinations and accordingly specifies the actor to be moved and the *gainer* processing element.

F. Moving the actor to the gainer processor

The PE, after finishing the execution of the current running actor, retrieves the new mapping information and sends directly a confirmation packet so that the manager processor manages the transfer of the object code corresponding to the new mapped actor. Before running the moved actor, the PE checks the availability of the object file corresponding to the actor in its cache memory. Note that for the initial mapping, the manager generates and sends packets to all PEs in charge of executing actors. Whereas, after executing the RR algorithm, the manager informs only the *gainer* and *looser* processors. This procedure reduces the traffic in the network and maintain the processing performance since the PEs that are not affected by remapping process are not disturbed. In fact, the manager informs first the *looser* processor about the new mapping information. Then, it waits until the *looser* processor confirms the well reception. The *looser* processor sends a confirmation packet to the manager whenever it finishes the execution of the moved actor. When the manager receives the confirmation packet, it sends the new mapping information to the *gainer* processor. Later, the *gainer* sends a confirmation packet to the manager that directly manages the transferring of the object code of the mapped actor from the shared memory into the cache memory of the *gainer* processor by making use of BCPs described in subsection III-D2b. This guarantees that the actor is executed by only one PE in the whole platform and ensure better controlling of the traffic while migrating the binary codes. In fact, the manager sends a CTP (subsection III-D1h) which includes the ID of the *gainer* processor, the ID of the moved actor and the size of the BCPs (capacity) as described in subsection III-C1. After receiving the CTP, the MAM module, which is integrated into the NI of the BCM (subsection III-C1), manages retrieving the binary code from the shared memory and dividing it into sections according to the capacity specified by the manager. The generated BCPs will be transferred to gainer processor. In our work, we consider that the gainer processor can start executing the actor once at least 256 bytes, which construct 8 lines of the L1-I cache, are received and stored to the *gainer* processor local memory. The hierarchy of the PEs' local memories includes L1 and L2 caches. L1 cache is broken up into to halves, instruction (L1-I) and data (L1-D) each of 32KB. L2 cache size is of 256KB and is used for instructions and data.

G. RR Algorithm Complexity

The devised algorithm consists of several steps summarized in Algo. 4. The complexity of each step is illustrated to determine the overall complexity. The complexity of Step1, the step of finding the period of each processor, is $O(|\mathbb{P}|)$.

Algorithm 4 Run-time Remapping (RR)

Step 1: *Calculate the period of each PE*
Step 2: *Find PE with Max. period and assign it as looser*
Step 3: *Find the total gain (performance - migration cost)*
for each move **do**
  a- *Find the performance gain*
    . find the period for each PE
    . find the maximum period
  b- *Find the migration cost*
  c- *Calculate the total gain*
end for
Step4: *Choose the move with Max. positive total gain*

Then, Step2, the step of finding the processor with maximum period has the complexity of $O(|\mathbb{P}|)$. The complexity of Step3, estimating the total gains corresponding the move of the candidate actors to all PEs rather than the looser processor, is $O((|\mathbb{P}|-1).|A_c \in \mathbb{C}|)$. The complexity of Step4, choosing the best move, is $O(|A_c \in \mathbb{C}|.(|\mathbb{P}|-1))$. If we consider a well balanced distribution of actors among the processors at initiation ($|A_c \in \mathbb{C}| \approx \frac{|\mathbb{A}|}{|\mathbb{P}|}$), the overall complexity becomes $O(|\mathbb{P}|) + O(|\mathbb{A}|)$ knowing that $\frac{|\mathbb{P}|-1}{|\mathbb{P}|} \approx 1$. Note that the algorithm is computed when all monitoring data is collected, so the maximum rate is once per execution of the whole data flow, and in practice can be tuned to be slower. With respect to the complexity and the execution rates of actors, this complexity is extremely low.

V. Experiments and Results

A. Application Model

In this work we target the multimedia application domain. We adopt the well-known MPEG4 part 2 Simple Profile video decoder (MPEG4-SP). This multimedia application is typically used in de-compression of encoded video digital data. Fig. 11 presents the structure of decoder as described in Reconfigurable Video Coding framework (RVC) [3] [33].

MPEG4-SP is specified with heterogeneous dataflow MoCs and includes up to 40% of dynamic actors [34]. It is composed of 41 actors and 70 FIFOs specified in RVC-CAL language. The ORCC tool is utilized for compiling and software synthesis [3] and we make use of the generated C-code for multi-core platforms. We also use the structure of the software FIFO presented in Fig. 1-b), which is generated by ORCC.

A FIFO may have several reader actors but only one writer actor. It opts an indexing mechanism such that a specific index is assigned to each reader or writer actor. These indexes are used to determine the number of available tokens corresponding to each reader actor and the free space in a FIFO. The number of available tokens ($T_f[R_i]$) in a FIFO (f) is the difference between the reader index ($I_f[R_i]$) and the writer index ($I_f[W]$). The free space in a FIFO is the number of memory addresses that contain no more needed data from all reader actors. In other words, it is the subtraction of the maximum available tokens from the total FIFO size ($Size_f$).

Each actor has its input and output ports and includes one or several actions. An action describes a specific functionality

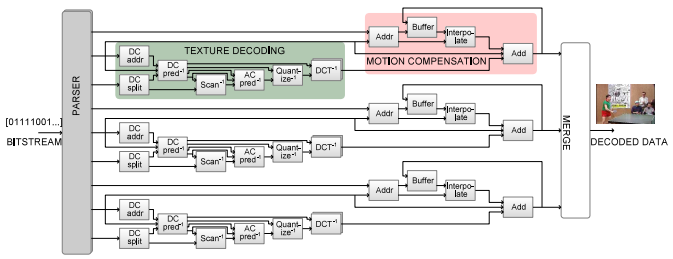


Fig. 11. MPEG4 part 2 SP decoder [33]

and is executed (fired) when a set of conditions, so-called firing rules, are satisfied. As an example, a firing rule consists of checking if the number of available tokens in the input FIFO is greater than the required number for computation, and that the output FIFO has sufficient empty room to store the produced tokens. In MPEG4-SP, the number of reader actors ranges from 1 (at least) to 6 (at most).

MPEG RVC defines RVC-CAL applications as dynamic dataflow applications, where the uncertainty of computing due to data-dependency prevents from any static scheduling. They are based on dataflow process network (DPN) model [6]. In such model, the actor executes when at least one of its firing rules is satisfied. For cases where several firing rules are satisfied simultaneously, only one is selected according to its priority. Consequently, its corresponding satisfied action is fired. Each firing consumes input tokens and produces output tokens. The number of the consumed or produced tokens may be fixed or variable.

B. Experimental framework and setup

In order to assess the feasibility of our proposed run-time remapping method, we developed a real-time simulator. The simulator is described in SystemC TLM model [35]. The devised simulator models a MPSoC platform using NoC concept for interconnecting embedded modules. The platform incorporates heterogeneous processing elements (Table I), memory blocks, and the manager. The simulator platform has been designed with hierarchical modules that can work concurrently and intercommunicate via ports using simple or complex communication channels. SystemC features have been exploited to mimic the accurate functionality of the modules described in section III.

The adopted NoC-based architecture, presented in section III, is implemented in the devised simulator platform. In order to accurately model the adopted application, all involved actions are functionally simulated to determine their execution-timing features and generate the real data exchanged by actors during video decoding. The SystemC model adopted in the simulation platform is cycle accurate at the level of the NoC and the network interfaces. The timing of all corresponding action executions on PE is compensated in the simulation according to the profiling data extracted while running the application on a reference computer. Profiling data provides, for each involved action, the mean value of the number of cycles required to execute it. In this work, profiling data has been extracted using on a desktop computer (i7-2620M

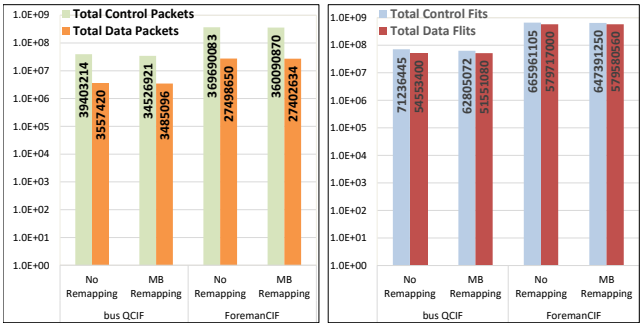


Fig. 12. Classification of transported packets and flits

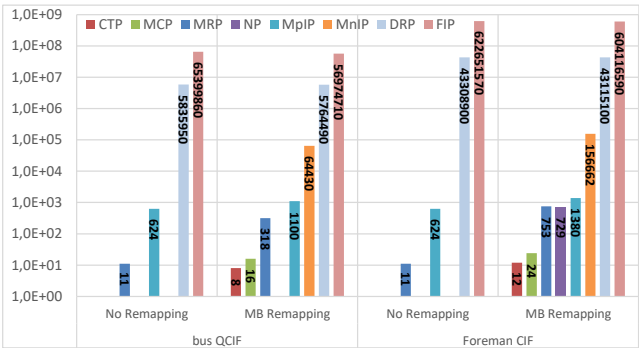


Fig. 13. Classification of control flits according to their types

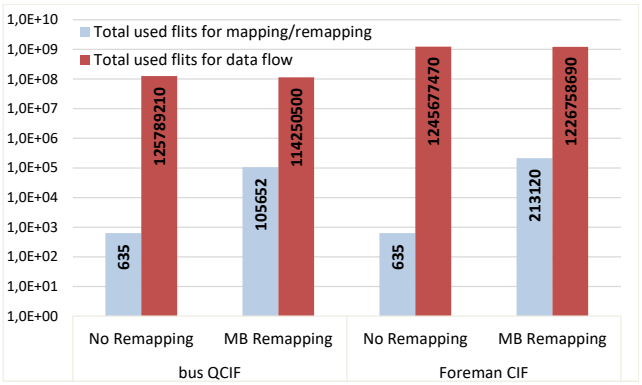


Fig. 14. Number of transported flits

CPU@2.7 GHz and 8GB memory). We consider that the NoC operating frequency f is 500 MHz. The clock cycle in each PE is determined according to Table II. During SystemC simulations, for each fired action, its corresponding execution time determined in profiling is mapped according to the processor frequency and used as time delay to compensate the real execution time. In addition, several benchmark video sequences with different formats from [36] have been encoded. The selected video sequences have different manner in changes between successive frames. This guarantees to evaluate the performance of the proposed algorithm for different data-dependent behaviors. The resultant data has been used as input to the decoder. These same encoded videos have been decoded on a desktop computer and the FIFO contents have been traced over the decoding period. To verify the proper functionality of each actor, the contents stored in the FIFOs in the simulator have been compared to the traced FIFO data. Also, the output data of the simulator have been reconstructed into visual video in order to verify the functionality of the devised simulator. The video sequences have been decoded without applying remapping targeting the same NoC-base architecture and the obtained results have been compared to that obtained when the video sequences are decoded adopting the MB remapping algorithm applying ALTD and APTD for estimating the communication time delay while considering an observation window of $N_f = 10$.

C. Experimental Results

1) Transported data: The number of packets that travel through the network during the decoding of the video sequences, and their corresponding flits are recorded in the case of applying the MB remapping and the case of decoding the video without remapping. Fig. 12 presents the number of transported packets and flits in logarithmic scale during decoding the Foreman video with CIF format and Bus video with QCIF format for the case of adopting MB remapping algorithm and the case of ordinary decoding. The packets and flits are classified into control and data categories. The figure shows that the flits of control packets form about 53% of all transported flits in the two cases.

Furthermore, investigating thoroughly the types of transported control flits illustrates that 93% of control flits belong to FIP. This refers to the MoC adopted in dataflow applications which requires checking the firing rules (availability of input data and output buffer space). Fig. 13 shows in logarithmic scale the number of each type of control flits transported while decoding the Foreman video sequence in CIF format and Bus video sequence in QCIF format for the case of MB remapping and the case of ordinary decoding.

Also, Fig. 13 shows that additional flits are transported in the network due to the remapping. In fact, applying remapping induces additional control and data packets. In order to evaluate the effect of applying the MB remapping algorithm on the traffic in the network, the transported flits are classified into two main categories. The first category includes the flits which are used basically for dataflow. This category encompasses the flits which occupy the payload of all FIP, DRP and DFP. The second category includes the induced flits by applying the remapping algorithm. Hence, the second category compromises the flits listed in the payloads of NP, MRP, MCP, M$_n$IP, M$_p$IP, CTP, and BCP. Note that both categories include data and control packets. Fig. 14 illustrates the comparison summary in terms of the number of transported flits of both categories. In the figure, the number of transported flits, which is obtained while processing the Foreman video with CIF format and Bus video with QCIF format, is presented in logarithmic scale for both cases (decoding while applying remapping algorithm and ordinary decoding). The comparison shows that the additional flits induced by applying the MB algorithm forms less than 0.02% from total transported flits. In addition, Table V shows the percentage of flits transporting the binary code of migrated actors from the total number of transported flits in the network while decoding several video

TABLE V
PERCENTAGE OF FLITS TRANSPORTED IN BCP FROM TOTAL FLITS

Video		Remapping Algorithm	
Sequence	Format	MB-ALTD	MB-APTD
Foreman	CIF	0.0044%	0.0067%
Bus	CIF	0.0008%	0.0125%
Ice	4CIF	0.0027%	0.0019%
Bus	QCIF	0.0348%	0.0272%

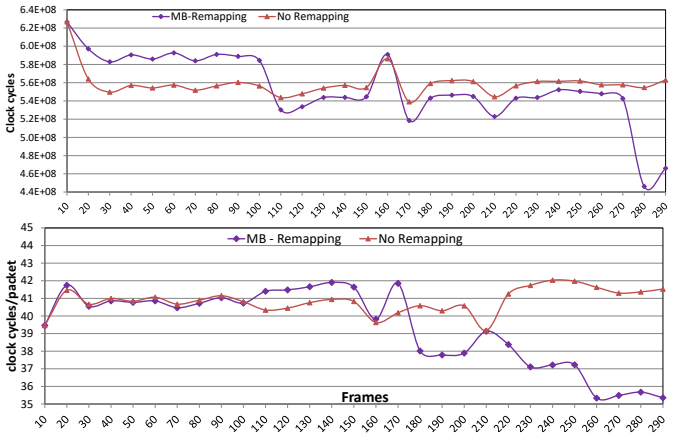


Fig. 15. Sum of packet time-delays (top) and average packet time-delay (bottom) while decoding Foreman video with CIF format [36]

sequences. The presented percentages illustrate that the impact of actor migration on the traffic is negligible.

2) Packet time-delay: The packet time-delay is recorded while decoding the video sequences, following the procedure explained in subsection IV-B. Fig. 15 presents the variation of the sum of packet time-delays throughout the observing windows during the decoding of the Foreman video sequence with CIF format when adopting the MB remapping technique. It is noticed that applying the MB remapping algorithm affects the time-delay of the packets. In addition, the figure shows the comparison with the case of ordinary decoding. The comparison illustrates that using MB remapping decreases gradually the total packet time-delay. Note that the task moves occur after processing 80, 100, 160, and 270 frames. Fig. 15 shows that the total packet delay decreases after the conducted moves. This refers to the fact that task remapping contributes in distributing the tasks on PEs that are nearer to the memory modules accommodating the input and output FIFOs. Also, Fig. 15 presents a comparison in terms of average time-delay of packets transported during the decoding of the Foreman video with CIF format when adopting the MB remapping technique and when using ordinary decoding. The comparison confirms that the use of MB remapping technique contributes significantly in reducing the time-delay.

3) Timings: Fig. 16 presents the recorded total communication time and total computational time throughout the observing windows during the decoding of the Foreman video with CIF format when adopting the MB remapping technique and when using ordinary decoding. It shows that the communication time represents 90% of the total execution time in both cases. Hence, the total execution time is affected more by the variation of the total communication time. Also, Fig. 16(a) shows that the total communication time is almost not changing among observation windows in the case of ordinary decoding. Whereas, when MB technique is applied, the communication time varies significantly and tends to follow a decreasing manner as shown in Fig. 16(b). This illustrates that reducing the time-delay achieved by MB remapping has a direct impact on the communication time.

The communication time of each processing element is investigated through the decoding of all video frames. It is noticed that when applying the MB remapping technique, the variation between communication times of all involved PEs is reduced. The communication time values of all PEs converges gradually to a specific interval as shown in Fig. 17.

4) Performance results: Multiple simulations have been conducted to decode several benchmark video sequences from [36]. Fig. 18(a) presents the achieved throughput in terms of frames per second (FPS) when decoding Foreman video (CIF format) and using ALTD and APTD respectively for estimating the NoC communication time delay. The figure also shows the achieved throughput when decoding the Foreman video (CIF format) without remapping. The letter "M" shown on the curves represents when an actor move occurs. Fig. 18(a) shows that using MB results in significant performance enhancement. In addition, the figure illustrates that adopting ALTD for estimating the NoC communication time delay, while decoding Foreman video sequence with CIF format, increases the achieved enhancement ratio. Other similar simulations have been conducted targeting other video sequences with different formats (CIF, 4CIF and QCIF). The selected videos are of diverse characteristics to ensure that the proposed remapping algorithm is not related to specific formats or video content. The obtained results confirm that adopting MB algorithm ensures enhanced performance when compared to decoding the video without remapping. Also, the results demonstrate that adopting ALTD rather than APTD leads to additional performance enhancement.

D. Discussion and Comparison

In order to determine the relevancy of the devised algorithm, it is compared to the STM method introduced in [31]. To achieve fair comparison, the STM method has been modeled and implemented on our devised NoC-based architecture. We have also implemented the exact method presented in [20] for the initial mapping, with two differences: we have used constraint programming instead of ILP, and the objective function is the maximum period, Eqn. 12, as it is our optimization goal. The workload used for the computation time of the actors is based on the profiling of Foreman video. Simulations have been conducted while running the MPEG4 decoder to process real-life videos.

1) Performance enhancement of MB remapping: The results presented in Fig. 18 show that for Foreman video sequence with CIF format (Fig. 18(a)), the use of MB remapping algorithm when adopting ALTD leads to a maximum performance enhancement of 38.2% (frame 280) and adopting MB-APTD leads to a maximum performance enhancement of 14.8% (frame 210) when compared to the results of processing

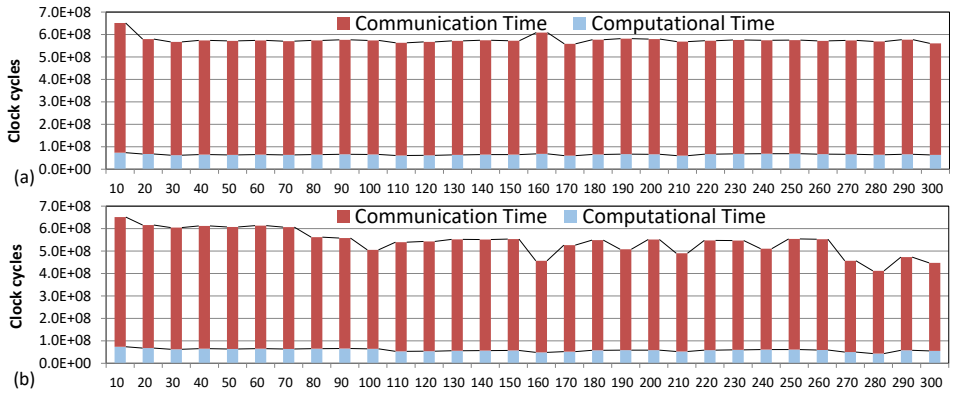


Fig. 16. Total communication and computational times recorded throughout the observing windows during the decoding of the Foreman video with CIF format [36]; when adopting (a) ordinary decoding and (b) MB remapping

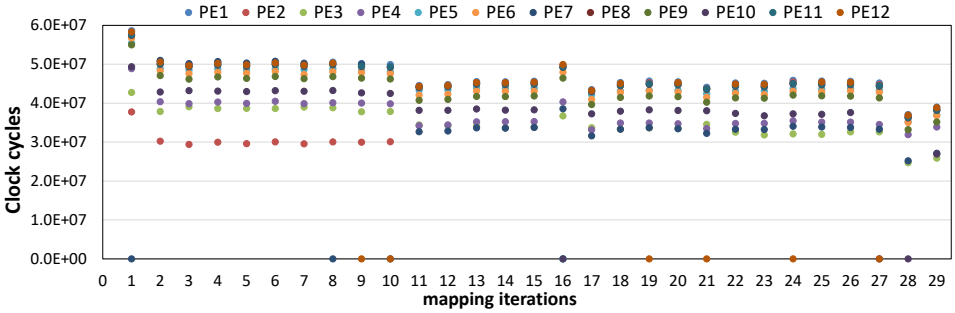


Fig. 17. PE communication time in terms of FPS of decoding Foreman video with CIF format [36] using MB remapping algorithm

the video without remapping. For Ice video sequence with 4CIF format (Fig. 18(b)), maximum performance enhancement of 56% (frame 450) and 16.5% (frame 250) are recorded when applying the MB algorithm adopting ALTD and APTD respectively. Furthermore, the use of MB algorithm adopting ALTD and APTD leads to a maximum performance enhancement of 10.92% (frame 120) and 7.6% (frame 50) for Bus video sequence with QCIF format (Fig. 18(c)) and Bus video with CIF format (Fig. 18(d)). For Grandma video with QCIF format (Fig. 18(e)), maximum performance enhancement of 33.2% (frame 170) and 23.9% (frame 190) are recorded when applying the MB algorithm adopting ALTD and APTD respectively. The simulation results show that the link level estimation of ALTD is more accurate and usually leads to better performance compared to APTD. However, in some cases APTD performs better such as for some observation windows of Grandma video (Fig. 18(e)). This refers to the fact that the heuristic is data-dependent and the link level prediction depends on the monitoring information collected during the previous data which may not match with the that of the current processed data.

2) MB remapping in comparison to STM remapping: Fig. 18 shows a comparison between our proposed remapping and the STM algorithm in terms of throughput (FPS). The results shows that the MB remapping outperforms STM remapping technique when considering either APTD or ALTD for estimating the NoC communication time delay.

Besides, the graphs in Fig. 18 show that in some cases the STM method leads to deterioration in the performance. In fact, the STM method selects critical task to be moved in each observation window without estimating the resulting total performance gain. Moving the task without determining its effects on the whole system performance degrades the overall performance. While in our proposed algorithm, the maximum achieved total gain among all mapping combinations is first determined as explained in subsection IV-E2c. Accordingly, a task is specified to be moved if the estimated maximum total gain is positive. It is noticed that in some observation windows no tasks are moved when the proposed algorithm is applied. A move is indicated by letter "M" in Fig. 18(a). In these cases, the estimation shows that no performance enhancement will be achieved for all mapping combinations.

3) MB remapping in comparison to optimal mapping: Fig. 18 also shows the results obtained from the mapping approach proposed in [20]. Note that the "optimal" mapping corresponds to the best mapping found based on the profiling of Foreman video after a time out of one hour (like the original paper), and the optimality is not proven. The results show that the MB algorithm, starting from a random mapping (without significant initial delay), performs better that the optimal with no remapping for Foreman video sequence in CIF format (Fig. 18(a)). As the optimality is searched for the Foreman profile, we used the optimal mapping as a starting point for the MB algorithm, and the results show that it further improves the throughput. As expected, the optimal mapping for Foreman does not perform good for the Ice video sequence in 4CIF

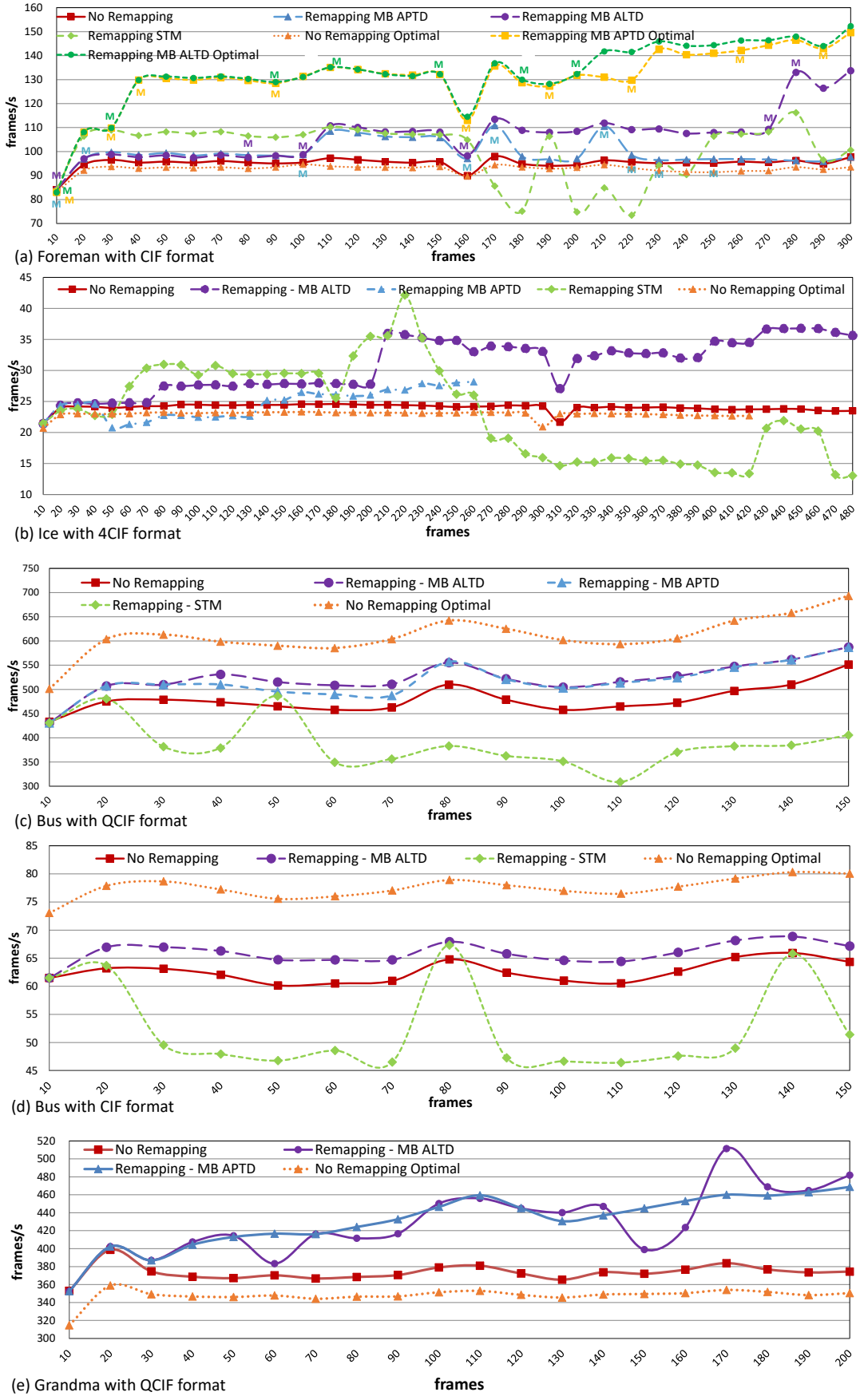


Fig. 18. Throughput in terms of FPS when decoding video sequences [36] using MB and STM remapping algorithms

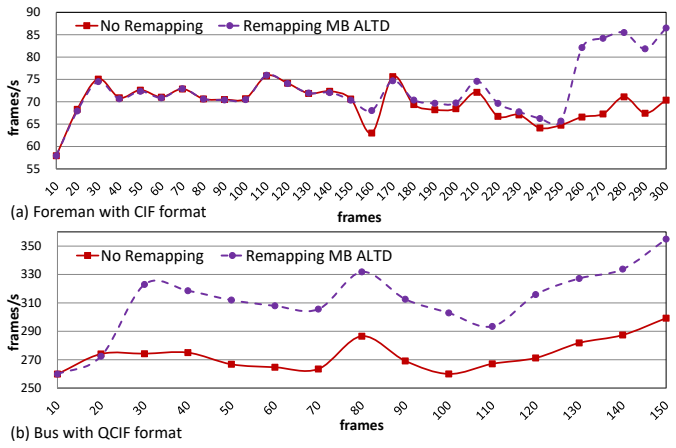


Fig. 19. Throughput in terms of FPS when decoding video sequences [36] using MB remapping algorithms targeting 4×6 NoC

TABLE VI
ACHIEVED RESULTS ADOPTING DIFFERENT REMAPPING TECHNIQUES

Video		Remapping Algorithm		
Sequence	Format	MB-ALTD	MB-APTD	STM
Foreman	CIF	11.4%	5%	4.1%
Bus	CIF	5.4%	5.4%	−17.7%
Ice	4CIF	26.1%	2%	−13.04%
Bus	QCIF	9%	8%	−20%
Grandma	QCIF	14.91%	14.11%	NA

format (Fig. 18(b)) and Grandma video sequence in QCIF format (Fig. 18(e)). But surprisingly, it performs good for the Bus video in QCIF format (Fig. 18(c)) and Bus video in CIF format (Fig. 18(d)). The so-called optimal method cannot be used for two reasons. First it introduces an unpractical initialization delay without guaranty of optimality. Secondly, a static solution is not appropriate to data-dependent applications since a solution can be good for one data-stream and inefficient for another one and more importantly the efficiency of a mapping varies over time.

4) Comparison summary: Table VI summarizes the comparison of average FPS achieved when processing multitude video sequencing while adopting different remapping techniques. The table shows that the MB algorithm achieves the maximum average performance enhancements of 26% and 14.11% when adopting ALTD and APTD respectively compared to the achieved throughput of processing the frames without remapping. Whereas, remapping using STM algorithm achieves a maximum average enhancement of 4%.

E. Scalability and generality

The scalability of our approach relies first on a negligible extra payload in the context of actor-level dataflow models, which intrinsically require a large amount of small control packets. For example, when decoding the Foreman video sequence the extra flits imposed by remapping (including the flits holding the binary codes of moved actors) constitute less than 0.02% of the flits used for dataflow. The proposed remapping method enhances the performance by exploiting the NoC structure and the characteristics of the available resources.

TABLE VII
REDUCTION OF PACKET HOPS WITH MB-ALTD AND MB-APTD

Video		Remapping Algorithm	
Sequence	Format	MB-ALTD	MB-APTD
Foreman	CIF	20.94%	12.64%
Bus	QCIF	3.24%	5.29%
Grandma	QCIF	14.18%	8.33%

The results show that our method positively impacts the NoC performance. Table VII illustrates the reduction percentages of packet hops when decoding different video sequences adopting the proposed MB remapping compared to ordinary decoding without remapping. The comparison shows that the proposed remapping method reduces the packet hops. The percentage of reduction is more than 20%. Secondly, the method includes the migration cost and so limits the number of moves.

Fig. 19 shows the results obtained for a 4×6 NoC, for Foreman and Bus video sequences, starting from a random mapping. The results show that our approach can also improve the throughout for a larger NoC. On average, the throughout is improved by 13.5% and 4% for Bus QCIF and Foreman CIF videos respectively.

VI. CONCLUSION

This paper presents an original *Move*-based algorithm and NoC-based architecture to map the tasks of dataflow application during run-time. The method monitors the performance and intercommunication, takes the proper mapping decision and applies the required mapping configurations. The algorithm and the devised architecture are thoroughly presented. The best way to verify the effectiveness of a run-time mapping, which is by definition data dependent, is to simultaneously execute the target application. However such demonstrations are complex, time consuming and so ignored in the literature. In this paper we address this issue by conducting a SystemC simulation of the MPEG4-SP decoder with several real-life video sequences. The obtained results demonstrate that the proposed algorithm significantly enhances the performance. In addition, the proposed algorithm outperforms the available run-time mapping technique. Future work will consider the implementation of integrated module in the NIs and estimating the overhead in terms of area and energy.

REFERENCES

[1] W. A. Najjar *et al.*, "Advances in the dataflow computational model," *Parallel Computing*, vol. 25, no. 13, pp. 1907 – 1929, 1999.
[2] K. J. M. Martin *et al.*, "Notifying memories: a case-study on data-flow applications with NoC interfaces implementation," in *Proc. of the Design Automation Conf. (DAC)*, June 2016.
[3] H. Yviquel *et al.*, "Orcc: Multimedia development made easy," in *Proc. of the ACM Int. Conf. on Multimedia*, ser. MM '13. New York, NY, USA: ACM, 2013, pp. 863–866.
[4] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.
[5] Y. Lesparre *et al.*, "Evaluation of synchronous dataflow graph mappings onto distributed memory architectures," in *Proc. of Euromicro Conf. on Digital System Design (DSD)*, Aug. 2016, pp. 146–153.
[6] E. A. Lee and T. Parks, "Dataflow process networks," in *Proc. of the IEEE*, 1995, pp. 773–799.

[7] C. Wang *et al.*, "Dynamic application allocation with resource balancing on NoC based many-core embedded systems," *J. Syst. Archit.*, vol. 79, no. C, pp. 59–72, Sep. 2017. [Online]. Available: https://doi.org/10.1016/j.sysarc.2017.07.004

[8] J. Henkel *et al.*, "Dynamic resource management for heterogeneous many-cores," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[9] S. Kaushik *et al.*, "Computation and communication aware run-time mapping for NoC-based MPSoC platforms," in *Proc. of IEEE Int. SOC Conf.*, Taipei, Taiwan, 2011, pp. 185–190.

[10] T. Maqsood *et al.*, "Dynamic task mapping for network-on-chip based systems," *J. of Systems Architecture*, vol. 61, no. 7, pp. 293 – 306, 2015.

[11] H. R. Mendis *et al.*, "Dynamic and static task allocation for hard real-time video stream decoding on nocs," *Leibniz Transactions on Embedded Systems*, vol. 4, no. 2, pp. 01:1–01:25, Jul. 2017.

[12] M. Rapp *et al.*, "Neural network-based performance prediction for task migration on S-NUCA many-cores," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[13] S. Paul *et al.*, "Adaptive task allocation and scheduling on NoC-based multicore platforms with multitasking processors," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 1, Dec. 2020.

[14] ——, "A hybrid adaptive strategy for task allocation and scheduling for multi-applications on NoC-based multicore systems with resource sharing," in *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2021, pp. 1663–1666.

[15] Z. Li *et al.*, "Chordmap: Automated mapping of streaming applications onto CGRA," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[16] D. Huff *et al.*, "Clockwork: Resource-efficient static scheduling for multi-rate image processing applications on FPGAs," in *Proc. of IEEE Annual Int. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 186–194.

[17] T. D. Ngo *et al.*, "Move based algorithm for runtime mapping of dataflow actors on heterogeneous MPSoCs," *J. of Signal Processing Systems*, vol. 87, no. 1, pp. 63–80, Apr. 2017.

[18] A. Singh *et al.*, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. of the Design Automation Conf. (DAC)*, May 2013, pp. 1–10.

[19] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for Network-on-Chip design," *J. of Systems Architecture*, vol. 59, no. 1, pp. 60–76, 2013.

[20] K. Huang *et al.*, "A scalable and adaptable ILP-Based approach for task mapping on MPSoC considering load balance and communication optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1744–1757, Sep. 2019.

[21] C. Rubattu *et al.*, "Pathtracing: Raising the level of understanding of processing latency in heterogeneous mpsocs," in *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. DroneSE and RAPIDO '21. NY, USA: ACM, 2021, pp. 46–50. [Online]. Available: https://doi.org/10.1145/3444950.3447282

[22] C. Chou and R. Marculescu, "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 1, pp. 78–91, 2010.

[23] E. L. d. S. Carvalho *et al.*, "Dynamic task mapping for MPSoCs," *IEEE Design Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

[24] M. Fattah *et al.*, "Adjustable contiguity of run-time task allocation in networked many-core systems," in *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2014, pp. 349–354.

[25] A. K. Singh *et al.*, "Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 72–85, 2016.

[26] C. Ykman-Couvreur *et al.*, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *IET Computers & Digital Techniques*, vol. 5, pp. 123–135(12), Mar. 2011.

[27] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for MPSoCs," in *Proc. of the Annual Design Automation Conf. (DAC)*, New York, NY, USA, 2013.

[28] A. K. Singh *et al.*, "Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, Jan. 2013.

[29] S. Skalistis and A. Simalatsar, "Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees," in *Proc. of the IEEE Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2017, pp. 752–757.

[30] H. Yviquel *et al.*, "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms," in *Proc. of the Int. Symposium on Image and Signal Processing and Analysis (ISPA)*, 2013, pp. 732–737.

[31] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 14, no. 1, pp. 14:1–14:25, Jan. 2015.

[32] J.-P. Diguet *et al.*, "Networked power-gated MRAMs for memory-based computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, Dec. 2018.

[33] S. S. Bhattacharyya *et al.*, "Overview of the MPEG reconfigurable video coding framework," *Journal of Signal Processing Systems*, vol. 63, no. 2, Dec. 2011.

[34] M. Wipliez and M. Raulet, "Classification of dataflow actors with satisfiability and abstract interpretation," *Int. J. of Embedded and Real-Time Communication Systems (IJERTCS)*, vol. 3, no. 1, pp. 49–69, 2012.

[35] T. Grotker *et al.*, *System Design with SystemC*. Springer, 2002.

[36] Xiph.org video test media. [Online]. Available: http://media.xiph.org/video/derf/

C. Selected publication on synchronisation of tasks

This appendix provides the author version of the paper "Subutai: Speeding Up Legacy Parallel Applications Through Data Synchronization", as a complement to the work presented in chapter 6.

Publication details

Title	*Subutai: Speeding Up Legacy Parallel Applications Through Data Synchronization*
Authors	Rodrigo CATALDO, Ramon FERNANDES, Kevin J. M. MARTIN, Jarbas SILVEIRA, Gustavo SANCHEZ, Johanna SEPÚLVEDA, César MARCON, Jean-Philippe DIGUET
In	IEEE Transactions on Parallel and Distributed Systems vol. 32, no. 5, pp. 1102-1116, 1 May 2021
DOI	10.1109/TPDS.2020.3040066
URL	`https://hal.archives-ouvertes.fr/hal-03082831`

Subutai: Speeding up Legacy Parallel Applications through Data Synchronization

Rodrigo Cataldo, Ramon Fernandes, Kevin J. M. Martin, Jarbas Silveira, *Member, IEEE*, Gustavo Sanchez, Johanna Sepúlveda, César Marcon, *Senior Member, IEEE*, Jean-Philippe Diguet, *Senior Member, IEEE*

***Abstract*—The decrease of the performance gain dictated by Moore's Law boosted the development of manycore architectures to replace single-core architectures. These new architectures must employ parallel applications and distribute its workload over a multitude of cores to reach the desired performance. Parallel applications are harder to develop than sequential ones since the developer must guarantee data integrity using synchronization primitives. While multiple novel solutions have been proposed to speed up parallel applications through handling one type of data synchronization primitive, exceptionally few works support multiple types of synchronization primitives and legacy code. This work proposes Subutai, a hardware/software co-design solution for accelerating multiple synchronization primitives without modifying the application source code. By providing a new user library, while retaining an existing synchronization API, legacy and novel applications can benefit from our solution. Our experimental evaluation, which provides a POSIX Threads implementation, demonstrates Subutai speeds up to 2.71× and 4.61× the execution of single- and multiple-application executions, respectively.**

***Index Terms*—Legacy Parallel Applications, PThreads, Network-on-Chip, Distributed Scheduler**

I. Introduction

Since the end of the last century, a significant shift has occurred in the industry, transitioning the processor chips from a single- to a multicore design using a dozen cores. This paradigm has evolved to incorporate hundreds and soon thousands of simple cores, performing a manycore architecture, to continue to deliver higher performance.

Unfortunately, only increasing the number of cores does not imply increasing the performance, as the applications must be parallel-compatible to exploit the hardware parallelism. Where once a single sequential thread could do the execution, now the developer has to partition the workload into multiple execution threads and synchronize their execution [1], dealing with deadlock, livelock, race condition, and non-deterministic events [2]. Decisions regarding both partitioning and synchronization of the workload are crucial to determine the achievable performance of the application on manycore systems since even small sequential portions of execution can have a significant performance impact, as observed in Amdahl's law. Because of this impact, parallelization is primarily done manually, allowing fine-grained performance optimizations.

Synchronization, namely the access and update of the application data, is a vital concern in any parallel application. The typical limitation to novel synchronization solutions is that developers have to refactor the source code. The redesign applies even to already parallel-compatible code, as the Application Programming Interface (API) of different solutions are not the same. The refactoring of source code due to API changes has substantial limitations; we highlight these three: (i) software redevelopment cost, (ii) challenge of parallel code refactoring, and (iii) lost legacy source code.

Software development cost already dominates new System-on-Chip (SoC) designs, as the manycore architecture and its counterpart, the parallel applications, are common elements of such designs [3]. Besides, the Read-Copy-Update (RCU) synchronization primitive used by the Linux kernel, for instance, influences over 16 million Lines of Code (LoC) across 15 kernel subsystems [4]; thus, even experienced developers do not easily achieve a refactoring of it.

Source code modification is always an error-prone task. McConnell estimates that up to 100 bugs can be present per thousand LoC [5]. Refactoring parallel code is even more susceptible than sequential code because often the developers are befuddled with the use of synchronization techniques. For instance, while RCU shows impressive results, it demands a thorough understanding of computer architecture design, presenting the tradeoff of rising performance gains but increasing code and maintainability complexity [2].

Finally, the essential requirement for refactoring a legacy application is the source code availability. However, often the legacy source code is lost, leaving only the binary code. Hence, the developers need to rewrite the entire code, increasing the software development cost. Moreover, given the amount of legacy software, a complete rewrite of the entire code is unlikely to happen [6].

Therefore, **we propose a novel synchronization solution that accelerates parallel applications without modifying the application source code**. Our solution speeds up even applications that do not or cannot share their codes; in this case, as

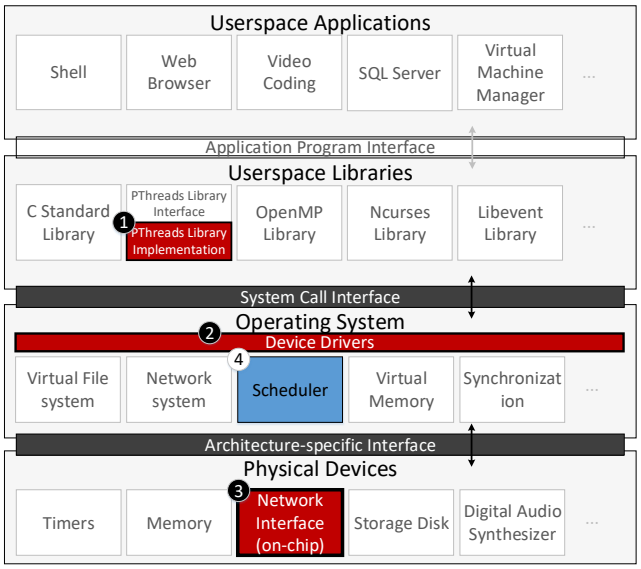


Fig. 1. Subutai components are highlighted in red (1, 2, 3) in the computing stack. Subutai only requires changes in the (1) PThreads implementation, (2) OS NI driver, and (3) on-chip NI. Additionally, (4) a new scheduling policy (in blue) is explored in this work as an optional optimization.

long as the binary is dynamically linked. Otherwise, static or dynamic linked binaries are supported. Our hardware/software solution, called Subutai, tackles the synchronization problem within a low-level Network-on-Chip (NoC) Interface (NI).

Software-wise (Subutai-SW), we implemented the POSIX Threads (PThreads) according to the IEEE Std 1003.1 standard [7][1]. Thus, any application employing the PThreads API (i.e., pthread.h) is compatible with Subutai. The PThreads compatibility restricts a multitude of optimizations since we cannot inject the source code with extra synchronization metadata or change the application communication model. In addition to interfacing with the application, our software must work with new functionalities on the hardware-side; hence, we provide an Operating System (OS) driver responsible for the latter activity.

Hardware-wise (Subutai-HW), we extended an existing on-chip NI to support, in a distributed way, the following synchronization primitives: mutex, barrier, and condition. NI handles new types of packets and requires access to a small (less or equal to 1KiB) memory to record synchronization events and metadata. Fig. 1 depicts the Subutai solution with a general-purpose computing stack, highlighting the components required for its operation.

We demonstrate that our solution speeds up single parallel applications ranging from $1.05\times$ up to $2.71\times$ for 64-thread executions. Moreover, in a competitive scheduling scenario, Subutai speeds up multiple parallel applications ranging from $1.58\times$ up to $4.61\times$. For these results, the hardware requirement for Subutai increases the area of the NI in, approximately, 46%; however, the overhead is insignificant compared to the total chip area (less than 1% for a 400mm^2 chip). The key contributions of this paper are listed next:

1) This work proposes a novel synchronization technique that avoids modifying parallel applications while accelerating their execution. The work supports both legacy and novel applications designed using the PThreads API.
2) We designed all the components of Subutai and provided a detailed analysis of its performance in accelerating standard synchronization primitives. Moreover, we evaluate it with state-of-the-art related work.
3) We conducted experiments using parallel applications provided by PARSEC, a well-known benchmark for this domain. The experiments were analyzed for both single- and multiple parallel executions. Besides, we evaluated scheduling policies for executing parallel applications. Such experiments are essential to evaluate the performance of Subutai on several execution scenarios.

This paper extends a conference version [8] by (i) evaluating Subutai with state-of-the-art related work, (ii) providing new estimations of the Subutai-HW design including memory, (iii) presenting details of the Subutai-SW implementation (userspace library and OS driver), (iv) evaluating an additional application (x264), (v) evaluating a scheduler policy proposal, (vi) evaluating concurrent application execution, and (vii) presenting the synchronization model of the analyzed applications.

II. Related Work

A program can be comprised of many computational units like threads, processes, coroutines, and interrupt handlers. We employ the term thread as a generic word to encompass these computational units. We organize the related work in software-oriented and hardware-oriented/mixed solutions. Table I summarizes the essential characteristics of these solutions and compares our work to the state-of-the-art.

A. Software-oriented Solutions

PThreads, Open MultiProcessing (OpenMP), and Intel Threading Blocks Building (TBB) are established solutions that use software to synchronize parallel applications. These solutions provide analogous implementations of a similar set of synchronization primitives, but with different abstraction levels. In contrast, PThreads provides a low-level interface for developers, OpenMP and TBB offer abstract programming models (fork-join and task-based models, respectively) [20].

DeLozier et al. [1] propose SOFRITAS, a software-only robust memory consistency model that can detect and prevent atomic violations on parallel applications at the cost of execution overhead (roughly 59%). Unfortunately, the applications must be annotated with a novel API when using library calls.

Boehm [10] and France-Pillois et al. [11] provide optimizations on the implementations of the PThreads and OpenMP libraries, respectively. The first work suggests relaxing the reordering rules for load and store operations, while the last work identifies an expansive function that was uselessly being called during the barrier waking process.

[1]Includes mutex, barrier, and conditions. Besides, we provide the PThreads software implementation for supporting the options provided by the `attribute` parameter.

TABLE I
RELATED WORK SUMMARY.

Solution	Orientation	Requirements	Legacy code compatible*	Uses PThreads	Target data synchronization	Experimental results
PThreads	Software	Latency	No	Yes	Barr., cond., mutex	Real applications
OpenMP	Software	Latency, app. model	No	Yes (libgomp)	Atomic, barr., mutex	Real applications
TBB	Software	Latency, app. model	No	Yes (Linux)	Atomic, cond., mutex	Real applications
RCU [9]	Software	Latency	No	May use	Mutex	Linux kernel
Boehm [10]	Software	Latency	Maybe	Yes	Mutex	Synthetic
F.-P. et al. [11]	Software	Latency	Yes	Indirectly[b]	Barrier	IS and synthetic
SOFRITAS [1]	Software	Code correctness	Limited	Yes	Barr., cond., mutex	PARSEC, ...
Sivaram et al. [12]	Mixed	Fault-tolerance	No[a]	No	Barrier	Synthetic
Abellán et al. [13]	Mixed	Latency and area	No[a]	Indirectly[b]	Barrier	Synthetic
Stoif et al. [14]	Mixed	Latency	No[a]	No	Barrier, mutex	FPGA, synthetic
MCAS [15]	Mixed	Latency and area	No	No	Atomic	Synthetic
CASPAR [16]	Hardware	Latency	Yes	No	Atomic	FFT, IS, ...
HTM [17] [18]	Mixed	Latency	Maybe	May use	Mutex, spin lock	Indirectly[c]
Not. Mem. [19]	Hardware	Latency, app. model	Yes	May use	Spin lock	MPEG-4 decoder
Subutai	**Mixed**	**Latency and area**	**Yes**	**Yes**	**Barr., cond., mutex**	**PARSEC**

Barr. = Barrier; cond. = Condition; app. = Application; Not. = Notifying; Mem. = Memories; F.-P. = France-Pillois;
* This term is defined in Section II-C;
[a] Not addressed in the work;
[b] The work employs OpenMP, and it employs PThreads internally;
[c] HTM can be used on the PThreads implementation.

Attiya et al. [21] formally proved that deterministic structures, as employed by the previously discussed libraries, cannot eliminate the use of expensive synchronization. Therefore, non-deterministic solutions focusing on relaxing the constraints that force the use of such expansive synchronization have been proposed to tackle this problem. Kirsch et al. [22] propose k-FIFO, which is a lock-free queue that removes up to $k - 1$ out-of-order elements from the queue. Desnoyers et al. [9] describe a synchronization technique based on the publish-subscribe mechanism called RCU. Parallel applications that rely on RCU have to deal with stale data. The bottleneck of these solutions is that the application code adaptation is passed on to the developer.

B. Hardware-oriented/Mixed Solutions

Sivaram et al. [12] propose a fault-tolerant hardware-based barrier synchronization. Their design uses a tree structure to sum intermediate values, decreasing the number of packets injected into the network. Their work is complementary to our solution. Abellán et al. [13] explore three HW barrier architectures and integrate them on the OpenMP programming model. Unfortunately, they evaluated only synthetic applications. Stoif et al. [14] implement an arbiter on FPGA that guarantees mutual exclusion to a portion of the shared memory area and an HW-based synchronization barrier that speeds up the application execution; however, their work does not implement full barriers and conditions, and it is limited to simple test cases instead of real applications.

CASPAR [16] improves the performance of CAS operations by breaking the serialization of multiple CAS calls and executing them in parallel. Patel et al. [15] propose a special HW instruction, called MCAS, to change multiple memory positions atomically, optimizing the synchronization process. Hardware Transactional Memory (HTM)[2] provides an abstraction for executing blocks of code atomically. HTM guarantees correctness by aborting transactions that conflict with others [17].

Finally, Martin et al. [19] propose the Notifying Memories concept to reduce communication latencies introduced in the NoC by pruning useless memory accesses. This concept uses spinlocks and is applied to dataflow applications only. Our work is also based on extending the NI architecture, but targeting shared-memory systems.

C. Comparison with the state-of-the-art work

A direct comparison of works in the data synchronization field is unfeasible as they do not employ a common test scenario that standardizes the experimental evaluation. Especially hardware and mixed solutions employ a varied set of target applications. Table I shows that there is no intersection of applications in the experimental results. Consequently, we limit the comparison of experimental results to published results on Section VIII; here, we discuss the support for data synchronization primitives and legacy code.

Three solutions are generic API specifications (PThreads, OpenMP, TBB) for cross-platform use. All other works are optimizations on existing APIs, except for RCU, as it creates a read-write lock capable of reading and writing at the same time. Boehm optimizes memory barriers for lock and unlock PThreads procedures. The approaches proposed by France-Pillois et al. and Abellán et al. share the same idea of optimizing the use of barriers in OpenMP applications. The former achieves this through a software-only approach, while the latter uses a mixed solution. MCAS and CASPAR optimize the use of CAS procedures on lock-free applications. The Notifying Memories solution targets a specific programming

[2]HTM can be simulated in software, yet the overhead imposed by the software layer can be prohibitive [23].

model and synchronization scenario: data-flow and spinlocks, respectively. HTM allows speculative execution of critical sections guarded by mutexes or spinlocks. Finally, our solution accelerates PThreads data synchronization primitives through hardware execution while keeping legacy-code compatibility.

We define that a solution is legacy code compatible if a given application can use the set (or a subset) of the solution, either by (i) recompiling without source code changes, or (ii) dynamically linking to a library provided by the solution. Therefore, besides Subutai, the following solutions support legacy code: Boehm [10], France-Pillois et al. [11], CASPAR [16], Notifying Memories [19], and HTM [17].

The works of Boehm and France-Pillois et al. are entirely done at the software level; they are not directly related to our work, as the former does not support reordering I/O operations (which we use for Subutai-HW communication), and the latter is an optimization for OpenMP (which we only support indirectly). CASPAR accelerates a different type of application (lock-free applications) not supported directly by PThreads or Subutai. Notifying Memories can benefit from our work if the spinlocks usage is done through PThreads (i.e., `pthread_spin_lock`), which is not the case of the paper presented in [19]. Besides, Notifying Memories target the data-flow application model only, while we support any model that uses the shared-memory paradigm.

HTM has two operation modes, whereas the Hardware Lock Elision (HLE) is the only mode with legacy support. HLE extends the parallel library code (e.g., PThreads) to use a hot/slow path approach. Firstly, the operation is executed speculatively using HTM; if it fails, then the legacy code is executed. HTM uses the same approach of Subutai, making changes in the library synchronization routines only. Besides, HTM is complementary to our solution, as both can be used in unison to handle synchronization primitives.

To the best of the authors' knowledge, Subutai is the only solution that speeds up various types of synchronization primitives while keeping unchanged the userspace interface (i.e., API).

III. Software-only and Subutai Solutions

Solutions for data synchronization are implemented in software-only (SW-only) or in a hardware/software composition. The solutions provide trade-offs according to the constraints on the target design (e.g., portability, performance). This section aims to clarify the target architecture used for achieving the experimental results, as well as to clarify the control flows used to synchronize shared data, using an example based on the Linux OS.

A. Target Architecture

Fig. 2 shows a schematic representation of the target architecture. Each core communicates with caches and a local NI. An NoC with routers using a standard design that includes buffers, a crossbar switch, and a switch allocator implements the interprocessor communication.

Modern multiprocessors consist of double digits of processing core units [24]. Thus, we target an NoC-based manycore architecture composed of 64 processing cores. Each core has access to instruction and data caches. The Level 1 cache is private and is divided into instruction and data caches. The Level 2 cache is shared among the cores, and banks are distributed on the system. Therefore, our target architecture uses a Non-Uniform Cache Memory Access (NUCA) architecture with faster L2 accesses for nearby banks. We explore synchronization solutions for Symmetric Multiprocessing (SMP), because it facilitates the development of parallel applications, as developers do not need to concern themselves with data placement [25]. Hence, cache coherence is required and used.

The SW-only uses a single instance of Linux, while Subutai employs a decentralized approach, where each core has its self-governing OS. The decentralized OS design enables the scheduler to be decentralized as well. A decentralized scheduler provides a faster thread switching, which benefits parallel applications. Additionally, for dozens or more cores, message passing can be much faster than memory sharing [26].

B. Target Parallel Library

Subutai transforms software events (e.g., mutex lock, condition wait) in hardware events (e.g., NoC packets). As such, we can target any number of available library interfaces. We chose the PThreads interface because (i) it is widely employed as a *de facto* standard to parallel application implementation, and (ii) it is used internally as the base of multiple synchronization solutions, as shown in Table I. Consequently, PThreads provides Subutai a broad range of applicability.

We focused on three of the four main groups of the PThreads standard operations: mutex, barrier, and condition handling. Thread events (create, exit, join) are not on the critical path, and so are left to be handled at the OS level. An extensive description of PThreads operations is out of our scope. We limit the discussion to the essentials of the three focused groups.

The mutex group contains *locking* and *unlocking* functions. Locking is a blocking function that exclusively locks a vari-

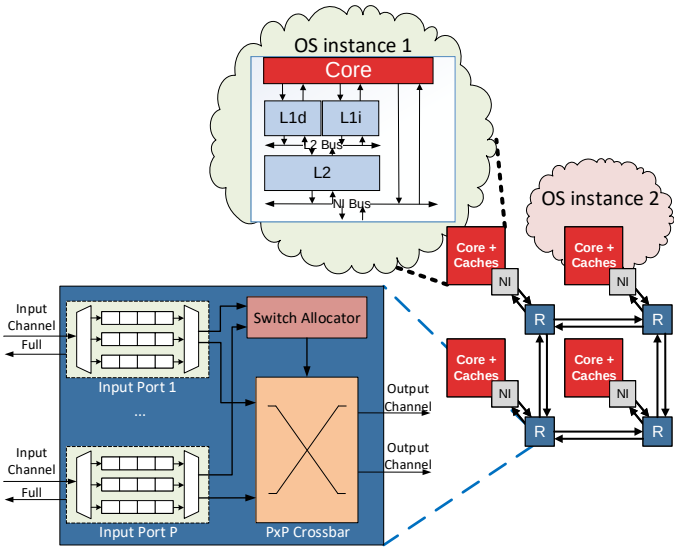


Fig. 2. Schematic of the manycore target architecture.

able. If the variable is already locked, the calling thread is blocked. Otherwise, this operation returns the variable locked by the calling thread. Unlocking is a non-blocking function that changes the variable state and wakes up blocked functions if there are any waiting threads.

The barrier group contains a single blocking function, called *wait*, which synchronizes participating threads at a user-specified code point. A barrier has a fixed number of threads decided at allocation time; participating threads are only woken up when they all hit the barrier.

The condition group contains *wait*, *signal* and *broadcast* functions. Wait is an unconditionally blocking function that inserts threads on a waiting list for a condition event. The operation of the wait function requires locking a mutex variable, which is passed as a reference to the function; this mutex is unlocked once the wait function concludes its work. The signal and broadcast are non-blocking functions that wake up one and all threads, respectively, waiting for a condition event. In these cases, the mutex is optional.

For all groups, one or more queues are required to record blocked threads. Condition functions need to handle two queues due to the associated mutex. Barrier and mutex functions deal with only one queue. Besides, the three groups have non-blocking functions that allocate and deallocate variables. This work replaces the handling of these operations from an entire software solution to a hardware/software approach.

C. Software-only Solution

Fig. 3 exemplifies the synchronization flow for the SW-only solution deployed on Linux. The example starts with the user application requesting a synchronization operation through a function call, such as a mutex lock. The function is associated with a PThreads interface that acquires the requested lock for the thread using a memory shared among application threads.

The SW-only implementation tries to acquire the mutex atomically multiple times. The first moment occurs within the PThread library, which, on success, immediately returns to the application (delay marked with t_1). Otherwise, the PThread library calls the Linux Kernel (specifically the Futex subsystem), which has another codepoint for obtaining the mutex; the last codepoint is the most time-critical point, as it implies that the current thread goes to the sleeping state, waiting for a mutex unlock event originated by the owner thread to be awakened for requesting the mutex again. t_2 and t_3 reference the delays associated with these two accesses to the shared memory, respectively. Additionally, if the thread cannot acquire the mutex lock after being awakened (due to another thread executing on t_1), then the SW-only solution implements a loop to re-execute its code. At each loop, the basic delays referenced by t_2 and t_3 may be increased by Δt_a, or $\Delta t_a + \Delta t_b$, depending on where the mutex lock is obtained [27].

Throughout the timing of the synchronization flow, extra delays may occur due to access to the shared cache memory, as well as possible cache misses in the shared data that would imply a much more significant delay. Also, since Futex is potentially distributed into the manycore caches, these two last codepoints imply communication costs through the NoC.

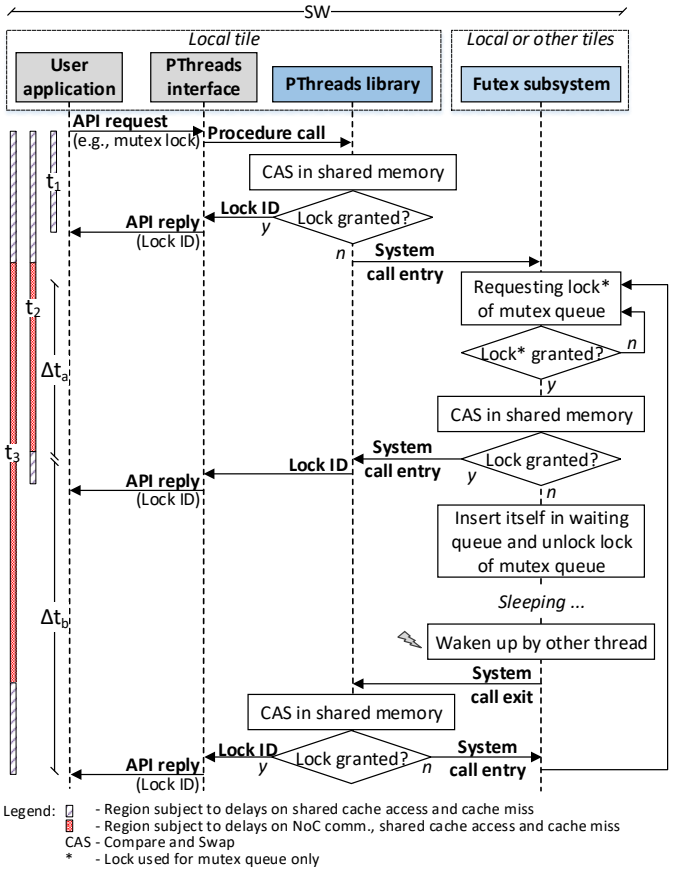


Fig. 3. Synchronization control flow employed on Linux-only based solution.

D. Subutai Solution

Subutai is a synchronization solution for legacy and novel parallel applications comprised of a software/hardware co-design to perform fast synchronization operations. This section describes the high-level interaction of the Subutai's components, which are illustrated in Fig. 1, together with a general-purpose computing stack.

Subutai encompasses a userspace library, a kernelspace driver, a hardware module, and an optional scheduler policy (discussed in Section VII). The userspace library mimics an existing synchronization solution intended for parallel applications. Therefore, the Subutai library procedures provide the same interfaces (i.e., API) with different implementations. The ability to mimic existing synchronization libraries is an essential feature of Subutai to speed up parallel applications.

Each core in the system has a Subutai-HW module that extends the NI and is responsible for accelerating synchronization operations. Subutai-HW is a Finite State Machine (FSM) coupled with a small dedicated memory (details in Section V). Once the user application calls a procedure, the Subutai library employs kernel services through system calls, providing the link between the hardware and software parts. Thus, the userspace library abstracts the hardware protocol (Subutai-SW - details in Section IV).

Fig. 4 depicts the Subutai communicating flow. A unique identifier (ID) on the entire system addresses each synchronization variable. An incremental counter determines the NI

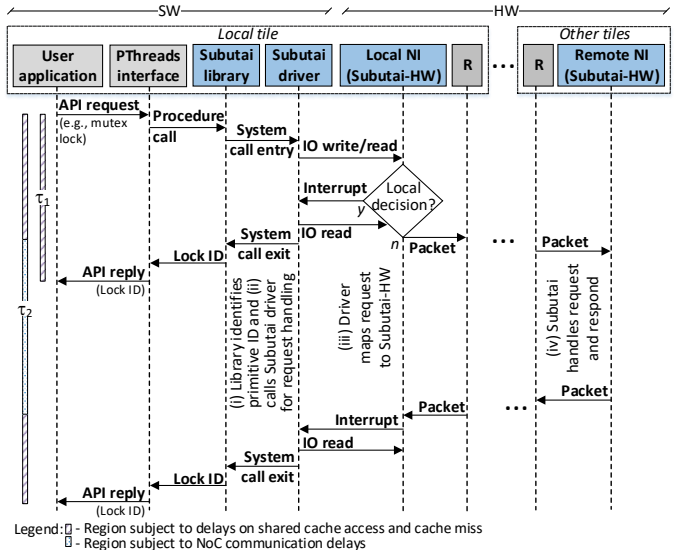


Fig. 4. Synchronization control flow employed on Subutai.

that hosts the synchronization primitive: NI_0 hosts the first primitive; NI_1 hosts the second one, and so on, following a fairness method. Other dynamic allocation strategies can be further studied, but this is out of the scope of this work.

The communication flow of Subutai starts with the application making a PThreads interface request through any function described in this section; the Subutai library identifies the unique ID for this primitive and passes it to the driver along with the interface request. Then, the driver writes to either registers or a memory that the NI has access to; this decision is made at the driver level with the capabilities available in the system. Next, the driver writes in a control register to inform the command to the NI and waits for an interrupt to receive the remote response.

In case the local Subutai-HW hosts the lock, the NI can respond immediately, performing a prompt request from the driver. Thus, the driver does not use the router, avoiding the injection of packets in the NoC; the delay of this procedure is marked with τ_1. Therefore, situations where the local Subutai-HW hosts the synchronization primitive implies a quick response as the request does not propagate across the NoC.

If the local Subutai does not host the lock, then the local NI injects a packet into the NoC targeting the remote Subutai-HW, which handles the request and responds to the local NI with a new packet. The address of the remote Subutai-HW is embedded into the ID packet field (discussed in Section V-A). This procedure implies an additional delay of packet traffic on the network, being noted by τ_2.

E. Subutai vs SW-only Solutions

The comparison of Fig. 4 with Fig. 3 allows us to understand the differences between Subutai and SW-only approaches. Synchronous flows marked by t_1 and τ_1 exemplify situations where the local processing manages the lock. Thus, regardless of the approach used, the response latency is lower compared to the latencies of the decentralized processing flows.

Subutai offers a more efficient hardware-level solution for a decentralized decision; thus, the flows marked with t_2 and t_3 have higher latencies than the one marked with τ_2.

The reasons for the lower latency of Subutai are: (i) locking for the mutex queue (marked with * in Fig. 3) is required only in the SW-only approach, as Subutai-HW has access to private memory area to handle the concurrent threads (Section V-A); (ii) susceptibility to data conflicts in distributed shared caches, which does not occur in Subutai that implements this functionality using dedicated queues in Subutai-HW (Section V-A); (iii) the efficiency of a lock event when another thread is using it. The Subutai implementation returns this information to the local tile as soon as it is available, while the SW-only implementation is delayed by the OS scaling (Fig. 4 – HW events can occur concurrently to the execution of the thread); (iv) the use of dedicated control packets allows to employ Quality-of-Service (QoS) techniques, providing differentiated priority for the traffic of control packets (Section V-A). The consequence is that control packets are propagated with lower average latency and that the variability between latencies is also lower compared to data packet latencies.

As a conclusion, either because of cache conflicts or packet latency variability in the NoC, Subutai ensures more predictability than the SW-only solution.

IV. SUBUTAI-SOFTWARE (SUBUTAI-SW)

Subutai provides a new PThreads library for parallel applications to use our solution. Every time the user application requests an operation on a mutex, barrier, or condition, the library passes on the request through a system call for the OS driver (items (i) and (ii) from Fig. 4). The request is changed in terms of structure, as the user application handles these synchronization variables by variable names, which are memory positions, while the hardware tracks these variables with a unique ID unrelated to the memory and name of the variable. The driver receives the unique ID for the variable, which is known by the library (not known by the application) and decodes the packet destination through reserved fields in this ID (Section V-A) (item (iii) from Fig. 4). The request is processed in hardware, and, eventually, the response is received in the local NI. Then, the local NI interrupts the software to notify it of a packet arrival event; the OS driver reads it and is able to finish the user application request (last four steps of Fig. 4). Finally, other types of PThreads operations (i.e., thread management) are kept unchanged.

Because most operations of PThreads are offloaded to be handled on the hardware, valuable cache space can be saved (up to 91.7% compared to x86_64) for the respective structures of the synchronization variables, as shown in Table II. All synchronization primitives have the same size in Subutai since they only contain the 4-byte unique ID (refer to Section V-A). The on-chip NI driver implementation was based on an existing driver that performs basic procedures for sending and receiving packets. We reuse these procedures for the requests from the PThreads library. Additional logic is introduced in the driver to understand the packets sent and received, as Subutai makes the driver an active component to change thread states on its own (e.g., wake up a thread when it owns a mutex).

TABLE II
MEMORY SPACE REDUCTION OF SYNCHRONIZATION PRIMITIVES.

Primitive (name)	GNU LibC x86_64 (bytes)	Subutai (bytes)	Reduction (Percentage)
mutex	40	4	90.0%
barrier	32	4	87.5%
condition	48	4	91.7%

V. SUBUTAI-HARDWARE (SUBUTAI-HW)

A. Architecture and Implementation Choices

Subutai-HW extends a standard NI architecture for handling synchronization operations fastly. Fig. 5 shows the schematic representation of Subutai-HW and its location on the target architecture. The main components of Subutai-HW are (i) an FSM, (ii) a set of registers; and (iii) a local ScratchPad Memory (SPM), which is entirely controlled in HW by the FSM, except for memory initialization. Initialization is done through the OS driver and requires the creation of a free double-linked queue. We validated and implemented the Subutai-HW architecture by Register-Transfer Level (RTL) simulation [28] and synthesis [29]. Besides, we developed an analytical model to demonstrate its operation latencies and scalability.

The left-hand side of Fig. 5 shows that Subutai-HW employs double-linked queues to record events. As an alternative to statically allocating for the worst case, the double-linked queues allow Subutai-HW to employ a dynamic allocator for reducing memory consumption to the minimum, at the cost of additional pointer arithmetic logic. Besides, condition variables are dealt more efficiently with such structure, as it avoids the thundering herd problem [30]. We based the queue manipulation on the futex implementation of the Linux kernel [31].

Subutai-HW operates using two structures for recording information. Fig. 6 shows the first one, which records the metadata of the synchronization primitives. Software only knows the first 32-bit field, which is employed as an ID of this primitive. However, for Subutai-HW, the first bit "F" is used to allocate/deallocate this structure. The next 7-bit field is the unique ID for the NI on the system. Lastly, the furthest 24-bit field is used as a pointer to itself; we employ this technique to avoid the cost of searching for an element of the structure every time a new request has arrived. The second 32-bit field encompasses the head and tail of the double-linked queue. The last 32-bit field records values used for some of the primitives. The first 16-bit field is employed to (i) record the thread and core that owns a mutex, and (ii) store the current number of threads waiting on a barrier. The barrier primitive uses the furthest 16-bit field to record the maximum number of threads allowed in a barrier.

Fig. 7 shows the second structure – a double-linked queue element composed of six fields. The first bit is employed to allocate/deallocate the element. The "prev" and "next" fields are pointers to the previous and next elements, respectively, or nil if they do not exist. The 16th bit "R" is reserved and used for memory alignment. The last 32-bit field identifies the requesting thread. The "Core ID" field is padded with zeroes because the NoC packet uses only 8-bit to identify the core.

TABLE III
LATENCIES FOR SUBUTAI-HW FSM STATES. c = CYCLE LATENCY, m = MEMORY LATENCY, n = NUMBER OF SYNCHRONIZATION VARIABLES HANDLED BY SUBUTAI-HW, ρ = NUMBER OF THREADS ON A BARRIER.

State	Best response time	Worst response time	Packet Injection
Allocation	$4m + c$	$(n \times m) + (3m + c)$	$(n \times m) + (m + c)$
Deallocation	$3m$	$3m$	None
Mutex Lock	$2m + c$	$11m$	$2m + c$
Mutex Unlock	$2m$	$10m + c$	$2m + c$
Barrier Wait	$7m$	$(m + c) + \rho \times (11m + 3c)$	$(m + c) + (12m + 4c) + (23m + 7c)... = (m + c) + \rho \times (11m + 3c)$
Condition Wait	$5m + c +$ Mutex Unlock	$10m + c +$ Mutex Unlock	None
Condition Broadcast	m	$18m + c$	$11m + c$
Condition Signal	m	$29m + 2c$	$11m + c$

The minimum memory requirement for the SPM is one control element and 63 queue positions, regarding a target 64 core architecture. Since we have to record up to $p - 1$ cores, the minimum SPM size is $\frac{1\times96+63\times64}{8} = 516$ bytes. Note that Subutai-HW is incorporated into every NI; consequently, we handle up to 64 primitive variables even with minimum sizing. The target architecture employs an SPM of 1 KiB (4 control elements and 122 queue elements) that handles up to 256 primitive variables in hardware. A double-linked queue allocates elements dynamically, allowing Subutai to consume memory on demand. A static allocator, on the other hand, cannot handle more than one control element with only 122 positions available ($< 2\times63$) – since the worst-case scenario is 63 positions per element[3], as explained earlier. Thus, a static solution would be either too limited or a waste of memory resources.

Although the number of primitives used in the experimental results is far from the SPM memory limit, there are two scenarios where the SPM cannot handle a request. In one scenario, the system does not have more primitive space available in any SPM; thus, Subutai rolls back to provide the SW-only implementation of the primitive. In the other scenario, there are no more queue elements available in a given primitive; therefore, we respect the POSIX standard and set `errno` to `EAGAIN` [7], hinting to the developer that it should try again later.

B. Response Time

Table III shows the latencies of the states as dependent on the Subutai-HW cycle c, the SPM write/read latency m, the number of synchronization primitives handled n, and the maximum number of threads on a barrier ρ. Each memory operation can either be a write or read operation in a given m cycle. The first column identifies the Subutai-HW state. The second and third columns identify the fastest and slowest

[3]We assume for the sake of size estimation that the number of threads does not exceed the number of cores. However, the queue is capable of handling such a scenario.

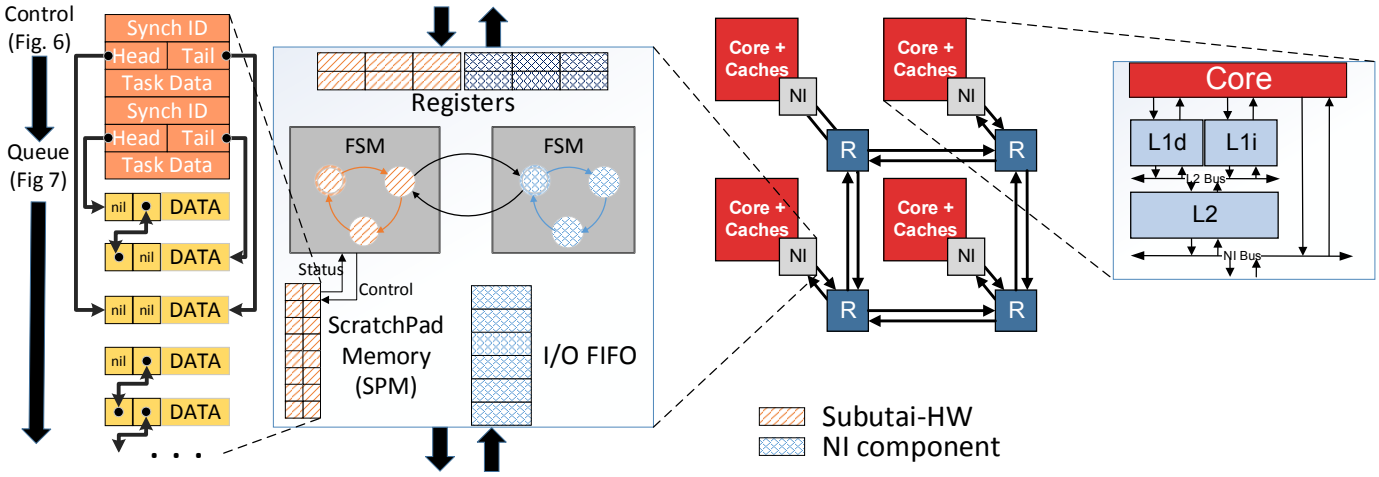


Fig. 5. Subutai-HW organization - specific Subutai-HW elements are highlighted in orange, whereas standard NI components are in blue.

latencies for the state, respectively. Finally, the last column shows when the packet is ready to be injected into the NoC – as, for some states, packets can be injected before finalizing the request processing. Additionally, some states (e.g., Deallocation) do not need to generate packets at all.

To illustrate the best and worst response time of Table III, we describe the Mutex Lock state, which models the `pthread_mutex_lock` operation. The fastest scenario, whose latency is $2m + c$, happens when the mutex is unlocked. It requires two memory operations: (i) fetch the control structure (field "Value" from Fig. 6) to check the owner of the mutex (latency = m); and (ii) rewrite this field with the requesting thread (latency = m). Finally, the NI is notified that a new packet can be injected (latency = c). The injected packet is the same as the requesting packet except for the header. The worst scenario takes more time (latency = $11m$) because the state deals with two queues entries. It starts with the same memory operation that reads the control structure for this primitive. Thus, the circuit realizes there is already an owner, which demands to queue up the request. First, Subutai-HW allocates a free queue entry and updates its queue pointers (takes up to 4 memory operations); then, it writes the requesting thread information into it and the tail information in the primitive metadata (6 more memory operations), performing 11 memory operations in total. The

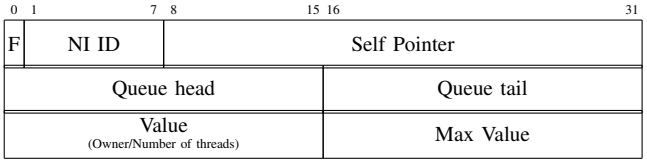


Fig. 6. Subutai-HW control structure.

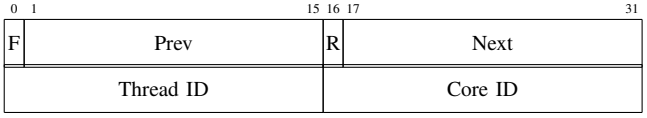


Fig. 7. Subutai-HW queue structure.

TABLE IV
SUBUTAI-HW STATES LATENCY WITH $c = 1ns$, $m = 2ns$, $n = 4$, $\rho = 63$, $FSMentry = 4ns$, $FSMexit = 1ns$.

State	Best response time (empty queue)	Worst response time (queued)	Packet Injection	
			Best	Worst
Allocation	14 ns	20 ns	10 ns	15 ns
Deallocation	11 ns	11 ns	None	
Mutex Lock	10 ns	27 ns	None	10 ns
Mutex Unlock	9 ns	26 ns	None	12 ns
Barrier Wait	19 ns	1583 ns	None	7, 32, 57, ... ns
Condition Wait	20 ns	47 ns	None	
Condition Broadcast	7 ns	42 ns	None	27 ns
Condition Signal	7 ns	65 ns	None	27 ns

latency for the other states follows a similar procedure.

Table IV shows the latencies used in the experimental results. We clocked Subutai-HW at the same frequency as the NI (1 GHz). SPM employs the previously discussed 1 KiB single-port SRAM-based implementation with uniform access of 2 cycles, 4 control structures, and 122 queue entries. Besides the Subutai-HW state latencies of Table III, Table IV includes the values of the NI used in this work; let $FSMentry$ and $FSMexit$ be the entry and exit latencies for Subutai-HW, then $FSMentry = 4$ ns (3 cycles for 3 flits of 32 bits and 1 cycle to decide the next state) and $FSMexit = 1$ ns (1 cycle to set a flag) to reach any state. A detailed report of equations and values described in Tables III and IV, and the pseudo-code implementation of Subutai-HW can be found in [32].

The latency required to release threads on a barrier exceeds one thousand nanoseconds due to the queue size of threads waiting on the barrier – it does not represent the packet injection latency. Thus, some threads execute much earlier than the total value. As shown in the last column, the packets are injected periodically at every 25 ns, except for the first packet, which is injected in 7 ns. Thus, the total number of cycles is 1583 ns, which is composed of the following parameters: $FSMentry + FSMexit + m + c + \rho \times (11m + 3c)$.

The *Condition Broadcast* and *Condition Signal* states show interesting latency results. At first glance, it would seem more plausible that releasing one thread (signal) would be faster than

releasing all threads (broadcast). However, this conjecture is not valid due to the following reasons. First, by releasing all threads, the state has to deal with only one queue (mutex) instead of two queues (mutex and condition). Second, due to the way condition works, only a single thread is indeed released since a mutex is associated with it. Consequently, the broadcast state avoids the scenario previously described for the barrier state – only the owner of the mutex will be released.

Subutai-HW also includes six 32-bit and three 1-bit registers; three are used for the packet fields (Fig. 8), and six more to (i) handle the free queue; (ii) memory swapping operations; and (iii) control flags to receive and send packets. For receiving/sending packets, Subutai-HW reuses the already available registers of the NI. The packet structure is combined with the recorded information in the two control structures (Figs. 6 and 7) to handle any request.

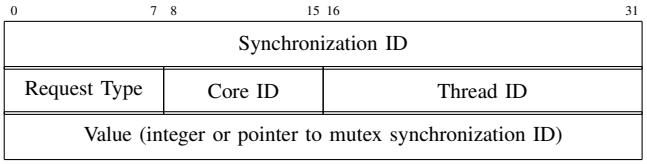


Fig. 8. Subutai's packet format.

VI. Application Synchronization Model

The performance of Subutai is evaluated through the widely used PARSEC benchmark, as it provides a wide range of application domains, parallelization models and data sharing. From their application set, we employ Bodytrack, Streamcluster, and x264; we limit our discussion to the synchronization model used by these applications. An extensive overview of these applications is outside the scope of this paper (more information can be obtained in [33]).

Bodytrack is a computer-vision application that tracks a 3D pose of a mark-less body. It uses mutexes for data sharing, and conditions and barriers to make sure all threads are synchronized and able to handle more requests. The workflow of Bodytrack starts with a single 'master' thread (T_0) responsible for creating synchronization primitives, creating T_{n-1} threads, and sending computation requests for them. Then, the threads $T_1, \ldots, T_{n-2}$ do the actual computation through the requests from T_0. Finally, the last thread (T_{n-1}) performs asynchronous I/O operations (e.g., loading images from disk to memory).

Fig. 9 depicts the workflow of Bodytrack. Initialization is done exclusively by T_0, where the synchronization variables and threads are created. Then, T_0 divides the computational work among the worker threads and sends a condition broadcast for all worker threads. Meanwhile, each worker thread checks if its work is available - if so, the thread skips the condition and moves on to the next phase; otherwise, the thread waits for the condition variable.

Each thread uses mutexes to access shared memory while performing the computational work. Meanwhile, T_0 uses a barrier to wait for all worker threads to conclude their jobs. The barrier guarantees that all worker threads are ready to handle the next work request. As the worker threads finish

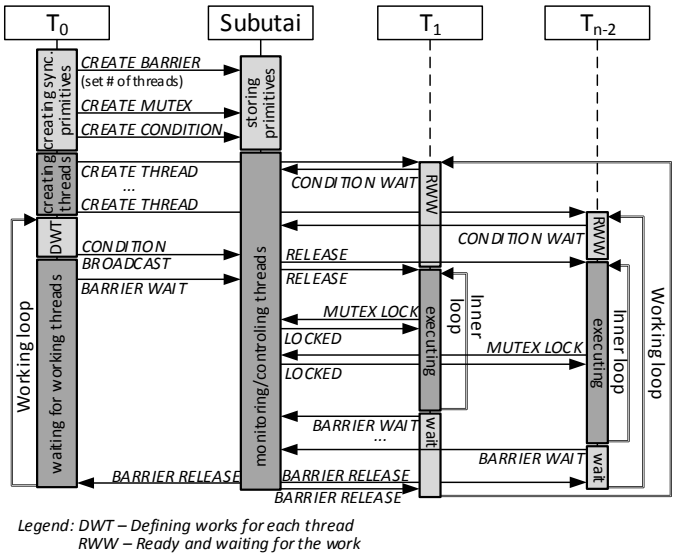


Fig. 9. Bodytrack's synchronization scheme. T_{n-1} is not shown as it does not participate in the Bodytrack workflow.

their work, they join the barrier as well. Only when all threads have joined the barrier, they are released to execute the next phase, which loops back to the generation of more work to the worker threads. This loop is executed until no more work is available.

The workflow plotted in Fig. 9 simplifies three aspects of the work of the Bodytrack application. Firstly, after the worker threads have received a request through the condition, they acknowledge it using another barrier (not shown in Fig. 9), and the associated mutex of the condition. Secondly, Fig. 9 does not show the thread responsible for asynchronous I/O (T_{n-1}) because it communicates only with T_0 and the number of requests is at most 10 events, which is tiny compared to the core workflow. Thirdly, Fig. 9 abstracts the application process conclusion because this process does not use data synchronization.

Streamcluster is a data-mining application that solves the online clustering problem for a stream of input points; it computes an approximation for the optimal clustering of them. This application has a simpler communication model than Bodytrack, using a single instance of mutex, barrier, and condition. Nonetheless, Streamcluster shares the same barrier-based synchronization scheme as Bodytrack.

x264 is a lossy video encoder for high-quality streams that do not employ barrier synchronization primitives, and all mutexes variables are associated with condition variables. This application uses a sliding pipeline model, whose number of pipeline stages equals the number of video frames, while the sliding window is determined at runtime by the number of thread requested. The total number of stages created is $1 + 2 \times videoframes$ [34].

Table V displays the number of synchronization primitives used by each PARSEC application. Additionally, Table VI depicts the number of synchronization events for the same set of applications. On the one hand, neither the number of threads nor the input size affects the number of synchronization

TABLE V
NUMBER OF SYNCHRONIZATION PRIMITIVES FOR PARSEC (SIMMEDIUM INPUT).

Application	Mutex	Condition	Barrier
Bodytrack	3	1	4
Streamcluster	1	1	1
x264	95	95	0

TABLE VI
NUMBER OF EVENTS OF SYNCHRONIZATION PRIMITIVES DURING THE EXECUTION OF PARSEC APPLICATIONS (SIMMEDIUM INPUT).

Application	Type	Events per number of threads		
		16	32	64
Bodytrack	Barrier[1]	2101	4293	13416
	Condition	447	750	1529
	Mutex	9000	10472	8677
Streamcluster	Barrier[1]	208048	364480	728960
	Condition	381	802	1274
	Mutex	510	1054	2142
x264	Barrier	0	0	0
	Condition	86	310	354
	Mutex	4154	4340	4344

(1) Every packet is counted as an event. Thus, in a 64-thread barrier, for instance, 64 events are generated for waiting on a barrier, and other 64 events are generated for releasing them, as the NoC does not support broadcast messages.

variables, except for x264, where the number of frames (i.e., input size) affects the number of primitives. On the other hand, Table VI displays that both affect the number of calls of these primitives.

VII. SCHEDULING

We employ multiple parallel applications that share computing resources to minimize the global idle time and maximize the rate of application instances. This work proposes a new scheduling policy to accelerate the critical sections of parallel codes. Additionally, we evaluate its performance impact on parallel applications against the Round-Robin (RR) scheduler, which does not differentiate the sections of parallel applications. We do not propose a new scheduler, instead, a scheduler policy that any scheduler can adopt in its decisions.

We target the scheduler rather than the application, as it does not require modifying the application code. Thus, we intend to further speed up applications by aggregating multiple parallel applications with a critical section-aware policy. Running multiple applications makes every application slower (i.e., increases the execution time), as they have to contend for computational resources. However, the scheduling impact on execution time can be mitigated by the policies employed on the scheduler.

The fair scheduler employs equal-priority for all applications making them have the same slowdown to enforce fairness. Eq. 1 shows the lower-is-better unfairness metric [35] that can be used to evaluate the fairness of the scheduler.

$$Unfairness = \frac{MAX(Slowdown_1, \dots Slowdown_n)}{MIN(Slowdown_1, \dots, Slowdown_n)} \quad (1)$$

Where n is the number of applications in the workload and $Slowdown_i = \frac{ET_{schedi}}{ET_{alonei}}$, where ET_{schedi} denotes the execution time of application i under a given scheduler, and ET_{alonei} is the execution time of the application i when executing alone.

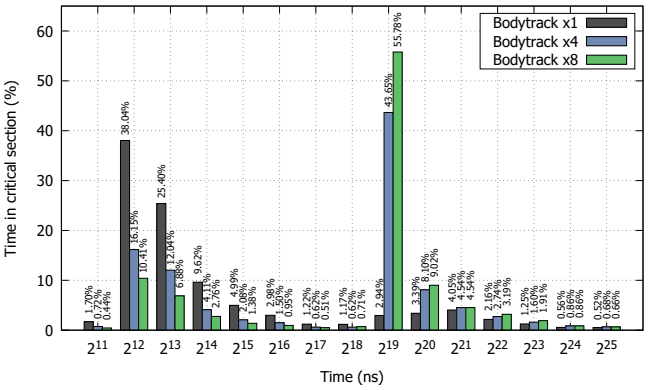


Fig. 10. Comparison of three application sets for the distribution of time spent in critical section on a RR scheduler with a timeslot of 1ms.

The baseline scheduler employs the RR policy that avoids starvation by running the application set in a deterministic order and uses a fixed share of execution time (timeslot). These features provide a fair distribution of execution runtime as all applications have the same number of timeslots regardless of the application type (i.e., low unfairness)[4]. The experimental unfairness values obtained will be discussed in the experimental results section.

Parallel applications can be roughly divided into sequential and parallel execution modes. Every parallel application contains at least a small sequential part for initialization, such as thread creation and parsing of application parameters. Mutual exclusion data access is another sequential execution commonly used among parallel portions. By using a mutex, either independently or associated with a condition, a thread is exclusively executing a given portion of code (i.e., a critical section) and potentially limiting all other threads. Consequently, delaying the execution of critical section code should be avoided to decrease the overall sequential time of an application.

Fig. 10 compares the critical section latency for three sets of the Bodytrack application: (i) standalone ($\times 1$), with four ($\times 4$), and with eight instances of Bodytrack ($\times 8$). The Y-axis is the percentage of overall critical section time on a given time interval (X-axis). The X-axis comprises the last value until the current value, except for the first case, where it starts at zero and goes until 2^{11} ns. For instance, the X value equals to 2^{12} comprises the time spent on a critical section of $[2^{11}, 2^{12})$ ns. As discussed previously in Section VI, all threads of Bodytrack, except the last one, participate in the synchronization scheme; since this example employs 64 threads, up to 63 threads share the same critical section.

We compare the same application on these three scenarios; consequently, the number of accesses into the critical section is approximately the same. However, the time spent into a critical section by a thread is not the same, as the scheduler

[4]Assuming the scheduler shares the same timeslot for all threads of a given application.

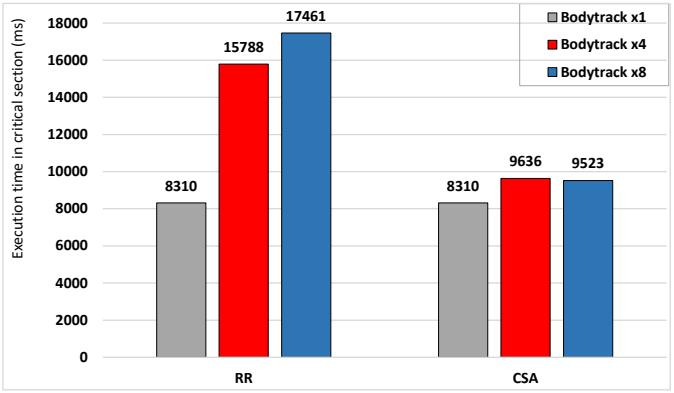


Fig. 11. Overall time spent in critical section for the sum of the work-related mutexes of Bodytrack on RR and CSA-enabled schedulers.

can interrupt the application execution. Fig. 10 shows that as the number of applications increases, the time each thread spent into a critical section also tends to increase since the scheduler does not differentiate execution on a sequential or parallel mode. The figure shows two situations where the time spent in critical sections spikes upward: from 2^{11} to 2^{12} ns and 2^{18} to 2^{19} ns. The first spike results from the application synchronization usage: this interval has the most number of cases for Bodytrack, as shown by the standalone case. The second spike happens only when multiple applications are introduced (Bodytrack$\times$4 and Bodytrack$\times$8). As we use a timeslot of 1 ms, the time spent in the critical section revolves around half this value (i.e., 2^{19}). The threads do not necessarily use the entire timeslot, as they can request to be scheduled out to wait for an event, for instance. Bodytrack has shown significant enough cases of such scenarios that the spike happens around half of the total timeslot. Nonetheless, this experiment shows that a scheduler based on the RR policy affects the synchronization latency and execution time.

Fig. 11 (legend RR) shows the total time spent in the critical section for the same set of applications. As expected, the time spent increases as more applications content for core usage. The behavior shown here by not taking note of the critical sections of parallel applications motivates the proposal of introducing the context of critical sections into the scheduler decisions. Therefore, the gains of utilizing Subutai can be maintained in a massive scheduler contention scenario. We call this proposal the **critical section-aware policy (CSA)**, whose results are depicted in Fig. 11 (legend CSA). The critical section execution time is kept as close as possible to the single application execution by applying the CSA policy.

A. Critical Section-Aware Policy (CSA)

We introduce CSA into the scheduler policies for executing critical section code as fast as possible. The policy works as follows. Every time a given thread has CSA enabled and is currently inside a critical section (i.e., holding a mutex), it has priority over the execution of all other threads that are not in the same scenario. In case another thread also has CSA enabled and is inside another critical section, an RR policy is applied to switch between them until either one finishes. Finally, if there are no threads that meet those requirements, an RR policy is applied to switch between the entire application set. We use RR as the baseline scheduler policy, yet more complex policies can also be applied.

Unfortunately, increasing the priority of a given thread over all others without any limitations generates two issues: (i) the scheduler deadlocks if the application also deadlocks; and (ii) it affects negatively on the performance of all other threads (i.e., high unfairness). Therefore, a time limit, defined in Eq. 2, was implemented in the CSA to deal with both issues.

$$CSALimit = (ThrReady + ThrRun - 1) \times (2 \times TS) \quad (2)$$

where $ThrReady$ and $ThrRun$ are the numbers of threads currently in the ready and running states, respectively. For both cases, the idle thread is ignored. TS is the chosen timeslot for the RR policy, generally in milliseconds. For instance, for a scheduler with a sum of 8 threads on the $ThrReady$ and $ThrRun$ states and a TS of 1ms, when one of these threads gains CSA priority, its time limit is 14ms.

$CSALimit$ has a direct proportionality between a parallel application execution time and the scheduler's unfairness. In other words, a high $CSALimit$ value will produce a fast parallel application execution for an unfair scheduler, and the opposite is also true. As we aim to keep the scheduler's fairness, we chose a limit that accelerates parallel application without increasing the scheduler's unfairness. Eq. 2 is a first empirical proposal, but a dynamic limit can be used to rebalance the CSA policy according to the scheduler profile. We chose the limits defined in Eq. 2 as it restricts the delay on other threads at most three times compared to the RR policy. When all threads are running on the RR policy, the maximum delay is $(ThrReady+ThrRun-1)\times TS$. Thus, the scheduler changes to the RR policy to maintain a fair scheduling, if the execution of the critical section reduces the priority of the other threads. The time limit deals with deadlock situations; however, to avoid livelocks, the scheduler requires a system-specified limit on the use of CSA policy for a given timeframe. Such methodology has been used effectively against other types of scheduler livelocks [36].

The fairness restriction of $CSALimit$ allows accelerating only a subset of critical sections; this is the reason why the critical section time is lower with Bodytrack $\times$8 than with Bodytrack $\times$4 (Fig. 11 - legend CSA). Table VII shows the impact of $CSALimit$ on the Bodytrack application set. The second, fourth, and fifth columns show the number of scheduling requests made outside of any critical section, inside a critical section with CSA enabled, and inside a critical section where RR has been enabled due to $CSALimit$, respectively. The third column shows that all scheduling requests for a critical section are either using CSA or RR policy. Approximately 10% and 8% of the total critical sections had CSA disabled as their time surpassed the $CSALimit$ time on Bodytrack $\times$4 and $\times$8, respectively. Even though we are analyzing the same Bodytrack, while running in a set of 4 and 8 applications, there are some discrepancies in the total number of requests for scheduling due to the use of synchronization primitives. Streamcluster and x264 presented

TABLE VII
IMPACT OF $CSALimit$ ON THE BODYTRACK APPLICATION SET EMPLOYING A TIMESLOT OF 1ms. CS = CRITICAL SECTION.

Application set	Schedule requests (not CS)	Schedule requests (CS)	CSA (CS)	RR (CS)
Bodytrack ×4	305517	CSA (CS) + CSA (RR)	15267	1558
Bodytrack ×8	323379		15274	1274

TABLE VIII
NI, SUBUTAI-HW AND SPM SYNTHESIS WITH 28 NM SOI.

Components	Area (μm^2)	Overhead
Basic NI	13539.23	–
Subutai FSM	2626.21	19 %
SPM	3702.00	27 %
Basic NI + Subutai-HW*	**19867.44**	**46 %**

*Subutai FSM + SPM

shorter critical sections on our experimental results, and they never triggered the $CSALimit$.

VIII. EXPERIMENTAL RESULTS

We demonstrate our solution results using a two-fold approach. Firstly, the system area and scalability of our solution are evaluated through an RTL implementation of Subutai-HW. Secondly, the system performance and scheduler are evaluated through architecture simulation and parallel applications from the PARSEC benchmark. Like Butko et al. [37], we employ the Gem5 simulator [38] to produce synchronization points of the applications; next, we feed this information into an in-house SystemC simulator [32], which enables us to collect experimental results. We run applications with and without Subutai: the former will henceforth be called Subutai, and the latter SW-only (i.e., Linux Kernel).

A. Area

Subutai-HW comprises a register-based NI, an FSM for synchronization control and linked pointer manipulation, and a 1 KiB SPM to store metadata and events. We use a very basic NI with 32-bit links, packing and unpacking logic, no virtual channel and 2 I/O buffers of 16×32 bits. It is worth noting that using HW synchronization operations releases valuable memory and cache space that would otherwise be required. Besides, the memory requirement is negligible if compared to a typical processor cache (less than 10%, if the cache size is 16 KiB). Table 8 summarizes the synthesis results showing our solution increases by 46% the basic NI area, including the local SPM; however, the overhead is amortized when the entire chip area is considered. For instance, using the Patel et al. [15] chip area of $400\mathrm{mm}^2$, the percentage of total area consumption of Subutai-HW is $\frac{64\times0.00632821}{400} = 0.101\%$, while the enhanced NI is $\frac{64\times0.01986744}{400} = 0.317\%$ for 64 cores. We synthesized all hardware elements using Synopsis DC [29] with 28 nm Silicon on Insulator (SOI) technology and 1 GHz clock frequency. Additionally, the SPM was synthesized with Cut Explorer [39].

B. State-of-the-Art Area Comparison

We compare our solution to those related work that provide enough data about the absolute area consumption (i.e., not in percentages) and technology used. Table IX depicts the area consumption of five hardware-based solutions. For a fairer analysis of the area consumption of each solution, we divided the total area consumed by the estimated number of cores in the system (i.e., area per core).

Subutai is second-to-last in terms of area consumed per core in the system. Additionally, Subutai and HTM have an additional area requirement per core; i.e., HTM needs to change the first cache level of the system for its functionality, and Subutai needs an SPM memory for synchronization handling. Even so, Subutai is third-to-last in terms of area consumption when both areas are combined. The hardware of Abellán et al. [13] has the overall smallest consumption as it is mainly comprised of wires and controllers. The last line of Table IX shows the estimation of area consumption for a $400\mathrm{mm}^2$ chip [15] for the same set of related work. Subutai only consumes approximately 0.1% of the total chip. Once again, it is third-to-last in overall area consumption.

C. Acceleration of Single Parallel Application

Fig. 12 shows the results obtained for the three PARSEC applications analyzed in this work. We analyze the entire application execution but plot the results for two threads for each application: the master thread (T_0), responsible for global synchronization, and a worker thread instance (T_7). Besides, the results are divided into two: synchronization operations and processing. The former aggregates all calls to PThreads (e.g., mutex lock), while the latter collects the processing needed by the application. NoC communication and Subutai-HW latencies did not contribute significantly for the execution time; thus, they are not visually perceivable on the figure, although they are present. Nonetheless, the figure shows that our solution reduces the application total time by handling synchronization faster.

From the designer point-of-view, the master thread (T_0) shows the effective speedup, as it is responsible for initializing and finalizing the application. Bodytrack achieved a speedup of $1.78\times$, and $1.77\times$ for 32 and 64 cores, respectively. Streamcluster achieved a speedup of $2.71\times$, and $2.20\times$ for the

TABLE IX
STATE-OF-THE-ART AREA CONSUMPTION.

	HTM [18]	MCAS [15]	Abellán et al. [13]	Notifying Memories [19]	**Subutai**
Area per core (mm^2)	0.32800	0.01824	0.00022	0.00534	**0.00262**
Additional area per core (mm^2)	0.01560	No	No	No	**0.00370**
Target Frequency (GHz)	Not addressed	3.40	0.62	0.50	**1.00**
Target System	8-core	32-core	64-core	12-core	**64-core**
Technology (nm)	65	14 (scaled)	45	65	**28**
Technique	Estimation	Synthesis	Synthesis	Synthesis	**Synthesis**
Overhead for a 64-core $400\mathrm{mm}^2$ chip	5.497 %	0.291 %	0.003 %	0.008 %	**0.101 %**

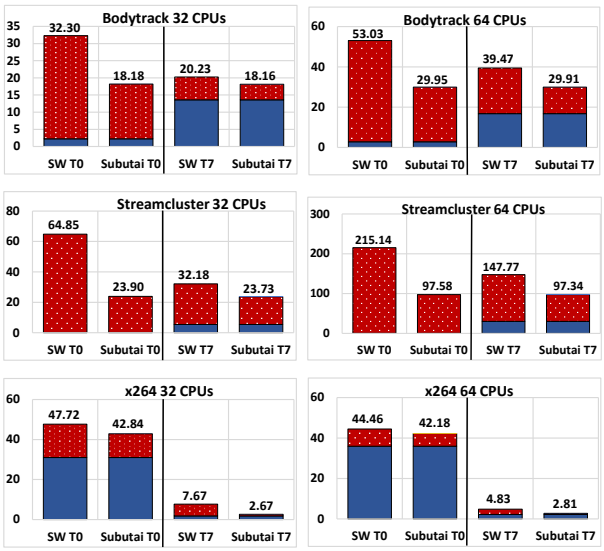


Fig. 12. Experimental results showing acceleration for a single parallel application. Values are in seconds of execution; the dotted red color is the sum of synchronization operations; the flat blue color is processing time.

same core set. Finally, x264 achieved a speedup of $1.11\times$ and $1.05\times$ for the same core set. Therefore, our solution achieved a speedup of $1.77\times$, on average. Table VI displays the number of synchronization calls, explaining the speedup difference; for instance, Streamcluster requires, roughly, 18, 23, and 31 times the equivalent of Bodytrack for 16, 32, and 64 cores, respectively. Thus, we can better optimize worker threads, as they are the ones using these primitives. The results also show that Bodytrack and Streamcluster are not scalable to 64 cores. Southern et al. [40] have independently corroborated this limitation as well. Our solution works the same regardless of the application scaling – as will be shown with a producer-consumer application on Section VIII-E.

The x264 application does not employ barriers because it uses hundreds of synchronization variables instead of dozens (Table V), and it does not have a logical dependency that involves all threads; therefore, x264 has less contented synchronization primitives. While Bodytrack and Streamcluster utilize synchronization in all worker threads, some of the worker threads of x264 have almost no synchronization; in turn, the application is not penalized with significant synchronization overhead. Another application like x264 from PARSEC, named Facesim, is available in [32], and it shows similar speed up results: $1.10\times$ and $1.27\times$ for 32 and 64 cores.

Our solution provides less direct benefit to x264 compared to the other two applications since it is designed to accelerate synchronization overhead. In other words, when synchronization primitives are not used to control most threads, their acceleration may not affect significantly the execution time since the synchronization may not be in the critical path.

Since we aim for legacy code compatibility, no changes have been made to any applications, either to increase the use of PThreads or to insert metadata for Subutai. Therefore, we target scenarios of running multiple applications to improve the speedup of our solution further.

D. Accelerating Multiple Parallel Applications

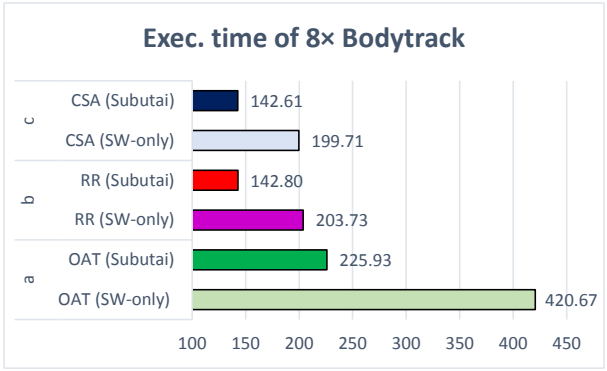
Fig. 13 displays the experimental results organized into sets of eight applications each: (a) eight instances of Bodytrack, (b) eight instances of Streamcluster, (c) eight instances of x264, and (d) a combination of 3, 2 and 3 instances of Bodytrack, x264, and Streamcluster, respectively. All applications have been set to use 64 threads and cores without restriction regarding mapping threads to cores.

Figs. 13a to 13d illustrate the entire execution time in seconds of an application set (i.e., from initialization to termination of all applications), comparing RR, CSA, and a One Application at a Time (OAT) scheduler. The latter scheduler is used for representing a mono application system (i.e., OAT can only execute one application). Lines `a` in Figs. 13a through 13d show that Bodytracks, Streamclusters, x264s, and mixed application sets have accelerations of $1.86\times$, $2.13\times$, $1.07\times$, and $1.91\times$, respectively, when running with Subutai compared to SW-only implementations with an OAT scheduler.

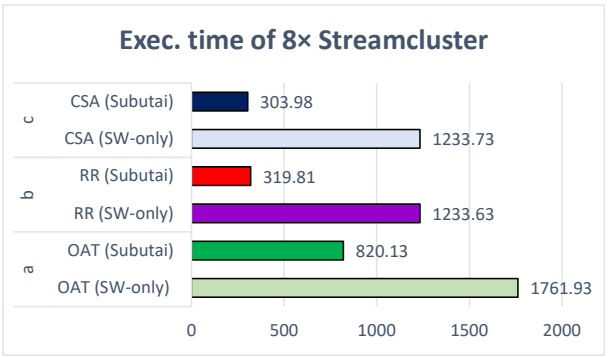
Additionally, lines `b` and `c` of Figs. 13a to 13d show that placing these applications in a competitive scheduling scenario increases the gains further because the idle time for a given application can be used as working time for another application; i.e., comparing CSA with OAT the speedups for Bodytracks, Streamclusters, x264s, and mixed applications are $1.58\times$, $2.69\times$, $4.61\times$, and $2.09\times$, respectively. The SW-only implementation has also presented gains, but the execution time of it is always higher compared to Subutai for the set of applications analyzed here. For the Streamcluster and mixed applications (Fig. 13b and 13d), executing them on Subutai with an OAT scheduler is faster than executing them on SW-only with either scheduler policies used in this work.

Table VII shows the impact of a scheduling policy restricted to critical sections is limited, as for an application such as Bodytrack, this section is approximately 5.12% of the total execution time. For Streamcluster and x264 application sets, not shown in Table VII, the critical section scheduling requests are 0.26% and 9.29% of the total number of requests, respectively. The set of Streamclusters with the CSA-enabled scheduler presented the highest speedup when compared to the same set of applications with an RR scheduler. Bodytrack and x264 presented a less significant speedup of less than $1.01\times$.

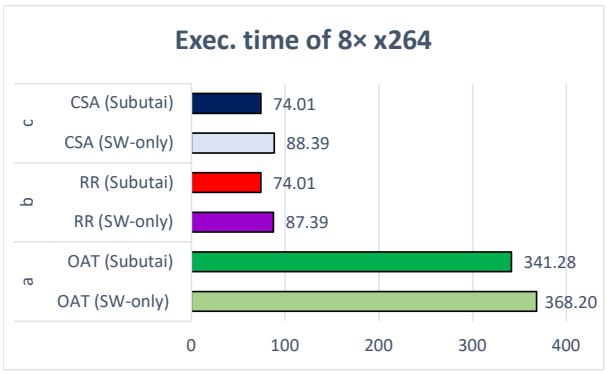
Table VI shows that Streamcluster has by far the most significant number of synchronization events of the application set. The number of synchronization events is a crucial factor for both Subutai and CSA in terms of their capacity to accelerate applications. For Subutai, these events are accelerated through the HW/SW co-design proposed by our work. For CSA, the same set of events are the only moments where it can apply its policy. Additionally, CSA relies on the premise that accelerating critical sections will decrease the overall execution time. This premise works well on barrier-based workloads, such as Streamcluster and Bodytrack, where the application is always working on the worst-case scenario (i.e., all worker threads blocked waiting for the slowest thread to join the barrier). However, pipeline applications, such as x264, can start working on new data as soon as the first thread



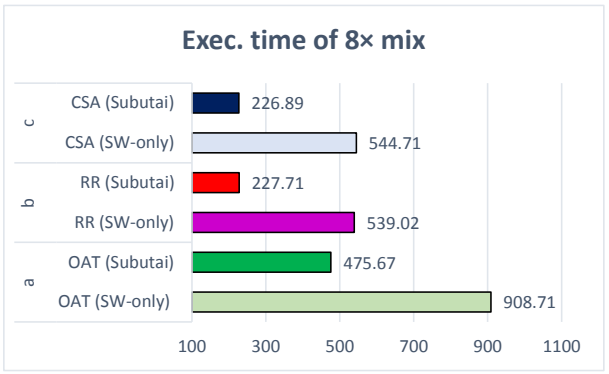
(a) Bodytrack application set (OAT × RR × CSA).



(b) Streamcluster application set (OAT × RR × CSA).



(c) x264 application set (OAT × RR × CSA).



(d) mix application set (OAT × RR × CSA).

Fig. 13. Execution in seconds for multiple eight-application sets (lower is better) - (Exec = Execution).

TABLE X
UNFAIRNESS METRIC FOR CSA AND RR SCHEDULERS (LOWER IS BETTER).

Application set	SW-only		Subutai	
	RR	CSA	RR	CSA
Bodytrack ×8	1.04	1.04	1.16	1.15
Streamcluster ×8	1.11	1.11	1.19	1.19
x264 ×8	1.27	1.24	1.12	1.20
mix ×8	2.00	1.71	1.88	1.83

has finished; therefore, CSA has a lesser impact on such applications.

Table X presents the unfairness metric. For all cases, CSA either maintains or decreases the unfairness of the scheduler for the application set, except for x264. Nonetheless, Fig. 13c shows that x264 has the same overall execution time in both cases. Consequently, these results indicate that the use of the CSA policy keeps the fairness of the baseline scheduler.

E. Synthetic Benchmark

The results presented in the previous sections provide a systemic view of Subutai, but they do not convey the optimization in the synchronization itself. The lack of a microcosm view happens because the applications use at least thousands of synchronization primitives during their execution. Consequently, we employ a one producer many consumers synthetic application encompassing a few calls to the three synchronization primitives (mutex, barrier, and condition) using six threads. Table XI shows the average absolute time of Subutai and SW-only for these primitives.

Subutai speeds up significantly every synchronization primitive compared to the SW-only implementation. The comparison is made from the application perspective; for instance, the condition broadcast and mutex unlock operations have no response packet; consequently, Subutai can return to the application immediately after the request packet is sent. Thus, the processing is offloaded to the HW, and the primitive is handled faster from the caller perspective. The SW-only implementation depends on the following costs to handle synchronization primitives (Fig. 3): (i) context switching; (ii) synchronization for queue operations; and (iii) kernelspace switching. Item (i) is reduced in Subutai by using a distributed OS. As stated in Section III-A, we can use a faster context switch with a distributed OS. The faster OS is useful for functions that are blocking, and every group handled by Subutai has these functions. Item (ii) is reduced by offloading

TABLE XI
RESULTS FOR PRODUCER-CONSUMER APPLICATION.

Primitive	Type	Avg. SW	Avg. Subutai
Mutex	Lock empty	1537 ns	127 ns
	Lock queued	64178 ns	916 ns
	Unlock	4400 ns	60 ns
Barrier	Wait (released)	102467 ns	1183 ns
Condition	Broadcast	25209 ns	60 ns
	Queued	42844 ns	1022 ns

all queue operations to hardware. Finally, item (iii) is not present in our OS. Subutai adds the cost of I/O operations to deal with Subutai-HW (Fig. 4), which is not present in the SW-only solution. Nonetheless, these factors explain the gains shown in Table XI.

IX. Conclusion

This paper presents Subutai, an HW/SW co-design solution for accelerating legacy and novel parallel applications through data synchronization. Unlike other synchronization solutions [1] [9] [15], our approach does not require any user-level modification, such as source code changes. Subutai overrides the shared library of PThreads while maintaining its functionality and API. Ergo, any binary using PThreads for data synchronization can benefit from the proposed solution.

Subutai relies on hardware-handled operations to accelerate common synchronization techniques found on parallel applications. By doing so, the overall execution time speeds up to $1.77\times$, on average. Besides, we show that our solution is efficient in the general case of multiple applications sharing computing resources as we propose the CSA scheduling policy to accelerate applications further on a resource-contention scenario by providing priority to threads that are currently running in a critical section. We have implemented this policy using an approach that improves the balance of the scheduler (i.e., low unfairness), making the policy highly portable across different scheduling techniques. Even with such limitations, we achieved a speedup of up to $4.61\times$ for shared-memory parallel applications.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) -– Finance Code 001.

References

[1] C. DeLozier, A. Eizenberg, B. Lucia, and J. Devietti, "SOFRITAS: Serializable Ordering-Free Regions for Increasing Thread Atomicity Scalably," in *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 286–300.

[2] P. McKenney, *Is Parallel Programming Hard, And, If So, What Can You Do About It?*, kernel.org, Corvallis, USA, 2019.

[3] E. Sperling, "How Much Will That Chip Cost?," http://semiengineering.com/how-much-will-that-chip-cost/, Aug. 2020.

[4] P. McKenney, S. Boyd-Wickizer, and J. Walpole, "RCU Usage In the Linux Kernel: One Decade Later," https://pdos.csail.mit.edu/6.828/2017/readings/rcu-decade-later.pdf, Aug. 2020.

[5] S. McConnell, *Code Complete, Second Edition*, Microsoft Press, Redmond, USA, 2004.

[6] K. Fürlinger, T. Fuchs, and R. Kowalewski, "DASH: A C++ PGAS library for distributed data structures and parallel algorithms," *CoRR*, vol. 1610.01482, 2016.

[7] IEEE, "IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7," *IEEE Std 1003.1-2017 (Rev. of IEEE Std 1003.1-2008)*, pp. 1–3951, Jan 2018.

[8] R. Cataldo, R. Fernandes, K. Martin, J. Sepulveda, A. Susin, C. Marcon, and J.-P. Diguet, "Subutai: Distributed Synchronization Primitives in NoC Interfaces for Legacy Parallel-applications," in *Annual Design Automation Conf. (DAC)*, 2018, pp. 83:1–83:6.

[9] M. Desnoyers, P. McKenney, A. Stern, M. Dagenais, and J. Walpole, "User-Level Implementations of Read-Copy Update," *IEEE Trans. on Parallel and Distributed Systems*, vol. 23, no. 2, Feb 2012.

[10] H.-J. Boehm, "Reordering Constraints for Pthread-style Locks," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2007, pp. 173–182.

[11] M. France-Pillois, J. Martin, and F. Rousseau, "Optimization of the GNU OpenMP Synchronization Barrier in MPSoC," in *International Conference of Architecture of Computing Systems (ARCS)*, Apr. 2018, pp. 57–69.

[12] R. Sivaram, C. Stunkel, and D. Panda, "A Reliable Hardware Barrier Synchronization Scheme," in *Intl. Parallel Processing Symp. (IPPS)*, Apr 1997, pp. 274–280.

[13] J. Abellán, J. Fernández, M. Acacio, D. Bertozzi, D. Bortolotti, A. Marongiu, and L. Benini, "Design of a Collective Communication Infrastructure for Barrier Synchronization in Cluster-Based Nanoscale MPSoCs," in *Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2012, pp. 491–496.

[14] C. Stoif, M. Schoeberl, B. Liccardi, and J. Haase, "Hardware Synchronization for Embedded Multi-Core Processors," in *IEEE Intl. Symp. of Circuits and Systems (ISCAS)*, May 2011.

[15] S. Patel, R. Kalayappan, I. Mahajan, and S. Sarangi, "A Hardware Implementation of the MCAS Synchronization Primitive," in *Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2017, pp. 918–921.

[16] T. Gangwani, A. Morrison, and J. Torrellas, "CASPAR: Breaking Serialization in Lock-Free Multicore Synchronization," *SIGPLAN Not.*, vol. 51, no. 4, pp. 789–804, Mar. 2016.

[17] N. Diegues, P. Romano, and L. Rodrigues, "Virtues and Limitations of Commodity Hardware Transactional Memory," in *Intl. Conf. on Parallel Architecture and Compil. Tech. (PACT)*, 2014, pp. 3–14.

[18] A. Shriraman, S. Dwarkadas, and M. Scott, "Flexible Decoupled Transactional Memory Support," in *Intl. Symp. on Computer Architecture (ISCA)*, 2008, pp. 139–150.

[19] K. Martin, M. Rizk, M. Sepulveda, and J.-P. Diguet, "Notifying Memories: a case-study on Data-Flow Applications with NoC Interfaces Implementation," in *Design Automation Conf. (DAC)*, 2016, pp. 1–6.

[20] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*, O'Reilly Media, 1 edition, July 2007.

[21] H. Attiya, R. Guerraoui, D. Hendler, P. Kuznetsov, M. Michael, and M. Vechev, "Laws of Order: Expensive Synchronization in Concurrent Algorithms Cannot Be Eliminated," in *ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)*, 2011, pp. 487–498.

[22] C. Kirsch, M. Lippautz, and H. Payer, "Fast and scalable, lock-free k-FIFO queues," in *Intl. Conf. on Parallel Computing Technologies (PaCT)*, 2013, pp. 208–223.

[23] C. Cascaval, C. Blundell, M. Michael, H. Cain, P. Wu, S. Chiras, and S. Chatterjee, "Software Transactional Memory: Why Is It Only a Research Toy?," *Queue*, vol. 6, no. 5, pp. 40:46–40:58, Sept. 2008.

[24] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *Intl. Symp. on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[25] J. Hennessy D. Patterson, *Computer Organization and Design: The Hardware Software Interface [RISC-V Edition]*, vol. 1, Morgan Kaufmann, 1st edition, 2017.

[26] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The Multikernel: A New OS Architecture for Scalable Multicore Systems," in *ACM SIGOPS Symp. on OS Principles (SOSP)*, 2009, pp. 29–44.

[27] Ulrich Drepper, "Futexes Are Tricky," https://cis.temple.edu/~giorgio/cis307/readings/futex.pdf, Aug. 2020.

[28] Mentor, "ModelSim ASIC and FPGA Design – Mentor Graphics," www.mentor.com/products/fv/modelsim/, Aug. 2020.

[29] Synopsys, "DC Ultra," www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, Aug. 2020.

[30] Linux man page, "futex(2)," https://linux.die.net/man/2/futex, Aug. 2020.

[31] H. Franke, R. Russell, and M. Kirkwood, "Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux," in *Ottawa Linux Summit*, 2002, pp. 479–495.

[32] R. Cataldo, *Subutai: Distributed Synchronization Pritimives for Legacy and Novel Parallel Applications*, Ph.D. thesis, Université Bretagne-Sud, Lorient, France, 2019.

[33] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," Tech. Rep., Princeton University, 01 2008.

[34] R. Cataldo, "Design and Exploration of 3D MPSoCs with on-Chip Cache Support," M.S. thesis, Pontifícia Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2015.

[35] A. Garcia-Garcia, J. Saez, and M. Prieto-Matias, "Contention-Aware Fair Scheduling for Asymmetric Single-ISA Multicore Systems," *IEEE Transactions on Computers*, vol. 67, no. 12, pp. 1703–1719, Dec 2018.
[36] G. Nakagawa and S. Oikawa, "Fork Bomb Attack Mitigation by Process Resource Quarantine," in *Intl. Symp. on Computing and Networking (CANDAR)*, 2016, pp. 691–695.
[37] A. Butko, R. Garibotti, L. Ost, V. Lapotre, A. Gamatie, G. Sassatelli, and C. Adeniyi-Jones, "A Trace-driven Approach for Fast and Accurate Simulation of Manycore Architectures," in *Asia and South Pacific Design Automation Conf. (ASPDAC)*, 2015, pp. 707–712.
[38] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, and D. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.
[39] STMicroelectronics, "Standard Technology offers at CMP in 2017 Deep Sub-Micron, SOI, and SiGe Processes," https://mycmp.fr/IMG/pdf/2017_cmp_usersmeeting_st_mpw_services.pdf, Dec. 2020, slide 16.
[40] G. Southern and J. Renau, "Deconstructing PARSEC Scalability," *Annual Workshop on Duplicating, Deconstructing and Debunking*, vol. 1, no. 1, pp. 1–10, 2015.

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud
ubs

Titre : Déploiement d'applications parallèles sur architectures parallèles

Mot clés : Architectures numériques, Parallélisme, CGRA, processeur multi-cœurs, déploiement, compilation

Résumé : Avec le ralentissement de la loi de Moore, et la fin de la course à la fréquence, l'amélioration des performances vient du parallélisme. Plusieurs types de parallélisme peuvent être exploités par les architectures de calcul. Nous nous concentrons sur le parallélisme au niveau instruction, parallélisme de données, et le parallélisme de tâche. La multiplication du nombre de ressources de calcul permet de proposer du parallélisme spatial. La question qui se pose est le déploiement des applications sur les architectures parallèles, pour exploiter au mieux les ressources disponibles.

Dans nos travaux, nous étudions tout d'abord des architectures reconfigurables à gros grains, permettant d'exploiter le parallélisme au niveau instructions et le parallélisme de données. Nous avons proposé une architecture et une approche originale de déploiement d'application incluant le flot de contrôle et la gestion des boucles pour en faire un accélérateur autonome embarqué à haute efficacité énergétique. Nous avons aussi proposé des algorithmes de déploiement d'applications, décrites selon le modèle de calcul dataflow, sur des architectures multi-cœurs, permettant d'exploiter le parallélisme de données et le parallélisme de tâche. Enfin, la synchronisation est un point clé des performances sur architectures parallèles. Nous avons proposé des solutions matérielles pour accélérer la synchronisation des applications dataflow, et des applications suivant la norme Pthread.

Title: Mapping parallel applications on parallel architectures

Keywords: Parallelism, CGRA, multicore processor, mapping, compilation

Abstract: With the slowdown of Moore's law and the end of the frequency race, the performance comes from the parallelism. Several types of parallelism can be exploited by computing devices. Our work focuses on instruction-level parallelism, data-level parallelism, and task-level parallelism. Multiplying the number of computing resources exposes spatial parallelism. The question is to find how to map the applications on parallel architectures, to fully make use of the available computing resources.

In our work, we first study Coarse Grained Reconfigurable Architectures (CGRA) which can exploit instruction- and data-level parallelism. We proposed a new architecture and an original mapping approach that can manage the control flow of an application, including the loop control, to build an ultra-low power and energy efficient programmable accelerator for embedded systems. We also developed a runtime mapping algorithm for mapping parallel applications following the dataflow model of computation on multicore processors, making use of data- and task-level parallelism. Finally, synchronisation is a key issue for the performance of parallel architectures. We designed hardware solutions for accelerating synchronisation for dataflow applications, and for Pthread applications.