Exploration de l'espace de conception des architectures reconfigurables

Lilian Bossuet, Guy Gogniat, Jean Luc Philippe

Exploration de l'espace de conception des architectures reconfigurables

Lilian Bossuet[2] – Guy Gogniat[1] – Jean-Luc Philippe[1]

[1] *LESTER – Université de Bretagne Sud*
Centre de Recherche – BP 92116
56321 Lorient cedex
guy.gogniat@univ-ubs.fr

[2] *IXL – Université de Bordeaux 1*
351 Cours de la Libération
F-33405 Talence cedex
bossuet@ixl.fr

RÉSUMÉ. *Entre les solutions logicielles programmées très flexibles et les solutions matérielles spécifiques très performantes il existe une autre possibilité offerte aux développeurs de systèmes électroniques numériques : les solutions reconfigurables. Celles-ci sont aujourd'hui de réelles alternatives aux solutions spécifiques puisque les avancées technologiques ont permis via l'augmentation de l'intégration d'imaginer des solutions reconfigurables performantes pouvant intégrer des systèmes complets. Ainsi le concept de systèmes sur puce reconfigurable est né. Ces solutions reconfigurables ouvrent de plus de nouveaux domaines d'applications qui profitent de leurs caractéristiques de reconfiguration en particulier la reconfiguration dynamique. La recherche d'architectures performantes dans le contexte reconfigurable bute cependant sur un manque d'outils d'estimation permettant à haut niveau d'abstraction d'explorer le large espace de conception de ces systèmes reconfigurables. Dans cet article nous proposons de faire l'état de l'art de cette problématique en apportant la vision que nous développons.*

ABSTRACT. *During many years, the designers just had two possibilities to design embedded systems; microprocessors and/or application specific integrated circuits. Nevertheless, a new possibility has appeared lately, the reconfigurable circuits. These systems have an important lack of tools at all design levels. Designers are then facing the difficult task of designing their target reconfigurable architectures, which is a critical issue since it can strongly affect the final system performances. To help them in that task, it is necessary to develop tools that enable an efficient design space exploration. With such tools, designers could improve synergy between application and architecture and then choose the best one for their application. In this article, we propose a design space exploration method for reconfigurable architectures dedicated to data intensive applications. The design space exploration takes place at the first stages of the design flow before any architectural definition.*

MOTS-CLÉS : *Architecture des systèmes reconfigurables, Estimation haut niveau des performances, Exploration de l'espace de conception.*

KEYWORDS: *Reconfigurable System Architecture, High-Level Performance Estimation, Design Space Exploration.*

1. Introduction

Longtemps les concepteurs de systèmes numériques étaient face à une alternative lors du choix d'intégration de leurs systèmes : solution logicielle programmable (microprocesseurs, DSP) *vs* solution matérielle spécifique (ASIC). Dans ce cadre à alternative unique, les contraintes des applications déterminent rapidement le meilleur choix, puisque les solutions logicielles sont très flexibles mais peu performantes et que les solutions matérielles ont des caractéristiques opposées. Il apparaît que le fossé entre solution logicielle et matérielle doit être comblé par une solution intermédiaire flexible et performante. C'est dans cette problématique que les architectures reconfigurables sont aujourd'hui développées. Mais le concept de circuit reconfigurable fut introduit initialement dans d'autres buts : le prototypage et l'émulation sous la forme des FPGA. Cependant les évolutions technologiques (réductions de la taille des transistors, augmentation du nombre de couches de métallisation) ont permis d'amener les capacités d'intégrations des FPGA au niveau des besoins moyens des applications modernes. Les évolutions technologiques (passage de la technologie aluminium à cuivre, diminution des tensions d'alimentation) ont permis d'augmenter les performances de ces mêmes circuits. Enfin les évolutions technologiques (augmentation de la taille des wafers) ont permis de réduire les coûts de revient des éléments logiques reconfigurables de base ce qui couplé avec l'augmentation des capacités permet de disposer de circuits FPGA plus gros et plus performants à coûts constants d'une génération à une autre. Ainsi au fil des évolutions les FPGA sont devenus une alternative réaliste aux ASIC.

Bien que très largement utilisés, les circuits FPGA ont tout de même de sérieux handicaps (en particulier en ce qui concerne la consommation de puissance) liés aux ressources de routages trop pénalisantes, et une granularité de traitement réduite au niveau bit via des fonctions logiques. Pour remédier à ces problèmes de nombreux laboratoires universitaires ou industriels se sont lancés dans la définition d'architectures reconfigurables plus efficaces. L'efficacité d'une architecture peut être vue sous bien des points, tels que les performances (vitesse, consommation), mais aussi la flexibilité (liée au potentiel de reconfiguration) ou encore l'adéquation à un domaine d'application.

Les architectures reconfigurables sont aujourd'hui au centre d'un important domaine de recherche en pleine ébullition et elles représentent une réponse intéressante au challenge des systèmes sur puce [Hartenstein01]. Elles ont permis l'élaboration de nouvelles applications tirant parti de leurs caractéristiques propres tels qu'un fort parallélisme matériel et des possibilités de reconfiguration statique et/ou dynamique (c'est à dire en cours d'exécution). Par exemple le domaine de la Radio Logicielle pour lequel les architectures reconfigurables dynamiquement apportent une réponse pour relever le défi de la multiplicité des standards de communication et de leurs complexités intrinsèques [Delahaye04]. Les architectures reconfigurables sont donc au centre d'une importante révolution technologique de l'électronique numérique. Enfin, les architectures reconfigurables, de grain fin et de

gros grain, sont aujourd'hui définies avec de plus en plus d'hétérogénéité afin de répondre aux problèmes des systèmes sur puce. Les ressources de communications, des bus aux réseaux filaires de routages, sont souvent intégrées conjointement au sein d'une même architecture afin de faciliter les communications locales comme globales. Les systèmes sur puce configurables mettent le plus souvent en œuvre des zones programmables (cœurs de processeurs) couplées plus ou moins étroitement avec des zones reconfigurables (de grain fin et/ou de gros grain). Les structures mises en œuvre sont le plus souvent hiérarchiques mais peuvent avoir des topologies différentes pour des besoins de déroulement des traitements ou pour les communications. Le domaine des architectures reconfigurables est donc comme on le voit une véritable jungle (comme le disaient les auteurs de [Shaumont01] "*A Quick Safari Through the Reconfigurable Jungle*") dans laquelle le concepteur aventurier a bien besoin qu'on le guide.

La problématique de l'étude décrite dans cet article est la suivante ; face à la multitude d'architectures reconfigurables hétérogènes (de gros grain et de grain fin) potentielles, est-il possible de développer une méthode permettant rapidement de converger vers la définition d'une architecture reconfigurable efficace (performante, flexible, facile à mettre en œuvre) pour une application ou un domaine d'applications donné et ceci très tôt dans le flot de conception. Aujourd'hui peu d'architectures reconfigurables de gros grain sont effectivement réalisées sur silicium ce qui met le concepteur soucieux de cibler une architecture flexible et évolutive, face à un manque d'informations pour définir correctement sa cible architecturale. Il lui faut donc un guide qui puisse intervenir très tôt dans le flot de conception afin de lui réduire le champ d'investigation pour converger rapidement vers une solution architecturale efficace pour l'application développée.

La suite de cet article est constituée de trois parties, dans la première partie (chapitre 2) nous définirons les caractéristiques des architectures reconfigurables afin de définir plus globalement leur espace de conception. Dans une deuxième partie (chapitre 3) nous ferrons une étude des méthodes d'exploration d'espace de conception proposées aujourd'hui dans la littérature en présentant des travaux qui ciblent des architectures reconfigurables (du FPGA aux architectures reconfigurables de gros grain). Enfin nous proposons dans une dernière partie (chapitre 4) une méthode générique d'exploration qui s'appuie sur l'environnement d'aide à la conception des systèmes sur puces *Design Trotter* développé au sein du laboratoire LESTER. Cette méthode vise à définir pour un domaine d'applications une architecture reconfigurable efficace du point de vue de la consommation de puissance.

2. Espace de conception des architectures reconfigurables

2.1 Définition

Le domaine des architectures reconfigurables n'a pas de frontières bien définies

car cela dépend des définitions que l'on donne aux notions d'architecture et de reconfigurabilité. Nous proposons donc aux lecteurs des définitions qui sont celles que nous utiliserons par la suite.

Dans notre cas l'architecture est celle interne aux systèmes reconfigurables sur puce (RSoC : *Reconfigurable System on Chip*). L'architecture décrit l'agencement et les communications entre les éléments de traitement, de mémorisation et de contrôle. C'est donc à la fois la description du schéma de calcul et du plan de communication. Il n'en reste pas moins qu'il s'agit d'une vision au niveau registre du circuit.

L'adjectif reconfigurable se rapporte évidemment au terme architecture défini précédemment. Si l'architecture décrit les éléments de traitement, de mémorisation et de contrôle ainsi que les ressources de communication, alors l'adjectif reconfigurable peut se rapporter à chacun d'eux. La configuration décrit le potentiel pour un élément d'avoir plusieurs états possibles et distincts de fonctionnement. La reconfiguration décrit la possibilité pour un élément de changer d'état (donc de configuration) au cours du temps par un processus déterministe. Effectivement le changement de configuration ne doit être dû qu'à un ordre bien précis de reconfiguration et il doit être possible de revenir dans un état précédent. Certains circuits sont uniquement configurables une seule fois à cause de la technologie utilisée pour mémoriser la configuration (technologie antifusible par exemple), nous considérons ces circuits comme configurables et non comme reconfigurables. Une architecture peut être qualifiée de reconfigurable si au moins une partie des éléments qu'elle décrit est reconfigurable, telles que les ressources de traitement ou les ressources de communication.

2.2. Caractéristiques des architectures reconfigurables

Pour pouvoir explorer l'espace de conception des architectures reconfigurables, il faut pouvoir les caractériser. Cinq caractéristiques essentielles peuvent être définies:

La granularité des ressources de traitement est employée pour caractériser l'architecture, une distinction est souvent faite entre grain fin et gros grain (appelé également grain épais). Le grain fin correspond à des ressources logiques le plus souvent de type LUT (Look Up Table). Le gros grain se rapporte le plus souvent à des architectures du type systolique lorsqu'elles sont construites autour de matrices d'ALU, voir de cœurs de processeurs élémentaires. Certaines architectures sont composées d'opérateurs arithmétiques de type additionneur, multiplieur ou MAC. Le plus souvent les architectures reconfigurables sont hétérogènes, c'est à dire qu'elles ont des ressources de traitement de granularité différente. Il n'est pas rare d'avoir une partie de grain fin qui pourra efficacement traiter des manipulations binaires et des parties de granularité plus importante pour les calculs arithmétiques. Nous verrons dans le développement de cet article qu'il s'agit d'une tendance confirmée par les résultats de notre étude.

La granularité des ressources de communication peut, elle aussi, être plus ou moins fine. Typiquement il y a trois possibilités pour réaliser les communications

[Zhang99] ; Les crossbars qui sont des dispositifs de communication offrant une connectivité complète, les réseaux de bus qui sont des dispositifs partagés, et les réseaux de connexions locales point à point.

La topologie du réseau de communication décrit comment les lignes de communication (bus ou canaux de fils) sont disposées dans l'architecture et comment les éléments de connexion sont agencés. Par exemple les FPGA de la société Xilinx [Xilinx04] ont une topologie de type matricielle, les éléments à connecter sont disposés sous forme de matrice, les connexions sont verticales et horizontales. Certaines architectures ont vocation à être particulièrement adaptées aux calculs pipelinés, alors une topologie de type matricielle associée à un réseau linéaire de communications (horizontales, verticales ou circulaires) permet de faire évoluer les données dans le sens des calculs. Enfin, le plus souvent les architectures mélangent les aspects globaux et locaux on parle d'architectures hiérarchiques.

La liaison avec le microprocesseur n'a d'objet que dans le cadre où l'architecture reconfigurable joue le rôle d'accélérateur matériel. Le couplage entre la partie logicielle et la partie matérielle peut se faire de façon direct, via un bus ou une interface [David03].

La reconfiguration des architectures peut prendre différentes formes, que ce soit pour reconfigurer les ressources opératoires ou que ce soit pour reconfigurer les réseaux de communication. Tout d'abord il est possible de caractériser la reconfiguration en considérant le nombre minimum M_{ec} d'éléments configurables à chaque reconfiguration par rapport au nombre total T_{ec} d'éléments configurables, le rapport est nommé γ :

Si $\gamma = \frac{M_{ec}}{T_{ec}} < 1$ alors la configuration est partielle c'est à dire qu'il est possible de configurer seulement une partie de l'architecture sans remettre en cause le reste de la configuration. Si ce n'est pas le cas, la configuration est dite complète. Le rapport γ peut varier dans l'intervalle $[1/T_{ec} ; 1]$. Plus γ est proche de sa borne inférieure plus il est possible de modifier finement l'architecture de façon partielle. Par exemple l'architecture matricielle grain fin du FPGA Virtex de la société Xilinx [Xilinx04] supporte une reconfiguration partielle, dans ce cas M_{ec} est égal au nombre d'éléments logiques configurables compris dans une colonne. De plus la reconfiguration peut être statique ou dynamique, simple ou multi-contextes [Trimberger97].

3. Méthode d'exploration de l'espace de conception des architectures reconfigurables

3.1. Méthodes d'exploration de l'espace de conception

Le but de l'exploration de l'espace de conception est de faire correspondre deux espaces, celui du problème ou espace de conception, et celui des objectifs ou contraintes de l'application. L'exploration vise donc l'adéquation de l'architecture avec l'application développée. L'exploration peut intervenir à tous les niveaux d'abstraction du flot de conception comme on peut le voir sur la figure 1 [Pimentel01]. L'espace à explorer peut être raffiné à chaque niveau, l'exploration est donc un processus itératif et hiérarchique. La pyramide de la figure 1 a pour l'exemple quatre niveaux d'abstraction différents, ces niveaux ne sont pas les mêmes suivant le type de système visé. Pour chaque niveau une exploration de l'espace de conception est mise en œuvre afin de réduire cette espace jusqu'à converger vers une architecture cible.

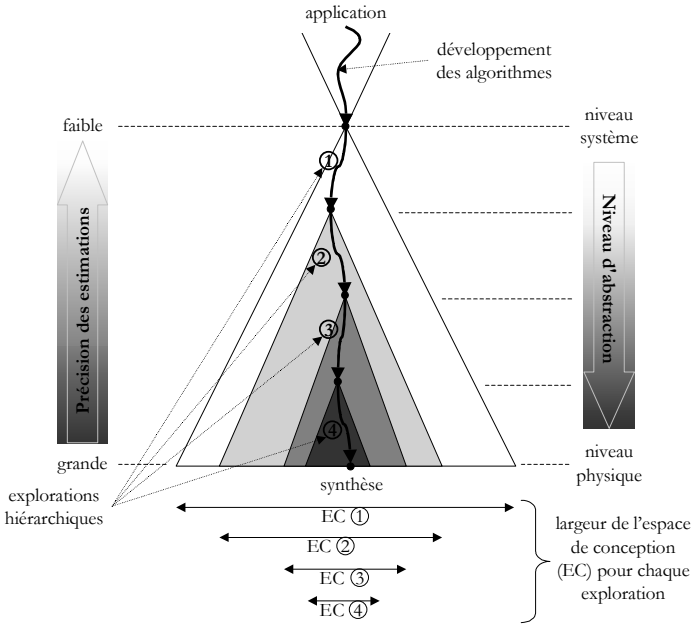

Figure 1. *Pyramide des niveaux d'abstraction du flot de conception et positionnement de l'exploration hiérarchique de l'espace de conception.*

Dans le cas des architectures matérielles, il existe deux grandes méthodes pour l'extraction des performances et l'exploration de l'espace de conception ;

Synthétiser puis comparer : cette méthode met en œuvre un flot de synthèse complet de l'application sur chacune des architectures à comparer. Ainsi il est possible d'obtenir une mesure très précise des performances. Cependant la synthèse complète met en jeu des algorithmes complexes qui demandent un temps long pour obtenir un résultat. De ce fait il n'est pas possible d'envisager l'exploration d'un large espace de conception mais il est nécessaire de concentrer la recherche sur un espace restreint. De plus, il faut disposer d'outils de synthèse pour chaque

architecture. L'utilisateur est obligé de limiter l'exploration aux architectures couplées à des outils, ce qui réduit l'espace d'exploration. Cependant, une possibilité est de disposer d'outils génériques comme dans [Lagadec01] et [Betz97], mais mêmes génériques les outils ciblent un ensemble architectural, donc un espace de conception réduit.

Estimer puis comparer : Afin de combler les défauts de la précédente méthode, cette méthode remplace la phase de synthèse par des estimations de performances. Les algorithmes mis en place pour les estimations sont moins complexes et plus rapides que ceux utilisés lors de la synthèse. Ainsi cette méthode permet de parcourir rapidement un large espace de conception [Bossuet03]. Les algorithmes d'estimation sont plus flexibles que les algorithmes de synthèse, surtout lors d'estimation à un haut niveau d'abstraction. Cependant les performances estimées peuvent être peu précises, mais la précision n'est pas le critère essentiel lorsque l'on cherche uniquement la comparaison des différentes solutions. Il est par contre indispensable d'assurer une erreur relative constante, l'estimation doit donc avoir une bonne fidélité dans ses résultats. Ainsi les performances estimées sont relatives et n'ont d'intérêts que dans la comparaison.

Ces deux méthodes sont complémentaires et vont être utilisées à des niveaux différents d'abstraction. A haut niveau, il existe peu d'outils de synthèse, et l'espace à explorer est très large, de ce fait il faut utiliser une méthode *estimer et comparer* pour réduire l'espace rapidement. Avec un niveau d'abstraction faible l'espace de conception est beaucoup moins large. Dans ce cas, il faut converger vers une solution fiable ce qui demande une précision que la méthode *synthétiser et comparer* apporte. Un flot d'exploration met donc en œuvre différentes méthodes afin de raffiner l'espace de conception avant de spécialiser les outils pour converger vers la meilleure solution possible.

L'exploration qui est fait avec ces deux méthodes peut être multi-objectifs ou mono-objectif afin de mieux répondre aux contraintes du système et/ou de diminuer le nombre de dimensions de l'espace des objectifs. Plus le nombre d'objectifs est important plus le processus d'exploration est complexe et lent. De ce fait, dans le cas multi-objectifs il n'est pas certain de converger vers une unique solution. Pour réduire le nombre d'objectifs il est souvent possible d'établir une fonction de coûts dans laquelle plusieurs objectifs interviennent.

3.2. *Comparatif de travaux visant les architectures reconfigurables*

Ce paragraphe va donner les caractéristiques de quelques travaux portant sur l'exploration de l'espace de conception ciblant des architectures reconfigurables. Ceux-ci sont regroupés dans trois catégories : les outils génériques de placement et routage ciblant des architectures grains fin du type FPGA, les outils d'estimations de performances ciblant aussi des FPGA et des outils d'exploration architecturale ciblant des architectures reconfigurables de gros grain.

3.2.1. Outils génériques de placement routage pour FPGA

Des outils de placement et routage peuvent être mis à profit dans une approche d'exploration architecturale des FPGA en particulier en ce qui concerne les ressources de routage nécessaires pour réaliser une application donnée. Ces outils s'inscrivent de par leur nature dans une approche du type *synthétiser puis comparer*.

L'outil VPR (Versatile Place and Route) développé à l'Université de Toronto au Canada est un exemple très intéressant. L'architecture modélisée est du type FPGA îlot de calcul avec des clusters de taille variable. La modélisation est physique et s'appuie sur un modèle physique P-Spice [Betz97]. Ce modèle physique permet de décrire des paramètres tels que la technologie utilisée, le type et la taille des ressources de routage utilisées, le type et le nombre de matrice de connexions des réseaux de routage et de connexions aux éléments logiques de base, l'architecture et la taille de ces derniers. Cette description est donc très précise ce qui donne une grande précision aux résultats d'estimation obtenus. VPR a un mode de fonctionnement automatique dans lequel il tente de router le circuit synthétisé en partant d'un nombre minimum de fils de routage et en incrémentant ce nombre à chaque échec de routage. Ainsi il permet d'explorer par une approche itérative le nombre de ressources de routages de chaque type. Avec cet outil il est aussi possible d'explorer manuellement plusieurs aspects comme la taille des LUT (Look Up Table) et leur nombre dans les clusters [Ahmed00] et la taille des éléments mémoire embarqués [Wilton99].

Madeo-Bet est un autre exemple d'outil générique de placement routage pour FPGA et architectures reconfigurables, il est développé à l'Université de Bretagne Occidentale [Lagadec00]. Bien qu'il utilise les mêmes algorithmes de placement routage que l'outil VPR (algorithme PathFinder basé sur l'algorithme de routage en labyrinthe) Madeo-Bet permet de cibler un ensemble plus large d'architectures. Effectivement, contrairement à VPR, Madeo-bet utilise un modèle fonctionnel pour définir l'architecture du composant ciblé. Ainsi chaque ressource (de routage ou logique) est modélisée par la fonction qu'elle réalise. De cette façon l'espace modélisable est bien plus large que celui défini par une modélisation physique. De plus l'approche fonctionnelle est indépendante de la technologie. La contre partie est évidemment une imprécision des estimations de performances. Mais celle-ci n'a cependant pas d'incidence lors d'une étude comparative des solutions architecturales si l'erreur est continue dans tout l'espace architectural. Madeo-bet permet aussi d'obtenir une description VHDL de l'architecture modélisée et un fichier de configuration (bitstream) si les informations sur la génération de ce fichier sont libres [LeBeux04].

3.2.2. Outils d'estimations des performances des FPGA

Une deuxième approche consiste à utiliser des outils d'estimations de performances ciblant des architectures reconfigurables. Naturellement il s'agit là d'une méthode d'exploration du type *estimer puis comparer*. Les estimations peuvent cibler un (exploration mono-objectif) ou plusieurs paramètres (exploration

multi-objectifs).

Une approche originale est proposée par l'Université de Californie du Sud [Choi02]. Cette approche propose d'estimer uniquement la consommation de puissance en utilisant un modèle haut niveau de la consommation de puissance pour un *domaine*. Celui-ci est défini par la réunion d'une famille d'architectures et d'un ensemble d'applications proches (utilisant les mêmes algorithmes ou primitives de calculs). L'estimation de la consommation de puissance est basée sur des fonctions d'estimation préétablies par l'utilisateur. Elles-mêmes utilisent des paramètres architecturaux et algorithmiques qui sont choisis et spécifiés par l'utilisateur comme des paramètres influant sur la consommation d'énergie. De ce fait, l'utilisateur doit avoir une connaissance détaillée du domaine pour identifier ces paramètres architecturaux et algorithmiques. Toutefois, la modélisation étant réduite à un domaine (architectures plus algorithmes) le nombre de paramètres est réduit ce qui facilite le développement des fonctions d'estimation de la consommation de puissance. Ce système permet d'obtenir des estimations rapides et précises, les fonctions utilisées étant développées via des simulations effectuées au préalable. L'exploration de l'espace de conception (réduit au domaine ici) se fait de façon manuelle en ajustant les paramètres architecturaux et algorithmiques. Cette étude présente des concepts intéressants comme la réduction de l'exploration à un domaine, ce qui se conçoit bien dans l'évolution des architectures reconfigurables vers des architectures spécifiques à un domaine d'applications. Malheureusement cette étude reste trop théorique et demande une grande expertise de la part de l'utilisateur.

Les travaux développés à l'Institut Technologique Fédéral de Suisse permettent une estimation à haut niveau de la surface et de la vitesse d'une application régulière (type flot de données) sur un FPGA cible [Enzler00]. Les estimateurs utilisent, comme dans [Bilavarn02], une pré-caractérisation d'opérateurs arithmétiques et logiques de bases sur le composant ciblé (qui doit donc être disponible physiquement). Une fois les estimations obtenues plusieurs paramètres algorithmiques peuvent êtres explorés comme le nombre de registres de pipeline dans le chemin de données, le nombre de réplication du bloc estimé (par exemple cœur de boucle dans le cas de déroulage de boucle) et le niveau de décomposition (ou granularité) des blocs estimés. Ainsi l'exploration est ici principalement orientée vers l'implémentation de l'application (ce qui correspond à l'architecture logique) et non pas vers l'architecture physique reconfigurable.

3.2.3. Outils d'exploration de l'espace de conception des architectures gros grain

Les travaux présentés précédemment visaient principalement les architectures de grain fin du type FPGA même ci pour certains il est possible d'imaginer une extension aux architectures de plus gros grain. Des travaux visent toutefois spécifiquement des architectures de gros grains, nous en présenterons deux.

L'architecture reconfigurable RAW (Reconfigurable Architecture Workstation) est une architecture gros grain sous forme de matrice de tuiles, chaque tuile

contenant un processeur RISC élémentaire et une mémoire SRAM pour les instructions et les données [Waingold97]. Pour cette architecture un flot d'exploration de l'espace de conception a été présenté dans [Moritz98]. Celui-ci propose, pour une application donnée, d'explorer un certain nombre de paramètres de l'architecture RAW tels que le nombre de tuiles, la puissance de calcul d'une tuile, la taille de la mémoire embarquée, la bande passante et la latence des communications (inter-tuile et entre RAW et la mémoire externe). L'application est décomposée en sous-problèmes qui sont caractérisés séparément par le nombre de ressources (tuiles et routage) qu'ils nécessitent pour être implémentés sur l'architecture. Des fonctions de coûts permettent d'estimer les performances (surface et délais) en fonction de la caractérisation des sous-problèmes. Cette approche cible donc uniquement l'architecture RAW pour laquelle les fonctions de coûts ont été déterminées ce qui délimite un espace de conception très faible.

De la même façon l'outil Xplorer développé à l'Université de Kaiserslautern en Allemagne [Nagelinger01] propose une exploration des caractéristiques de l'architecture reconfigurable gros grain KressArray développée dans la même université [Kress96]. Cet outil opère au niveau algorithmique et propose à l'utilisateur une aide interactive pour la définition d'une architecture KressArray en adéquation avec l'application développée. L'outil est effectivement semi-automatique puisqu'il fournit automatiquement des estimations et des suggestions d'évolution de l'architecture à l'utilisateur mais lui laisse le soin de faire évoluer l'architecture en suivant ou non les suggestions proposées. A l'issue de l'exploration, l'outil fournit une description en Verilog de l'architecture KressArray ainsi caractérisée pour effectuer des simulations. Cet outil est donc très dépendant de l'architecture. Pour proposer des suggestions d'exploration architecturale à l'utilisateur l'outil est basé sur un processus en logique floue qui est une approche très originale et efficace pour répondre au problème de l'exploration de l'espace de conception.

3.2.4. Etude comparative

Le tableau 1 résume les principales caractéristiques des travaux cités dans les trois paragraphes précédents. La dernière ligne de ce tableau donne les caractéristiques de la méthode d'exploration basée sur l'utilisation de l'outil d'aide à la conception des systèmes sur puce Design Trotter que nous présenterons dans le chapitre suivant, ainsi l'apport de nos travaux apparaît clairement dans ce tableau par rapport aux approches existantes. Dans les six travaux présentés nous pouvons voir que seul les travaux de l'E.T.H. permettent spécifiquement une exploration algorithmique (donc au niveau de l'implémentation de l'application). Nous pouvons voir aussi qu'il n'y a pas de travaux qui couvrent complètement l'espace de conception depuis les architectures de grain fin (FPGA) aux architectures les plus hétérogènes (mélangeant grain fin et gros grain). Cependant, parce qu'ils sont génériques, les outils VPR et Madeo-bet couvrent le plus large espace architecturale, les autres travaux sont trop dépendants de la technologie pour permettent l'exploration d'espaces architecturaux

larges.

Travaux (outils)	Cible	Spécification des applications	Spécification des architectures	Type d'exploration	Résultat de l'exploration
VPR Univ. Toronto [Betz97]	FPGA complexe	Netlist BLIF (ou EDIF)	Modèle physique	**Architecturale** Mono-objectif (routage) Manuelle et exhaustive	Informations de conception (taille des clusters, taille des éléments de base, routage)
MADEO Univ. Brest [Lagadec01]	FPGA complexe multi-grains	DFG	Modèle fonctionnel	**Architecturale** Multi-objectifs (routage, surface et chemin critique) Manuelle et exhaustive	Informations de conception (taille des clusters, taille et fonctions des éléments de base, routage)
Univ. Californie Sud [Choi02]	FPGA simple	Paramétrique	Paramétrique	**Architecturale** Mono-objectif (puissance) Manuelle et exhaustive	Une architecture parmi un ensemble
E.T.H. [Enzler00]	FPGA simple	DFG	Pré-caractérisation d'opérateurs de base	**Algorithmique** Multi-objectifs (surface et vitesse) Manuelle et exhaustive	Informations de conception (parallélisme et niveau de pipeline)
M.I.T. [Moritz98]	Architecture gros gain RAW	Paramétrique	Paramétrique	**Architecturale** Mono-objectif (temps d'exécution) Automatique et heuristique	Une configuration optimale de l'architecture Raw
Xplorer Univ. Kaiserslautern [Nageldinger01]	Architecture gros gain KressArray	ALE-X	Paramétrique et modèle structurel	**Architecturale** Multi-objectifs (puissance et routage) Semi-automatique et heuristique	Une configuration optimale de l'architecture KressArray
Design Trotter Univ. Bretagne Sud [Bossuet04]	Architecture hétérogène	HCDFG	Modèle fonctionnel hiérarchique	**Architecturale et Algorithmique** Mono-objectif (puissance) Semi-automatique et heuristique	Informations de conception (taille des clusters, taille et fonctions des éléments de base, distribution des communications et taux d'utilisation des ressources)

Tableau 1. *Comparatif de travaux pouvant être mis en œuvre dans un flot d'exploration de l'espace de conception des architectures reconfigurables.*

De façon générale, toutes ces études présupposent que le concepteur qui réalise l'exploration ait une bonne connaissance du domaine des architectures reconfigurables en particulier pour les études basées sur des pré-caractérisations ou des modèles paramétriques. Il semble en effet très compliqué et même irréaliste de chercher à définir une méthode complètement automatique d'exploration (architecturale ou algorithmique) qui ne s'appuierait pas sur les connaissances techniques et l'expérience du concepteur.

4. Méthode générique d'exploration de l'espace de conception des architectures reconfigurables basée sur l'environnement Design Trotter

Le chapitre précédent a montré que, malgré des travaux de grande qualité, il manque dans le domaine des architectures reconfigurables une méthode générique d'exploration de l'espace de conception. Le terme générique s'inscrit principalement dans le choix architectural (architectures grain fin, gros grain et hétérogène) ainsi que dans le choix du domaine d'applications (traitement des images, cryptographie, télécommunication …). L'exploration doit porter sur l'aspect architectural de la cible mais aussi sur l'implémentation de l'application, c'est à dire qu'elle vise simultanément l'architecture physique et l'architecture logique. En s'appuyant sur l'outil Design Trotter nous proposons une méthode d'exploration qui répond à ces exigences en tirant parti d'estimations effectuées à haut niveau d'abstraction et intervenant très tôt (dés les phases de spécifications) dans le flot de développement des applications. Les paragraphes suivants vont présenter l'environnement Design Trotter et les principes de la méthode d'exploration, enfin quelques résultats valideront la démarche.

4.1. *Un environnement d'aide à la conception des architectures reconfigurables : Design Trotter*

La figure 2 donne une vision schématique de l'environnement Design Trotter. Celui-ci se compose de trois parties distinctes. Notons en premier lieu que le point d'entrée au niveau application de l'environnement est une spécification en langage C qui est transformée à l'aide d'un analyseur syntaxique en graphe hiérarchique de contrôle et flot de données (ou HCDFG : *Hierarchical Control Data Flow Graph*).

La première partie de l'environnement Design Trotter est l'outil automatique d'estimation système [LeMoullec03]. Celui-ci permet d'estimer les réalisations possibles de l'application spécifiée en proposant des solutions de réalisation appelées courbes de compromis (représentées en haut à droite de la figure 2). Il y a une courbe de compromis par ressource (de traitement et de mémorisation) nécessaire à la réalisation de l'application. Chaque point d'une courbe de compromis donne le résultat d'estimation du nombre de ressources à mettre en œuvre pour un nombre de cycles d'horloge (en considérant une architecture abstraite [LeMoullec03]). On peut ainsi, en positionnant une contrainte de temps, connaître une estimation des besoins en termes de ressources matérielles pour la réalisation de l'application. Si la contrainte de temps est forte (c'est à dire que l'application doit être exécutée dans un temps court) alors on sélectionne des réalisations parallèles comptant de nombreuses ressources, nécessitant ainsi une réalisation matérielle (à fort parallélisme) pour l'application.

Lorsqu'une solution de réalisation est sélectionnée, le concepteur peut estimer les performances de l'application sous contrainte de temps sur un composant FPGA commercial avec la deuxième partie de l'environnement qui est la projection physique [Bilavarn03]. Il faut toutefois s'assurer que la librairie contienne les informations nécessaires concernant le circuit visé, si ce n'est pas le cas il faut pré-caractériser le circuit. La projection physique sur FPGA donne une estimation précise des performances de l'application (15 % en moyenne ce qui est bon par rapport à d'autres outils et étant donné le niveau d'abstraction de la spécification initiale [Bilavarn03]). Cependant cet outil est très dépendant de la technologie et ne vise que les FPGA commerciaux. Pour étendre les possibilités de l'environnement, un troisième outil automatique a été développé pour viser plus largement les architectures reconfigurables, c'est la projection architecturale.

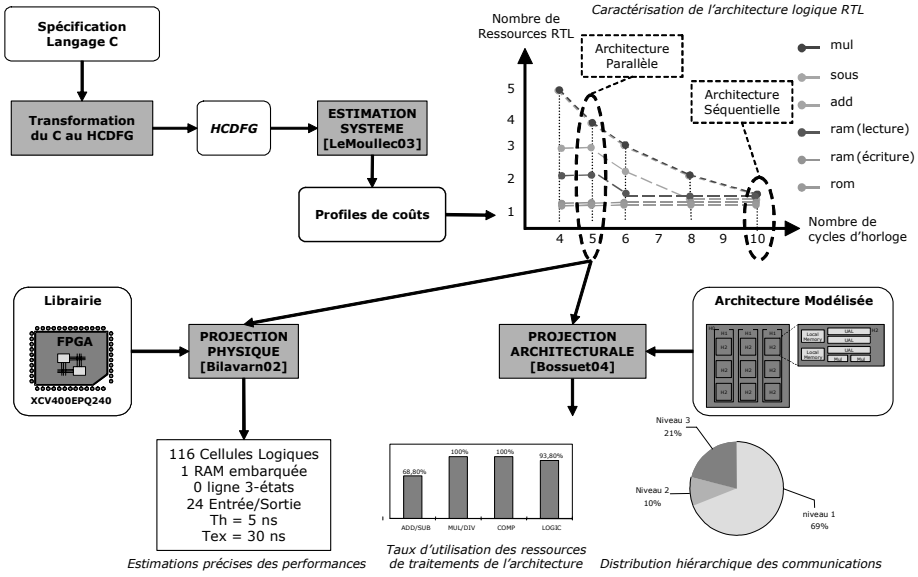

Figure 2. *L'environnement Design Trotter avec ses trois parties : l'estimation système, la projection physique sur FPGA et la projection architecturale.*

La projection architecturale [Bossuet04] utilise les résultats de l'estimation système et une modélisation hiérarchique fonctionnelle des architectures (nous présenterons cette modélisation dans la suite). Les estimations obtenues en résultats sont l'estimation de la distribution hiérarchique des communications dans l'architecture et l'estimation du taux d'utilisation des ressources de traitement de l'architecture. Ces paramètres permettent effectivement d'évaluer l'efficacité en consommation de puissance d'une architecture. Ces estimations sont utilisées par l'utilisateur lors de l'exploration architecturale pour modifier en conséquence

l'architecture modélisée et converger vers la définition d'une architecture efficace.

En résumé l'estimation système permet donc une exploration de l'architecture logique via une estimation des réalisations possibles de l'application (de la plus parallèle à la plus séquentielle) et la projection architecturale permet une exploration de l'architecture physique. L'environnement Design Trotter permet donc, par une approche originale, une exploration très large de l'espace de conception.

4.2. *Informations en entrée et en sortie de la projection architecturale*

La méthode d'exploration architecturale que nous proposons s'appuie sur les informations de conception fournies par la projection architecturale dans Design Trotter. Afin d'appréhender le concept d'exploration que nous avons développé, il est nécessaire de connaître les spécifications en entrée de la projection architecturale et les informations (estimations) produites en sortie.

4.2.1. Spécification des applications

Comme toutes les parties de l'environnement Design Trotter, la projection architecturale reçoit en entrée la description HCDFG de l'application. Cependant celle-ci est très riche d'informations puisqu'elle décrit le flot de donnée et de contrôle complet de l'application ce qui n'est pas nécessaire à la projection architecturale. Effectivement cette dernière vise à estimer rapidement la distribution des communications dans l'architecture lors de l'exécution de l'application. Aussi en tirant profit des résultats de l'estimation système le HCDFG est transformé pour mettre en avant les communications entre les ressources (arithmétique, logique ou mémoire) qui sont nécessaires à l'application pour un ordonnancement choisi. Le graphe ainsi obtenu s'appelle l'ACG (*Average Communication Graph*). Ce graphe ne permet plus de décrire le flot de données et de contrôle inhérent à l'application mais bien les communications qui en découlent. La figure 3 présente la transformation d'un DFG simple (avec uniquement des nœuds traitement qui représentent des opérations arithmétiques) en ACG. Le DFG initial a dans cet exemple sept nœuds, chaque nœud correspond à une opération parmi les opérations *a*, *b*, *c* ou *d*. Lors de la création de l'ACG correspondant, seuls quatre nœuds sont créés pour représenter les quatre types d'opérateurs qui interviennent lors de la réalisation de l'application. Des arcs entre ces nœuds sont créés si il existe dans le DFG initial un échange de données (peut importe le sens) entre les opérations correspondantes aux nœuds. Par exemple dans le DFG initial le nœud *a* en haut du graphe est relié à deux nœuds *b*, donc dans l'ACG un arc est créé entre le nœud *a* et le nœud *b*. La différence fondamentale avec le DFG est que cet arc est pondéré, il prend une valeur qui dépend du nombre d'échanges de données entre les nœuds *a* et les nœuds *b* dans le DFG initial. Cette pondération dépend aussi du nombre d'opérateurs de type *a* et *b* nécessaires à la réalisation de l'application pour l'ordonnancement choisi. Par exemple ici un opérateur *a* et deux opérateurs *b*, ce nombre apparaît entre parenthèses près des nœuds sur l'ACG.

La valeur des arcs prend en compte le nombre de communications entre les opérateurs et le nombre de ces opérateurs. La valeur $Varc_{n1n2}$ de l'arc entre deux nœuds n_1 et n_2 se calcule par la formule suivante :

$$Varc_{n_1n_2} = \frac{NC_{n_1n_2}}{N_{Op_1} + N_{Op_2}}$$

avec NC_{n1n2} le nombre de communications entre les nœuds du type *n1* et les nœuds du type *n2* dans le DFG et avec N_{Op1} et N_{Op2} respectivement le nombre d'opérateurs nécessaires pour réaliser les opérations de type *n1* et *n2* conformément aux résultats de l'estimation système. Dans le cas du nœud *a* et du nœud *b* de l'ACG la valeur de l'arc vaut 1 puisque le nombre de communications (3) est égale au nombre d'opérateurs (1 *a* plus 2 *b*). Cette valeur a pour but de permettre aux algorithmes qui vont par la suite parcourir le graphe, de sélectionner les arcs les plus critiques (avec la plus grande valeur).

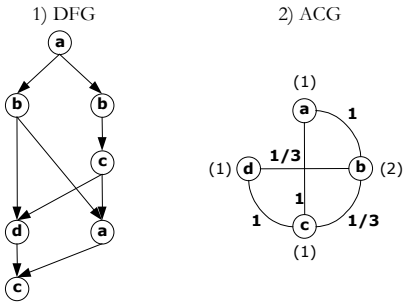

Figure 3. *1) le DFG initial et 2) l'ACG qui en résulte après le choix d'un ordonnancement.*

Dans l'exemple présenté nous n'avons pas considéré les nœuds mémoires qui apparaissent dans la spécification d'un DFG. Ces nœuds représentent des données scalaires ou multidimensionnelles. L'estimation système considère que les données scalaires sont réalisées à l'aide de registres et donc seules les données multidimensionnelles sont utilisées pour estimer la taille des mémoires physiques nécessaires à l'application. Les données scalaires sont donc effacées du graphe. Ce n'est pas le cas des données multidimensionnelles qui sont réalisées dans des mémoires RAM ou ROM. Pour réduire le graphe nous suivons la même stratégie pour les ressources de mémorisation que pour les ressources de traitement. Il existe toutefois des différences ; nous ne considérons pas l'ordonnancement des mémoires et le partage des données entre mémoires physiques. Effectivement le niveau actuel des outils de synthèse pour architecture reconfigurable ne gère pas le partage des données entre les mémoires distribuées. Les nœuds mémoires (toujours associés à un ordonnancement égal à 1) que nous conservons dans l'ACG représentent donc des données multidimensionnelles. La figure 4 donne un exemple de transformation

d'une partie d'un DFG qui comporte des nœuds mémoires représentant une donnée multidimensionnelle A[ ] et une donnée scalaire B. Ce DFG partiel comporte trois nœuds traitements qui réalisent deux opérations, addition et multiplication. Une première étape de la transformation du DFG (figure 4-1) en ACG (figure 4-3) et l'élimination des nœuds mémoires représentant la donnée scalaire B (figure 4-2). A partir du nouveau DFG, l'ACG est créé comme indiqué précédemment.
Nous savons maintenant comment dériver l'ACG depuis un simple DFG, mais la spécification d'entrée est plus riche puisqu'il s'agit d'un HCDFG. Il faut donc déterminer l'ACG de ce type de graphe, c'est à dire être capable, en particulier, de prendre en compte les contrôles, les branchements conditionnels et les boucles. Pour ce qui est des nœuds de contrôle (=, !=,< ou > par exemple) qui interviennent dans les boucles et les branchements conditionnels, ils sont considérés comme des traitements (car ils sont ordonnancés avec des ressources de types comparateurs). Pour les branchements conditionnels (*if* ou *case*) toutes les branches possibles sont prises en compte. Cependant elles vont plus ou moins intervenir dans le nombre total de communication en fonction de la probabilité de passage dans ces branchements (calculée par *profiling*). Le nombre de communications initiales entre les nœuds d'une branche est multiplié par la probabilité de passage. Dans le même esprit, le nombre de passages dans un cœur de boucle est connu à l'avance puisque les boucles sont toutes déterministes. Les communications à l'intérieur du cœur sont réalisées autant de fois que le cœur est réalisé. Il faut donc multiplier le nombre de communications dans la boucle par le nombre de fois que le cœur de boucle est exécuté.

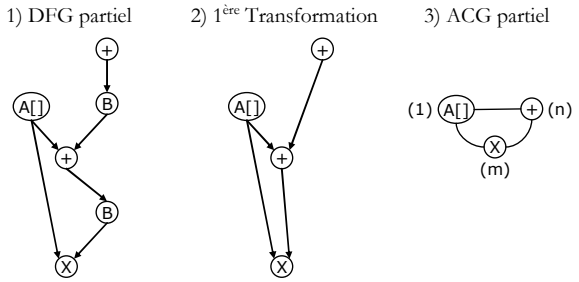

Figure 4. *Transformation d'un DFG comportant des nœuds mémoires.*

4.2.2. Modélisation des architectures

Concernant la spécification des architectures nous avons défini une modélisation fonctionnelle qui répond aux besoins indispensables suivants :

- Décrire un large espace architectural contenant des architectures grain fin, gros grain et hétérogènes.
- Décrire la localité physique des ressources de traitement et de mémorisation embarquées dans l'architecture.

- Evoluer facilement et rapidement pour prendre en compte les nouvelles possibilités et caractéristiques des architectures reconfigurables.
- Permettre à l'utilisateur de faire varier très facilement les caractéristiques de l'architecture modélisée afin de ne pas perdre de temps lors du processus d'exploration.

Pour cela nous avons développé un modèle hiérarchique et fonctionnel. Ce modèle est composé d'éléments hiérarchiques qui sont des contenants et d'éléments fonctionnels qui sont contenus uniquement au niveau le plus bas de la hiérarchie. La figure 5 donne un exemple d'architecture modélisée avec trois niveaux de hiérarchie. Notre modèle utilise deux hypothèses très importantes qui par la suite limitent l'exploration architecturale pour le concepteur afin de respecter des considérations technologiques ;

- Les coûts de communication dans un élément hiérarchique sont homogènes. Le concepteur doit toujours garantir cette hypothèse et donc ne pas définir d'architecture avec des éléments hiérarchiques de taille trop importante.
- Les communications dans un élément hiérarchique sont moins pénalisantes que les communications entre éléments hiérarchiques. Ainsi les éléments hiérarchiques décrivent la localité physique des éléments (hiérarchiques ou fonctionnels) qu'ils contiennent.

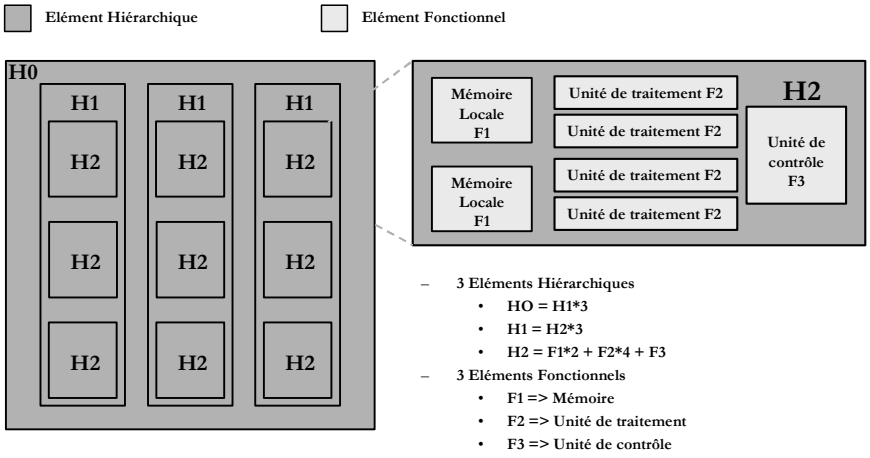

Figure 5. *Exemple de modélisation hiérarchique d'une architecture gros grain hiérarchique.*

L'exemple illustré à la figure 5 est une modélisation d'une architecture hiérarchique de gros grain. Cette architecture comprend trois niveaux de hiérarchie. Au niveau le plus haut l'élément hiérarchique *H0* contient trois éléments hiérarchiques *H1*. Les communications entre les éléments *H1* dégraderont le plus les performances en terme de consommation de puissance de l'application sur

l'architecture. Lors de l'exploration architecturale le concepteur veillera à diminuer le nombre de communications à ce niveau. Au niveau intermédiaire, chaque élément *H1* contient trois éléments *H2*. Ces derniers contiennent sept éléments fonctionnels ; deux éléments *F1* (mémoire), quatre éléments *F2* (unité de traitement) et un élément *F3* (unité de contrôle). Les communications entre des éléments fonctionnels appartenant à un même élément hiérarchique *H2* sont les moins pénalisantes pour les performances en consommation de puissance lors de l'exécution de l'application sur l'architecture.

Durant la phase d'exploration architecturale, le concepteur change (manuellement) les paramètres de l'architecture comme le nombre d'éléments fonctionnels au niveau le plus bas de la hiérarchie (dans les éléments H2 pour la figure 5), et/ou la taille des éléments hiérarchiques (par exemple le nombre d'éléments H2 dans chaque élément H1). De plus il intervient aussi sur les éléments fonctionnels, par exemple la taille des éléments mémoire ou les opérations supportées par les éléments fonctionnels de traitement. Le chapitre suivant explicitera ces interventions lors du flot d'exploration architecturale.

4.2.3. Informations en sortie de la projection architecturale

Le but de la projection architecturale et "d'émuler" les outils de synthèse donc de projeter l'ACG sur l'architecture. C'est à dire que chaque ressource (opération, mémoire ou contrôle) contenue dans un nœud de l'ACG doit être associée à un élément fonctionnel de l'architecture (opérateur, LUT, RAM, ROM, comparateur, etc.). La stratégie suivie est de grouper au niveau hiérarchique le plus bas les éléments qui communiquent le plus lors de l'exécution de l'application.

Une première approche consiste à chercher une solution optimale au problème de regroupement hiérarchique. Cependant nos estimateurs fonctionnant à haut niveau, il n'est pas certain qu'une solution optimale à ce niveau le soit finalement étant donné que nous ne prenons pas en compte l'impact des outils de plus bas niveau. Comme nous cherchons à comparer entre elles des solutions architecturales, il est plus pertinent de borner les solutions dans des intervalles d'efficacité. C'est pourquoi lors du développement de la projection architecturale nous avons développé trois algorithmes exprimant le pire cas, le meilleur cas et l'intermédiaire (ces trois algorithmes sont décrits dans [Bossuet04]). L'estimation de la distribution hiérarchique des communications est le principal résultat de la projection architecturale, c'est elle (sous forme d'intervalle min/moyen/max) qui va guider le processus d'exploration architecturale. On estime donc le nombre de communications à chaque niveau de la hiérarchie, le résultat est donné en pourcentage du nombre total de communications.

Une information supplémentaire vient compléter les informations utilisables par le concepteur lors de la phase d'exploration. Il s'agit du taux d'utilisation des ressources fonctionnelles de l'architecture. Effectivement ce taux d'utilisation va directement intervenir sur l'efficacité en consommation de puissance de l'architecture [Rouxel02], en particulier pour les ressources de gros grain.

4.3. Méthode d'exploration architecturale

La méthode d'exploration architecturale vise à définir une architecture efficace du point de vue de la consommation de puissance pour une application (et pour un domaine d'applications par extension). Cette méthode exige un flot complexe que nous schématisons à la figure 6.

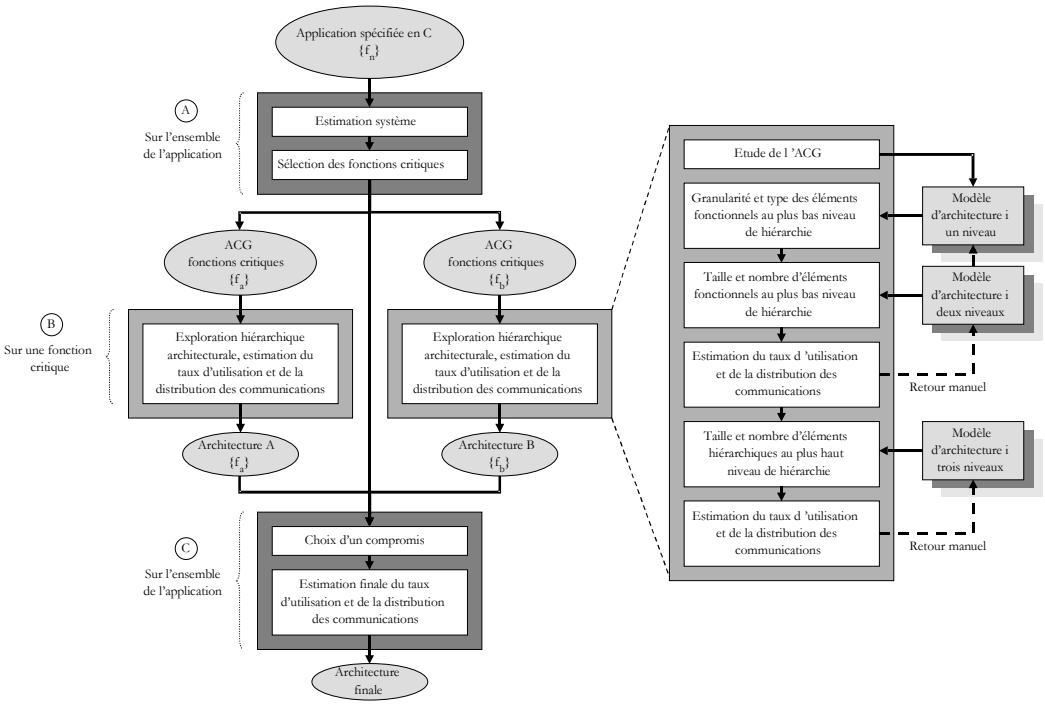

Figure 6. *Flot d'exploration hiérarchique pour les fonctions critiques et pour l'application complète.*

La première étape de ce flot (étape A sur la figure 6) correspond à la mise en œuvre de l'estimation système. L'application, spécifiée en langage C, est le plus souvent décomposée en *n* fonctions $\{f_n\}$ et dans ce cas un HCDFG est généré pour chaque fonction. Nos expériences sur l'exploration architecturale nous ont rapidement montré que certaines fonctions de l'application, le plus souvent une ou deux, avaient un impact très fort sur les performances finales de l'application [Bossuet04]. Aussi, il nous a paru plus judicieux et plus efficace de parcourir le flot d'exploration uniquement pour ces fonctions que nous appelons fonctions critiques. Effectivement, si le flot est parcouru pour les *n* fonctions de l'application il en résultera *n* architectures, hors si l'on souhaite définir une architecture homogène structurellement il est extrêmement compliqué, voir intractable, de trouver un compromis parmi *n* dès que *n* atteint plusieurs unités. Afin de déterminer clairement

les fonctions critiques et à partir de nos résultats expérimentaux nous avons établi trois critères de criticité :

- **Le degré de parallélisme d'exécution de la fonction** va nous permettre de vérifier que la solution d'ordonnancement retenue pour chaque fonction est bien adaptée à une implémentation matérielle massivement parallèle. Si il n'est pas possible de réaliser la fonction de façon parallèle elle sera réalisée séquentiellement ce qui peut entrainer un ralentissement de l'application. La fonction peut donc être critique d'un point de vue temporel.
- **La localité potentielle des communications dans l'architecture** représente le nombre moyen de communications par ressource de traitement. C'est l'ACG généré d'après le HCDFG initial qui nous permet d'obtenir rapidement le nombre de communications. Un problème se pose si le nombre de ressources de traitement est faible pour un grand nombre de communications. Il faut prévoir alors des ressources de routage particulièrement adaptées ainsi que des ports d'entrées/sorties des éléments hiérarchiques assez larges pour le flux de communications. Il ne s'agit ici que d'une étude rapide nous permettant de discriminer les fonctions qui n'ont pas un nombre important de communications dans leur réalisation.
- **La congestion temporelle potentielle des ressources de routage** représente le nombre moyen de communications par cycle. Le problème ici vise l'occupation temporelle des ressources de routage. Effectivement il peut y avoir des problèmes si le nombre de communications est grand pour un nombre de cycles relativement faible. Le problème se pose particulièrement pour les grands nombres de communications. Il faut alors prévoir une répartition des communications telle que les congestions soient évitées.

Une fois les fonctions critiques sélectionnées le flot d'exploration (partie B de la figure 6) est parcouru pour chacune de ces fonctions. La première étape vise à établir la granularité et le type des éléments fonctionnels (opérateurs et mémoires) contenu dans les clusters (ou éléments hiérarchiques) de niveau le plus bas de hiérarchie (niveau 1). Cette première étape consiste à étudier la répartition des communications dans le graphe ACG de la fonction critique et elle permet de modéliser le premier niveau de hiérarchie de l'architecture. Les deux étapes suivantes vont s'appuyer sur les résultats de la projection architecturale (estimation du taux d'utilisation des éléments fonctionnels et distribution hiérarchique des communications) pour déterminer le nombre et la taille des éléments fonctionnels. De cette façon les éléments hiérarchiques au niveau supérieur sont modélisables. Rappelons les objectifs du concepteur lors de l'exploration :

- Obtenir une distribution des communications dans l'architecture telle que le plus grand nombre possible de communications soit réalisé au niveau bas de hiérarchie c'est à dire dans les éléments hiérarchiques qui contiennent les éléments fonctionnels (par exemple les éléments H2 de la figure 5).

- Ne pas chercher à modéliser des éléments hiérarchiques de taille trop importante car il faut être capable d'assurer que les communications dans ces éléments sont à coûts constants.
- Enfin, pour une plus grande efficacité en consommation de puissance, obtenir un taux d'utilisation des éléments fonctionnels le plus grand possible en particulier pour les ressources de gros grain (ALU, additionneur).

Lors de cette phase d'exploration le concepteur change manuellement les paramètres de l'architecture modélisée pour améliorer les estimations, c'est donc une démarche itérative et empirique. Enfin une dernière phase d'exploration, qui ne remet pas en question l'exploration au niveau hiérarchique inférieur, vise à déterminer le nombre d'éléments hiérarchiques dans les niveaux supérieurs de hiérarchie (partie C sur le flot de la figure 6).

Lorsque la phase d'exploration architecturale est terminée pour l'ensemble des fonctions critiques, il est nécessaire de faire converger les architectures définies pour chaque fonction critique. Dans une première approche nous avons proposé de définir l'architecture "compromis" avec les éléments hiérarchiques de taille maximum entre les différentes architectures correspondant aux fonctions critiques, mais cette approche mérite d'être raffinée [Bossuet04]. Le taux d'utilisation des éléments fonctionnels et la distribution hiérarchique des communications dans l'architecture "compromis" sont alors estimés pour l'ensemble $\{f_n\}$ des fonctions de l'application.

4.4. *Exemple d'utilisation : codeur MPEG-2, codeur AES*

Afin d'illustrer nos propos, notre flot d'exploration est utilisé pour deux applications : une application de traitement des images : le codeur MPEG-2 et une application de cryptographie : le codeur AES (sans générateur de clef). Ces deux applications n'ont pas la même complexité, le codeur AES est plus simple et sa spécification ne contient qu'une fonction. Le codeur MPEG-2 est spécifié avec neuf fonctions en langage C, dont deux sont critiques pour les performances en consommation de puissance suivant les critères présentés au chapitre précédent.

L'étude de l'ACG pour les deux applications ne donne pas le même résultat. Effectivement en additionnant les communications sur les graphes (en utilisant la valeur des arcs de l'ACG) nous trouvons que plus de 90% des communications pour le codeur MPEG-2 sont relatives à des échanges entre ressources de traitement de gros grain et entre ces mêmes ressources et les ressources de mémorisation. Par contre dans le cas du codeur AES les communications sont réparties entre les ressources de traitement de gros grain, de grain fin et les ressources de mémorisation. Les étapes du flot d'exploration nous permettent de déterminer pour les deux applications, le nombre d'opérateurs de grain fin et de gros grain ainsi que la taille (en nombre de mot) des mémoires dans les clusters de bas niveau.

Les figures 7 et 8 donnent une représentation schématique des architectures hiérarchiques définies pour les deux applications. Nous voyons sur la figure 7 que

l'architecture définie pour le codeur MPEG-2 à deux clusters différents au plus bas niveau de hiérarchie : un cluster avec des opérateurs de gros grain (cluster 1) et un cluster avec des opérateurs de grain fin (cluster 2). Ce qui n'est pas le cas de l'architecture définie pour AES et schématisée à la figure 8 puisqu'il n'y a qu'un seul type de cluster au niveau le plus bas (cluster 3) dans lequel sont embarqués les opérateurs de grain fin et de gros grain. Nous voyons que pour deux domaines d'application différents nous pouvons converger vers des architectures différentes. Les estimations obtenues pour ces deux applications sont données au tableau 2. La dernière ligne de ce tableau donne les estimations obtenues en termes de distribution des communications pour le codeur AES avec l'architecture définie pour MPEG-2.

Application	Taux d'utilisation				Distribution des communications		
	ADD/SUB	MUL	COMP	LUT	Haut Niveau	Niveau Moyen	Bas Niveau
MPEG-2	67,0 %	70,0 %	13,0 %	2.0 %	29 %	8%	63 %
AES	63,8 %	100 %	100 %	93.8 %	21 %	10 %	69 %
AES*	-	-	-	-	36 %	14 %	50 %

Tableau 2. *Résultats d'estimations obtenus pour les deux applications.*

Les distributions hiérarchiques des communications obtenues par estimation pour les deux applications sont satisfaisantes puisque dans les deux cas la part des communications au plus bas niveau de hiérarchie est supérieure à 60 % du nombre total de communications. La dernière ligne du tableau 1 nous donne l'estimation de la distribution pour le codeur AES pour l'architecture définie pour le codeur MPEG-2, nous pouvons voir que les résultats estimés sont fortement dégradés puisqu'il y a une diminution de 19 % des communications au niveau bas et une augmentation de 15 % des communications au niveau haut. Nous montrons par cette expérience que le concepteur peut définir des architectures efficaces pour un domaine d'application et définir une architecture différente pour un autre domaine d'application.

Cependant il ne s'agit ici qu'une première étape dans la définition d'une architecture, il est nécessaire de continuer l'exploration architecturale en utilisant cette fois un outil génériques de plus bas niveau comme Madeo-Bet [Lagadec00]. Alors une modélisation physique prenant en compte le routage permet un raffinement des estimations et une validation possible sur des architectures réelles (ou émulation sur FPGA).

5. Conclusion

Nous avons présenté dans cet article l'espace de conception des architectures

reconfigurables ainsi que les techniques et outils développés aujourd'hui pour parcourir ce large espace. Dans cette problématique nous proposons une méthode d'exploration de l'espace de conception ciblant des architectures reconfigurables. L'exploration se fait sur la réalisation de l'application et sur les caractéristiques de l'architecture physique. Pour cela notre méthode utilise deux outils automatiques de l'environnement d'aide à la conception des systèmes sur puce Design Trotter développé au sein du laboratoire LESTER.

Grâce à des applications du traitement des images et de la cryptographie nous avons montré qu'un concepteur pouvait converger vers la définition d'architectures physiques efficaces en effectuant certains compromis. De plus le concepteur peut définir des architectures efficaces pour des domaines d'applications. Nous préconisons donc une spécialisation des architectures reconfigurables aux domaines d'applications ce qui semble être par ailleurs une tendance actuelle.

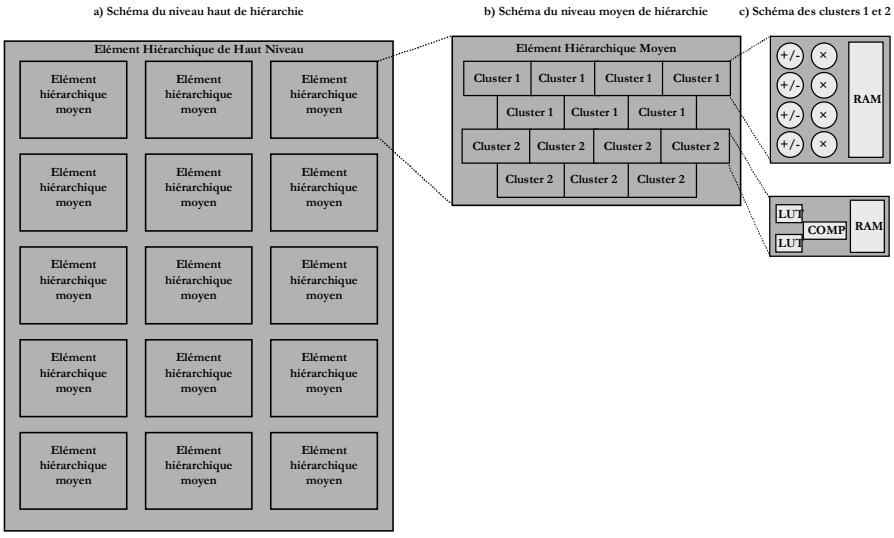

Figure 7. *Représentation schématique de l'architecture hiérarchique définie pour le codeur MPEG-2.*

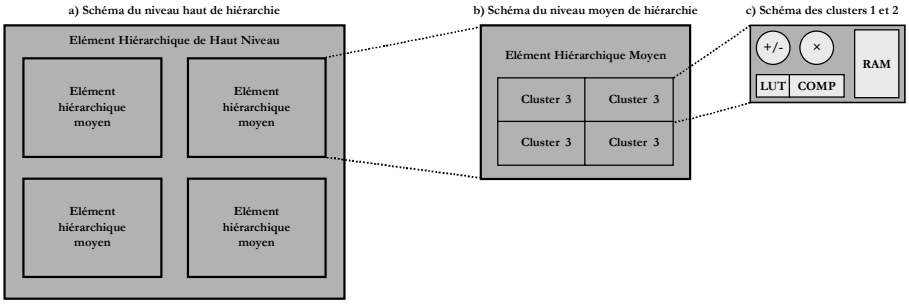

Figure 8. *Représentation schématique de l'architecture hiérarchique définie pour le codeur AES.*

Références

[Ahmed00] E. Ahmed and J. Rose. The Effect of LUT and Cluster Size on Depp-Submicron FPGA Performance and Density. *In International ACM Symposium on Field Programmable Gate Arrays, FPGA 00*,. February 2000, pp.3-12.

[Betz97] V. Betz, J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *In International Workshop on Field Programmable Logic and Application, FPL 97*, 1997.

[Bilarvarn02] S. Bilavarn. Exploration Architectural au Niveau Comportemental - Application aux FPGA. PhD Thesis, Université de Bretagne Sud, Lorient, 2002.

[Bossuet04] L. Bossuet. Exploration de l'Espace de Conception des Architectures Reconfigurables. PhD Thesis, Université de Bretagne Sud, Lorient, Septembre 2004.

[Bossuet03] L. Bossuet, G. Gogniat, J.L. Philippe. Communication Costs Driven Design Space Exploration for Reconfigurable Architectures. *In the 13th International Conference Field Programmable Logic and Applications, FPL'03*, Lisbon, Portugal, September 1-3, 2003.

[Choi02] S. Choi, J.W. Jang, S. Mohanty, V. K. Prasanna. Domain-Specific Modeling for Rapid System-Level Energy Estimation of Reconfigurable Architectures. *In Proceedings of International Conference of Engineering of Reconfigurable Systems and Algorithms, ERSA 02*, June 2002, Las Vegas, Nevada, USA.

[David03] R. David. Architecture reconfigurable dynamiquement pour applications mobiles. Ph.D. thesis, Univesité de Rennes 1, juillet 2003.

[Delahaye04] J.P. Delahaye, G. Gogniat, C. Roland, P. Bomel. Software Radio and Dynamic Reconfiguration on a DSP/FPGA platform. *For Publication in Special Issue of Software Radio, Frequenz, Journal of Telecommunications, 2004*.

[Enzler00] R. Enzler, T. Jeger, D. Cottet, G. Tröster. High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs. *In Field-Programmable Logic and Applications Forum on Design Language*, Villach, Austria, August 28 - 30, 2000.

[Gries03] M. Gries. Methods for Evaluating Covering the Design Space during Early Design Development. Technical Memorandum MO3/32, Electronics Research Laboratory, University of California at Berkeley, August 2003.

[Hartenstein01] R. Hartenstein. A Decade of Reconfigurable Computing : a Visionary Retrospective. *In Design Automation and Test in Europe, DATE'01*, Munich, Germany, 13-16 March, 2001.

[Kress96] R. Kress. A Fast Reconfigurable ALU for Xputers. PhD Thesis,University of Kaiserslautern, Germany, 1996.

[Lagadec00] L. Lagadec. Abstraction, modélisation et outils de CAO pour les circuits intégrés reconfigurables. PhD Thesis, Université de Bretagne Occidentale, Brest, 2000.

[LeBeux04] S. Le Beux. Extention de Madeo-Bet pour la génération de desciption architecturale en VDHL. Master Thesis, Université de Bretagne Occidentale, Brest, 2004.

[LeMoullec03] Y. Le Moullec. Aide à la conception des systèmes sur puce hétérogène par l'exploration paramétrable des solutions au niveau système. PhD Thesis, Université de Bretagne Sud, Lorient, April 2003.

[Nageldinger01] U. Nadelginder. Coarse-Grained Reconfigurable Architecture Design Space Architecture Exploration. Ph.D. Thesis, University of Kaiserslautern, Germany, June 2001.

[Pimentel01] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, Ed. F. Deprettere. Exploring Embedded-Systems Architectures with Artemis. *In IEEE Computer*, pp 57-63, vol 3 (n°11), November 2001.

[Rouxel02] S. Rouxel. Caractérisation de l'impact du routage sur les performances (vitesse et consommation de puissance) d'un FPGA. Matser Thesis, université de Bretagne Sud, Lorient, Septembre 2003.

[Schaumont01] P. Schaumont, I. Verbauwehe, K. Keutzer, M. Sarrafzadeh. A Quick Safari Through the Reconfigurable Jungle. *In 38th DAC*, Las Vegas, Nevada, USA, June 18-21, 2001.

[Ttrimberger97] S. Trimberger, D. Carberry, A. Johnson and J. wong. A Time-Multiplexed FPGA. *In IEEE Symposium on FPGAs for Custom Computing Machines , FCCM 97, pp 22-28*, 1997.

[Waingold97] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Franck, P. Finch. Baring it all to Software : The Raw Machine. *IEEE Computer*, pages 86-93, September 1997.

[Wilton99] S. J.E. Wilton, J. Rose, Z. G. Vranesic. The Memory/Logic Interface in FPGA's With Large Embedded Memory Arrays. *In IEEE Transactions on VLSI Systems, Vol. 7, No.1, March 1999*.

[Xilinx04a] Xilinx. Virtex-II Pro Platform FPGAs. Technical Document, February 2004.

[Zhang99] H. Zhang, M. Wan, V. George, J. Rabaey. Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs. In IEEE Computer Society Workshop on VLSI, April 1999.