Exploration de l'Espace de Conception des Architectures Reconfigurables

Lilian Bossuet

N°d'ordre : 41

THESE

présentée devant

L'UNIVERSITE DE BRETAGNE SUD

pour obtenir le grade de

DOCTEUR DE

L'UNIVERSITE DE BRETAGNE SUD

mention

ELECTRONIQUE ET INFORMATIQUE INDUSTRIELLE

Ecole Doctorale Pluridisciplinaire

Composante Universitaire :

UFR SCIENCES

ET

SCIENCES POUR L'INGENIEUR

par

M. Lilian BOSSUET

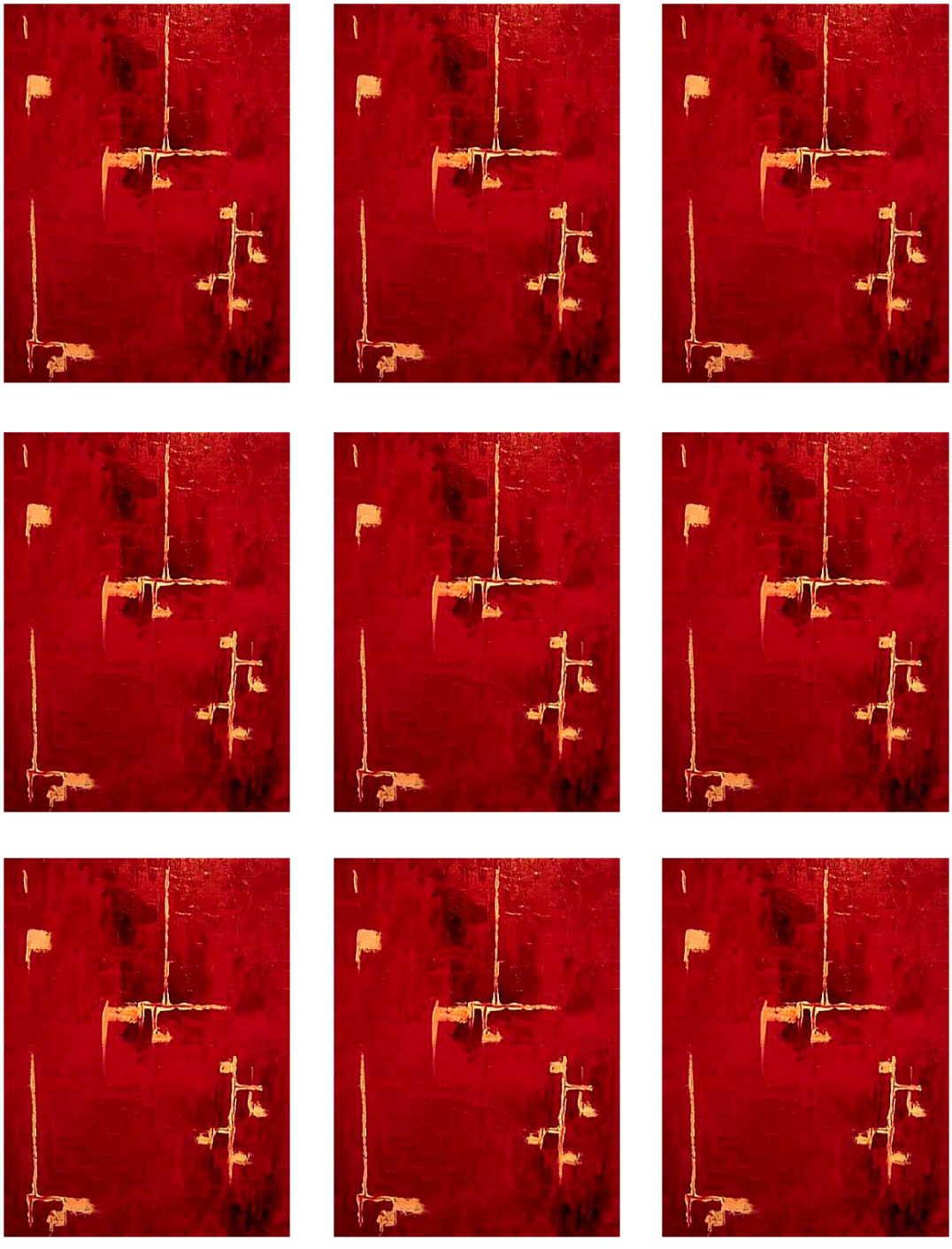
Exploration de l'Espace de Conception des Architectures Reconfigurables

soutenue le 10 Septembre 2004 devant la commission d'examen composée de :

M. J. LIENARD	Professeur, LIS, ENSIEG, Grenoble	Président
M. D.DEMIGNY	Professeur, LASTI, IUT Lannion	Rapporteur
M. L. TORRES	Maître de Conférences, HDR , LIRMM, Montpellier	Rapporteur
M. J-L. PHILIPPE	Professeur, LESTER, UBS, Lorient	Directeur de Thèse
M. B. POTTIER	Maître de Conférences, A&S, UBO, Brest	Examinateur
M. G. GOGNIAT	Maître de Conférences, LESTER, UBS, Lorient	Examinateur
M. J. CAMBONIE	Ingénieur, St Microelectronics, Grenoble	Invité

Lilian Bossuet

Exploration de l'Espace de Conception des Architectures Reconfigurables



Couverture ; peinture au couteau, avec l'amicale autorisation de Caroline Klein,

Résumé

Les travaux présentés dans ce document concernent l'exploration de l'espace de conception des architectures reconfigurables pour des applications orientées traitement intensif à partir d'un haut niveau d'abstraction (niveau système).

Longtemps les concepteurs de systèmes n'avaient que deux choix de réalisation ; l'utilisation de processeurs et/ou de circuits dédiées (ASIC). Depuis quelques années une troisième possibilité est apparue, les circuits reconfigurables. Les circuits FPGA sont aujourd'hui les principaux circuits reconfigurables disponibles sur le marché. Si ils ont longtemps été utilisés uniquement pour le prototypage des ASIC, ils sont aujourd'hui en mesure de fournir une solution efficace à la réalisation matérielle d'applications dans de nombreux domaines. Néanmoins, ces circuits souffrent encore d'un certain nombre d'handicaps, entre autres leur granularité fine de traitement et leur réseau dense de routage. Aussi, de nombreux laboratoires académiques et industriels ont mis en place des travaux pour définir de nouveaux concepts d'architectures reconfigurables. Ces nouveaux concepts ont élargi la vision des FPGA, en augmentent la granularité des traitements, en modifiant les topologies et ressources de routages, en augmentant l'hétérogénéité des architectures ainsi que leur hiérarchie. De ce fait, les architectures reconfigurables constituent aujourd'hui une solution efficace pour répondre au challenge des systèmes sur puces.

Cependant les architectures reconfigurables dans leur ensemble sont pénalisées par un manque d'outils de conception indispensables à tous les niveaux du flot de conception. Dans ce mémoire, nous proposons une méthode d'exploration de l'espace architectural de conception afin de converger rapidement vers la définition d'une architecture efficace pour une application donnée. Cette méthode intervient très tôt dans le flot de conception, ainsi dès les premières phases de spécification de l'application, les concepteurs peuvent définir une architecture adaptée pour leurs applications. Notre méthode s'appuie principalement sur l'estimation de la répartition des communications dans l'architecture ainsi que sur le taux d'utilisation des ressources de l'architecture. Ces métriques permettent en effet d'orienter le processus d'exploration afin de minimiser la consommation de puissance de l'architecture puisque cette dernière est directement corrélée à ces deux métriques.

Ces travaux ont conduit au développement d'un outil qui s'inscrit dans un environnement logiciel plus large développé au LESTER ; Design Trotter. Nous avons appliqué notre méthode d'exploration architecturale à des applications du traitement des images et de la cryptographie. Les résultats obtenus montre que notre méthode permet de converger rapidement vers une architecture efficace en ce qui concerne la consommation de puissance. De plus le concepteur obtient de nombreuses informations sur l'architecture reconfigurable en adéquation avec l'application développée. Enfin, nos travaux nous ont permis de mettre en évidence des styles d'architectures reconfigurables adaptés à des domaines d'applications.

Mots clefs : architectures reconfigurables, FPGA, impact des ressources de routage, exploration de l'espace de conception, estimation de la répartition des communications, estimation de performances, outil Design Trotter.

Abstract

During many years, the designers just had two possibilities to design embedded systems; microprocessors and/or application specific integrated circuits (ASIC). Nevertheless, a new possibility has appeared lately, the reconfigurable circuits. Field Programmable Gate Array (FPGA) is the most famous version of these circuits. Today, FPGA capacities and speed are large enough to offer an interesting solution for application implementation.

However, FPGA is characterized by a fine grain architecture, which leads to a tremendous number of routing elements to connect together the computation resources (typically LUT). Hence, this kind of architecture offers a very large flexibility but its performance is reduced due to the communication overhead (latency and power consumption).

Actually, to propose more efficient reconfigurable circuits many academic and industrial laboratories work to improve the reconfigurable architecture concept (flexibility and high performances). Mainly, the new concept is coarse grain architectures. These architectures are characterized by an improvement of the routing resources, and an increase of the architecture heterogeneity and hierarchy. Thanks to these improvements, reconfigurable architectures are becoming very attractive and efficient solutions to meet with system on chip challenges in a near future (e.g. software radio and security applications).

Nevertheless, reconfigurable architectures have an important lack of tools at all design levels. Designers are then faced to 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 to perform 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 thesis, 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. Indeed, an algorithmic description of the application is considered (subset of the C language). The designer can define quickly an efficient reconfigurable architecture.

This method is mainly based on communication analysis. The aim is to provide an efficient repartition of communications within the architecture to minimize their overhead. For that purpose a hierarchical distribution is considered to improve communication locality. Architecture resources are also analyzed to increase their utilization rate. To provide this information a suited specification graph has been defined (ACG, Average Communication Graph) as a functional model to describe the architecture.

Synergy values are provided to the designers thanks to an estimation tool, which has been developed during this thesis. The integration of this tool within the CoDesign Tool, Design Trotter, permits to perform a thorough application and architectural exploration from an algorithmic specification. According to the results of this work, we have shown that it is possible to define efficient dedicated reconfigurable architectures for specific application domains. This specialization of reconfigurable architecture is a condition to meet with in order to define high performance and flexible architectures.

Key words: Reconfigurable Architectures, FPGA, Design Space Exploration, Communication Distribution, Performances Estimation, and Design Trotter a CoDesign Tool.

Remerciements

Ce document présente les travaux que j'ai effectués durant trois ans au LESTER et je remercie les personnes de ce laboratoire qui m'ont permis de les réaliser dans de bonnes conditions et en particulier M. Eric Martin le directeur du laboratoire. Je remercie aussi chaleureusement tout le personnel de ce laboratoire qui m'a très bien accueilli, même les reconfigurable-sceptiques.

Je remercie M. Didier Demigny et M. Lionel Torres d'avoir accepté d'être les rapporteurs de cette thèse et de faire partie du jury lors de la soutenance. Je remercie M. Lionël Liénard qui a accepté de présider le jury ainsi que M. Bernard Pottier et M. J. Cambonie qui ont montré tout l'interêt qu'ils portaient à mes travaux en participants à ce jury.

Je remercie particulièrement mon directeur de thèse M. Jean-Luc Philippe, car il a toujours été disponible et n'a en aucun cas cherché à m'imposer ses vues. Nos échanges sur mes travaux furent toujours enrichissants et sources de compromis intéressants.

Je tiens à remercier chaleureusement mon encadrant et ami Guy Gogniat. Merci Guy car tu as toujours su assurer ma progression que ce soit au laboratoire ou le long des falaises bretonnes. Ta rencontre fait partie de celles qui comptent, j'espère avoir l'occasion de travailler longtemps avec toi car c'est un vrai plaisir.

Je remercie M. Wayne Burleson et toute son équipe de l'Université du Massachusetts que j'ai eu la chance de visiter durant l'été 2003. J'y ai vécu un des grands moments de ma thèse.

Je remercie tous les collaborateurs du projet Design Trotter ainsi que tous les stagiaires d'IUT, de maîtrise et de DEA qui ont participé à ce projet et à mes travaux.

Ma thèse fut l'occasion pour moi d'enseigner en tant que moniteur à l'IUP de Lorient, à l'UBS et à l'ENS de Cachan, aussi je tiens à remercier tous les étudiants à qui j'ai transmis la bonne parole de l'électronique. Même s'ils n'ont pas toujours été sensibles aux douces mélodies des circuits électroniques, ils ont, à leur insu, amélioré ma façon d'enseigner et je les en remercie en leur souhaitant de grandes réussites.

Je ne peux pas oublier tous ceux qui partagent ma vie personnelle et qui m'ont soutenu durant cette thèse, et bien souvent durant les années antérieures. Je pense évidement à mes parents et ma famille proche, à mes petits-neveux et leurs sourires d'enfants. Je pense aussi à tous mes amis qui se reconnaîtront, alors merci à tous pour nos romans e-pistolaires journaliers remplis d'humour et de bonne humeur, pour nos randonnées nocturnes Lorientaises, Angevines, Rennaises et Pornichéaines, pour nos bonnes bouffes, pour nos

dégustations de vodka polonaise parfumée à l'herbe de bisons, et pour ces fous rires qui font que la vie est plus sympa, merci.

Table des matières

Chapitre 4

Chapitre 5

Conclusions et perspectives

Table des figures

Liste des tableaux

Introduction

Motivation

L'innovation, moteur de la recherche, est liée fortement au domaine de l'électronique. Cette innovation a été la source de nombreux bouleversements scientifiques mais également bien au-delà puisqu'elle a induit des évolutions sociologiques profondes. Depuis une dizaine d'années on assiste à une très forte accélération du rythme des innovations dans le domaine de l'électronique ce qui conduit parfois à l'apparition de décalages plus ou moins marqués entre les attentes des utilisateurs et les solutions technologiques existantes.

Afin d'illustrer ce propos et de montrer le caractère fondamentalement mouvant du domaine de l'électronique il est intéressant de faire un rapide survol de quelques faits marquants de son histoire. Le premier tube à incandescence fut inventé en 1875 par Sir J.W. Swan, le premier transistor bipolaire par les ingénieurs Bardeen, Brattain et Shockley dans les laboratoires de la société Bells en 1948 et le premier brevet concernant les circuits intégrés fut déposé en 1958 par la société Texas Instruments [Riordan04]. L'électronique a cette propriété fascinante d'être à la croisée de nombreux domaines scientifiques et technologiques qui fait d'elle une discipline en perpétuelle évolution. Aujourd'hui elle fait complètement, et de façon de plus en plus transparente, partie de nos vies dans nos sociétés modernes, en particulier grâce à l'évolution d'un concept : les systèmes embarqués (le plus souvent sans fils). Ce concept a étendu significativement le champ des possibilités applicatives de l'électronique, elle est devenue nomade, diffuse, de moins en moins coûteuse mais de plus en plus complexe !

L'électronique numérique prend quant à elle une part de plus en plus importante dans le domaine plus large de l'électronique. Elle correspond au cœur des systèmes embarqués. Pour réaliser ces derniers, les concepteurs ont longtemps disposé d'un choix réduit à deux possibilités. Ils pouvaient mettre en œuvre un système programmé via un microprocesseur ou alors ils devaient se lancer dans la complexe réalisation d'un circuit spécifique à l'application développée (ASIC, Application Specific Integrated Circuit). Cependant, depuis quelques années une autre possibilité s'offre à eux, les circuits reconfigurables. Ces derniers correspondent à des circuits matériels dont l'architecture est configurable en fonction de l'application à développer. L'utilisation de ces derniers au sein des systèmes embarqués résulte d'un processus d'évolution et d'innovation relativement long puisque s'étalant sur plusieurs dizaines d'années.

Les ambassadeurs de ces circuits sont les FPGA (Field Programamble Gate Array), circuits commerciaux reconfigurables. En 1984 (tout juste vingt ans !) la société américaine Xilinx fut précurseur du domaine en lançant le premier circuit FPGA commercial, le XC2000. Ce composant avait une capacité maximum de 1500 portes logiques. La technologie utilisée était alors une technologie aluminium à 2µm avec 2 niveaux de métallisation. Xilinx sera suivi un peu plus tard, et jamais lâché, par son plus sérieux concurrent Altera qui lança en 1992 la famille de FPGA FLEX 8000 dont la capacité maximum atteignait 15 000 portes logiques.

Ces composants apportent de nouveaux concepts grâce à leur universalité applicative et à leurs possibilités de reconfiguration. Longtemps ils ne pouvaient être intégrés directement aux produits commerciaux, à part pour réaliser quelques fonctions logiques. Ils étaient pourtant très utilisés dans les bureaux d'études pour le prototypage d'ASIC. Leur faculté de reconfiguration offrait aux concepteurs un processus de développement avec des possibilités de conception itérative. Cependant, à leur début ils n'avaient pas des capacités, en nombre de portes logiques équivalentes, suffisantes pour permettre la réalisation d'applications complètes, de plus, leurs performances étaient très en dessous de celles des ASIC. Les outils dédiés aux FPGA offraient des performances limitées car ils étaient issus des outils de conception d'ASIC qui ne prenaient pas en compte toutes les spécificités des composants reconfigurables.

Depuis les années 2000, année de début des travaux présentés dans ce document, des évolutions majeures ont été apportées. Tout d'abord les technologies utilisées pour les FPGA sont les mêmes que celles utilisées pour les ASIC. Par exemple, la technologie cuivre est également mise en œuvre pour la réalisation des métallisations au sein des FPGA. Elle permet une réduction d'environ 70% des temps de propagation des signaux le long des métallisations par rapport à la technologie aluminium. En 2000, les technologies utilisées étaient des technologies CMOS 0,15µm avec 8 niveaux de métallisation, aujourd'hui la technologie utilisée est la technologie CMOS 0,09µm avec 12 niveaux de métallisation. En 2000 et 2001 les deux concurrents Xilinx et Altera ont franchi une nouvelle étape au niveau de la densité d'intégration en sortant respectivement leurs circuits Virtex et Apex-II dont les capacités maximums avoisinaient les 4 millions de portes logiques équivalentes avec de plus l'introduction de larges bancs de mémoires embarqués. Aujourd'hui, les fréquences de fonctionnement de ces circuits sont de l'ordre de quelques centaines de mégaHertz, (ces dernières sont en réalité très dépendantes de l'application). Bien que ces valeurs soient relativement réduites par rapport aux ASIC, elles sont suffisantes pour une très large majorité d'applications actuelles comme illustré sur la figure 1.

La figure 1 [Altera01a] met en évidence qu'à partir des années 2000 les capacités des FPGA ont permis d'offrir aux concepteurs une solution supplémentaire de réalisation pour une majorité d'applications. De plus, les outils de mise en œuvre des FPGA ont évolué et bien qu'encore pénalisants lors de la conception, ils permettent la réalisation rapide d'applications complexes.

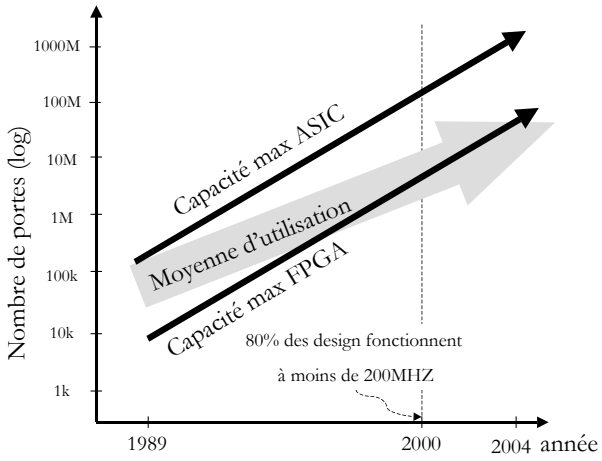


Figure 1 – Comparaison de l'évolution des capacités des circuits ASIC et FPGA par rapport à la moyenne d'utilisation des ressources des applications.

Dans un contexte économique mondial incertain les FPGA apparaissent donc comme une solution flexible bien adaptée aux contraintes économiques telles que le temps de mise sur le marché et le potentiel d'évolution ou de flexibilité des produits. De plus, le modèle économique lié aux FPGA, qui est un modèle linéaire devient de plus en plus avantageux par rapport aux modèles ASIC dont le coût du premier échantillon fabriqué rend l'amortissement d'une telle solution long et viable économiquement uniquement pour de très grandes productions. Dans [Tredennick03] l'auteur explique bien ce phénomène et propose la figure 2 qui montre que le nombre de circuits fabriqués à partir duquel la solution ASIC est économiquement plus rentable (point de cross-over) tend à augmenter avec l'évolution des technologies. Les solutions FPGA sont donc de plus en plus intéressantes tant sur le point technique qu'économique.

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. Effectivement, la granularité fine des opérateurs de traitements demande un réseau de routage dense. Or les ressources de routage sont fortement pénalisantes pour la consommation de puissance des circuits. De plus, ces mêmes ressources dégradent les performances temporelles des circuits. Pour remédier à ces problèmes de nombreux laboratoires universitaires ou industriels se sont lancés dans la définition d'architectures reconfigurables 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. Parfois proches des FPGA, parfois proches de microprocesseurs élémentaires mis massivement en parallèle, les architectures reconfigurables sont aujourd'hui au centre d'un important domaine de recherche en pleine ébullition. Comme nous le verrons tout au long de ce mémoire.

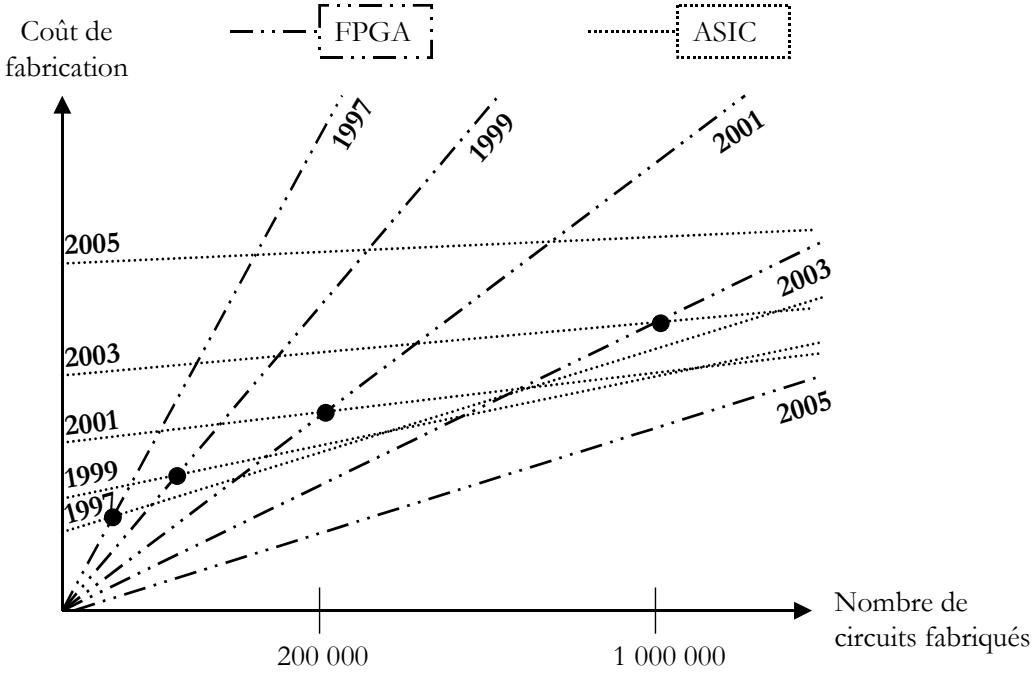


Figure 2 – Progression du point de cross-over entre solutions ASIC et FPGA depuis l'année 1997 à l'année 2005 [Tredennick03].

Ces architectures se présentent aujourd'hui comme une réponse intéressante au challenge des systèmes sur puce. 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. 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, ce qui conduit leurs concepteurs à intégrer nombre d'éléments différents (par exemple des mémoires ou des multiplieurs). Les ressources de communications, des bus aux réseaux filaires de routages, sont souvent intégrées conjointement au sein d'une même architecture afin de faciliter des 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 jungle (comme le disaient les auteurs de [Shaumont01] "*A Quick Safari Through the Reconfigurable Jungle*") dans laquelle le concepteur aventurier à bien besoin qu'on le guide.

Historiquement de nombreuses études ont été réalisées pour explorer le domaine des architectures reconfigurables de grain fin (FPGA), nous pouvons citer en exemple les travaux remarquables réalisés à l'Université de Toronto au Canada [Wilton97], [Betz99]. Les explorations visaient par exemple la taille des fonctions logiques, des éléments de mémorisation, le type de routages ou de matrices de connexion. De telles études n'existent pas pour le moment en ce qui concernent les architectures hétérogènes de gros grain. Si ces études n'ont pu être réalisées cela est dû, en partie, par un manque de méthodes et d'outils d'estimation des performances et/ou d'exploration architecturale ciblant spécifiquement les architectures reconfigurables hétérogènes et de gros grain.

Problématique de l'étude

La problématique de l'étude décrite dans ce document 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.

Contributions

Les travaux décrits dans ce document ont abouti au développement d'une méthodologie d'aide à la conception d'architectures et plus précisément de mise en adéquation d'une architecture reconfigurable avec une application donnée. Cette méthodologie permet au concepteur d'effectuer une exploration architecturale d'un large espace de conception. D'autres méthodes d'exploration de l'espace de conception ciblant des architectures reconfigurables existent mais elles ciblent toutes un ensemble très réduit d'architectures, voir même le plus souvent une seule architecture dont ils font varier quelques paramètres.

Notre méthode permet au concepteur d'obtenir de nombreuses informations sur la réalisation potentielle d'une application sur une architecture modélisée. Le modèle que nous proposons pour la spécification des architectures reconfigurables est un modèle bien adapté à une large exploration de l'espace de conception.

La consommation de puissance étant un des enjeux majeurs actuels, en particulier pour les systèmes embarqués, il nous est apparu essentiel de se concentrer sur la prise en compte de

l'impact des communications (point le plus pénalisant en ce qui concerne la consommation de puissance) sur les performances des architectures reconfigurables et de guider principalement notre méthode d'exploration sur ce critère. D'autres critères sont importants, parmi ceux-ci nous en considérons plusieurs tels que : le taux d'utilisation des ressources de l'architecture, la granularité des opérateurs, la taille des mémoires embarquées, la hiérarchie de l'architecture et la taille des niveaux hiérarchiques.

Nos travaux permettent donc une exploration et une comparaison des architectures reconfigurables. Il nous est possible de déterminer quel style d'architecture est plus efficace pour tel domaine d'applications.

Contexte

Les travaux présentés dans ce document ont été réalisés au sein du projet Design Trotter développé au LESTER. Ce projet intègre plusieurs outils logiciels autour d'un même environnement. Au début de ces travaux, le laboratoire LESTER avait peu d'expérience dans le domaine des architectures reconfigurables mise à part la mise en œuvre de circuits FPGA. Ces travaux ont été les instigateurs d'une "culture reconfigurable" au sein du laboratoire. De plus, le LESTER était alors engagé dans le projet RNTL EPICURE qui visait la conception d'une architecture hétérogène dont la partie matérielle était reconfigurable. D'où la nécessité d'adjoindre au projet Design Trotter la possibilité de prendre en compte de telles architectures dans ses cibles architecturales. Au sein du projet EPICURE la partie reconfigurable était du type grain fin mais nous avons rapidement étendu nos travaux aux architectures gros grain.

Plan du mémoire

Ce document présente les travaux qui ont conduit au développement d'une méthode d'exploration de l'espace de conception des architectures reconfigurables. Tout d'abord afin de fournir au lecteur les éléments indispensables à une bonne compréhension de la méthode d'exploration, le premier chapitre définit et décrit l'espace de conception exploré. Dans un souci pédagogique ce premier chapitre propose une classification des architectures reconfigurables et la présentation d'architectures représentatives du domaine.

Le deuxième chapitre propose au lecteur un état de l'art des méthodes d'exploration de l'espace de conception d'une façon générale, puis d'une façon spécifique aux architectures reconfigurables. Ce chapitre présente aussi succinctement les différentes entités de Design Trotter en les positionnant par rapport aux méthodes présentées dans l'état de l'art. Ce chapitre met en évidence l'importance des spécifications d'entrées que ce soit pour les applications ou pour les architectures.

Les spécifications d'entrée de l'outil d'estimation et d'exploration sont donc présentées au troisième chapitre. Une première partie de ce troisième chapitre propose une étude technique de l'impact des éléments constitutifs des architectures reconfigurables de grain

fin sur les performances des circuits. Cette partie permet de justifier au lecteur les choix de spécifications faits dans ces travaux.

Le quatrième chapitre aborde plus en détail l'outil d'estimation qui est utilisé pour l'exploration. Ce chapitre présente les algorithmes mis en œuvre dans l'outil d'estimation ainsi que la démarche d'exploration architecturale. Des exemples didactiques sont utilisés dans ce chapitre pour illustrer les différentes étapes de la démarche d'exploration. Cependant ils ne sont pas suffisants à la compréhension précise de toutes les étapes. Le lecteur est amené alors à suivre des exemples plus complexes au cinquième chapitre.

Des exemples d'applications du domaine du traitement des images et du domaine de la cryptographie sont donc présentés au cinquième chapitre. Ces exemples permettent au lecteur de mieux saisir la méthode d'exploration et de bien comprendre l'intérêt de nos travaux. Enfin, la conclusion du document propose au lecteur des perspectives à plus ou moins longs termes pour les travaux présentés.

Chapitre 1

Bienvenue dans un monde *reconfigurable*

Les architectures reconfigurables, depuis l'apparition des FPGA au début des années 1980, sont devenues incontournables. Elles proposent une alternative attractive entre la grande flexibilité des solutions programmables et les hautes performances des circuits spécifiques. De plus, elles ont fait évoluer la conception des systèmes en profitant de leurs propriétés de reconfiguration et d'adaptabilité. Grâce à une communauté scientifique convaincue de leur intérêt, de nouveaux champs applicatifs (tel que la radio logicielle) se sont ouverts afin d'être en adéquation avec les propriétés de ces architectures.

Aujourd'hui nous avons tout juste dépassé le stade de la préhistoire des architectures reconfigurables (l'invention de la roue fut celle des FPGA), peut être en sommes nous à la fabrication et mise en forme des métaux, très loin avant la révolution industrielle. Ce chapitre se veut donc un état de l'art passé et présent et tente d'indiquer quelques futures évolutions de ce domaine émergent.

Les architectures reconfigurables sont la cible des travaux présentés dans ce document, il est donc essentiel de le débuter par un chapitre qui définit clairement leurs limites et caractéristiques. Afin d'imaginer leur espace de conception il est intéressant de proposer une classification des architectures industrielles et académiques en détaillant et comparant quelques architectures représentatives du domaine. Ce chapitre se terminera par un paragraphe proposant quelques pistes sur les futures évolutions de ces architectures.

1.1 Les architectures reconfigurables

1.1.1 Définitions

Bien des débats peuvent être engagés sur la signification des mots dans un contexte même précis. Si un nouveau domaine scientifique naît sans aucun rapport avec l'ensemble des connaissances actuelles alors les débats peuvent êtres courts car le choix des termes est à l'honneur des premiers explorateurs du domaine (bien sûr en cas d'ex æquo il peut y avoir quelques débats passionnels). Mais si ce nouveau domaine apparaît en relation avec d'autres domaines bien installés alors les explications terminologiques sont généralement incessantes. Chacun évaluant le potentiel du nouveau domaine, souhaite l'intégrer au sien, en rapprochant les termes et en s'appropriant finalement un peu de la paternité du nouveau-né. Dans ce cas il est important de bien définir les termes utilisés afin d'éviter (et ce n'est pas systématique) les confusions.

Le domaine des architectures reconfigurables est bien dans le cas d'un nouveau domaine, dont les premiers concepts datent du début des années 1980 (le 1er FPGA le XC2000 fut introduit par Xilinx en 1984). A cette même époque le domaine des circuits programmables (microprocesseur) avait déjà 10 ans (le 1er microprocesseur le 4004 fut introduit par Intel en 1972) et le domaine des ASIC (Application Specific Integrated Circuit) était bien plus ancien (positionnons son début à l'invention en 1958 du circuit intégré par Jack Kibly de la société Texas Instruments). De ce fait les architectures reconfigurables eurent droit à un baptême peu précis ; le terme "architecture" pouvant décrire bien des concepts différents dans de nombreux domaines tout comme le terme "reconfigurable". Il faut donc dans un premier temps tenter de définir ces termes dans le cadre de notre étude. Nous avons conscience que les définitions que nous allons apporter ne sont pas universelles car, nous le verrons par la suite, le domaine des architectures reconfigurables est très large. Il est donc difficile, voir impossible, de définir l'ensemble des architectures reconfigurables sous une unique définition sans soulever de critiques. Par exemple la disparité entre les architectures dites "gros grain" et "grain fin" est difficilement représentable par un unique concept. Tâchons de définir les termes "architecture" et "reconfigurable" afin de voir si cela suffit à définir le domaine des architectures reconfigurables par rapport au domaine du spécifique et du programmable.

Architecture : Dans notre cas l'architecture est celle interne aux systèmes reconfigurables sur puce (RSoC : *Reconfigurable System on Chip*), ces systèmes étendent le concept de système sur puce (SoC : *System on Chip*) en intégrant des parties reconfigurables. L'architecture décrit comment les éléments de traitement, de mémorisation et de contrôle sont agencés et comment ils communiquent. 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. Si l'on fait une analogie avec une ville, le mot architecture ainsi utilisé décrit la topologie des habitations et des zones d'activités ainsi que le plan des réseaux routiers et de communications (l'auteur a d'ailleurs utilisé cette analogie pour vulgariser ces travaux [Bossuet02e]).

Reconfigurable : cet adjectif 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. Dans une architecture reconfigurable chacun de ces éléments peut être configuré et reconfiguré. La configuration décrit le potentiel pour un élément d'avoir plusieurs états possibles et distincts de fonctionnement. Chaque état porte le nom de configuration. La reconfiguration décrit la possibilité pour un élément de changer d'état (donc de configuration) au court 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 (OTP : *One Time Programmable*). Ceci est dû à la technologie utilisée pour mémoriser la configuration (technologie antifusible par exemple), nous considérons ces circuits comme configurables mais pas 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. Maintenant que nous avons défini le terme d'architecture reconfigurable, nous devons savoir s'il délimite clairement un espace de conception. Effectivement dans le cadre de la réalisation d'applications de l'électronique numérique il existe plusieurs choix d'implémentation.

Tout d'abord **les solutions spécifiques** : Dans ce cas le circuit intégré est *spécifiquement* conçu pour l'application développée. Soit le circuit est lors de sa conception en fonderie spécialement réalisé pour une application c'est le cas des ASIC, soit c'est un circuit configurable une seule fois qui après la configuration ne peut réaliser qu'une application (la société Actel propose de nombreux circuits de ce type). Il est possible, bien entendu, de décrire l'architecture de tels circuits conformément à la définition que nous en avons fait précédemment. Mais un circuit spécifique est statique, après fabrication ou configuration suivant le cas, aucun élément interne ne bénéficie de la propriété de reconfiguration.

Une autre solution de réalisation concerne l'utilisation de **circuits programmables**. Cette catégorie représente déjà un large espace. Il existe des processeurs généralistes tels que les processeurs CISC (Complex Instruction Set Computer) à jeu d'instructions complexe ou les processeurs RISC (Reduced Instruction Set Computer) à jeu d'instructions réduit. Il existe des processeurs spécialisés pour un type d'application, tels que les DSP (Digital Signal Processor) qui disposent d'un jeu d'instructions complexe optimisé pour un domaine d'applications. Dans tous les cas l'exécution de l'application passe par la lecture et le décodage d'instructions, organisées dans une mémoire (ou des mémoires), qui modifient le chemin de données (par aiguillage) ainsi que la fonction de l'unité de traitement (ou des unités de traitements dans le cas d'un processeur super scalaire). De ce point de vue nous pouvons dire que la définition d'architecture reconfigurable précédemment développée peut être appliquée à ce type de système. Certains ont d'ailleurs avec intelligence choisi cette option [David03]. Dans notre étude nous souhaitons séparer les architectures reconfigurables des systèmes programmables puisque notre étude précisément ne porte pas sur ces derniers.

Pour séparer les architectures reconfigurables des systèmes programmables il faut trouver au moins une différence. Analysons la reconfigurabilité des systèmes programmables. Si nous considérons que chaque instruction se déroule sur un cycle, la reconfiguration intervient à chacun des cycles. La configuration, ou l'état des éléments constitutifs de l'architecture du système programmable est donc remis en cause à chaque cycle. La configuration a donc besoin d'une excitation constante à chaque cycle. Cette excitation correspond donc à une instruction et l'ensemble des excitations (donc des instructions) décrit le programme. Le système est programmable car il a besoin d'un programme, donc d'une suite d'instructions pour connaître son état. On peut dire qu'un système programmable n'est pas conscient de son état puisqu'il doit se le "remémorer" à chaque cycle. De cette discrimination des systèmes programmables, nous pouvons séparer les architectures reconfigurables en disant qu'elles sont conscientes de leur état et donc de leur configuration. En effet cette dernière n'a pas besoin d'être remise en question à chaque cycle car la configuration est choisie afin qu'un nombre important de données puisse être traité. L'objectif est donc de trouver les états les plus stables car c'est à ce niveau que peut se faire d'importants gains en performance. En conclusion nous pouvons dire que la configuration des systèmes programmables est astable (aucun état stable) alors que celle des architectures reconfigurables est multi-stable (plusieurs états stables). Mais ce n'est pas là la seule différence entre les deux domaines puisque la répartition des traitements est temporelle dans le cas des architectures programmables et elle est spatiale dans le cas des architectures reconfigurables [David03]. Cette dernière différence est un point essentiel de divergence entre les concepts d'architectures reconfigurables et le concept d'architectures programmables. Le schéma de la figure 3 donne une synthèse des définitions précédentes.

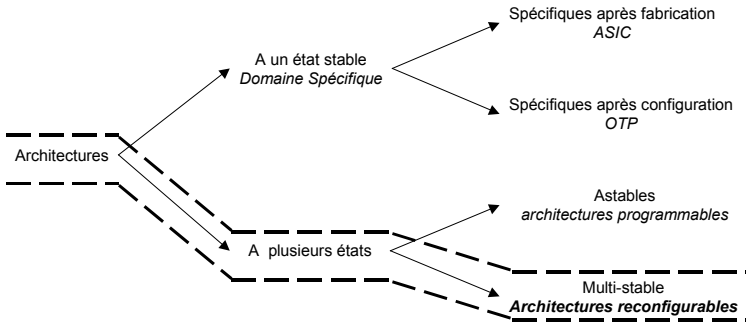


Figure 3 – Schéma de définition des architectures spécifiques, programmables et reconfigurables.

1.1.2 Positionnement du domaine du reconfigurable

Pour compléter notre analyse nous pouvons positionner entre eux les différents systèmes que sont les circuits spécifiques (ASIC), les architectures reconfigurables (AR) et les systèmes programmables (µp et DSP). Ce positionnement se fera selon les "*cinq P*" ; **P**rogrammabilité, **P**erformance, **P**uissance, **P**arallélisme et **P**rix. Ces cinq mots simples

décrivent en fait un ensemble de propriétés de chacun des systèmes qui permettent de les comparer. Détaillons chacun de ces termes :

Programmabilité : ce terme a avant tout l'avantage de commencer par un P, mais il aurait aussi bien pu être remplacé par *Reconfigurabilité* ou *Flexibilité*. Il s'agit de caractériser le degré avec lequel on peut modifier le système par un processus déterministe. Dans [Demigny02], les auteurs présentent le concept de *rémanence* qui s'apparente à la programmabilité dynamique du système en considérant le cycle de reconfiguration et le nombre d'éléments reconfigurables. Les processeurs généraux (µp) ont la plus grande reconfigurabilité, car l'état du système est changé à chaque cycle, et les possibilités d'états sont nombreuses. Le nombre d'états possibles du système est moins grand dans le cas de processeurs DSP car le jeu d'instruction est réduit ce qui diminue le nombre de configurations architecturales possibles. A l'opposé les ASIC ont un, et un seul état stable, leur programmabilité est minimum.

Performances : pour les systèmes électroniques numériques, les performances sont liées au nombre de calculs qu'ils peuvent réaliser durant un temps donné (on parle par exemple de MOPS pour million d'opérations par seconde), mais aussi au nombre de données qu'ils sont capables d'échanger avec leur environnement. On parle alors de débit en Giga Bits par seconde. Suivant le type d'application développée la performance en nombre de calculs peut être plus importante que celle en nombre d'échanges avec l'extérieur. On peut lier ces deux critères de performance pour une comparaison générale des systèmes. Mais il est nécessaire de les séparer lors d'une comparaison plus fine entre des éléments d'un même domaine.

Puissance : il s'agit en fait de puissance consommée par le système. Bien qu'elle puisse s'apparenter à la performance du système, il faut la séparer de la puissance de calcul et du débit puisque souvent son amélioration passe par la dégradation des autres performances. La réduction de la puissance consommée demande de plus une conception adaptée qui prenne en compte cette contrainte à toutes les étapes du flot de conception [Havinga00]. Cette contrainte sur la puissance est apparue depuis peu, entraînée par l'augmentation du marché des applications mobiles (ordinateurs portables, téléphonie, …) qui n'a pas été suivie par une amélioration significative de la capacité des batteries d'alimentation en énergie.

Parallélisme : le parallélisme évoqué ici n'est pas tant celui de l'architecture support, que celui de l'application. En effet il s'agit de comparer les différents systèmes par leur possibilité d'exploiter le parallélisme potentiel de l'application développée. Cette caractéristique est intéressante car elle permet de voir le potentiel d'optimisation de l'application en fonction de la cible. Cependant il faut disposer pour ce faire d'outils d'aide à la conception qui permettent d'estimer ou d'exploiter rapidement dans le flot de conception le parallélisme potentiel de l'application.

Prix : enfin cette dernière métrique est bien plus complexe qu'elle n'y paraît. En effet plusieurs paramètres peuvent intervenir dans le calcul du prix d'un circuit intégré et différents prix existent, du prix de revient au prix de vente. Nous considérons un prix de revient comprenant le prix de la conception (directement lié au prix des outils logiciels et au temps de conception), le prix du prototypage et du premier circuit. Nous ne nous

positionnons pas dans le cadre de grandes séries pour lesquelles des facteurs d'échelles permettent de modifier les modèles de prix de revient. L'étude complète des prix de revient demanderait un document sûrement plus important que celui-ci, l'auteur demande donc de l'indulgence sur les approximations qu'il apporte à la notion de prix de revient.

Afin de comparer simultanément les trois domaines de réalisation : domaine du spécifique, domaine du reconfigurable et domaine du programmable. L'auteur propose une approche concentrique. Dans cette approche, on peut indiquer cinq axes représentant le positionnement relatif des trois domaines suivant les "cinq P" : programmabilité, performance, puissance (consommée) prix et parallélisme.

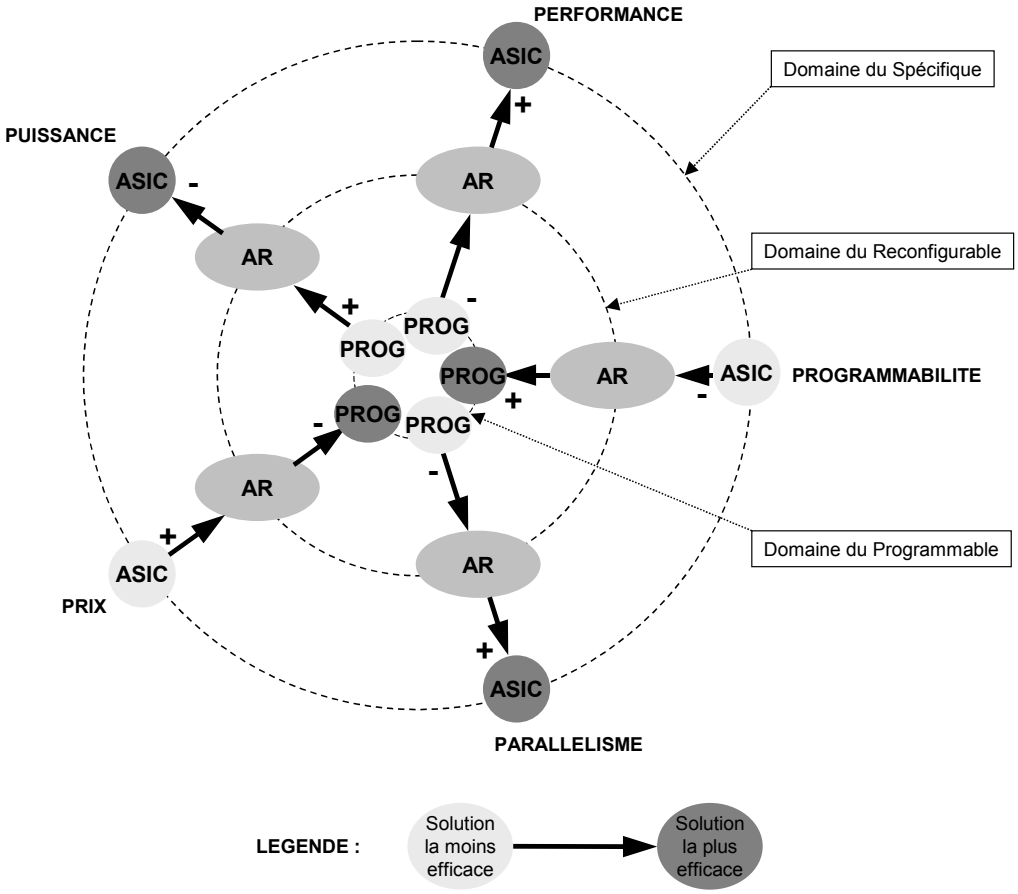


Figure 4 - Diagramme des "cinq P": positionnement concentrique des domaines spécifique, reconfigurable et programmable.

La figure 4 montre le positionnement concentrique des différents domaines, et l'orientation des axes représentant les "cinq P". Les flèches sur le graphe pointent les solutions les plus efficaces. Sur ce graphe le domaine du reconfigurable prend suivant tous les axes une place intermédiaire entre les domaines spécifique et programmable. Le domaine spécifique est caractérisé par de grandes performances pour une faible consommation de puissance, mais aussi par un coût de revient excessivement important dont une large partie correspond aux

coûts de conception et de réalisation du premier circuit (cependant les circuits spécifiques deviennent économiquement plus intéressants à partir d'un nombre élevé de circuits fabriqués comme on peut le voir sur la figure 2 du chapitre d'introduction). De plus, le domaine spécifique, de par la nature même des ASIC permet une adaptation au plus près de l'application, donc un potentiel de parallélisme maximum et une programmabilité minimum, l'ASIC n'étant pas censé réaliser autre chose que l'application pour laquelle il a été optimisé. Le domaine programmable est caractérisé par une grande programmabilité mais des performances relativement faibles car les systèmes programmés exécutent de façon séquentielle l'application, ou avec un faible parallélisme. De par le marché important des composants programmables, de leur large diffusion et de la dimension de leur domaine applicatif, ils sont très peu coûteux. Le domaine reconfigurable se place, ou a vocation à se placer entre les deux. C'est le cas des FPGA qui sont des architectures commerciales à grain fin. Pour les architectures gros grain, peu d'entre elles ont une réalisation physique et encore moins une ouverture commerciale, ce qui rend l'étude de leurs performances et prix de revient un peu spéculative, mais de nombreuses études montrent que ce positionnement est justifié [Bondalapiti02], [Hartenstein01], [Tessier01], [Rabaey00].

Nous pouvons maintenant développer plus en détails le domaine des architectures reconfigurables, pour cela il est nécessaire de définir l'ensemble des éléments et caractéristiques de ces architectures.

1.1.3 Caractéristiques des architectures reconfigurables

Pour pouvoir proposer une classification des architectures reconfigurables, il faut pouvoir les caractériser. Cinq caractéristiques essentielles peuvent être définies : la granularité des ressources de traitement, la granularité des ressources de communication (échanges de données), la topologie du réseau de communication, la liaison avec un éventuel processeur (pouvant faire partie de la même puce) et le type de reconfiguration. Bien sûr, il est possible de trouver d'autres caractéristiques, mais celles retenues ici sont essentielles et permettent de séparer et de classer les architectures entre elles.

La granularité des ressources de traitement est souvent employée pour caractériser l'architecture, une distinction est souvent faite entre grain fin et gros grain (appelé également grain épais). Le plus fin correspond à des ressources logiques le plus souvent de type LUT (Look Up Table). Les LUT, ou tables de scrutation, sont des petites mémoires que l'on retrouve dans tous les FPGA, elles sont capables de réaliser n'importe quelles fonctions booléennes des entrées. Le gros grain se rapporte le plus souvent à des architectures du type systolique lorsqu'elles sont construites autour de matrice d'ALU, voir de cœurs de processeurs élémentaires. Certaines architectures sont composées d'opérateurs arithmétiques de type additionneur, multiplieur ou MAC. La distinction est parfois compliquée car 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 ce document 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 sont des dispositifs de communication offrant une connectivité complète, c'est à dire qu'ils permettent la connexion de tous les communicants qui leur sont reliés. Ils ont donc une flexibilité importante mais un surcoût important en surface. La figure 5 montre une vue conceptuelle d'un crossbar. Sur celle-ci on voit qu'une communication d'une entrée à une sortie ne traverse qu'une connexion configurable. Le passage de signaux dans des connexions configurables est pénalisant tant pour la fréquence que pour la consommation de puissance, dans le cas du crossbar le nombre de passage à travers une connexion configurable est réduit à un ce qui le rend très performant.

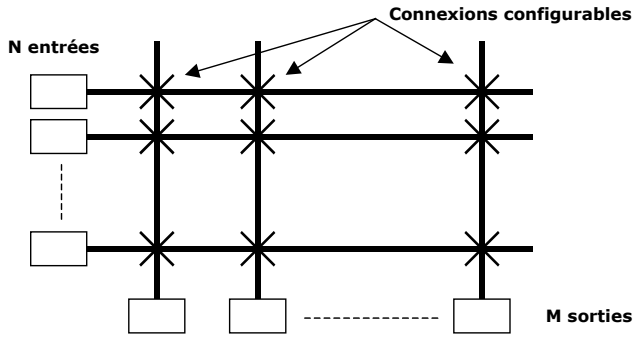


Figure 5 – Vue conceptuelle d'un crossbar

Les réseaux de bus sont des dispositifs partagés, cette solution est la plus souvent utilisée car elle propose un compromis intéressant entre la flexibilité des communications et la surface occupée. La figure 6 montre un schéma de connexions à bus, le nombre de bus à choisir correspond au nombre maximum d'interconnexions qui peuvent être réalisées de façon concurrente.

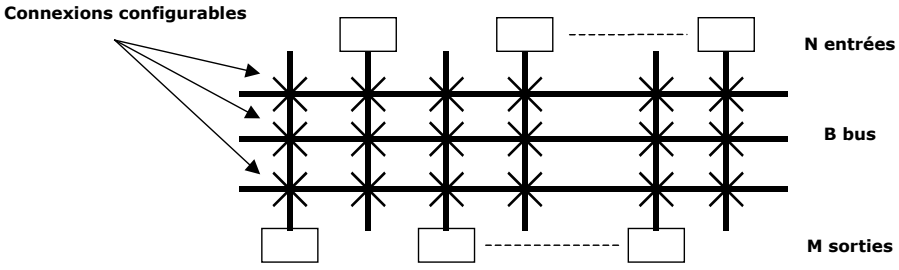


Figure 6 – Schéma de connexions à bus

Les deux dispositifs de communication présentés forment des réseaux d'interconnexion globaux qui fournissent des connexions à des coûts identiques entre n'importe quels modules (ressources de traitement, mémoires, entrées/sorties). Certaines architectures, de par le nombre de modules, et donc le nombre de connexions, doivent utiliser un réseau de

connexion local. Dans ce cas les connexions se font point à point. C'est le cas des architectures grain fin. La figure 7 montre une architecture grain fin utilisant ce type de connexions. L'utilisation de matrices de connexions configurables est indispensable pour assurer la connectivité des modules, mais les matrices de connexions configurables dégradent les caractéristiques des signaux, diminuent les performances (fréquence de fonctionnement et consommation de puissance) et nécessitent des outils de placement-routage efficaces [Wilton97]. Sur la figure 7 on peut voir en gras des liaisons point à point entre deux éléments configurables. Ces liaisons utilisent : les ports d'entrées/sorties des éléments configurables, les connexions configurables qui permettent la connexion des éléments configurables au réseau de routage, les lignes de routages et les matrices de connexions configurables. Autant d'éléments parcourus qui dégradent les performances du circuit mais qui permettent une flexibilité importante. Le début du troisième chapitre de ce document reviendra sur les dégradations de performances liées aux ressources de connexions de ce type d'architectures.

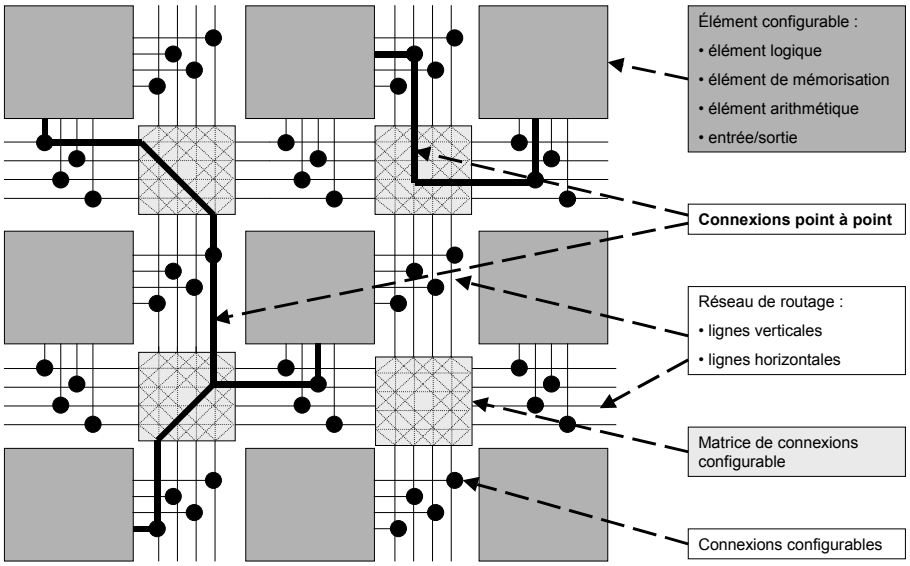


Figure 7 – Exemple d'architecture reconfigurable grain fin, utilisant une connexion point à point filaire.

La topologie du réseau de communication décrit comment les lignes de communication (bus ou canaux de fil) sont disposées dans l'architecture et comment les éléments de connexion sont agencés. La figure 7 montre par exemple une topologie de type matricielle, les éléments à connecter sont disposés sous forme de matrice, les connexions sont verticales et horizontales. Du fait de la régularité de cette topologie elle est très souvent mise en œuvre. Cependant 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.

De cette façon en fonction de la longueur des connexions le type de routage est choisi afin de minimiser les coûts de communication. Il est ainsi possible d'utiliser des bus pour des connexions entre blocs hiérarchiques appelés clusters et des canaux de fils segmentés à l'intérieur des clusters. Ces architectures sont très efficaces, puisque les ressources de communication sont adaptées à la longueur des échanges de données. Mais elles nécessitent des outils de placement adaptés qui soient en mesure de profiter de la hiérarchie du réseau de communication en favorisant les communications locales, rapides et peu consommatrices en énergie, pour réaliser les connexions entre ressources fortement communiquantes. La mise en cluster doit refléter les communications inhérentes à l'application [Bossuet03b].

La liaison avec le microprocesseur n'a d'objet que dans le cadre où l'architecture reconfigurable joue le rôle d'accélérateur matériel. Les processeurs n'ont pas été développés pour faire du calcul arithmétique parallèle intensif (bien que les processeurs spécialisés comme les DSP puissent être efficaces dans ce sens). Les architectures matérielles et reconfigurables sont bien adaptées à traiter massivement et parallèlement les calculs arithmétiques, particulièrement en profitant du parallélisme potentiel de ces calculs. Aussi en tirant parti des propriétés respectives des systèmes programmables et des systèmes reconfigurables il est possible d'améliorer l'adéquation du système global avec l'application développée. Dans ce cas l'utilisation de méthodes de conception conjointe logicielle/matérielle est indispensable et demande un effort important en développement d'outils [C.T.I. 98]. Dans le cas de coopération entre un processeur et une partie matérielle reconfigurable, leur liaison peut être plus ou moins "proche". La partie reconfigurable peut être couplée avec le processeur comme un périphérique, les échanges de données se faisant donc via une interface (parallèle ou série) qui gère les communications. Dans ce cas le processeur échange des données avec l'accélérateur matériel comme avec n'importe quel périphérique. Il est possible de coupler directement le processeur et l'accélérateur matériel reconfigurable via un bus. Dans ce cas le dialogue entre le processeur et l'accélérateur se fait avec un protocole commun aux deux entités. Enfin, lorsqu'il est possible d'intégrer sur la même puce de silicium le processeur et l'accélérateur (on parle alors de coprocesseur reconfigurable) le couplage peut être direct. La partie reconfigurable se trouve intégrée comme unité de traitement du processeur, elle est donc directement sur le chemin de données [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 (présentée dans le paragraphe 1.2.4) 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. La reconfiguration du FPGA Virtex est donc partielle par colonne. La figure 8 montre différents cas de reconfiguration complète, partielle, partielle par colonne et partielle maximum. Dans ce dernier cas M_{ec} est égal à un. C'est à dire qu'il est possible de configurer séparément (donc de façon partielle) au minimum un élément configurable de base. Dans ce cas le rapport γ est donc minimum. Sur la figure 8 chaque grande matrice représente l'architecture reconfigurable, et chaque petite matrice représente la configuration minimum nécessaire (par exemple le bitstream pour les FPGA). Un carré de la grande matrice représente un élément reconfigurable de base. Un carré de la petite matrice représente la configuration d'un de ces éléments reconfigurables de base. Sur cette figure, les zones de niveaux de gris différents sur la grande matrice décrivent des zones qui peuvent être reconfigurées indépendamment les unes des autres. Plus le rapport γ est petit plus la taille de la configuration minimum est petite. Il est intéressant d'avoir une taille minimum de configuration faible si l'on souhaite avoir une grande flexibilité de reconfiguration. La figure 8 reste une vision très schématique de la reconfiguration. Effectivement les ressources reconfigurables ne sont pas des "carrés" disposés en matrices (en particulier en ce qui concerne les ressources de communication). Mais elle permet de mieux comprendre qu'il est possible de reconfigurer plus ou moins finement les architectures de façon partielle.

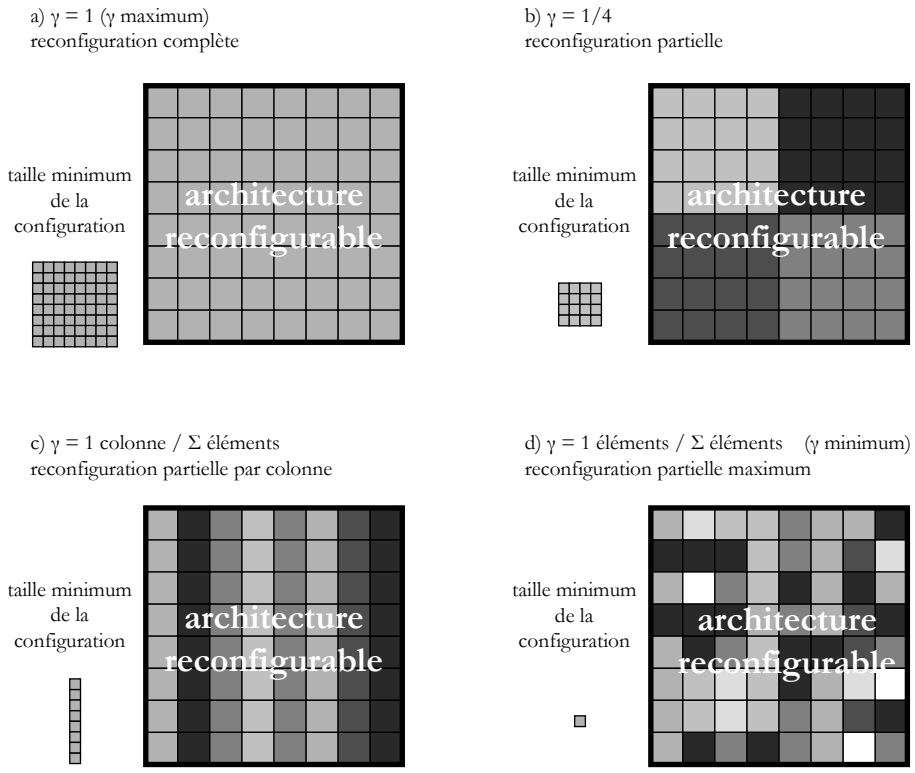


Figure 8 – Exemple de reconfiguration complète et partielle à différents degrés.

La reconfiguration de l'architecture peut être mise en œuvre de plusieurs façons au cours de l'exécution de l'application. Elle peut intervenir une seule fois sans remise en cause durant l'exécution de l'application. Nous parlerons dans ce cas de reconfiguration statique. Les processus de reconfiguration et d'exécution de l'application sont, dans ce cas, distincts et clairement séparés dans le temps. Une nouvelle reconfiguration sera effectuée dans deux cas. Tout d'abord elle peut être la conséquence d'une perte de la configuration pouvant être due pour certains dispositifs à un arrêt de la source d'alimentation (cas des FPGA de technologie SRAM par exemple). Mais elle peut aussi faire suite à une modification de l'application par le concepteur si celle-ci est défaillante ou peut être améliorée.

Cependant une étude approfondie de l'exécution d'une application peut éventuellement mettre en évidence que certaines parties configurées de l'architecture ne sont nécessaires qu'un temps court vis à vis de la durée de l'exécution, et peuvent par contre utiliser une place conséquente dans l'architecture. Le ratio entre le temps d'utilisation de la partie configurée et la taille qu'elle occupe sur l'architecture peut être faible. D'où l'idée d'introduire un dynamisme temporel à la reconfiguration. Lorsqu'une partie de l'application est exécutée, et dans le cas ou elle ne l'est plus dans un temps proche, on peut reconfigurer les éléments qui lui étaient dédiés afin de les utiliser pour une autre partie de l'application. Cette reconfiguration se fait concurrentiellement à l'exécution de l'application. La reconfiguration dynamique permet donc d'optimiser la surface configurée dans le temps. Etant donné qu'il s'agit de modifier dans le temps uniquement une partie des éléments configurables il faut nécessairement utiliser une architecture à reconfiguration partielle. Ici l'inconvénient vient qu'il est nécessaire de bien déterminer le partitionnement dans le temps de l'application afin de profiter de toute la surface du circuit. Il peut exister des problèmes de fragmentation comme sur les disques durs des ordinateurs [Compton99], [Compton00]. De plus, il faut correctement établir les communications entre les partitions [Delahaye03].

Une solution pour améliorer le dispositif de reconfiguration partielle est de rajouter une dimension à la mémoire de configuration. Dans ce cas il faut parler de reconfiguration multi-contextes [Trimberger97], un contexte correspondant à une configuration. L'approche multi-contextes peut se voir comme un prépositionnement de configurations dans la vue de leur utilisation dans un futur proche, c'est le concept des mémoires caches d'instructions des processeurs, à un degré moindre. La mémoire de contextes va permettre d'anticiper le prochain état de configuration de l'architecture. Ainsi le basculement de configuration est accéléré et permet un grand dynamisme de reconfiguration. Mais l'utilisation de cette technique prometteuse demande le développement de méthodes d'aide au choix des contextes, et de gestion de la reconfiguration [Auguin03]. Cependant, l'architecture modulaire Ardoise [Kessal00], basée sur l'utilisation de cicuits FPGA Atmel AT40K40, met en œuvre la reconfiguration dynamique par pré-positionnement de configurations, cette architecture est principalement dédiée au domaine du traitement temps réel des images.

Les figures 9 et 10 présentent schématiquement le déroulement de la reconfiguration durant l'exécution de l'application dans différents cas. Pour chacune de ces figures, un carré gris représente un élément configuré, un carré blanc représente un élément non utilisé. Si deux carrés sont de niveaux de gris différents alors c'est qu'ils sont dédiés à la réalisation de

deux parties (ou tâches) distinctes de l'application (notamment dans le cas des reconfigurations partielles). La figure 9 présente le cas de reconfigurations simples contextes. Cette figure nous permet de schématiser la différence entre les reconfigurations complète et partielle. Durant le temps d'exécution de l'application la reconfiguration complète est statique contrairement à la reconfiguration partielle qui sous certaines conditions permet de reconfigurer une partie du circuit sans affecter les parties configurées qui ne doivent pas être modifiées.

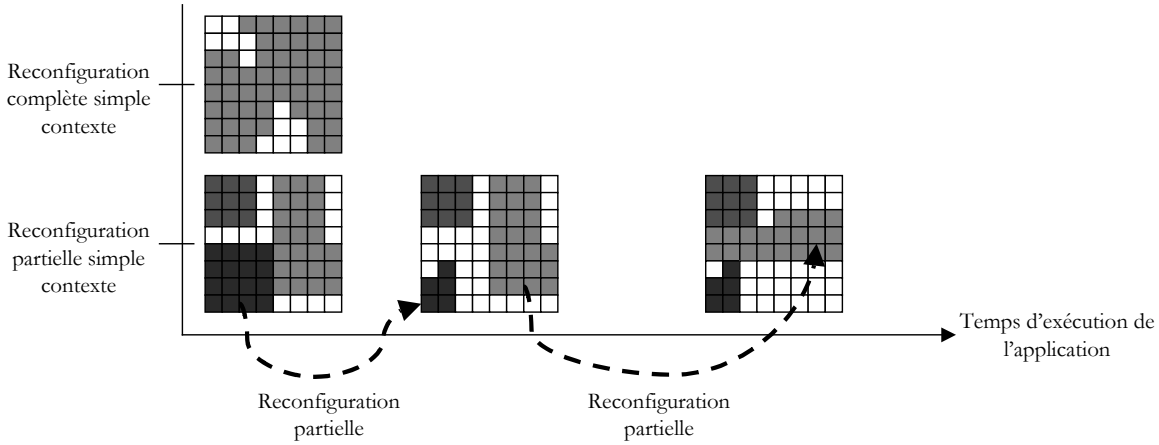


Figure 9 – La reconfiguration simple contexte, complète et partielle.

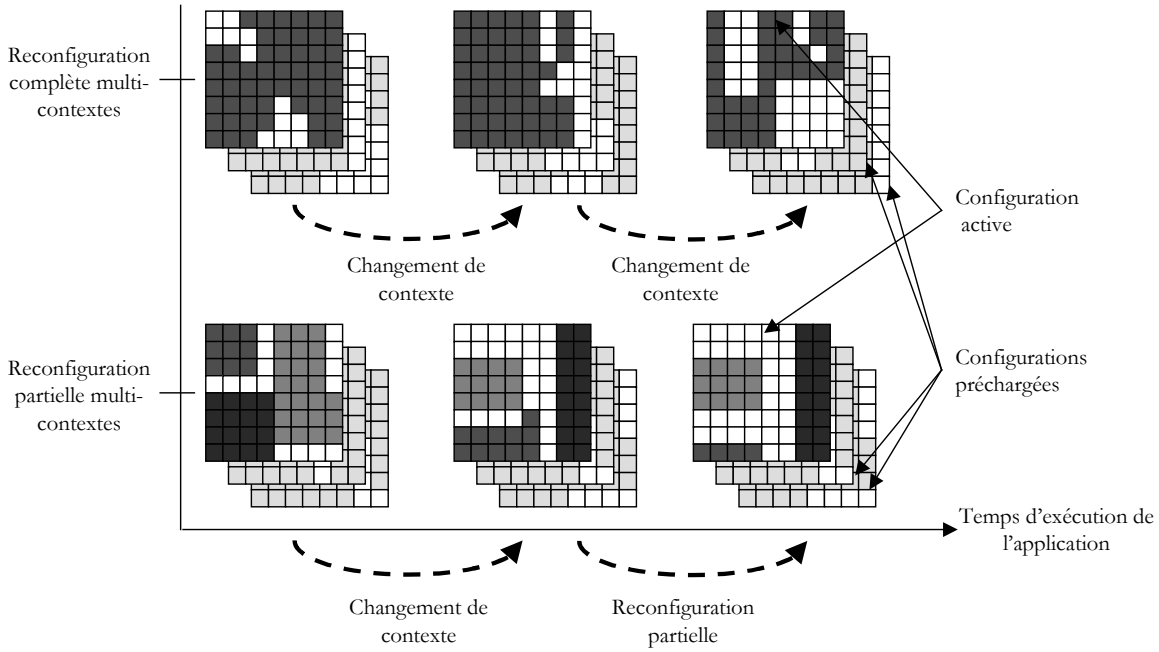


Figure 10– La reconfiguration multi-contextes, complète et partielle.

La figure 10 présente des reconfigurations multi-contextes. Pour l'exemple le nombre de contextes choisi dans la figure 10 est de trois, mais il peut être différent en fonction des technologies. Dans le cas de la reconfiguration multi-contextes et partielle, il est possible de modifier partiellement le contexte actif, et de gagner du temps lors de la modification des contextes préchargés en ne configurant que les éléments configurables qui changent d'états. Ainsi on peut constater sur la figure 10 que dans le cas de la reconfiguration partielle multi-contextes la première reconfiguration fait intervenir un changement de contexte alors que la deuxième reconfiguration fait intervenir une reconfiguration partielle du contexte actif.

Dans tous les cas la gestion de la configuration, en particulier avec la notion de reconfiguration dynamique, est complexe. Le plus souvent il faut ajouter un circuit auxiliaire pour la gérer. Ce circuit peut être un processeur externe ou interne à l'architecture. Il est possible d'utiliser les mémoires internes à l'architecture comme mémoires caches de configurations. C'est la solution proposée par Xilinx [Blodget03] dans son système d'auto-reconfiguration, basé sur l'utilisation d'un processeur embarqué de type MicroBlaze [Xilinx02]. [Ulmann04] présente une utilisation de l'auto-reconfiguration.

Pour conclure, et avant de donner une classification des architectures reconfigurables puis quelques exemples pertinents, le tableau 1 ci-dessous rappelle les principales caractéristiques des architectures reconfigurables présentées dans ce paragraphe.

Caractéristiques	**Notions fondamentales**
Granularité des ressources de traitement	Gros grain : ALU, opérateurs arithmétiques Grain fin : LUT
Granularité des ressources de communication	Gros grain : crossbar, bus Grain fin : canaux de fils, liaison point à point
Topologie du réseau de routage	Topologie matricielle, linéaire ou hiérarchique
Liaison avec le microprocesseur	Couplage périphérique, par bus ou direct
Reconfiguration	Simple ou multi-contextes, partielle ou non

Tableau 1 – Rappel des principales caractéristiques des architectures reconfigurables.

1.2 Taxonomie des circuits reconfigurables : exemples d'architectures

1.2.1 Classification des architectures reconfigurables

La classification que nous proposons se base d'abord sur la granularité des ressources de traitement. La séparation des architectures se fait suivant trois branches : grain fin (ressources de traitement de type LUT), gros grain (ressources de traitement de type ALU)

enfin multi-grains (mélangeant cœurs de processeurs, matrices d'ALU et/ou de LUT). Chacune des branches grain fin et gros grain se divise en fonction de la topologie du réseau de communication. Dans le cas d'architectures hétérogènes, multi-grains, il existe deux branches principales ; les architectures de type structure à tuiles, pour lesquelles chaque tuile embarque un élément reconfigurable ou non de granularité très différente (cœur de FPGA, cœur de processeur, IP) et les architectures pour lesquelles la partie reconfigurable agit comme un accélérateur matériel en couplage proche avec un processeur embarqué. Pour cette dernière catégorie nous effectuons une séparation en fonction de la granularité des parties reconfigurables (grain fin ou gros grain). Si nous effectuons cette séparation pour les architectures hétérogènes c'est parce que les deux structures (à tuiles et processeur plus accélérateur) sont bien différentes. La structure processeur plus accélérateur comporte une entité logicielle et une entité matérielle. La structure en tuiles peut comporter plusieurs tuiles logicielles et plusieurs tuiles matérielles, ce qui rend leur étude plus complexe.

La figure 11 présente la classification des architectures reconfigurables que nous proposons. Il est bien évidemment possible de classer d'une autre façon ces architectures [Hartenstein01], [Radumovic98], mais la classification que nous proposons nous semble justifiée par la ressemblance des architectures dans une même branche. Nous avons classé 32 architectures, mais dans le flot soutenu de publications concernant les architectures reconfigurables certaines ont pu nous échapper, ou nous ont paru peu avancées pour ce classement. Les architectures retenues sont ; Atmel AT 40K* [Atmel02], Lattice ispXPGA* [Lattice04], **Xilinx Virtex-II*** [Xilinx03b], **Altera Startix-II*** [Altera04] et Apex* [Altera01b], **Chameleon*** [Salefski01], Morphosys [Singh00], PACT XPP* [Baumgarte01], REMARC [Miyamori98], Systolix PulseDSP* [Systolix00], Altera Excalibur* [Altera02], Atmel FPSLIC* [Atmel03], CHIMAERA [Hauck97], FIPSOC* [Sidsa02], **Garp** [Hauser97], Pleiades [Abnous96], One Chip [Witting96], Trsicend E5* [Triscend01] et A7* [Triscend02], Xilinx Virtex-II Pro* [Xilinx04], **aSoC** [Laffely01], E-FPFA [Amerijckx98], CHESS* [Marshall99], **KressArray** [Kress96], MATRIX [Mirsky96], RAW [Waingold97], PipeRench [Goldstein99], **Systolic Ring** [Sassatelli01], RaPiD [Ebeling96], , ALFA [Verdoscia96], **DART** [David01], FPFA [Heysters00].

Une première constatation est que parmi les architectures commerciales (*) une majorité sont réalisées autour d'une matrice d'éléments reconfigurables à grain fin, couplée ou non à un processeur. Effectivement la granularité fine des éléments configurables permet l'utilisation d'outils fortement éprouvés dans le cadre des composants FPGA. Dans le cas de structures reconfigurables à gros grain les outils sont moins avancés et donc moins intéressants dans une démarche commerciale. Le milieu académique joue ici son rôle puisqu'il développe principalement des architectures à éléments configurables de gros grain. Ces architectures vont jouer un rôle important dans les développements futurs des applications de traitement du signal [Rabaey00], [Hartenstein03]. Bien sûr l'effort doit être mis également sur la réalisation d'outils permettant de tirer profits de ces architectures innovantes.

Dans les sous-sections suivantes nous allons présenter succinctement quelques architectures reconfigurables (en gras dans la liste ci-dessus) académiques ou commerciales qui sont représentatives du domaine, en partant d'architectures à gros grain et en allant vers des architectures à grain fin.

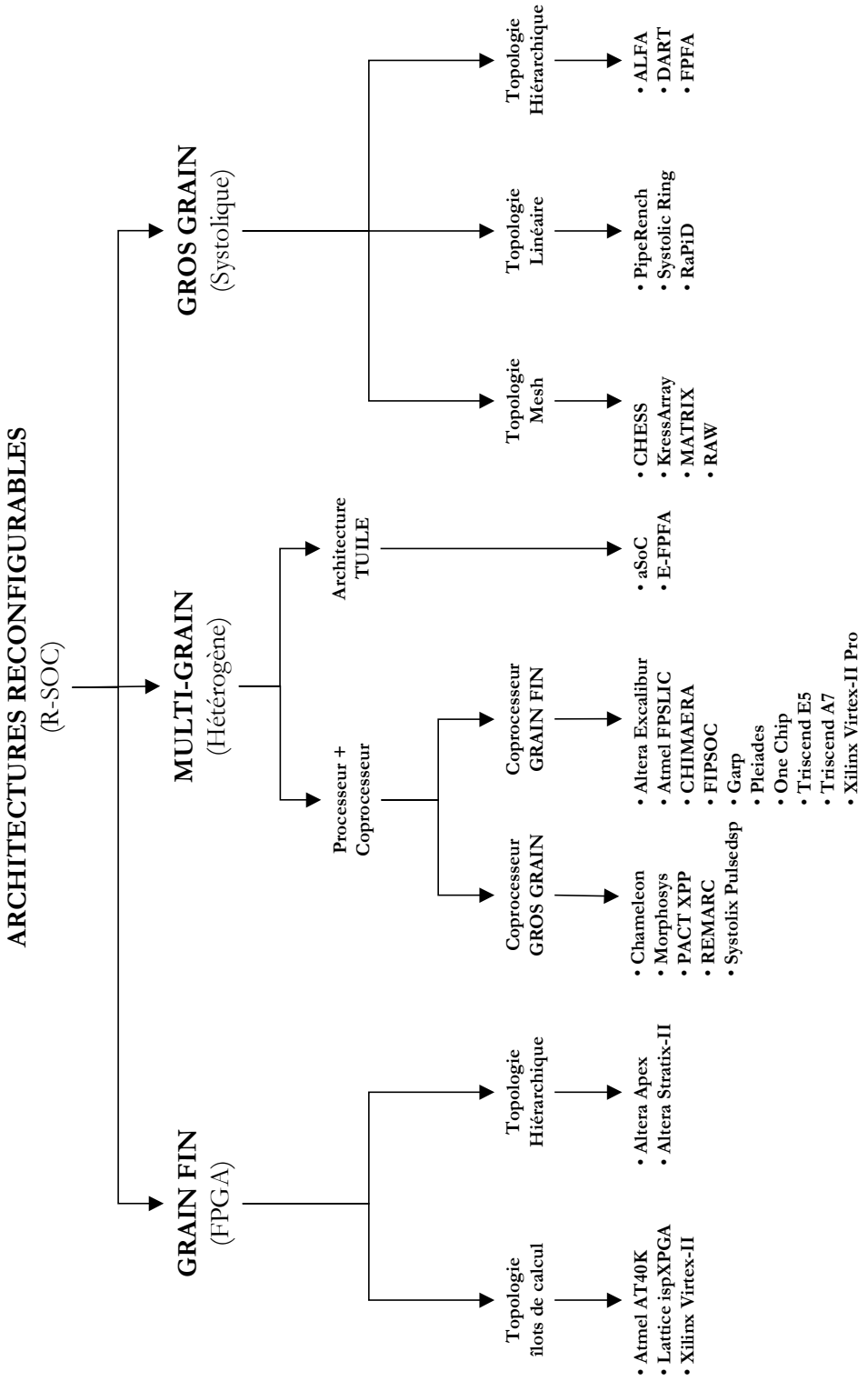


Figure 11 – Taxonomie des architectures reconfigurables

1.2.2 Architectures reconfigurables à gros grain

DART : architecture reconfigurable dynamiquement pour applications mobiles [David 03]

L'architecture DART est une architecture à grain épais dont la structure est hiérarchique. Elle est développée par le laboratoire LASTI de l'ENSSAT à Lannion. Au plus haut niveau, le niveau système, l'architecture DART est composée d'un contrôleur de tâches qui est chargé d'assigner aux clusters hiérarchiques les différents traitements devant être exécutés. Donc ce contrôleur gère la configuration des clusters. Une fois configurés les clusters sont autonomes. Sur la figure 12 qui montre cette architecture, apparaissent les mémoires nécessaires au système. Ces mémoires sont adressées au cours du temps par les clusters en fonction de leurs besoins en instructions et en données. Un module d'entrées/sorties permet à DART une connexion vers l'extérieur.

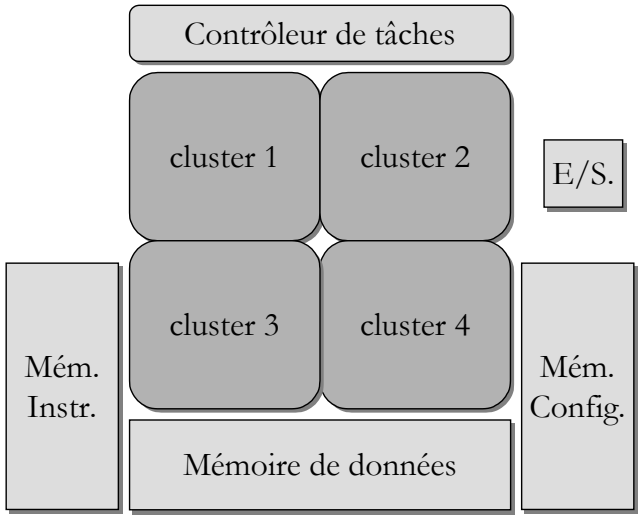


Figure 12 – Architecture système de DART.

Les clusters de cette architecture sont aussi hiérarchiques, ils se composent de 6 DPR (DataPath Reconfigurable) reliés par un réseau local de communications segmentées. A ce niveau de hiérarchie on retrouve, comme au niveau supérieur, une mémoire de configuration et une mémoire de données. Les DPR sont de nature gros grain, cependant bon nombre de chaînes de traitements nécessitent des traitements de grain fin, d'où l'idée d'adjoindre aux DPR une structure à grain fin FPGA. Pour le moment le cœur de FPGA n'est pas clairement identifié. Un contrôleur gère la configuration des clusters. Sur la figure 13, l'architecture d'un cluster de DART est schématisée.

Enfin les DPR sont le niveau le plus bas de la hiérarchie, leur architecture est essentiellement basée autour d'un réseau multi-bus comme on peut le voir sur la figure 14. Ce réseau permet une connectivité totale des éléments qui lui sont reliés, de plus, il est totalement flexible et permet ainsi des variations architecturales (nombre d'éléments connectés). Les unités fonctionnelles peuvent être de deux natures ;

multiplieurs/additionneurs ou ALU. Ces unités sont reconfigurables dynamiquement, elles permettent par ailleurs la réalisation de décalages en parallèle avec les traitements arithmétiques. Les générateurs d'adresses locaux (AG) gèrent les mémoires locales dans lesquelles sont stockées les données.

Figure 13 – Architecture d'un cluster de DART.

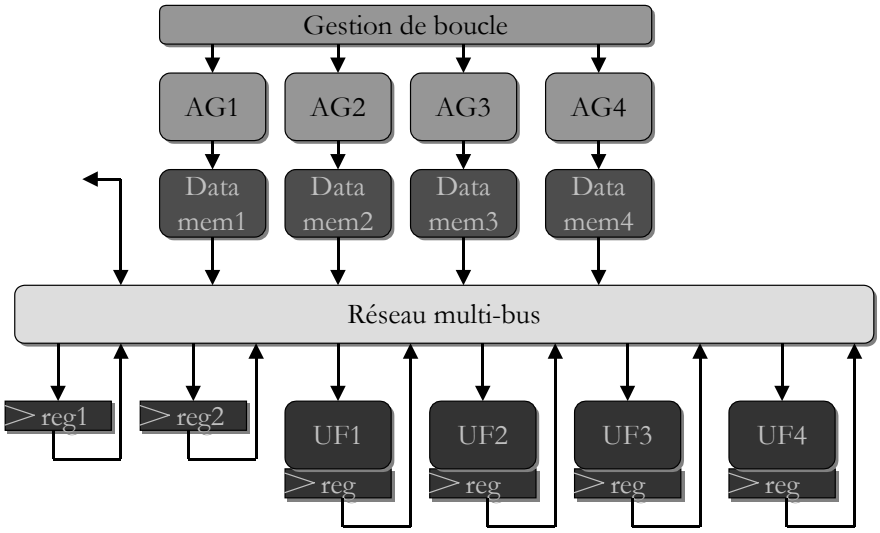


Figure 14 – Architecture des DPR.

Systolic Ring : architecture reconfigurable dynamiquement pour les systèmes sur puce [Sassatelli02a]

Cette architecture a été développée par le laboratoire LIRMM de l'Université de Montpellier particulièrement pour le domaine du traitement du signal et des images dans lequel les applications sont orientées flot de données. De ce fait cette architecture adopte au niveau le plus haut de hiérarchie une topologie de communication linéaire en anneau. Cette structure favorise de par sa nature le déroulement linéaire des calculs propres aux applications flot de données. La figure 15 montre l'architecture en anneau de la couche opérative du Systolic Ring, une structure de rebouclage pipelinée qui n'apparaît pas sur cette figure permet les calculs récursifs.

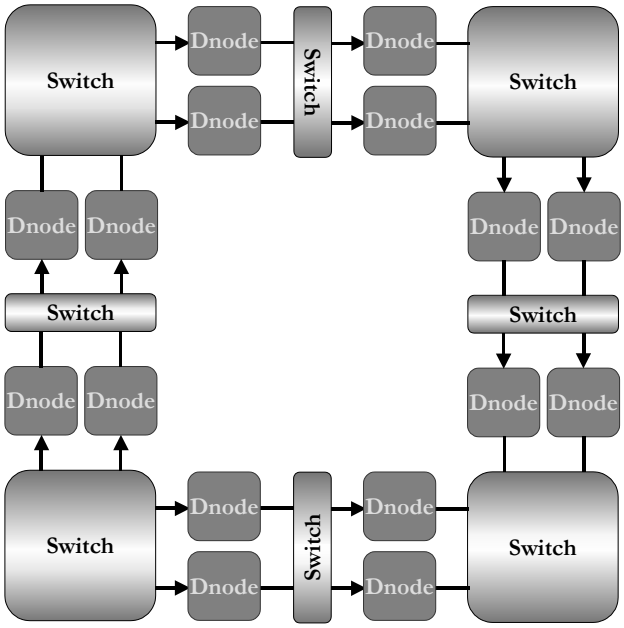


Figure 15 – Architecture de la couche opérative du Systolic Ring.

Au niveau hiérarchique inférieur, le Dnode est un composant reconfigurable à grain épais. Il traite des données sur 16 bits. C'est essentiellement un Data-Path : il est capable de réaliser toute une panoplie de fonctions arithmétiques et logiques grâce à une ALU câblée (notamment des opérations de type MAC par le biais d'un multiplieur/additionneur). Il possède deux entrées et une sortie, une banque de registres de 4 mots de 16 bits et un accumulateur de sortie. Il est capable de réaliser toutes les opérations possibles entre les entrées, la banque de registre et l'accumulateur. Sa fonctionnalité peut être modifiée à chaque cycle par un mot d'instruction codé sur 17 bits et contenu dans un séquenceur d'instructions. Celui ci est contenu en interne dans chaque Dnode comme on peut le voir sur la figure 16. La surface d'un Dnode est de 0.7 mm² en technologie 0,35 µm [Benoit02] et de 0.04 mm² en technologie 0,18µm [Sassastelli02b].

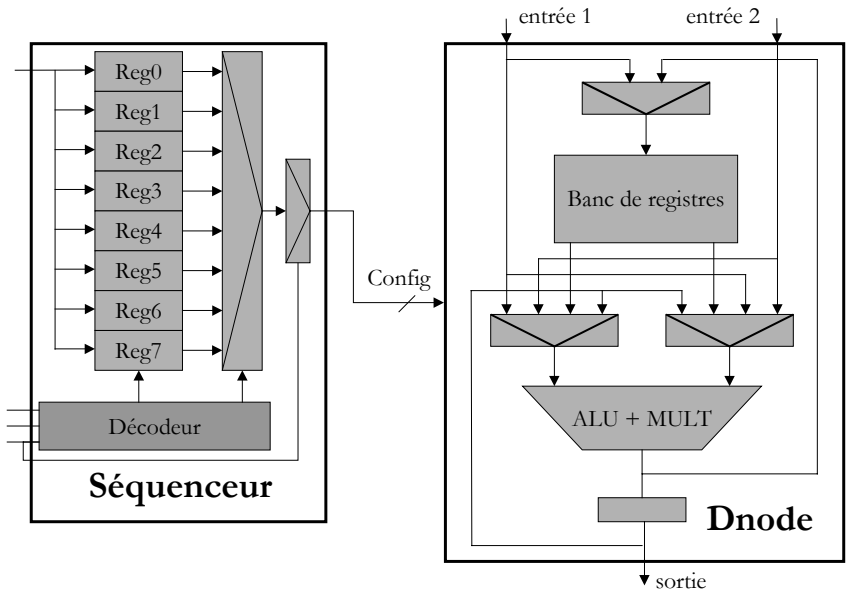


Figure 16 – Structure interne d'un Dnode.

L'architecture complète a été décrite en VHDL. Une version à 8 Dnodes traitant des données sur 16 bits a été complètement simulée, prototypée et synthétisée. La phase de prototypage utilise une carte Altera APEX [Sassastelli02b].

KressArray : Une ALU reconfigurable rapidement [Kress96]

Cette architecture développée par l'Université de Kaiserslautern en Allemagne fut d'abord introduite en 1995 sous le nom de rDPA (reconfigurable Data-Path Architecture) [Hartenstein95]. Elle est utilisée uniquement pour des applications du type flot de données. Bien qu'elle accepte la reconfiguration partielle dynamique, le flot de données n'est typiquement pas reconfiguré durant l'exécution de l'application.

La structure générale de l'architecture KressArray est donnée sur la figure 17, elle est constituée d'une matrice de ressources de traitement à gros grains les rDPU (reconfigurable Data-Path Units) et d'un contrôleur (rDPA control unit). Le contrôleur gère les entrées et les sorties des données vers la matrice de traitement ainsi que sa configuration. Le contrôleur joue le rôle d'interface entre le bus externe et la matrice de traitement, des buffers en entrées/sorties sont utilisés pour la synchronisation des données. Afin de transférer les bonnes données aux bons opérateurs, il est possible d'adresser chacun d'entre eux séparément grâce à une unité de génération d'adresses. Un ensemble de registres permet de stocker des résultats intermédiaires. Une unité de configuration gérée par le contrôleur permet l'envoi des instructions de configuration à chaque ressource de traitement.

Chaque rDPU contient une ALU et un chemin de données de 32 bits de largeur. Une étude de conception en technologie 0,18 µm CMOS à 4 niveaux de métallisation, donne une surface de 0,06 mm² [Hartenstein00].

L'architecture KressArray est proche de l'architecture reconfigurable CHESS développée par le laboratoire britannique de Hewlett Packard [Masharll99]. Cette architecture est matricielle, tout comme pour KressArray, les éléments reconfigurables sont des ALU de 4 bits reliées directement entre elles. De plus, CHESS intègre des mémoires en colonne, structure que l'on retrouve dans les FPGA Virtex Xilinx.

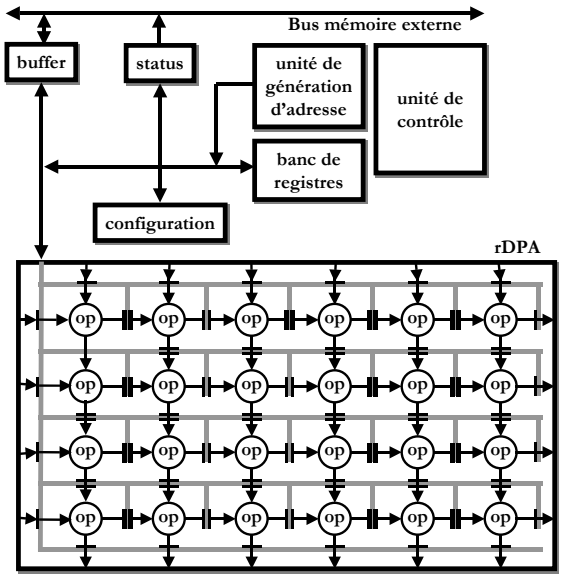


Figure 17 – Architecture KressArray.

1.2.3 Architectures reconfigurables multi-grains

aSoC : adaptive System on Chip [Laffely01] [Liang00]

L'aSoC est une architecture reconfigurable développée par le laboratoire VLSI Signal Processing Group (VLSI-SPG) de l'Université du Massachusetts aux USA. Cette architecture est hautement hétérogène puisqu'elle est constituée de tuiles qui peuvent être de granularités très différentes. Par exemple la figure 18 montre un aSoC avec des cœurs câblés pour des traitements spécifiques (estimation de mouvement par exemple) et des cœurs configurables du type grain fin FPGA. Cette architecture est hautement flexible puisque le concepteur peut définir les cœurs (IP) qu'il souhaite intégrer à son aSoC. Certaines tuiles peuvent donc être des tuiles reconfigurables de gros grain ou de grain fin. Cette caractéristique donne à l'aSoC une potentialité impressionnante.

Toute l'intelligence de cette architecture tient dans les interfaces de communications qui relient point à point les tuiles entre elles. Les communications sont établies avant l'exécution, puis elles sont traduites en instructions de connexion. Ces instructions

orientent un crossbar qui peut établir des liaisons entre les quatre tuiles du voisinage (au nord, au sud, à l'est et à l'ouest) direct de la tuile dont il fait partie. Mais il peut aussi créer une liaison entre une de ces tuiles de voisinage et le cœur avec lequel il peut communiquer. Un compteur programme permet de sélectionner la bonne instruction de communication. Puis un décodeur et un contrôleur assurent le déroulement de la programmation.

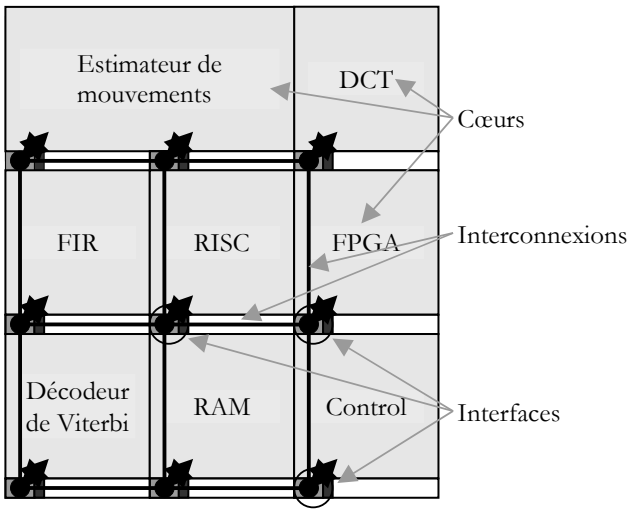


Figure 18 – Architecture multi-tuiles de l'aSoC

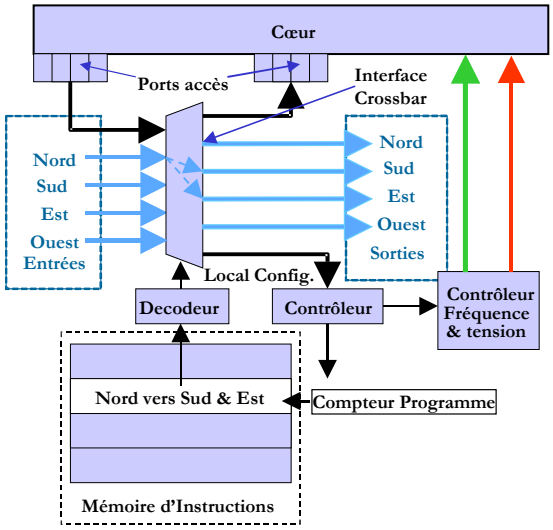


Figure 19 – Architecture de l'interface de communication d'une tuile

Comme indiqué sur la figure 19, le crossbar est au centre de l'interface de communication muni de la mémoire d'instructions. Un dispositif innovant lui est joint afin d'établir une

gestion dynamique de la consommation de puissance de chaque tuile. Effectivement il est possible pour chaque tuile de sélectionner une tension d'alimentation parmi quatre (1.8V, 1V, 0.72V et 0.6V) pour une réduction potentielle de la consommation de puissance de 0 à 90 %. De même il est possible de sélectionner une horloge parmi huit dont les valeurs de fréquence sont issues de la division par une puissance de 2 (de 2^0 à 2^7) de la fréquence de l'horloge globale [Laffely03a], [Laffely03b].

Cette architecture est particulièrement adaptée aux applications de traitement du signal, des applications de traitement d'images sont données dans [Burleson01a], [Burleson01b]. Un prototype de l'interface de communication a été réalisé en technologie 0,18 µm pour une fréquence de fonctionnement de 400 MHz [Liang02].

Chameleon : Architecture reconfigurable pour applications sans fils [Salefski01]

L'architecture reconfigurable Chameleon est commercialisée par la société américaine Chameleon Systems. C'est une architecture hétérogène comprenant un processeur ARC et un coprocesseur reconfigurable à gros grain comme indiqué sur la figure 20.

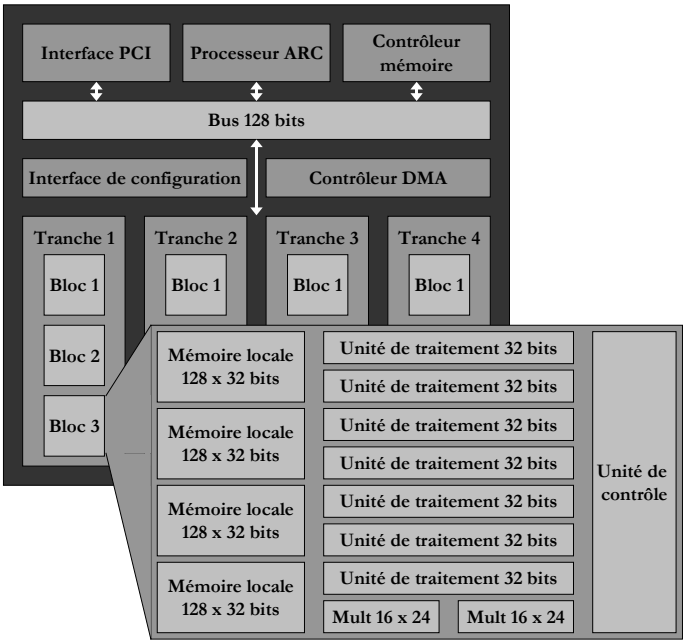


Figure 20 – Architecture de Chameleon

Le processeur ARC dispose d'une mémoire cache de 8 kilo octets pour les instructions et une autre de même taille pour les données. De plus, le processeur dispose d'un bus PCI de 32 bits associé à un contrôleur pour échanger avec l'extérieur en plus d'un bus mémoire de 64 bits relié aux mémoires externes (SDRAM, SRAM ou mémoires flash). La partie reconfigurable à gros grain est hiérarchique, elle se décompose en quatre tranches, elles mêmes décomposées en trois blocs. Chaque tranche est configurable individuellement.

Chaque bloc regroupe quatre mémoires locales, sept unités de traitement, deux multiplieurs et de la logique de contrôle (architecture proche des clusters de DART). Les unités de traitements décrites sur la figure 21 disposent d'un opérateur 32 bits multifonctions (32 bits ou deux fois 16 bits en parallèle) : addition, addition jusqu'à saturation, minimum, maximum, encodeur de priorité. Les opérandes sont sélectionnés par des multiplexeurs (24 vers 1) et passent ensuite à travers un opérateur de décalage et des masques.

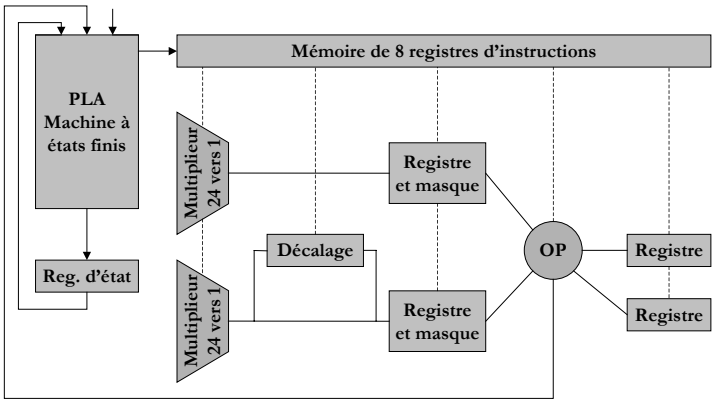


Figure 21 – Architecture des unités de traitement de Chameleon

Pour le transfert des données, il faut compter un délai d'une période d'horloge à l'intérieur d'une tranche et de deux périodes si l'on passe d'une tranche à une autre. Les résultats sont stockés dans les registres de sortie. La configuration de chacun de ces éléments est mémorisée dans un registre d'instruction. Huit instructions peuvent être préprogrammées pour chaque unité de traitement. La sélection de l'instruction courante est réalisée par la logique de contrôle.

Garp : Un processeur MIPS couplé à un coprocesseur reconfigurable [Hauser00]

L'architecture Garp développée par le groupe de recherche BRASS (Berkeley Reconfigurable Architectures, Systems and Software) de l'Université de Berkeley aux USA, est un exemple typique d'un système couplant un processeur classique (en l'occurrence un processeur MIPS-II) à une structure reconfigurable grain fin très proche d'un FPGA classique.

La vue système de Garp schématisée à la figure 22, couple étroitement le processeur, la structure reconfigurable, une mémoire cache d'instructions pour le processeur et une mémoire cache de données (qui sert de mémoire cache de configurations pour la structure reconfigurable). Une liaison externe par bus permet un accès rapide à la mémoire.

La partie reconfigurable de cette architecture est du type grain fin, les éléments de traitements sont réalisés avec des LUT à 4 entrées, une chaîne de propagation rapide de la retenue (afin de synthétiser des additionneurs rapides) et des registres de sortie assurant la synchronisation des données. Les éléments configurables sont reliés par des canaux de routage segmentés. Cette structure est la structure de base d'une écrasante majorité des

premiers FPGA (par exemple la famille 4000 de la société Xilinx). Ceci s'explique en partie par l'année de première publication de cette étude, 1997. Effectivement il s'agissait d'une des premières études dans le domaine des systèmes reconfigurables embarquant un cœur de processeur. A cette époque les structures reconfigurables évoluaient essentiellement autour des structures de FPGA. Cette étude peut être considérée comme un succès si l'on regarde aujourd'hui le nombre de FPGA commerciaux qui embarquent un processeur standard au plus près de la structure reconfigurable grain fin, [Xilinx04], [Altera02] et [Atmel03] par exemple.

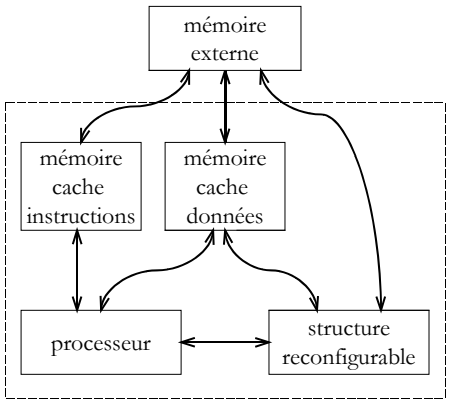


Figure 22 – Architecture système de Garp

Cette étude propose un système de reconfiguration partielle et dynamique que l'on retrouve aujourd'hui dans des FPGA commerciaux. La reconfiguration partielle se fait par colonne. Il faut 8 octets pour configurer un élément configurable, et 192 octets pour une colonne (il y a 24 éléments reconfigurables par colonne). La partie reconfigurable est composée de 32 colonnes, la taille du fichier de configuration totale est donc de 6144 octets. Le processeur gère la configuration du FPGA grâce à la mémoire cache. Il est possible au maximum de stocker quatre configurations totales dans cette mémoire cache, donc les configurations de 128 colonnes qui peuvent être utilisées à des moments différents au cours de l'exécution.

Dans [Hauser97] une comparaison de Garp avec un processeur SPARC montre sur des exemples choisis que Garp traite plus rapidement certaines applications avec une fréquence d'horloge plus faible.

1.2.4 Architectures reconfigurables à grain fin

Il est évident qu'il est difficile de parler d'architectures reconfigurables sans impliquer les FPGA dans la discussion. Leur concept a vu le jour dans les années 80, on ne peut pas passer sous silence les excellents travaux de Jonathan Rose et de son équipe de l'Université de Toronto sur le sujet [Betz99], mais depuis ces années il existe pléthore de travaux sur les architectures et les outils associés à ce type de circuits. Longtemps utilisés uniquement pour le prototypage d'ASIC ou pour réaliser quelques parties de logique combinatoire dans un système, les FPGAs ont tardé à trouver une véritable place dans le monde de l'électronique et ceci pour deux raisons. Tout d'abord ils ne bénéficiaient pas à leurs débuts de

technologies assez avancées pour leur assurer une intégration à la hauteur des besoins applicatifs. Mais aussi, les outils de placement-routage et de synthèse qui leur étaient dédiés n'étaient pas assez performants. Ceci s'explique par le fait que les FPGA ont à leurs débuts hérité des méthodologies de conception propres aux ASIC qui sont peu adaptées aux ressources de routage reconfigurables à grain fin (une des caractéristiques des FPGA). Les ressources de routage sont en effet la source principale de dégradation des performances d'une application entre une implémentation sur ASIC et sur FPGA. Aujourd'hui cela reste vrai et la contribution des ressources de routage s'accentue même à cause de la réduction des technologies [Dehon99].

La révolution technologique subie par les FPGA est sans aucun doute le fruit d'une bataille commerciale. Effectivement dans le lot des fabricants de circuits reconfigurables, deux fabricants nord américains, Xilinx et Altera ont rapidement conquis le marché, et sont devenus respectivement leader et challenger. Malgré tout, le marché des FPGA reste oligopole (c'est à dire qu'il existe plusieurs sociétés actives sur le marché) car de nombreuses sociétés proposent aujourd'hui des produits qui répondent à des besoins spécifiques. Mais il est clair que le marché des FPGA de technologie SRAM à forte capacité est clairement duopole (c'est à dire que deux sociétés au coude à coude se partagent une importante part de marché). Les deux sociétés Xilinx et Altera se sont lancées dans une bataille commerciale à grands coups d'évolutions technologiques (ce qui est la résultante caractéristique de ce type de marché). Certes ces évolutions sont d'abord dues à la recherche, mais leur mise en pratique tient de cette dualité mercantile.

Dans le cas des FPGA développés par les sociétés Xilinx et Altera, et dans la plupart des cas, l'élément configurable de base se compose d'une LUT à 4 entrées, d'une chaîne de propagation rapide de la retenue et d'un registre de sortie afin d'assurer la synchronisation des signaux comme on le voit sur la figure 23. Le nombre d'entrées des LUT n'est pas dû au hasard, des études montrent qu'il s'agit là d'un bon compromis entre les performances du circuit et les contraintes des algorithmes de placement-routage, leur complexité et leur efficacité [Amhed00], [Chow99a]. Ces éléments configurables peuvent être rassemblés en clusters hiérarchiques afin de favoriser une connectivité locale et rapide.

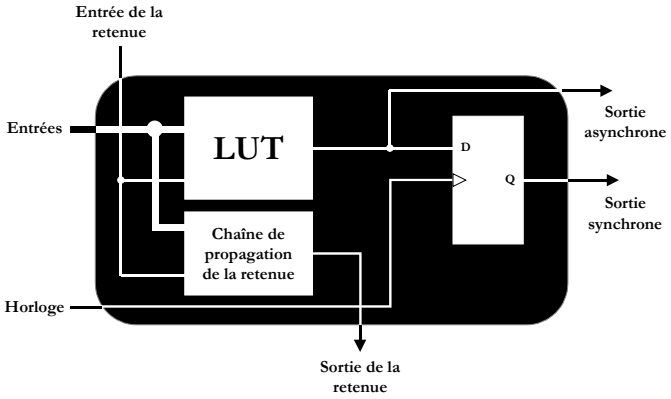


Figure 23 – Elément configurable de base des FPGA classiques.

La plupart des FPGA, étant donnée leur taille, ont besoin de réseaux de distribution de l'horloge spécialement adaptés à une transmission homogène des horloges sur toute l'architecture. Des dispositifs de régulation et d'asservissement des horloges sont implantés dans ces circuits. Les horloges sont traitées de façon bien particulière puisque des broches du circuit leurs sont réservées. Pour les autres entrées/sorties, les blocs dédiés à la communication avec l'environnement extérieur sont aussi configurables afin de s'adapter aux différents standards de communication de l'environnement du circuit. Les entrées/sorties sont aussi spécialement conçues pour permettre des débits importants.

Rapidement, afin de réaliser complètement des applications modernes, les FPGA ont dû se doter d'éléments de mémorisation configurables (apparition en 1999 dans les composants Virtex et Apex de Xilinx et Altera). Sans ceux-ci la mémoire synthétisée doit être distribuée sur les LUT, ce qui laisse peu de place pour les traitements. Ils sont donc indispensables, cependant leur taille sur le silicium est bien plus importante que celle des éléments configurables comme on le voit sur la figure 24, le placement-routage s'en trouve complexifié [Clifford00], [Wilton99]. Il est vrai que le routage reste l'inconvénient et l'avantage principal des FPGA. Grâce aux ressources de routage extrêmement flexibles, le FPGA est capable de réaliser n'importe quelles applications à condition qu'il dispose d'un nombre suffisant de ressources à allouer. Mais pour profiter pleinement de cette flexibilité il faut disposer d'outils qui soient en mesure de tenir compte des caractéristiques du routage (segmentation, population, longueur…) [deDinechin99]. Dans tous les cas, le routage configurable est pénalisant pour les performances du circuit par rapport à une réalisation à routage fixe telle que l'ASIC.

De plus, nombreuses sont les applications qui nécessitent la synthèse d'opérateurs du type multiplieur, additionneur et multiplieur/accumulateur. S'il est possible, grâce aux chaînes de propagation rapide de la retenue de réaliser sur un petit nombre de LUT des additionneurs efficaces, ce n'est pas le cas pour les multiplieurs très coûteux en ressources. Les industriels ont donc choisi d'implanter de façon matérielle des multiplieurs reconfigurables (la reconfiguration intervient en particulier sur la taille des données à traiter) au sein même de la matrice de grain fin. En positionnant ces multiplieurs près des colonnes d'éléments mémoires et d'éléments reconfigurables de grain fin, il est possible de synthétiser des opérateurs MAC. Cette solution fut retenue par Xilinx pour les composants Virtex-II [Xilinx03b]. Altera a choisi d'implanter dans les circuits de la famille Stratix [Altera04] des opérateurs câblés plus complexes pouvant directement être configurés en opérateur MAC.

Depuis le lancement des premiers FPGA Xilinx, leur architecture de type îlots de calculs agencés de façon matricielle n'a guère évoluée car elle est très performante associée aux outils de synthèse, placement et routage de Xilinx. Cependant la taille des éléments configurables de base a augmenté. Dans le cas des composants de dernière génération Virtex-II [Xilinx03b], ces éléments, les CLB (Configurable Logic Bloc) sont composés de quatre *slices* reliés par des connexions locales. Chaque *slice* contient deux éléments configurables élémentaires comme celui de la figure 23. De plus, des colonnes de multiplieurs 18 bits configurables et des colonnes de blocs de mémoires de 2 kilo octets configurables viennent renforcer les possibilités du composant. La figure 24 donne une vue schématique de l'architecture des composants Virtex-II de Xilinx (le nombre des éléments configurables, mémoires, multiplieurs, dépend du modèle choisi).

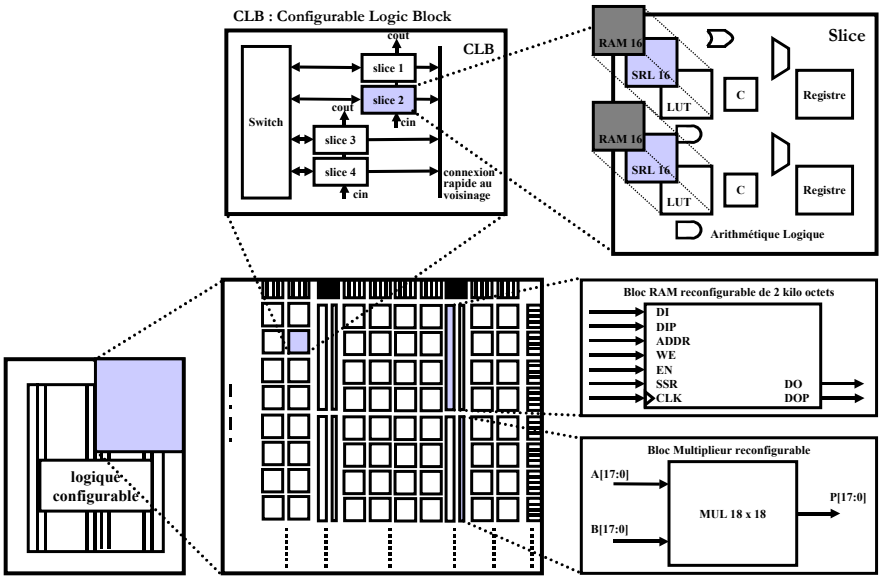


Figure 24 – Vue schématique de l'architecture des composants Virtex-II de la société Xilinx.

Altera a lancé au début de l'année 2004 un nouveau composant le Stratix-II [Altera04], ce composant est marqué par un certain nombre de changements par rapport aux architectures classiques des premiers FPGA Altera (Flex et Apex [Altera01b]) à trois niveaux de hiérarchie. Le Stratix-II comme le Stratix dont il a hérité de nombreuses caractéristiques, est moins hiérarchique et n'a plus que deux niveaux de hiérarchie. Le niveau le plus haut, figure 25, consiste en un ensemble d'éléments configurables LAB (Logic Array Bloc). Tout comme les CLB des FPGA Virtex, ils sont répartis en matrice. A ce même niveau, des mémoires de différentes tailles (72 octets, 576 octets et 81 kilo octets) sont réparties sur la matrice, ainsi que des blocs dits "blocs DSP" apparaissant sur la figure 26. Ces derniers permettent, entre autre, de réaliser des multiplieurs 36 bits ou des opérateurs MAC de 18 bits. Au niveau inférieur les LAB sont constitués de 8 ALM (Adaptive Logic Modules) et d'un réseau de connexions locales. Cette structure est donc très proche de celle du Virtex-II avec les CLB et les "slices". Cependant les ALM ont une architecture qui évolue par rapport aux LE (Logic Element) qui sont les briques reconfigurables élémentaires des anciennes générations d'architectures Altera. Un LE est très proche d'un élément configurable de base présenté figure 23. Les ALM, schématisés à la figure 27, sont eux réalisés autour d'un bloc de logique combinatoire à 8 entrées, de deux additionneurs et des registres de sortie. Le bloc combinatoire est en fait réalisé avec deux LUT à quatre entrées et de quatre LUT à trois entrées.

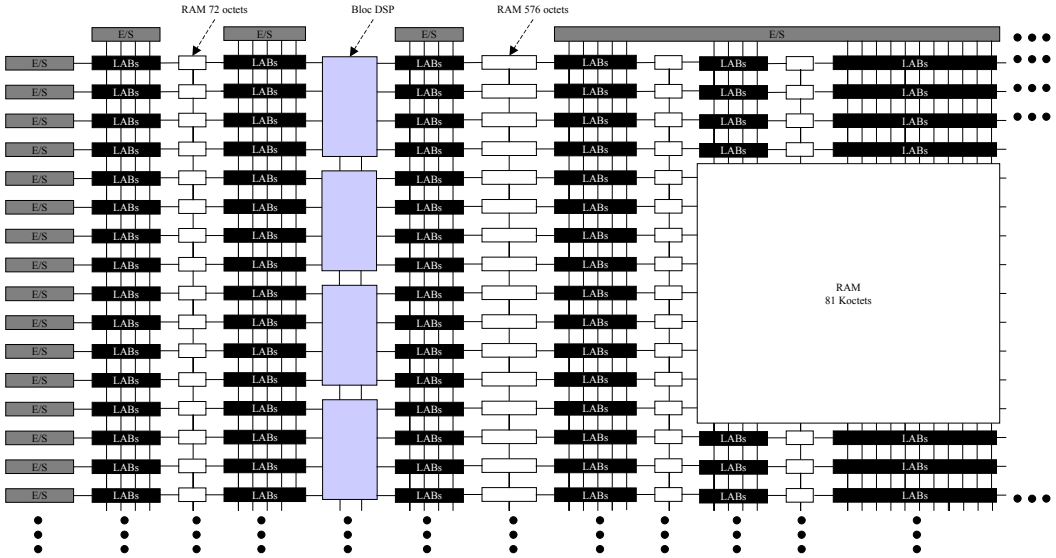


Figure 25 – Niveau supérieur de la hiérarchie de l'architecture du Stratix-II de la société Altera.

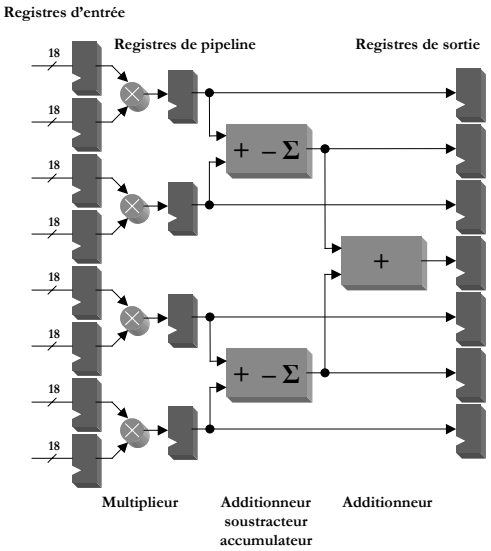


Figure 26 – Architecture d'un bloc DSP du Stratix-II.

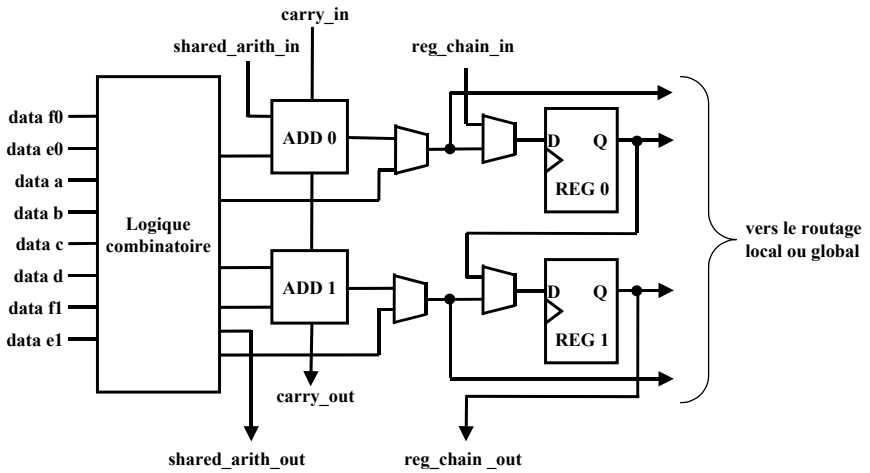


Figure 27 – Architecture d'un ALM au niveau inférieur de la hiérarchie de l'architecture du Stratix-II.

Concernant les types d'applications adaptées aux FPGA, ces derniers sont particulièrement efficaces pour traiter des opérations sur les bits, ce qui correspond à certaines applications de codage canal en télécommunication et aux applications de la cryptographie. Dans ce dernier cas la reconfigurabilité des FPGA leur permet de proposer des systèmes flexibles particulièrement adaptés aux évolutions des algorithmes et des architectures des cœurs de cryptage et décryptage. De nombreuses publications portent d'ailleurs sur l'implémentation de cœurs de cryptage sur des FPGA. Citons comme exemple dans le cadre d'AES (Advance Enrcyption System) [Dandalis00], [Elbirt00], [Gaj00] et [Weaver00]. Cependant les FPGA souffrent encore de failles de sécurité [Wollinger03] qu'il faut combler afin de proposer des solutions fiables pour les systèmes de sécurité [Bossuet04].

1.2.5 Comparatif par caractérisation des architectures présentées

Les architectures présentées précédemment sont très différentes, et c'est d'ailleurs la raison de leur choix pour cette présentation des architectures reconfigurables afin d'étudier un large panel d'architectures. Cependant il est possible d'effectuer un comparatif simple de ces architectures afin de résumer leurs caractéristiques principales. Pour ceci nous avons choisi une représentation par diagramme de caractérisation dans un plan. Chaque diagramme a cinq branches ; **GO** pour la granularité des opérateurs de la partie reconfigurable (hors microprocesseur), **GC** pour la granularité des ressources de communication, **NH** pour le niveau de hiérarchie, **DR** pour le dynamisme maximum possible pour la reconfiguration, enfin **LM** pour le lien avec un microprocesseur interne ou externe le cas échéant. La figure 28 donne le graphe de caractérisation des huit architectures présentées précédemment.

Deux architectures proches ont des surfaces internes à leur diagramme qui sont proches, par exemple DART et Systolic Ring, ou encore Virtex-II et Stratix-II. Nous pouvons

remarquer que les surfaces les plus grandes correspondent aux architectures de plus importante granularité, comme aSoC et Chameleon par exemple. Celles-ci ont bien entendu un réseau de communication en correspondance avec la granularité des traitements

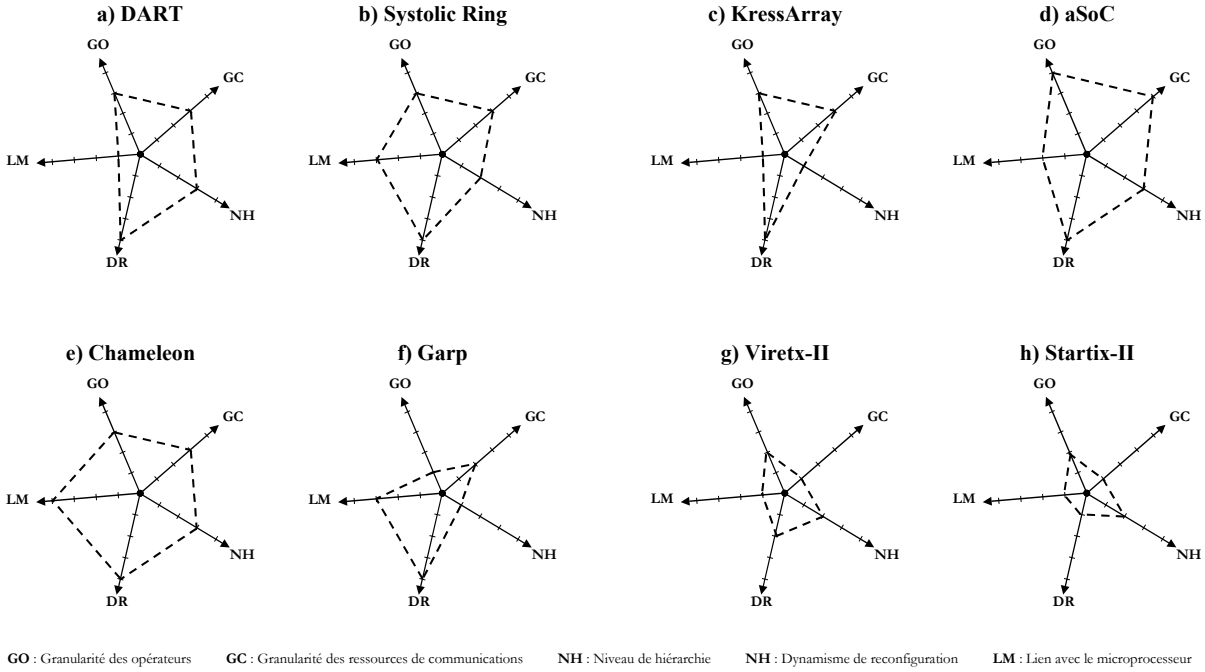


Figure 28 – Diagrammes de caractérisation des huit architectures présentées précédemment ; a) DART, b) Systolic Ring, c) KressArray, d) aSoC, e) Chameleon, f) Garp, g) Virtex-II et h) Stratix-II.

Une faible surface du graphe de caractérisation donne une architecture de type FPGA classique (Xilinx XC4000), une surface maximum nous conduirait à une architecture système flexible et fortement hétérogène (telle que l'aSoC).

Malgré une étude précise de toutes les caractéristiques de ces architectures il est difficile de les comparer objectivement sans prendre en compte les outils qui leur sont associés et qui sont indispensables à la réalisation d'applications avec ces architectures. Ces outils vont intervenir de façon significative dans les performances des applications, aussi même paré de la meilleure architecture, un circuit reconfigurable n'est rien sans outils performants associés [Hauck98].

Les architectures reconfigurables sont donc variées et nombreuses, leur domaine en pleine ébullition est riche de promesses. De nombreuses évolutions sont à venir car le domaine reste très ouvert. Le paragraphe suivant, sans être exhaustif, souhaite donner quelques pistes de futures évolutions.

1.3 Evolution du domaine

Comme nous l'avons indiqué dès le début de ce chapitre, le domaine des architectures reconfigurables est un domaine récent en pleine évolution. Pour s'en convaincre on peut comparer le nombre d'architectures commerciales et le nombre d'architectures au stade de la recherche. Dans la classification proposée au paragraphe 1.2 le pourcentage d'architectures commerciales est autour de 40 %, mais le classement ne prend pas en compte toutes les publications sur les architectures reconfigurables. Il s'agit en tout cas d'un domaine en phase exploratoire, en particulier pour les architectures gros-grain. Dans ce paragraphe nous souhaitons donner un certain nombre de pistes sur les possibles futures évolutions des architectures reconfigurables. Ces évolutions peuvent apparaître sur différents points : les architectures, les technologies, les outils, les applications et enfin les mentalités.

Les architectures sont très évolutives, car l'espace de conception sous-jacent est large, nous l'avons vu lors de la classification des architectures, en particulier en ce qui concerne la granularité des ressources de traitement (qui irrémédiablement entraîne une certaine granularité de communication). Une tendance toutefois semble s'imposer ; la vision, processeur embarqué auquel on couple un accélérateur reconfigurable, apparaît comme efficace et une réponse intéressante aux challenges des SoC dans un avenir proche par le biais du concept de R-SoC, ou SoC reconfigurable [Hauck98]. Toutes les architectures commerciales à gros grain sont basées sur ce type de structure. De plus, il est à noter l'essor des circuits de type FPGA qui embarquent un ou plusieurs cœurs de microprocesseur au plus près de la matrice reconfigurable (Xilinx Virtex-II Pro [Xilinx04], Altera Excalibur [Altera02] ou encore Atmel FPSIC [Atmel03] par exemple). Une seconde tendance dans le cadre des applications de traitement du signal qui nécessitent un nombre important d'opérateurs arithmétiques (additionneur, multiplieur et MAC principalement), est de proposer dans la matrice reconfigurable un nombre important de ces opérateurs câblés et reconfigurables. Cependant, ces mêmes applications ont besoin d'opérateurs logiques en particulier pour effectuer les opérations de contrôle (opérations logiques de base et comparateurs). Les futures architectures devront donc embarquer probablement un processeur (qui pourra gérer la configuration de la partie reconfigurable), accompagné de mémoires caches, d'un bus spécial pour connecter les parties matérielles et logicielles et d'une matrice reconfigurable à deux granularités d'opérateurs : des opérateurs arithmétiques de gros grain et des opérateurs logiques de grain fin. Enfin un ensemble de mémoires distribuées sur la matrice reconfigurable matérielle est indispensable pour le stockage des données nécessaires aux calculs et pour éviter les accès aux entrées/sorties (très consommatrice d'énergie puisque fonctionnant généralement à des tensions plus élevées que le cœur). En fonction des applications, des cœurs de fonctions spéciales (codage canal, compression, gestionnaire de bus …) devront pouvoir être embarqués matériellement dans le R-SoC. Un très bon exemple est proposé par la société ST Microelectronics [Cambonie03].

Les architectures vont clairement devoir également évoluer vers une plus grande adaptation aux applications. De ce fait les architectures reconfigurables vont certainement être de plus en plus dédiées à un domaine d'application. Elles seront alors, en comparaison avec les

ASIC, des ASRA pour "Application Specific Reconfigurable Architecture" ou ASPP "Application Specific Programmable Product" [Hartenstein01]. Citons par exemple le cas de la société Xilinx qui propose le concept d'ASMBL pour "Application Specific Modular Block Architecture" [Xilinx03a]. Le but de ce concept est de proposer aux concepteurs de choisir le contenu de l'architecture par colonne. Ils pourront par exemple choisir des colonnes d'éléments logiques, de DSP élémentaires, de mémoires, d'IP matériel, d'entrées/sorties et d'autres futures propositions. Cette démarche n'est pas surprenante et répond à des impératifs de rentabilité économique : les sociétés qui développent des applications souhaitent acheter des composants qui ne répondent qu'à leur besoin. Il est intéressant de se rappeler que ce fut les mêmes soucis qui ont imposé la réduction/spécialisation du jeu d'instruction des processeurs généralistes comme dans le cas des DSP. La même démarche devrait bientôt apparaître pour les architectures reconfigurables.

La technologie adaptée aux architectures reconfigurables est particulièrement importante. Effectivement l'évolution des architectures reconfigurables est contrainte principalement par deux problèmes technologiques, les cellules mémoires dans lesquelles est stockée la configuration et le support des communications (routage). En ce qui concerne le premier point, la configuration des parties opératives est le plus souvent due à la programmation d'une mémoire support de cette configuration. Cette solution technique entraîne entre autre une importante consommation statique d'énergie et une perte importante de place et de densité par rapport à une solution statique telle que l'ASIC. De plus, les délais dus à la traversée de points de mémorisation sont longs et détériorent les performances en particulier le long des lignes de routage. D'où la nécessité d'améliorer le point de mémorisation. De nombreuses études portent aujourd'hui sur ce sujet, des technologies nouvelles apparaissent et promettent des évolutions importantes. Parmi celles potentiellement intéressantes pour le domaine des architectures reconfigurables il est possible de citer les FRAM (Ferroelectric RAM) et les MRAM (Magnetoresistive RAM) [Tredennick03].

Les FRAM ont été introduites en 1988 (première commercialisation en 1992) par Ramtron international Corp. [Ramtron]. Avec cette technologie la capacité de mémorisation des DRAM est remplacée par un matériau cristallin ferroélectrique, généralement un alliage PZT, Plomb-Zirconium-Titane. Les atomes dans le matériau ferroélectrique occupent un ou deux états stables (qui représentent les niveaux haut et bas) et peuvent changer d'état en fonction de l'application d'un champ électrique. Ces nouvelles mémoires ont des accès rapides particulièrement en écriture, de plus leur très faible consommation de courant leur donne une faible consommation de puissance. Néanmoins ces mémoires nécessitent des tensions de programmation élevées. Comme pour les DRAM, chaque lecture demande une réécriture de la donnée pour la conserver, d'où un temps de lecture plus long que pour l'écriture. Cette technologie est déjà bien avancée et de nombreux circuits sont commercialisés.

Les MRAMS semble la technologie la plus prometteuse. Dans ce cas la capacité des DRAM est remplacée par un aimant en matériau ferromagnétique. La direction du champ magnétique peut occuper une ou deux positions stables. Cette direction change en fonction de l'application d'un champ magnétique. L'intérêt principal de ces mémoires est qu'elles

conservent les informations mémorisées quasi indéfiniment en l'absence de source d'énergie contrairement aux mémoires vives actuelles. Cependant le champ magnétique à appliquer doit être important afin de changer le moment magnétique du point de mémorisation. De ce fait la consommation de puissance de ces mémoires est élevée et elles doivent être capable de dissiper des quantités importantes de chaleur. Ces mémoires ont toutefois un intérêt supplémentaire car elles résistent de par leur nature magnétique aux rayonnements, ce qui leur donne une place de choix pour l'intégration dans des milieux hostiles comme le spatial.

Concernant les lignes de communication, une évolution majeure a récemment fait évoluer la technologie des pistes de routage de l'aluminium vers le cuivre permettant une accélération par trois de la vitesse des signaux sur les lignes. Mais il est indispensable de remplacer les interrupteurs actuels à base de transistors trop consommateurs d'énergie et dont les capacités internes freinent la propagation des signaux. Les évolutions technologiques à venir sont variées mais à longs termes, citons par exemple l'utilisation des nanotubes de carbone et les interrupteurs moléculaires [Butts02]. Cependant les nanotubes de carbone n'ont été découverts que depuis 1991 (grâce aux microscopes à effet tunnel) et leur manipulation est loin d'être maîtrisée. Mais leur très faible résistivité, poids et encombrement pourraient bien faire de leur maîtrise une révolution technologique. De même les interrupteurs moléculaires, ou transistors moléculaires sont très prometteurs bien qu'aujourd'hui les physiciens ne soient capables que d'en créer un à la fois pour une durée courte avec la mise en œuvre de matériel extrêmement perfectionné (microscope à force atomique). Les évolutions technologiques sont prometteuses mais elles se feront par saut en fonction des avancées dans tous les domaines scientifiques.

Les outils sont un point faible des architectures reconfigurables [Hauck98]. Effectivement la flexibilité et le grand parallélisme matériel de ces structures rendent complexes les algorithmes de synthèse et de placement-routage indispensables à la mise en œuvre de tels circuits. De plus, de nombreuses architectures académiques sont développées sans proposer de flot de conception complet et automatique. Il est possible de donner quelques pistes concernant les futures évolutions. D'abord le besoin en outils génériques capables de rapidement s'adapter à des architectures distinctes, afin d'élargir le domaine de conception d'une application avec des architectures reconfigurables [Lagadec01]. Cependant, même générique les outils devront se focaliser sur un type architectural (grain fin ou gros grain par exemple). Pour faciliter le choix des architectures et augmenter l'adéquation entre les applications et ces architectures, il semble important de développer des outils d'aide à la conception qui travaillent à un haut niveau d'abstraction. Grâce à eux il est possible de dégager, très tôt dans le flot de conception, des architectures potentiellement intéressantes pour la ou les applications à développer [Bossuet03b]. De par les évolutions des architectures et le regroupement au sein de la même puce de parties matérielle et logicielle étroitement couplées, il est indispensable de poursuivre le développement d'outil de conception conjointe logicielle/matérielle (CoDesign). Le dernier point reste la maîtrise plus importante par les outils de la reconfiguration partielle et dynamique, afin de profiter de ces nouvelles caractéristiques de flexibilité qu'apportent les architectures reconfigurables. Ainsi il sera possible d'ouvrir de nouveaux champs applicatifs réservés aux architectures

reconfigurables comme le montrent les travaux autour de l'architecture reconfigurable dynamiquement Ardoise [Kessal00].

Les applications doivent être adaptées aux architectures reconfigurables afin de tirer parti de leurs caractéristiques, en particulier en ce qui concerne la reconfiguration dynamique et sa mise en œuvre. Les applications devront donc dès le début du flot de conception intégrer la notion de reconfiguration. La reconfiguration peut en particulier être mise en œuvre de façon bénéfique dans les systèmes évolutifs afin de remplacer la mise à jour logicielle traditionnelle par des mises à jour matérielles. Cette solution est exploitée dans le cadre particulier de la radio logicielle [Delahaye04]. D'autres domaines sont à même de tirer partie de la mise à jour matérielle qu'offrent les architectures reconfigurables, en fait tous les domaines qui nécessitent des mises à jour fréquentes. Le domaine de la sécurité des systèmes est un bon exemple, il nécessite une adaptation constante aux nouvelles attaques par du matériel sécurisé [Lockwood03]. En développant des techniques de conception sécurisée qui tiennent compte du potentiel de reconfiguration dynamique il est possible d'augmenter la sécurité globale du système [Bossuet04].

Les mentalités jouent un rôle important dans l'évolution des nouveaux domaines. Celui des architectures reconfigurables fait face à deux domaines bien implantés, les systèmes programmables (microprocesseur, DSP …) et les systèmes spécifiques (ASIC). Les applications comme les outils sont le plus souvent hérités de ces deux derniers domaines. Les architectures reconfigurables entraînent, nous l'avons vu, une nouvelle façon de penser les applications et les outils, elles doivent donc être l'objet de formations particulières [Hartenstein03]. Effectivement l'enseignement de l'électronique numérique est largement occupé par le domaine des systèmes programmables. Classiquement à l'université dans la filière génie électrique et informatique industrielle (GEII) l'enseignement orienté vers les composants reconfigurables du type FPGA représente deux à trois fois moins en volume horaire par rapport à l'enseignement orienté vers les composants programmables. Avant le troisième cycle quasiment aucune heure n'est dédiée aux architectures reconfigurables au sens large. Ainsi la notion de reconfiguration est vue bien longtemps après celle de programmation (ce qui entraîne souvent les étudiants à considérer un langage de description matérielle tel que le VHDL comme un langage de programmation conduisant à des difficultés d'apprentissage et de mise en œuvre). Pour faire évoluer le domaine des architectures reconfigurables ainsi que les méthodes de conception qui lui sont dédiées, il est indispensable de revoir l'enseignement de l'électronique numérique afin de préparer les futurs concepteurs à la maîtrise de ces nouvelles architectures [Tredennick03]. Le monde scientifique a bien compris la portée de ce nouveau domaine, aussi toutes les conférences internationales majeures du domaine de la conception en électronique numérique ont depuis plusieurs années des sessions portant exclusivement sur les architectures reconfigurables et/ou les outils et méthodes qui leurs sont associés.

1.4 Conclusion

Le récent domaine des architectures reconfigurables est en pleine extension. Leur large domaine de conception se délimite plus ou moins difficilement des autres domaines et en

particulier avec le domaine des systèmes programmables (particulièrement pour les architectures reconfigurables à gros grain). Cependant il est possible de caractériser finement les architectures reconfigurables par le biais de la granularité de leurs ressources de traitement, de mémorisation et de communication, ainsi que par leur topologie de routage et leur lien avec un processeur interne ou externe le cas échéant. Enfin les mécanismes de reconfiguration permettent de caractériser le dynamisme de l'architecture. Muni de cette caractérisation, il est possible de classer et de comparer des architectures entre elles. Néanmoins la comparaison ne prend en compte que des vues architecturales sans considérer une application particulière. Sachant que les outils qui vont être utilisés pour synthétiser et implanter l'application dégradent généralement les performances de celle-ci, la comparaison des architectures seules n'est pas totalement en correspondance avec la comparaison des performances d'une application sur ces architectures. La comparaison post-implémentation englobe en effet celle des outils et celle des architectures. Il faut toutefois un moyen pour comparer entre elles les architectures reconfigurables en tenant compte des applications et en s'affranchissant de l'impact des outils sur les performances des architectures, afin de caractériser celles en adéquation avec les applications à développer. Pour cela il faut définir des méthodes d'exploration de l'espace de conception qui interviennent à différents niveaux du flot de conception. C'est ce que nous présenterons dans les chapitres à venir.

Ce chapitre a montré le potentiel des architectures reconfigurables, et les possibilités de développement du domaine. Les évolutions à venir sont nombreuses et vont permettre de développer de nouvelles méthodes de conception intégrant très tôt dans les flots les caractéristiques particulières des architectures reconfigurables telle que la notion de reconfiguration dynamique. Ainsi de nouvelles applications dédiées à ce type de technologie verront bientôt le jour.

Chapitre 2

L'exploration de l'espace de conception

Le chapitre premier de ce document a montré que l'espace de conception délimité par les architectures reconfigurables est très large. La classification et la caractérisation des architectures reconfigurables donnent une vision du nombre important de choix à effectuer avant de sélectionner une architecture pour la ou les applications à développer. Le but du développeur d'applications est de viser l'architecture la plus en adéquation avec son développement. Or cela devient de plus en plus difficile, voir même impossible d'effectuer la recherche d'adéquation à la main. Le développeur a donc besoin de méthodes et d'outils qui lui permettent à chaque niveau d'abstraction (du niveau système au niveau RTL*) d'explorer l'espace de conception rapidement.*

L'exploration de l'espace de conception se décline pour tous les systèmes électroniques, de la vision spécifique à la vision programmable en passant bien sûr par la vision reconfigurable. Ce chapitre se veut dans un premier temps un état de l'art des méthodes d'exploration de l'espace de conception dans le cadre large des systèmes de l'électronique numérique. Puis dans un deuxième temps quelques exemples particuliers aux architectures reconfigurables permettent de mieux saisir les challenges de l'exploration dans ce cas. Enfin ce chapitre présente la vision du LESTER *à travers l'outil d'aide à la conception des systèmes hétérogènes sur puce, Design Trotter.*

2.1 Introduction et définitions

2.1.1 Nécessité de l'exploration

Comme le premier chapitre l'a montré, il est possible de décomposer les systèmes de l'électronique numérique en trois grandes catégories ; les systèmes spécifiques, les systèmes reconfigurables et les systèmes programmables. Dans les trois cas lors du développement d'un système les choix se proposant aux concepteurs sont nombreux. Ces choix définissent un espace appelé espace de conception. Avec des systèmes dont la réalisation met en œuvre des millions de portes logiques, il devient de plus en plus complexe voir impossible aux concepteurs de définir le système approprié d'une façon manuelle ou intuitive. L'hétérogénéité de plus en plus présente rend complexe la tâche des concepteurs, en particulier lorsque des parties logicielles et matérielles sont regroupées sur la même cible entraînant une conception conjointe. De plus, d'un point de vue industriel la réduction des temps de conception et de mise sur le marché (*time to market*) des produits, rend indispensable le développement d'outils d'aide à la conception qui permettent de rapidement cibler les bons systèmes sur lesquels les applications seront développées.

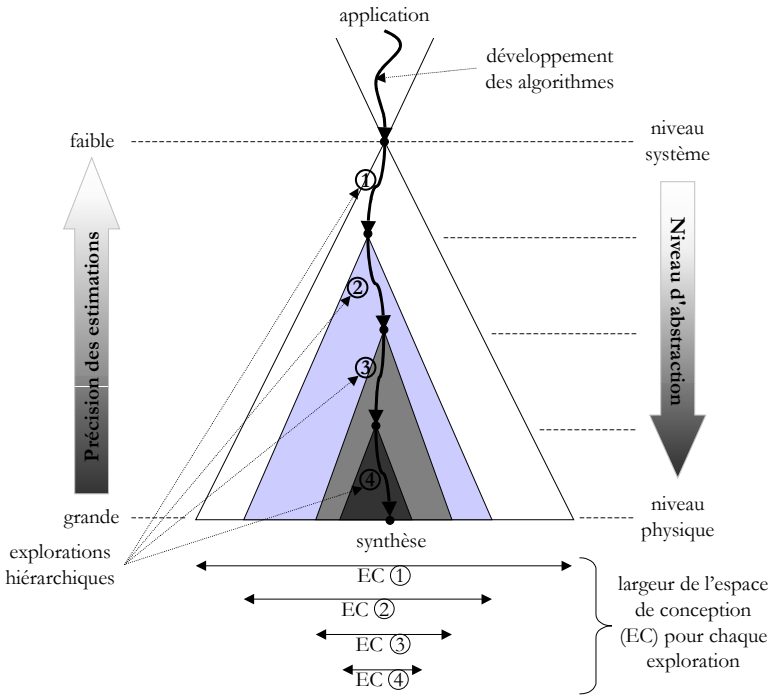


Figure 29 – Pyramide des niveaux d'abstraction du flot de conception et positionnement de l'exploration hiérarchique de l'espace de conception.

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. Pour être performante l'exploration doit être rapide et efficace. Pour cela elle peut intervenir à tous les niveaux d'abstraction du flot de conception comme on peut le voir sur la figure 29 [Pimentel01]. Ainsi l'espace à explorer peut être raffiné à chaque niveau, l'exploration est donc un processus itératif et hiérarchique. La pyramide de la figure 29 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 l'espace de conception jusqu'à converger vers une architecture cible. L'exploration a nécessairement besoin de mesures et/ou d'estimations pour comparer les diverses solutions offertes dans l'espace exploré. Les estimateurs sont peu précis au niveau système, du fait même de l'abstraction mais ils sont rapides, ce qui permet d'explorer de larges espaces de conception. A contrario au niveau physique les estimateurs sont précis mais leurs mises en œuvre nécessitent du temps, c'est pourquoi il faut avoir, en amont, réduit considérablement l'espace de conception. En outre, plus le niveau d'abstraction est bas plus les estimateurs sont dédiés à un type de solution, les outils sont de moins en moins génériques. La précision est en correspondance avec la spécialisation des outils. Le nombre des niveaux d'exploration n'est pas fixe et dépend du type de système visé. Une exploration complète depuis un haut niveau d'abstraction jusqu'au niveau physique demande toujours la mise en œuvre de plusieurs outils.

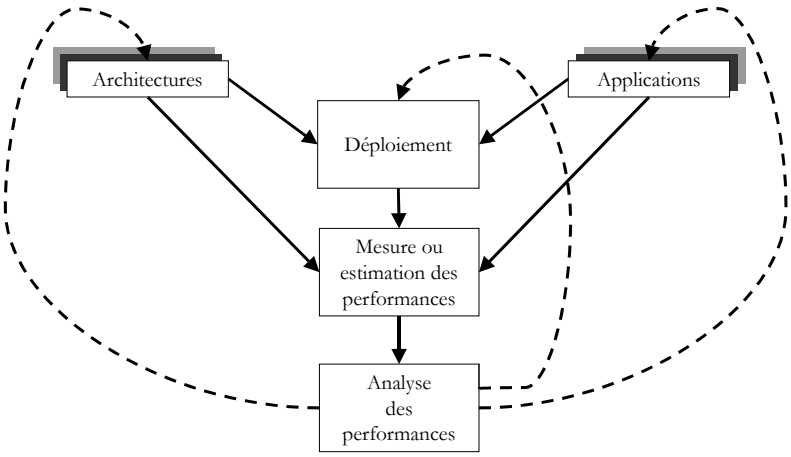


Figure 30 – Diagramme en Y, flot d'exploration de l'espace de conception.

Le flot général de l'exploration de l'espace de conception est appelé diagramme en Y [Kienhuis97] schématisé figure 30. Ce diagramme montre que les applications et les architectures sont spécifiées séparément. Il montre bien que l'exploration peut porter sur l'architecture et/ou sur l'application. Il peut y avoir plusieurs itérations dans le flot, il est possible de revenir sur le déploiement de l'application sur l'architecture (choix du partitionnement par exemple), sur l'application (choix du déroulage de boucle ou choix d'un ordonnancement par exemple) ou sur l'architecture (choix des caractéristiques architecturales). Les itérations peuvent faire partie d'un processus automatique ou alors

elles peuvent être le fait du concepteur qui s'enrichit des premières explorations pour faire converger les suivantes vers une solution adéquate au problème. C'est à dire une solution qui dans l'espace du problème parvient à remplir les objectifs qui définissent à leur tour un espace. Le paragraphe 2.1.2 définit plus précisément ces deux espaces que sont celui du problème et celui des objectifs.

La figure 30 montre la nécessité d'avoir une mesure ou une estimation des performances de l'application sur l'architecture, nous pouvons parler alors d'extraction des performances. Dans le cas des architectures matérielles, il existe trois grandes méthodes pour l'extraction des performances. Ces méthodes, après analyse, permettent trois méthodes d'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. Or il n'existe pas forcément d'outils de synthèse pour toutes les architectures que le concepteur souhaiterait comparer. Donc il 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 [Lagadec01] et [Betz97], mais même 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 [Bossuet03b]. 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 donc 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.

Compiler, estimer puis comparer : Il s'agit dans ce cas d'une exploration sur l'application via l'architecture logique qui est générée après la compilation du code de

l'application [So03], [Kulkarni02]. Cette architecture logique peut être estimée sur l'architecture matérielle cible. Ainsi il est possible d'exploiter les paramètres de l'application tels que le déroulage de boucle ou le degré de parallélisme. Cette méthode peut être combinée avec les deux précédentes afin d'explorer simultanément les caractéristiques de l'application et celles de l'architecture matérielle cible. Cependant le processus d'exploration devient rapidement complexe et les stratégies d'exploration doivent être bien étudiées préalablement. La méthode basée sur la compilation est issue du domaine des architectures logicielles. Dans le cas des architectures matérielles elle est surtout utilisée pour les applications de traitement numérique intensif intégrant des boucles imbriquées.

Historiquement la méthode d'exploration *synthétiser puis comparer* fut la première utilisée car la complexité des systèmes numériques ne nécessitait pas d'algorithmes trop coûteux en temps de synthèse. Cependant avec l'accroissement rapide de cette complexité, la méthode *estimer puis comparer* permet de travailler à plus haut niveau et d'obtenir plus rapidement des solutions. Enfin, depuis peu, la méthode *compiler, estimer puis comparer* a hérité de l'exploration des architectures logicielles, pour s'appliquer aux architectures matérielles. Cette dernière méthode, contrairement aux deux autres, ne s'applique qu'avec des langages de haut niveau pour la spécification de l'application.

2.1.2 Espace des objectifs et espace du problème

Comme indiqué précédemment, le but de l'exploration de l'espace de conception est de réunir deux espaces, celui du problème et celui des objectifs [Gries03]. L'espace du problème définit l'espace de conception. L'espace des objectifs est lié aux contraintes et performances du système à développer. Détaillons les limites de ces derniers :

L'espace des objectifs est défini par les objectifs principaux de l'exploration tels que :

La vitesse de fonctionnement du système qui peut être exprimée par la fréquence maximum de fonctionnement, le débit des communications, le nombre d'opérations par cycle (ou seconde) suivant le type d'applications et d'architectures visées.

La consommation de puissance qui devient de plus en plus souvent l'objectif principal [Mudge01]. Elle rentre en compte, pour les systèmes embarqués, dans les contraintes de durée de vie du système (à cause de l'échauffement et de la durée de vie des batteries) et d'encombrement (taille des batteries).

La flexibilité est particulièrement étudiée dans le cas des architectures reconfigurables appliquées à des systèmes dont la mise à jour matérielle est un plus considérable. Le dynamisme de la reconfiguration ou l'adaptation des caractéristiques de l'architecture peuvent rentrer dans les objectifs.

La réduction des coûts de la conception peut définir un objectif de l'exploration en privilégiant les solutions les moins coûteuses. Cela peut passer par une réduction de la surface de silicium consommée, ou par, dans le cas des FPGA, le choix d'un taux d'utilisation optimum des ressources. Le coût en temps de développement dû aux outils nécessaires à la synthèse finale peut rentrer en compte. Il est lié, dans le cas des FPGA, à la complexité des algorithmes de placement-routage.

Des objectifs combinés peuvent être créés à partir de ces objectifs principaux afin de mieux répondre aux contraintes du système et/ou de diminuer le nombre de dimensions de l'espace des objectifs. Ce sont par exemple des ratios du type nombre de calculs effectués par watt, ou vitesse sur coûts, ou encore des produits tel que le produit énergie délais.

Dans tous les cas l'exploration peut être mono-objectif, elle se concentre alors sur un objectif qui peut être issu d'une combinaison de plusieurs objectifs. Ou alors l'exploration peut être multi-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, contrairement au cas mono-objectif, 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. Ainsi en réduisant cette fonction il est possible de satisfaire plusieurs objectifs en même temps, mais l'expression de cette fonction de coûts peut s'avérer extrêmement délicate à trouver. L'exploration peut conduire à un ensemble de solutions proposant toutes un certain compromis entre les différents objectifs, c'est souvent alors qu'interviennent des objectifs complémentaires permettant le choix d'un compromis. Parmi ces objectifs complémentaires nous pouvons citer :

> *Le taux d'utilisation des ressources dans le temps*. Effectivement suivant l'architecture choisie, ainsi que le déploiement de l'application sur celle-ci il est possible que des ressources soient sur ou sous utilisées dans le temps d'exécution de l'application.
>
> *La fiabilité* du système est souvent aussi importante que la consommation de puissance ou la vitesse de fonctionnement et en particulier pour des applications sensibles (l'aéronautique ou le spatial par exemple). La tolérance aux fautes peut par exemple être prise en compte dans la définition de l'architecture.
>
> *La compatibilité* peut rentrer en ligne de compte lorsque le système développé est embarqué dans un environnement hétérogène et communicant.
>
> *La facilité de mise en œuvre et d'utilisation* en particulier concernant l'initialisation du système et/ou la reconfiguration dans le cas de systèmes reconfigurables.
>
> *La testabilité* est nécessaire au système développé afin de pouvoir augmenter sa fiabilité et de vérifier en particulier qu'il a un comportement déterministe. La testabilité peut être pensée rapidement par l'ajout de prise en compte de standards de test tel que le standard JTAG Boundary Scan.
>
> *La sécurité* traite de la vulnérabilité de la conception par rapport à l'espionnage industriel et de la vulnérabilité du fonctionnement contre le sabotage. Le problème de sécurité, longtemps vu sous l'aspect logiciel, est aujourd'hui pris en compte du point de vue matériel. Il peut donc rentrer comme objectif de l'exploration afin de cibler des systèmes proposant une sécurité maximum [Bossuet04].

Enfin, chaque domaine applicatif spécifique peut apporter de nouveaux objectifs qui n'ont d'intérêt que dans un cadre très précis.

L'espace du problème ou l'espace de conception décrit la zone d'exploration dans laquelle une solution (ou un ensemble de solutions) du problème (réponse aux objectifs) est cherchée durant le processus d'exploration. Cet espace est délimité par les caractéristiques

architecturales des systèmes visés. Il ne sera pas le même dans le cadre des systèmes programmables ou dans le cas des systèmes reconfigurables par exemple. Dans ce dernier cas nous pouvons donner quelques dimensions de l'espace :

> *Le nombre et la granularité des ressources de traitement, de mémorisation et de contrôle*, entraînent une recherche qui s'appelle "le problème de la balance" [Moritz98]. Il s'agit de trouver le compromis entre la granularité de l'architecture et celle des outils associés. Par exemple une granularité fine augmente les possibilités de configurations de l'architecture mais en contrepartie augmente la complexité des algorithmes de placement-routage. Effectivement dans ce cas les ressources de communication doivent être nombreuses et réparties. Dans le cas particulier de la mémoire, la granularité et la distribution des ressources de mémorisation sur l'architecture vont jouer un rôle important dans la distribution et la localisation des données au cours de l'exécution.
>
> *La topologie et la granularité des ressources de communication* (routage) représentent les niveaux de hiérarchie. Suivant ceux-ci l'hétérogénéité des ressources de communication peut être plus ou moins importante. Leur granularité va aussi être un point d'exploration important qui entraîne une complexité des outils de placement-routage.
>
> *La taille des éléments hiérarchiques* (aussi appelés clusters) qui représentent donc le niveau de hiérarchie de l'application. Ces éléments sont souvent liés à la topologie du réseau de routage. Effectivement à l'intérieur d'un cluster le routage est différent de celui inter-cluster.
>
> *Le type et le dynamisme de la reconfigurabilité* peut être considéré dans l'exploration. Comme le premier chapitre l'a montré, la reconfiguration peut être complète, partielle, statique ou dynamique, simple ou multi-contextes. Dans le cas de la reconfiguration partielle le niveau de reconfiguration peut être par exemple un objet de l'exploration.
>
> Lorsqu'un *processeur* est embarqué proche de la structure reconfigurable différentes explorations peuvent être réalisées tant au niveau du *lien entre les deux parties* qu'au niveau de l'architecture de la partie programmable (*type de processeur, hiérarchie mémoire, interface de communication, etc.*).
>
> Comme sur les autres types de circuits des caractéristiques telles que le *nombre d'entrées/sorties* et les *standards de communication* associés peuvent rentrer dans l'exploration. *Le nombre d'horloges* peut aussi être exploré.

D'autres caractéristiques spécifiques à une architecture peuvent évidemment être exploitées lors de l'exploration, mais nombreux sont les processus d'exploration qui n'explorent qu'une partie des caractéristiques citées. De cette façon le processus cible un espace moins large mais plus efficacement.

Une fois les deux espaces délimités, le processus d'exploration peut être mis en œuvre. Pour cela diverses techniques d'exploration et de réduction de l'espace de conception peuvent être utilisées. Le paragraphe suivant nous en donne un aperçu.

2.2 Méthodes de réduction et d'exploration de l'espace de conception

Avant d'explorer l'espace de conception il est intéressant d'étudier une possible réduction de celui-ci afin de diminuer les temps de recherche. Le paragraphe 2.2.1 suivant donne quelques méthodes de réduction de l'espace de conception. Puis le paragraphe 2.2.2 se penche sur les méthodes d'exploration qui sont bien adaptées aux domaines des architectures matérielles reconfigurables.

2.2.1 Méthodes de réduction de l'espace de conception

La durée de l'exploration est directement liée à la largeur de l'espace à explorer. Plus celui-ci sera large et plus le temps nécessaire à son exploration sera long. Il est donc intéressant de chercher à diminuer cet espace avant même l'exploration. Différentes méthodes peuvent être mises en œuvre nous en détaillons trois.

Méthode de réduction par exploration hiérarchique : en utilisant plusieurs méthodes d'exploration travaillant chacune à des niveaux d'abstraction différents il est possible d'explorer l'espace de conception de façon hiérarchique, comme on peut le voir sur la pyramide de la figure 29. Des premières explorations basées sur des estimations d'abord grossières puis fines permettent de réduire à chaque itération l'espace de conception. Au cours du temps le niveau d'abstraction diminue, les modèles de caractérisation des applications et des architectures changent.

Méthode de découpage en petits éléments : la devise *« diviser pour régner »* s'applique évidemment dans le cas de l'exploration de l'espace de conception. Il peut être intéressant de diviser l'espace en sous-espaces de tailles inférieures et d'effectuer une exploration dans chaque espace. A l'issue du choix des solutions finales correspondant à chaque sous-ensemble, une mise en concurrence de ces solutions permet de trouver la solution pour l'espace complet. Cette méthode permet d'utiliser des algorithmes d'exploration complexes, qui sont intéressants lorsque l'espace à explorer n'est pas large.

Méthodes de découpage en parties indépendantes : pour certains problèmes les caractéristiques à explorer qui délimitent l'espace de conception peuvent être indépendantes (au moins sur un des objectifs). Alors il est possible de découpler les caractéristiques afin de réduire l'espace de conception à plusieurs espaces indépendants. La difficulté dans ce cas consiste à prouver que les sous-problèmes (ou sous-espaces) sont bien indépendants.

2.2.2 Méthodes d'exploration de l'espace de conception

Dans un premier temps donnons les principales caractéristiques des méthodes d'exploration de l'espace de conception.

Les méthodes mises en œuvre pour parcourir l'espace de conception vont dépendre dans un premier temps du nombre d'objectifs auxquels on souhaite répondre. Dans le cas d'une recherche mono-objectif l'exploration doit déboucher sur une solution unique qui répond

au mieux à l'objectif. Dans le cas multi-objectifs, si les objectifs sont trop nombreux, l'exploration proposera généralement un espace de solutions. La solution finale est trouvée par compromis sur les objectifs. Toutefois il est possible d'éliminer une partie des solutions avant compromis en utilisant la méthode de l'économiste Pareto [Pareto1896] comme par exemple dans les travaux présentés dans [Haulbert03].

Dans tous les cas (mono-objectif ou multi-objectifs), l'élimination des solutions superflues comme le choix des solutions finales peut se faire de façon automatique ou manuelle. Effectivement il est possible de guider manuellement l'exploration, ainsi le processus tire bénéfice des connaissances techniques et pratiques du concepteur. Cette technique d'exploration est certainement la plus réaliste, elle permet de converger plus rapidement vers une bonne solution. Cependant l'intrusion du concepteur dans les choix d'exploration entraîne un biais propre au concepteur. Dans ce cas l'exploration peut donner des résultats différents sur un même problème sous l'influence de deux concepteurs différents. Pour diminuer le biais les décisions d'exploration guidées peuvent être par exemple prises en équipe.

Que l'exploration soit manuelle ou automatique elle doit déboucher sur un résultat utilisable pour le concepteur. Ce résultat peut être donné de deux façons, soit le processus donne une ou des solutions architecturales répondant à un ou des objectifs, soit il donne des informations de conception qui permettront au concepteur de faire évoluer son application ou les architectures visées. Cela dépend du niveau d'abstraction de l'exploration. A haut niveau il est généralement plus intéressant de donner des informations de conception au concepteur contrairement aux niveaux plus bas d'abstraction pour lesquels le concepteur doit connaître la meilleure solution architecturale.

Maintenant que nous avons les caractéristiques des méthodes d'exploration nous pouvons donner les types d'exploration couramment utilisés. Pour explorer l'espace de conception il existe trois grands types de méthodes : exhaustives, stochastiques et enfin heuristiques comme indiqué sur la figure 31 [Gries03]. Sur cette figure les trois types de méthodes sont schématisés lors de l'exploration de solutions suivant deux paramètres P_1 et P_2. Ces paramètres sont des caractéristiques de l'espace de conception que nous avons vu précédemment.

Les méthodes exhaustives évaluent toutes les solutions et combinaisons possibles dans l'espace de conception. Dans ce cas la stratégie d'exploration est non guidée par le concepteur, de ce fait elle n'est pas biaisée par celui-ci mais bien évidemment elle nécessite un temps d'exploration beaucoup plus important. Ces méthodes s'utilisent par exemple avec des estimations au niveau système [Baghdadi00], [Auguin01]. De nombreuses références concernant ce type de méthode sont données dans [Gries03].

Les méthodes stochastiques mettent en œuvre des modèles probabilistes et aléatoires afin de ne tester qu'un nombre limité de solutions tout en garantissant de trouver une solution répondant aux objectifs avec un certain intervalle de tolérance. Ces méthodes ne tiennent pas compte du concepteur dans l'exploration et le choix des solutions explorées puisqu'elles sont choisies aléatoirement. L'algorithme éprouvé de Monte Carlo peut par exemple être utilisé pour le choix des solutions [Bruni01].

Les méthodes heuristiques utilisent les connaissances du concepteur apportées entre autres par les phases antérieures d'exploration. Ces méthodes ne garantissent pas de trouver une solution optimum mais elles permettent de rapidement converger vers une solution. L'exploration est localisée autour de points et elle est guidée par le concepteur. De nombreuses références concernant ce type de méthode sont données dans [Gries03].

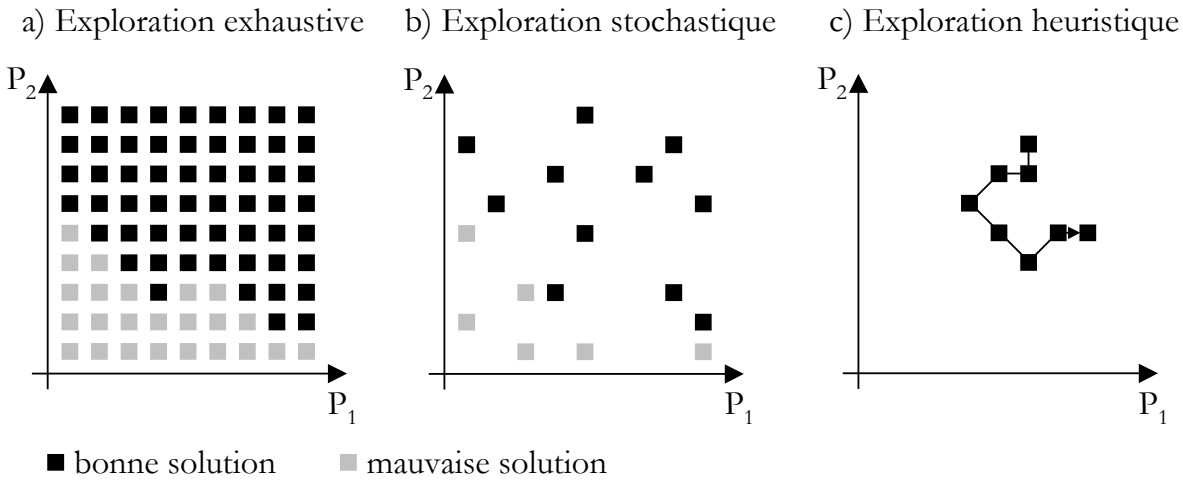


Figure 31 – Les trois types d'exploration, a) exploration exhaustive, b) exploration stochastique et c) exploration heuristique.

2.3 Représentation de l'espace de conception

Dans le diagramme en Y de la figure 30 apparaissent la spécification des applications et la spécification des architectures matérielles cibles. Ces spécifications permettent la variation des caractéristiques et donc l'exploration. Elles peuvent être plus ou moins génériques en fonction de leur niveau d'abstraction.

2.3.1 Spécification de l'application

Il existe de nombreuses spécifications pour les applications avec des niveaux d'abstraction très variés. La spécification peut être directement exécutable ou non, dans ce dernier cas elle est abstraite. Il serait long de vouloir donner tous les formalismes possibles, mais certains exemples permettent de voir l'étendue du choix de spécifications.

Les spécifications au niveau système sont principalement utilisées dans les outils de CoDesign afin d'exécuter la phase de partitionnement de l'application avant de spécifier chaque partie de l'application en fonction de la cible (logicielle ou matérielle). Citons comme exemple le langage SDL dans l'outil de CoDesign MUSIC [Jerraya99], ou le langage Esterel [Berry00] développé par la société Esterel Technologies et utilisé dans le projet RNTL EPICURE [Auguin03].

Les langages de haut niveau sont souvent utilisés car ils mettent à disposition du concepteur de nombreux outils. Ce peut être un langage comme le langage C utilisé par exemple dans l'outil Design Trotter [LeMoullec03a] [Bossuet03b] ou des langages orientés objet comme le C++ ou Java [Helaihel97]. Souvent cette spécification peut être transformée en un langage facilitant le développement des outils tel que par exemple le format intermédiaire de représentation de programme SUIF (*Stanford University Intermediate Format*). Ce formalisme peut encore être transformé en une spécification plus proche du matériel comme VHDL. Un exemple de ce type de transformations successives est donné dans [So02]. De plus en plus, dans le cas des FPGA le langage Matlab est utilisé car il permet une spécification efficace des applications de traitement du signal et des images. De plus, nombreux sont les outils de synthèse pour FPGA qui acceptent en entrée une spécification sous ce format (les outils Xilinx System Generator for DSP et Altera DSP Builder par exemple). La spécification Matlab est souvent utilisée dans les systèmes d'estimations, pour l'estimation temporelle [Bjuréus02] ou l'estimation en surface [Nayak02]. Mais Matlab peut être utilisé dans des flots d'exploration, citons comme exemple l'outil Laura ciblant des FPGA [Zissulecu03]. Certains langages de haut niveau sont développés uniquement pour une architecture, comme le langage ALE-X dont la syntaxe proche du langage C est développée spécifiquement pour l'architecture KressArray [Kress96] vue au paragraphe 1.2.2. Le langage Handel-C [Celoxica03] utilisé dans les outils de la société Celoxica est un autre exemple de spécialisation (par extension et réduction) du langage C pour la spécification d'applications ciblant des architectures matérielles.

Les graphes de flot de données ou DFG (*Data Flow Graph*) sont des graphes dont les nœuds représentent des traitements et les arcs des transferts de données. Les DFG ne peuvent représenter que des applications simples et séquentielles, ils sont donc utilisés à bas niveau, c'est par exemple le cas pour l'outil d'estimation de performances et d'exploration pour FPGA développé à l'institue technique fédérale de Suisse [Enzler00], ou l'outil d'estimation rapide de surface (ou de taux d'utilisation du FPGA) présenté dans [Kulkani02]. Insuffisants pour modéliser des applications dans lesquelles interviennent des contrôles, ce qui se traduit par du parallélisme d'exécution et/ou de l'exclusivité, les DFG ont évolué en CDFG (*Control Data Flow Graph*). Les CDFG peuvent ainsi décrire toute l'application, une utilisation est donnée dans [Zivkovic03]. Mais lorsque l'application est découpée en fonctions, il peut être intéressant d'introduire de la hiérarchie dans le graphe, c'est ce qui est fait dans la spécification HCDFG (*Hierarchical Control Data Flow Graph*) [Diguet00] de l'outil Design Trotter [LeMoullec03a].

Les graphes de communication sont des représentations de l'application qui ne se concentrent que sur les communications. Ce sont donc des spécifications abstraites puisque elles ne considèrent qu'une partie des caractéristiques de l'application. Par exemple dans le cas d'un système d'exploration des communications pour les SoC, un graphe appelé CAG (*Communication Analysis Graph*) est utilisé [Lahiri00b] & [Lahiri01]. Dans les travaux exposés dans ce document il apparaîtra dans le chapitre suivant qu'un graphe de communication appelé ACG (*Average Communication Graph*) est utilisé pour guider l'exploration architecturale [Bossuet03b].

Les modèles abstraits de coûts peuvent être utilisés pour caractériser l'application. Le modèle choisi dépend de ce que le concepteur souhaite prendre en compte dans son

exploration. Ce type de modèle a été développé au MIT pour un projet de recherche de granularité dans un système systolique [Yeung94], travaux étendus par la suite à l'architecture reconfigurable Raw [Moritz98], architecture de type tuile ou chaque tuile est un processeur élémentaire.

D'autres spécifications peuvent être utilisées pour caractériser l'application, cependant celles décrites ci-dessus paraissent particulièrement bien adaptées aux cas des applications ciblant des architectures reconfigurables. Effectivement elles sont adaptées aux formats d'entrée des outils associés à ce type d'architectures. Mais aussi, ces spécifications permettent de bien mettre en valeur le parallélisme potentiel de l'application. Il est également possible de faire un lien entre les opérations de traitement, les transferts de données et les contrôles présents dans la spécification et les ressources de l'architecture. Ces ressources sont spécifiées sous différentes formes lors de la spécification de l'architecture. Le chapitre suivant présente cette spécification architecturale.

2.3.2 Spécification de l'architecture

Concernant l'architecture il existe plusieurs familles de spécifications qui se séparent par leur utilisation. Une première famille de spécifications regroupe les modèles abstraits qui sont bien adaptés à l'estimation. Une autre famille est celle des modèles structurels et fonctionnels qui donnent une représentation plus ou moins fine de l'architecture matérielle. Enfin une dernière famille de spécifications est la famille des langages de description architecturale ou matérielle. Ils sont bien adaptés à la réalisation d'outils de synthèse générique ou de génération de compilateurs.

Les modèles abstraits décrivent avec plus ou moins de précision les performances de l'architecture pour exécuter des fonctions. Ces modèles associent donc à des fonctions (suivant le type d'architecture : instructions, tâches, traitements, mémorisations, communications …) des performances (vitesse, consommation, surface, coûts …). L'estimation qui s'en suit consiste après le déploiement de l'application sur l'architecture à estimer seulement les fonctions mises en œuvre lors de l'exécution. Dans le cas de l'outil Design Trotter (le paragraphe 2.5 reviendra plus longuement sur cet exemple), il existe deux modèles abstraits de performance. Le premier donne le nombre de cycles pour réaliser les traitements de l'application sur une architecture RTL virtuelle définie par l'utilisateur [LeMoullec03a]. Les performances temporelles indiquées dans ce modèle sont basées sur les connaissances de l'utilisateur. Le deuxième modèle concerne les architectures reconfigurables de grain fin, les FPGA. Les performances pour réaliser telle ou telle fonction sont données en nombre d'éléments logiques de base utilisés et en temps d'exécution [Bilavarn02]. Ces performances sont mesurées lors de pré-caractérisations. Cette méthode consistant à caractériser l'architecture en mesurant à l'avance les performances de l'exécution de traitements de base est souvent utilisée. [Enzler00] donne un autre exemple de ce type de modélisation abstraite par pré-caractérisation. Une autre solution de modélisation abstraite de performance est la paramétrisation de l'architecture pour définir des fonctions de coûts. Dans ce cas une étude complexe est nécessaire au préalable pour définir les paramètres de l'architecture qui vont intervenir dans l'amélioration ou la dégradation de telle ou telle performance, la consommation de puissance par exemple. Une fois ces paramètres déterminés il faut mesurer leur impact sur

la ou les performances étudiées afin de définir des fonctions de coûts. Cette méthode est utilisée dans [Choi02] et [Moritz98].

Les modèles structurels physiques ou fonctionnels décrivent la structure interne de l'architecture. Par exemple, un processeur est décrit par les unités de traitements, les registres et les bus de communications, mais aussi par les mémoires caches et les coprocesseurs, comme c'est le cas pour l'environnement de CoDesign PICO [Kathail02] du laboratoire de recherche de Hewlett Packard. Certains modèles structurels ne se concentrent que sur un aspect de l'architecture comme les communications par exemple [Lahiri00a]. Dans le cas des architectures reconfigurables de grain fin par exemple, un modèle structurel va décrire les ressources de traitement comme les ressources de routage. Ce modèle est dit physique lorsqu'il caractérise physiquement les ressources via des mesures ou l'utilisation de la modélisation P-Spice comme dans [Betz97]. La caractérisation physique est indispensable pour développer des estimateurs de performances précis avec un modèle structurel. Cependant la finesse physique proposée par de tels modèles structurels n'est pas toujours désirable ou possible à obtenir. Par exemple dans le cas d'outils génériques qui se veulent capables de cibler plusieurs architectures matérielles, la caractérisation physique de chaque possibilité architecturale est complexe à réaliser. Dans ce cas les modèles fonctionnels conviennent bien. Ils ne décrivent que les fonctions réalisables par les éléments de l'architecture. C'est notamment cette approche qui est utilisée dans [lagadec01]. La précision physique n'est pas toujours désirable, par exemple pour des estimations que l'on souhaite rapides et indépendantes de la technologique. Dans ce cas, la modélisation fonctionnelle convient bien puisqu'elle ne dépend pas de la technologie. De plus, elle est généralement moins détaillée que la modélisation physique ce qui permet le développement d'estimateurs plus rapides comme dans [Bossuet03]. Mais les estimations sont alors obligatoirement relatives car elles ne peuvent être précises sans caractérisation physique des différentes ressources de l'application. Le choix entre modélisation structurelle ou fonctionnelle s'impose parfois de lui-même en fonction des données dont on dispose. Effectivement, en particulier dans le cas des architectures reconfigurables, de nombreuses architectures ne sont pas aujourd'hui réalisées physiquement ce qui rend difficile la validation d'une modélisation physique.

Les spécifications en langage de description architecturale sont particulièrement utilisées dans le cas des systèmes programmables afin de générer des compilateurs. Chaque langage est spécialisé pour un type d'architecture ou de modèle de calcul. Parmi le nombre important de ces langages on peut citer *LISA* (*Language Instruction Set Architecture*) [Pees00], langage de description conçu spécifiquement pour la génération automatique de simulateurs compilés au niveau cycle et rapide. Il est basé sur un modèle de description comportementale. *LISA* permet de décrire les détails architecturaux et le pipeline des DSP récents. *ISDL* (*Instruction Set Description Language*) [Hadjiyiannis97] a été conçu spécifiquement pour l'exploration architecturale. C'est un langage comportemental qui permet une modélisation extrêmement détaillée du jeu d'instructions et une description de l'ensemble des mécanismes présents dans les architectures spécialisées. Il est basé sur une grammaire où les règles de production sont utilisées pour abstraire les motifs communs dans la définition des opérations. Le langage *ISDL* a pour objectif de couvrir un large éventail d'architectures avec une emphase plus particulière sur les architectures de type

VLIW (*Very Long Instruction Word*). *nML* [Fauth95] est un langage de description de haut niveau qui peut être utilisé pour supporter des outils générés automatiquement. Le jeu d'instructions est représenté sous une forme comportementale en utilisant une grammaire. La structure du langage sous forme de grammaire a l'avantage de factoriser les regroupements d'informations puisqu'une règle de grammaire suffit à modéliser différentes alternatives. *nML* est très similaire à *ISDL*. Enfin *ARMOR (Architecture Modeling for Retargetability)* [Messé99] utilise la philosophie de *nML* : une description est une grammaire dont chaque dérivation est un comportement possible du processeur. La grammaire définie des composants assimilables à des instructions et crée des groupes de composants parallèles pour refléter le parallélisme d'instructions. *ARMOR* ne décrit pas une liste d'instructions, mais plutôt la liste des comportements possibles du processeur mis à disposition du programmeur via le jeu d'instructions. Dans une description du processeur cible, il est nécessaire de connaître à quel cycle une instruction utilise les ressources. *ARMOR* attribue pour cela des dates d'utilisation aux ressources opératoires ou registres.

2.4 Exemple dans l'espace reconfigurable

L'exploration de l'espace de conception est particulièrement développée pour les systèmes spécifiques et pour les systèmes programmables. En effet ces deux domaines sont étudiés depuis bien plus longtemps que le domaine du reconfigurable. Les travaux d'exploration de l'espace de conception des architectures reconfigurables sont peu nombreux en particulier en ce concerne les architectures gros grains. Des études ont été faites concernant l'exploration de l'espace de conception d'applications ciblant des FPGA. Il est également possible d'explorer l'architecture du composant avec des outils génériques (ces derniers permettent des estimations ou des synthèses avec des caractéristiques flexibles pour le composant cible). Ce paragraphe va donner quelques exemples d'outils d'exploration ciblant des architectures reconfigurables grain fin et gros grain.

2.4.1 Outils d'exploration et d'estimation ciblant des FPGA

Travaux de l'Université de Californie du Sud [Choi02]

L'approche proposée par ces travaux se concentre uniquement sur l'estimation de la consommation en énergie et en puissance. L'approche utilise un modèle haut niveau de la puissance qui est spécifique à un domaine. Les auteurs de ces travaux définissent un domaine par la réunion d'une famille d'architectures et d'un ensemble d'algorithmes réalisant tous la même application. 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. Comme la modélisation est 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.

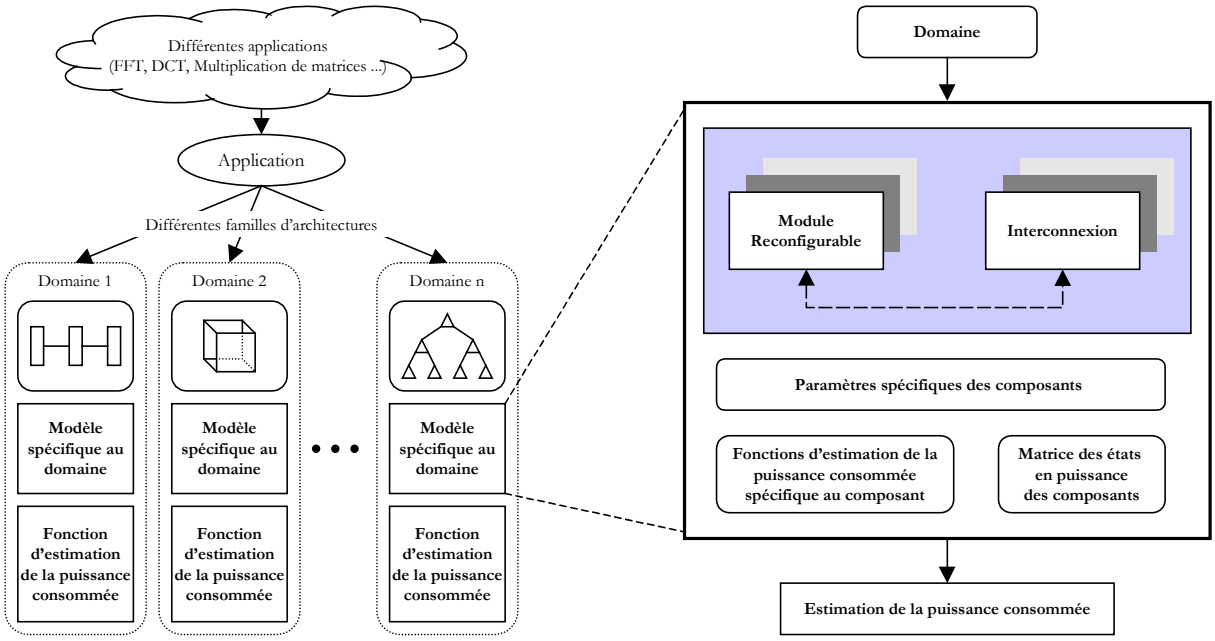


Figure 32 – Modélisation ciblant un domaine, et description du flot d'estimation

Le modèle architectural utilise deux entités, les modules reconfigurables et les interconnexions. Les modules reconfigurables peuvent correspondre aux éléments logiques reconfigurables de base des FPGA mais peuvent aussi être des clusters plus importants comprenant d'autres modules reconfigurables et des interconnexions, il s'agit alors d'une vision hiérarchique des modules. Les modules reconfigurables sont décrits par un ensemble de caractéristiques qui agissent sur la consommation de puissance telles que : la fréquence de fonctionnement, la largeur des mots pour les traitements, le nombre de ces modules dans le circuit, l'état de la consommation de puissance du module, etc. Une hypothèse est formulée sur les modules reconfigurables et indique que la consommation de ces modules est indépendante de leur position dans le circuit. Les auteurs notent que cette hypothèse entraîne une erreur d'estimation (non quantifiée dans l'article) mais qu'elle permet une simplification notable du système. Les interconnexions sont définies par leurs caractéristiques telles que leur longueur, leur largeur ou le taux d'activité des données. Les auteurs appellent "composant" un élément constitué de modules reconfigurables et ou d'interconnexions.

L'utilisateur peut utiliser un outil, développé par les Universités américaines de Californie du Sud et de Vanderbilt au Tennessee, appelé MILAN [Bakshi01] [Mohanty02] pour définir les paramètres de chaque composant qui influent sur la consommation de puissance du circuit. Cet outil s'appuie sur d'autres outils d'estimation et de simulation comme l'estimateur de puissance Xilinx Xpower [Xilinx04b] et le simulateur ModelSim [Model03].

L'utilisateur doit disposer d'une bonne connaissance du domaine visé, voir même d'une expertise de la consommation de puissance des architectures visées. Le système est donc guidé et influencé par l'utilisateur ce qui peut biaiser les résultats. Il met en œuvre des concepts intéressants comme la prise en compte de l'état concernant la consommation de puissance de chaque composant à chaque cycle de l'exécution de l'application, mais cet état reste difficile à estimer.

La figure 32 reprend le principe de la modélisation et de l'estimation de puissance par domaine et développe le modèle de haut niveau. 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 applicatif. De plus, la prise en compte d'un état de consommation de puissance des composants internes apporte une connaissance fine de la distribution de la dissipation de puissance. Malheureusement cette étude reste trop théorique et demande une grande expertise de la part de l'utilisateur.

Cette étude propose donc une exploration architecturale mono-objectif puisqu'elle ne s'occupe que de la consommation de puissance. Les spécifications de l'application et de l'architecture sont faites par paramétrisation par rapport à cet objectif. L'exploration est réalisée de façon exhaustive (toutes les solutions du domaine sont explorées) et manuelle. Elle débouche sur la proposition d'une architecture proposant la consommation minimum de puissance.

Travaux de l'Institut Technologique Fédéral de Suisse (ETH) [Enzler00]

Ces travaux sont basés sur une méthode d'estimation de performances (surface et vitesse) d'applications très régulières sur FPGA. Effectivement la méthode proposée cible des algorithmes très réguliers et de type flot de données, tels que des algorithmes multimédia, de télécommunication ou encore de cryptographie. La spécification de l'application utilise une description non formelle telle qu'un DFG. L'estimation de performances est effectuée avant synthèse, placement et routage mais utilise comme dans [Bilarvarn02] une pré-caractérisation d'opérateurs arithmétiques et logiques de base sur l'architecture ciblée.

La figure 33 présente un schéma du flot mis en œuvre, il est très proche du flot en Y proposé à la figure 30. Effectivement dans cet exemple les descriptions et caractérisations de l'application et de l'architecture sont clairement séparées. Ce flot se déroule en quatre étapes. La première étape est la caractérisation de l'application. Les caractéristiques retenues sont : le nombre de données en entrée et en sortie, le nombre de signaux de contrôle, le nombre d'opérateurs (additionneurs, soustracteurs, multiplieurs, multiplexeurs, comparateurs et fonctions logiques), le nombre de registres (hors registres de pipeline) et le nombre d'itérations (lors d'estimation d'un cœur de boucle). En fonction de ces caractéristiques, des paramètres sont calculés pour caractériser l'application en surface, temps et nombre d'entrées/sorties. Des facteurs correctifs empiriques sont introduits dans ces fonctions de coûts pour prendre en compte l'effet du routage sur les performances. La deuxième étape consiste en la pré-caractérisation des traitements sur le FPGA cible. Il s'agit

de synthétiser et de caractériser des opérateurs de base qui sont susceptibles d'être utilisés par les applications. La caractérisation peut être effectuée à l'avance et enrichie au cours d'utilisations successives. La troisième étape rassemble les caractéristiques de l'application et du FPGA pour calculer les estimations de performances. Ces estimations sont liées à l'expérience pratique des auteurs de cette étude. Cette troisième étape donne en résultat les performances en terme de surface (en éléments logiques configurables de base), fréquence, débit, latence (en cycles) et nombre d'entrées/sorties. La dernière étape du flot d'exploration et d'estimation permet de modifier les caractéristiques de l'application suivant trois paramètres S, R et D. Le paramètre S indique le nombre de registres de pipeline insérés dans le chemin de données afin de réduire la longueur du chemin critique. Le paramètre R indique le nombre de fois que le bloc estimé est reproduit, afin d'estimer une augmentation du parallélisme (cas du déroulage de boucle). Enfin le paramètre D correspond à une division du bloc estimé en sous-blocs identiques, en fait en modifiant la granularité des opérateurs. L'exploration de l'espace de conception conformément à la figure 33 ne se fait que sur les caractéristiques de l'application.

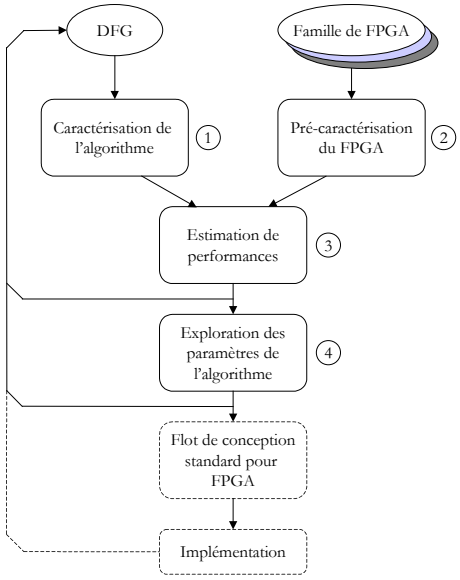


Figure 33 – Flot d'estimation et d'exploration

Cette méthode a l'avantage d'être simple mais elle a quelques défauts. En particulier les effets du routage ne sont pas assez pris en compte pour une estimation qui se veut relativement précise. De ce fait la méthode correspond mieux à des applications régulières dont le déploiement sur l'architecture entraînera des connexions localisées. Pour des applications moins régulières la méthode sous estime la surface et donne une estimation optimiste des délais. Ces travaux ciblent des architectures classiques de FPGA, mais ne sont pas prévus pour les nouvelles architectures qui embarquent des blocs de mémoires (puisqu'il n'y a pas d'estimation de la mémoire) et des opérateurs câblés.

Cette étude propose donc une exploration algorithmique multi-objectifs (surface et vitesse). La spécification de l'application est un DFG. Celle de l'architecture est un modèle abstrait de performances obtenues par pré-caractérisation. L'exploration est réalisée de façon exhaustive et manuelle. Elle donne en résultat des informations de conception sur l'application telle que l'exploitation du parallélisme ou du niveau de pipeline et leurs implications sur les objectifs.

2.4.2 Outils génériques ciblant des FPGA et pouvant être intégrés dans un flot d'exploration architecturale

Les deux outils présentés, VPR et Madeo, sont avant tous des outils génériques de placement-routage pour FPGA. Cependant du fait de leur généricité il est possible d'effectuer à la main une l'exploration architecturale en modifiant les caractéristiques de l'architecture modélisée. C'est pourquoi il est intéressant de les étudier.

Versatile Place and Route, VPR, outil de l'Université de Toronto [Betz97]

VPR est un outil de placement-routage générique basé sur un modèle structurel de FPGA de type îlot de calcul (largement inspiré des architectures Xilinx). Le modèle s'appuie sur une liste de caractéristiques de l'architecture telles que la taille des LUT, le nombre de LUT dans les clusters de base (comme les CLB dans les architectures Xilinx), largeur, longueur et population (nombre de connexions) des canaux de routage, le choix du type de matrice de connexions entre les lignes verticales et horizontales de routage. Enfin l'outil utilise une modélisation physique de type P-Spice pour l'estimation finale des performances, il faut lui préciser la technologie utilisée via la taille des transistors.

La modélisation des caractéristiques du FPGA est transformée en graphe pour être utilisée plus facilement par les outils de placement-routage. Chaque ligne de routage, ou élément configurable est représenté par un nœud, les switchs sont représentés par des arcs orientés (unidirectionnellement dans le cas de buffers, bidirectionnellement dans le cas de transistors). L'algorithme de placement-routage utilisé est l'algorithme PathFinder [McMurchie95] basé sur l'algorithme de routage en labyrinthe [Lee61].

L'outil peut prendre en entrée un placement émanant d'un autre outil, ce qui est une propriété intéressante pour comparer des outils se positionnant en amont dans le flot de conception. Mais l'outil peut directement prendre une netlist via l'utilisation d'un outil complémentaire T-Vpack [Betz00] qui effectue le déploiement d'une netlist au format BLIF (*Berkeley Logic Interchange Format*) sur des éléments configurables à base de LUT comme illustré à la figure 23 du paragraphe 1.2.4. Il est possible de décrire l'application dans le format EDIF (*Electronic Data Interchange Format*) qui est le format de sortie de nombreux outils de synthèse haut niveau, un utilitaire permet de passer du format EDIF au format BLIF [Leventis98].

La figure 34 présente le flot de l'outil VPR, en sortie duquel des informations concernant l'utilisation des ressources logiques et des ressources de routage sont données, ce qui permet un retour efficace vers l'architecture afin d'améliorer les performances. L'outil fonctionne selon deux modes. Dans un mode automatique VPR place et route l'application sur le FPGA en faisant automatiquement varier le routage (en particulier la largeur des canaux). L'outil répète plusieurs fois le processus afin de trouver le nombre minimum de lignes de routage satisfaisant pour l'application. Dans le mode forcé, l'outil ne fait pas varier le routage, il prend en compte les considérations de l'utilisateur, et lui indique finalement si le circuit est routable ou non pour l'application développée.

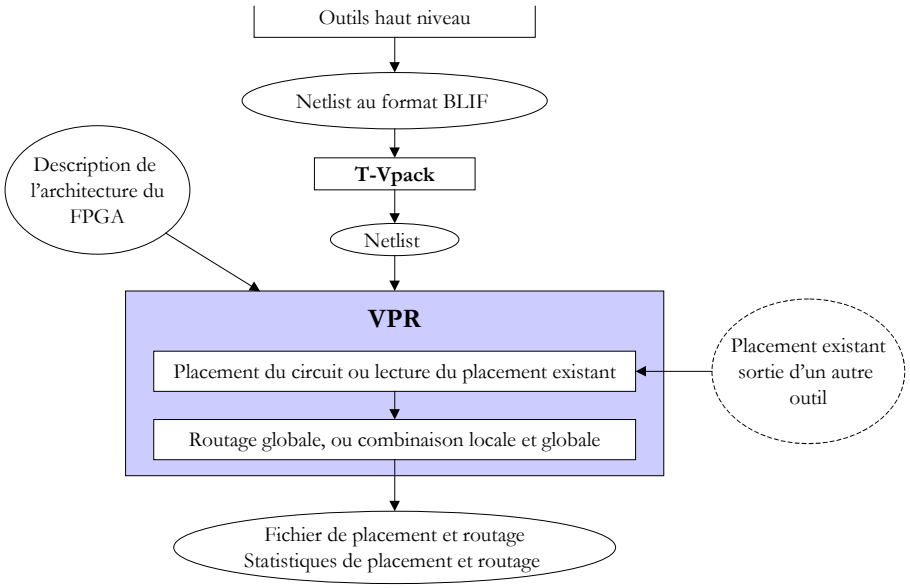


Figure 34 – flot complet de l'outil VPR et de son environnement.

Cet outil a permis à l'équipe qui l'a développé d'effectuer des études d'explorations (manuelles) architecturales. En particulier concernant la taille des LUT et le nombre de LUT dans les clusters pour assurer une bonne routabilité des circuits sans pour autant utiliser des algorithmes trop complexes [Ahmed00]. Cet outil s'est adapté à l'évolution des architectures de FPGA, il permet la modélisation de circuit embarquant des mémoires [Wilton99].

Cette étude permet donc via un outil générique de placement routage une exploration architecturale mono-objectif puisqu'elle se focalise uniquement sur l'utilisation du routage. La spécification de l'application correspond dans ce cas à une netlist, c'est à dire au résultat d'un outil de synthèse. Un modèle structurel est utilisé pour la spécification de l'architecture. La modélisation est physique, elle se base sur la modélisation P-Spice. L'exploration est réalisée de façon exhaustive et manuelle. Elle débouche sur des informations de conception de l'architecture du FPGA (taille des clusters, taille des éléments de base, routage) basées sur l'utilisation des ressources de routage.

Madeo, outils de l'Université de Bretagne Occidentale de Brest [Lagadec01]

Madeo constitue une chaîne de synthèse générique pour architecture reconfigurable basée sur un outil bas niveau de placement-routage Madeo-Bet. Ce dernier est très proche de l'outil VPR vu précédemment car il prend en entrée deux fichiers de description, un pour l'architecture et l'autre pour l'application. De plus, il utilise les mêmes algorithmes pour le placement-routage à savoir l'algorithme PathFinder [McMurchi95] basé sur l'algorithme de routage en labyrinthe [Lee61]. La différence entre les deux outils est que dans Madeo-Bet les architectures sont modélisées en utilisant un langage objet (la raison de ce choix est largement commentée dans [Lagadec00]) et qu'aucune hypothèse n'est faite quant à leur régularité, ce qui permet de modéliser des architectures complètement irrégulières et hétérogènes.

Une couche supérieure à Madeo-Bet, permet la compilation de la grammaire architecturale pour instancier un modèle d'architecture. A ce niveau, l'application décrite avec un graphe flot de données, DFG, est synthétisée vers de la logique niveau registre ou RTL (*Register Transfert Level*).

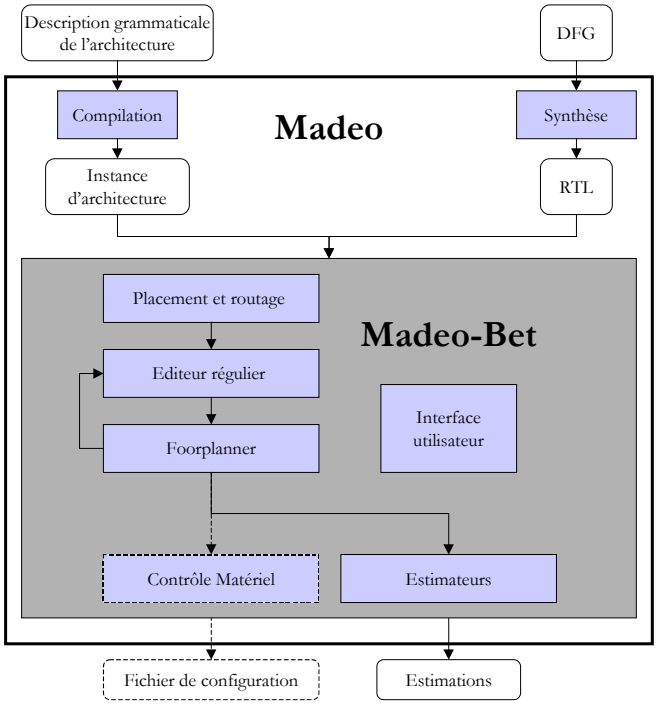


Figure 35 – flot des outils Madeo et Madeo-Bet.

Le flot, figure 35, présente plusieurs outils utilisés au niveau bas par Madeo-Bet. Le placement-routage convertit la description au niveau RTL en une configuration de l'architecture. L'éditeur régulier permet la réplication régulière de motifs de configuration dans l'architecture. Cette fonctionnalité permet la production de circuits réguliers par factorisation de la phase de placement-routage. Un foorplanner permet des optimisations

sur l'agencement des motifs de configuration via l'éditeur régulier, il utilise un algorithme du type recuit simulé. Une fois la configuration de l'architecture stabilisée, des estimateurs s'appuyant sur les facteurs de coûts des différents éléments de l'architecture calculent des fonctions de coûts globales. Ces fonctions évaluent la qualité du circuit produit suivant divers critères : surface englobante, coût du routage, chemin critique, etc. Il est possible d'utiliser un contrôle matériel si la définition de l'architecture est publique afin de générer le fichier de configuration du circuit (*bitstream*). Enfin une interface utilisateur interactive offre à l'utilisateur une représentation visuelle et lui permet de commander les divers outils.

Pour une plus grande efficacité l'outil peut être spécialisé pour une architecture sur laquelle il est possible d'effectuer un certain nombre d'explorations architecturales, telles que : la prospection sur la largeur des canaux de routage, sur la granularité des éléments logiques, sur le type de connexions ou sur le type de matrices de routage. Il est aussi possible de remplacer certains éléments de l'architecture et ainsi de faire évoluer sa structure. Des expériences ont été menées sur des architectures commerciales : Xilinx XC6216 [Lagadec 00], Xilinx Virtex-II et Atmel AT40K [Gouyen03].

Cette étude permet donc via un outil générique de placement routage une exploration architecturale multi-objectifs (utilisation du routage, surface, chemin critique). La spécification de l'application est un DFG. Un modèle fonctionnel est utilisé pour la spécification de l'architecture. L'exploration est réalisée de façon exhaustive et manuelle. Elle débouche sur des informations de conception de l'architecture du FPGA (taille des clusters, taille et fonctions des éléments de base, routage).

2.4.3 Outils d'exploration architecturale pour architectures reconfigurables gros grains

Flot d'exploration développé au MIT pour l'architecture Reconfigurable Raw [Moritz98]

L'architecture reconfigurable Raw est une architecture du type tuile qui a largement inspiré les concepteurs de l'architecture reconfigurable aSoC [Laffely01] présentée au paragraphe 1.2.3. La différence est que dans l'architecture Raw toutes les tuiles sont identiques, elle est donc bien moins hétérogène que l'architecture aSoC. Chaque tuile est constituée d'un cœur de processeur RISC (qui peut être ou non superscalaire) et d'une mémoire SRAM pour les données et les instructions. Comme dans aSoC, les tuiles sont disposées en matrice et reliées par un réseau de communication point à point, chaque tuile dispose d'une interface de communication programmable (ou routeur). De plus, une mémoire DRAM est connectée à la matrice de tuiles, ce qui fait deux niveaux hiérarchiques de mémoire et de communication : communications locales (entre tuiles) et communications globales (entre les tuiles et la mémoire DRAM).

Le but du flot d'exploration, figure 36, dans ce cas est de trouver, pour une surface de silicium contrainte, les paramètres de l'architecture assurant un temps d'exécution minimum de l'application. Il faut disposer d'un partitionnement de l'application sur l'architecture, chaque partition devant exiger la même quantité de ressources à l'architecture. De ce fait le flot ne s'applique qu'à des applications très régulières, ou à des cœurs de boucles que l'on souhaite exécuter en parallèle.

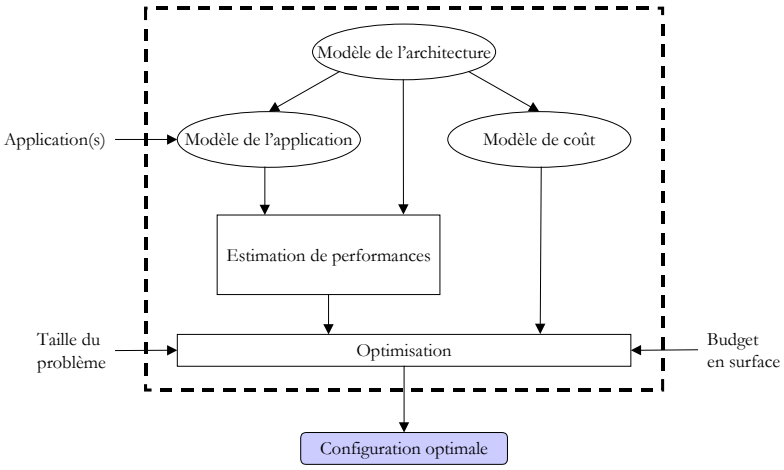


Figure 36 – Flot d'exploration pour l'architecture reconfigurable Raw

S'appuyant sur les travaux de Yeung et al. [Yeung94] la modélisation de l'architecture est une modélisation abstraite paramétrique. C'est à dire que l'architecture n'est pas modélisée par sa structure ni par ses capacités fonctionnelles, mais par quelques paramètres suffisant pour estimer des performances via des fonctions de coûts. Les paramètres utilisés pour la modélisation de l'architecture sont : le nombre de tuiles, la puissance de calcul par tuile (en nombre d'opérations par cycle), la quantité de mémoire SRAM par tuile (en nombre de mots), la bande passante des communications entre tuiles voisines (en nombre de mots par cycle) la latence de ces communications (en nombre de cycles) et de même pour les communications globales avec la mémoire DRAM.

Associé à ce modèle, un modèle de coût en surface basé sur des observations empiriques et statistiques d'implémentations de cœurs de processeurs superscalaires et de routeur, est utilisé pour estimer le coût en surface de l'architecture. L'unité retenue est le bit de SRAM équivalent afin d'être indépendant de la technologie. Ce modèle donne donc en fonction des paramètres de l'architecture le coût surfacique du cœur de processeur en fonction de la puissance de calcul et du nombre d'unités de traitement, le coût surfacique de la mémoire en fonction de la capacité des mémoires SRAM et le coût surfacique des communications en fonction de la bande passante et du routeur.

Enfin l'application partitionnée en sous-problèmes est décrite par le nombre de ressources de l'architecture qu'elle va utiliser. La description de l'application se base donc sur le modèle paramétrique de l'architecture. Dans ce cas on ne retrouve pas tout à fait le diagramme en Y de la figure 29, puisque la modélisation de l'architecture est nécessairement faite avant celle de l'application qui en dépend.

Finalement des fonctions de coûts vont estimer le temps d'exécution des sous-problèmes sur l'architecture, le temps dédié aux communications locales et globales pour estimer le temps d'exécution global de l'application sur l'architecture décrite. Enfin, un processus

simple d'optimisation, fait le lien entre une surface imposée et un temps d'exécution minimum. Le processus donne la configuration optimale de l'architecture pour répondre au problème à surface contrainte.

Dans [Moritz98] un certain nombre d'exemples et résultats avec des applications régulières sont donnés. Les inconvénients majeurs de cette étude sont qu'elle demande une grande expérience a priori pour déterminer les fonctions de coûts (en particulier concernant l'implémentation de cœurs de processeurs) et elle ne s'applique qu'à des applications très régulières. Cependant l'exploration peut être très rapide puisque les estimations issues des fonctions de coûts sont simples à calculer. Ces travaux restent néanmoins des travaux théoriques.

Cette étude propose l'exploration architecturale de l'architecture reconfigurable à gros grain Raw. L'exploration est mono-objectif puisqu'elle se focalise uniquement sur la réduction du temps d'exécution sous contrainte surfacique. L'architecture et l'application sont modélisées de façon paramétrique. L'exploration est réalisée de façon heuristique et automatique. Elle débouche sur une configuration optimale de l'architecture Raw.

Xplorer, outil d'exploration développé à l'Université de Kaiserslautern pour l'architecture Reconfigurable KressArray [Nageldinger01]

L'outil Xplorer a été développé pour explorer l'espace de conception de l'architecture KressArray [Kress96] vue au paragraphe 1.2.2. C'est une architecture matricielle dont les éléments reconfigurables de base, les rDPU, peuvent réaliser des opérations arithmétiques de granularité moyenne (addition, multiplication, etc.) et du routage point à point. Cet outil a vocation à accompagner l'utilisateur dans la création d'une architecture de type KressArray qui soit en adéquation avec l'application développée. Effectivement cet outil est semi-automatique, il laisse à l'utilisateur le soin de faire évoluer l'architecture en fonction d'informations sur les performances estimées et de suggestions architecturales. A l'issue du processus l'outil peut générer un fichier de description matérielle (actuellement en format Verilog) pour effectuer des simulations.

L'architecture est modélisée par un ensemble de paramètres tels que :

- La taille de la matrice (le nombre de rDPU).
- Les fonctions réalisables par les rDPU.
- Les connexions au voisinage pour les rDPU, le nombre de connexions horizontales et verticales avec des liens unidirectionnels ou bidirectionnels.
- Le nombre de lignes et de colonnes de bus, leur population (nombre de connexions à des rDPU) et leur segmentation.
- Les zones dans lesquelles les rDPU ont des fonctionnalités différentes. Par exemple une zone dans laquelle tous les rDPU peuvent réaliser une multiplication contrairement aux autres rDPU de l'architecture.
- La longueur maximale du chemin de routage pour les connexions au voisinage (définition du voisinage).

- Le nombre de canaux de routage traversant les rDPU.
- Le positionnement des ports de communication avec des périphériques de la matrice (mémoires ou autres circuits).
- Le regroupement possible des ports de communication.
- Enfin il est possible de geler le placement de certaines parties du circuit pour intégrer des IP par exemple.

L'application est spécifiée dans un langage de haut niveau proche du langage C développé dans ce cadre, le langage ALE-X [Kress95]. L'outil contient un compilateur de ce langage qui transforme la spécification de haut niveau dans un format intermédiaire adapté aux outils du flot présenté figure 37. Le premier de ces outils est une estimation des besoins minimums de l'architecture, le résultat de cet outil complète les informations du format intermédiaire de spécification de l'application. Puis l'utilisateur peut lancer les différentes phases qui vont lui permettre l'exploration architecturale. En premier lieu l'application va être déployée sur l'architecture caractérisée. Puis un ordonnancement des données d'entrées est effectué, une analyse statistique en est faite. Puis les données du déploiement et l'analyse statistique des données d'entrées sont analysées, fournissant des suggestions architecturales à l'utilisateur avec un pourcentage de confiance dans chaque suggestion, elles sont classées de la plus probable à la moins probable.

Pour gérer le flot d'exploration (figure 37), l'utilisateur a à sa disposition un éditeur qui lui permet de modéliser par caractérisation l'architecture, de visualiser et de modifier le résultat du déploiement et enfin d'optimiser les paramètres d'estimations. Bien sûr les suggestions architecturales de fin de flot lui sont transmises via l'éditeur.

Afin de parvenir à donner des suggestions d'améliorations architecturales à l'utilisateur, Xplorer va prendre en compte un certain nombre de données issues du déploiement telles que : le taux d'utilisation des bus (en pour-cent), le taux de communication entre rDPU voisins (en pour-cent), le nombre de connexions aux bus et l'estimation du temps d'exécution en nombre de cycles. L'outil en déduit entre autre une estimation de la consommation de puissance. Enfin l'analyseur va utiliser une démarche du type système expert associé à la logique floue [Kandel92] pour proposer des suggestions. C'est à dire que les suggestions de l'analyseur sont déjà connues et répertoriées dans une "mémoire de suggestion". L'utilisateur peut au cours de ses expériences enregistrer de nouvelles suggestions. L'analyseur prend donc les données générées par les outils amont, il les combine puis utilise une démarche de logique floue pour en tirer des suggestions. Du fait de la démarche floue, l'analyseur ne peut donner des solutions strictes, mais il va donner un pourcentage de confiance dans des solutions qui pourraient améliorer l'architecture pour l'application développée. C'est une approche très intéressante puisque le domaine de l'exploration de l'espace de conception des architectures reconfigurables est souvent basé sur des visions empiriques et sur les connaissances des concepteurs. Dans ce cas l'approche floue donne de la liberté à l'utilisateur et propose des informations sans contraindre l'exploration. Peut être s'agit-il là d'une des meilleures façons de répondre à ce problème.

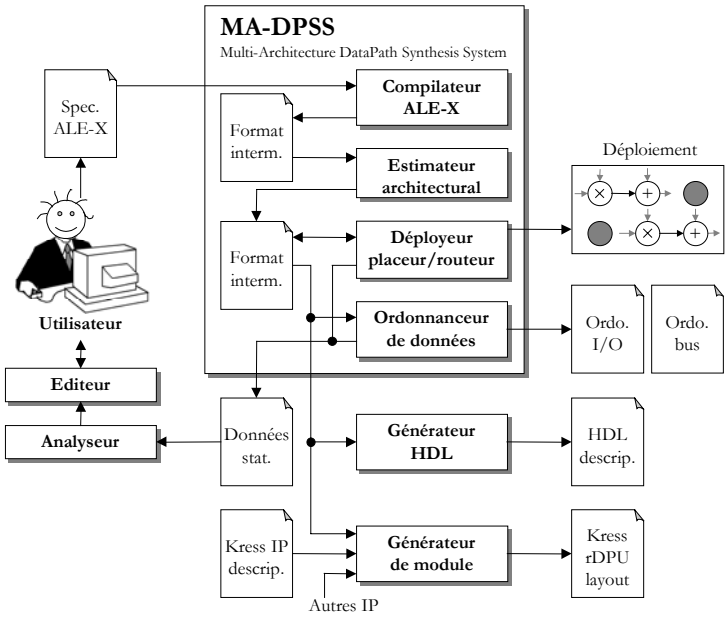


Figure 37 – Flot Xplorer simplifié

Cette étude propose l'exploration architecturale de l'architecture reconfigurable à gros grain KressArray. L'exploration est multi-objectifs puisqu'elle se focalise à la fois sur la réduction de la consommation de puissance comme sur la bonne utilisation des ressources de routage. La spécification de l'application est réalisée avec le langage haut niveau ALE-X spécialement développé pour cet outil. L'architecture est modélisée de façon paramétrique et aussi via un modèle structurel. L'exploration est réalisée de façon heuristique (avec utilisation de la logique floue) et semi-automatique puisque les suggestions d'explorations sont proposées à l'utilisateur pour approbation. Elle débouche sur une configuration optimale de l'architecture KressArray.

2.4.4 Quels outils d'exploration pour quelles architectures reconfigurables ?

Les trois paragraphes précédents ont montré différents outils d'exploration de l'espace de conception des architectures reconfigurables. Ces exemples sont intéressants car ils ne travaillent pas tous au même niveau d'abstraction pour les spécifications des applications et des architectures. Certains effectuent une exploration de l'application et d'autres effectuent une exploration de l'architecture. Il serait intéressant d'utiliser en parallèle ces outils pour explorer l'ensemble de l'espace de conception. Une différence existe aussi dans le type d'architecture visée, d'abord concernant le grain fin et épais, mais aussi en fonction de la liberté donnée à l'exploration architecturale. Concernant les outils visant des architectures à gros grain ils sont spécifiques à une architecture, réduisant du coup l'espace de conception autour d'une architecture bien définie. D'où la nécessité de développer des outils

génériques pour ces architectures. Le tableau 2 reprend l'essentiel des caractéristiques des six exemples détaillés dans les paragraphes précédents. Dans ce tableau les cibles architecturales sont qualifiées de "FPGA simples" si l'outil ne prend pas en compte les mémoires embarquées, et de "FPGA complexes multi-grains" lorsque l'outil prend en compte les mémoires embarquées ainsi que les opérateurs arithmétiques câblés.

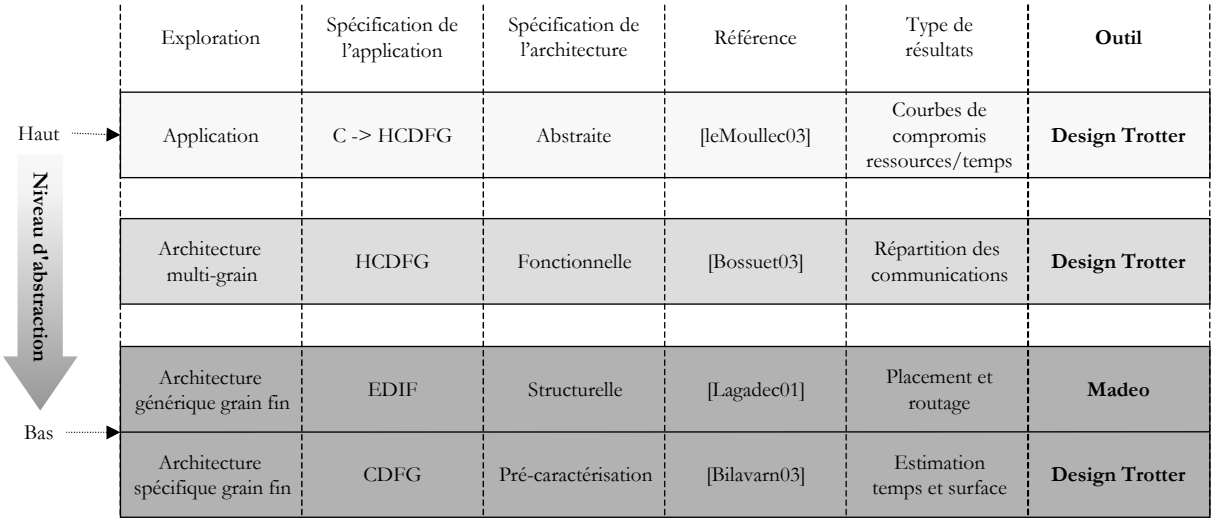
Outils	Cible architecturale	Spécification des applications	Spécification des architectures	Type d'exploration	Résultat de l'exploration
Univ. Californie Sud [Choi02]	FPGA simple	Paramétriques	Paramétriques	Architecturale Mono-objectif (puissance) Manuelle et exhaustive	Une architecture parmi un ensemble (domaine)
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 de l'application (parallélisme et niveau de pipeline)
VPR Univ. Toronto [Betz97]	FPGA complexe	Netlist BLIF (ou EDIF)	Modèle structurel	Architecturale Mono-objectif (routage) Manuelle et exhaustive	Informations de conception de l'architecture du FPGA (taille des clusters, taille des éléments de base, routage)
MADEO Univ. Brest [Lagadec01]	FPGA complexe multi-grains	DFG	Modèle structurel	Architecturale Multi-objectifs (routage, surface et chemin-critique) Manuelle et exhaustive	Informations de conception de l'architecture du FPGA (taille des clusters, taille et fonctions des éléments de base, routage)
Raw M.I.T. [Moritz98]	Architecture gros gain Raw	Paramétriques	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

Tableau 2 – Comparaison d'outils d'exploration de l'espace de conception des architectures reconfigurables

Après l'étude de plusieurs exemples, nous pouvons nous demander quelles seraient les caractéristiques d'un bon outil d'exploration de l'espace de conception pour architectures reconfigurables en nous inspirant des avantages et inconvénients des outils présentés auparavant. Tout d'abord il semble intéressant d'effectuer une double exploration ; exploration de l'application via le niveau de parallélisme ou le déroulage de boucle et une exploration architecturale. Les architectures explorées doivent pouvoir être gros grain et grain fin, mais aussi hétérogènes, ce qui semble difficile à faire compte tenu des exemples présentés qui ciblent un certain niveau de granularité. Il doit également pouvoir explorer un large espace tout en étant capable au final de proposer des estimations fiables à l'utilisateur.

Cette dernière doléance semble quelque peu délicate à mettre en œuvre et paraît même contradictoire. Une solution serait de ne pas chercher l'outil unique et universel capable de résoudre tous les problèmes, mais de proposer un ensemble d'outils qui se compléteraient et travailleraient à des niveaux d'abstraction différents et exploreraient des espaces de conception de largeur différente. Un outil pourrait se charger d'explorer à haut niveau l'espace de conception de l'application. Il proposerait diverses solutions d'implémentations aux outils plus bas niveau. Partant de là, un outil d'exploration architecturale de haut niveau, permettrait de rapidement explorer, pour le choix d'une implémentation de l'application, les architectures reconfigurables hétérogènes (gros grain et grain fin). Dans le cas spécifique du grain fin qui vise des composants commerciaux, il serait enfin possible d'aller plus loin soit en utilisant un système d'estimation basé sur une pré-caractérisation d'opérateurs de base, soit en utilisant un outil générique permettant une exploration architecturale et pourquoi pas la création de fichiers de configuration de l'architecture définie.

L'ensemble d'outils Design Trotter développé à l'Université de Bretagne Sud associé à l'outil Madeo développé à l'Université de Bretagne Occidentale, propose une solution à ce problème d'exploration de l'espace de conception ciblant des architectures reconfigurables. La figure 38 donne un aperçu de la contribution de chacune des parties :



Exploration	Spécification de l'application	Spécification de l'architecture	Référence	Type de résultats	**Outil**
Application	C -> HCDFG	Abstraite	[leMoullec03]	Courbes de compromis ressources/temps	**Design Trotter**
Architecture multi-grain	HCDFG	Fonctionnelle	[Bossuet03]	Répartition des communications	**Design Trotter**
Architecture générique grain fin	EDIF	Structurelle	[Lagadec01]	Placement et routage	**Madeo**
Architecture spécifique grain fin	CDFG	Pré-caractérisation	[Bilavarn03]	Estimation temps et surface	**Design Trotter**

Figure 38 – Contribution de l'ensemble d'outils Design Trotter et de l'outil Madeo à l'exploration de l'espace de conception des architectures reconfigurables

Le prochain paragraphe va présenter les diverses parties de Design Trotter telles que l'estimation système, la projection sur FPGA et l'estimation relative de la répartition des communications dans les architectures reconfigurables.

2.5 Design Trotter

L'ensemble d'outils Design Trotter a été développé dans le but de venir en aide aux concepteurs très tôt dans le flot conception de systèmes sur puce hétérogènes. Sous le terme "systèmes hétérogènes" nous entendons des systèmes regroupant des parties programmables, des parties matérielles fixes et enfin des parties reconfigurables (de grain fin ou de gros grain). Ainsi Design Trotter s'intègre dans la conception conjointe logicielle matérielle comme il en a été le cas lors du projet RNTL EPICURE [Auguin03]. Design Trotter utilise une spécification de haut niveau de l'application en langage C, cette spécification est traduite en un HCDFG (*Hierarchical Control and Data Flot Graph*) partagé par tous les outils de Design Trotter. Ces outils sont aujourd'hui au nombre de quatre : l'estimation système [LeMoullec03a], l'estimation relative ciblant des architectures reconfigurables [Bossuet03a], la projection matérielle ciblant des FPGA [Bilavarn03]. Les deux derniers outils utilisent les résultats, sous forme de courbes de compromis, fournis par le premier. Un quatrième outil [Azzedine02] de plus haut niveau utilise les résultats de l'estimation système pour l'ordonnancement et le partitionnement de tâches dans un système temps réel. Ce dernier outil ne sera pas développé dans la suite.

Ce paragraphe va présenter les différentes parties comprises dans Design Trotter en débutant par la spécification de l'application en HCDFG.

2.5.1 Spécification de l'application : le HCDFG

Pour des questions de convivialité et de rapidité de développement l'utilisateur ne spécifie pas son application directement sous la forme du HCDFG mais en langage C. Cependant une bonne connaissance du graphe interne utilisé pour représenter l'application dans l'ensemble des parties de Design Trotter permet de bien comprendre les algorithmes utilisés. Lors de la spécification de l'application complète en C si celle-ci est partitionnée en plusieurs fonctions. La spécification en langage C de chaque fonction est transformée, via un analyseur synthaxique, en un HCDFG différent, il y a donc autant de HCDFG que de fonctions pour décrire l'application. Le HCDFG permet de représenter sous la forme d'un graphe hiérarchique les fonctions contenant des structures de contrôles (boucles, conditions), des traitements et manipulations de données de type scalaire ou tableau. La hiérarchie d'un HCDFG est composée des éléments suivants (par ordre de granularité croissant) : nœuds élémentaires (nœuds mémoires et nœuds données), DFG et CDFG. Ces différents éléments sont détaillés dans ce qui suit et la figure 39 en donne une représentation graphique.

Les DFG sont des graphes qui ne contiennent que des nœuds élémentaires, des nœuds traitements ou des nœuds mémoires. Ils représentent une séquence d'opérations non conditionnelles. **Les nœuds traitements** contenus dans les DFG représentent soit une opération logique soit une opération arithmétique. Les opérations arithmétiques peuvent être de différentes granularités, par exemple il est possible de représenter un nœud additionneur ou multiplieur mais aussi un nœud multiplieur-accumulateur. **Les nœuds mémoires** représentent des transferts de données. Les données manipulées sont donc explicitement représentées par des nœuds dans le graphe et ne sont pas associées comme

dans de nombreuses représentations aux arcs. Dans le cas des données multidimensionnelles (tableaux, vecteurs), les mécanismes d'adressage sont explicitement représentés dans le graphe à l'aide de DFG d'indices. Les paramètres principaux des nœuds mémoires sont le format de la donnée, le sens de transfert (lecture ou écriture) et la valeur de la donnée.

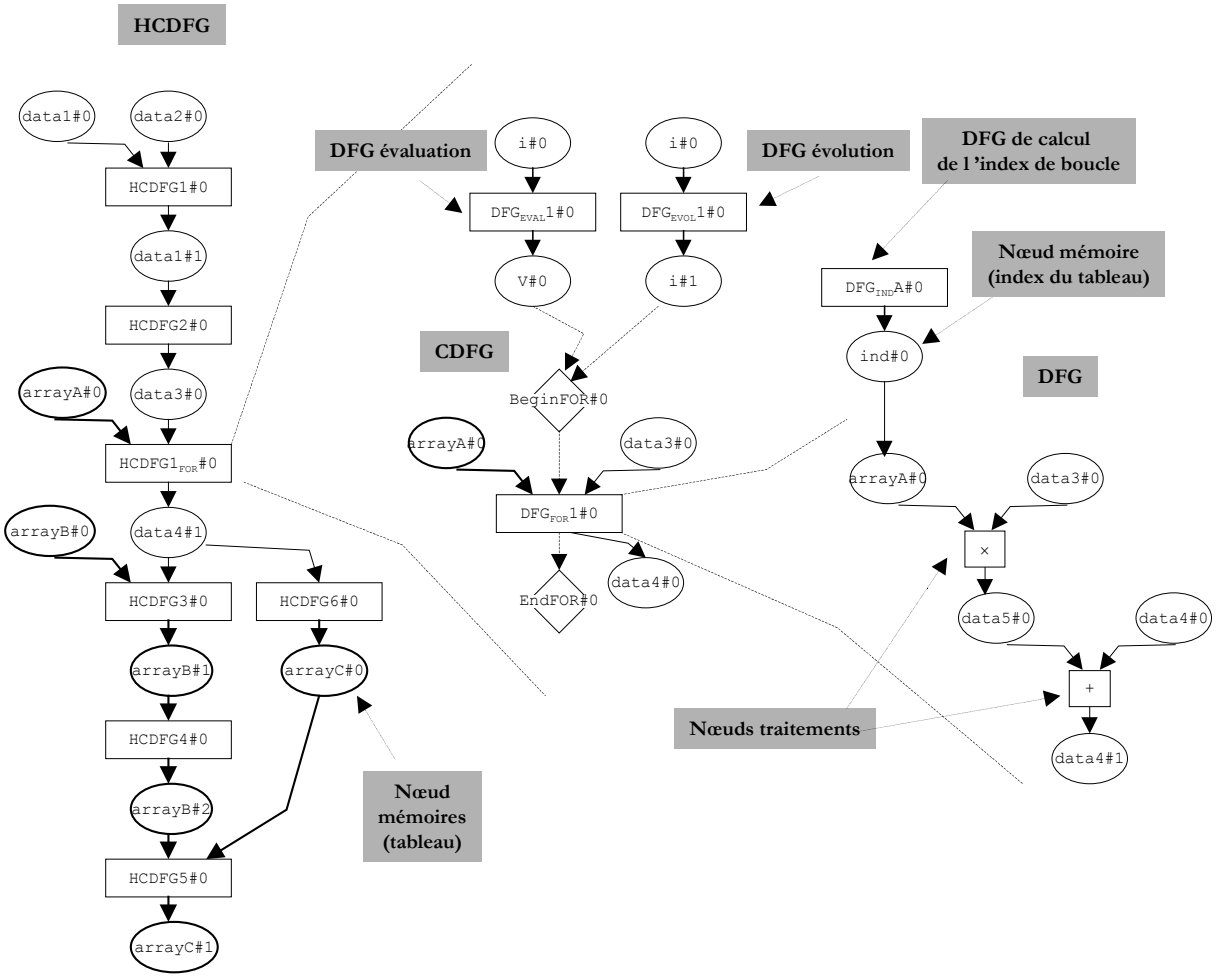


Figure 39 – Représentation graphique du HCDFG

Les CDFG correspondent aux contrôles intervenant dans la spécification de haut niveau en langage C. Un CDFG est donc associé dans tous les cas à un graphe d'évaluation de la condition pour connaître l'état du contrôle et orienter vers le traitement à effectuer. Dans le cas d'une structure *pour* (*for*) pour laquelle il est nécessaire de faire évoluer un indice (seules les évolutions linéaires sont acceptées), un graphe d'évolution de l'indice est alors associé en plus du graphe d'évaluation de la condition. Le CDFG est également composé d'un

nœud contrôle qui indique le type de la structure (if, switch-case, while, do-while et for). Le CDFG comporte aussi tous les graphes nécessaires à la réalisation des différentes possibilités en fonction de l'évaluation de la condition, par exemple deux graphes dans le cas d'une structure if pour décrire le cas où la condition est vraie et le cas où la condition est fausse.

Les HCDFG représentent le dernier niveau hiérarchique de graphe, ils peuvent comporter d'autres HCDFG, des CDFG ou des DFG. Ils permettent de représenter la hiérarchie de l'application, c'est à dire l'imbrication des structures de contrôles et des graphes s'exécutant en série ou en parallèle.

Les arcs sont les liens nécessaires entre les nœuds et les graphes hiérarchiques. Les arcs représentent les dépendances entre les données scalaires et multidimensionnelles ainsi que les dépendances d'ordre entre les opérations sans transfert mémoire (par exemple une opération de calcul d'indice avant l'accès au tableau).

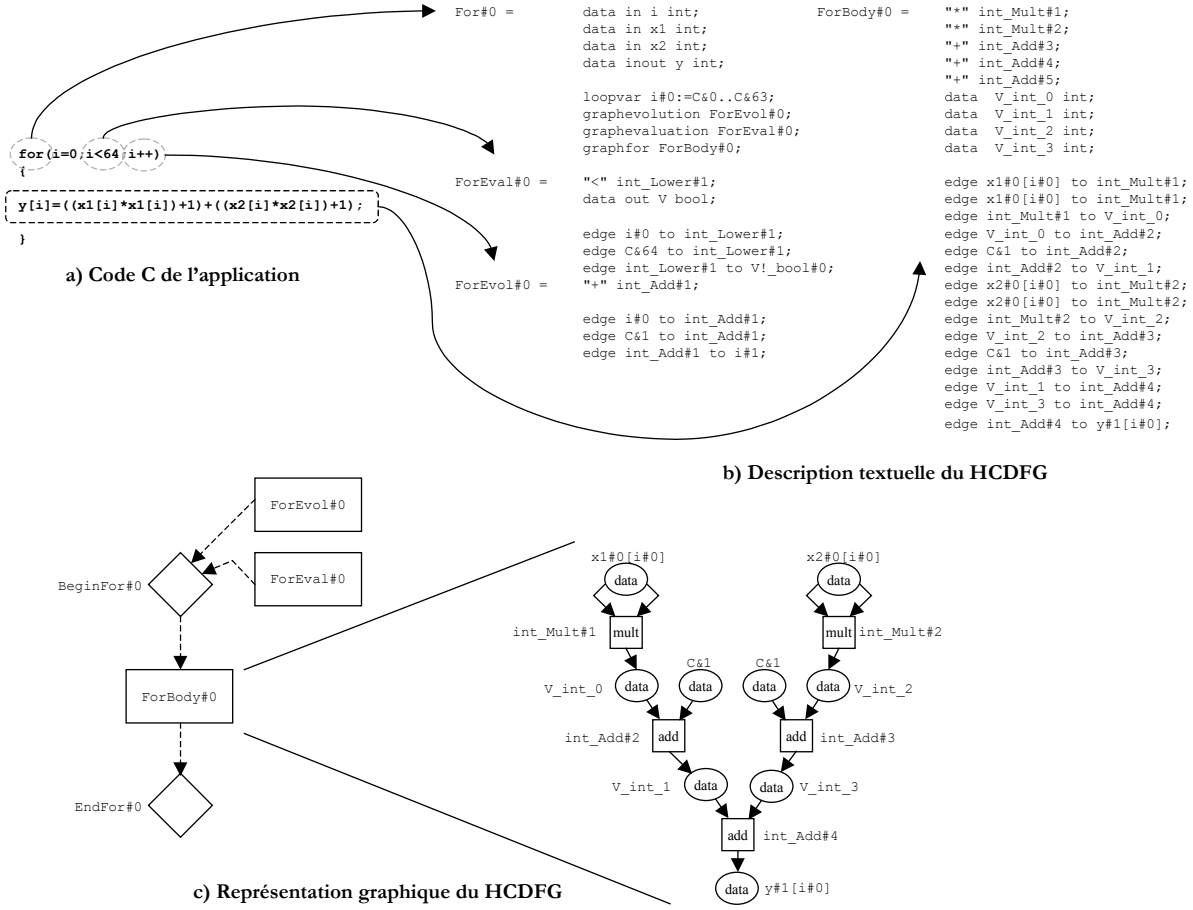


Figure 40 – Passage du langage C au HCDFG : a) spécification langage C, b) représentation textuelle du HCDFG, c) représentation graphique du HCDFG

Pour construire le HCDFG à partir de la spécification en C la structure de données issue du parser est parcourue avec un algorithme de type *depth-first search*. Un HCDFG ou un CDFG est créé lorsqu'un nœud conditionnel est trouvé dans le niveau de hiérarchie supérieur. Lorsqu'il n'y a plus de nœud conditionnel un DFG est construit. La figure 40 donne la réalisation d'un HCDFG à partir d'un exemple simple en langage C (boucle for), via la représentation grammaticale ou représentation textuelle et via la représentation graphique.

2.5.2 L'estimation système

L'estimation système est la première partie de Design Trotter. Elle a pour but de guider le concepteur dans le choix de réalisation de l'application qu'il développe [LeMoullec03d], en particulier en choisissant un ordonnancement et donc un niveau de parallélisme ou un déroulage de boucle. L'estimation système, comme indiqué sur la figure 41, se déroule en deux principales phases : la caractérisation [LeMoullec03b] qui ne dépend que de la spécification de l'application, et l'estimation intra-fonction [LeMoullec03c] qui s'appuie sur les résultats de la caractérisation (les métriques), sur la spécification de l'application et sur un modèle abstrait de l'architecture. L'utilisateur analyse les résultats de chacune des deux principales phases du flot, ainsi il s'enrichit des informations délivrées à tous les niveaux de l'estimation système.

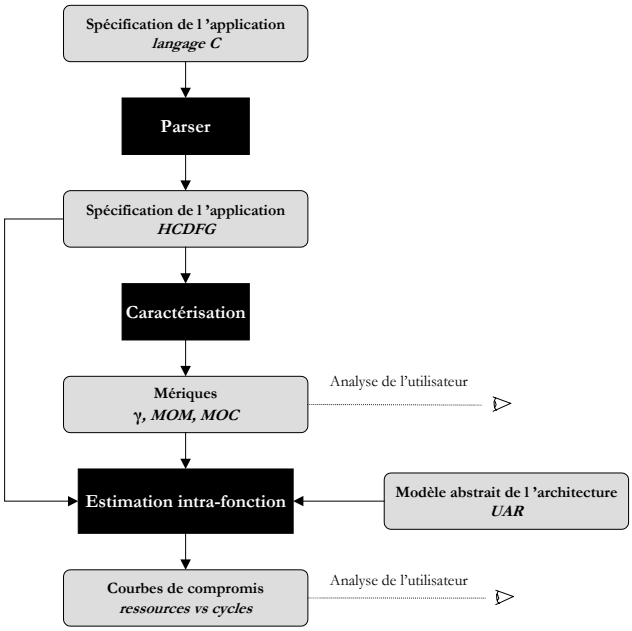


Figure 41 – flot de l'estimation système.

La caractérisation [LeMoullec03b] a pour but de transformer la spécification de l'application en métriques qui seront exploitées par la suite pour orienter la conception. Les métriques vont définir l'orientation de l'application, c'est à dire mettre en avant la

prédominance d'un certain type d'opération comme les traitements, les échanges de données avec la mémoire ou encore les contrôles. Ainsi les métriques indiquent si l'application est orientée traitement, mémoire ou contrôle. Cette caractérisation de l'application permet par la suite un ordonnancement adapté à l'application en traitant en premier le type d'opération prédominant. Mais les métriques donnent également la possibilité de ranger les fonctions par ordre de criticité, permettant ainsi de traiter prioritairement les fonctions de l'application qui sont les plus critiques. Ces fonctions critiques sont celles qui nécessitent les réalisations les plus parallèles et sont donc les plus coûteuses en ressources. Les fonctions les moins critiques viendront utiliser ces mêmes ressources lorsqu'elles seront à leur tour traitées. Enfin les métriques renvoient une expression du parallélisme potentiel de l'application qui informe le concepteur des gains potentiels que pourrait apporter une réalisation plus parallèle de l'application.

Les métriques principales calculées lors de la caractérisation sont au nombre de trois : la métrique de criticité, la métrique d'orientation mémoire et la métrique d'orientation contrôle. Une définition simple de chacune des métriques peut être donnée pour les DFG mais des règles de combinaisons plus complexes permettent de calculer ces métriques dans le cas particulier des CDFG et HCDFG [LeMoullec03b].

La métrique de criticité (γ) indique le parallélisme spatial moyen disponible à un certain niveau hiérarchique. Cette métrique permet la classification des fonctions de l'application en fonction de leur criticité à savoir leur capacité à exploiter un parallélisme moyen. Les fonctions avec un fort γ sont considérées comme appropriées aux architectures permettant une réalisation avec un parallélisme matériel, celles avec un faible γ sont plus séquentielles dans ce cas les accélérations ne seront le fait que de l'exploitation d'un parallélisme temporel (pipeline). Cette métrique est définie par la relation suivante :

$$\gamma = \frac{NbOp}{CheCrit}$$

NbOp est le nombre d'opérations de type traitements ou transfert de données dans le graphe et *CheCrit* est le chemin critique du graphe. Le chemin critique d'un DFG est donné par la plus longue chaîne d'opérations séquentielles de celui-ci.

La métrique d'orientation mémoire (MOM) indique la fréquence des accès mémoires dans le graphe. Contrairement à γ cette métrique est normalisée dans l'intervalle [0 ; 1]. Une fonction qui a un MOM proche de 1 est orientée mémoire, c'est à dire que les opérations de type transfert de données sont prédominantes. Dans ce cas la structure et la hiérarchie mémoire doivent être particulièrement soignées. Cette métrique est définie par la relation suivante :

$$MOM = \frac{NbMemAcc}{NbMemAcc + NbOpTrait}$$

NbMemAcc est le nombre d'accès mémoire dans le graphe ce qui traduit le nombre de transfert de données, et *NbOpTrait* est le nombre d'opérations de traitement dans le graphe.

La métrique d'orientation contrôle (MOC) dans le cas des CDFG (puisque les DFG n'ont pas de nœud contrôle) indique la fréquence des opérations de contrôle. Comme la

métrique MOM, la métrique MOC est normalisée dans l'intervalle [0 ; 1]. Une fonction avec un MOC proche de 1 est orientée contrôle, les opérations de contrôle sont prédominantes. Il faut donc dans ce cas prévoir une réalisation qui facilite les contrôles. Cette métrique est définie par la relation suivante :

$$MOC = \frac{NbOpContr}{NbOpContr + NbMemAcc + NbOpTrait}$$

NbOpContr est le nombre d'opérations de contrôle (telles que <=, <, >, >=, !=, ==) dans le graphe, *NbMemAcc* et *NbOpTrait* sont comme dans le cas de MOM le nombre d'accès mémoire et le nombre d'opérations de traitement dans le graphe.

Une fonction qui n'est ni orientée mémoire (MOM proche de 0), ni orientée contrôle (MOC proche de 0) est orientée traitement. Les opérations de traitement sont dans ce cas prioritaires, l'architecture cible doit donc disposer de structures de traitement performantes.

D'autres métriques secondaires sont utilisées comme par exemple la métrique DRM qui spécialise la métrique MOM afin de séparer les accès à la mémoire locale et globale. Toutes les métriques sont utilisées dans la phase d'estimation intra-fonction. Elles permettent de choisir entre trois possibilités l'ordonnancement le plus adapté à l'application : ordonnancement prioritaire des traitements, ordonnancement prioritaire de la mémoire et ordonnancement mixte.

L'estimation intra-fonction [LeMoullec03c] cette étape a pour objectif de produire des courbes de compromis "ressources nécessaires/nombre de cycles alloués" pour les fonctions de l'application à estimer et pour tous les niveaux de hiérarchie de la spécification. Les courbes de compromis reflètent donc les possibilités d'exploration du parallélisme de l'application. L'estimation intra-fonction donne une courbe de compromis par type de ressources. Chaque proposition de temps alloué à la fonction donne un point sur chaque courbe ce qui correspond à une réalisation architecturale possible de l'application. Il y a donc autant de réalisations possibles que de points sur les courbes de compromis. Pour réaliser les ordonnancements, qui sont tous basés sur des algorithmes de type *list scheduling*, l'estimation système a besoin d'une caractérisation, même abstraite, de l'architecture finale, c'est le but du modèle abstrait utilisé : le fichier UAR (*User Abstract Rules*).

Le fichier UAR donne un minimum d'informations concernant les ressources disponibles sur l'architecture finale. L'utilisateur définit l'unité de temps abstraite avec laquelle il va caractériser temporellement les opérateurs disponibles sur l'architecture finale, par exemple un nombre de cycles. Pour la partie traitement et contrôle, l'utilisateur doit définir le type de ressource disponible : UAL, MAC, additionneur, multiplieur, comparateur, fonction logique… Le grain des ressources peut donc être très variable. A chaque type d'opérateur est associé un nombre de cycles et un ensemble d'opérations supportées. En ce qui concerne la mémoire, l'utilisateur définit le nombre de niveaux de la hiérarchie mémoire et le nombre de cycles associés à chaque type de transfert (lecture/écriture).

Les résultats finaux correspondant à une fonction complète sont issus de la combinaison hiérarchique des résultats des différents graphes composant cette fonction. Les résultats

des CDFG sont combinés en traitant les CDFG en parallèle puis en série, de même pour les résultats des HCDFG.

Une fois les courbes de compromis établies pour la fonction, le concepteur peut choisir une solution, en positionnant soit une contrainte de temps soit une contrainte matérielle. Par exemple, sur la figure 42 qui donne un cas typique de courbes de compromis, si le concepteur souhaite une exécution rapide de sa fonction il connaîtra les ressources à mettre en œuvre pour réaliser l'application et naturellement il sera amené à envisager une implémentation matérielle de celle-ci. Par exemple en choisissant l'allocation minimum de temps qui est de 1893 dans cet exemple, il obtient comme résultat que 20 ALU et autant de multiplieurs sont nécessaires à la réalisation de la fonction estimée. Au contraire s'il choisit de réduire les coûts matériels pour envisager une solution d'implémentation séquentielle et logicielle (système programmable) il connaîtra l'estimation du temps d'exécution de la fonction. Par exemple, sur la figure 40 la réalisation de l'application avec une ALU et un multiplieur se fera en un temps de 20 355 cycles. Ainsi pour l'ensemble de l'application le concepteur peut choisir de façon manuelle le partitionnement des fonctions de l'application pour une implémentation sur des ressources matérielles ou logicielles. Pour affiner le partitionnement il est nécessaire de disposer de projection matérielle et logicielle, c'est à dire d'estimateurs de haut niveau qui permettent de rapidement estimer les performances de la fonction étudiée sur les différentes cibles. Aujourd'hui Design Trotter ne dispose pas encore de projection logicielle, mais dispose d'une projection matérielle ciblant des circuits reconfigurables de grain fin de type FPGA. Ce processus est présenté dans le prochain paragraphe.

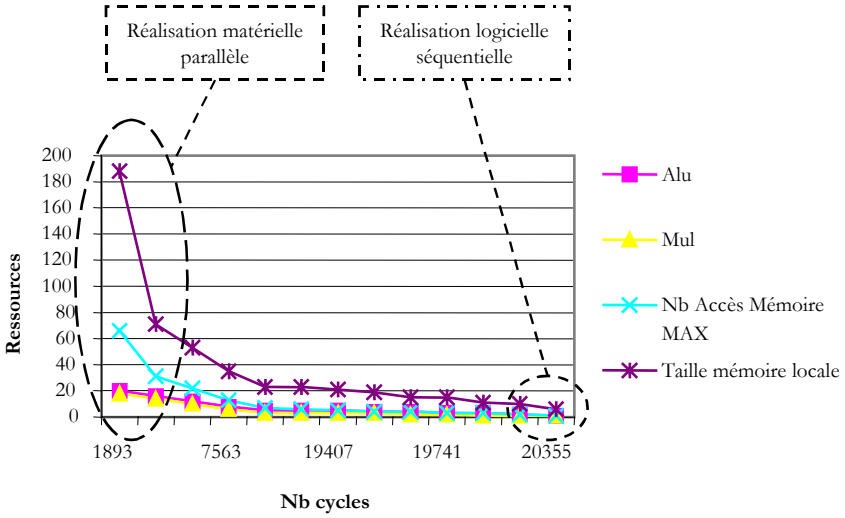


Figure 42 – Exemple typique de courbes de compromis d'une fonction

2.5.3 La projection matérielle sur FPGA

La projection matérielle sur FPGA [Bilavarn02] a pour but d'effectuer très rapidement une estimation de la surface et du temps d'exécution de la partie de l'application à implanter en

fonction de la solution d'implémentation choisie, c'est à dire en fonction du point choisi sur les courbes de compromis. La projection matérielle prend donc en entrée les résultats de l'estimation système comme schématisé sur la figure 43. La méthode utilisée pour effectuer les estimations est très dépendante de la technologie puisqu'elle se base sur des pré-caractérisations d'opérateurs de base qui sont susceptibles d'être synthétisés. Ces caractérisations viennent enrichir et spécialiser les fichiers UAR. Aujourd'hui il existe des pré-caractérisations pour des circuits de la famille Xilinx Virtex et pour des circuits de la famille Altera Apex. Cependant l'augmentation de la bibliothèque de caractérisations n'est pas complexe.

La figure 43 schématise le principe de la projection matérielle. Une solution issue de l'estimation système renseigne la projection matérielle sur les ressources nécessaires pour réaliser la fonction estimée. Dans le cas de la figure 43 la solution indique qu'il est nécessaire de synthétiser trois multiplieurs et deux additionneurs. La projection matérielle va traduire ces besoins en opérateurs par des besoins en ressources logiques configurables du composant cible, ici un FGPA Virtex Xilinx. Il en résulte que 116 cellules logiques (*slices* dans la terminologie Xilinx) sont nécessaires pour réaliser les traitements et une cellule dédiée (bloc de mémoire RAM) est nécessaire pour réaliser les besoins en termes de ressources de mémorisation. Comme les ressources sont pré-caractérisées l'estimation temporelle donnée en cycles par l'estimation système est traduite en nanoseconde par la projection matérielle.

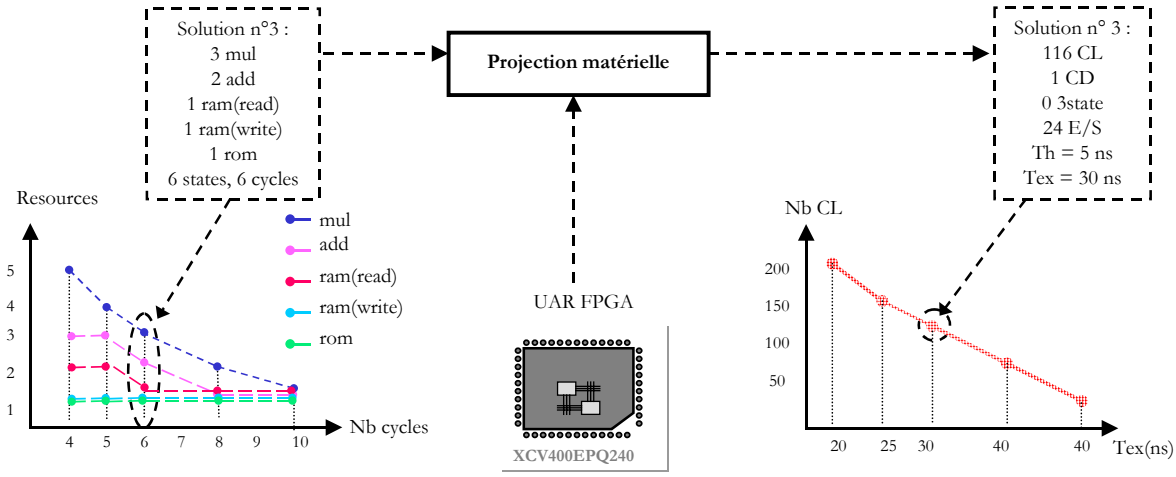


Figure 43 – Principe de la projection matérielle

Les estimations sont obtenues aujourd'hui en moins d'une minute en moyenne et elles donnent des résultats avec une précision moyenne de 15 % [Bilavarn03] par rapport aux mesures. Il est à noter que l'estimation de la surface dans les FPGA est donnée en nombre d'éléments reconfigurables de base utilisés et en nombre d'éléments dédiés utilisés (blocs de mémoires reconfigurables, multiplieurs, blocs DSP). Le dernier point montre que cette méthode prend en compte, grâce à sa grande flexibilité, les toutes dernières évolutions des circuits FPGA.

La projection matérielle ainsi effectuée donne des résultats d'estimation très rapidement par rapport au temps nécessaire à la synthèse de la fonction. Mais ce processus ne permet pas à priori d'explorer un large espace de conception puisque celui-ci est réduit au nombre de circuits pré-caractérisés et donc à un sous-ensemble des circuits commerciaux. Il serait intéressant, à partir des résultats de l'estimation système, de réaliser une exploration de l'espace de conception des architectures reconfigurables sans être dépendant de la technologie. C'est ce que propose le troisième outil de Design Trotter et qui est l'objet de la thèse proposée dans ce document.

2.5.4 L'estimation relative ciblant des architectures reconfigurables

Le but de cette partie de Design Trotter est d'ouvrir l'exploration de l'espace de conception vers les architectures reconfigurables dans leur ensemble. Le principe de base est d'effectuer des estimations relatives de performance de fonctions de l'application sur des architectures reconfigurables. Les estimations sont relatives car il ne s'agit pas de donner des estimations précises qui ne peuvent être obtenues qu'au prix d'une réduction drastique de l'espace de conception et d'une spécialisation des estimateurs. Il s'agit ici de comparer entre eux des choix architecturaux portant essentiellement sur la hiérarchie de l'architecture, sur la taille des clusters ainsi que sur le type et le nombre de ressources embarquées (ressources de traitement et ressources de mémorisation).

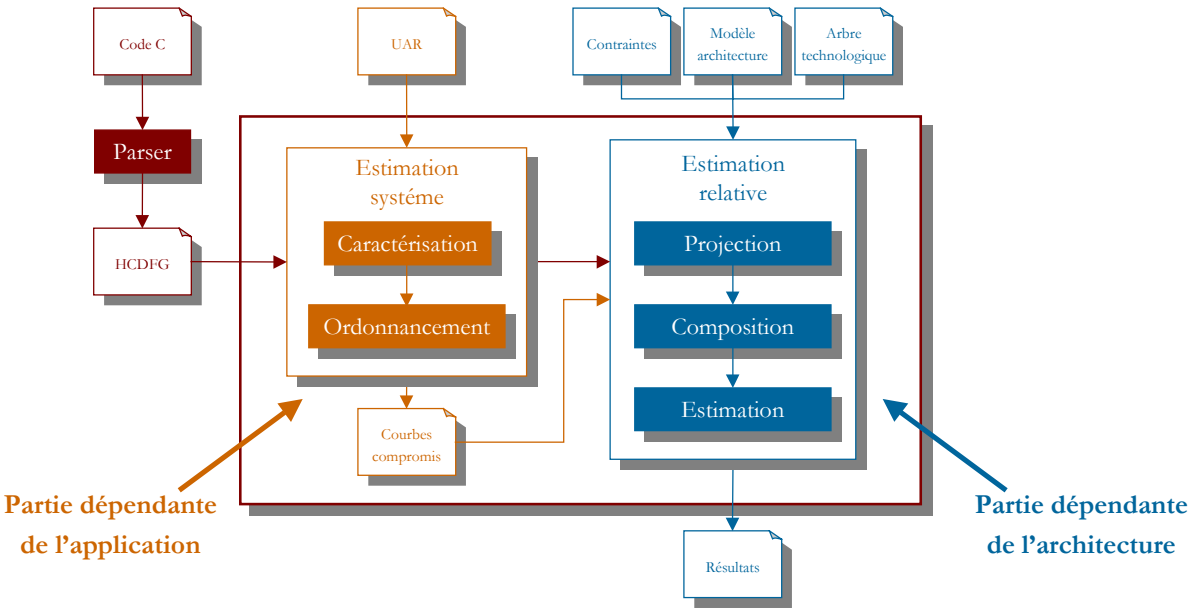


Figure 44 – Flot d'estimation relative rattachée à l'estimation système

L'exploration de la réalisation de l'application est effectuée préalablement lors de l'estimation système. Tout comme pour la projection physique visant des FPGA, l'estimation relative ciblant des architectures reconfigurables utilise en entrée les résultats de l'estimation système.

Le flot d'estimation relative, figure 44, prend donc en entrée les courbes de compromis issues de l'estimation système ainsi que la spécification HCDFG de l'application. En prenant en compte les deux parties, estimation système et estimation relative, le flot obtenu contient une partie d'exploration de l'application et une partie d'exploration de l'architecture reconfigurable cible. Pour cette dernière exploration l'estimation relative a nécessairement besoin d'une modélisation de l'architecture reconfigurable cible. Pour répondre à ce besoin une modélisation fonctionnelle a été développée [Bossuet02a]. Elle est présentée dans le chapitre 3 de ce document. Le reste du flot et en particulier les algorithmes utilisés pour estimer les performances sont présentés dans le chapitre 4 de ce document. Ces algorithmes utilisent un principe de base qui est d'estimer la répartition des communications dans les différents niveaux de hiérarchie de l'architecture en imaginant comment ces communications seraient établies par un outil de synthèse et placement-routage [Bossuet03a]. Les deux prochains chapitres vont présenter les spécifications en entrée de l'estimation système, puis les algorithmes mis en œuvre et enfin la méthode d'exploration architecturale basée sur les résultats de l'estimation relative.

2.5.5 Interface graphique Design Trotter

L'ensemble des outils présentés ci-dessus ; le parser du C vers HCDFG, l'estimation système, la projection physique sur FPGA et l'estimation relative pour architectures reconfigurables, ont fait l'objet d'un développement informatique. Le langage de développement choisi est le Java, le choix a pris en compte la portabilité du code donc de l'outil. Une interface graphique multi-fenêtres permet à l'utilisateur de passer de façon progressive d'un outil vers un autre. Des captures d'écrans, figure 45, donnent une vision des outils.

2.6 Conclusion

Ce chapitre nous a montré que l'exploration de l'espace de conception est un processus complexe avec de nombreuses alternatives. En outre l'exploration peut se faire sur l'application, sur l'architecture virtuelle de celle-ci ou sur l'architecture matérielle cible. Chaque exploration pouvant intervenir à différents niveaux et explorer tout ou partie de l'espace de conception.

Les exemples présentés dans ce chapitre montrent qu'il y a un manque d'outils spécialement développés pour effectuer une exploration de l'espace de conception à différents niveaux du flot de conception dans le cadre des architectures reconfigurables (grain fin et gros grain). De plus nous avons vu qu'il n'existe pas d'outils d'exploration génériques visant les architectures reconfigurables de gros grain (et hétérogènes) mais uniquement des outils spécifiques à une cible architecturale. Le LESTER propose une solution appelée Design Trotter qui regroupe plusieurs travaux. Ces travaux réunis donnent à l'utilisateur une grande flexibilité d'exploration en séparant l'exploration au niveau de l'application de celle au niveau des architectures. Dans ce dernier cas différents niveaux d'estimation sont proposés ; du niveau abstrait pour comparer rapidement les architectures entre elles jusqu'au niveau précis structurel pour affiner l'exploration à une famille

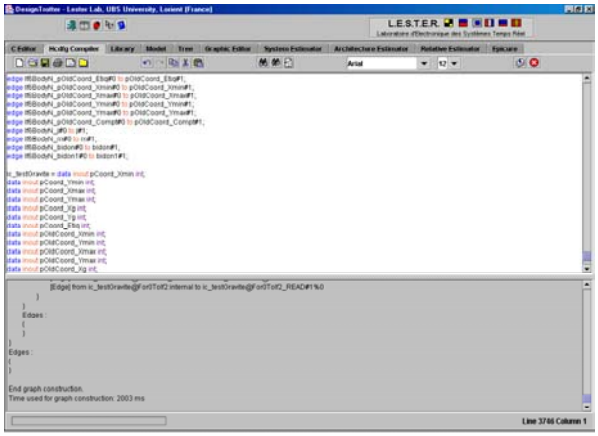
d'architectures (une famille d'architectures correspond à un chemin entier dans la classification des architectures proposée à la figure 11 du paragraphe 1.2.1). Le tableau 3 reprend les principales caractéristiques d'exploration (algorithmique et architecturale) de Design Trotter. Ce tableau est de même forme que le tableau 2 reprenant au paragraphe 2.4.4 les caractéristiques des flots d'exploration présentés en exemple. On peut voir par comparaison, que grâce aux trois outils d'exploration qu'il englobe, Design Trotter propose une large gamme d'exploration suivant plusieurs méthodes en fonction des cibles architecturales. Il est intéressant de se pencher sur l'exploration de l'espace de conception des architectures reconfigurables de gros grain et grain fin qui est le résultat des travaux de recherches décrit dans ce document.

Outils	Cibles architecturale	Spécification des applications	Spécification des architectures	Type d'exploration	Résultat de l'exploration
Design Trotter	Architecture RTL virtuelle	HCDFG	Modèle abstrait	Algorithmique Multi-objectifs (puissance, surface, temps) Automatique et heuristique	Information de conception de l'application (parallélisme et déroulage de boucle)
	Architectures reconfigurables gros grain et grain fin	HCDFG caractérisation	Modèle fonctionnel	Architecturale Mono-objectif (puissance) Semi-automatique et heuristique	Une architecture parmi un ensemble et des informations de conception (taille cluster, type des éléments de base, répartition des communications)
	FPGA	CDFG	Prè-caractérisation	Architecturale Multi-objectifs (puissance, surface, temps) Manuelle et heuristique	Une architecture parmi un ensemble (famille de FPGA)

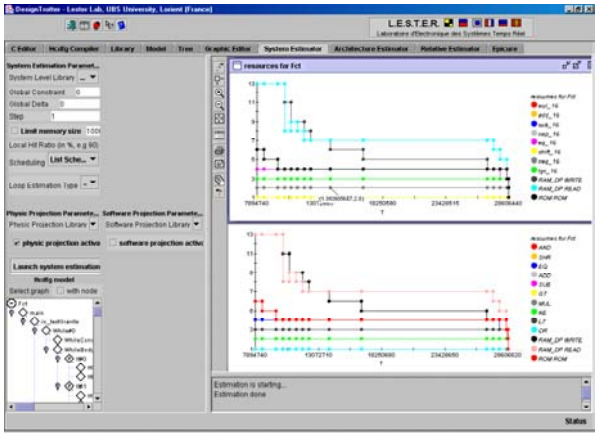
Tableau 3 – Comparaison d'outils d'exploration de l'espace de conception des architectures reconfigurables

L'approche proposée contrairement aux travaux de [Moritz98] et [Nageldinger01] ne se focalise pas sur une seule architecture gros grain. Les résultats de l'exploration donnent des informations de conception de l'architecture. En particulier des informations concernant la hiérarchie de l'architecture et la taille des éléments hiérarchiques (ou clusters). Mais de plus, une information concernant la répartition des communications dans les différents niveaux de hiérarchie de l'architecture permet à l'utilisateur d'orienter son choix de structure et de support de communication.

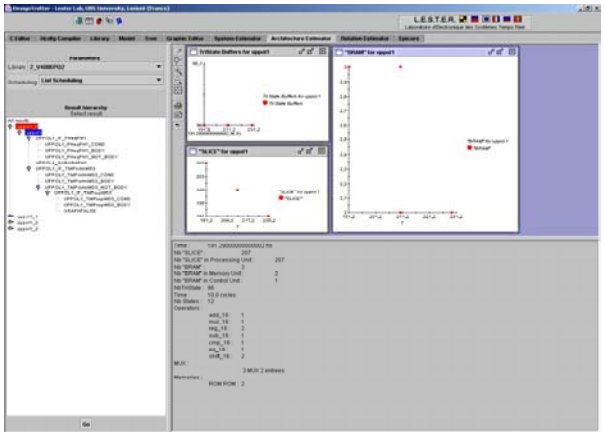
Les deux prochains chapitres de ce document vont se focaliser sur le système d'exploration de l'espace de conception des architectures reconfigurables, basé sur une modélisation abstraite et fonctionnelle des architectures et sur une estimation relative de la répartition des communications dans ces architectures.



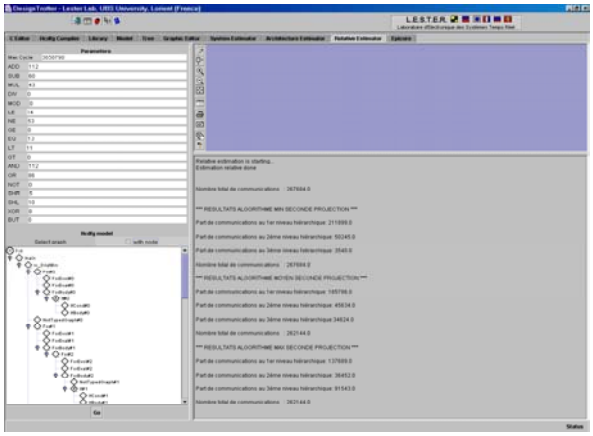
a) La spécification HCDFG de l'application



b) L'estimation système



c) La projection matérielle sur FPGA



C) L'estimation relative pour architectures reconfigurables

Figure 45 – Captures d'écrans des différentes parties de Design Trotter

Chapitre 3

Spécifications en entrée de l'estimation relative dans Design Trotter

L'outil Design Trotter apporte au concepteur une aide à la conception et ceci très tôt dans le flot de conception des systèmes hétérogènes sur puce. Aussi le niveau d'abstraction des spécifications en entrée doit être élevé. Ces spécifications sont celles de l'application et celles des architectures. Dans le cas de l'estimation relative, les architectures sont reconfigurables. La spécification de ces architectures est basée sur un modèle fonctionnel permettant la représentation d'un large ensemble architectural. Les applications sont, d'une façon commune pour tous les outils de Design Trotter, spécifiées par l'utilisateur en langage C puis cette spécification est transformée via un parser en spécification HCDFG. L'estimation relative prend pour métrique principale la répartition des communications dans l'architecture, aussi une modification de la spécification HCDFG a lieu afin de faire ressortir les communications entre les nœuds du graphe. Le HCDFG est donc transformé en une représentation interne appelée ACG.

Ce chapitre va explorer les spécifications en entrée de l'outil concernant les applications et les architectures. Pour ces dernières il va dans un premier temps proposer l'étude de l'impact des différentes ressources de l'architecture sur la consommation de puissance des architectures reconfigurables de grain fin. Puis il proposera d'extrapoler les résultats aux architectures à plus gros grain. En se basant sur cette étude le chapitre se poursuivra en proposant un modèle fonctionnel d'architecture reconfigurable ainsi qu'un modèle de graphe de communication l'ACG. Ce graphe permet de mettre en valeur les communications entre les nœuds du HCDFG de l'application étudiée. Ces deux éléments sont centraux pour la définition des spécifications d'entrée.

3.1 Besoins et objectifs

Dans le paragraphe 2.5.4 du chapitre précédent, la présentation rapide de la partie estimation relative de Design Trotter a montré qu'il est nécessaire, dans le but de comparer entre elles des architectures reconfigurables, de disposer d'une modélisation de ces architectures afin de les spécifier en entrée du flot. De même l'application initialement en C doit être disponible dans une spécification apte à favoriser le développement d'algorithmes d'estimation. Cependant en plus de la spécification de l'application l'estimation relative doit obtenir plus d'informations sur l'application en particulier sur la réalisation de celle-ci (comme le niveau de parallélisme par exemple). Ces informations supplémentaires sont tirées des résultats de l'estimation système.

Pour spécifier l'application Design Trotter utilise une transformation de la spécification en langage C en un graphe HCDFG. L'estimation relative utilise ce formalisme d'entrée et s'intègre ainsi aux autres outils de Design Trotter. De cette façon elle peut profiter de leurs résultats (en particulier ceux de l'estimation système). Cependant nous allons voir que l'estimation relative s'appuie sur une métrique principale qui est la répartition des communications au sein de l'architecture. Il est donc intéressant de modifier la spécification HCDFG de l'application afin de mettre en relief les caractéristiques de l'application qui s'orientent vers la métrique choisie.

Comme nous l'avons vu précédemment l'estimation relative conduit à l'exploration de l'espace de conception des architectures reconfigurables. Pour cela elle utilise en entrée les résultats de l'estimation système, notamment les courbes de compromis. Ces courbes donnent en chacun de leurs points une solution d'implémentation de la fonction traitée de l'application à développer. L'estimation relative débute lorsque le choix d'une solution est réalisé et donné par l'utilisateur. Nous verrons dans le prochain chapitre que ce choix peut être remis en cause sous certaines conditions dès le début du flot. Une solution fournit, outre une estimation du temps d'exécution (en nombre de cycles abstraits), les ressources qu'il est nécessaire d'implémenter sur l'architecture afin de réaliser la fonction étudiée. L'estimation système enrichie donc la spécification de l'application avec ses résultats. La spécification de l'application pour l'estimation relative est un donc un HCDFG complété par des informations concernant la réalisation de l'application (nombre de ressources nécessaires).

Etant donné le haut niveau d'abstraction et le principe même de l'estimation relative qui a pour but de comparer des solutions entres elles, la spécification en entrée des architectures reconfigurables doit permettre de modéliser un large espace architectural. Comme le premier chapitre l'a montré, l'espace de conception des architectures reconfigurables est très large. Vouloir un modèle capable de représenter l'ensemble des architectures semble complexe voir difficilement faisable. Aussi pour arriver au modèle que nous proposons nous avons d'abord étudié l'impact des différentes ressources des architectures de grain fin (FPGA) sur la consommation de puissance. Effectivement la surconsommation de puissance des FPGA par rapport aux ASIC est un inconvénient de poids pour leur intégration dans les systèmes embarqués [Garcia00]. Il est donc indispensable de se

concentrer sur ce problème afin d'analyser finement la consommation de puissance des architectures reconfigurables. Dans [Mudge01] l'auteur montre que l'objectif de la réduction de la consommation est crucial pour les nouvelles architectures. Effectivement leurs performances temporelles sont très bonnes aujourd'hui grâce à l'important parallélisme spatial qu'elles proposent. L'étude que nous proposons est basée sur l'utilisation de carte de tests avec des circuits commerciaux ainsi que d'outils de synthèse et d'estimation. Des études sur ce sujet ont également été publiées et permettent de comparer nos résultats et d'affiner nos conclusions. Il serait évidemment très intéressant après l'étude de la consommation de puissance des FPGA de réaliser la même étude avec des architectures de gros grain. Malheureusement les architectures de gros grain ne sont pas ou peu disponibles physiquement tout comme les outils nécessaires à leur utilisation. De ce fait il est donc pratiquement impossible de disposer d'une étude de la répartition de la consommation de puissance dans les architectures reconfigurables de gros grain. Cependant les architectures reconfigurables de grain fin et de gros grain ont des caractéristiques proches. Par exemple un grand parallélisme matériel spatial ce qui conduit à de grandes surfaces de silicium. Elles utilisent de nombreuses ressources de communication pour relier les éléments internes. Ces ressources représentent une part importante de la surface des architectures. Aussi nous verrons que dans le cas des FPGA, les ressources de routage consomment la part la plus importante de la puissance. Nous généraliserons donc les résultats obtenus pour les FPGA aux architectures de plus gros grain par extrapolation. Bien sûr cette généralisation entraîne un certain nombre d'hypothèses. Ces hypothèses sont sources d'erreurs sur les estimations que nous faisons par la suite. Cependant comme nous avons choisi de comparer les architectures entre elles, si l'erreur est relative, c'est à dire constante sur l'ensemble des architectures estimées, alors elle est éliminée lors de la comparaison.

L'évaluation de l'impact des différentes ressources des architectures reconfigurables sur leur consommation de puissance va nous permettre de montrer l'impact prédominant des ressources de routage. Nous donnerons également plusieurs conclusions sur ce point qui nous permettrons de justifier les choix postérieurs.

3.2 Evaluation de l'impact des différentes ressources des architectures reconfigurables de grain fin

3.2.1. Résultats d'études sur le sujet

Il existe de nombreuses études portant sur la caractérisation en puissance des différentes ressources qui composent les circuits reconfigurables à grain fin que sont les FPGA. Cependant les résultats de ces études sont différents dans les proportions qu'ils donnent sur la répartition de la consommation de puissance. Devant ce fait nous avons aussi réalisé des études [Rouxel00] et [Elleouet03] afin de mieux comprendre les résultats proposés par les études antérieures. Le résultat le plus important que nous ayons obtenu est qu'il n'est

pas possible de donner une répartition fixe et universelle de la consommation de puissance des différentes ressources des FPGA. Cette répartition dépend de caractéristiques dues à l'application telles que :

- le taux de commutation des données en entrées,
- le nombre d'entrées/sorties utilisées,
- la norme de communication de ces entrées-sorties (induisant leur tension d'alimentation),
- la fréquence de fonctionnement,
- le nombre d'horloges utilisées,
- le taux de remplissage du circuit (en nombre d'éléments reconfigurables de base),
- les options des algorithmes de synthèse et de placement-routage utilisées (telles que l'optimisation en vitesse ou en surface).

Certaines caractéristiques propres au circuit modifient aussi cette répartition comme la technologie qui par exemple induit la valeur des capacités des transistors et la tension d'alimentation du cœur. Cependant, malgré toutes ces caractéristiques qui influent sur la répartition de la consommation de puissance, un résultat ressort de toutes les études : les ressources de routage (des signaux et des horloges) sont très majoritairement les ressources qui consomment le plus d'énergie.

Le tableau 4 donne un résumé des résultats des principales études sur ce sujet. Il est à noter que la puissance considérée est dans tous les cas la puissance dynamique. La puissance statique représente de 5 à 20 % de la puissance totale consommée pour ce type de circuit [Shang02]. Les ressources considérées sont les ressources de routage, les ressources de transport des horloges, les entrées-sorties et les ressources logiques. Les mémoires ne sont pas considérées dans le tableau 4 mais elles seront évoquées lors de l'étude des conclusions des travaux présentés dans le tableau. Seules les études [Rouxel02] et [Shang02] donnent, de façon très précise, les informations concernant le taux d'activité des données en entrée (Ta), le taux d'occupation (To) en pourcentage de ressources logiques de base (ou slices dans la terminologie Xilinx) ainsi que la fréquence de fonctionnement du circuit. Ces deux études sont aussi celles qui s'intéressent aux FPGA récents tels que les Xilinx Virtex et Virtex-II. La dernière étude [Poon02] est une étude plus théorique basée sur la modélisation physique de l'outil VPR (voir paragraphe 2.4.2), mais les résultats obtenus sont proches des résultats obtenus avec de véritables circuits puisque la modélisation est issue d'une caractérisation P-Spice d'un circuit réel (Xilinx XC 4000).

Le tableau 4 nous permet de confirmer que dans tous les cas les ressources de routage sont les ressources les plus consommatrices de puissance, surtout si le transport des horloges y est ajouté. Leur consommation de puissance dépasse largement plus de la moitié de la consommation totale. Par exemple dans [Rouxel02] la consommation de puissance des ressources de routage représente 58% de la consommation totale avec un taux d'activité des données en entrée de 12,5%. Dans [Shang02] la consommation de ces mêmes ressources passe à 67% de la consommation totale et un taux d'activité des données en

entrée de 25%. De plus, la consommation utile (celle des ressources logiques) est très faible en comparaison, seulement 23% pour [Rouxel02] et 21% pour [Shang02]. Pour ces deux travaux, l'étude est basée sur une estimation de puissance utilisant l'outil Xilinx XPower puis sur des mesures.

Référence de l'étude	Circuit étudié	Répartition de la consommation de puissance en %				Condition de l'étude
		Réseau d'horloges	Réseau de routage	Entrées/sorties	Ressources logiques	
[Garcia99] ENST Paris France	Altera FLEX10K100	51	33	10	6	Mesures
	Xilinx XC4010	50	36	5	9	Mesures
[George99] Univ Berkeley USA	Xilinx XC4003 A	21	65	9	5	Mesures
Xilinx USA	Xilinx XC4010E	1	78	15	6	Mesures
[Rouxel02] LESTER UBS France	Xilinx Virtex-E XCV400	-	58	19	23	Estimations et mesures Ta = 12,5% To = 56 % f = 100Mhz
[Shang02] Univ Princeton USA	Xilinx Virtex-II 2V1000	4	63	12	21	Estimations et mesures Ta = 25% To = 56 % f = 100Mhz
[Poon02a] Univ British Columbia Canada	FPGA type îlot de calcul	19	57	-	24	Simulation VPR

Tableau 4 – Principaux résultats des études menées sur la répartition de la consommation de puissance entre les différentes ressources d'un FPGA.

Il est très intéressant de regarder les conclusions des études du tableau 4, hormis les conclusions classiques du traitement de la consommation de puissance qui consistent à dire qu'il faut réduire la tension d'alimentation et la fréquence de fonctionnement du circuit :

[Garcia99] et [Garcia00] : les auteurs indiquent que dans le cas des FPGA il faut tenter d'utiliser le moins possible de lignes longues pour le routage et qu'il faut éviter de synthétiser des mémoires distribuées sur les LUT. Ces mémoires consomment beaucoup de LUT et de ressources de routage. Elles sont donc peu performantes.

[George99] et [Krusse98] : Les auteurs pensent qu'il faut dans un premier temps se pencher sur l'optimisation énergétique des ressources de routage (y compris le transport des horloges). Ils proposent une solution de routage afin d'augmenter la localité des

traitements communicants dans l'application. Pour cela leur solution s'appuie sur un routage hiérarchique avec pour le niveau le plus bas un réseau de connexions au voisinage.

[Rouxel02] et [Elleouet03] : Les auteurs ont montré qu'il est plus efficace pour la consommation de puissance d'avoir un taux d'utilisation grand, afin d'avoir un minimum de ressources non utilisées. Cette question du choix d'un taux d'occupation élevé a été soulevée dans [Dehon99]. Les algorithmes de synthèse proposent classiquement deux optimisations, vitesse ou surface. Les auteurs ont montré que l'optimisation en vitesse permet une réduction de l'ordre de 10% de la consommation de puissance par rapport à l'optimisation en surface. Effectivement l'optimisation en vitesse privilégie l'utilisation de ressources à capacités faibles ce qui améliore les performances temporelles et diminue la consommation de puissance. Enfin une étude précise sur l'impact des ressources de mémorisation a montré qu'il était toujours plus efficace d'utiliser les blocs de mémoires embarqués plutôt que de distribuer la mémoire sur les LUT.

[Shang02] : Dans ses conclusions cette étude est très proche de l'étude précédente, les auteurs indiquent effectivement qu'il est plus intéressant du point de vue de la consommation de puissance d'avoir des taux d'occupation élevés. De même ils concluent que l'optimisation en vitesse des algorithmes de synthèse entraîne une diminution de la consommation de puissance. Enfin les auteurs insistent sur le fait qu'il faut tenter d'augmenter la localité des ressources qui communiquent le plus.

[Poon02a] et [Poon02b] : Les auteurs indiquent que l'utilisation de fils de routage courts est favorable à une diminution de la consommation de puissance des FPGA. De plus, ils pensent que la taille des clusters qui regroupent des éléments logiques de base doit être importante afin de favoriser une connectivité locale entre de nombreux éléments à l'intérieur même des clusters.

Toutes les études concluent sur l'importance de la consommation des ressources de routage dans le FPGA par rapport aux autres ressources. De plus, elles indiquent que l'utilisation de lignes courtes (de voisinage) est indispensable à l'amélioration de la consommation de puissance globale, en d'autres termes qu'il est indispensable d'augmenter la localité des ressources de traitement qui communiquent le plus.

Afin de comprendre pourquoi toutes les études précédentes amènent aux mêmes conclusions concernant les ressources de routage, le paragraphe suivant va donner quelques notions technologiques sur le routage des FPGA.

3.2.2 Les ressources de routage dans les FPGA

Afin d'illustrer les conclusions présentées précédemment le tableau 5 donne les capacités des différentes ressources de routage pour deux circuits de la société Xilinx, un circuit de l'ancienne génération le XC4003A [Kusse98] et un circuit de la nouvelle génération le Virtex-II [Shang02]. Ainsi on peut voir l'évolution des capacités due à l'évolution de la technologie. Dans ce tableau, et suivant la terminologie Xilinx, un CLB consiste en un cluster de deux éléments logiques de base (LUT plus bascule D) pour le circuit XC4003A et en un cluster de huit éléments logiques de base pour le circuit Virtex-II.

Le tableau 5 montre une évolution dans la différence de capacité entre les lignes longues (donc les connexions éloignées) et les lignes courtes (les connexions locales de voisinage et internes aux CLB). Cette évolution est due à l'évolution de la technologie. Elle entraîne une surconsommation de puissance des lignes longues par rapport aux lignes courtes. Le tableau 6 reprend les résultats de [Shang02]. Il montre que même si le nombre de ressources longues utilisées est faible par rapport aux autres types de ressources de routage, leur consommation est loin d'être négligeable.

Type d'interconnexions	Capacité en pF	
	Xilinx XC4003A Technologie 0,6µm 2 niveaux de métallisation	Xilinx Virtex-II 2v1000 Technologie 0,15µm 8 niveaux de métallisation
Entrées du CLB	1,6 – 2,4	26,4
Ligne de longueur le circuit	-	26,1
Ligne de longueur 10 CLB	2,7 – 4,4	-
Ligne de longueur 6 CLB	-	18,4
Ligne de longueur 5 CLB	1,1 – 2,2	-
Ligne de longueur 2 CLB	3 – 5, 3	13,2
Ligne de longueur 1 CLB	2,4 – 4,4	5,1 – 9,4
Chaîne de propagation de la retenue	2,5	2,7
Connexion locale à l'horloge	1,5	0,72
Réseau de distribution de l'horloge	1 fil d'une colonne 6,4	Réseau en entier 300

Tableau 5 – Capacité des différentes ressources intervenant dans le réseau de routage de deux composants FPGA de la société Xilinx

Type de routage	Pourcentage de ces ressources dans l'utilisation global du routage	Pourcentage de la consommation de puissance par rapport à la consommation totale du routage
Locale	46 %	16 %
Lignes de longueur 2 CLB	31 %	13 %
Lignes de longueur 6 CLB	20 %	11 %
Lignes de longueur le circuit	3 %	12 %

Tableau 6 – Répartition de l'utilisation des ressources de routage et de leur consommation de puissance classiquement pour un FPGA Xilinx Virtex-II.

Le tableau 6 met bien en évidence que les connexions les plus lointaines sont les plus consommatrices d'énergie. Par exemple parmi toutes les ressources de routage utilisées dans le circuit étudié dans [Shang02], seulement 3% sont des lignes longues. Ces 3% de ressources de routage consomment une puissance équivalente à 12% de la consommation totale des ressources de routage. A l'opposé, 46% des ressources de routage utilisées sont des ressources locales (ou lignes courtes). Elles ne consomment qu'une puissance équivalente à 16% de la puissance totale consommée par toutes les ressources de routage. Il est donc indispensable d'utiliser au maximum les communications locales en utilisant les lignes les plus courtes.

3.2.3 Répercussions sur les spécifications en entrée de l'estimation relative

Nous disposons uniquement d'études de la consommation de puissance ciblant des architectures reconfigurables de grain fin. Effectivement il n'existe pas d'études aussi précises sur la répartition de la consommation de puissance dans une architecture reconfigurable de gros grain. Ces architectures sont trop peu souvent réalisées physiquement ce qui empêche toutes études de leur comportement physique. De plus, la structure interne de routage (ligne ou bus de longueurs différentes, structure hiérarchique avec routage global et local) est généralement proche entre architectures de grain fin et de gros grain, ou du moins elles peuvent être considérées comme telle [George99]. Nous allons donc débuter par une hypothèse : nous extrapolons les résultats obtenus avec des architectures de grain fin aux architectures de grain plus épais. C'est à dire que nous reprenons pour toutes les architectures reconfigurables les conclusions des études précédentes. Elles ont montré qu'il est indispensable de favoriser une localité des traitements qui communiquent le plus en tentant de réduire l'utilisation des communications les plus longues. Aussi nous considérons que plus les communications sont longues plus elles vont avoir un coût élevé en consommation de puissance. Cette hypothèse semble réaliste étant donné les évolutions technologiques même s'il est vraisemblable que les écarts obtenus seront moins marqués pour les architectures reconfigurables gros grain.

Fort de ces conclusions nous pouvons prendre comme deuxième hypothèse que les outils de synthèse et placement-routage vont chercher, dans le but de diminuer la consommation de puissance, à rapprocher les traitements (et mémoires) qui communiquent le plus entre eux afin de réduire l'utilisation des lignes de communications longues. Cette hypothèse nous permet de donner le grand principe de l'estimation relative. Dans celle-ci nous cherchons à imiter le comportement des outils de synthèse en rapprochant virtuellement dans l'architecture cible les ressources qui communiquent le plus. Dans la suite du document nous dirons que nous cherchons à "émuler" les outils de synthèse. La connaissance des communications entre opérateurs, nécessite de connaître celles entre opérations apparaissant dans la spécification sur laquelle nous voulons estimer la part relative entre communications locales et globales. Partant de là nous retenons comme modèle architectural, un modèle en adéquation favorisant les communications locales (plus intéressantes d'un point de vue de la consommation de puissance). En résumé, nous avons choisi de modéliser les architectures avec une vision hiérarchique et fonctionnelle et de

modéliser l'application par les communications entre les nœuds du graphe HCDFG de la spécification initiale. Ces points font l'objet des paragraphes suivants.

3.3 Spécification des architectures reconfigurables

Pour spécifier les architectures reconfigurables qui sont les cibles de l'estimation relative nous avons choisi d'utiliser un modèle structurel. Effectivement le paragraphe 2.4.2. a montré que l'utilisation de langage de description architecturale est plus adaptée aux architectures programmables (par exemple pour la génération de compilateur). Deux modèles structurels peuvent être utilisés dans le cas des architectures reconfigurables. Premièrement un modèle structurel physique qui va décrire précisément (au niveau transistor ou au niveau RTL) l'architecture afin de pouvoir effectuer des estimations précises ou développer des outils de placement routage génériques comme VPR [Betz97]. Cette possibilité est à écarter dans notre cas puisque nous souhaitons être capables de modéliser un large espace indépendamment de la technologie. Or ce n'est pas le cas avec une modélisation physique. La deuxième option consiste en un modèle fonctionnel. Cette fois il faut modéliser ce que les ressources sont capables de faire et non plus comment elles sont faites. Cette modélisation convient bien à notre vision, comme l'auteur l'avait démontré dans [Bossuet01], puisque nous souhaitons, conformément à la philosophie de Design Trotter, intervenir très tôt dans le flot de conception donc à un niveau élevé d'abstraction.

A un niveau élevé d'abstraction, il nous est difficile de modéliser les structures fines, comme les ressources de routage. Celles-ci sont très variées d'une architecture à une autre, et même, au sein d'une architecture. Précédemment nous avons montré que les ressources de routage ont un impact des plus important sur la consommation de puissance du circuit. Cependant notre but est "d'émuler" les outils de synthèse afin d'estimer la répartition des communications et non l'utilisation des ressources de routage. Pour prendre en compte l'impact du routage il est nécessaire de faire apparaître dans notre modèle une notion de routage local, global et intermédiaire. La solution choisie consiste à effectuer dans tous les cas une modélisation hiérarchique des architectures même si les ressources de routage de ne le sont pas.

Effectivement il s'agit ici de spécifier que des ressources (traitements ou mémoires) de l'architecture sont proches, et que d'autres sont plus ou moins éloignées. Pour représenter cela avec notre modèle il faut spécifier que les ressources éloignées ne font pas partie d'un même cluster hiérarchique. Ainsi en faisant des clusters de clusters on peut donner différents niveaux de localité aux ressources de routage sans avoir à les modéliser finement. Afin de préciser ces propos deux exemples schématiques de modélisation sont donnés figure 46 et figure 47.

La figure 46 prend pour exemple une architecture gros grain hiérarchique largement inspirée de l'accélérateur reconfigurable de l'architecture Chameleon présentée au paragraphe 1.2.3. La figure 47 prend pour exemple une architecture grain fin du type îlot de

calculs largement inspirée de l'architecture du FPGA Virtex de la société Xilinx présentée au paragraphe 1.2.4. Ces deux architectures sont bien différentes, pourtant il est possible de les modéliser avec notre concept de modèle fonctionnel et hiérarchique.

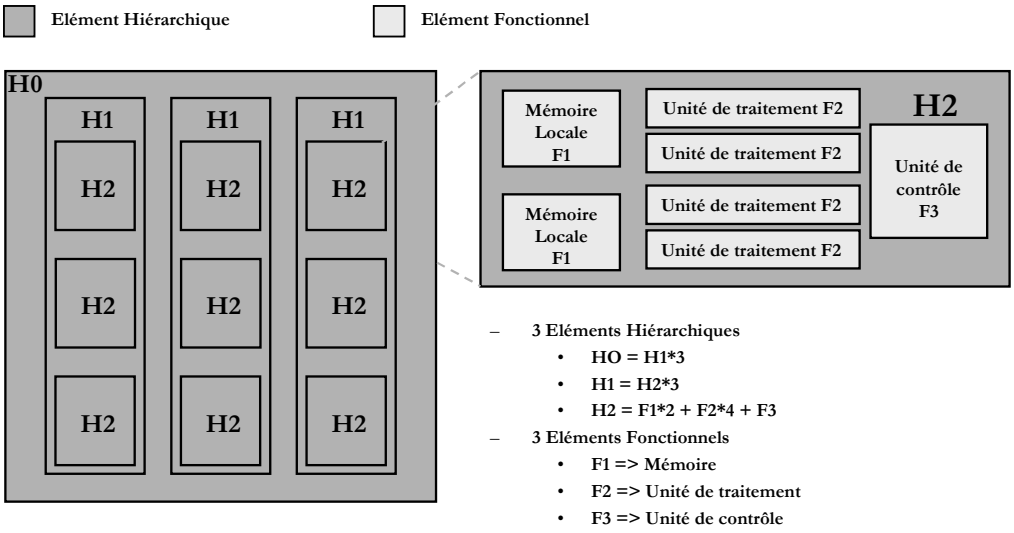


Figure 46 – Exemple de modélisation hiérarchique d'une architecture gros grain hiérarchique (inspirée de l'architecture Chameleon).

Concernant la figure 46, il est très facile de déterminer **les éléments hiérarchiques** qui composent le modèle de l'architecture. Effectivement, il s'agit d'une architecture pour laquelle la structure des ressources de communication est hiérarchique. Notons que dans le cas de l'architecture Chameleon, qui est une architecture commerciale, nous ne disposons pas de plus d'informations sur les ressources de routage. Il faut par la suite définir les fonctions que peuvent réaliser **les éléments fonctionnels** qui sont inclus dans les éléments hiérarchiques.

La modélisation est plus délicate dans le cas de l'architecture grain fin, proposée à la figure 47, puisque celle-ci n'a pas une structure de routage hiérarchique. Cependant par une étude fine du routage il est possible de partitionner l'architecture afin de dégager des zones pour lesquelles les longueurs des ressources de routage sont courtes (connexions locales). Ici les colonnes constituées de mémoires (éléments fonctionnels F3) et de multiplieurs (éléments fonctionnels F2) jouissent de ressources de routage internes particulièrement courtes. De ce fait il est possible de les englober dans des éléments hiérarchiques. Les éléments hiérarchiques H1 sont donc des colonnes constituées d'éléments fonctionnels F2 et F3. Puis il est nécessaire de réunir les éléments logiques (éléments fonctionnels F1) par paquet afin de montrer que des connexions trop longues sont coûteuses. Les éléments hiérarchiques H2 sont des paquets de 16 éléments F1 (quatre par quatre). Cela se traduira par le changement d'élément hiérarchique H2 entre deux éléments logiques F1 éloignés. Cependant il existe un "effet de bord" puisque deux éléments F1 voisins peuvent appartenir à deux éléments hiérarchiques H2 distincts (et aussi voisins). Comme nous

travaillons à haut niveau et que nous modélisons une architecture non hiérarchique avec une vision hiérarchique, il existe des problèmes localement. Mais globalement la modélisation hiérarchique permet de bien faire ressortir la distance entre les éléments sans modéliser de façon physique les lignes de routages. C'est un compromis dans lequel la vision globale l'emporte puisque ce sont les communications de longue distance qui pénalisent l'architecture. Comme pour le premier exemple de modélisation, une fois les ressources de routage modélisées par une vision hiérarchique, il faut définir les fonctions réalisables par les éléments fonctionnels inclus dans les éléments hiérarchiques.

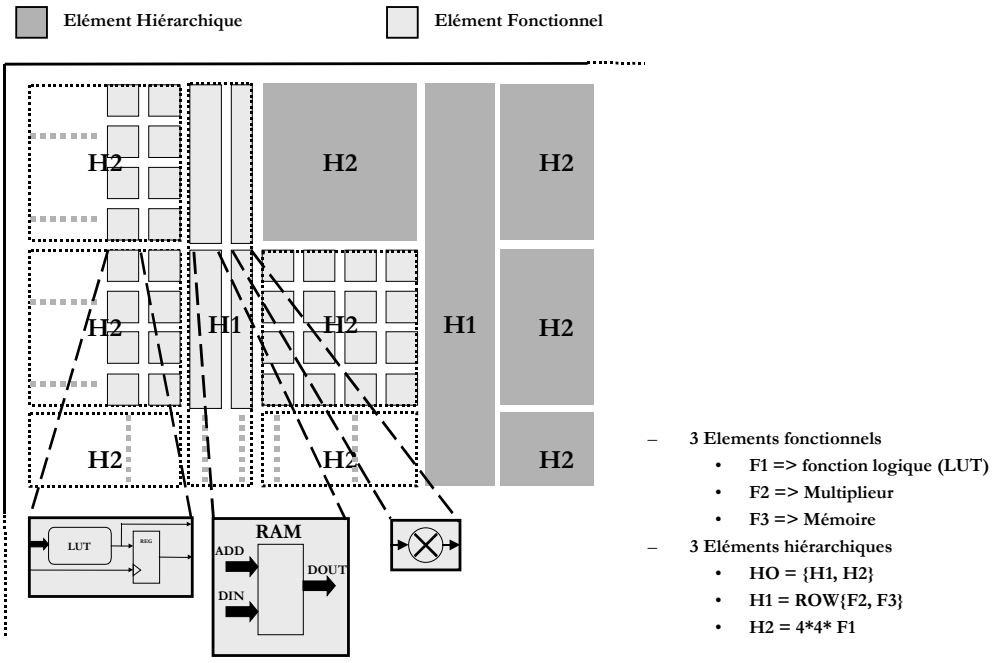


Figure 47 – Exemple de modélisation d'une architecture grain fin du type îlot de calculs (inspirée du FPGA Virtex Xilinx).

Deux éléments de base constituent donc notre modèle [Bossuet02a] : **les éléments fonctionnels** et les **éléments hiérarchiques**. Tous les éléments, qu'ils soient hiérarchiques ou fonctionnels, sont repérés par leur nom et leur numéro d'identification (ID). Il faut veiller à ce que dans une instance de modélisation deux éléments distincts ne portent pas le même nom ou le même numéro d'identification. Cependant Design Trotter vérifie la concordance des champs de la modélisation par compilation et indique à l'utilisateur si il y a des erreurs dans la spécification tel qu'un doublon d'identificateur. Une dernière caractéristique commune à tous les éléments est le nombre de ces éléments dans l'élément hiérarchique supérieur. C'est à dire que la description d'un élément du type A avec un nombre N indique qu'il y a N éléments du type A dans l'élément hiérarchique du type B dans lequel sont contenus les éléments du type A. Enfin les éléments hiérarchiques et fonctionnels ont des caractéristiques et paramètres différents :

Les éléments hiérarchiques sont utilisés pour modéliser la hiérarchie (réelle ou virtuelle) des ressources de routage. Ils peuvent contenir des éléments fonctionnels ou d'autres éléments hiérarchiques. Ainsi il peut y avoir autant de niveaux de hiérarchie que le concepteur le souhaite. L'architecture modélisée est au minimum constituée d'un élément hiérarchique qui contient tous les autres éléments. Les éléments hiérarchiques sont modélisés par leur contenu. Il est nécessaire de spécifier le nombre de types d'éléments hiérarchiques et le nombre de types d'éléments fonctionnels contenus dans l'élément hiérarchique spécifié. Puis une liste des éléments contenus repérés par leur identificateur permet de préciser la spécification de l'élément hiérarchique. Notons que le nombre d'éléments d'un type dans l'élément hiérarchique est spécifié au niveau de l'élément contenu et non au niveau de l'élément hiérarchique contenant. Cette modélisation permet une définition très large d'architectures mais les algorithmes, qui vont réaliser l'estimation relative de la répartition des communications, vont quelque peu freiner la liberté du concepteur. En particulier le nombre de niveaux de hiérarchie est aujourd'hui limité à trois mais il pourra évoluer rapidement. Une contrainte sur les éléments hiérarchiques impose qu'ils ne peuvent pas contenir une unique ressource. Ils doivent au minimum avoir deux ressources identiques ou alors avoir deux ressources distinctes. Le tableau 7 donne la structure grammaticale de la spécification d'un élément hiérarchique. Le paragraphe 3.4 suivant donne des exemples concrets de modélisations qui permettront au lecteur de mieux visualiser la modélisation complète d'une architecture.

<HIERARCHICAL> nom
<ATTRIBUTES>
ID:= *identificateur (nombre entier)*
number:= *nombre d'éléments de ce type contenu dans le hiérarchique supérieur*
h_number:= *nombre de types d'éléments hiérarchiques contenus*
f_number:= *nombre de types d'éléments fonctionnels contenus*
<ELEMENT>
element_1:= *ID du premier élément (hiérarchique ou fonctionnel)*
element_2:= *ID du deuxième élément (hiérarchique ou fonctionnel)*
…
<ENDELEMENT>
<ENDATTRIBUTES>
<ENDHIERARCHICAL>

Tableau 7 – Grammaire de spécification d'un élément hiérarchique.

Les éléments fonctionnels permettent de décrire, par l'ensemble de leurs fonctionnalités, les éléments reconfigurables hors ressources de routage. Les éléments fonctionnels peuvent réaliser des traitements ou des contrôles, on parlera alors de "fonctions opérateurs". Ils peuvent également réaliser des mémorisations, on parlera alors de "fonctions mémoires". Un même élément fonctionnel peut avoir des fonctions opérateurs et des fonctions

mémoires. Prenons le cas typique des LUT dans les architectures de grain fin qui peuvent être utilisées comme fonction logique (fonction opérateur) ou comme mémoire élémentaire (fonction mémoire). Mais un élément fonctionnel peut ne réaliser que des fonctions opérateurs ou que des fonctions mémoires. Un élément fonctionnel doit au minimum être capable de réaliser une fonction. Le tableau 8 donne la structure grammaticale de la spécification d'un élément fonctionnel.

<FUNCTIONAL> nom
<ATTRIBUTES>
ID:= *identificateur (nombre entier)*
number:= *nombre d'éléments de ce type contenu dans le hiérarchique supérieur*
<OPERATOR>
nom_opérateur type_opérateur largeur_donnée
…
<ENDOPERATOR >
<MEMORY>
nom_mémoire type_mémoire largeur_mot nombre_mot
…
<ENDMEMORY>
<ENDATTRIBUTES>
<ENDHIERARCHICAL>

Tableau 8 – Grammaire de spécification d'un élément fonctionnel.

Conformément au tableau 8, les fonctions opérateurs sont spécifiées par un nom dont le choix est libre pour l'utilisateur, un type qui doit absolument être choisi dans la liste donnée dans le tableau 9 en gras, et enfin une largeur des données en entrées. De même les fonctions mémoires sont spécifiées par un nom libre, un type choisi dans la liste donnée dans le tableau 9, la largeur des mots mémorisés et le nombre de mots mémorisés.

La liste donnée au tableau 9 n'est pas limitée. Elle peut facilement évoluer en fonction des besoins. Dans une première version de nos travaux elle est suffisante pour démontrer la pertinence de nos méthodes d'estimation. Mais dans les futures évolutions de l'estimation relative elle sera enrichie. Par exemple pour l'instant la liste ne sépare pas les additionneurs des soustracteurs prenant pour hypothèse que leur taille et performances sont identiques, de même que pour les multiplieurs et les diviseurs.

En ce qui concerne la caractérisation des éléments fonctionnels, quelques idées initiales qui n'ont pas été mises en œuvre lors du développement logiciel de l'outil, pourraient aussi être prises en compte lors de futures évolutions. Par exemple il pourrait être intéressant, dans le cas particulier des architectures de grain fin, d'introduire un paramètre d'utilisation maximum des éléments fonctionnels afin de traduire que lorsque le taux d'utilisation du

circuit devient très élevé, le routage du circuit est très complexe voir difficilement réalisable [Dehon99].

Pour illustrer la spécification fonctionnelle et hiérarchique des architectures reconfigurables le paragraphe suivant présente trois exemples de modélisation d'architectures de granularité différente.

Liste des types des fonctions opérateurs :

LOGIC : fonction logique

ADD : additionneur/soustracteur

MUL_U : multiplieur (ou diviseur) non signé

MUL_S : multiplieur (ou diviseur) signé

MAC : multiplieur accumulateur

ABS : valeur absolue

TWO_COMPL : complementeur à deux

ALU : unité arithmétique et logique

SHIFT_REG : registre à décalage

COMP : comparateur

Liste des types des fonctions mémoires :

RAM : RAM double port

RAM_SP : RAM simple port

ROM : ROM double port

ROM_SP: ROM simple port

Tableau 9 – Liste des types (en gras) des fonctions opérateurs et des fonctions mémoires.

3.4 Exemples de modélisations d'architectures reconfigurables

Ce paragraphe propose trois exemples de modélisation : la modélisation de l'architecture matricielle grain fin du FPGA Xilinx Virtex-II, la modélisation de l'accélérateur matériel reconfigurable gros grain Chameleon à structure hiérarchique et l'architecture matérielle gros grain KressArray à structure matricielle. Ces exemples nous permettent de voir les qualités de notre approche, en particulier la facilité de modélisation d'architectures complexes et donc la possibilité de rapidement faire varier les paramètres architecturaux du modèle. Ainsi l'exploration architecturale n'est pas freinée par une spécification longue et pénible. Mais ces exemples nous permettent aussi de voir les limites de notre approche, entre autres les structures qu'il ne nous est pas permis de modéliser à l'heure actuelle. Nous reviendrons sur les limites de l'approche à la fin de ce paragraphe.

3.4.1 Modélisation fonctionnelle de l'architecture du circuit FPGA Xilinx Virtex-II

L'architecture du circuit reconfigurable à grain fin Xilinx Virtex-II est donnée à la figure 24 au paragraphe 1.2.4. Sur cette figure on voit qu'il y a plusieurs niveaux de hiérarchie dans cette architecture. Par exemple pour les éléments reconfigurables, un CLB contient 4 SLICE et chaque SLICE contient 2 LUT. Les LUT, les blocRAM et les multiplieurs sont les éléments fonctionnels de cette architecture.

```
<MODEL> Xilinx Virtex-II

<HIERARCHICAL> FPGA
 <ATTRIBUTES>
  ID:=1
  number:=1
  h_number:=2
  f_number:=0
   <ELEMENT>
    element_1:=2 /* H1 */
     element_2:=3 /* H2 */
   <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> H1
  <ATTRIBUTES>
   ID:=2
  number:=27
  h_number:=0
  f_number:=1
   <ELEMENT>
    element_1:= 4 /*  CLB */
   <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> H2
  <ATTRIBUTES>
  ID:=3
  number:=40
  h_number:=0
  f_number:=2
   <ELEMENT>
    element_1:= 6 /* MUL */
    element_2:= 7 /* BlokRAM*/
   <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> CLB
  <ATTRIBUTES>
  ID:=4
  number:=16
  h_number:=1
  f_number:=0
   <ELEMENT>
    element_1:= 5 /* SLICE */
   <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> SLICE
  <ATTRIBUTES>
  ID:=5
  number:=4
  h_number:=0
  f_number:=1
   <ELEMENT>
    element_1:= 8 /* LE */
   <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<FUNCTIONAL> MUL
  <ATTRIBUTES>
   ID:=6
   number:=1
    <OPERATOR>
     f1  MUL_U 18
     f2  MUL_S  8
     f3  ABS 18
     f4  TWO_COMPL 18
    <ENDOPERATOR>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<FUNCTIONAL> BlocRAM
  <ATTRIBUTES>
  ID:=7
  number:=1
    <MEMORY>
     m1  RAM_S  1  16384
     m2  RAM_S  2  8292
     m3  RAM_S  4  4096
     m4  RAM_S  9  2048
     m5  RAM_S 18  1024
     m6  RAM_S 36  512
     m7  RAM    1  16384
     m8  RAM    2  8292
     m9  RAM    4  4096
     m10 RAM    9  2048
     m11 RAM   18  1024
     m12 RAM   36  512
    <ENDMEMORY>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<FUNCTIONAL> LE
  <ATTRIBUTES>
  ID:= 4
  number:= 4
    <OPERATOR>
     f1  LOGIC 4
     f2  ADD 2
     f3  SHIFT_REG 4
    <ENDOPERATOR>
    <MEMORY>
     m1 RAM 1 16
    <ENDMEMORY>
    <ENDATTRIBUTES>
  <ENDFUNCTIONAL>

<ENDMODEL> Xilinx Virtex-II
```

Tableau 10 – Modélisation fonctionnelle de l'architecture Xilinx Virtex-II

Le tableau 10 donne la modélisation fonctionnelle de cette architecture suivant la grammaire proposée. Elle débute par le nom du modèle, et par un élément hiérarchique atypique. Effectivement cet élément hiérarchique, nommé *FPGA* ici, n'est pas inclut dans

un élément hiérarchique à un niveau supérieur. L'élément hiérarchique *FPGA* est donc l'élément de plus haut niveau de hiérarchie. Cet élément inclut deux éléments hiérarchiques *H1* et *H2* qui représentent les ressources locales de routage. C'est à dire que l'on suppose que les éléments à l'intérieur de *H1* ou à l'intérieur de *H2* bénéficient d'un routage de voisinage. Cette modélisation induit que les communications entre éléments hiérarchiques (de *H1* à *H1* ou de *H1* à *H2* par exemple) sont plus coûteuses (du point de vue de la consommation de puissance) que des communications internes à un élément hiérarchique. Pour cet exemple nous avons considéré que *H1* englobait un carré de 4 par 4 *CLB*, soit 16 *CLB*, et que *H2* englobait un bloc de mémoire *RAM* et un multiplieur. Les deux éléments hiérarchiques suivants sont les *CLB* qui englobent des *SLICES* et ces derniers qui englobent des éléments fonctionnels de type *LUT*.

Il y a donc trois types d'éléments fonctionnels, les *LE* (*ou LUT*) réalisent des fonctions logiques ou des mémoires élémentaires, les *multiplieurs* réalisent des multiplications mais aussi quelques fonctions supplémentaires comme des complémentations à deux et les blocs de mémoires *RAM* qui peuvent être configurés de façon à réaliser des mémoires dont la taille des données est différente (en gardant une taille de mémorisation constante).

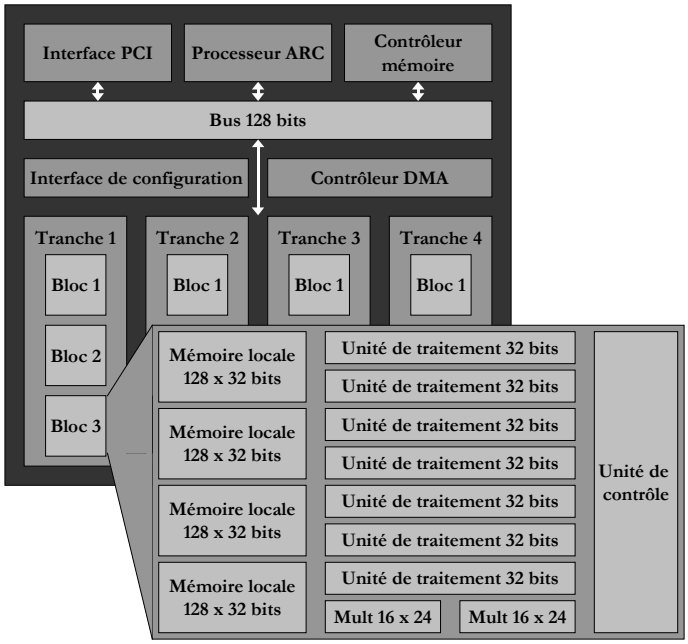


Figure 20 – Architecture de Chameleon

3.4.2 Modélisation fonctionnelle de l'accélérateur matériel reconfigurable du circuit Chameleon

L'architecture du circuit reconfigurable à gros grain Chameleon est donnée à la figure 20 au paragraphe 1.2.3. Cette architecture est faite de deux parties reliées par un bus, une partie logicielle construite autour d'un processeur ARC et une partie matérielle reconfigurable à

gros grain (ou accélérateur matériel reconfigurable). La modélisation que nous proposons ne concerne que l'accélérateur matériel. Cet accélérateur a une structure très hiérarchique comme on le voit sur la figure 20. Au niveau le plus bas, des opérateurs de gros grain (ALU et multiplieurs) couplés à des mémoires forment des *blocs*. Ces derniers sont englobés dans des *tranches*. Les *tranches* sont positionnées en colonnes pour réaliser le dernier niveau de hiérarchie.

Le tableau 11 donne la modélisation fonctionnelle matérielle reconfigurable de Chameleon. Comme dans le cas du Virtex, la modélisation débute avec, au plus haut niveau de hiérarchie, un élément hiérarchique unique appelé *Accélérateur*. Celui-ci se compose de quatre éléments hiérarchiques du type *Tranche*. Une *Tranche* se compose de trois *Blocs*. L'*Accélérateur*, les *Tranches* et les *Blocs* forment donc les trois niveaux de hiérarchie. Il y a trois éléments fonctionnels : les unités de traitement (*UT*), les multiplieurs (*MUL*) et les mémoires (*RAM*).

Par rapport au schéma de la figure 20, l'unité de contrôle n'apparaît pas comme élément fonctionnel à l'intérieur des éléments hiérarchiques *Blocs*. Effectivement, notre modèle fonctionnel n'est pas un modèle d'exécution, il lui est donc difficile de modéliser une machine d'état pour, par la suite, la faire correspondre à une spécification sous forme de graphe de l'application.

```
<MODEL> Chameleon

<HIERARCHICAL> Accélérateur
  <ATTRIBUTES>
   ID:=1
   number:=1
   h_number:=1
   f_number:=0
    <ELEMENT>
      element_1:=2  /* Tranche */
    <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> Tranche
  <ATTRIBUTES>
   ID:=2
   number:=4
   h_number:=1
   f_number:=0
    <ELEMENT>
      element_1:= 3 /* Bloc */
    <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> Bloc
  <ATTRIBUTES>
   ID:=3
   number:=3
   h_number:=0
   f_number:=3
    <ELEMENT>
      element_1:= 4 /* UT */
      element_2:= 5 /* MUL */
      element_3:= 6 /* Mémoire */
    <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<FUNCTIONAL> UT
  <ATTRIBUTES>
   ID:=4
   number:=7
    <OPERATOR>
      f1  ALU 32
    <ENDOPERATOR>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<FUNCTIONAL> MUL
  <ATTRIBUTES>
   ID:=5
   number:=2
    <OPERATOR>
      f1  MUL_S  16
    <ENDOPERATOR>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<FUNCTIONAL> RAM
  <ATTRIBUTES>
   ID:=6
   number:=4
    <MEMORY>
      m1  RAM  32  128
    <ENDMEMORY>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<ENDMODEL> Chameleon
```

Tableau 11 – Modélisation fonctionnelle de l'accélérateur matériel reconfigurable du circuit Chameleon

3.4.3 Modélisation fonctionnelle de l'accélérateur matériel reconfigurable KressArray

Enfin pour terminer, la spécification fonctionnelle de l'architecture de l'accélérateur matériel reconfigurable à gros grain KressArray est donnée au tableau 12. Le schéma de cette architecture reconfigurable à gros grain est donné sur la figure 17 du paragraphe 1.2.2. Cette architecture à structure matricielle est composée de vingt-quatre ressources de traitement de type ALU. Elles sont réparties en quatre lignes de six. Pour la modélisation, très simple dans ce cas, nous avons opté pour réunir les lignes de 6 ALU, modélisées en éléments fonctionnels *OP*, dans un même élément hiérarchique *ligne*. Ce choix implique une hypothèse sur les ressources de routage. Effectivement si les *OP* sur une ligne font parties d'un même élément hiérarchique, il faut qu'ils jouissent de connexions moins coûteuses à utiliser que des connexions entre lignes. Cependant nous ne disposons pas des informations précises sur les ressources de routage. Notre modélisation a besoin de ces informations même si elle ne décrit pas précisément les ressources de routage. Des essais et/ou des caractéristiques techniques permettraient de valider notre choix.

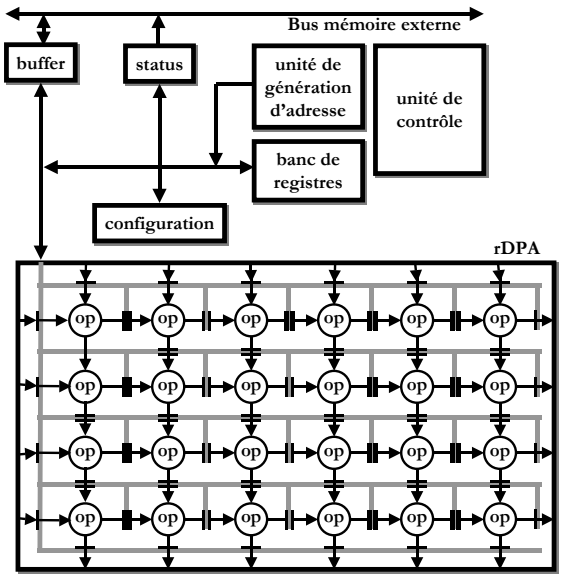


Figure 17 – Architecture KressArray.

S'il est possible de décrire les traitements des éléments fonctionnels du type *OP*, il n'est pas possible avec la modélisation fonctionnelle proposée de modéliser les possibilités de routage des *OP*. Effectivement dans la définition de l'architecture KressArray, il est possible d'utiliser ces ressources pour effectuer un traitement et/ou pour transférer un flot de données.

```
<MODEL> KressArray

<HIERARCHICAL> Accélérateur
  <ATTRIBUTES>
    ID:=1
    number:=1
    h_number:=1
    f_number:=0
      <ELEMENT>
        element_1:=2  /* ligne */
      <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<HIERARCHICAL> Ligne
  <ATTRIBUTES>
    ID:=2
    number:=4
    h_number:=0
    f_number:=1
      <ELEMENT>
        element_1:= 3 /* Op */
      <ENDELEMENT>
  <ENDATTRIBUTES>
<ENDHIERARCHICAL>

<FUNCTIONAL> Op
  <ATTRIBUTES>
    ID:=3
    number:=6
      <OPERATOR>
        f1  ALU 32
      <ENDOPERATOR>
  <ENDATTRIBUTES>
<ENDFUNCTIONAL>

<ENDMODEL> KressArray
```

Tableau 12 – Modélisation fonctionnelle de l'accélérateur matériel reconfigurable KressArray.

3.4.4 Limites de la modélisation fonctionnelle proposée

Les exemples que nous avons donnés dans ce paragraphe nous permettent de voir les limites de notre approche. Tout d'abord seules les parties matérielles sont modélisables. Comme nous l'avons vu au premier chapitre, il n'est pas rare que les architectures des systèmes reconfigurables sur puce soient réalisées autour d'un processeur accompagné de mémoires (mémoires caches entre autres) et d'un accélérateur matériel reconfigurable (plus ou moins couplé avec le processeur). Nous nous intéressons uniquement à la partie matérielle puisque la spécification que nous faisons de l'architecture est uniquement utilisée par l'estimation relative de la répartition des communications. Or cet estimateur est développé uniquement pour les parties matérielles. Mais Design Trotter devrait bientôt évoluer et intégrer un outil de projection sur cible logicielle. Celui-ci disposera d'une modélisation des parties logicielles.

Nous avons vu, avec l'exemple de Chameleon, que les unités de contrôle du type machine d'états ne sont pas prises en compte lors de la modélisation de l'architecture, car elles ne sont pas prises en compte lors de l'estimation système. Les seuls contrôles pris en compte sont ceux internes à l'exécution de l'application.

Certaines structures sont difficilement modélisables du fait de leur trop forte hétérogénéité telle que l'architecture aSoC (présentée au paragraphe 1.2.3.) ou du fait de leur structure particulière de communication telle que l'architecture circulaire du Systolic Ring (présentée au paragraphe 1.2.2). Effectivement comme le modèle met en évidence une vision hiérarchique des ressources de routage, il ne prend pas en compte une spécificité telle qu'un routage linéaire.

Gardons bien en tête que la modélisation fonctionnelle a l'avantage de pouvoir facilement évoluer. Son niveau actuel dépend essentiellement de ce que permet l'outil d'estimation relative de la répartition des communications dans les architectures reconfigurables ciblées. Cet outil sera présenté au chapitre suivant de ce document. En conclusion sur la modélisation fonctionnelle nous pouvons lister ce qu'elle est capable de modéliser :

- Les éléments de traitements de gros grain (ALU, multiplieur, additionneur …),
- Les éléments de traitements de grain fin (LUT, comparateur),
- Les éléments de mémorisation de toutes tailles (RAM, ROM, registre),
- La hiérarchie du réseau de routage,
- Des clusters de tailles différentes,
- L'hétérogénéité de l'architecture (différents grains de traitements par exemple ou des traitements différents),
- Des architectures de grandes ou de petites tailles.

Comme nous l'avons déjà évoqué au début de ce chapitre, l'outil d'estimation relative nécessite une spécification de l'application basée sur la spécification de base en HCDFG. De plus, il doit bénéficier d'informations supplémentaires concernant les communications entre les différents nœuds du graphe. Le paragraphe suivant présente cette spécification.

3.5 Spécification de l'application

3.5.1. Définition du graphe des communications moyennes ACG

Comme nous l'avons vu plusieurs fois dans ce chapitre, la spécification de l'application, dans le cadre de l'estimation relative des communications dans les architectures reconfigurables, part d'un enrichissement de la spécification HCDFG vue au paragraphe 2.6.1. Effectivement, il est dispensable à l'estimation relative d'utiliser le même formalisme d'entrée que les autres outils de Design Trotter, en particulier l'estimation système vue au chapitre 2.6.2. Cette dernière est réalisée avant l'estimation relative afin de sélectionner une solution d'implémentation de l'application (sur une architecture abstraite). Une solution correspond à un nombre de cycles abstraits et à un ordonnancement (sans assignation temporelle d'opérations à des opérateurs). A l'issue de l'estimation système il est donc possible de connaître les ressources de traitement nécessaires à l'application. Cependant l'estimation de la taille mémoire, lors de l'estimation système, est peu précise. Des travaux sont en cours afin d'intégrer une méthode performante (dite "méthode de Balasa" [Balasa95]) d'estimation de la mémoire à l'estimation système. L'estimation relative va dans un premier temps enrichir la spécification HCDFG avec les résultats de l'estimation système afin de mettre en valeur les communications entre les nœuds du graphe.

Le but de l'estimation relative est de rendre compte de la répartition des communications aux différents niveaux de hiérarchie de l'architecture modélisée, comme montré au paragraphe précédent. Pour cela, l'estimation relative doit connaître quels sont les types de nœuds (donc de traitements, de contrôles ou de mémorisations) qui communiquent le plus, afin de virtuellement rapprocher ces nœuds dans la réalisation de l'application. L'estimation relative va donc dans un premier temps enrichir la spécification HCDFG avec les résultats de l'estimation système afin de connaître les besoins de l'application en ressources de

traitement et de mémorisation. Puis les communications entre les nœuds du graphe seront mises en valeur à travers un graphe de communication, *l'ACG* (*Average Communication Graph*).

Prenons le cas simple d'un DFG où les nœuds sont uniquement des nœuds traitements (les arcs représentent alors les transferts de données). Ce DFG, schématisé à la figure 48, comporte sept nœuds de quatre types : a, b, c et d. Chaque nœud représente une opération telle qu'une addition ou une multiplication par exemple. Ce DFG est ordonnancé par l'estimation système. Parmi les solutions proposées par cette étape, l'utilisateur choisit un ordonnancement. Celui-ci donne par exemple les besoins en ressources de traitement suivants :

- un opérateur A pour réaliser les deux opérations a,
- deux opérateurs B pour réaliser les deux opérations b,
- un opérateur C pour réaliser les deux opérations c,
- un opérateur D pour réaliser l'opération d.

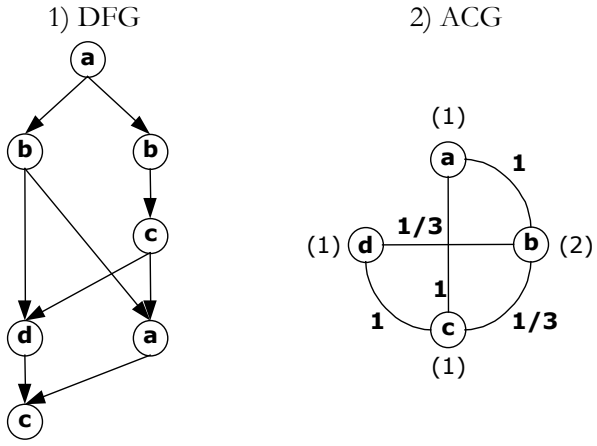


Figure 48 – 1) Le DFG exemple et 2) l'ACG qui en résulte.

Il est possible dans un premier temps de compter sur le DFG, figure 48, le nombre de communications (pour un DFG un arc correspond à une communication) entre les différentes opérations. Ce nombre apparaît dans la deuxième colonne du tableau 13. Par exemple il y a trois communications entre les opérations a et b. Bien sûr dans ce cas très simple le nombre de communications est faible.

Type de Communication	Nombre de communications	Valeur des arcs
Entre les opérations a et b	3	1
Entre les opérations a et c	2	1
Entre les opérations b et c	1	1/3
Entre les opérations b et d	1	1/3
Entre les opérations c et d	2	1

Tableau 13 – Types et nombres de communications dans le DFG de la figure 48.

Il y a donc cinq types de communications possibles. Nous pouvons remarquer par exemple qu'il n'existe pas de communication entre les opérations a et d. A l'aide de ces informations et avec les résultats de l'estimation système, il est possible de construire un nouveau graphe. Ce nouveau graphe est le graphe de communications ACG. La figure 48 présente le DFG, exemple ainsi que l'ACG qui lui est relié. La différence entre les deux graphes tient en deux axes : l'ACG n'a qu'un nœud par type d'opérations (par exemple un seul nœud b pour 2 opérations et 2 opérateurs) et les arcs de l'ACG, contrairement à ceux du DFG, ne sont pas orientés et sont valués. 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 *n1et n2* conformément aux résultats de l'estimation système. Le nombre d'opérateurs nécessaires est noté entre parenthèses près des nœuds correspondants dans l'ACG, comme on peut le voir sur la figure 48.

Si on calcule, par exemple, la valeur de l'arc entre le nœud a et le nœud b dans l'ACG, on obtient l'unité. Effectivement le nombre de communications dans le DFG entre les opérations de type a et les opérations de type b est de trois. L'estimation système a donné comme résultat qu'il est nécessaire d'implémenter un opérateur A et deux opérateurs B soit trois opérateurs au total pour trois communications, d'où une valeur d'arc égale à l'unité. Cette valeur ne correspond pas directement à un nombre moyen de communications entre les nœuds, mais elle fait un rapport entre le nombre de communications et d'opérateurs. Son but et de permettre à l'algorithme qui va par la suite parcourir le graphe, de sélectionner les arcs les plus critiques.

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 ne dispose pas encore d'une estimation précise de la mémoire et considère que les données scalaires seront réalisées à l'aide de registres. Seules les données multidimensionnelles sont utilisées pour estimer la taille des mémoires

physiques nécessaires à l'application. Dans ce cadre nous considérons les nœuds mémoires de la même façon que l'estimation système. C'est à dire que les données scalaires sont réalisées avec des registres et que les données multidimensionnelles sont réalisées avec des mémoires ROM ou RAM (s'il s'agit de constantes ou non). Dans le cas des architectures matérielles reconfigurables, les traitements sont largement parallèles et nombreux. De ce fait, toutes les architectures prévoient, en sortie des traitements, des registres de synchronisation. Les données scalaires, qui sont bien souvent des résultats intermédiaires de calcul sont donc naturellement dirigées vers les registres de sorties. L'estimation du nombre de ces registres n'est pas nécessaire puisque ces architectures matérielles et parallèles en disposent largement. Nous ne considérons donc pas les données scalaires dans l'estimation des communications, ils sont remplacés par des arcs. Ce n'est pas le cas des données multidimensionnelles qui sont réalisées dans des mémoires RAM ou ROM. Ces mémoires sont le plus souvent embarquées au sein de l'architecture matérielle, il faut donc favoriser leur rapprochement avec les ressources de l'architecture (opérateurs ou autres mémoires) les plus communicantes avec elles. Nous suivons ainsi la même stratégie pour les ressources de mémorisation que pour les ressources de traitement. Il existe toutefois une différence, dans une première approche 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 que nous conservons dans le DFG représentent donc des données multidimensionnelles et sont associés à un nombre unitaire entre parenthèses. La figure 49 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. A partir du DFG initial (figure 49-1), on fait une première transformation visant à l'élimination des nœuds mémoires représentant la donnée scalaire B (figure 49-2). Sur ce nouveau graphe, le nœud mémoire représentant la donnée multidimensionnelle A est considéré comme les autres nœuds mais avec l'unité comme résultat de l'ordonnancement (entre parenthèses). A partir de ce nouveau DFG, l'ACG (figure 49-3) 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 qui interviennent dans les boucles et les branchements conditionnels, ils sont considérés comme des nœuds de traitement. Ils sont donc ordonnancés avec des ressources de types comparateurs lors de l'estimation système et seront réalisés avec ce type de ressources dans l'architecture cible.

Pour les branchements conditionnels (*if* ou *case*) toutes les branches possibles sont considérées lors de l'estimation système. Cependant elles vont plus ou moins intervenir dans le nombre total de communication en fonction de la probabilité de passage dans les différentes branches. Cette probabilité est calculée par *profiling* lors de l'estimation système. Le nombre de communications initiales entre les nœuds d'une branche est réduit par multiplication par la probabilité de passage.

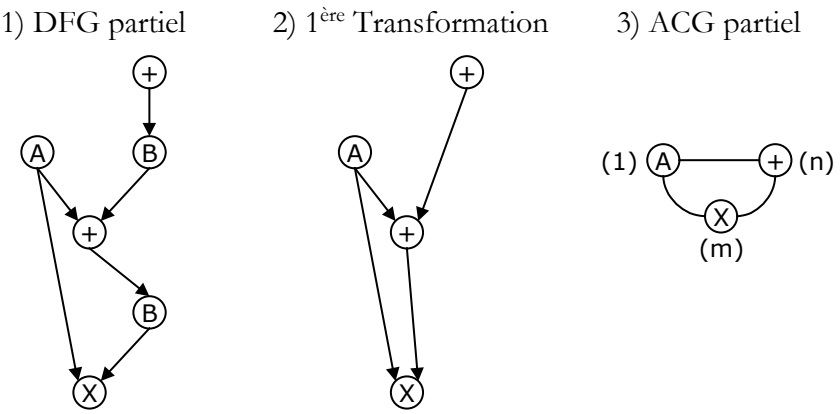


Figure 49 – Transformation d'un DFG comportant des nœuds mémoires en ACG.

Dans le même esprit, le nombre de passage 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. De même pour les communications du graphe de calcul et de test de l'indice de boucle qui est exécuté autant de fois (plus une fois) que le cœur de boucle.

3.6 Conclusion

Ce troisième chapitre est capital dans notre évolution vers la compréhension du système d'estimation relative de la répartition des communications dans une architecture reconfigurable pour une application donnée. Ce système vise l'exploration de l'espace de conception architecturale. Comme le deuxième chapitre de ce document l'a montré, les spécifications en entrées des systèmes d'exploration sont capitales pour définir l'espace architectural explorable.

Nous avons montré par l'étude de la répartition de la consommation de puissance dans les éléments constitutifs des architectures reconfigurables à grain fin, que la consommation de puissances des ressources de routage était prédominante. Une extrapolation des résultats aux architectures de gros grain, nous fait dire que dans tous les cas il est préférable de privilégier la localité des communications propres à l'application. De ce fait les outils de synthèse doivent aller dans ce sens, c'est à dire tenter de rapprocher les ressources qui communiquent le plus. Nous sommes donc arrivés à la conclusion qu'il fallait émuler la stratégie de ces outils pour pouvoir estimer avec une bonne prédiction la répartition des communications dans une architecture. Notre postulat est que pour une même application, une architecture qui favorise la localité des communications sera plus performante en terme de consommation de puissance. Notons que ce postulat sera quelque peu précisé et affiné par la suite afin de prendre en compte d'autres paramètres tels que le taux d'utilisation des ressources par exemple mais aussi la taille des clusters.

Nous avons donc démontré la pertinence de nos choix de spécifications : un modèle abstrait et fonctionnel pour spécifier les architectures et la spécification HCDFG d'entrée ainsi que les résultats de l'estimation système pour l'application. Le modèle architectural utilise une représentation hiérarchique afin de mettre en valeur des zones de communications plus ou moins locales dans celles-ci. Sans décrire physiquement et précisément les ressources de routage, ce modèle permet de les prendre en compte. De plus, Il est simple à réaliser et très facilement évolutif. Il permet de rapidement faire varier les paramètres architecturaux facilitant ainsi l'exploration. Pour l'application, la spécification HCDFG d'entrée ainsi que les résultats de l'estimation système sont utilisés dans le but de créer un graphe simple dans lequel les communications entre les opérations nécessaires à l'application sont mises en valeur. Ce graphe évolue par la suite de façon dynamique lors de l'exécution de l'estimation relative. Nous avons conçu un graphe de communication l'ACG qui répond à ces attentes.

Maintenant nous pouvons nous attarder, dès le prochain chapitre, à étudier les algorithmes mis en œuvre dans l'estimation relative de la répartition des communications. Puis nous étudierons dans ce même chapitre la méthode d'exploration de l'espace de conception qui s'appuie sur l'estimation relative.

Chapitre 4

Algorithmes d'estimation relative et méthode d'exploration de l'espace de conception

Le flot d'exploration de l'espace de conception des architectures reconfigurables de Design Trotter est orienté par l'estimation de la répartition des communications dans l'architecture. Pour obtenir ce résultat, l'estimation relative (outil inséré dans le flot Design Trotter) met en œuvre un algorithme de projection matérielle qui utilise la même stratégie qu'un outil de synthèse mais bien plus rapidement. En effet, cet algorithme cherche à rapprocher dans l'architecture les opérateurs qui communiquent le plus. Cependant la définition d'un algorithme optimum est très complexe à haut niveau d'abstraction, lorsque l'on choisit, et c'est notre cas, une modélisation structurelle abstraite des architectures. C'est pourquoi trois algorithmes ont été développés afin de proposer un encadrement (valeur minimum, maximum et intermédiaire) pour les résultats. Donner une seule valeur aurait demandé une exactitude dans les résultats que nous ne pouvions assurer à ce niveau d'abstraction.

L'exploration que nous proposons est une exploration architecturale qui fournit des informations de conception à l'utilisateur. Conformément au chapitre 2, nous pouvons caractériser notre approche comme mono-objectif (efficacité en consommation de puissance) et semi-automatique.

Afin de bien comprendre le flot d'exploration, ce chapitre présente dans un premier temps l'estimation relative et les algorithmes qu'elle utilise dans son flot de projection matérielle. Puis le flot d'exploration est développé et des exemples didactiques permettent de l'illustrer pas à pas.

4.1 Flot d'estimation relative de la répartition des communications

Le paragraphe 2.5.4 a présenté succinctement l'estimation relative de la répartition des communications dans les architectures reconfigurables. Ce paragraphe a également montré le positionnement de l'estimation relative dans Design Trotter et son lien avec l'estimation système qui lui fournit ses résultats comme entrées. Le chapitre 3 a présenté les spécifications en entrée de l'estimation système. Les applications sont spécifiées par leur HCDFG qui est modifié lors de l'estimation relative en un graphe de communication, l'ACG. Ce graphe utilise les résultats de l'estimation système. Les architectures sont spécifiées par un modèle hiérarchique structurel et fonctionnel.

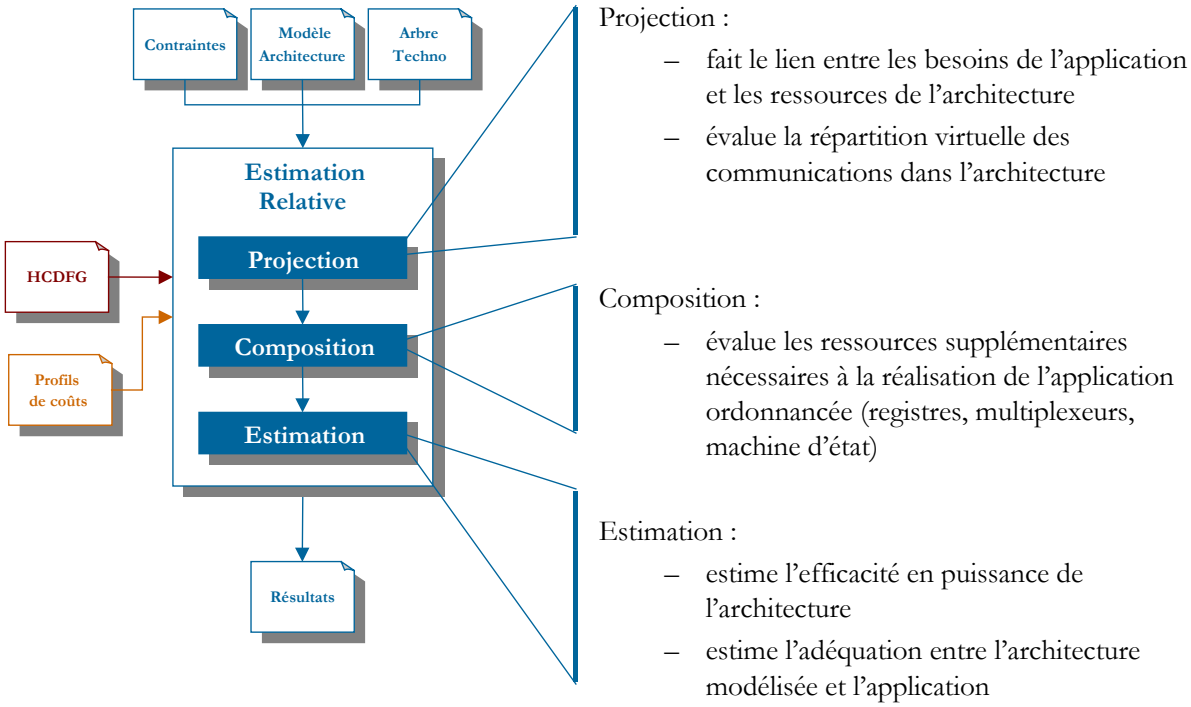


Figure 50 – Flot complet de l'estimation relative dans Design Trotter.

La figure 50 présente le flot complet de l'estimation relative. Elle prend en entrée la spécification HCDFG de la fonction à estimer, ainsi que les résultats de l'estimation système (profils de coûts). Le modèle architectural permet à l'utilisateur de spécifier les architectures ciblées. Un ensemble de contraintes sur la réalisation de l'application permet à l'utilisateur de faire des choix lors de l'exploration de l'espace de conception. Enfin une bibliothèque de pré-décompositions de traitements appelée arbre technologique, permet de

mettre en correspondance la granularité des traitements dans l'application et la granularité des ressources de l'architecture. Si l'application nécessite des traitements de gros grain, par exemple du type multiplieur, mais que l'architecture est une architecture de grain fin, comme nous ne faisons pas de synthèse il faut pouvoir décomposer les opérateurs de gros grain en opérateurs grain fin. Cette bibliothèque permet donc à l'outil de sélectionner une architecture grain fin pour la réalisation d'opérateurs de gros grain.

L'estimation relative se déroule en trois phases successives :

La projection matérielle permet de faire correspondre les besoins en traitements et mémorisations de l'application avec les ressources (opérateurs et mémoires) de l'architecture. Pour cela la spécification de l'application permet la création du graphe de communication ACG comme nous l'avons vu au chapitre précédent. Ce graphe est utilisé pour chercher les opérations de l'application qui communiquent le plus et qui doivent virtuellement être rapprochées dans l'architecture. Cette partie est construite autour d'un algorithme principal de parcours et de transformations du graphe. Cet algorithme met en œuvre trois algorithmes de regroupement hiérarchique des opérations communicantes et de calcul de la valeur des arcs du graphe. Ils vont plus ou moins privilégier la localité des opérations.

La composition est une étape intermédiaire qui estime le nombre de ressources supplémentaires qui doivent être mises en œuvre afin d'assurer l'exécution de la fonction estimée. Effectivement une solution de l'estimation système correspond à un ordonnancement. Cette solution peut impliquer, suivant le parallélisme choisi, l'utilisation d'une même ressource pour réaliser plusieurs traitements. Pour ce faire des ressources supplémentaires doivent être ajoutées à la réalisation telles que des registres, des multiplexeurs et des machines à états finis.

L'estimation finale calcule à partir de la projection matérielle et de la répartition hiérarchique des traitements et des mémorisations qui en découle, la répartition des communications dans l'architecture modélisée pour la fonction estimée. Cette répartition couplée au taux d'utilisation permet une estimation de l'efficacité en puissance de l'architecture et de l'adéquation entre l'architecture modélisée et la fonction estimée. De plus, les informations fournies peuvent orienter l'utilisateur dans son choix de développement à travers la complexité des algorithmes de synthèse qui seront nécessaires au développement final comme nous le verrons dans ce chapitre.

A l'heure actuelle la bibliothèque de pré-décompositions correspondant à l'arbre technologique n'a pas été développée, il était plus important de mettre en œuvre les estimateurs avant d'élargir les possibilités de l'outil. Cependant nous introduisons cette notion d'arbre technologique car elle devrait faire l'objet d'une étude dans un futur proche. De même la composition n'a pas encore fait l'objet d'un développement car notre attention a d'abord été porté sur l'algorithme de projection qui est le cœur de notre système. Effectivement c'est grâce à ce dernier que l'on peut estimer rapidement l'efficacité d'une architecture pour une application. C'est notre but principal, ce qui justifie le fait que certaines parties du flot ne soient pas encore développées.

Les paragraphes suivants détaillent l'algorithme de projection matérielle avant de se concentrer sur les trois algorithmes de regroupement hiérarchique. Nous verrons entre autres pourquoi il est intéressant de donner un intervalle comme résultat plutôt qu'une unique valeur.

4.2 Algorithme de projection matérielle

4.2.1 Présentation

L'algorithme de projection est présenté à la figure 51. Il reçoit en entrée le graphe ACG de communication, le modèle architectural et éventuellement un modèle de coûts pour les communications. Bien que sur l'algorithme de la figure 51 nous voyons apparaître deux modèles d'architectures, les modèles architecturaux de premier et de deuxième niveau, l'utilisateur ne spécifie qu'un modèle. Cependant les deux modèles sont nécessaires car, comme nous allons le voir, nous effectuons une première puis une deuxième projection.

La première projection a pour but d'assigner des opérations de l'ACG à des ressources de traitement de l'architecture. Ces ressources de traitement se trouvent au plus bas niveau de la hiérarchie de l'architecture qui en compte trois. Lorsqu'ils sont regroupés au sein d'un même élément hiérarchique les nœuds de l'ACG se transforment en nœuds composites qui représentent l'assignation d'opérations ou de mémorisations du graphe à des éléments fonctionnels de l'architecture. Le regroupement hiérarchique des nœuds de l'ACG se fait suivant trois algorithmes que nous détaillerons par la suite. Le déroulement de la première projection entraîne une modification de l'ACG et lorsque celui-ci ne peut plus être modifié la première projection est terminée. S'il reste des nœuds non assignés à des ressources de l'architecture, ils sont transformés tout de même en nœuds composites. Cette situation peut être provoquée par deux choses ; soit les éléments hiérarchiques au niveau le plus bas n'ont pas un nombre assez grand d'éléments fonctionnels pour réaliser tous les regroupements, où alors une opération de l'ACG n'est pas réalisable sur l'architecture. Il faut dans ce dernier cas revenir sur la spécification de l'architecture.

Une fois la première projection terminée, une deuxième projection est lancée, elle ne remet pas en jeu les assignations de la première. Maintenant il s'agit d'assigner les nœuds composites de l'ACG aux éléments hiérarchiques de deuxième niveau de l'architecture. Cette fois ce sont des nœuds super composites qui vont représenter les regroupements hiérarchiques. L'algorithme de projection est le même pour les deux cas. Dans un cas on cherche le regroupement de nœuds dans le niveau le plus bas de hiérarchie de l'architecture et dans l'autre cas on cherche le regroupement au niveau hiérarchique au-dessus. Il pourrait encore y avoir d'autres projections si l'architecture disposait de plus de trois niveaux de hiérarchie. Cependant il s'agit de cas très peu fréquents.

Les paragraphes suivants vont illustrer l'algorithme de projection. En particulier nous allons voir ce qui est entendu par regroupement hiérarchique, nœud composite et super composite. Aussi nous verrons comment il est possible de calculer le coût des communications à l'issue des deux projections.

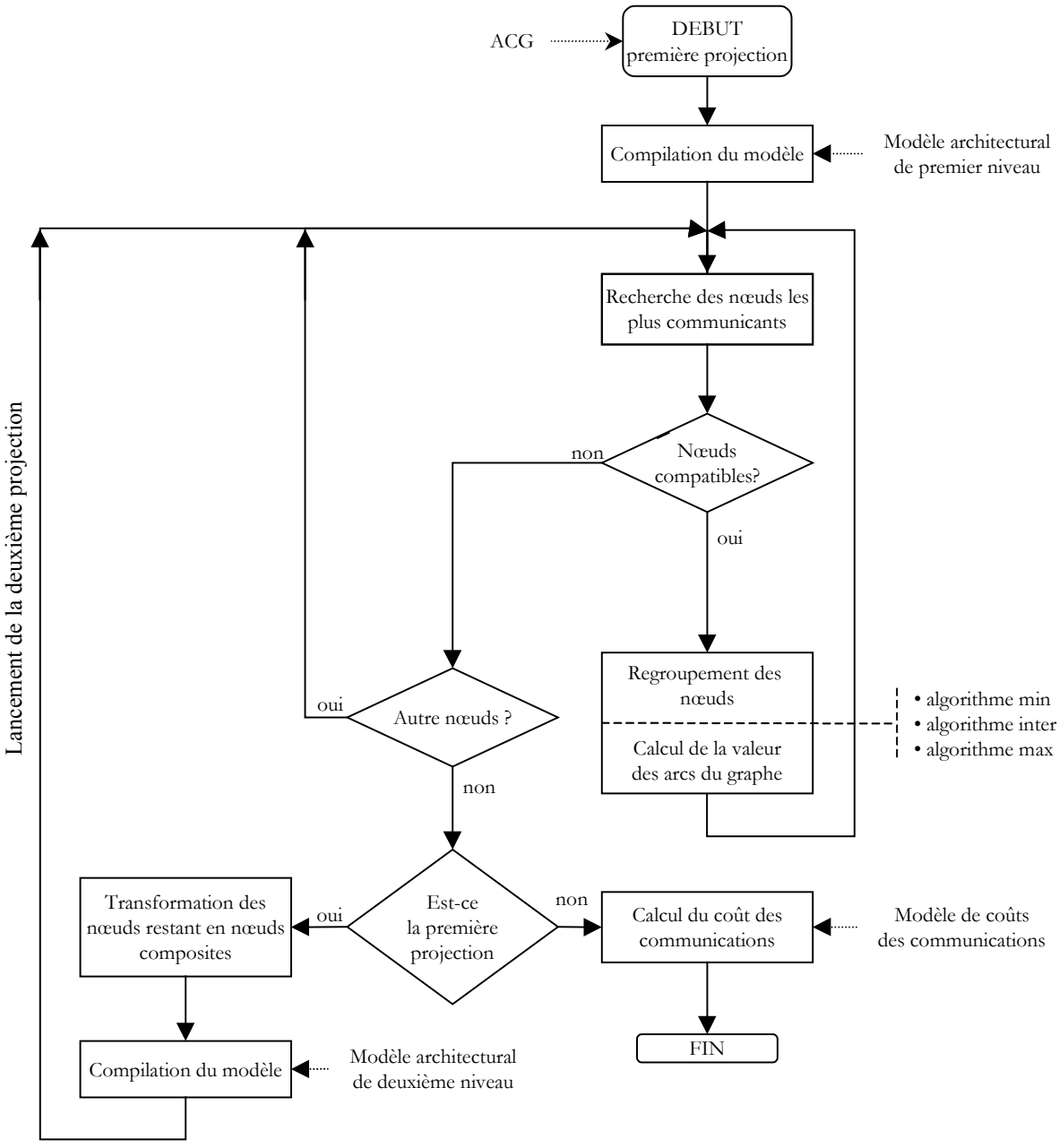


Figure 51 – Algorithme de projection matérielle

4.2.2 Première projection

La première projection débute avec un ACG directement issu de la transformation du HCDFG de l'application. La première étape de l'algorithme consiste en un parcours de cet ACG. Comme illustré sur la figure 51, l'algorithme cherche la paire de nœuds qui communique le plus, c'est à dire l'arc qui a la valeur relative la plus importante. Une fois les deux nœuds les plus communicants déterminés, l'algorithme étudie leur compatibilité. Deux nœuds représentant des opérations ou des mémorisations sont compatibles si et seulement si il existe un élément hiérarchique dans l'architecture modélisée qui dispose d'éléments fonctionnels non utilisés capables de réaliser les opérations ou mémorisations des deux nœuds. Si les nœuds sont compatibles alors leur regroupement virtuel au sein d'un même élément hiérarchique de l'architecture est modélisé par la création d'un nœud composite dans l'ACG. Ces nœuds composites sont différents des nœuds représentant les opérations et les mémorisations, ils représentent un mapping virtuel d'opérations et de mémorisations dans des éléments hiérarchiques de l'architecture. Afin de clarifier ces propos la figure 52 propose une explication schématique de ce qu'il se passe lors de l'étude de la compatibilité de deux nœuds et lors du regroupement de ces nœuds (création du nœud composite). Sur cette figure la valeur d'un arc donne le nombre de communications entre les deux nœuds qu'il relie.

Le cas (a) de la figure 52 présente un ACG partiel composé des deux nœuds les plus communicants d'un graphe. Il s'agit de deux nœuds traitements représentant m soustractions et n additions. La valeur x de l'arc représente le nombre de communications entre les deux nœuds. L'algorithme parcourt dans un premier temps le modèle de l'architecture ciblée. Il cherche si un élément hiérarchique de ce modèle contient un opérateur libre (non utilisé) capable de réaliser une soustraction et un opérateur libre capable de réaliser une addition. Si c'est effectivement le cas, l'algorithme conclut que les deux nœuds sont compatibles et sont réalisables dans un certain élément hiérarchique de l'architecture. Cet élément hiérarchique pourra, s'il lui reste des éléments fonctionnels non utilisés, être reconsidéré dans la suite pour la réalisation d'autres opérations. L'algorithme crée alors un nouveau nœud et le rajoute à l'ACG. Ce nœud est un nœud composite différent des nœuds traitements et des nœuds mémoires. Effectivement un nœud composite ne représente pas seulement une opération ou une mémorisation, mais un ensemble d'opérations et de mémorisations réalisées dans un élément hiérarchique précis. Dans le cas (a) de la figure 52, le nœud composite représente une addition et une soustraction réalisées dans un même élément hiérarchique. De ce fait il faut décrémenter le nombre d'opérations des deux nœuds. C'est à dire qu'il reste $m-1$ soustractions et $n-1$ additions à réaliser. Comme les nombres d'opérateurs sont changés et comme il y a un nouveau nœud dans le graphe (donc de nouveaux arcs) il est nécessaire de calculer les valeurs des arcs du graphe, mais aussi la quantité de communications qui vont être réalisées au sein de l'élément hiérarchique représenté par le nœud composite. Pour cela, et conformément à la figure 51, trois algorithmes vont être utilisés, ils seront présentés dans le paragraphe suivant.

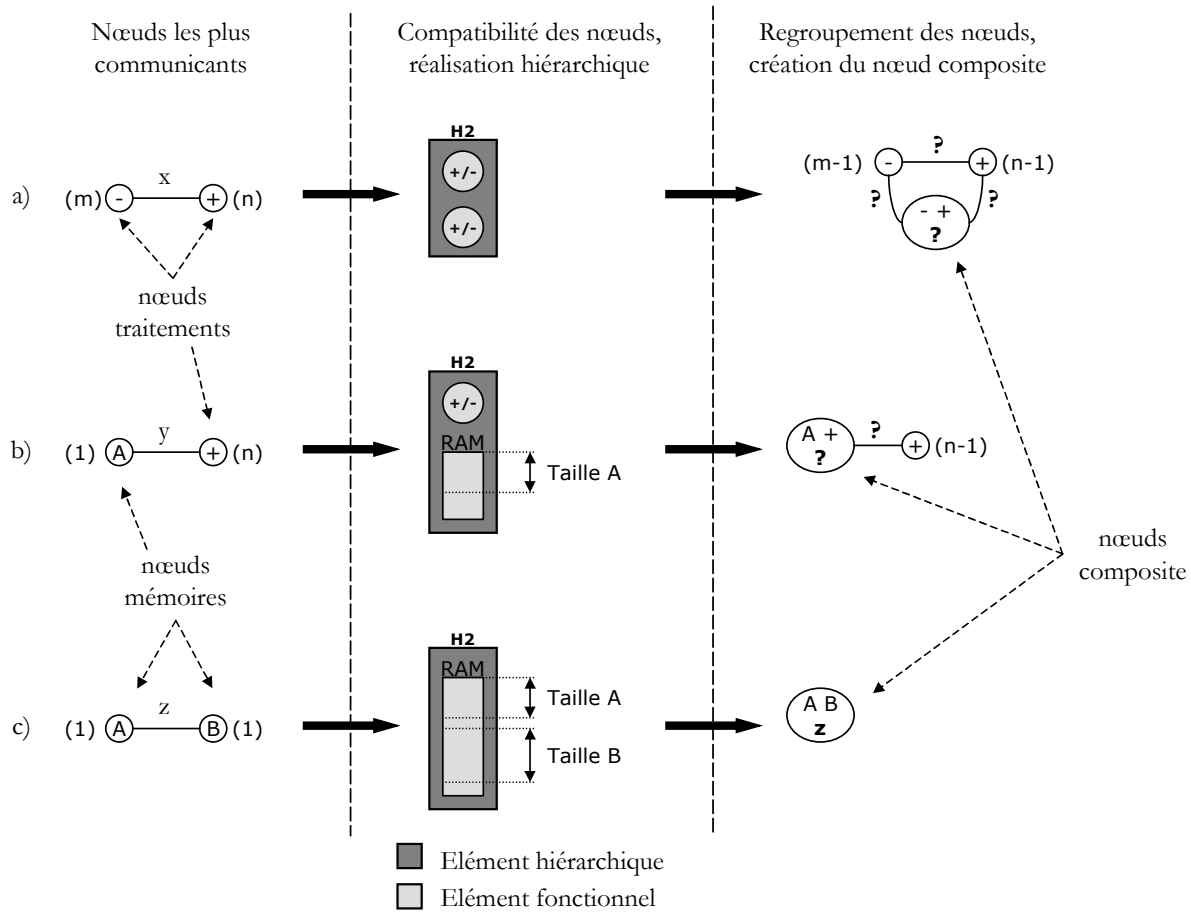


Figure 52 – Schématisation du déroulement de l'algorithme de projection.

Dans un souci de clarté, dans cette première approche nous laissons des points d'interrogation à la place des valeurs à calculer. En résumé, les nœuds composites ont comme propriétés la liste des opérations qu'ils représentent, une identification précise de l'élément hiérarchique dans lequel ces opérations sont réalisées et une estimation du nombre de communications réalisées dans cet élément hiérarchique.

Le cas (b) est un cas pour lequel un des deux nœuds les plus communicants de l'ACG représente une mémorisation. Ce nœud mémoire est systématiquement unitaire puisqu'il décrit la mémorisation d'une donnée multidimensionnelle unique, A. Il s'agit le plus souvent de tableaux issus de la spécification d'entrée. Il est donc possible de calculer la taille de mémoire nécessaire pour mémoriser la donnée A. Le résultat obtenu devient une propriété du nœud mémoire. L'algorithme parcourt le modèle de l'architecture ciblée pour chercher si un de ses éléments hiérarchiques contient un opérateur libre (non utilisé) capable de réaliser une addition et contient également une mémoire dont la capacité libre de mémorisation est supérieure ou égale à la capacité nécessaire à la mémorisation de la donnée A. Si c'est effectivement le cas, l'algorithme conclut que les deux nœuds sont compatibles et sont réalisables dans un certain élément hiérarchique de l'architecture.

Comme pour le premier cas, l'algorithme va créer un nœud composite représentant la mémorisation de la donnée A et une opération addition. Le nombre d'additions est décrémenté à *n-1*, le nœud mémoire est supprimé puisqu'il ne représentait qu'une mémorisation qui est désormais inclue dans le nouveau nœud composite. Il est encore nécessaire de calculer les nouvelles valeurs des arcs du graphe avec les trois algorithmes que nous présenterons dans la suite. Notons une nouvelle propriété du nœud composite qui est la liste des données mémorisées dans l'élément hiérarchique correspondant. Si le regroupement n'avait pas été possible la méthode ne permet pas de décomposer la donnée A dans plusieurs éléments mémoires.

Enfin, le dernier cas (le cas c) donne un exemple de regroupement de nœuds représentant chacun une mémorisation. Deux données multidimensionnelles A et B doivent nécessiter une capacité de mémorisation inférieure ou égale à la capacité interne de mémorisation d'un nœud hiérarchique pour pouvoir être compatibles. Dans ce cas un nœud composite représentant la mémorisation de A et de B dans un élément hiérarchique est créé. Les deux nœuds mémoires sont effacés du graphe puisqu'ils sont redondants avec le nœud composite.

Une fois le nœud composite créé et toutes les valeurs des arcs de l'ACG recalculées, l'algorithme reprend le parcours du graphe à la recherche de la paire de nœuds les plus communicants. Mais, il se peut que les nœuds les plus communicants ne soient pas compatibles, c'est à dire qu'aucun des éléments hiérarchiques de l'architecture ne dispose des opérateurs ou de la capacité de mémorisation nécessaire. Dans ce cas l'algorithme ne considère plus l'arc entre ces deux nœuds et recherche le prochain arc de plus grande valeur relative.

La figure 52 nous a montré le regroupement entre des nœuds du type nœuds traitements et/ou nœuds mémoires. Mais lorsque l'ACG contient des nœuds composites ils peuvent aussi être regroupés avec des nœuds traitements, des nœuds mémoires ou d'autres nœuds composites. La figure 53 schématise des regroupements pour lesquels au moins un des deux nœuds est un nœud composite.

Le cas (a) de la figure 53 est celui pour lequel les deux nœuds les plus communicants sont un nœud traitement et un nœud composite. Le nœud traitement représente *m* soustractions. Le nœud composite représente la réalisation dans un élément hiérarchique H2 d'une soustraction, d'une addition et de la mémorisation de la donnée A. Un nombre *w* de communications sont estimées comme internes à H2. L'algorithme doit donc vérifier que l'élément hiérarchique H2 est capable de réaliser tout ce qui est décrit dans le nœud composite (opérations plus mémorisations) plus une soustraction. Si c'est le cas alors le nœud composite évolue, on peut dire qu'il "grossit virtuellement". Effectivement il représente la réalisation d'une opération de plus et le nombre de communications internes à H2 est augmenté (le calcul sera présenté par la suite). Ainsi les nœuds composites peuvent évoluer au cours de l'exécution de l'algorithme de projection.

Le cas (b) considère le même nœud composite que celui du cas (a), mais cette fois il s'agit d'un regroupement avec un nœud mémoire. Ce dernier représente la mémorisation de la donnée multidimensionnelle B. L'étude de la comptabilité est proche de celle du cas précédent puisque l'algorithme vérifie seulement que la capacité mémoire de l'élément

hiérarchique H2 permet de mémoriser les données A et B. Si c'est le cas, le nœud composite absorbe le nœud mémoire représentant B. Le nombre de communications dans le nœud composite devient *w+y* puisque les arcs, donc toutes les communications, sont internes au nœud composite.

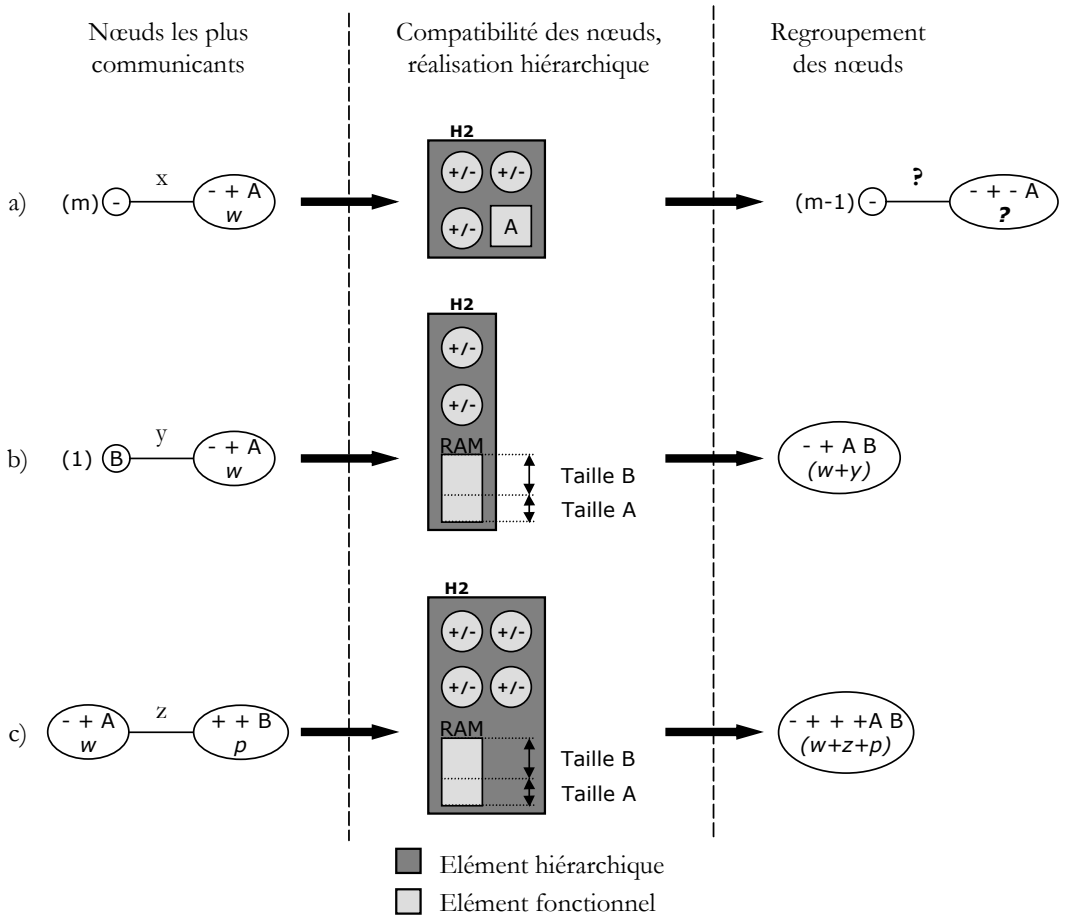


Figure 53 – Schématisation du déroulement de l'algorithme de projection avec des nœuds composites dans les regroupements.

Le dernier cas présente le regroupement de deux nœuds composites. Il s'agit alors de la réunion des cas (a) et (b) puisque l'algorithme vérifie que l'élément hiérarchique H2 comporte le nombre nécessaire d'opérateurs pour réaliser les opérations des deux nœuds composites et qu'il dispose d'une capacité de mémorisation suffisante pour mémoriser les données des deux nœuds composites. Si c'est le cas les deux nœuds sont regroupés pour n'en former qu'un seul. Puisque tous les arcs sont englobés dans le nœud composite résultant, le nombre de communications à l'intérieur de ce dernier est la somme du nombre de communications internes aux deux nœuds initiaux plus le nombre de communications entre ces deux nœuds.

Dans tous les cas l'algorithme de projection s'arrête lorsque plus aucune paire de nœuds ne peut être regroupée. Au final, l'ACG contient classiquement un nombre relativement grand de nœuds composites (cela dépend du nombre de traitements et de mémorisations à effectuer). Cependant il peut contenir également quelques nœuds mémoires ou traitements qui n'ont pu être regroupés avec d'autres nœuds. Plusieurs raisons peuvent être à la source de cette situation. Il est possible que les éléments fonctionnels dans les éléments hiérarchiques ne soient pas assez nombreux. Dans ce cas, des nœuds peuvent ne pas être regroupés dans un élément hiérarchique contenant des nœuds avec lesquels ils communiquent. Cela peut aussi venir du fait qu'aucune ressource de l'architecture ne peut en assurer la réalisation, il faut alors dans ce cas veiller à modifier la description de l'architecture. Les nœuds (traitements ou mémoires) qui ne sont pas assignés à un élément hiérarchique à la fin de la première projection vont être transformés en nœuds composites ne comportant qu'une opération ou mémorisation, afin d'effectuer la deuxième projection.

4.2.2 Deuxième projection

A la fin de la première projection, il y a une assignation entre les nœuds initiaux de l'ACG (nœuds traitement et nœuds mémoire) et les éléments fonctionnels de l'architecture. Ces éléments fonctionnels sont tous compris dans des éléments hiérarchiques de plus bas niveau dans l'architecture. L'ACG contient uniquement des nœuds composites à la fin de la première projection. Il y a trois niveaux de hiérarchie dans l'architecture, il est donc nécessaire de refaire une projection au niveau intermédiaire de hiérarchie. Cette fois l'algorithme va assigner les nœuds composites de l'ACG à des éléments hiérarchiques englobant les éléments hiérarchiques dans lesquels sont réalisés les traitements et mémorisations des nœuds composites. L'algorithme de projection est donc réalisé une seconde fois mais à un niveau supérieur de la hiérarchie. La seconde projection est nécessaire pour affiner les estimations de répartition des communications sur les trois niveaux hiérarchiques de l'architecture.

Pour réaliser la seconde projection il faut créer des nœuds "super-composites" qui représentent la réalisation de deux nœuds composites dans un même élément hiérarchique de niveau intermédiaire (deuxième niveau de la hiérarchie). Par exemple, si deux nœuds composites correspondent à deux éléments hiérarchiques H2 et si au niveau supérieur de la hiérarchie les éléments H1 englobent les éléments H2, alors les deux nœuds composites peuvent former un nœud super-composite qui est lié à H1. Ainsi nous continuons le rapprochement des opérateurs ou mémoires qui communiquent le plus mais à un niveau hiérarchique supérieur à celui pris en compte lors de la première projection.

La figure 54 schématise la seconde projection qui suit la même philosophie de regroupement des nœuds que la première projection. Dans ce cas les deux nœuds composites sont liés au même type d'élément hiérarchique (H2) mais ce n'est pas forcément nécessaire. Il faut que les deux éléments hiérarchiques qui sont liés aux deux nœuds composites soient inclus dans un même élément au niveau supérieur (ici l'élément hiérarchique H1). Alors les deux nœuds composites sont regroupés dans un nœud super-composite. La deuxième projection ne remet pas en cause les regroupements faits lors de la première projection.

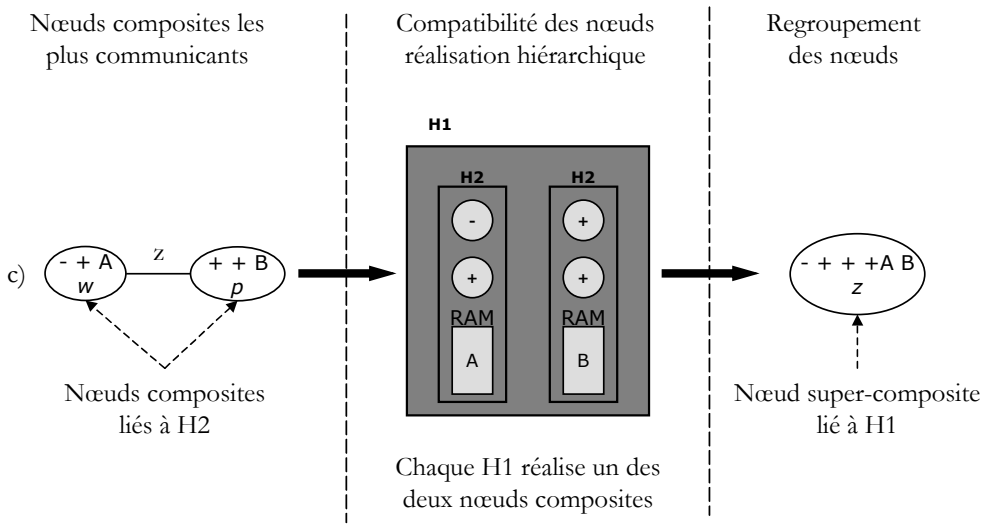


Figure 54 – Schématisation du déroulement de la seconde projection.

4.2.3 Calcul du coût des communications

A la fin de la première projection, en sommant l'estimation du nombre de communications à l'intérieur des nœuds composites on peut estimer le nombre de communications au niveau le plus bas de la hiérarchie. Dans le cas de la figure 54 il s'agirait des communications dans les éléments H2 qui sont estimées à w+p.

De même à la fin de la seconde projection, on peut connaître l'estimation du nombre de communications au deuxième niveau de hiérarchie (ou niveau intermédiaire puisque seulement trois niveaux de la hiérarchie sont considérés). Il s'agit du nombre de communications internes aux nœuds super-composites. Dans le cas de la figure 54 il s'agirait de l'estimation du nombre de communications à l'intérieur des éléments H1 de l'architecture qui est égal à z. Ce nombre correspond aux communications entre les éléments H2 qui sont compris dans l'élément H1.

Si il y a dans l'architecture plusieurs H1 compris dans un élément H0 supérieur, les communications entre les nœuds super-composites (donc entre les éléments H1) sont estimées. Ainsi nous disposons à la fin des deux projections des estimations du nombre de communications aux trois niveaux de la hiérarchie de l'architecture. Si le concepteur dispose d'un modèle de coûts des communications (typiquement une fonction de coût) il peut en déduire une valeur globale du coût des communications de l'utilisation de l'architecture ciblée pour réaliser la fonction estimée.

Les figures 52, 53 et 54 ont montré des cas simples ou seuls deux nœuds apparaissent. En réalité la paire de nœuds est incluse dans un graphe, c'est à dire que ces nœuds peuvent avoir des arcs avec d'autres nœuds. Lors de la création des nœuds composites des arcs sont créés vers les nœuds reliés aux nœuds d'origines. En particulier lorsqu'un nœud est effacé du graphe (c'est systématiquement le cas d'un nœud mémoire lorsque la donnée qu'il représente est mise dans un nœud composite), alors ses arcs sont directement reliés au

nœud composite le remplaçant. Ce processus est complexe car il est combiné au calcul des valeurs sur les arcs représentant les communications entre les nœuds. C'est pourquoi nous devons nous pencher plus sur les algorithmes de regroupement hiérarchique (ou algorithmes de clustering) qui réalisent ce processus. Ce sera l'objet du prochain paragraphe.

4.3 Algorithmes de regroupement hiérarchique

Comme nous l'avons vu précédemment dans la présentation de l'algorithme de projection et comme illustré sur la figure 51, trois algorithmes sous-jacents sont mis en jeu lors des changements de l'ACG. Ces algorithmes ont principalement pour but de déterminer les arcs de l'ACG à chaque transformation du graphe (rajout d'un nœud, effacement, regroupement) et de calculer les valeurs de tous les arcs.

4.3.1 Vers une approche multi-algorithmes

Plusieurs approches peuvent être envisagées afin d'évaluer les performances d'une application sur une architecture sur la base des coûts de communication. Une première approche consiste à effectuer une assignation optimale des opérations aux opérateurs. Ainsi il est possible d'avoir une vision précise des couples opérations/opérateurs et donc des interconnexions à mettre en œuvre. Avec une telle approche la définition d'un algorithme d'estimation associé à un algorithme d'optimisation doit conduire à une solution unique de bonne qualité. Toutefois ce type d'approche s'apparente à des techniques de compilation ou synthèse de haut niveau dont la complexité est rapidement élevée suivant les algorithmes mis en œuvre. Les performances des outils utilisés ont alors un fort impact sur les estimations finales. De plus, ces techniques nécessitent une modélisation fine des architectures. Certains travaux ont abordé le sujet avec cette approche et ont été contraints de prendre des facteurs correctifs afin de prendre en compte les optimisations des outils ainsi que le routage et aboutir à une solution unique, nous pouvons citer par exemple [Nayak02] et [Kulkani02].

Nous pensons que créer une approche aboutissant à une solution unique a montré ses limites étant donné le niveau d'abstraction considéré. Il est donc nécessaire de proposer une autre approche. Celle que nous proposons vise à lever la complexité en ne cherchant pas une assignation optimale des opérations aux opérateurs. Nous utilisons des processus d'assignations moins complexes qui permettent de borner les résultats d'estimations dans des intervalles. Ceux-ci permettent de lever le verrou lié aux outils.

Les intervalles d'estimations sont les résultats de trois algorithmes. Ils donnent un encadrement et une valeur intermédiaire comme résultats. Le choix final du concepteur est alors plus libre, il peut intégrer des éléments supplémentaires dans sa décision comme la complexité potentielle des outils qu'il sera nécessaire d'utiliser par la suite (outils de synthèse par exemple). La figure 55 présente un exemple de comparaison de trois architectures A, B et C suivant leur coût de communication (nous supposons alors qu'elles ont le même modèle de coûts).

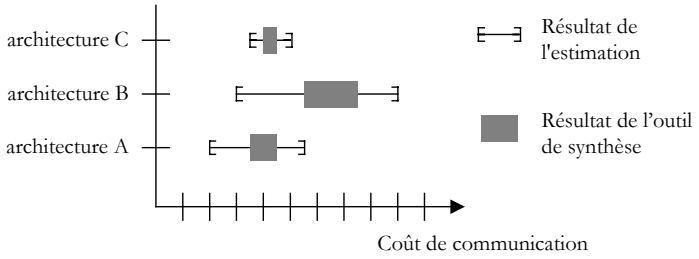


Figure 55 – Comparaison de trois architectures.

Sur cette figure, les intervalles représentent le résultat de l'estimation relative pour chaque architecture. Nous avons aussi représenté les résultats potentiels d'outils de synthèse sous forme d'un ensemble car ces outils peuvent donner plusieurs résultats en fonction d'options choisies par le concepteur. Ceci rend d'autant plus difficilement interprétable un résultat unique de l'estimation relative et renforce donc notre choix de donner un intervalle pour résultat. Les bornes minimum et maximum nous permettent de comparer l'efficacité en communication (donc en consommation de puissance) des architectures pour l'application développée. De plus, la largeur de l'intervalle peut nous donner une information supplémentaire particulièrement intéressante. Effectivement après quelques études nous avons établi qu'un large intervalle correspondait à des architectures pour lesquelles les solutions de conception (placement, allocation) étaient nombreuses. Au contraire pour les architectures donnant un intervalle d'estimation étroit, celles-ci proposent peu de choix de conception. Dans ce dernier cas il ne sera pas nécessaire que les algorithmes à utiliser par les outils de synthèse et de placement routage soient très performants car ils auront un impact sur les performances finales moins important que dans le cas où l'intervalle d'estimation est large. Effectivement dans ce cas les algorithmes seront devant des choix plus nombreux, il est donc nécessaire qu'ils soient particulièrement efficaces pour converger vers une solution optimale. Pour l'exemple de la figure 55 il est probable que l'architecture A soit plus efficace pour réaliser l'application estimée que l'architecture C car l'estimation du côut de communication est plus faible pour A que pour C, à condition d'utiliser des outils performants lors de la réalisation.

Donner les résultats de l'estimation relative sous forme d'intervalle nous paraît une bonne approche pour le concepteur, lui apportant une plus grande flexibilité dans ses choix architecturaux et des informations sur l'impact potentiel des outils qu'il devra utiliser par la suite.

4.3.2 Les algorithmes

Comme nous l'avons vu précédemment les algorithmes que nous allons utiliser ont pour but de créer ou d'éliminer les arcs de l'ACG lors de l'évolution de ce dernier consécutivement à des regroupements de nœuds. Ces algorithmes doivent calculer le nombre de communications à l'intérieur des nœuds composites. Nous les étudierons dans

le cadre de la première projection. En effet, ils sont identiques dans le cas de la seconde projection mais ne traitent que des nœuds composites (ou super-composites). Ces algorithmes aboutissent à trois répartitions différentes des communications dans les niveaux de hiérarchie de l'architecture. Si nous appliquons un modèle de coût de communications aux résultats alors l'algorithme minimum donne un coût de communication minimum car il répartit un maximum de communications au niveau hiérarchique le plus bas de l'architecture. Les communications à ce niveau (c'est à dire entre éléments fonctionnels) sont les moins coûteuses. Cet algorithme ne donne pas une vision réaliste de l'assignation des opérateurs et des communications mais une borne minimum. Effectivement la stratégie de cet algorithme est de considérer que deux nœuds regroupés dans un élément composite prennent en charge l'intégralité des communications entre les nœuds du même type.

L'algorithme maximum considère une répartition uniforme des communications : que les opérateurs (ou mémoires) soient ou non dans un même élément hiérarchique, l'algorithme impose qu'ils communiquent de la même façon. De ce fait les communications au plus bas niveau de la hiérarchie ne sont pas privilégiées par rapport aux communications aux niveaux plus élevés de la hiérarchie. L'algorithme maximum donne donc une valeur maximum du coût de communication.

Enfin, l'algorithme intermédiaire, contrairement à l'algorithme maximum, privilégie les communications locales au niveau le plus bas de la hiérarchie. C'est à dire que deux opérateurs (et/ou mémoires) dans un même élément hiérarchique vont communiquer plus que deux éléments du même type séparés hiérarchiquement dans l'architecture. Cependant cet algorithme ne considère pas, contrairement à l'algorithme minimum, que toutes les communications entre deux types de nœuds peuvent être prises en charge dans un seul nœud composite. En résumé l'algorithme intermédiaire donne une valeur du coût de communication comprise entre celles données par les algorithmes minimum et maximum.

Sur un exemple simple la différence entre les trois algorithmes se voit à deux niveaux, les arcs qui sont effacés et/ou créés après un regroupement de deux nœuds en nœuds composites, et le calcul de la valeur des arcs du graphe après ce regroupement. La figure 56 nous montre la transformation de l'ACG à la première étape de l'algorithme de projection dans un cas simple sur lequel nous verrons par la suite le détail des trois algorithmes. L'ACG comprend ici trois nœuds traitements qui représentent respectivement les opérations addition, soustraction et multiplication (cas classique des applications de traitements du signal). Pour ce graphe l'estimation système a proposé une solution nécessitant un additionneur, deux multiplieurs et deux soustracteurs. La valeur des arcs de l'ACG correspond au nombre de communications entre les nœuds reliés par ces arcs.

Pour les trois algorithmes la recherche des nœuds les plus communicants est la même. Pour le cas de l'ACG de la figure 56, ce sont les nœuds multiplieur et soustracteur qui communiquent le plus. Pour l'exemple il y a vingt échanges de données entre les deux nœuds pour deux opérateurs multiplieurs et deux opérateurs soustracteurs. Les trois algorithmes vont donc créer un nœud composite représentant l'assignation d'un multiplieur et d'un soustracteur dans un même élément hiérarchique, après avoir bien sûr vérifié que

cela était compatible avec l'architecture modélisée. L'ACG résultant contient donc un nœud supplémentaire qui est un nœud composite.

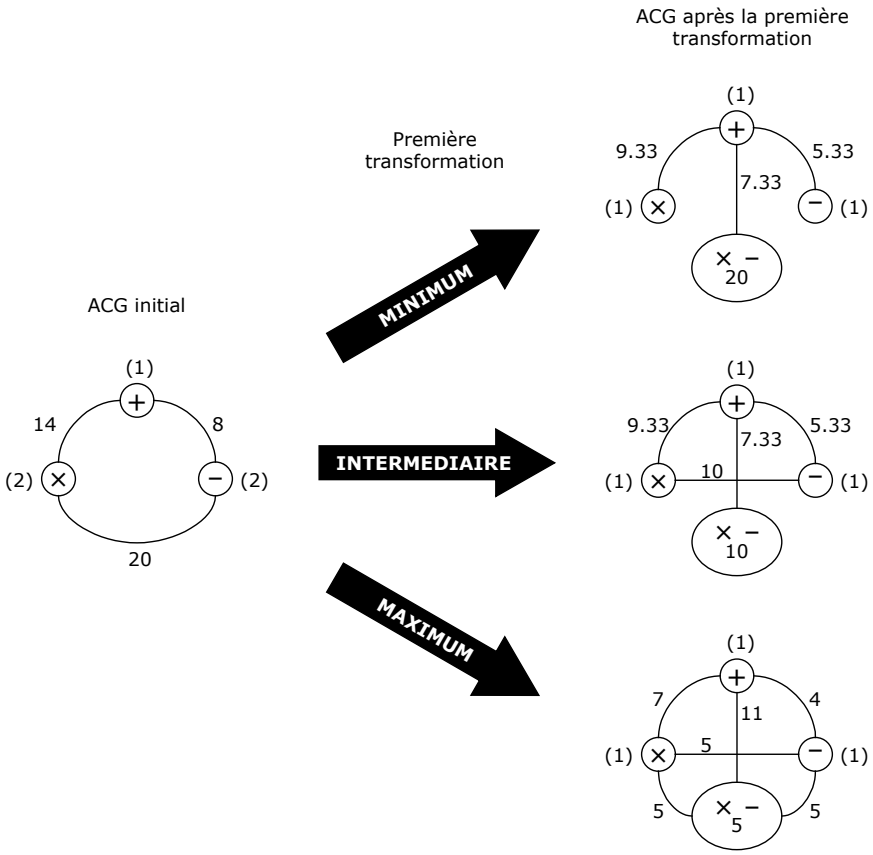


Figure 56 – Première transformation de l'ACG suivant les trois algorithmes.

Comme nous l'avons dit, la différence entre les trois algorithmes se fait au niveau des communications prises en charge dans l'élément composite et au niveau des arcs de ce nœud. Pour l'algorithme minimum 100 % des communications entre les opérateurs multiplieurs et soustracteurs sont réalisées par les deux opérateurs que représente le nœud composite. Ce nombre, pour l'exemple proposé dans la figure 56, est réduit à 50 % pour l'algorithme intermédiaire et seulement 25 % pour l'algorithme maximum. Dans ce dernier cas le nœud composite a plus d'arcs que dans les deux autres cas, car les communications entre les nœuds sont plus réparties et tiennent moins compte du regroupement hiérarchique des opérateurs. Dans le cas de l'algorithme minimum l'arc entre les nœuds multiplieur et soustracteur est effacé car toutes les communications entre ces types d'opérateurs sont maintenant liées au nœud composite.

Pour comprendre les algorithmes sous forme textuelle (algorithme 1, algorithme 2 et algorithme 3) nous devons introduire un certain nombre de notations utilisées par la suite. Notons que dans un souci de simplicité nous ne considérons que des nœuds du type

traitement dans les algorithmes que nous présentons dans les paragraphes suivants, mais les nœuds mémoires sont traités de la même façon.

- N_i: nœud i de l'ACG,
- t_i: type de traitement du nœud N_i ,
- n_{ti}: nombre d'opérations du nœud N_i (résultat de l'estimation système),
- C_{ij}: nœud composite représentant deux traitements t_i et t_j,
- CI_{Cij}: nombre de communications internes au nœud C_{ij},
- $A_{i,j}$: arc entre les nœuds N_i et N_j,
- $P_{i,j}$: nombre de communications entre les nœuds N_i et N_j (valeur associée à l'arc $A_{i,j}$),
- $A_{k,ij}$: arc entre le nœud N_k et le nœud composite C_{ij},
- $P_{k,ij}$: nombre de communications entre le nœud N_k et le nœud composite C_{ij} (valeur associée à l'arc $A_{k,ij}$).

De plus, des fonctions sont utilisées par les trois algorithmes :

- CREATION_ COMPOSITE(t_i, t_j) : cette fonction crée et ajoute à l'ACG un nouveau nœud composite représentant les deux traitements entre parenthèses,
- CREATION_ARC(N_i, N_j) : cette fonction crée et ajoute à l'ACG un arc entre le nœud N_i et le nœud N_j,
- EFFACE_ARC($A_{i,j}$) : cette fonction efface de l'ACG l'arc $A_{i,j}$ entre le nœud N_i et le nœud N_j.,
- MIN($x1$, $x2$) : cette fonction retourne le plus petit nombre des deux nombres entre parenthèses.

Pour aider à la compréhension, les trois algorithmes vont être illustrés par des exemples didactiques (figure 57, figure 58 et figure 59) à partir de l'ACG de la figure 56. L'architecture modélisée pour ces exemples est également très simple, et ne comporte que deux niveaux de hiérarchie. Le niveau le plus haut est réalisé par un élément H1 qui contient deux éléments H2. Ce dernier contient deux éléments fonctionnels capables de réaliser une addition ou une soustraction, et un élément fonctionnel capable de réaliser une multiplication.

Chacun des algorithmes présentés débute avec la création d'un nœud composite. Nous considérons que l'algorithme principal de projection (figure 51) a déjà déterminé la paire de nœuds la plus communicante de l'ACG.

4.3.3 L'algorithme intermédiaire (algorithme 1)

La philosophie de cet algorithme est la plus proche de notre postulat qui consiste à dire que des opérations implémentées dans des éléments fonctionnels d'un même élément hiérarchique doivent communiquer davantage que des opérations réalisées dans des

éléments hiérarchiques différents, ceci afin de profiter un maximum de la connectivité locale au sein d'un élément hiérarchique.

Pour clarifier ces propos nous allons voir l'algorithme intermédiaire via l'exemple de la figure 57. Sur cette figure, la première étape de l'algorithme va regrouper une opération multiplication et une opération soustraction car les deux nœuds correspondants sont les plus communicants. C'est ce qui est illustré à la seconde étape où les valeurs des arcs sont calculées. Tout d'abord le nœud composite ainsi créé prend en interne la moitié des communications entre les deux opérations puisqu'il prend dans ce cas la moitié du nombre d'opérations. Parallèlement une assignation de ressources aux opérations est faite dans l'architecture (éléments fonctionnels grisés sur la figure 57). Le nœud composite est seulement relié au nœud représentant l'addition. Effectivement comme les nombres d'opérations multiplication et soustraction, à l'intérieur et à l'extérieur du nœud, sont égaux le nœud composite prend la moitié des communications et laisse l'autre moitié aux nœuds initiaux.

A l'étape deux, l'algorithme trouve que la nouvelle paire la plus communicante est encore une fois la paire de nœuds représentant les opérations multiplication et soustraction. Un nœud composite est créé et les deux nœuds représentant initialement les multiplications et les soustractions sont effacés car toutes les opérations de ce type sont assignées à des éléments fonctionnels de l'architecture (elles sont donc dans des nœuds composites). Puis l'algorithme regroupe le nœud représentant l'addition et le nœud composite avec lequel il communique le plus. A la dernière étape il ne reste que des nœuds composites. Ils ne peuvent pas être regroupés car les éléments hiérarchiques H2 de l'architecture ne disposent pas d'un nombre suffisant d'éléments fonctionnels.

L'algorithme 1 va nous permettre de généraliser notre approche. Cet algorithme débute par la création d'un nœud composite C_{ij} en fonction du couple de nœuds traitements initiaux à regrouper. Le nombre de communications qu'il englobe dépend du nombre de communications entre les nœuds initiaux P_{ij}. Il est calculé par la fonction i) de l'algorithme 1. Dans le cas où les nombres d'opérations pour les deux nœuds, n_{ti} et n_{tj} sont égaux, comme dans le cas de la figure 57, alors l'élément composite prend en charge une fraction du nombre de communications égale à :

$$CI_{c_{ij}} = \frac{p_{i,j}}{n_{t_i}}$$

La deuxième étape de l'algorithme correspond à la création des arcs entre le composite et les nœuds initiaux. Si le nombre d'opérations est le même pour les deux nœuds, le nœud composite n'est relié à aucun des deux nœuds initiaux sinon il est relié uniquement à celui des deux nœuds qui a le plus d'opérations. La valeur des arcs est calculée par les fonctions ii) et iii) de l'algorithme 1. Enfin, il faut créer des liaisons entre le nœud composite et tous les nœuds qui étaient initialement en relation avec les deux nœuds initiaux. Effectivement si un nœud traitement N_k était relié à un nœud traitement N_i et qu'une opération N_i est regroupée dans un élément composite C_{ij}, il faut créer une liaison entre N_k et C_{ij} afin de représenter les communications du nœud traitement N_k avec l'opération N_i du composite C_{ij}. Cependant si le nœud traitement N_i n'est pas effacé, il s'agit du cas où il représente

encore des opérations, l'arc entre N_k et N_i est conservé. La fonction iv) permet de calculer la valeur de l'arc entre N_k et C_{ij}. La fonction v) retire la valeur précédemment calculée à la valeur de l'arc entre N_k et N_i.

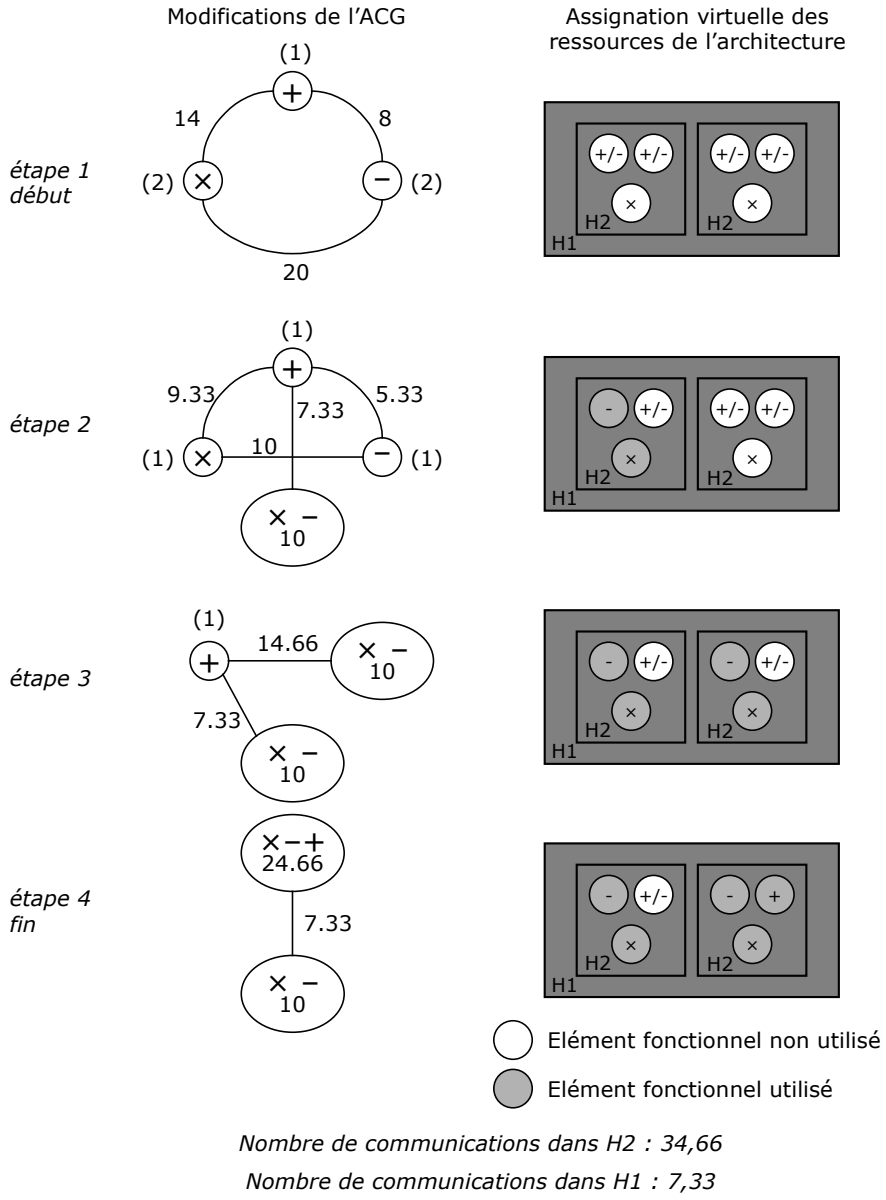


Figure 57 – Exemple de l'exécution de l'algorithme intermédiaire.

Début

C_{ij} = CREATION_ COMPOSITE(t_i ,t_j)

$$CI_{c_{ij}} = MIN\left(\frac{p_{i,j}}{n_{t_i}}, \frac{p_{i,j}}{n_{t_j}}\right) \quad \text{i)}$$

si ($n_{ti} > n_{tj}$)

$A_{i,ij}$ = CREATION_ARC(N_i, C_{ij})

$$p_{i,ij} = \frac{p_{i,j}}{n_{t_i}} - \frac{p_{i,j}}{n_{t_j}} \quad \text{ii)}$$

sinon si ($n_{ti} < n_{tj}$)

$A_{j,ij}$ = CREATION_ARC(N_j, C_{ij})

$$p_{j,ij} = \frac{p_{i,j}}{n_{t_j}} - \frac{p_{i,j}}{n_{t_i}} \quad \text{iii)}$$

fin si

pour tout $N_k \in$ ACG

/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*

si ($N_k \neq N_i$, $N_k \neq N_j$, $N_k \neq C_{ij}$)

si ($A_{k,i}$ existe)

$A_{k,ij}$ = CREATION_ARC(N_k, C_{ij})

$$p_{k,ij} = \frac{p_{k,i}}{n_{t_k} + n_{t_i}} \quad \text{iv)}$$

fin si

$$p_{k,i} = \frac{p_{k,i} \times (n_{t_k} + n_{t_i} - 1)}{n_{t_k} + n_{t_i}} \quad \text{v)}$$

fin si

/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*

/ idem que pour le nœud N_j mais avec le nœud N_j */*

fin pour

$$p_{i,j} = p_{i,j} - IC_{c_{ij}} - p_{i,ij} - p_{j,ij}$$

$$n_{t_i} = n_{t_i} - 1 \quad \text{vi)}$$

$$n_{t_j} = n_{t_j} - 1$$

fin

Algorithme 1 – L'algorithme intermédiaire.

La dernière étape consiste, avec la fonction vi), à retirer à la valeur de l'arc entre les nœuds traitements initiaux ($N_i$et N_j), la valeur des arcs entre ces nœuds et le nœud composite, ainsi que le nombre de communications prises en charge dans l'élément composite. Enfin, le

nombre d'opérations des nœuds traitements initiaux est décrémenté pour prendre en compte qu'une opération est assignée à un élément hiérarchique donc liée à un nœud composite.

4.3.4 L'algorithme minimum (algorithme 2)

Cet algorithme ne donne pas une vision réaliste de la répartition des communications mais une borne minimum du coût des communications. C'est à dire qu'il ne peut pas avoir mieux que les résultats de cet algorithme. Son principe consiste en effet à dire que des opérations réalisées dans des éléments fonctionnels d'un même élément hiérarchique prennent en charge l'ensemble des communications entre les opérations impliquées. Si deux opérations sont regroupées dans un nœud composite et donc assignées à un élément hiérarchique, l'arc entre les nœuds qui les représentent est effacé. Effectivement si toutes les communications sont réalisées en interne à l'élément hiérarchique, donc inscrites dans le nœud composite, elles n'existent plus entre les nœuds initiaux.

Pour clarifier ces propos nous allons voir l'algorithme minimum via l'exemple de la figure 58. Cette figure décrit sur un cas simple le déroulement de l'algorithme minimum. A la première étape l'algorithme regroupe une opération multiplication et une opération soustraction car les deux nœuds correspondants sont les plus communicants. C'est ce qui est illustré à la seconde étape où les valeurs des arcs sont calculées. Tout d'abord le nœud composite ainsi créé va prendre en interne la totalité des communications entre les deux opérations. Alors l'arc entre les nœuds représentant les deux opérations regroupées est effacé puisque toutes les communications sont virtuellement associées au nœud composite. Parallèlement une assignation de ressources aux opérations est faite dans l'architecture. Le nœud composite est seulement relié au nœud représentant l'addition puisqu'il n'y a plus d'autres communications entre les opérations multiplication et soustraction. Le calcul des valeurs des arcs est identique à celui fait lors de l'algorithme intermédiaire, la différence entre les deux algorithmes tient essentiellement au nombre de communications associées aux nœuds composites.

A l'étape deux, l'algorithme trouve que la nouvelle paire la plus communicante est la paire de nœuds représentant les opérations multiplication et addition. Un nœud composite est créé et les deux nœuds représentant initialement les multiplications et l'addition sont effacés car toutes les opérations de ce type sont assignées à des éléments fonctionnels de l'architecture (elles sont donc dans des nœuds composites). Puis l'algorithme regroupe le nœud représentant la soustraction restante et le nœud composite avec lequel il communique le plus. A la dernière étape il ne reste que des nœuds composites. Ils ne peuvent pas être regroupés car les éléments hiérarchiques H2 de l'architecture ne disposent pas d'un nombre suffisant d'éléments fonctionnels.

L'algorithme 2 va nous permettre de généraliser notre approche. Cet algorithme débute par la création d'un nœud composite C_{ij} en fonction du couple de nœuds traitements initiaux à regrouper. Le nombre de communications qu'il englobe est directement égal au nombre de communications entre les nœuds initiaux P_{ij} suivant le principe même de cet algorithme. C'est ce que montre la fonction i) de l'algorithme 2. La deuxième étape de l'algorithme correspond à la création des arcs entre le composite et les nœuds qui étaient initialement en

relation avec les deux nœuds initiaux. Effectivement si un nœud traitement N_k était relié à un nœud traitement N_i et qu'une opération N_i est regroupée dans un élément composite C_{ij}, il faut créer une liaison entre N_k et C_{ij} afin de représenter les communications du nœud traitement N_k avec l'opération N_i du composite C_{ij}. Cependant si le nœud traitement N_i n'est pas effacé, dans le cas où il représente encore des opérations, l'arc entre N_k et N_i est conservé. La fonction ii) permet de calculer la valeur de l'arc entre N_k et C_{ij}. La fonction iii) calcule la nouvelle valeur de l'arc entre N_k et N_i. Notons bien ici la différence avec l'algorithme intermédiaire (et nous le verrons avec l'algorithme maximum) : dans aucun cas il n'y a d'arc entre le nœud composite et les deux nœuds initiaux. Effectivement toutes les communications entre les opérations représentées par les deux nœuds initiaux sont virtuellement réalisées à l'intérieur du nœud composite.

---

début
C_{ij} = CREATION_COMPOSITE(t_i ,t_j)
$IC_{c_{ij}} = p_{i,j}$ *i)*
pour tout $N_k \in$ ACG
/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*
si ($N_k \neq N_i$, $N_k \neq N_j$, $N_k \neq C_{ij}$)
si ($A_{k,i}$ existe)
$A_{k,ij}$ = CREATION_ARC(N_k, C_{ij})

$$p_{k,ij} = \frac{p_{k,i}}{n_{t_k} + n_{t_i}} \qquad \text{ii)}$$

fin si

$$p_{k,i} = \frac{p_{k,i} \times (n_{t_k} + n_{t_i} - 1)}{n_{t_k} + n_{t_i}} \qquad \text{iii)}$$

fin si
/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*
/ idem que pour le nœud N_j mais avec le nœud N_j */*
…
fin pour
$n_{t_i} = n_{t_i} - 1$
$n_{t_j} = n_{t_j} - 1$
EFFACE_ARC($A_{i,j}$)
fin

Algorithme 2 – L'algorithme minimum.

La dernière étape consiste à décrémenter le nombre d'opérations des nœuds traitements initiaux (N_i et N_j) pour prendre en compte qu'une opération est assignée à un élément hiérarchique, donc liée à un nœud composite, puis à effacer l'arc entre les deux nœuds initiaux puisque toutes les communications sont virtuellement internes au nœud composite.

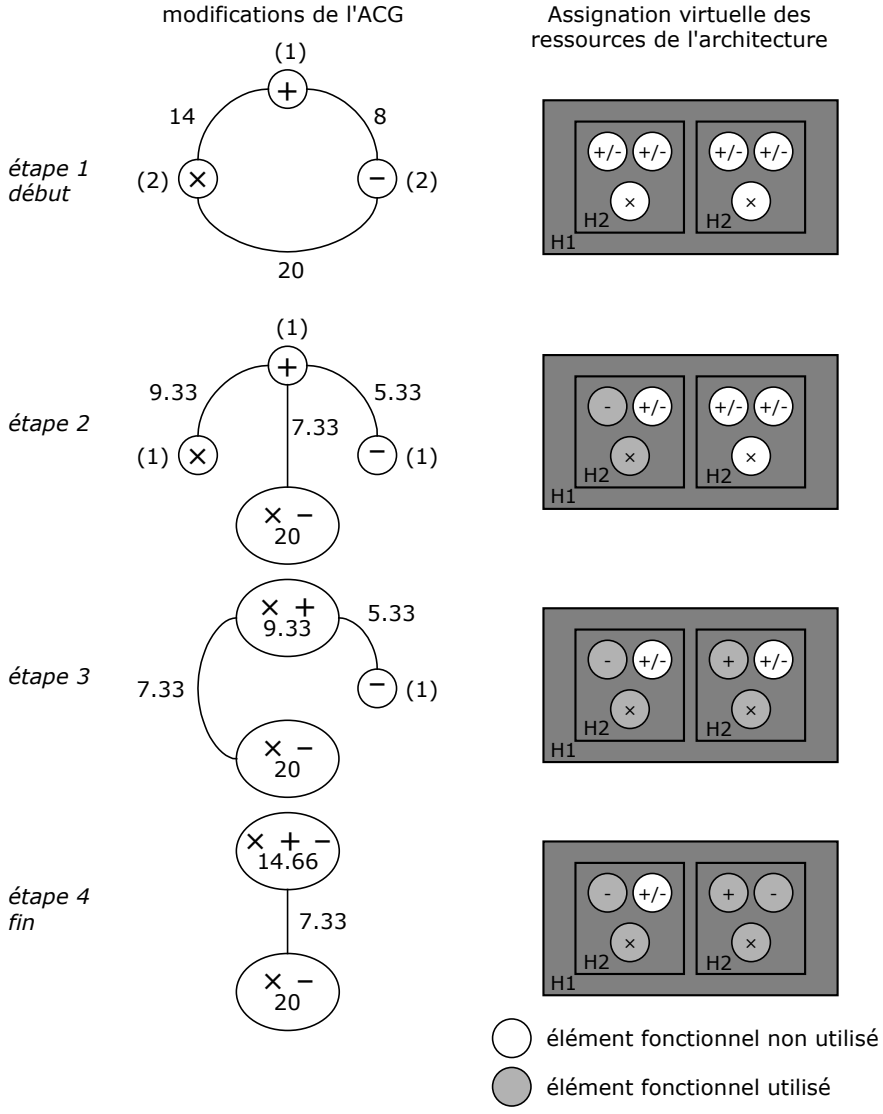


Figure 58 – Exemple de l'exécution de l'algorithme minimum.

4.3.5 L'algorithme maximum (algorithme 3)

Cet algorithme donne une borne que nous qualifions de maximum du coût de communication. Effectivement cet algorithme ne répartit pas les communications entre opérateurs en fonction de leur réalisation hiérarchique. C'est à dire que des opérations réalisées dans des éléments fonctionnels d'un même élément hiérarchique ont en charge autant de communications que si elles n'étaient pas réalisées dans le même élément hiérarchique. En modifiant l'ACG cet algorithme crée un maximum d'arcs. Effectivement le nœud composite créé lors du regroupement de deux opérations est lié aux nœuds qui les représentent, ainsi qu'à tous les nœuds du graphe qui sont liés à au moins un des deux nœuds initiaux. Pour clarifier ces propos nous allons voir l'algorithme maximum via l'exemple de la figure 59.

La figure 59 décrit sur un cas simple le déroulement de l'algorithme maximum. A la première étape l'algorithme regroupe une opération multiplication et une opération soustraction car les deux nœuds correspondants sont les plus communicants. C'est ce qui est illustré à la seconde étape où les valeurs des arcs sont calculées. Tout d'abord le nœud composite ainsi créé prend en interne un quart des communications entre les deux opérations puisque alors il y aura quatre possibilités de communications. L'arc entre les nœuds représentant les deux opérations regroupées prend alors la même valeur ainsi que les deux arcs créés entre ces deux nœuds et le nœud composite. Ainsi les communications sont virtuellement réparties de façon homogène. Parallèlement une assignation de ressources aux opérations est faite dans l'architecture. Le nœud composite a donc au final trois arcs dans ce cas précis. Le calcul des valeurs des arcs est un calcul de répartition moyenne entre le nombre de communications et le nombre d'arcs plus un (qui correspond à l'intérieur du nœud composite).

A l'étape deux, l'algorithme trouve que la nouvelle paire la plus communicante est le nœud représentant l'addition avec le nœud composite créé à la première étape. Ce nœud composite absorbe donc le nœud addition. Puis l'algorithme regroupe les nœuds multiplication et soustraction restants. A la dernière étape il ne reste que des nœuds composites. Ils ne peuvent pas être regroupés car les éléments hiérarchiques H2 de l'architecture ne disposent pas d'un nombre suffisant d'éléments fonctionnels.

L'algorithme 3 va nous permettre de généraliser notre approche. Cet algorithme débute par la création d'un nœud composite C_{ij} en fonction du couple de nœuds traitements initiaux à regrouper. Le nombre de communications qu'il englobe dépend du nombre de communications entre les nœuds initiaux P_{ij}. Il est calculé par la fonction i) de l'algorithme 3. La deuxième étape de l'algorithme correspond à la création des arcs entre le nœud composite et les deux nœuds initiaux. La valeur des arcs est calculée par les fonctions ii) et iii) de l'algorithme 3. Puis comme pour les deux algorithmes précédents, il faut créer des liaisons entre le nœud composite et tous les nœuds qui étaient initialement en relation avec les deux nœuds initiaux. Effectivement si un nœud traitement N_k était relié à nœud traitement N_i et qu'une opération du nœud N_i est regroupée dans l'élément composite C_{ij}, il faut créer une liaison entre N_k et C_{ij} afin de représenter les communications du nœud traitement N_k avec l'opération N_i du composite C_{ij}. Cependant si le nœud traitement N_i n'est pas effacé, dans le cas où il représente encore des opérations, l'arc entre N_k et N_i est

conservé. La fonction iv) permet de calculer la valeur de l'arc entre N_k et C_{ij}. La fonction v) calcule la nouvelle valeur de l'arc entre N_k et N_i.

---

début
C_{ij} = CREATION_COMPOSITE(t_i ,t_j)

$$IC_{c_{ij}} = \frac{p_{i,j}}{n_{t_i} \times n_{t_j}} \qquad \text{i)}$$

$A_{i,ij}$ = CREATION_ARC(N_i, C_{ij})

$$p_{i,ij} = \frac{p_{i,j} \times (n_{t_i} - 1)}{n_{t_i} \times n_{t_j}} \qquad \text{ii)}$$

$A_{j,ij}$ = CREATION_ARC(N_j, C_{ij})

$$p_{j,ij} = \frac{p_{i,j} \times (n_{t_j} - 1)}{n_{t_i} \times n_{t_j}} \qquad \text{iii)}$$

pour tout $N_k \in$ ACG
/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*
si ($N_k \neq N_i$, $N_k \neq N_j$, $N_k \neq C_{ij}$)
si ($A_{k,i}$ existe)
$A_{k,ij}$ = CREATION_ARC(N_k, C_{ij})

$$p_{k,ij} = \frac{p_{k,i}}{n_{t_i}} \qquad \text{iv)}$$

fin si

$$p_{k,i} = \frac{p_{k,i} \times (n_{t_i} - 1)}{n_{t_i}} \qquad \text{v)}$$

fin si
/ calcul de la valeur des arcs entre le nœud N_k, le nœud N_j et le nœud composite C_{ij} */*
/ idem que pour le nœud N_j mais avec le nœud N_j */*
fin pour

$$p_{i,j} = \frac{p_{i,j} \times (n_{t_i} - 1)(n_{t_j} - 1)}{n_{t_i} \times n_{t_j}}$$

$$n_{t_i} = n_{t_i} - 1 \qquad \text{vi)}$$

$$n_{t_j} = n_{t_j} - 1$$

fin

Algorithme 3 – L'algorithme maximum.

Nous pouvons remarquer que les fonctions de calcul de la valeur des arcs utilisées dans cet algorithme sont différentes de celles utilisées dans l'algorithme intermédiaire afin d'obtenir

dans ce cas une répartition plus homogène des communications entre les nœuds. Ce qui conduit à ne pas privilégier ici la localité des communications.

La dernière étape consiste avec la fonction vi) à calculer la valeur de l'arc entre les nœuds traitements initiaux (N_i et N_j). Enfin, le nombre d'opérations des nœuds traitements initiaux est décrémenté pour prendre en compte qu'une opération est assignée à un élément hiérarchique donc liée à un nœud composite.

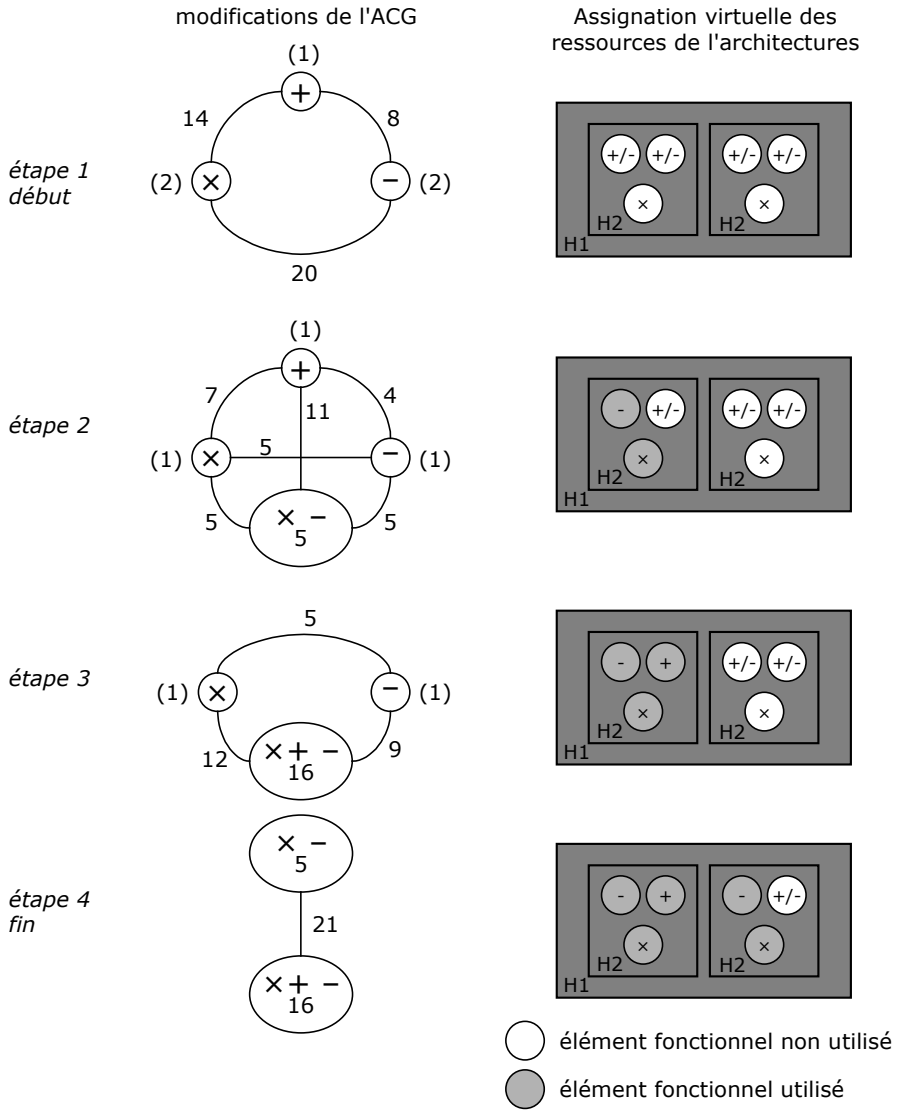


Figure 59 – Exemple de l'exécution de l'algorithme maximum.

4.3.6 Calcul du coût des communications

A l'issue des trois algorithmes on peut extraire le nombre de communications internes aux éléments H2 (plus bas niveau de la hiérarchie) en sommant les communications internes aux nœuds composites. On peut aussi extraire le nombre de communications internes à l'élément H1 (donc entre les éléments H2) en sommant le nombre de communications entre les nœuds composites. Ainsi on est capable de fournir une répartition virtuelle des communications dans les différents niveaux de la hiérarchie de l'architecture.

Avec les exemples simples des figures 57, 58 et 59 on obtient les résultats suivants :

- Pour l'algorithme minimum plus de 82 % des communications sont estimées être réalisées à l'intérieur des éléments H2.
- Pour l'algorithme intermédiaire plus de 82 % des communications sont estimées être réalisées à l'intérieur des éléments H2.
- Pour l'algorithme maximum 50 % des communications sont estimées être réalisées à l'intérieur des éléments H2 et donc 50 % en externe (dans H1).

Le résultat obtenu est le même pour les algorithmes minimum et intermédiaire car l'exemple choisi (ACG plus modèle d'architecture) est très simple et que les deux algorithmes sont assez proches. Ils utilisent entre autres les même fonctions pour le calcul des nouvelles valeurs des arcs. Cependant dès que les exemples sont plus importants et donc plus proches de la réalité, ce n'est plus le cas. L'algorithme minimum donne toujours un nombre supérieur (ou égal dans les cas très simples) de communications au plus bas niveau de la hiérarchie par rapport au nombre donné par l'algorithme intermédiaire. Donc le coût des communications est minimum. L'algorithme maximum donne une estimation du nombre de communications au niveau le plus bas toujours inférieur aux estimations de l'algorithme minimum et de l'algorithme intermédiaire. Ceci donne une estimation maximum du coût des communications puisqu'elles sont plus nombreuses aux niveaux les plus hauts de la hiérarchie de l'architecture.

Pour obtenir directement le coût des communications il faut disposer d'une fonction de coût qui détermine le coût d'une communication à chaque niveau de la hiérarchie de l'architecture. Ce qui sous-entend que l'utilisateur dispose d'une très bonne connaissance de l'architecture modélisée, en particulier en ce qui concerne les ressources de routage.

4.3.7 Conclusion sur ces trois algorithmes

Nous avons vu les trois algorithmes donnant des estimations de la répartition des communications de façon minimum (coût de communication minimum), maximum ou intermédiaire. Les exemples que nous avons montrés sont très simples et ils n'ont qu'un but pédagogique, en réalité des cas plus complexes sont à traiter et les algorithmes prennent en compte un nombre bien plus important de cas. Par exemple nous n'avons pas vu en détail comment nous effaçons un nœud du graphe, car il faut alors prendre en compte le fait que ce nœud est relié à d'autres nœuds par des arcs et que ces arcs représentent des communications, or il ne faut pas perdre de communications au cours de l'évolution de l'ACG. D'ailleurs, une façon de vérifier l'intégrité de l'ACG au cours des modifications

successives est de calculer le nombre de communications représentées par le graphe en sommant la valeur des arcs et le nombre de communications internes aux nœuds composites. Le nombre total de communications doit rester constant. Nous n'avons donc pas détaillé toutes les possibilités prises en compte par les algorithmes.

Nous avons vu le flot de projection et d'estimation de la répartition des communications utilisé dans l'estimation relative. Les algorithmes mis en œuvre sont extrêmement moins complexes que ceux nécessaires à une synthèse haut niveau. Aussi ils nous permettent d'obtenir très rapidement une estimation de l'assignation des opérations (ou des mémorisations) du graphe initial aux opérateurs (ou aux mémoires) de l'architecture, ceci suivant trois stratégies. Grâce à cela nous obtenons l'estimation de la répartition des communications dans l'architecture modélisée. Les résultats obtenus sont donnés sous forme d'intervalle avec une valeur intermédiaire. Ces résultats doivent permettre de guider l'exploration architecturale. Nous allons donc maintenant voir comment utiliser ce flot dans l'exploration de l'espace de conception architecturale.

4.4 Méthode d'exploration de l'espace de conception architecturale

4.4.1 Flot d'exploration

L'exploration que nous proposons est principalement une exploration architecturale, ce qui veut dire que nous cherchons l'architecture la plus en adéquation avec l'application spécifiée en entrée. Cependant nous pourrions aussi fixer l'architecture et chercher à spécifier l'application de telle façon qu'elle soit plus adaptée à l'architecture, mais ce n'est pas l'objet de notre étude. Il n'en reste pas moins qu'une exploration algorithmique est faite dans Design Trotter au niveau de l'estimation système qui propose plusieurs choix de réalisations de l'application.

L'exploration architecturale que nous proposons donne essentiellement des informations de conception sur la répartition des communications dans l'architecture ainsi que les taux d'occupation des ressources de l'architecture (ressources de traitement en particulier). Grâce à ces informations, le concepteur peut choisir dans un ensemble d'architectures celle qui est la plus en adéquation avec l'application. Le critère, ou l'objectif étant d'avoir une architecture qui consomme un minimum de puissance ce qui se traduit par une bonne répartition hiérarchique des communications, comme nous l'avons expliqué au chapitre 3. Nous avons une approche semi-automatique, les résultats d'estimation sont obtenus automatiquement mais l'exploration est manuelle. La figure 60 décrit le flot d'exploration en quatre étapes :

Le choix des fonctions critiques : les applications sont spécifiées en langage C, dans ce langage elles sont naturellement partitionnées en fonctions. Nous ne considérons dans un premier temps que des applications séquentielles pour lesquelles les fonctions sont réalisées successivement. Il peut être long d'estimer la meilleure architecture pour chaque fonction d'une même application alors que nous avons observé que dans la plupart des cas une ou

deux fonctions sont critiques et que leur étude permet à elle seule de converger vers une architecture efficace pour l'application. C'est pourquoi il est indispensable dans un premier temps de trouver les fonctions critiques de l'application. La figure 60 illustre par exemple le cas de deux fonctions (F1 et F3) jugées plus critiques que les autres. Elles seront les seules estimées lors de l'exploration architecturale.

L'exploration architecturale : il s'agit d'utiliser l'outil d'estimation relative de Design Trotter, pour les fonctions critiques, afin de définir une architecture efficace pour chaque fonction. Le choix se fait en fonction de la répartition des communications, du taux d'utilisation des ressources et de la taille des éléments hiérarchiques.

Le choix d'une architecture commune : dans le cas ou il y a plusieurs fonctions critiques, il est possible que l'étape précédente donne des architectures différentes pour les différentes fonctions. Il faut donc utiliser divers compromis pour définir une architecture qui satisfasse toutes les fonctions.

L'estimation globale de l'application : permet de donner à le concepteur une estimation globale de la répartition des communications et du taux d'utilisation de l'architecture (donc de l'efficacité en puissance de l'architecture). Il est toujours possible au concepteur de revenir en arrière dans le flot afin de corriger un choix architectural.

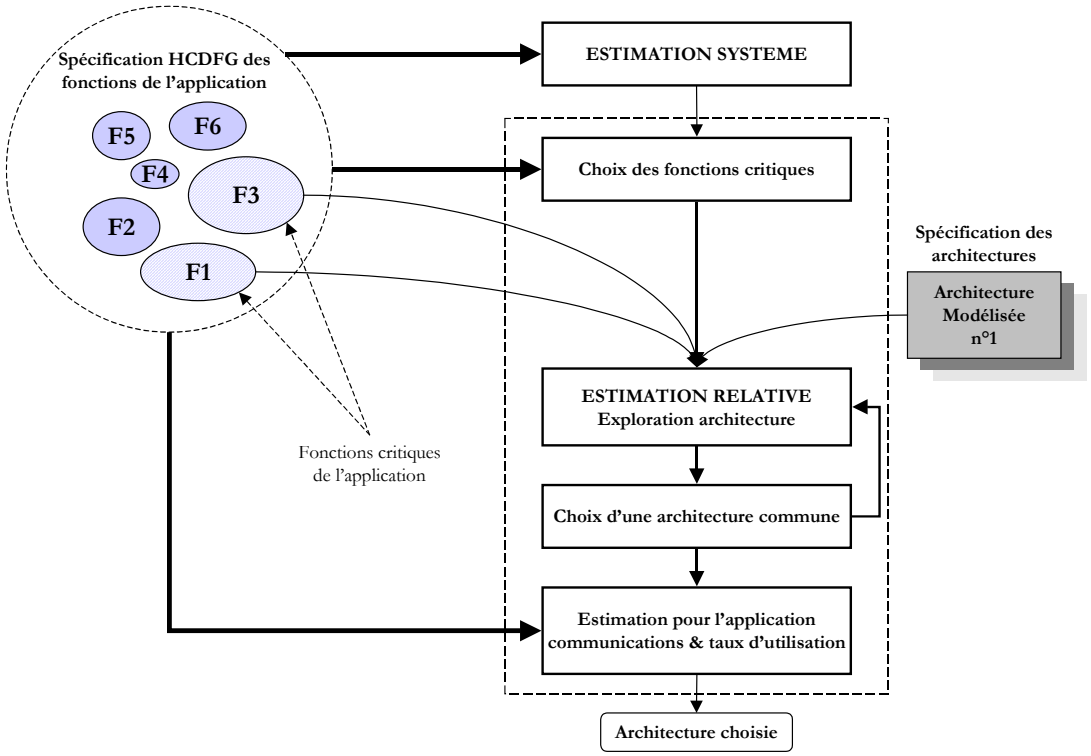


Figure 60 – Flot d'exploration architecturale.

Les prochains paragraphes vont détailler chacune des quatre étapes du flot d'exploration architecturale.

4.4.2 Définition des fonctions critiques

Comme nous l'avons dit, l'application à développer est le plus souvent réalisée par l'exécution séquentielle de plusieurs fonctions. Chaque fonction est décrite par un HCDFG. L'estimation système donne des solutions d'ordonnancement pour chacune de ces fonctions. Grâce au choix d'une solution d'ordonnancement, les fonctions sont ainsi caractérisées par le nombre de ressources de traitement et le nombre de cycles abstraits qui sont nécessaires pour les réaliser. Nous pourrions aussi utiliser la taille mémoire pour les caractériser mais l'estimation de cette taille qui est fait aujourd'hui par l'estimation système n'est pas assez précise, et fait l'objet de développements. Après la réalisation de l'ACG à partir des HCDFG il est facile de donner le nombre de communications représentées par les arcs de l'ACG en sommant leur valeur.

Les trois caractéristiques (nombre de ressources, nombre de cycles, nombre de communications) nous permettent de définir trois critères de choix :

Le degré de parallélisme d'exécution de la fonction va nous permettre de s'assurer que la solution d'ordonnancement est judicieusement choisie. Effectivement nous visons des réalisations sur des architectures reconfigurables matérielles qui proposent un parallélisme spatial important. De ce fait il faut chercher des réalisations parallèles pour les fonctions. Cependant il est possible que certaines fonctions bien que pouvant profiter d'un très large parallélisme (cas de grand déroulage de boucle) nécessitent un trop grand nombre de ressources de traitement pour être raisonnablement implémentées. Dans tous les cas il faut s'assurer que le choix d'un ordonnancement par l'utilisateur, ne peut pas être modifié pour un choix plus parallèle entraînant un faible surcoût en ressources matérielles.

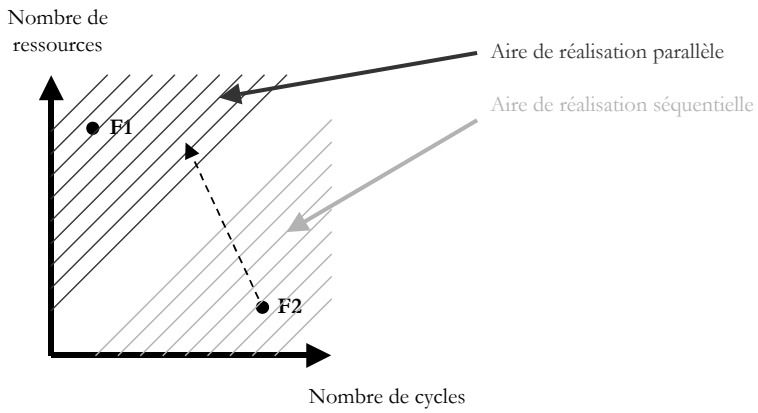


Figure 61 – Plan de réalisation avec les aires de réalisation parallèle et séquentielle.

Comme indiqué sur la figure 61, les fonctions doivent être placées dans un plan où les ordonnées représentent le nombre de ressources et les abscisses le nombre de cycles. Sur ce plan on peut séparer deux aires de réalisation, l'aire de réalisation parallèle pour les

fonctions avec un nombre de cycles faible par rapport au nombre de ressources nécessaires et inversement pour l'aire de réalisation séquentielle. Une fonction située dans cette dernière, comme F2 dans le cas de la figure 61, doit être étudiée afin de vérifier si il n'y a pas une meilleure réalisation possible. Il s'agit surtout de déterminer d'après les résultats de l'estimation système que le surcoût matériel est peu important pour une amélioration de la vitesse (ou speedup en anglais). Si c'est le cas il faut reconsidérer le choix de réalisation. Ce premier critère ne nous permet pas de sélectionner les fonctions critiques mais de simplement vérifier les choix de réalisation effectués à l'issue de l'estimation système. Les deux critères suivants vont nous permettre de trouver les fonctions critiques qui seront les seules utilisées lors de l'exploration architecturale.

La localité des communications dans l'architecture représente le nombre moyen de communications par ressources 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. Cela peut se traduire en disant qu'en moyenne les ressources de traitement vont beaucoup communiquer. 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. Il est possible de placer toutes les fonctions de l'application dans un plan à deux dimensions. Le nombre de ressources de traitement est en ordonnée de ce plan et le nombre de communications en abscisse, comme on peut le voir sur la figure 62.

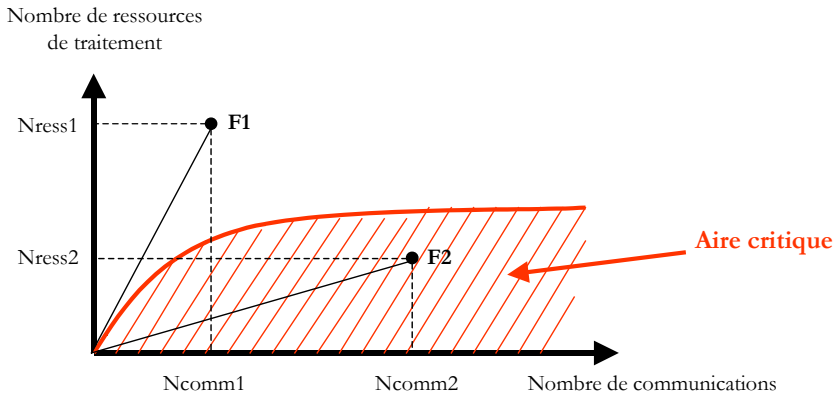


Figure 62 – Plan de criticité de la localité spatiale des communications.

La figure 62 montre qu'il existe une aire critique dans laquelle les fonctions ont un nombre de communications élevé pour un faible nombre de ressources de traitement. La position des fonctions dans ce plan nous permet de dire que suivant la localité spatiale des communications certaines fonctions sont plus critiques que d'autres. Par exemple sur la figure 62, la fonction F2 est plus critique que la fonction F1. Les fonctions sont caractérisées dans ce plan par leur nombre de ressources, par exemple *Nress1* pour F1 et

Nress2 pour F2, et par leur nombre de communications, par exemple *Ncomm1* pour F1 et *Ncomm2* pour F2. On peut décrire la règle de criticité suivante : la fonction F2 est plus critique que la fonction F1 au sens de la localité spatiale des communications si et seulement si :

$$\frac{Nress1}{Ncomm1} > \frac{Nress2}{Ncomm2}$$

Nous pourrions effectuer la même étude de criticité suivant le nombre de ressources mémoires et le nombre de communications qui leur sont associées. Cependant nous ne disposons pas d'une estimation des ressources mémoires assez performante pour être sûr de nos résultats.

La congestion temporelle 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. Dans ce cas, en moyenne, les ressources de routage vont sur des temps courts réaliser beaucoup de communications. 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. Il est possible de placer toutes les fonctions de l'application dans un plan à deux dimensions. Le nombre de cycles est en ordonnée de ce plan et le nombre de communications en abscisse, comme on peut le voir sur la figure 63.

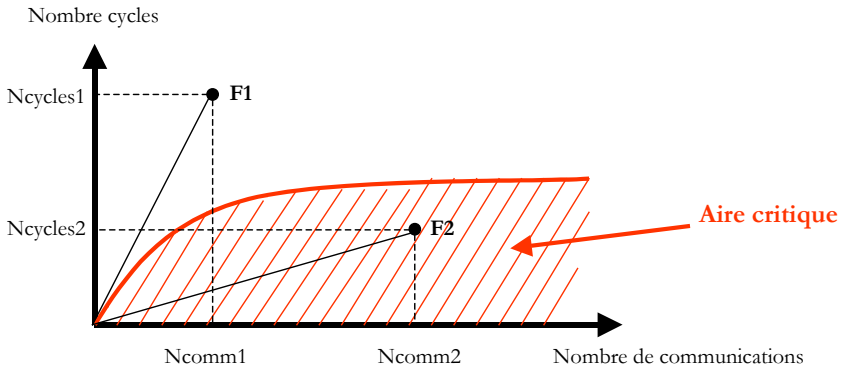


Figure 63 – Plan de criticité de la congestion temporelle du routage.

La figure 63 montre qu'il existe une aire critique dans laquelle les fonctions ont un nombre de communications élevé pour un faible nombre de cycles. La position des fonctions dans ce plan nous permet de dire que suivant la congestion temporelle du routage certaines fonctions sont plus critiques que d'autres. Par exemple sur la figure 63, la fonction F2 est plus critique que la fonction F1. Les fonctions sont caractérisées dans ce plan par leur nombre de cycles, par exemple *Ncycles1* pour F1 et *Ncycles2* pour F2, et par leur nombre de communications, par exemple *Ncomm1* pour F1 et *Ncomm2* pour F2. On peut alors écrire

cette autre règle de criticité : la fonction F2 est plus critique que la fonction F1 au sens de la congestion temporelle du routage si et seulement si

$$\frac{Ncycles1}{Ncomm1} > \frac{Ncycles2}{Ncomm2}$$

Le chapitre 5 présentera des exemples concrets et complets d'applications spécifiées avec plusieurs fonctions sur lesquelles une étude de criticité est nécessaire afin de rapidement converger vers une architecture.

4.4.3 Exploration architecturale

L'étape précédente nous a permis de sélectionner une ou plusieurs fonctions critiques. L'exploration architecturale, c'est à dire la recherche d'une architecture efficace, ne se fait qu'avec ces fonctions. Pour chaque fonction prise séparément le but est de définir les caractéristiques de l'architecture suivantes :

- le nombre de chaque type de ressources de traitement dans les éléments hiérarchiques,
- le nombre et la taille des ressources mémoires dans les éléments hiérarchiques,
- la taille des éléments hiérarchiques (ou clusters).

On peut représenter l'exploration architecturale dans un plan à trois dimensions comme sur la figure 64. L'exploration débute par le choix des clusters de plus bas niveau hiérarchique (point de départ) qui sont constitués d'éléments fonctionnels (traitements et mémoires). Ce point se situe donc dans le plan suivant les axes taille des mémoires et nombre de ressources de traitement. Il peut être compliqué de déterminer le type de cluster de base, dans ce cas une étude de l'ACG peut donner de nombreuses informations à au concepteur. Nous verrons cela lors de l'étude d'applications complètes dans le chapitre 5.

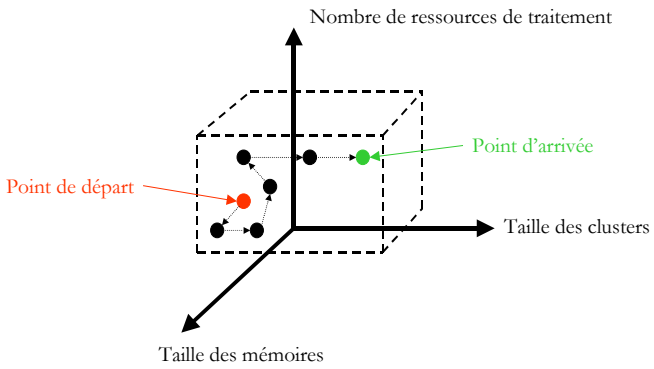


Figure 64 – Représentation de l'exploration architecturale dans un plan à trois dimensions.

Dans un cas simple les résultats de l'estimation système peuvent permettre de déterminer les ressources qui doivent apparaître dans les éléments hiérarchiques au plus bas niveau. Leur nombre reste à explorer. Prenons le cas d'une application simple comme un filtre non linéaire de Voltera. Cette application est mono-fonction, elle est spécifiée par un DFG (pas de contrôle, pas de boucle). L'estimation système propose les solutions de conception présentées au tableau 14.

Solutions	Nombre de cycles	Nombre d'additionneurs	Nombre de multiplieurs
7 – la plus parallèle	**14**	**4**	**9**
6	31	4	5
5	56	3	3
4	82	3	3
3	83	2	2
2	100	1	2
1 – la plus séquentielle	132	1	1

Tableau 14 – Résultat de l'estimation système pour le DFG filtre de Voltera.

Cette application simple ne nécessite que deux types d'opérateurs, des additionneurs et des multiplieurs. Nous choisissons la réalisation la plus parallèle (ligne en gris sur le tableau 14) qui nécessite la mise en œuvre de quatre additionneurs et de neuf multiplieurs, soit plus du double de multiplieurs que d'additionneurs. Dans ce cas simple les données manipulées sont toutes des scalaires ; nous n'avons donc pas d'estimation mémoire. Les éléments hiérarchiques de plus bas niveau seront réalisés avec des éléments fonctionnels capables de réaliser des additions et des éléments fonctionnels capables de réaliser des multiplications. Pour connaître le nombre de ces éléments fonctionnels il faut modéliser une architecture et faire varier ce paramètre. A chaque variation on estime le nombre de communications internes aux éléments hiérarchiques contenant les éléments fonctionnels. Nous obtenons les résultats de la figure 65.

La figure 65 donne le pourcentage des communications réalisées dans les éléments hiérarchiques de plus bas niveau en fonction du nombre d'éléments fonctionnels présents dans ces éléments hiérarchiques. Nous voyons sur cet exemple que le nombre d'opérateurs additionneur et multiplieur joue beaucoup sur le nombre de communications. De façon naturelle, plus il y a d'opérateurs dans l'élément hiérarchique et plus il y a de communications en interne. Si nous faisons une estimation avec des éléments hiérarchiques contenant quatre additionneurs et neuf multiplieurs alors nous obtenons que 100 % des communications sont réalisées dans un unique élément hiérarchique. Cependant nous visons une architecture hiérarchique, il ne faut donc pas choisir des clusters de taille trop importante afin d'assurer que les connexions internes soient bien locales. Le nombre de ressources internes ne doit pas être trop important. Ici un bon compromis par rapport aux autres solutions, entre le nombre de ressources internes et le nombre de communications,

est atteint pour la deuxième solution (en rayé sur la figure 65). Effectivement dans ce cas avec trois ressources de traitements par élément hiérarchique, le pourcentage de communications dans ces éléments est de 70 %.

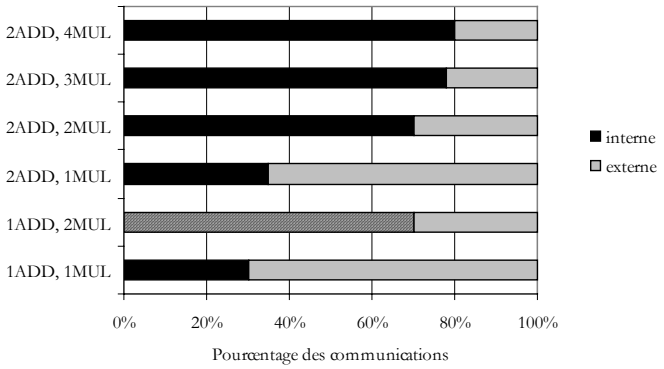


Figure 65 – Résultats de l'estimation relative pour le DFG filtre de Voltera et pour six architectures différentes.

Il en va de même pour les éléments mémoires lorsque la spécification de l'application contient des déclarations de données multidimensionnelles (typiquement des tableaux de données). Prenons un exemple d'additions de tableaux, chaque tableau contenant dix données entières codées sur 32 bits. On peut chercher la taille des mémoires qu'il faut positionner dans les éléments hiérarchiques contenant les éléments fonctionnels additionneurs. Si nous faisons un test avec des éléments hiérarchiques contenant deux puis quatre additionneurs nous obtenons les résultats de la figure 66.

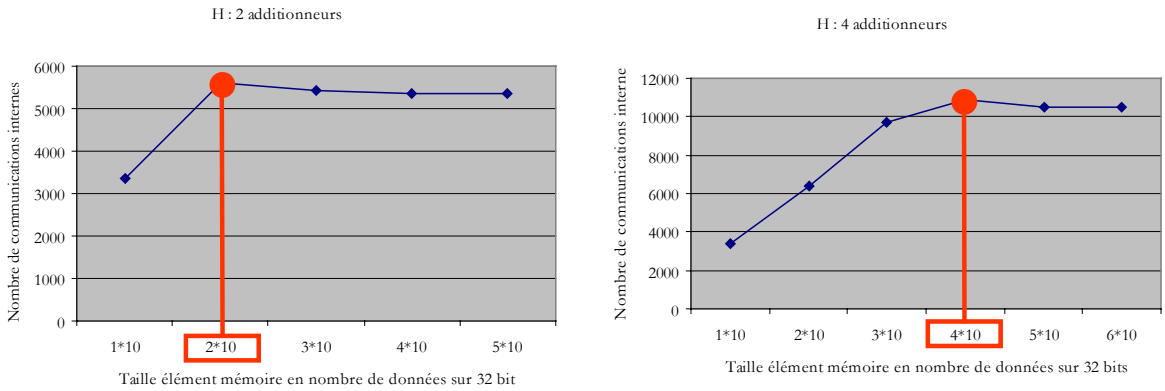


Figure 66 – Résultats de l'exploration de la taille des éléments mémoires dans un cas d'addition de tableaux.

L'exemple traité sur la figure 66 consiste à faire varier dans les deux cas (deux ou quatre additionneurs dans l'élément hiérarchique) la taille de la zone mémoire de l'élément hiérarchique. L'ordonnée des deux graphiques donne le nombre de communications internes à tous les éléments hiérarchiques, et l'abscisse la taille de l'élément mémoire en mots de 32 bits d'un élément hiérarchique. Nous observons que le nombre de communications internes est maximum lorsque la taille mémoire est de 20 mots soit deux tableaux dans le premier cas et de 40 mots dans le deuxième. Si la taille mémoire est inférieure le nombre de communications diminue car cela demande des accès vers les mémoires d'autres éléments hiérarchiques. Si la taille mémoire est supérieure le nombre de communications internes diminue aussi car il y a des regroupements locaux de données. Alors certains éléments hiérarchiques doivent systématiquement accéder à d'autres éléments hiérarchiques pour obtenir des données. De ce fait notre outil montre qu'il faut répartir soigneusement les mémoires sur l'architecture, en somme, il vaut mieux de petites mémoires bien réparties que de grandes mémoires qui ne seraient pas dans les éléments hiérarchiques. L'exemple proposé est très simple et les conclusions semblent évidentes, nous verrons dans le chapitre 5 que notre outil d'estimation relative nous permet d'arriver aux mêmes conclusions sur des exemples plus complexes.

Nous sommes donc capables d'effectuer une exploration de la taille mémoire des éléments hiérarchiques et une exploration du nombre de ressources de traitement dans ces mêmes éléments hiérarchiques. L'exploration peut débuter par celle de la taille mémoire ou celle des traitements. Pour choisir par quoi débuter, il est possible de regarder l'orientation de la fonction donnée par l'estimation système via les métriques décrites au paragraphe 2.5.2. Si la fonction est orientée mémoire il faut débuter par l'exploration de le taille mémoire sinon il faut débuter par l'exploration des traitements. Une fois que les éléments hiérarchiques de base sont déterminés, il faut explorer la taille des éléments hiérarchique de niveau supérieur. La figure 67 donne deux exemples d'architectures pour comprendre ce que l'on entend par cette exploration de la taille des éléments hiérarchiques de niveau supérieur.

Pour les deux exemples très simples de la figure 67 il y a un élément au troisième niveau de hiérarchie il s'agit du niveau le plus haut de la hiérarchie. Cet élément hiérarchique est H0, il correspond à l'architecture globale. Cet élément est de taille 2 H1. Les communications entre éléments hiérarchiques H1 sont les plus coûteuses suivant notre vision de la hiérarchie des architectures. La taille de l'élément hiérarchique H1 va varier suivant l'exemple, elle est de trois ou de deux H2. L'élément H2 ne contient que des éléments fonctionnels (traitements et mémoires). Sa structure a été déterminée durant les étapes précédentes. On peut aussi faire varier si nécessaire la taille de l'élément hiérarchique H0. Pour choisir la bonne taille de cet élément l'estimation relative va donner la répartition des communications dans les trois niveaux de hiérarchie. Il faut veiller à ne pas faire d'éléments hiérarchiques dont la taille est trop importante car il faut garder en tête que les communications doivent avoir un coût constant dans cet élément. La figure 68 donne le résultat d'exploration obtenue pour une addition de matrices. On voit qu'à partir d'une taille de H1 égale à 5 H2 l'augmentation du nombre de communications à ce niveau varie peu. Remarquons que le nombre de communications au niveau le plus bas (1er niveau de hiérarchie, donc internes à l'élément hiérarchique H2) est toujours constant car on ne modifie pas la structure interne de l'élément H2.

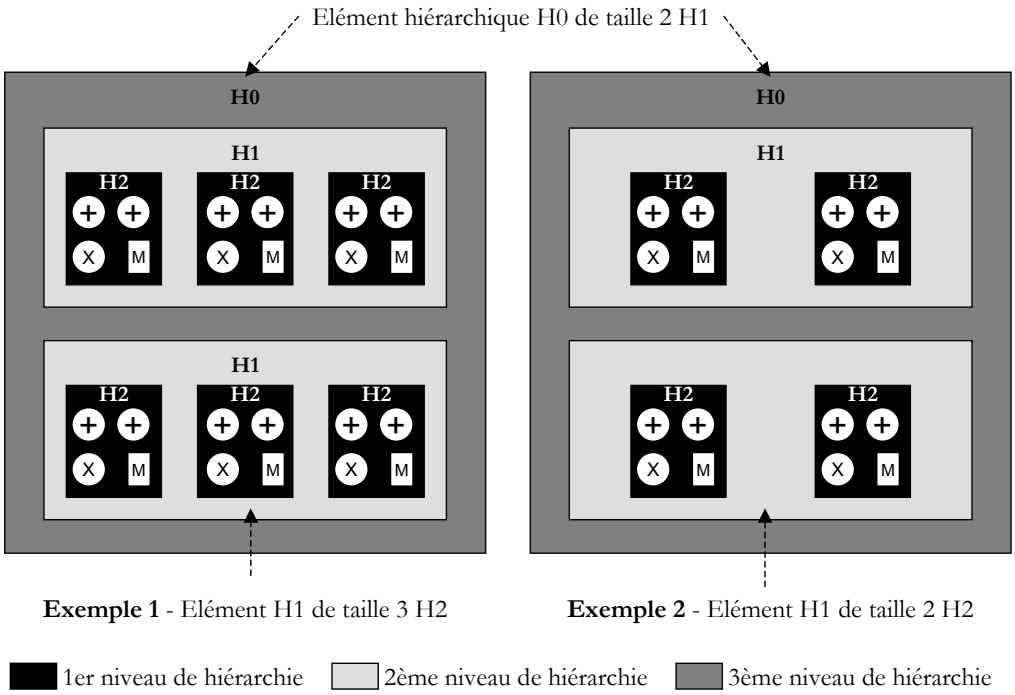


Figure 67 – Exemple de deux architectures avec un cluster au 2ème niveau de taille différente.

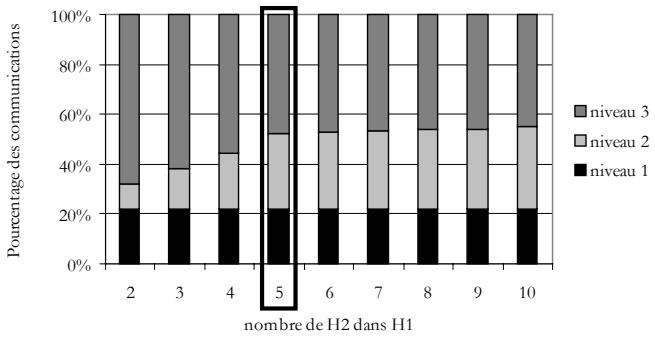


Figure 68 – Résultat de l'exploration de la taille de H1 pour un cas d'addition de matrices.

Nous avons pris ici comme mesure d'exploration uniquement la répartition des communications. Nous verrons dans les exemples plus significatifs présentés au chapitre suivant qu'un paramètre supplémentaire peut permettre de converger plus rapidement vers une solution. Il s'agit du taux d'utilisation des ressources de traitement.

4.4.4 Choix d'une architecture commune

Une fois que l'exploration architecturale a abouti à définir une architecture hiérarchique pour chaque fonction (point d'arrivée sur la figure 64), il faut déterminer une architecture commune. Effectivement l'application doit se réaliser complètement (c'est à dire toutes ses fonctions) sur une même architecture. Si les architectures sont différentes il faut converger vers une architecture unique. Cependant ce choix n'est à faire que lorsque l'exploration a été réalisée pour plusieurs fonctions critiques. Si, et cela est souvent le cas, une seule fonction est estimée, alors il n'est pas nécessaire d'effectuer cette étape.

Ce choix est compliqué et il dépend fortement des fonctions testées. Il est donc extrêmement difficile de chercher une méthode précise qui donne un résultat pour toutes les applications. Nous avons donc une approche, manuelle pour l'instant, satisfaisante car nous avons toujours obtenu des architectures proches après l'exploration architecturale. Cependant si les architectures obtenues sont très différentes en particulier en ce qui concerne la granularité des ressources de traitement on peut alors penser à partitionner l'architecture en parties de granularités différentes dédiées à chaque fonction, plutôt que de trouver un mauvais compromis. Le chapitre suivant montrera des cas de recherches de compromis pour satisfaire la réalisation de plusieurs fonctions estimées.

4.4.5 Estimation finale de l'application

Cette dernière partie du flot d'exploration architecturale pour une application consiste à prendre toutes les fonctions de l'application avec l'architecture choisie précédemment et à réaliser l'estimation relative. Les résultats sont donnés en termes de répartitions des communications dans les trois niveaux de hiérarchie de l'architecture et en terme de taux d'utilisation des ressources de traitement. Si le concepteur dispose d'un modèle de coûts il peut en déduire une estimation de la consommation de puissance de l'architecture pour réaliser l'application.

Si les résultats obtenus ne sont pas suffisamment satisfaisants, par exemple si le nombre de communications au troisième niveau de hiérarchie est trop élevé, il est possible alors de revenir en arrière vers l'exploration architecturale, afin de converger vers la définition d'une architecture efficace.

4.5 Conclusion

Ce chapitre a montré comment l'outil d'estimation relative est utilisé lors de l'exploration de l'espace de conception des architectures reconfigurables. Le flot d'exploration que nous proposons est un flot semi-automatique puisque les résultats de l'estimation relative sont obtenus automatiquement mais que l'utilisateur doit décrire les évolutions architecturales qui interviennent au cours du flot. Aussi l'utilisateur doit étudier les résultats de l'estimation système pour choisir une ou plusieurs fonctions critiques qui seront estimées. Ainsi par une étude manuelle on peut réduire considérablement le temps d'exploration totale pour l'application entière.

Ce chapitre a aussi décrit les algorithmes qui sont utilisés par l'outil d'estimation relative. Ces algorithmes opèrent principalement par transformations du graphe de communication. Les résultats donnés sont des estimations de la répartition des communications dans les différents niveaux de hiérarchie de l'architecture. Ce chapitre a montré l'intérêt de fournir ces résultats sous forme d'intervalles bornés par les valeurs minimum et maximum, associées à une valeur intermédiaire. Nous reviendrons à la fin du prochain chapitre sur ces intervalles.

La description pas à pas de l'exploration architecturale s'est faite dans ce chapitre grâce à quelques exemples très simples de fonctions. Cependant celles-ci ne nous permettent pas de saisir tout le flot d'exploration. Le chapitre suivant, et dernier chapitre de ce document, va présenter des exemples complets d'explorations architecturales pour des applications complexes.

Chapitre 5

Applications

Ce dernier chapitre, avant la conclusion de ce document, détaille les résultats obtenus lors de l'exploration de l'espace de conception des architectures reconfigurables pour trois applications du domaine du traitement des images. Il s'agit d'un flot de détection de mouvement ICAM (Intelligent Camera), de la compression d'images Matching Pursuit et du codage relatif à la norme MPEG-2. Ces applications se prêtent bien à un développement matériel sur cible reconfigurable, car elles ont un fort parallélisme potentiellement développable sur une architecture matérielle. Nous verrons aussi un exemple d'application du domaine de la cryptographie avec une partie du cryptage AES. Ainsi, nous pourrons mettre en évidence que les architectures obtenues avec l'outil dépendent bien du domaine d'application et non de l'outil.

Ces applications sont plus conséquentes que les exemples didactiques vus au chapitre précédent. C'est pourquoi ce chapitre permet aussi de mieux appréhender l'ensemble du flot d'exploration, en particulier en ce qui concerne la recherche des fonctions critiques de l'application et la recherche d'une architecture offrant un bon compromis pour ces fonctions. Enfin ce chapitre présente les résultats de l'estimation finale.

5.1 Introduction

5.1.1 Retour sur la démarche d'exploration

Nous avons choisi d'appliquer notre méthode d'exploration architecturale à trois applications issues du domaine du traitement des images, ICAM, Matching Pursuit et MPEG-2. Les applications de traitement des images sont bien adaptées à une réalisation ciblant des architectures reconfigurables qui sont des architectures matérielles proposant un fort parallélisme spatial. Il est donc possible de profiter du parallélisme potentiel de réalisation des applications ciblées et de respecter ainsi les contraintes de temps de l'application avec une flexibilité architecturale. Effectivement la même architecture reconfigurable peut être utilisée pour les trois applications ou peut être reconfigurée dynamiquement pour réaliser séquentiellement les fonctions d'une même application. Nous reviendrons dans la suite sur les modèles d'exécution et de reconfiguration que nous envisageons pour les applications et les architectures.

Nous complétons nos résultats avec l'étude d'une partie de l'application AES, qui est utilisée dans le cryptage des données. Il s'agit d'une application du domaine de la cryptographie. Celle-ci nous permet de montrer que les architectures définies grâce à l'outil sont dépendantes du domaine d'application et non de l'outil.

Dans tous les cas, comme nous souhaitons une réalisation des applications la plus parallèle possible en fonction des possibilités de l'architecture, un déroulage de boucle est systématiquement fait lors de l'estimation système afin de profiter des solutions de réalisation les plus parallèles [LeMoullec03a].

Pour chacune des applications présentées dans ce chapitre nous allons reprendre la démarche systématique vue dans le chapitre précédent. Celle-ci nous permet de partir de la spécification de l'application pour arriver au choix d'une architecture finale avec une estimation de la répartition des communications pour l'application complète. Cette démarche d'exploration comprend différentes étapes qui sont listées ci-dessous dans l'ordre de réalisation et que nous allons retrouver dans l'exemple de l'application ICAM :

- choix d'une solution parmi celles proposées par l'estimation système,
- choix des fonctions critiques qui seront les seules utilisées lors de l'exploration architecturale,
- étude de l'ACG de chacune des fonctions critiques pour en tirer les types d'éléments fonctionnels à placer dans les éléments hiérarchiques de l'architecture,
- exploration de la taille des éléments mémoires qui sont embarqués dans les éléments hiérarchiques de l'architecture,
- exploration du nombre d'opérateurs qui sont embarqués dans les éléments hiérarchiques de l'architecture,

- exploration de la taille des éléments hiérarchiques de l'architecture et étude du taux d'utilisation des opérateurs,
- choix d'une architecture commune lors de l'exploration avec plusieurs fonctions critiques,
- estimations finales pour toutes les fonctions de l'application.

Ces exemples d'applications permettent de mettre en évidence l'apport des travaux présentés dans ce document. Tout d'abord, l'approche proposée permet à partir d'une spécification en langage C d'une application, d'obtenir un modèle d'architecture qui soit efficace en ce qui concerne la répartition des communications donc en ce qui concerne la consommation de puissance pour l'application comme nous l'avons montré dans le chapitre 3. Ceci est effectué en donnant un certain nombre d'informations de conception intéressantes pour le concepteur. Nous montrons à travers ces exemples que le flot utilisé est simple à mettre en œuvre et nous permet de converger rapidement vers une bonne solution architecturale. Au final nous apportons une aide nouvelle au concepteur lui permettant une réelle exploration à partir d'une spécification en langage C de l'application. Les prochains paragraphes vont donc présenter chacune des applications et les résultats obtenus lors de l'exploration architecturale complète des applications. Dans tous les cas les résultats donnés sont ceux obtenus avec l'algorithme intermédiaire de la projection matérielle. Nous reviendrons lors de la conclusion sur les valeurs données par les trois algorithmes (minimum, intermédiaire et maximum).

5.1.2 Modèle d'exécution des applications et modèle de reconfiguration des architectures

Il est important avant d'étudier les résultats fournis lors de l'exploration architecturale sur des exemples d'applications de bien rappeler le modèle d'exécution pris en compte dans ces travaux. Tout d'abord, rappelons-nous que nous considérons des applications flot de données pouvant être partitionnées en fonctions. Chaque fonction est décrite par un graphe HCDFG à partir d'un code source initial en langage C. Chaque fonction peut donc contenir des structures de contrôle. En effet, les différents DFG composant une fonction peuvent être positionnés dans des structures de type boucles et conditions. Ces structures sont prises en compte lors de la transformation du HCDFG en ACG comme nous l'avons vu dans le chapitre 3. Par contre le déroulement de l'application considérée est de type flot de données et mono-cadence. C'est à dire que nous considérons actuellement uniquement des applications dont les fonctions ont un nombre identique d'itérations. Si cette condition n'est pas respectée, il est alors nécessaire de prendre en compte le nombre d'itérations dans la mesure de criticité des fonctions ainsi que dans les résultats d'estimation. Cette extension n'est pas actuellement intégrée dans nos travaux.

Les applications du type flot de données peuvent avoir deux modèles d'exécution, l'exécution séquentielle ou l'exécution pipelinée (parallélisme temporel). Ces deux modèles d'exécution peuvent être considérés dans notre méthode, mais alors le modèle de reconfiguration de l'architecture au cours de l'exécution de l'application, n'est pas le même. Les figures 67 et 68 présentent une vision schématique des modèles d'exécution séquentielle et pipelinée pour une même application partitionnée en quatre fonctions.

Dans le cas de l'exécution séquentielle, figure 67, l'architecture n'exécute qu'une fonction à la fois. Il est donc nécessaire de la reconfigurer dynamiquement à chaque exécution d'une nouvelle fonction. Les performances temporelles de l'application peuvent être dans ce cas réduites puisque l'architecture ne réalise qu'une fonction à la fois, de plus les temps de reconfiguration diminuent les performances. Mais la surface de l'architecture peut être minimum, elle sera déterminée par les besoins de la fonction dont la configuration est la plus importante en nombre de ressources utilisées.

Dans le cas de l'exécution pipelinée, figure 68, l'architecture est configurée une seule fois initialement. La configuration n'est pas remise en cause lors du déroulement de l'application puisque toutes les fonctions sont réalisées simultanément dès le régime permanent atteint. Le modèle de reconfiguration de l'architecture est donc statique. Dans ce cas le parallélisme d'exécution de l'application permet souvent des performances temporelles supérieures par rapport à l'exécution séquentielle. Cependant l'architecture doit comporter un nombre de ressources plus important que dans le cas précédent (somme des ressources fonctionnelles demandées par les différentes fonctions).

La démarche à suivre pour définir l'architecture dans les deux cas est la même. Toutefois, le choix des solutions de réalisation proposées par l'estimation système peut différer. Effectivement dans le cas d'une exécution séquentielle de l'application il est souvent plus efficace de choisir une réalisation très parallèle des fonctions. Effectivement dans ce cas la totalité des ressources peut être utilisée pour chaque fonction alors que dans le cas d'exécution pipelinée, les fonctions partagent les ressources de l'architecture. L'estimation de la répartition des communications dans l'architecture résulte de la même démarche dans les deux cas.

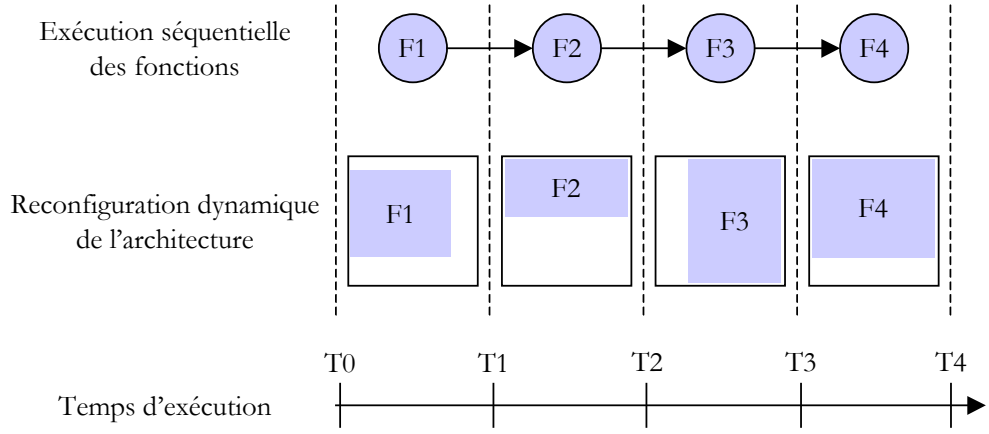


Figure 67 – Schéma du modèle d'exécution séquentielle de l'application et du modèle de reconfiguration dynamique de l'architecture.

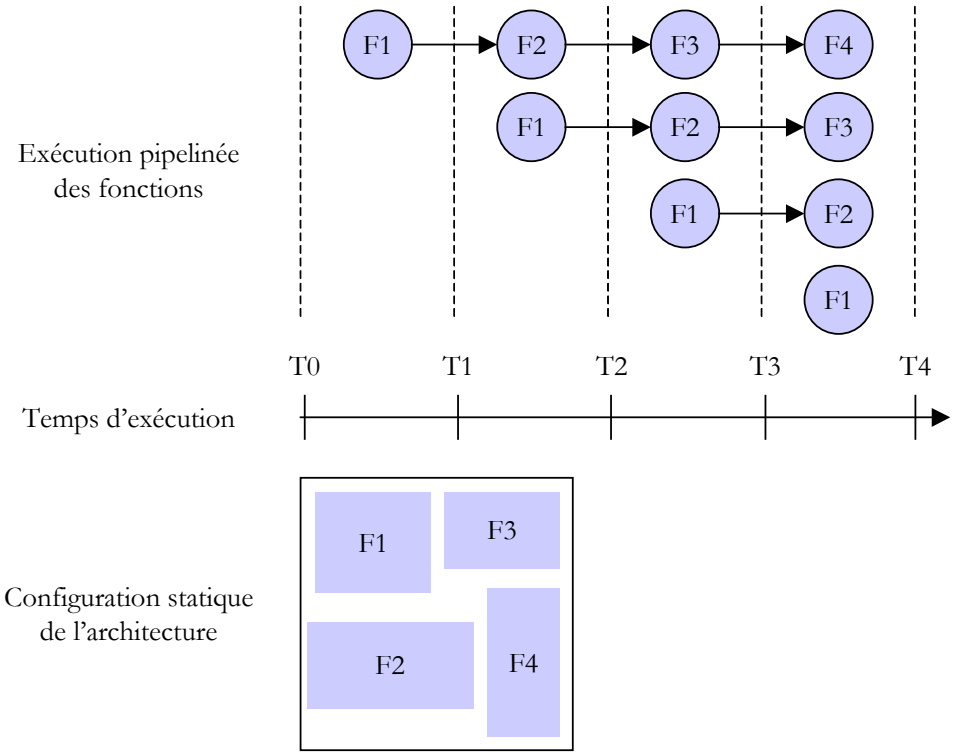


Figure 68 – Schéma du modèle d'exécution pipelinée de l'application et du modèle de reconfiguration statique de l'architecture.

Comme nous l'avons vu les modèles de reconfiguration de l'architecture pris en compte sont les modèles dynamique et statique. Dans les deux cas nous sommes en face d'architectures reconfigurables, c'est à dire que ces architectures ne sont pas dédiées à une seule application. Elles peuvent être reconfigurées pour exécuter plusieurs applications. Notre démarche d'exploration permet de définir dans un premier temps une architecture en adéquation avec une application. Nous allons montrer qu'une architecture peut être définie pour un domaine d'application. Par exemple les trois applications de traitement d'images présentées dans la suite aboutissent à un même modèle d'architecture, une reconfiguration de l'architecture permet le changement d'application.

Pour les exemples présentés dans les paragraphes suivants, le modèle d'exécution choisi est systématiquement un modèle pipelinée puisque nous sommes dans le cas d'applications sous contraintes de temps fortes : l'architecture définie à partir de la ou les fonctions critiques supporte dans tous les cas simultanément toutes les fonctions de l'application.

5.2 Application ICAM

5.2.1 Présentation

Cette première application est la plus importante en terme de taille de spécification, par rapport aux autres applications qui sont étudiées dans ce chapitre. Cette application de détection de mouvement pour caméra intelligente nous a été fournie par le laboratoire LIST* du CEA dans le cadre du projet RNTL EPICURE [Auguin03].

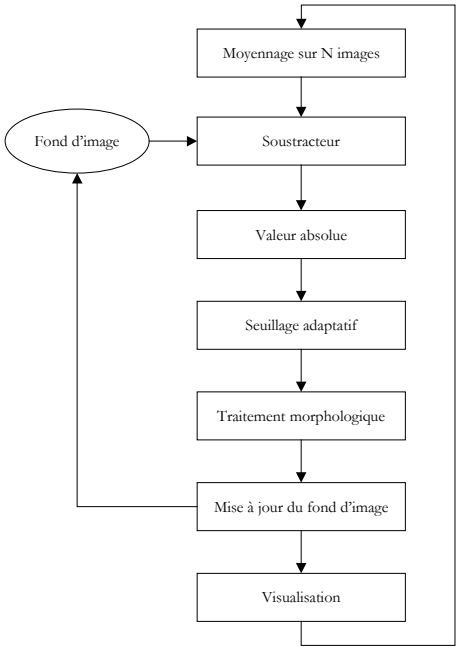


Figure 69 – Flot de traitements des images vidéo pour l'application ICAM.

Le but principal de l'application est de réaliser la détection et le suivi d'objets en mouvements sur une image de référence, par exemple des passagers sur les quais de métro, afin d'étudier les flux de voyageurs aux différentes heures de la journée. Pour ce faire des traitements sont appliqués aux images vidéo issues d'un capteur CMOS. La figure 69 présente le flot de traitements subi par les images vidéo.

La première étape de moyenne paramétrable permet de limiter la sensibilité de la chaîne au bruit. Le nombre d'images pris en compte dans le calcul de l'image moyenne dépend du niveau de bruit et de la vitesse de déplacement des objets dans les images. L'étape de soustraction de l'image de référence à l'image traitée permet de détecter les zones en

* Laboratoire d'Intégration des Systèmes et des Technologies, http://www-drt.cea.fr/fr/prog/list.htm

mouvements. Pour isoler les objets mobiles dans les zones mobiles, un seuillage est effectué. La valeur du seuil est paramétrable, elle est issue de l'application d'un masque de gradient sur l'histogramme de l'image traitée. Pour faire disparaître les points isolés de l'image un filtrage par traitement morphologique est réalisé.

5.2.2 Choix des solutions parmi celles proposées par l'estimation système

La première étape de la démarche d'exploration est le choix des solutions proposées par l'estimation système pour chacune des fonctions de l'application prises séparément. L'estimation système donne pour chacune des fonctions des courbes de compromis représentant différentes solutions d'implémentations [LeMoullec03a]. Chaque solution d'implémentation est caractérisée par un nombre de cycles et un nombre de ressources matérielles (opérateurs et mémoires). Il est du ressort du concepteur de choisir une solution pour chaque fonction en utilisant par exemple un critère d'efficacité (accélération/nombre de ressources). Cette première approche étudie les fonctions séparément les unes des autres, cependant elles font partie d'une application unique aussi il faut veiller à leur concordance.

Après le choix d'une solution pour chaque fonction prise séparément, il faut s'assurer que les solutions choisies sont effectivement en adéquation avec une réalisation sur cible matérielle reconfigurable à fort parallélisme spatial. Pour cela nous étudions le placement de toutes les fonctions de l'application dans un plan caractérisé par le nombre de ressources de traitement de l'application et le nombre de cycles. La figure 70 donne le placement des quinze fonctions de l'application ICAM. En fait cette application est composée de seize fonctions mais la fonction *dilatBin* s'exécute avec un nombre de cycles bien supérieur à ceux des quinze autres fonctions, alors pour une question d'échelle, il est difficile de la positionner dans le même plan que les autres. Cependant nous avons vérifié que sa réalisation est bien parallèle.

Nous avons indiqué la zone de réalisation séquentielle sur la figure 70, chaque point du plan est obtenu par choix du concepteur d'une solution proposée par l'estimation système. Nous constatons que trois fonctions, *envelop*, *reconsDilat* et *errorBin,* ont pour le choix effectué une réalisation plus séquentielle que les autres fonctions. En ce qui concerne les deux fonctions *reconsDilat* et *errorBin* la solution de réalisation choisie parmi celles proposées par l'estimation système est dans les deux cas la solution la plus parallèle. Il n'est donc pas possible de réaliser ces fonctions de façon plus parallèle.

Cependant il est possible de revenir alors sur la spécification de l'application afin d'intégrer, par exemple, ces fonctions à d'autres pour voir émerger un parallélisme potentiel plus important. Par contre en ce qui concerne la fonction *envelop* la solution de réalisation choisie parmi celles proposées par l'estimation système n'est pas la plus parallèle. Nous pouvons voir sur le tableau 15 une partie des solutions proposées par l'estimation système.

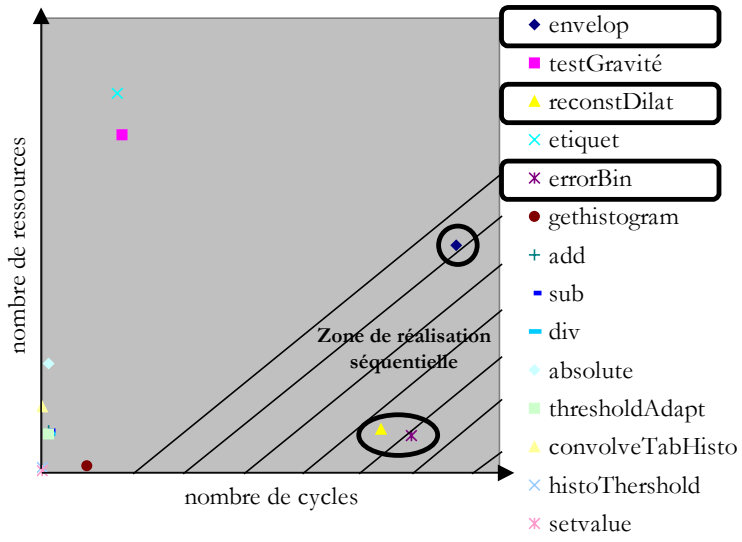


Figure 70 – Plan de réalisation des fonctions de l'application ICAM.

Nombre de cycles	Nombre d'opérateurs à mettre en œuvre					
	Additionneur	Soustracteur	Multiplieur	Diviseur	Comparateur	Logique
239 815	257	257	129	1	644	256
343 816	129	129	65	1	324	128
1 360 648	33	33	17	1	74	32
2 716 424	17	17	9	1	44	16

Tableau 15 – Une partie des résultats de l'estimation système pour la fonction envelop de l'application ICAM.

La ligne grisée du tableau 15 correspond à la solution sélectionnée qui n'est effectivement pas la solution la plus parallèle puisqu'il existe d'autres solutions plus rapides qui mettent en jeu plus d'opérateurs. La question à se poser est de savoir s'il est possible de choisir une réalisation plus parallèle. Il doit y avoir un compromis (critère d'efficacité) entre le nombre de cycles devant être le plus faible afin d'obtenir une exécution rapide, et un nombre de ressources de traitement qui ne soit pas trop important car les ressources matérielles sont limitées et partagées entre les fonctions. Dans l'exemple du tableau 15, en sélectionnant la ligne juste au-dessus de la ligne grisée, la solution choisie a un nombre de cycles presque quatre fois inférieur pour un nombre de ressources de gros grain (additionneur, soustracteur, multiplieur et diviseur) quatre fois supérieur. Le choix initial peut être remis en cause si la contrainte portait sur le temps d'exécution de l'application. Cependant si toutes les fonctions sont réalisées sur le même circuit sans reconfiguration dynamique alors c'est la contrainte en ressources matérielles qui prédomine. Cependant, dans ce cas le

nombre d'opérateurs nécessaires est trop élevé compte tenu que les ressources matérielles de l'architecture finale seront partagées entre toutes les fonctions de l'application. Nous voyons ainsi qu'il n'est pas toujours possible de choisir les réalisations les plus parallèles.

Une fois que nous avons vérifié que les solutions de réalisation des fonctions sont bien choisies, il faut trouver les fonctions critiques qui seront seules estimées lors de l'exploration architecturale.

5.2.3 Choix des fonctions critiques

Comme nous l'avons indiqué dans le chapitre précédent, cette étape de choix des fonctions critiques permet d'indiquer les seules fonctions sur lesquelles il est nécessaire de réaliser l'exploration architecturale. Pour cela nous devons caractériser toutes les fonctions de l'application dans deux plans, le plan de criticité de la localité spatiale des communications et le plan de criticité de la congestion temporelle du routage. Les figures 71 et 72 représentent le positionnement des fonctions de l'application ICAM dans ces deux plans. Nous voyons que la fonction *envelop* est la plus critique suivant la localité spatiale des communications (figure 71) et que la fonction *testGravité* est la plus critique suivant la congestion temporelle des ressources de routage (figure 72).

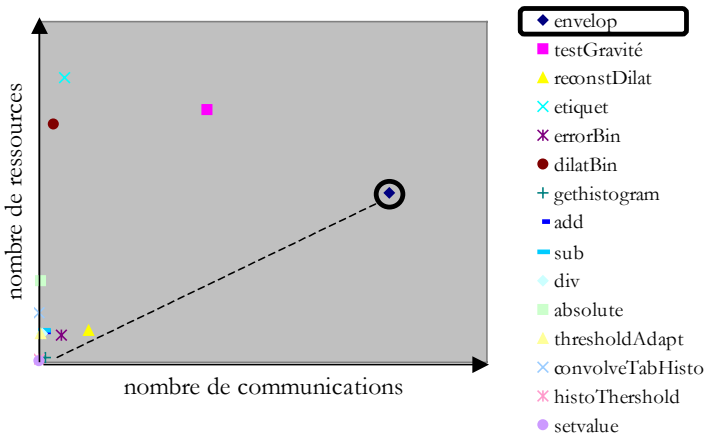


Figure 71 – Plan de criticité de la localité spatiale des communications pour les fonctions de l'application ICAM.

Ces deux fonctions sont les seules à être utilisées dans la suite car toutes les autres fonctions ne sont pas critiques. Cela nous permet un important gain de temps et nous permet de trouver plus facilement un bon compromis d'architecture pour l'ensemble de l'application. D'après nos expérimentations, le temps nécessaire à la définition d'une architecture pour l'application ICAM en utilisant les deux fonctions critiques est inférieur à une journée de travail. Il serait supérieur à une semaine si nous cherchions une architecture commune à celles trouvées pour l'exploration de toutes les fonctions. Nous montrerons pour l'architecture ICAM que si nous choisissons comme fonction directrice une qui ne

serait pas critique alors les résultats obtenus sont largement dégradés par rapport à ceux obtenus avec les fonctions critiques.

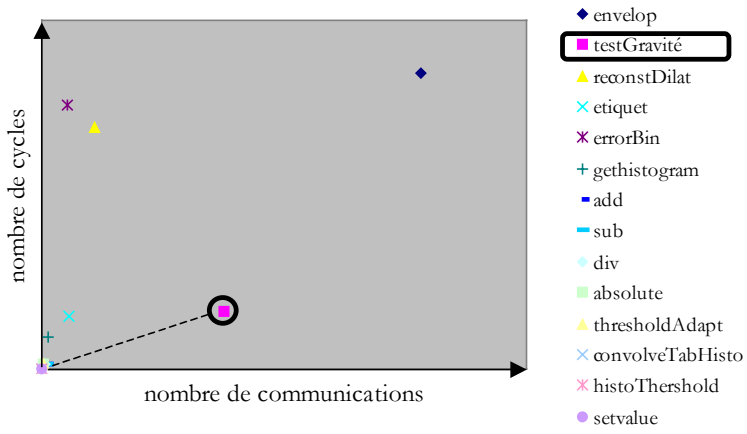


Figure 72 – Plan de criticité de la congestion temporelle des ressources de routage pour les fonctions de l'application ICAM.

Nous allons donc pouvoir effectuer l'exploration architecturale pour ces deux fonctions. Le tableau 16 donne les caractéristiques des deux fonctions étudiées, en particulier le nombre de ressources de traitement et de mémorisation qui leur sont nécessaires (estimation système). Dans le tableau, et pour tous les résultats donnés par la suite concernant cette application, le flot de données est sur 16 bits (entrées des opérateurs et largeur des mots mémorisés). La capacité des mémoires est donnée en nombre de mots.

Fonction	Nombre de communications	Nombre de cycles	Ressources de traitement				Ressources mémoires
			add/sub	mul/div	comp	logique	
envelop	15 643 324	1 360 648	66	18	84	32	7*20 + 3*35536
testGravité	7 523 319	264 758	60	1	176	59	15*20

Tableau 16 – Caractéristiques des deux fonctions critiques de l'application ICAM.

5.2.4 Exploration architecturale

L'étude de l'ACG est la première étape de l'exploration architecturale. Elle a pour but de définir les éléments hiérarchiques qui sont au plus bas niveau de hiérarchie. Ces éléments hiérarchiques sont uniquement composés d'éléments fonctionnels. L'étude du graphe des communications ne se fait que pour les deux fonctions critiques. Il s'agit de relever le

nombre de communications entre les différents types de traitements et de mémoires afin de guider le choix des éléments hiérarchiques. Par exemple, il ne sera pas forcément judicieux de mettre dans un même élément hiérarchique des éléments fonctionnels qui supportent des traitements qui communiquent peu ou pas du tout.

L'étude de l'ACG de la fonction *testGravité* nous donne la répartition suivante des communications :

- 50,3 % des communications de mémoires à mémoires,
- 32,9 % des communications entre comparateurs et mémoires,
- 16,4 % des communications entre comparateurs et logiques,
- 0,2 % des communications entre additionneurs et mémoires,
- 0,1 % des communications entre additionneurs et multiplieurs,
- 0,1 % des communications entre multiplieurs et mémoires.

D'après l'étude de ces communications on peut voir qu'il est possible de séparer les fonctions logiques et les comparateurs qui sont des opérateurs de grain fin, des opérateurs de gros grain que sont les additionneurs et les multiplieurs. Nous pouvons donc modéliser une architecture qui a, au niveau le plus bas de hiérarchie, deux types d'éléments hiérarchiques : 1- des éléments hiérarchiques de grain fin composés de fonctions logiques (LUT), de comparateurs et de mémoires, 2- des éléments hiérarchiques de gros grain composés d'additionneurs, de multiplieurs et de mémoires.

De la même façon, l'étude de l'ACG de la fonction *envelop* nous donne la répartition suivante des communications :

- 41,0 % des communications entre comparateurs et petites mémoires,
- 23,4 % des communications entre additionneurs et petites mémoires,
- 23,4 % des communications entre comparateurs et logiques,
- 5,8 % des communications entre additionneurs et grandes mémoires,
- 5,8 % des communications entre additionneurs et multiplieurs,
- 0,5 % des communications entre multiplieurs et grandes et petites mémoires, et entre additionneur et petite mémoires.

Pour ces communications, la différence entre les petites et les grandes mémoires, est le nombre de mots mémorisés, 20 pour les petites mémoires et 65536 pour les grandes mémoires ce qui représente une image. Du fait de la taille importante des grandes mémoires elles sont positionnées dans un élément hiérarchique à part qui ne contient qu'un élément mémoire. Les deux autres éléments hiérarchiques que l'on modélise au niveau le plus bas de hiérarchie sont identiques à la fonction précédente. Ce sont des éléments

composés de LUT, de comparateur et de petites mémoires et des éléments composés d'additionneurs, de multiplieurs et de petites mémoires.

Nous allons donc modéliser deux architectures pour les deux fonctions. Respectivement nous modéliserons les architectures $A_{testGravité}$ et $A_{envelop}$ pour les fonctions *testGravité* et *envelop*. Dans un premier temps nous décrivons ces architectures de la façon suivante :

$$A_{testGravité} = \{cluster1\{ADD, MUL, MEM1\}\ cluster2\{COMP, LOGIC, MEM1\}\}$$

$$A_{envelop} = \{cluster1\{ADD, MUL, MEM1\}\ cluster2\{COMP, LOGIC, MEM1\}\ cluster3\{MEM2\}\}$$

Ce qui veut dire que l'architecture $A_{testGravité}$ est composée d'un ensemble d'éléments hiérarchiques ou clusters de deux types : des éléments hiérarchiques du type *cluster1* et des éléments hiérarchiques du type *cluster2*. Les éléments hiérarchiques du type *cluster1* sont composés d'éléments fonctionnels du type *ADD* (additionneur-soustracteur), *MUL* (multiplieur-diviseur) et *MEM1* (mémoire de 20 mots de 16 bits). Les éléments hiérarchiques du type *cluster2* sont composés d'éléments fonctionnels du type *COMP* (comparateur), *LOGIC* (table de scrutation) et *MEM1*. De même pour l'architecture $A_{envelop}$ avec un type d'éléments hiérarchiques supplémentaire *cluster3* composé d'éléments fonctionnels du type *MEM2* (mémoire de 65536 mots de 16 bits).

L'exploration de la taille des mémoires et du nombre d'opérateurs dans les éléments hiérarchiques est la deuxième étape de l'exploration architecturale. Elle consiste à déterminer le nombre de chacun des éléments fonctionnels (mémoires et traitements) dans les différents éléments hiérarchiques de bas niveau. Il est possible de débuter par l'exploration de la taille des mémoires ou par le nombre d'opérateurs. Cependant nous avons remarqué lors de nos essais que l'ordre d'exploration entre mémoires et opérateurs n'influe pas sur le résultat architectural final.

Pour explorer la taille des mémoires nous modélisons plusieurs architectures basées sur le schéma des architectures $A_{testGravité}$ et $A_{envelop}$. Chaque architecture reçoit un nombre de *MEM1* différents dans ses éléments hiérarchiques *clusters1* et *cluster2*. Pour chaque architecture une estimation du nombre de communications à l'intérieur des éléments hiérarchiques est donnée afin de voir l'impact de la taille des éléments mémoires. Ce nombre de communication est directement issu des résultats de la première projection. La figure 73 donne les résultats obtenus pour la fonction *testGravité*. Nous obtenons des résultats similaires pour la fonction *envelop*.

Nous voyons sur la figure 73 qu'il faut retenir un élément mémoire pour le *cluster1*, car une augmentation du nombre d'éléments mémoire n'augmente pas la quantité de communications internes aux éléments hiérarchiques *cluster1*. Ceci est dû au fait que les additionneurs et les multiplieurs communiquent peu avec ces mémoires. Effectivement les tableaux de 20 données issus de la spécification sont principalement utilisés pour le test des indices de boucle et peu utilisés comme résultats de calculs. Le *cluster2* pourra lui bénéficier de deux éléments mémoires car, comme la figure 73 le montre, l'augmentation du nombre

d'éléments mémoires de un à deux augmente le nombre de communications internes aux éléments hiérarchiques *cluster2*. Effectivement dans ce cas les communications entre les comparateurs et les mémoires sont en nombre important. Cependant il n'est pas nécessaire d'avoir plus de deux éléments mémoires puisque cela ne fait pas directement augmenter le nombre de communications internes.

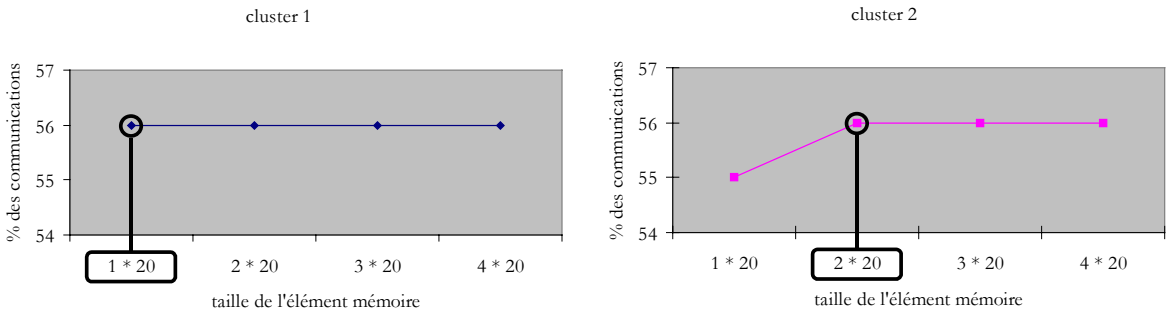


Figure 73 – Exploration de la taille mémoire dans les éléments hiérarchiques cluster1 et cluster2 de l'architecture $A_{testGravité}$ pour la fonction testGravité de l'application ICAM.

L'exploration du nombre d'éléments fonctionnels dans les éléments hiérarchiques *cluster1* et *cluster2* est la troisième étape de l'exploration architecturale. Pour cela il faut modéliser plusieurs architectures sur la base des architectures $A_{testGravité}$ et $A_{envelop}$ pour les deux fonctions. Chaque architecture modélisée a une combinaison différente du nombre de chaque type d'éléments fonctionnels. Pour ne pas explorer trop de cas il faut se rapporter aux besoins en ressources de traitement des fonctions qui sont donnés au tableau 16. Par exemple pour la fonction *testGravité* il est nécessaire d'avoir un nombre important d'additionneurs et de comparateurs. De ce fait nous allons principalement faire varier le nombre de ces éléments. La figure 74 donne une partie de l'exploration pour cette fonction.

Sur cette figure nous voyons que le pourcentage de communications réalisées à l'intérieur des éléments hiérarchiques augmente principalement avec le nombre de comparateurs dans le *cluster2*. L'étude de l'ACG nous avait indiqué que 49,3 % des communications impliquent (en émission ou en réception) un comparateur. Donc plus il y a de comparateurs dans les *cluster2* et plus il est possible de localiser de nombreuses communications dans ces clusters. Par contre du fait de leur plus faible implication dans les communications totales de l'ACG, le nombre d'additionneurs influe très peu. Cependant nous ne pouvons pas choisir des éléments hiérarchiques *cluster2* contenant trop d'éléments fonctionnels car alors il faut être sûr de pouvoir physiquement (c'est à dire avec des ressources de routage) assurer un coût de communication constant dans le *cluster2*. Il s'agit de faire un compromis. Nous avons choisi la sixième solution qui est de prendre un additionneur et six comparateurs respectivement dans les éléments hiérarchiques *cluster1* et *cluster2*. Le choix de ce compromis s'appuie sur une expertise du concepteur et sur la connaissance qu'il détient

concernant la réalisation et les performances des ressources de routages du type d'architectures visées.

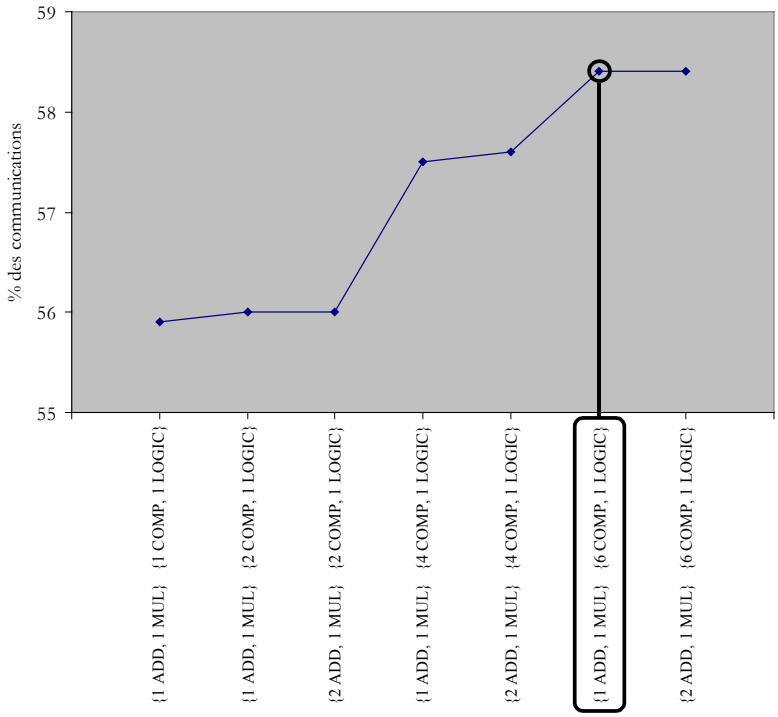


Figure 74 – Exploration du nombre d'additionneurs et de comparateurs dans les cluster1 et cluster2 de l'architecture $A_{testGravité}$ pour la fonction testGravité de l'application ICAM.

L'exploration de la taille des éléments hiérarchiques de niveau 2 et de niveau 3 est à proprement dit la dernière étape d'exploration architecturale. Nous avons comme contrainte de départ de réaliser des architectures à trois niveaux de hiérarchie. Nous venons de voir de quelle façon déterminer les éléments hiérarchiques de plus bas niveau (niveau 1) en explorant la taille de leurs éléments mémoires et le nombre des différents éléments fonctionnels. Il faut maintenant déterminer la taille des éléments hiérarchiques aux deux niveaux supérieurs (niveau 2 et niveau 3).

Nous débutons par l'exploration de la taille des éléments hiérarchiques de niveau 2. Ceux-ci sont composés des éléments hiérarchiques *cluster1* et *cluster2*. Il faut donc déterminer le nombre de *cluster1* et de *cluster2* par élément hiérarchique du niveau 2 que nous appellerons $H_{niveau2}$.

Les figures 75 et 76 nous donnent le résultat des explorations de taille que nous avons effectuées pour la fonction *testGravité*. Ces explorations nous montrent que le nombre de *cluster1* dans $H_{niveau2}$ intervient peu sur le nombre de communications dans l'élément $H_{niveau2}$ (figure 75). Ce résultat se traduit de la façon suivante : les communications sont bien

réparties dans les *cluster1* qui échangent peu de données entre eux. Ce qui n'est pas le cas des *cluster2*, plus leur nombre augmente et plus le nombre de communications dans les éléments hiérarchiques $H_{niveau2}$ augmente de façon linéaire (figure 76). La détermination de la taille des éléments hiérarchiques $H_{niveau2}$ doit prendre en compte qu'il faut assurer un coût de connexion constant à l'intérieur de ces éléments lors de la réalisation physique.

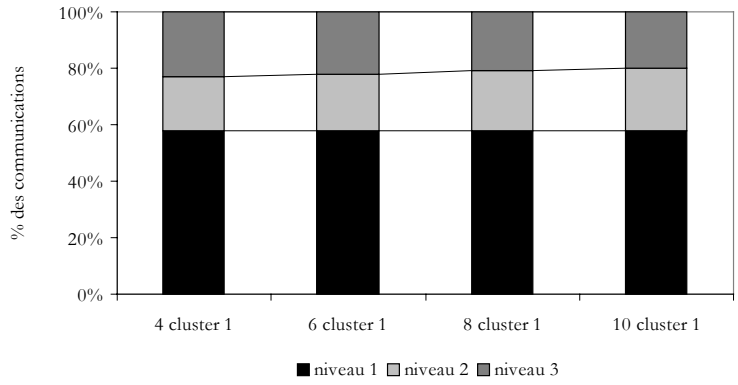


Figure 75 – Exploration du nombre d'éléments hiérarchiques cluster1 dans un élément hiérarchique de niveau supérieur $H_{niveau2}$ de l'architecture $A_{testGravité}$ pour la fonction testGravité de l'application ICAM.

Pour bien choisir la taille des éléments hiérarchiques $H_{niveau2}$, en plus de l'exploration que nous venons de voir, il est intéressant de calculer le taux d'utilisation des éléments fonctionnels de traitement. Pour la fonction *testGravité* le tableau 16 nous indique qu'il est nécessaire d'utiliser 60 additionneurs, 1 multiplieur, 176 comparateurs et 59 LUT. L'exploration du nombre d'éléments fonctionnels dans les éléments hiérarchiques nous a donné comme résultat que le *cluster1* devait contenir 1 additionneur et 1 multiplieur, et que le *cluster2* devait contenir 6 comparateurs et 1 LUT. Nous pouvons en conclure que la fonction *testGravité* nécessite l'utilisation de 60 *cluster1* pour réaliser les additionneurs, de 30 *cluster2* pour réaliser les comparateurs et de 59 *cluster2* pour réaliser les fonctions logiques. Nous avons vu au chapitre 3 que pour être efficace en puissance il faut utiliser le mieux possible les ressources de traitement de telle façon qu'il n'y ait pas un trop grand nombre de ressources non utilisées. Nous devons donc chercher, en respectant les explorations faites précédemment, à converger vers un maximum du taux d'utilisation pour chaque ressources de traitement en particulier celles de gros grain qui sont plus consommatrices de puissance.

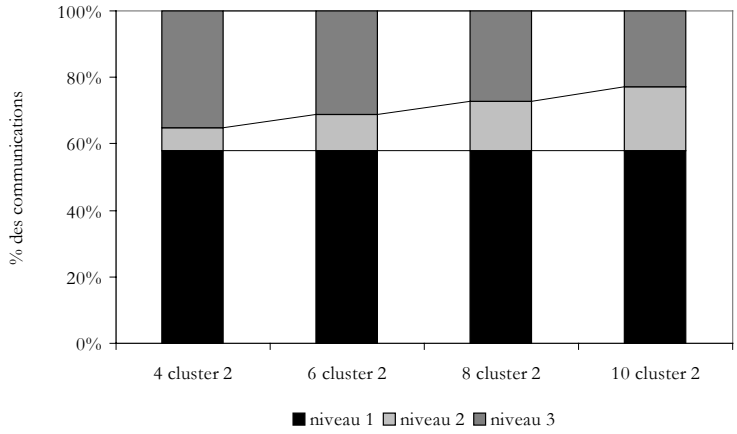


Figure 76 – Exploration du nombre d'éléments hiérarchiques cluster2 dans un élément hiérarchique de niveau supérieur $H_{niveau2}$ de l'architecture $A_{testGravité}$ pour la fonction testGravité de l'application ICAM.

L'ajustement de l'architecture pour obtenir un bon taux d'utilisation nous permet d'améliorer nos solutions initiales en cherchant une utilisation efficace des ressources de traitement. Il s'agit bien ici d'un ajustement, c'est à dire l'application de légères modifications ne remettant pas la solution architecturale en cause. Par exemple, dans le cas des résultats pour la fonction *testGravité*, si nous rajoutons une LUT dans les clusters2 alors avec 30 *cluster2* nous obtenons un taux d'utilisation de 100% pour les comparateurs et un taux d'utilisation de 98% pour les LUT. De plus ce rajout ne diminue pas le nombre de communications dans les *cluster2*. Cependant nous pouvons noter qu'en fonction des ressources de routage utilisées il peut être difficile d'atteindre un taux d'utilisation des ressources de grain fin très élevé. Le concepteur peut se baser sur son expérience pour se donner un taux d'utilisation maximum des ressources de grain fin.

La figure 76 nous montre qu'il serait efficace de mettre un nombre important de *cluster2* dans les éléments hiérarchiques $H_{niveau2}$. En choisissant de positionner 10 *cluster2* dans les éléments $H_{niveau2}$, il sera nécessaire d'utiliser 3 éléments $H_{niveau2}$ pour assurer d'avoir le nombre suffisant de comparateurs et de LUT. Cependant avec 3 $H_{niveau2}$ il faudrait alors 20 *cluster1* par $H_{niveau2}$ pour assurer d'avoir le nombre nécessaire d'additionneurs. La figure 75 montre qu'il n'est pas nécessaire pour obtenir un nombre important de communications internes aux éléments $H_{niveau2}$ d'avoir un grand nombre de *cluster1*. De plus ce nombre important de *cluster1* rendrait difficile la mise en place d'un routage à coût constant dans les $H_{niveau2}$. Un compromis intéressant est d'augmenter alors le nombre d'additionneurs dans les *cluster1*. Avec 4 additionneurs dans les *cluster1*, nous pouvons réduire la taille des éléments $H_{niveau2}$ à 5 *cluster1*. La recherche de compromis demande au concepteur une bonne expertise des architectures modélisées basée sur son expérience.

En résumé l'architecture $A_{testGravité}$ que nous obtenons pour réaliser la fonction *testGravité* seule est maintenant définie par,

$$A_{testGravité} = 3H_{niveau2}\{\ 5cluster1\{4ADD,\ 1MUL,\ MEM1\}\ 10cluster2\{6COMP,\ 2LOGIC,\ 2MEM1\}\ \}$$

En utilisant la même démarche pour la fonction *envelop*, nous obtenons l'architecture $A_{envelop}$ définie par,

$$A_{envelop} = 2H_{niveau2}\{\ 9cluster1\{4ADD,\ 1MUL,\ MEM1\}\ 9cluster2\{5COMP,\ 2LOGIC,\ 2MEM1\}\ 1cluster3\{MEM2\}\ \}$$

Nous avons donc obtenu pour chacune des deux fonctions critiques une définition d'architecture pour laquelle une part importante des communications est assurée au plus bas niveau de hiérarchie (en interne aux éléments hiérarchiques *cluster1*, *cluster2* et *cluster3*). Ces architectures sont définies en tenant compte de deux aspects : la taille des éléments hiérarchiques de chaque niveau et le taux d'utilisation des ressources de traitement. Le choix de la taille des éléments hiérarchiques à chaque niveau doit prendre en compte l'hypothèse de départ qui est que les communications dans un niveau de hiérarchie donné ont un coût constant. Ce n'est plus le cas si les éléments hiérarchiques sont de taille trop importante. La contrainte à positionner sur la taille des éléments hiérarchiques dépend des ressources de routage qui sont utilisées à chaque niveau et de la connaissance que le concepteur a des performances et caractéristiques de ces ressources. Il est donc indispensable au concepteur de baser ses explorations sur des expériences pratiques. Nous avons montré au chapitre 3 comment cela était possible sur des architectures de grain fin, la démarche est plus délicate dans le cas d'architectures de gros grain car il est aujourd'hui difficile de disposer de données technologiques. En ce qui concerne le taux d'utilisation des ressources de traitements il doit être le plus élevé possible en particulier pour les ressources de gros grain (additionneur et multiplieur par exemple) qui sont plus consommatrices de puissance.

Dans le cas de l'application ICAM il y a deux fonctions critiques pour lesquelles deux architectures, $A_{testGravité}$ et $A_{envelop}$, ont été définies. Il faut à partir de ces deux définitions converger vers une définition unique pour toute l'application et effectuer les estimations pour toutes les fonctions de l'application. C'est ce que nous allons voir au paragraphe suivant.

5.2.5 Définition d'une architecture et estimation pour l'application complète

Pour définir une architecture compromis à partir des deux précédemment définies, il est nécessaire de calculer grâce aux résultats de l'estimation système le nombre de ressources

pour chacune des fonctions de l'application ICAM. Le tableau 17 donne un résumé des ressources nécessaires (flot de données sur 16 bits) aux différentes fonctions de l'application ICAM.

Fonction	Ressources de traitement			
	add/sub	mul/div	comp	logique
envelop	66	18	84	32
testGravité	60	1	176	59
reconstDilat	11	5	17	4
etiquet	107	40	123	43
errorBin	8	6	13	4
dilatBin	132	5	13	128
gethistogram	4	0	1	0
add	32	0	3	1
sub	33	0	3	1
div	1	32	1	0
absolute	32	0	32	32
thresholdAdapt	1	0	33	0
convolveTabHisto	23	18	14	2
histoThershold	2	0	3	1
Total	**512**	**125**	**516**	**328**

Tableau 17 – Besoins en ressources de traitement des fonctions de l'application ICAM.

Pour l'application ICAM les deux fonctions critiques donnent deux architectures qui sont assez proches. Nous observons que les *cluster1* et *cluster2* des architectures $A_{testGravité}$ et $A_{envelop}$ sont les mêmes à ceci près que le *cluster2* de l'architecture $A_{envelop}$ compte un comparateur de plus. Il est donc évident que l'architecture finale aura les mêmes *cluster1* et *cluster2*. Pour ce qui est du nombre de comparateurs dans le *cluster2* nous choisissons systématiquement le nombre minimum entre les deux architectures afin de converger vers les éléments hiérarchiques de plus petite taille. Une taille plus petite rend plus probable nos hypothèses de départ même si elle pénalise les coûts de communication. De même, pour le nombre de *cluster1* et de *cluster2* dans l'élément hiérarchique $H_{niveau2}$, nous choisissons, pour les mêmes raisons, le nombre minimum entre les deux architectures, soit dans ce cas précis 5 *cluster1* et 9 *cluster2*. Conformément à la définition de l'architecture $A_{envelop}$ il est nécessaire d'adjoindre un *cluster3* contenant une large mémoire pour la mémorisation des images. Le nombre de $H_{niveau2}$ nécessaire est calculé de façon à ce que l'architecture contienne l'ensemble des ressources de traitement pour réaliser toutes les fonctions de l'application (voir tableau 17).

Le compromis auquel nous arrivons pour satisfaire les architectures $A_{testGravité}$ et $A_{envelop}$ ainsi que les besoins en ressources de traitement est le suivant :

$$A_{ICAM} = 26\ H_{niveau2}\ \{\ 5cluster1\{4ADD,\ 1MUL,\ MEM1\}\ 9cluster2\{5COMP,\ 2LOGIC,\ 2MEM1\}\ 1cluster3\{MEM2\}\ \}$$

Avec cette architecture nous obtenons les taux d'utilisation donnés à la figure 77 pour chacune des ressources de traitement. Nous pouvons voir qu'ils sont comme nous le voulions particulièrement élevés pour les opérateurs de gros grain qui sont plus consommateur d'énergie.

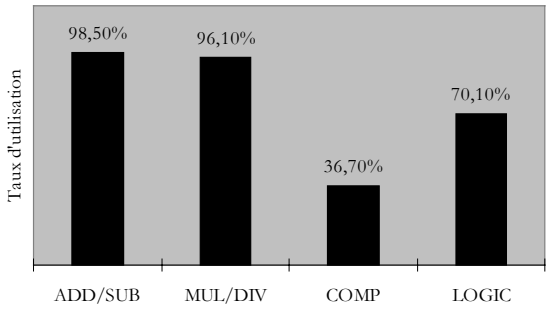


Figure 77 – Taux d'utilisation des ressources de traitement de l'architecture définie dans le cadre de l'application ICAM.

Le deuxième résultat que nous obtenons est la répartition des communications pour chacune des fonctions de l'application ICAM. Ce résultat est donné à la figure 78. Notons que sur les figures ci après, le niveau 3 (gris foncé) est le plus haut niveau de hiérarchie. Ce niveau est constitué d'une matrice d'éléments hiérarchiques de niveau 2 (gris clair) $H_{niveau2}$. Le niveau 1 (noir) est le niveau le plus bas à savoir l'intérieur des éléments hiérarchiques *cluster1* et *cluster2*.

Sur la figure 78, nous voyons que l'architecture est plus ou moins efficace selon la fonction considérée. Cependant les fonctions n'ont pas toutes la même importance en terme de nombre de communications comme on peut le voir sur la figure 79. Celle-ci nous montre le pourcentage de communications de l'application dans chacune des fonctions. Il apparaît clairement que les deux fonctions critiques *envelop* et *testGravité* représentent à elles seules plus de 75 % des communications de l'application, un peu plus de 53 % pour la fonction *envelop* et 25 % pour la fonction *testGravité*. Donc il est particulièrement important de regarder les résultats obtenus pour ces deux fonctions. La figure 80 montre la répartition précise des communications pour les deux fonctions *testGravité* et *envelop*. La répartition obtenue pour la fonction *testGravité* est conforme à ce que nous souhaitons obtenir. Dans ce cas 58 % des communications sont effectuées au niveau le plus bas de la hiérarchie et seulement 17 % au niveau le plus haut.

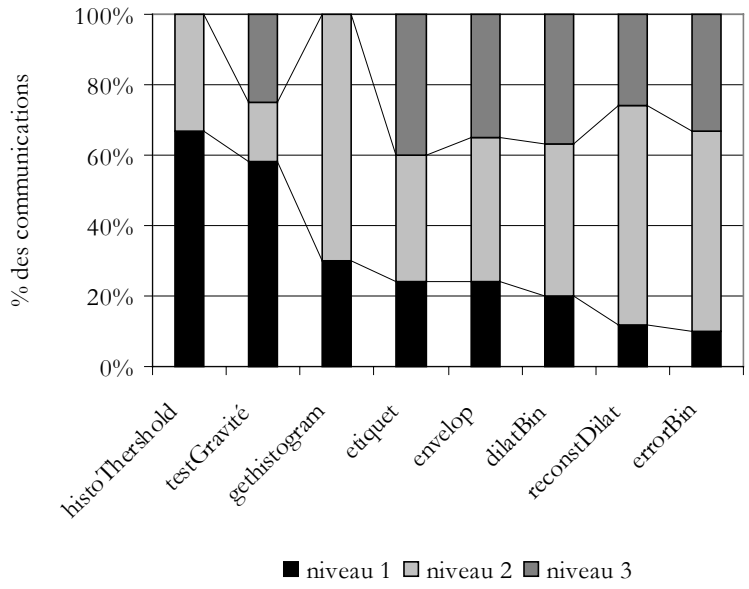


Figure 78 – Répartition des communications dans les trois niveaux de hiérarchie pour chacune des fonctions de l'application ICAM.

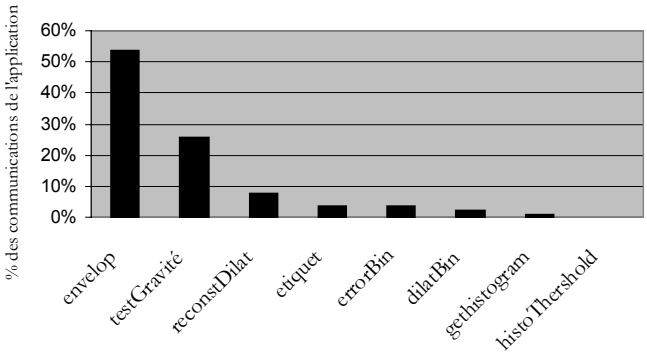
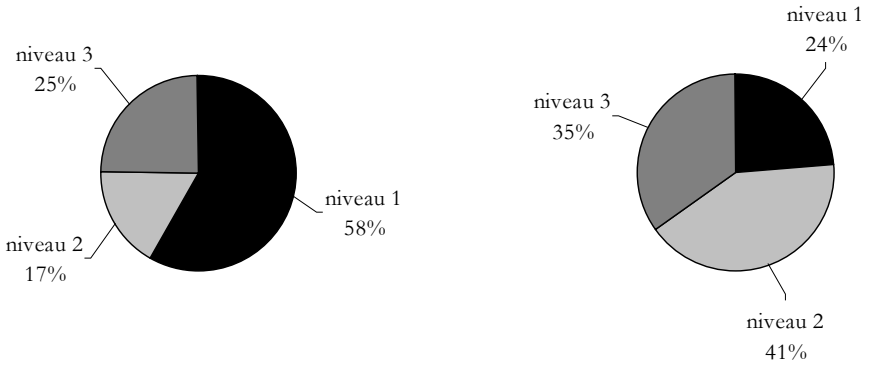


Figure 79 – Pourcentage du nombre total des communications représenté par chacune des fonctions de l'application ICAM.



a) répartition des communications pour la fonction *testGravité*

a) répartition des communications pour la fonction *envelop*

Figure 80 – Répartition des communications dans les trois niveaux de hiérarchie pour les fonctions a) testGravité et b) envelop.

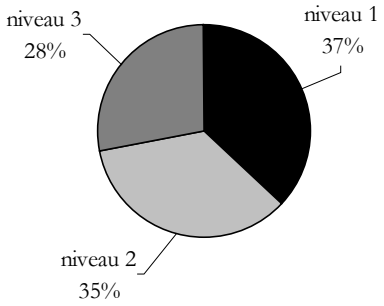


Figure 81 – Répartition des communications dans les trois niveaux de hiérarchie pour l'application ICAM.

Les résultats ne sont pas satisfaisants dans le cas de la fonction *envelop* pour laquelle le nombre de communications au niveau le plus bas est seulement de 24 %. Comme cette fonction représente plus de 50 % des communications, ses résultats dégradent ceux de l'application complète donnés à la figure 81. Nous allons par la suite tenter d'améliorer ce résultat.

La répartition des communications pour l'application complète (figure 81) est homogène avec environ un tiers des communications supportées par chacun des niveaux de hiérarchie. Le niveau le plus bas (niveau 1) supporte le plus grand nombre de

communications. Ce résultat ne paraît pas très bon dans une première approche car les communications au niveau 1, bien que plus nombreuses, ne sont pas largement prédominantes. Mais l'étude de la figure 80 nous a montré que cela tenait principalement aux mauvais résultats obtenus pour la fonction *envelop*. Il serait possible d'augmenter la part des communications de plus bas niveau en augmentant la taille des *cluster1* et *cluster2* pour assurer une meilleure répartition des communications pour la fonction *envelop*.

Nous avons tenté l'expérience d'augmenter la taille des *cluster1* et *cluster2* afin d'assurer une meilleure répartition des communications, c'est à dire d'augmenter le nombre de communications au niveau 1 au détriment du nombre de communications au niveau 3. La figure 82 donne la répartition obtenue qui est bien meilleure que celle obtenue auparavant et donnée à la figure 81.

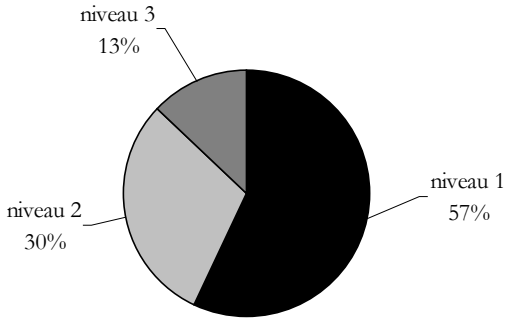


Figure 82 – Répartition des communications dans les trois niveaux de hiérarchie pour l'application ICAM avec des cluster1 et cluster2 de taille plus importante.

Cela pourrait bien entendu convenir mais il faut regarder la taille des éléments hiérarchiques de bas niveau :

Cluter1 : 20 additionneurs, 10 multiplieurs et 5 mémoires MEM1

Cluster2 : 21 comparateurs, 12 éléments logiques et 6 mémoires MEM1

Nous obtenons alors des éléments hiérarchiques de niveau 1 de taille bien trop importante pour pouvoir affirmer que les communications internes à ces éléments ont un coût constant. Si nous ne pouvons pas assurer notre hypothèse de base alors les résultats de la figure 82 ne sont pas valables. Dans le cas précis de l'application ICAM la fonction *envelop* est une fonction importante (53 % des communications totales), or les communications

qu'elle contient sont fortement réparties entre les ressources qui sont nécessaires à son exécution. De ce fait, la définition d'une architecture très efficace en ce qui concerne la répartition des communications est délicate dans ce cas.

Avant de clore la démonstration des résultats pour l'application ICAM, il est intéressant de montrer sur un exemple la répartition des communications obtenue si l'on choisit comme fonction critique une autre fonction que *testGravité* ou *envelop*. Nous avons choisi pour cet exemple la fonction *reconstDilat* qui conformément à la figure 79 est en troisième position en ce qui concerne le pourcentage des communications de l'application qu'elle représente (7,5 % du nombre total de communications).

La figure 83 donne la répartition des communications obtenue dans ce cas. Nous voyons la forte dégradation des résultats qui se présente sous deux formes. Tout d'abord, le pourcentage de communication au niveau le plus bas de hiérarchie passe de 37% dans le cas de la figure 81 à 27 % dans ce cas. De plus le pourcentage de communications au niveau le plus haut augmente de façon très conséquente de 28 % (figure 81) à 62 %. Or les communications au niveau le plus haut sont les plus coûteuses en terme de consommation de puissance. Nous voyons bien que si nous ne sélectionnons pas les bonnes fonctions critiques alors les résultats peuvent êtres considérablement dégradés. L'architecture définie dans ce cas n'est pas en adéquation avec les fonctions critiques de l'application.

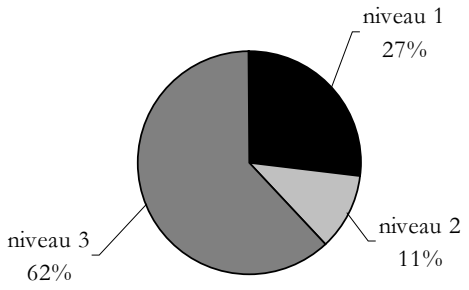


Figure 83 – Répartition des communications dans les trois niveaux de hiérarchie pour l'application ICAM en choisissant la fonction reconstDilat comme fonction critique.

Nous pouvons compléter les informations données à la figure 83 par celles du tableau 18. Celui-ci donne la dégradation de la répartition des communications dans l'architecture modélisée pour les fonctions *testGravité*, *envelop* et *reconsDilat* ainsi que pour l'application ICAM en entier. La dégradation est observée entre les résultats obtenus avec l'architecture orientée pour les fonctions critiques *testGravité* et *envelop* (Architecture 1 dans le tableau) et

les résultats obtenus avec l'architecture orientée pour la pseudo-fonction critique *reconstDilat*.

Le tableau 18 donne la dégradation observée pour le niveau 1 de communication (niveau le plus bas) qui correspond à une diminution du nombre de communications à ce niveau. Il donne aussi la dégradation observée pour le niveau 3 de communication (le niveau le plus haut) qui correspond cette fois à une augmentation du nombre de communications à ce niveau.

On peut remarquer à la vue de ces résultats, que la dégradation se fait essentiellement au niveau le plus haut pour les fonctions *testGravité* et *envelop* ainsi que pour l'application complète. Ce que nous avions déjà souligné lors de l'étude de la figure 83. Cette dégradation est très importante pour ces trois cas (jusqu'à 121 % de dégradation au niveau 3 pour l'application complète) et conduit inévitablement à une dégradation des performances de l'architecture. Pour la fonction *resconstDilat* la dégradation au niveau 3 est bien moins importante (seulement 14 % de dégradation). Elle se situe cette fois principalement au niveau le plus bas. Ces résultats confortent donc nos critères dans le choix des fonctions critiques.

	Répartition des communications					
	Architecture 1 *testGravité/envelop*		Architecture 2 *reconstDilat*		**Dégradation**	
	Niveau 1	Niveau 3	Niveau 1	Niveau 3	**Niveau 1**	**Niveau 3**
testGravité	58 %	25 %	50 %	47 %	**13 %**	**88 %**
envelop	24 %	35 %	15 %	77 %	**37 %**	**120 %**
reconstDilat	12 %	34 %	40 %	28 %	**- 75 %**	**- 14 %**
ICAM	37 %	28 %	27 %	62 %	**27 %**	**121 %**

Tableau 18 - Dégradation comparée de la réparation des communications entre les deux architectures considérées auparavant.

5.2.6 Analyse et conclusion

Cette application nous a permis de voir la démarche qui permet de définir une architecture à partir d'une spécification de haut niveau de l'application. L'architecture obtenue doit être efficace du point de vue de la répartition des communications pour exécuter l'application. De plus le taux d'utilisation des ressources de traitements, en particulier de gros grain, doit être élevé. Ces deux points permettent d'assurer conformément à ce que nous avons montré dans le chapitre 3, que l'architecture définie pour l'application est efficace en ce qui concerne la consommation de puissance.

Nous voyons que la démarche que nous mettons en œuvre n'est pas un processus simple et demande une expertise de la part du concepteur. Cependant cette démarche permet de

rapidement définir une architecture matérielle reconfigurable pour une application et ceci à haut niveau d'abstraction.

Nous allons voir dans les paragraphes suivants les résultats obtenus pour deux autres applications du traitement d'images. Les résultats obtenus pour ces applications sont assez proches de ceux obtenus pour l'application ICAM. En particulier ils conduisent tous à séparer dans l'architecture les traitements de gros grain et les traitements de grain fin ainsi qu'à une distribution spatiale des ressources de mémorisation plus efficace que le regroupement des données dans de grands bancs mémoires.

Nous avons pour cet exemple ICAM complété l'exploration classique par deux études. La première consistait à montrer qu'il n'était pas toujours possible de converger vers une architecture pour laquelle la répartition des communications serait optimale. Effectivement nos hypothèses imposent que la taille des éléments hiérarchiques soit maîtrisée afin de pouvoir toujours assurer un coût de communication constant dans un niveau hiérarchique. Le cas échéant il serait probablement intéressant d'étendre notre modèle d'architecture en augmentant le nombre de niveaux hiérarchiques (aujourd'hui limité à trois).

La deuxième étude devait nous permettre de confirmer que le choix de nos fonctions critiques était le bon. Pour cela nous avons effectué une exploration en considérant comme critique une fonction qui ne l'était pas par rapport aux autres. Aussi nous avons défini une architecture adaptée à cette fonction. Nous avons obtenu comme résultat qu'une forte dégradation des résultats obtenus pour les fonctions initialement considérées comme critiques entraînait une même dégradation pour les résultats de l'application complète.

5.3 Application Matching Pursuit

5.3.1 Présentation

L'application Matching Pursuit est une application de compression d'image. A l'instar des compressions déjà développées, elle n'opère pas au niveau des pixels de l'image mais sur des "atomes" représentant des motifs de base constitutifs de l'image. Cette application est intéressante à deux titres : d'une part la spécification de celle-ci est encore en cours et mouvante, il est donc intéressant pour le concepteur de disposer d'un outil lui permettant de caractériser et d'évaluer rapidement ses choix algorithmiques et architecturaux. D'autre part cet exemple est représentatif des applications multimédias futures.

La partie codage est réalisée à l'aide d'un algorithme génétique qui est exécuté sur un serveur, nous avons donc décidé de ne pas nous intéresser à cette partie. Par contre, la partie décodage peut être implantée sur différents types de systèmes, dont les systèmes embarqués, nous l'avons donc estimée. Elle est composée de quatre fonctions principales illustrées sur la figure 84. Pour cette application les traitements sont majoritairement des additionneurs et des multiplieurs, les comparateurs eux sont utilisés pour la prise de décision dans les boucles.

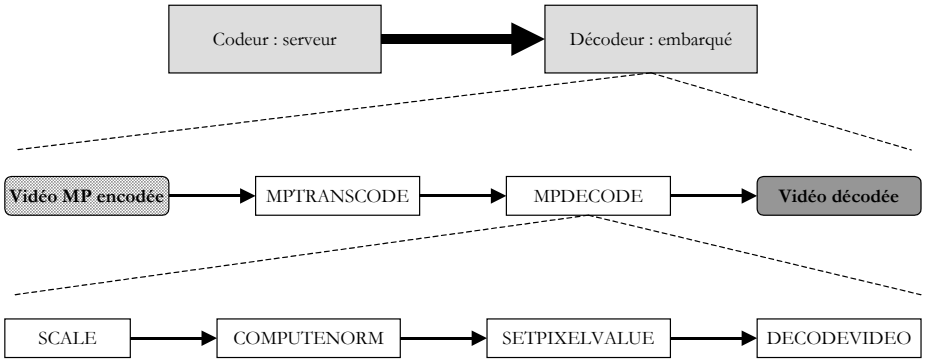


Figure 84 – Chaîne de traitements de l'application Matching Pursuit.

5.3.2 Exploration architecturale

Nous allons, comme pour l'application ICAM, réaliser le processus d'exploration architecturale. Dans un premier temps il faut déterminer la criticité des quatre fonctions qui composent l'application. Il faut donc les placer dans les plans de criticité suivant la localité spatiale des communications et la congestion des ressources de routage comme on le voit sur la figure 85. Sur celle-ci nous voyons clairement qu'une fonction est plus critique que les autres, la fonction *ComputeNorm*. Dans un premier temps nous effectuons l'exploration architecturale uniquement sur cette fonction.

L'étude de l'ACG de cette fonction, l'exploration de la taille mémoire et l'exploration du nombre d'éléments fonctionnels dans les éléments hiérarchiques de plus bas niveaux nous conduisent à la définition de l'architecture suivante :

$$\mathcal{A}_{ComputeNormt} = \{cluster1\{8ADD,\ 6MUL\ \}\ cluster2\{3COMP\}\ cluster3\{MEM\}\}$$

Cette architecture sera celle définie pour l'application puisque ici il n'y a qu'une fonction jugée critique. Celle-ci impose, aux trois autres fonctions, l'architecture la plus en adéquation avec elle suivant la répartition des communications et le taux d'utilisation des ressources de traitement.

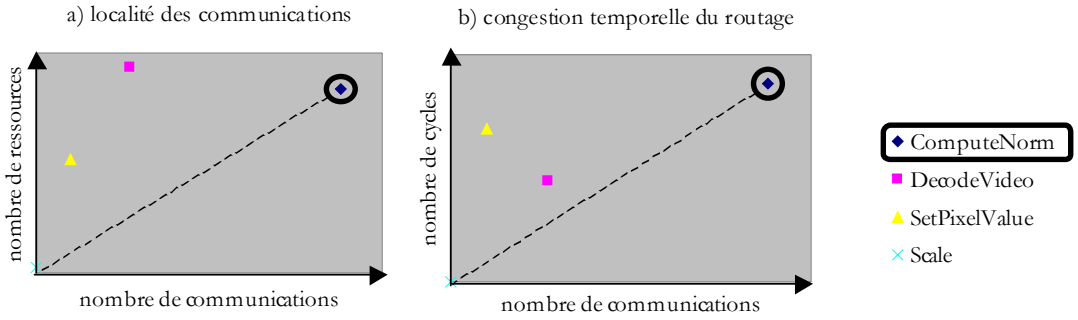


Figure 85 – Placement des quatre fonctions de l'application Matching Pursuit dans les plans de criticité suivant a) la localité spatiale des communications et suivant b) la congestion temporelle des ressources de routage.

5.3.3 Résultats obtenus

L'architecture pour l'application entière est obtenue par exploration de la taille des éléments hiérarchiques intermédiaires et par l'étude du taux d'utilisation des ressources de traitement. L'architecture obtenue en résultat est la suivante :

$$A_{MatchingPursuit} = 5H_{niveau2}\{\ 6cluster1\{8ADD,\ 6MUL\ \}\ 5cluster2\{3COMP\}\ 1cluster3\{MEM\}\}$$

La figure 86 donne le résultat obtenu en terme de taux d'utilisation des ressources de traitement. Rappelons que pour cette application les traitements sont majoritairement des additionneurs et des multiplieurs, les comparateurs eux sont utilisés pour la prise de décision dans les boucles. Les taux d'utilisation obtenus dans ce cas sont particulièrement élevés puisque compris entre 90 et 97 %. L'architecture est juste dimensionnée pour cette application. La figure 87 donne la répartition des communications à travers les trois niveaux de hiérarchie de l'architecture pour les quatre fonctions de l'application. Cette figure donne aussi séparément la répartition obtenue pour l'application complète. A la vue de ces résultats, nous pouvons observer que la répartition finale des communications dans l'architecture pour toute l'application est très proche de celle que nous avions obtenue pour l'application ICAM. Chaque niveau de hiérarchie prend en charge environ un tiers des communications. Cependant le niveau le plus bas (niveau 1) prend, en charge un plus grand nombre de communications que les deux autres niveaux. Il serait possible comme pour ICAM, de définir une architecture qui aboutirait à de meilleurs résultats mais alors la taille des clusters de bas niveau serait trop importante pour garantir des coûts de communications constants dans ces éléments.

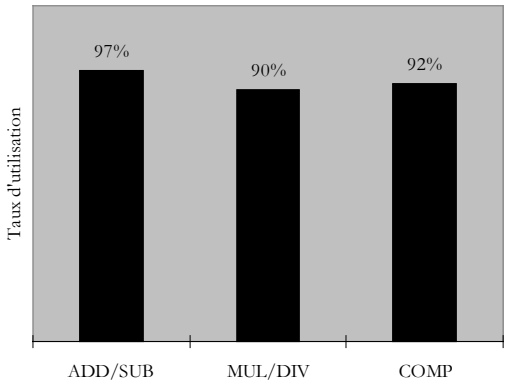
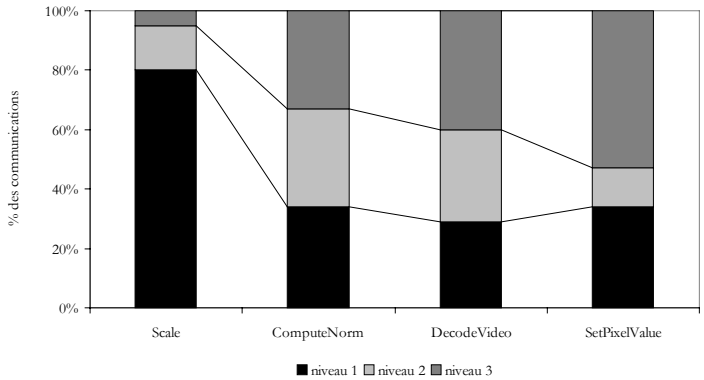
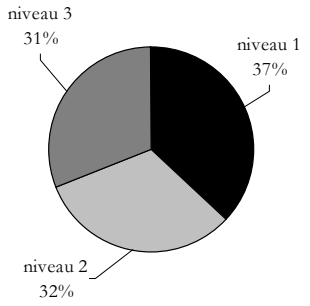


Figure 86 – Taux d'utilisation des ressources de traitement de l'architecture définie dans le cadre de l'application Matching Pursuit.

La ressemblance des résultats entre les deux applications est normale puisqu'elles manipulent le même type de données en utilisant les mêmes types de traitements. C'est pourquoi il pourrait être envisageable de définir une architecture qui puisse être utilisée pour les deux applications.



a) répartition des communications pour les quatre fonctions de l 'application



b) répartition des communications pour l'application complète

Figure 87 – Répartition des communications dans les trois niveaux de hiérarchie pour a) les quatre fonctions de l'application Matching Pursuit et pour b) l'application complète.

5.4 Application codeur MPEG-2

5.4.1 Présentation

Cette troisième application est aussi une application de traitement d'images pour la vidéo. Il s'agit du codeur compatible avec la norme MPEG-2. Cette norme est complémentaire à la norme MPEG-1. La norme MPEG-1 visait le marché des disques optiques à lecture seule, alors que MPEG-2 est destinée à la définition de programmes de télévision. Ces normes utilisent un codage proche du codage JPEG pour certaines des images du flux vidéo. Mais compte tenu des taux de compression obtenus pour les images fixes (JPEG) et de la cadence d'affichage de ces images (25 à 30 images par seconde), il n'est pas possible de les transmettre à des débits suffisamment faibles. L'idée des normes MPEG est de tirer profit de la forte redondance temporelle des images en utilisant un processus de prédiction des points de l'image à partir de leurs homologues dans les images voisines. La figure 88 donne le flot classique d'un codeur MPEG.

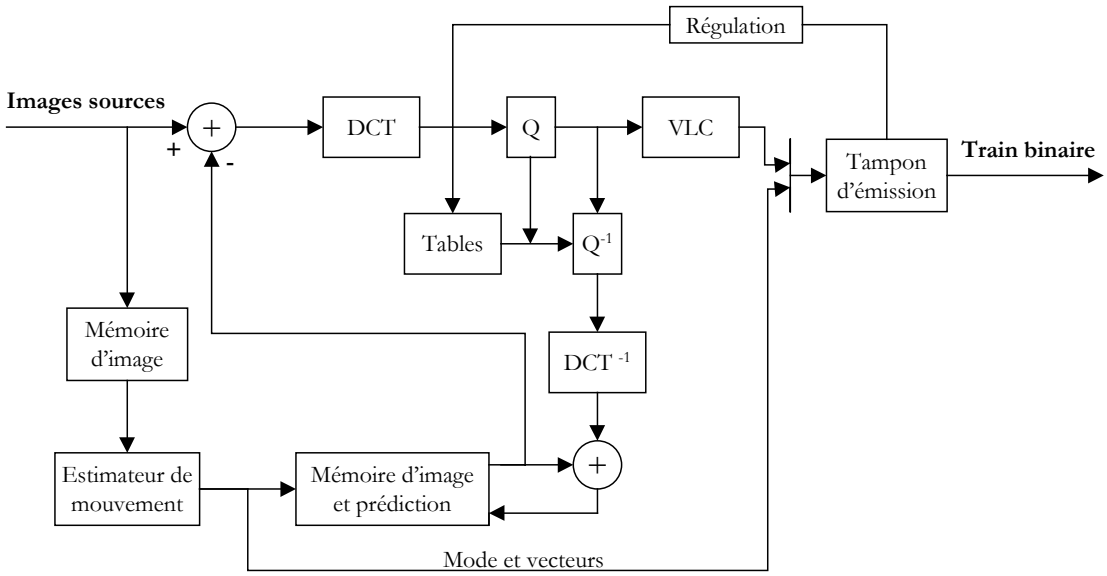


Figure 88 – Flot d'un codeur MPEG.

5.4.2 Exploration architecturale

Nous allons comme pour les applications précédentes réaliser le processus d'exploration architecturale. Dans un premier temps il faut déterminer la criticité des fonctions qui composent l'application. Il faut donc les placer dans les plans de criticité suivant la localité spatiale des communications et la congestion des ressources de routage comme on le voit sur la figure 89. Sur celle-ci nous voyons que la fonction *Core_motion* est la plus critique pour la localité spatiale des communications, et la fonction *transform_with_dct* est la plus

critique pour la congestion temporelle des ressources de routage. Nous effectuons l'exploration architecturale uniquement sur ces deux fonctions.

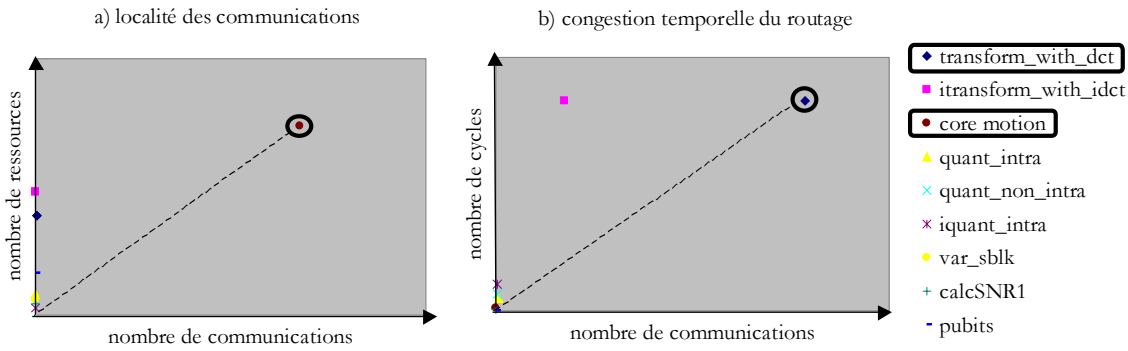


Figure 89 – Placement des fonctions de l'application codeur MPEG-2 dans les plans de criticité suivant a) la localité spatiale des communications et suivant b) la congestion temporelle des ressources de routage

L'étude de l'ACG des deux fonctions critiques, l'exploration de la taille mémoire et l'exploration du nombre d'éléments fonctionnels dans les éléments hiérarchiques de plus bas niveaux nous donnent la définition des deux architectures suivantes :

$$A_{transform\_with\_dct} = \{\ cluster1\{4ADD,\ 4MUL,\ MEM\}\ cluster2\{1COMP,\ 1LOGIC\}\ \}$$

$$A_{core\_motion} = \{\ cluster1\{3ADD,\ 2MUL,\ MEM\}\ cluster2\{2COMP,\ 1MEM\}\ \}$$

Ces architectures permettent de définir par compromis une architecture pour l'application puisque ici il y a deux fonctions critiques. On retrouve le même type d'éléments hiérarchiques *cluster1* et *cluster2* que pour les deux autres applications qui étaient des applications traitant les même types de données.

5.4.5 Résultats obtenus

L'architecture pour l'application entière est obtenue par exploration de la taille des éléments hiérarchiques intermédiaires et par l'étude du taux d'utilisation des ressources de traitement. L'architecture obtenue en résultat est la suivante :

$$A_{MPEG\text{-}2} = 15H_{niveau2}\{\ 7cluster1\{4ADD,\ 4MUL,\ MEM\}\ 7cluster2\{2COMP,\ 1LOGIC,\ MEM\}\}$$

La figure 90 donne le résultat obtenu en terme de taux d'utilisation des ressources de traitement. Pour cette application les taux d'utilisation des ressources de gros grain sont élevés et varient entre 67 et 70 %. Par contre les opérateurs de grain fin ont des taux d'utilisation bien inférieurs. Ceci est dû au faible nombre d'opérateurs de grain fin utilisés par l'application relativement au nombre d'opérateurs de gros grain. La figure 91 donne la répartition des communications à travers les trois niveaux de hiérarchie de l'architecture pour les fonctions de l'application. Cette figure donne aussi séparément la répartition obtenue pour l'application complète.

A la vue de ces résultats, nous pouvons observer que la répartition finale des communications dans l'architecture pour toute l'application est différente de celle que nous avions obtenue pour les deux applications précédentes. Une forte majorité des communications, 63 %, est estimée réalisée à l'intérieur des éléments hiérarchiques de plus bas niveau (le niveau 1). Ceci s'explique entre autres parce que l'architecture est bien adaptée à la fonction *core_motion* qui représente une importante part des communications, d'où l'intérêt de l'avoir sélectionnée comme fonction critique de l'application.

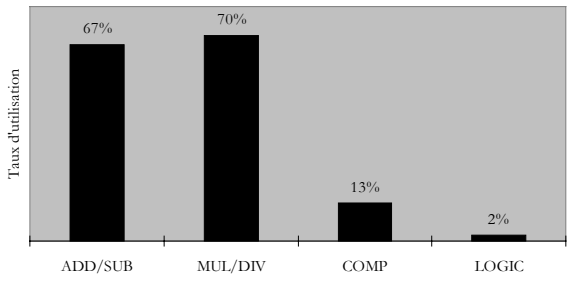
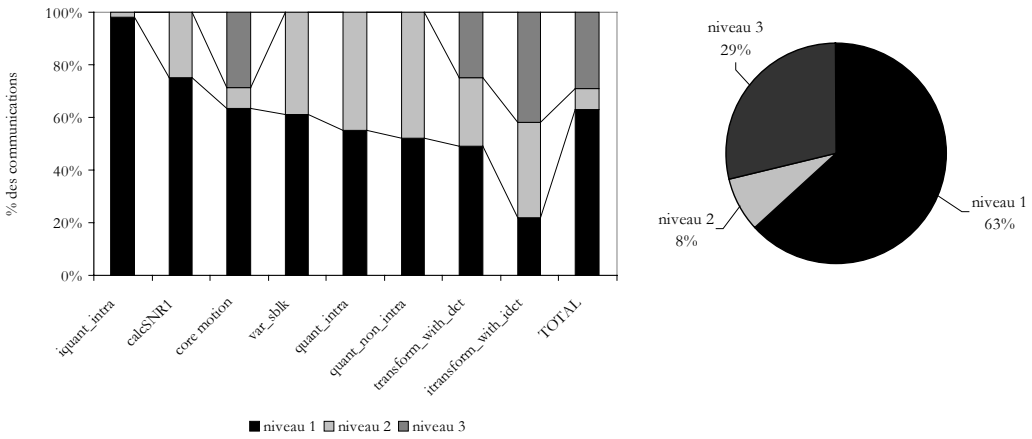


Figure 90 – Taux d'utilisation des ressources de traitement de l'architecture définie dans le cadre de l'application Codeur MPEG-2.

Les trois applications que nous venons d'étudier, ICAM, Matching Pursuit et le codeur MPEG-2 sont toutes des applications du domaine du traitement des images. De ce fait nous avons vu que les architectures qui sont en adéquation avec ces trois applications sont très proches. En particulier elles ont deux types d'éléments hiérarchiques de bas niveau : un type d'éléments contenant des ressources de traitement de gros grain et un deuxième contenant des ressources de traitement de grain fin. Qu'en est-il si nous prenons une application d'un autre domaine ?

La réponse à cette question est nécessaire car si nous convergeons vers les mêmes architectures alors cela montrera que le résultat architectural est dépendant de l'outil. Si ce n'est pas le cas, alors on pourra conclure que le résultat architectural est dépendant du domaine d'applications.



a) répartition des communications pour les quatre fonctions de l 'application

b) répartition des communications pour l'application complète

Figure 91 – Répartition des communications dans les trois niveaux de hiérarchie pour a) les quatre fonctions de l'application Codeur MPEG-2 et pour b) l'application complète.

5.5 Application au cœur de cryptage AES

5.5.1 Présentation

L'AES (*Advanced Encryption Standard*) est un algorithme de chiffrement par blocs à clé secrète, c'est à dire que les données sont considérées par blocs de taille fixe tout au long de l'algorithme et que la même clé sert au cryptage et au décryptage.

La figure 92 montre le déroulement de l'algorithme. Il se décompose en quatre opérations : *ByteSub*, *ShiftRow*, *MixColumn* et *AddRoundKey*. La transformation *ByteSub* est une substitution non-linéaire. L'opération *ShiftRow* est une rotation cyclique. Pendant l'opération *MixColumn* les données sous forme matricielle subissent une transformation polynomiale. Enfin, l'opération *AddRoundKey* effectue une addition par OU exclusif bit-à-bit entre les données et les éléments de la clé.

Cette application n'est pas d'une grande complexité et il nous est possible de la traiter sous la forme d'un seul HCDFG, donc d'une seule fonction. Ce qui veut dire que nous faisons l'économie de plusieurs explorations architecturales. De plus, il nous est possible à l'issue de l'estimation système de choisir une réalisation matérielle très parallèle pour cette application.

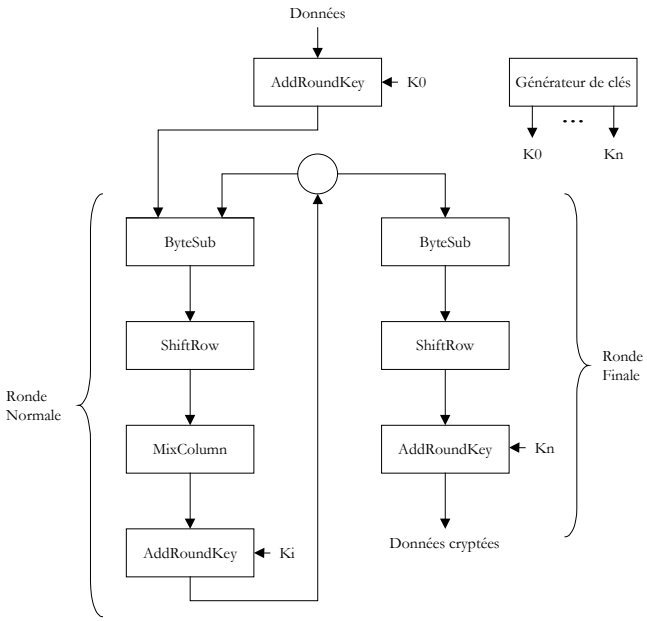


Figure 92 – Flot de l'algorithme AES.

5.5.2 Résultats obtenus

L'étude de l'ACG de l'application, l'exploration de la taille mémoire et l'exploration du nombre d'éléments fonctionnels dans les éléments hiérarchiques de plus bas niveaux nous donnent la définition d'une architecture. Celle-ci est affinée grâce à l'exploration de la taille des éléments hiérarchiques intermédiaires et par l'étude du taux d'utilisation des ressources de traitement. L'architecture obtenue en résultat est la suivante :

$$A_{AES} = 4H_{niveau2}\{\ 4cluster1\{1ADD,\ 1MUL,\ 1COMP,\ 1LOGIC,\ MEM\}\}$$

C'est donc une architecture qui ne contient qu'un type d'élément hiérarchique au plus bas niveau de la hiérarchie (*cluster1*) et non plus deux éléments hiérarchiques séparant les traitements de gros grain et de grain fin comme pour les applications précédentes. Ici du fait même des traitements appliqués aux données, il est indispensable de regrouper tous les types de ressources de traitement dans un même élément hiérarchique. Nous obtenons des taux d'utilisation des ressources de traitement particulièrement élevés qui varient entre 68 et 100%, comme l'indique la figure 93. La répartition des communications dans les différents niveaux de hiérarchie de l'architecture pour réaliser l'application est donnée à la figure 94. Nous obtenons une très bonne répartition des communications puisque 69% de celles ci sont pris en charge par le niveau le plus bas de la hiérarchie (niveau 1). Du fait de la faible complexité de cette application nous obtenons donc une très bonne adéquation entre l'architecture résultat et l'application.

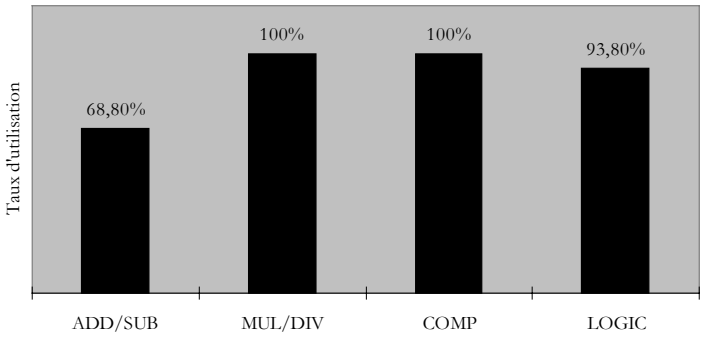


Figure 93 – Taux d'utilisation des différentes ressources de traitement pour l'application de cryptage AES.

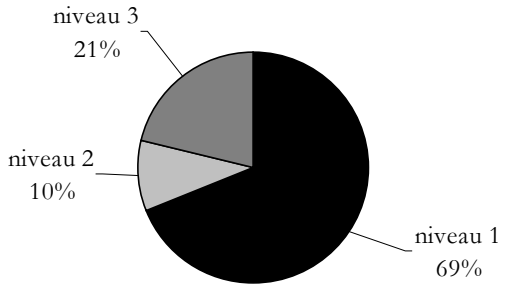


Figure 94 – Répartition des communications dans les trois niveaux de hiérarchie pour l'application de cryptage AES.

Nous obtenons pour cette application de cryptage un résultat architectural différent de ce que nous avons obtenu pour les applications du domaine du traitement des images. Effectivement la principale différence est que les ressources de traitement de grain fin et de gros grain ne sont pas séparées dans ce cas. Il est légitime de se demander alors quelle est la répartition des communications pour cette application avec une architecture qui sépare les ressources de traitement de grain fin et de gros grain dans deux types distincts d'éléments hierarchiques. La figure 95 donne l'estimation de cette réaprtition dans ce cas. Nous obeservons une dégradation forte des résultats puisqu'il y a par rapport à la figure 94 une diminution de 19% des communications au niveau 1 de l'architecture (le niveau le plus bas) et une augmentation de 15% des communications au niveau 3 de l'architecture (le niveau le

plus haut). Il est donc possible de définir des architectures spécifiques à des domaines d'application (cryptographie, traitements des images, télécommunication …).

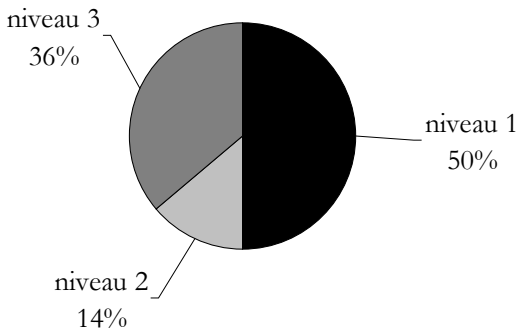


Figure 95 – Répartition des communications dans les trois niveaux de hiérarchie pour l'application de cryptage AES avec une architecture à granularité séparée (comme pour les applications du traitement des images).

5.7 Conclusion

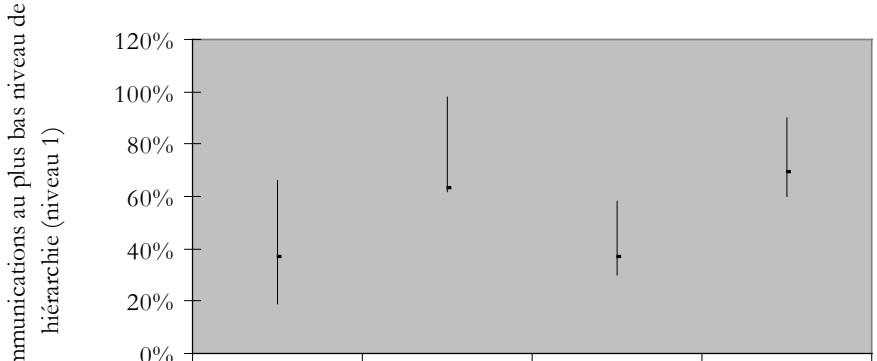
Ce chapitre a présenté les résultats de notre démarche d'exploration de l'espace de conception architecturale obtenus pour quatre applications. Trois applications sont du domaine du traitement des images et une dernière application est du domaine de la cryptographie. Une étude pas à pas de la démarche pour la première application nous a montré comment il est possible avec un haut niveau d'abstraction de converger vers une architecture matérielle. Bien sûr cette architecture reste abstraite puisque sa description est basée sur un modèle abstrait structurel et fonctionnel. Cependant l'outil d'estimation relative de Design Trotter, donne un certain nombre d'informations qui guident le concepteur dans sa démarche de définition d'une architecture. En particulier la répartition des communications dans l'architecture lui permet dans les raffinements successifs et à plus bas niveau d'abstraction de motiver son choix de ressources physiques de routage. Mais aussi il obtient des informations sur l'utilisation des ressources de traitement qui sont présentes dans l'architecture. Notre outil est le seul aujourd'hui à proposer de tels résultats avec un tel niveau d'abstraction tant au niveau de la spécification des applications qu'au niveau de la modélisation des architectures reconfigurables.

Nous avons vu que la démarche d'exploration demande une bonne expertise de la part de du concepteur, comme c'est le cas pour une grande majorité d'outils d'exploration

architecturale. Il est en outre nécessaire qu'il ait des données technologiques sur les ressources de routage qui seront par la suite mises en œuvre dans le circuit reconfigurable. Actuellement nous menons des travaux sur l'utilisation d'un outil de placement routage bas niveau, l'outil MADEO-BET présenté au chapitre 2. Celui-ci doit nous permettre d'enrichir notre exploration architecturale avec entre autres des informations sur la difficulté de placement routage en fonction des ressources de routages.

Tous les résultats concernant la répartition des communications dans ce chapitre ont été donnés suivant les résultats de l'algorithme intermédiaire de la projection matérielle. Cependant nous obtenons à chaque fois trois résultats pour les trois algorithmes. L'algorithme minimum donne un nombre maximum de communications dans les éléments hiérarchiques de plus bas niveau ce qui conduit à un coût de communication minimum. L'algorithme maximum donne un résultat inverse. Enfin, l'algorithme intermédiaire donne une valeur comprise entre celles fournies par les deux autres algorithmes, mais plus près de l'algorithme maximum. Effectivement nous avons vu que l'algorithme minimum prend des hypothèses non réalistes.

La figure 96 nous montre que les intervalles obtenus entre les valeurs données par l'algorithme minimum et par l'algorithme maximum diffèrent suivant les applications. L'application de cryptage AES et l'application de compression Matching Pursuit ont les intervalles les plus courts. L'application de détection d'objets en mouvement ICAM a l'intervalle le plus long. Conformément à notre hypothèse développée au quatrième paragraphe, le grand intervalle obtenu dans le cadre de la fonction ICAM prouverait que les choix d'assignations et de réalisations matérielles de l'application sont nombreux. Effectivement il s'agit d'une application complexe nécessitant l'utilisation d'un grand nombre de ressources de l'architecture. De ce fait il y a plus de possibilités d'optimisation lors de la synthèse. Nous pensons donc que les outils de conception qui seront utilisés par la suite doivent être particulièrement efficaces dans ce cas car ils auront un fort impact sur les performances finales de l'application. Dans le cas des applications AES et Matching Pursuit il est normal d'obtenir des intervalles plus réduits car ces applications sont beaucoup moins complexes. Elles nécessitent moins de ressources matérielles et donc seront plus simples à synthétiser sur l'architecture cible. Cela ne veut pas dire que les outils de synthèse n'auront pas d'impact sur les performances finales de ces deux applications. Mais ils auront un impact inférieur à ce qu'ils pourraient avoir pour l'application ICAM. Nous pouvons ainsi compléter les informations que l'outil donne à l'utilisateur. Celui-ci est en mesure de prendre en compte l'impact des outils de synthèse qu'il utilisera pour la réalisation finale de l'application qu'il développe.



	ICAM	MPEG	Matching	AES
MAX	19%	62%	30%	60%
MIN	66%	98%	58%	90%
INTER	37%	63%	37%	69%

Figure 96 – Résultats comparés des trois algorithmes concernant le pourcentage de communication au niveau le plus bas de hiérarchie pour les quatre applications.

Enfin, nous obtenons un résultat intéressant : les architectures sont dépendantes du domaine d'applications. Il s'agit bien entendu d'un résultat communément admis, il était donc indispensable de le confirmer avec notre outil. Ce résultat va dans le sens des évolutions futures des architectures des circuits reconfigurables qui seront de plus en plus orientées pour un domaine d'applications comme nous l'avions précisé à la fin du premier chapitre de ce document.

Conclusions et perspectives

6.1 Conclusion

Les architectures reconfigurables ont ouvert un large champ de recherches tant au niveau architectural qu'applicatif. La maîtrise de ces architectures est un processus complexe et celle-ci ne pourra se faire qu'avec l'apparition d'outils d'exploration et de synthèse travaillant à tous les niveaux de conception. Nous avons présenté dans ce document une méthode d'exploration de l'espace de conception des architectures reconfigurables qui prend place très tôt dans le flot de conception. Cette méthode a abouti à la définition d'un outil logiciel d'estimation de la répartition des communications dans les architectures reconfigurables.

6.1.1 Réponse à la problématique et travail réalisé

Nous avons dans ce document répondu à la problématique présentée en introduction, à savoir si il était possible de définir une méthode d'exploration des architectures reconfigurables travaillant à haut niveau d'abstraction. La méthode que nous proposons s'appuie essentiellement sur l'analyse de la répartition estimée des communications dans les architectures pour une application donnée. En effet, ce critère est essentiel afin de concevoir une architecture efficace tant du point de vue des performances temporelles que de la consommation de puissance. Cette analyse a été rendue possible grâce à des études que nous avons menées sur l'impact des éléments constitutifs des architectures reconfigurables de grain fin sur la consommation de puissance. Les résultats obtenus ont par extrapolation été appliqués aux architectures reconfigurables de gros grain dans la mesure ou aucune étude sérieuse n'a été menée sur le sujet. Les hypothèses que nous avons développées nous semblent néanmoins tout à fait pertinentes. La méthode d'exploration que nous proposons s'appuie sur un outil d'estimation, permettant l'exploration de façon semi-automatique. Il est complètement intégré comme entité de l'outil Design Trotter. Cet outil demande au concepteur une bonne expertise du domaine car son expérience reste un élément important lors du processus d'exploration. En effet l'outil développé correspond à un guide pour la conception et non à un générateur d'architectures.

6.1.2 Résultats

Nous avons montré dans ce document que notre méthode pouvait être utilisée pour des applications différentes, tant au niveau du domaine d'applications que de leur complexité.

Le flot est rapide à mettre en œuvre mais sa vitesse d'exécution dépend bien évidemment des performances du matériel informatique utilisé et de la complexité des applications.

Les explorations architecturales que nous avons pour le moment menées sont très riches d'informations et confirment les intuitions que nous avions sur le développement d'architectures reconfigurables. Entre autres, la structure des architectures reconfigurables dépend du domaine d'applications. C'est à dire qu'une architecture efficace (nous parlons essentiellement de la consommation de puissance) pour un domaine ne l'est pas forcément pour un autre ou du moins ne correspond pas forcément à la plus performante des architectures. Ceci confirme que dans l'avenir les architectures devront être relativement dédiées à un domaine d'applications. Par exemple, nous avons observé que pour des applications de traitement des images, il était plus intéressant de partager l'architecture en deux parties, une partie spécifique aux traitements de type gros grain et une partie spécifique aux traitements de type grain fin. Nous avons également mis en évidence qu'il était plus intéressant de partager les ressources mémoires sur l'ensemble de l'architecture plutôt que de créer de gros bancs mémoires, sauf dans le cas de mémoire d'images. Une architecture efficace doit donc être composée de plusieurs tailles d'éléments mémoires pour des applications du traitement des images. Il s'agit d'une tendance actuelle comme le prouve par exemple l'architecture du circuit FPGA Altera Stratix.

Nous n'avons pas encore effectué toutes les explorations possibles, car les applications que nous avons utilisées étaient limitées à deux domaines applicatifs. Cependant nous avons maintenant à notre disposition un outil complet qui peut nous permettre de mettre en place un processus d'exploration avec un spectre plus large d'applications. Cette étude permettra d'identifier les classes d'architectures adaptées aux classes d'applications.

6.1.3 Analyses critiques

Il est indispensable de garder en tête le niveau actuel de développement des architectures reconfigurables, en particulier à gros grain, pour critiquer de façon constructive nos travaux. Effectivement aujourd'hui très peu d'architectures développées et répondant à ce modèle, le sont réellement de façon physique (sur silicium), ce qui rend l'estimation des performances potentielles de ces architectures très complexe, voir impossible. C'est pourquoi les travaux présentés dans ce document sont avant tout des travaux fondamentaux, qui se basent sur une série d'hypothèses. Les hypothèses utilisées, comme nous l'avons montré au troisième chapitre, sont issues d'une étude des architectures de grain fin dont les résultats ont été extrapolés aux architectures de gros grain. La modélisation que nous faisons des architectures reconfigurables est très efficace mais peu précise. Aussi il ne faut pas percevoir l'outil développé comme une fin en soi. Il est un outil d'aide à la conception qui n'a pas pour but d'estimer de façon précise les performances d'une architecture pour une application, il a pour but de permettre l'exploration de l'espace de conception. Comme nous l'avons indiqué cet outil s'inscrit au sein d'un flot plus complet d'exploration qui conduit par raffinements successifs à la définition précise d'architectures reconfigurables.

Bien sûr, il est nécessaire de vérifier de façon plus approfondie les hypothèses utilisées dans ces travaux. Pour cela nous imaginons confronter les résultats que nous obtenons aux

résultats d'autres outils proches ne travaillant pas de la même façon ou à des implémentations réelles. Une spécialisation de nos travaux est aussi indispensable pour converger vers des architectures plus réalistes. De plus, ces travaux se sont concentrés sur une exploration orientée par la consommation de puissance, il serait intéressant d'étudier d'autres critères pour guider l'exploration. Ces réflexions nous amènent tout naturellement à envisager les perspectives de ces travaux.

6.2 Perspectives

6.2.1 Perspectives à courts termes

Les premiers travaux qui s'imposent sont de vérifier et/ou de compléter les résultats que l'outil d'estimation propose avec d'autres outils. Malheureusement il n'existe pas d'outil travaillant avec le même niveau d'abstraction. Nous pensons qu'il est toutefois possible de confronter les résultats obtenus en intégrant à notre approche un outil tel que l'outil de placement & routage générique Madeo BET développé par le laboratoire d'informatique de l'Université de Brest et présenté au deuxième chapitre de ce document. Cet outil peut nous donner des indications sur la routabilité et la pertinence des architectures que nous proposons. Ainsi nous pourrons confronter nos résultats et prendre en compte les possibles divergences. Nous pouvons aussi évaluer pour le niveau d'abstraction auquel nous travaillons les informations qu'il serait intéressant d'obtenir via Madeo BET et inversement, ainsi cet outil pourrait également évoluer grâce à notre expertise.

Le flot de l'outil que nous proposons doit être complété et étendu afin de proposer un outil logiciel complet. Certaines parties du flot ne sont effectivement pas encore développées, comme les arbres technologiques qui nous permettraient via des pré-décompositions d'opérateurs de gros grain de pouvoir cibler pour une même application, des architectures de gros grain comme de grain fin.

Il serait bien sûr intéressant de continuer nos investigations architecturales en modélisant d'autres applications et architectures. Nous portons un intérêt particulier au domaine de la cryptographie. Après AES il serait intéressant de confirmer les résultats obtenus avec d'autres algorithmes de cryptage. Nous pensons effectivement, en accord avec de nombreux spécialistes du domaine, que les architectures reconfigurables sont particulièrement adaptées au domaine de la cryptographie.

Enfin, nous pourrions utiliser notre outil dans le cadre d'applications du domaine de la Radio Logicielle qui est aussi un domaine particulièrement intéressant pour les architectures reconfigurables.

6.2.2 Perspectives à moyens termes

Il y a un intérêt particulier à spécialiser notre outil dans le cadre de travaux complémentaires. Effectivement notre outil souffre de sa vision très large du domaine architectural ce qui réduit sa précision. Aussi la spécialisation de notre outil, couplé à celle de Design Trotter pourrait permettre des travaux prometteurs. Citons deux exemples pour

lesquels notre outil pourrait être spécialisé, le choix de ces exemples est motivé par des collaborations actives du LESTER avec les laboratoires concernés.

Le premier exemple est celui de l'architecture reconfigurable Systolic Ring développée au laboratoire LIRMM de l'Université de Montpellier et présentée au premier chapitre de ce document. Cette architecture est tout à fait spécifique puisqu'elle met en œuvre une structure linéaire et circulaire de communications entre des éléments reconfigurables à gros grain de calcul (Dnode). Il serait intéressant dans le graphe de l'application de définir les parties du graphe qui forment un motif de calcul et qui doivent être programmées dans un unique Dnode. Ces derniers pouvant pour le moment contenir 8 instructions programmées, l'outil devrait donc pouvoir explorer le nombre d'instructions de chaque Dnode et le nombre de Dnode dans l'architecture, tout en prenant en compte les spécificités des ressources de communications de l'architecture.

Le deuxième exemple est celui de l'architecture reconfigurable aSoC développée au laboratoire VSPG de l'Université du Massachusetts aux USA. Cette architecture est présentée au premier chapitre de ce document, elle est composée de tuiles très hétérogènes reliées par un réseau point à point matriciel. Les différentes tuiles qui peuvent composer l'architecture peuvent être logicielles (cœur de processeur ou DSP) ou matérielles (IP, cœur reconfigurable de gros grain ou de grain fin). Actuellement un outil, AppMapper, est développé pour permettre la réalisation d'une application sur cette architecture. Mais il n'existe pas d'outil d'estimation permettant de rapidement faire le choix des tuiles de l'architecture. C'est dans ce sens que peut intervenir Design Trotter car il possède les outils suffisants pour orienter la conception de l'architecture. Certaines spécificités de l'architecture devront néanmoins être prises en compte telle que les interfaces de communications entre les tuiles.

La spécialisation de l'outil pour deux architectures ne doit pas nous empêcher de développer les caractéristiques intrinsèques de notre outil. En particulier la prise en compte du contrôle du système, le plus souvent géré par un processeur embarqué. Le modèle d'architecture doit évoluer dans ce sens (intégration de cœur de processeurs) afin de mieux caractériser l'interaction entre les parties logicielles et matérielles reconfigurables du système. Aussi le dynamisme de reconfiguration doit pouvoir être pris en compte dans nos estimations afin de mieux cibler les architectures qui seront capables de mettre en œuvre la reconfiguration dynamique.

L'automatisation de l'outil est aussi envisagée, nous sommes convaincus que l'introduction de décision par logique floue peut apporter une réponse intéressante. Effectivement la méthode d'exploration que nous avons établie s'appuie sur des connaissances techniques et expérimentales de l'utilisateur. Grâce à la logique floue, l'outil pourrait facilement évoluer avec les connaissances de l'utilisateur.

6.2.3 Ouverture

L'évolution de nos travaux sera liée aux développements futurs des architectures reconfigurables. Nous somme convaincus que les architectures reconfigurables de grain fin et de gros grain ont un avenir des plus prometteurs. Pour s'en convaincre, il suffit de voir la place de plus en plus importante qu'elles occupent dans les conférences et revues

internationales. Aussi en vue de leur plus grande utilisation un point semble devenir important : quel est le niveau de sécurité apporté par ces architectures contre l'espionnage industriel et le sabotage ?

Comme ce domaine est encore jeune et en pleine évolution, nous pouvons nous demander si il ne serait pas possible de concevoir des architectures qui dès le départ assureraient aux concepteurs un haut niveau de sécurité pour la conception. En effet, les architectures reconfigurables semblent être particulièrement intéressantes pour être utilisées dans les éléments de sécurité (sécurité des réseaux de communications par exemple) et de communications (radio logicielle), deux domaines sensibles qui demandent aux composants d'être particulièrement sûrs, très performants et flexibles. Il serait donc intéressant de réfléchir, grâce à nos travaux, à une méthode de conception d'architectures sécurisées. La sécurité aujourd'hui ne concerne pratiquement que les parties logicielles, qu'en sera-t-il demain pour les parties matérielles ?

Enfin pour clore ce mémoire, nous sommes convaincus de l'intérêt de nos travaux dans l'évolution des outils de conception pour les architectures reconfigurables. Nous n'avons pu répondre à toutes les questions levées par l'exploration de telles architectures, mais nous avons initié une première phase de recherche. Les architectures reconfigurables, particulièrement celles de gros grain, vont encore prendre de l'importance dans les années à venir. L'augmentation des applications nécessitant une flexibilité matérielle devrait être forte, en particulier pour les télécommunications. Les concepteurs ont besoin dès les premières phases de conception, comme par exemple pour l'application Matching Pursuit, d'outils et de méthodes leur permettant d'évaluer des choix architecturaux. Ainsi ils peuvent estimer des performances préliminaires alors que la spécification de l'application est encore mouvante. Explorer les architectures très tôt dans le flot de conception est la garantie de converger rapidement vers une architecture efficace.

Publications personnelles

Publications internationales

[Bossuet05] L. Bossuet, G. Gogniat, W. Burleson. *Dynamically Configurable Security for SRAM FPGA Bitstreams*. In Publication in International Journal of Embedded Systems, IJES, of From Inderscience Publishers.

[Bossuet04] L. Bossuet, G. Gogniat, W. Burleson. *Dynamically Configurable Security for SRAM FPGA Bitstreams*. 11th IEEE Reconfigurable Architectures Workshop, RAW 2004, Santa Fé, New Mexico, USA, 26-17 avril 2004.

[Bossuet03a] 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.

[Bossuet03b] L. Bossuet, G. Gogniat, J.L. Philippe. *Fast Design Space exploration Method for Reconfigurable Architectures*. In Proceedings of International Conference of Engineering of Reconfigurable Systems and Algorithms, ERSA'03, Las Vegas, Nevada, USA, June 23-26, 2003.

[Bilavarn04] S. Bilavarn, G. Gogniat, J.L. Philippe, L. Bossuet. *Low Complexity Design Space Exploration from Early Specifications*. In Revision Process in IEEE Transaction on Computer Aided Design, 2004.

[Bilavarn03] S. Bilavarn, G. Gogniat, J.L. Philippe, L. Bossuet. *Fast Prototyping of Reconfigurable Architectures From a C Program*. In IEEE International Symposium on Circuits and Systems, ISCAS'03, Bangkok, Thailand, 25-28 May, 2003.

[Bossuet03c] L. Bossuet, W. Burleson, G. Gogniat, V. Anand, A. Laffely, J.L. Philippe. *Targeting Tiled Architectures in Design Exploration*. In 10th Reconfigurable Architectures Workshop, RAW03, Workshop of IPDPS 03, Nice, France, April 22, 2003.

[Bossuet02a] L. Bossuet, G. Gogniat, J.P. Diguet, J.L. Philippe. *A Modeling Method for Reconfigurable Architectures*. In IEEE International Workshop on System-on-Chip for Real-Time Applications, IWSOC'02, Banff, Canada, July 6-7, 2002.

Publications nationales

[Bossuet02c] L. Bossuet, G. Gogniat, J.L. Philippe. *Flot d'xeploration des architectures reconfigurables*. Journée Francophones sur l'Adéquation Algorithme Architecture, JFAAA'02, Monastir, Tunisie, December 16-20, 2002.

[MACGTT02] Projet MACGTT. *Vers une approche unifiée pour la conception globale des terminaux de télécommunications*. Journée Francophones sur l'Adéquation Algorithme Architecture, JFAAA'2002, Monastir, Tunisie, December 16-20, 2002.

[Bossuet02d] L. Bossuet, G. Gogniat, J.L. Philippe. *Méthode d'estimation relative des performances des architectures de FPGA*. Colloque CAO, Paris, 15-16-17 mai 2002.

[Bossuet02b] L. Bossuet, Guy Gogniat. *Reconfigurable Architecture Modeling. The THF Model, Application to the aSoC Architecture*. Technical Report, Université de Bretagne Sud, Septembre 2002.

[Bossuet02e] L. Bossuet. *Exploration de l'espace de conception de l'architecture des circuits intégrés, ou comment construire une ville nouvelle ?*. Doctoriales de Bretagne 2002, Lorient, France, du 24 au 29 novembre 2002.

[Bossuet01] L. Bossuet. *Modélisation d'architectures reconfigurables embarquées*. Master Thesis, Université de Bretagne Sud, Lorient, Juin 2001.

Bibliographie

[Abnous96] A. Abnous and J. Rabaey. *Ultra-Low-Power Domain-Specific Multimedia Processors*. Proceedings of the IEEE VLSI Signal Processing Workshop, San Francisco, California, USA, October 1996.

[Amerijckx98a] C. Amerijckx, J.-D. Legat. *A Low-Power multiprocessor Architecture for Embedded Reconfigurable Systems*. International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS '98, Denmark, October 7-9, 1998.

[Altera04] Altera. *Stratix-II Device Handbook*. Technical Document, February 2003.

[Altera02] Altera. *Excalibur Devices Handbook*. Reference Manuel, November 2002.

[Altera01a] Altera SOPC Wold Seminar, 2001.

[Altera01b] Altera. *Apex 20K Programmable Logic Device Family*. Technical Document, April 2001.

[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.

[Atmel03] Atmel. *FPSLIC Series, Field Programmable Systeme Level Integrated Circuit*. Technical Document, November 2003.

[Auguin03] M. Auguin, K. Ben Chehida, J.P. Diguet, X. Fornari, A.M. Fouilliart, C. Gamrat, G. Gogniat, P. Kajfasz, Y Le Moullec. *Partitionning and CoDesign Tolls & Methodology for Reconfigurable Computing : the EPICURE Philosophy*. In Proceeding of the Third International Workshop on Systems, Architectures, Modeling Simulation, SAMOS 03, Samos, Greece, July 2003.

[Auguin01] M. Auguin, L. Capella, F. Cuesta, E. Gresset. *CODEF : a System Level Design Space Exploration Tool*. In IEEE International Conference on Acoustics, Speed and Signal Processing, volume 2, pages 1145-1148, 2001.

[Azzedine02] A. Azzedine, J-P. Diguet, J-L. Philippe. *Large exploration for hw/sw partioning of multirate and aperiodic real-time systems*. In International Symposium on Hardware/Software Codesign (CODES), Estate Park, USA, May 2002.

[Baghdadi00] A. Baghdadi, N. Zergainoh, W. Cesario, T. Roudier, A. Jerraya. *Design space exploration for hardware/software codesign of multiprocessor systems*. In 11th

international Workshop on Rapid System Prototyping, RSP, pages 8-13, Paris, France, June 21-23, 2000.

[Bakshi01] A. Bakshi, V.K. Prasanna, A. Ledeczi. *MILAN : A Model Based Integrated Simulation Framework for Design of Embedded Systems*. In Workshop on Languages, Compilers, and Tools for Embedded Systems, LCTES 2001. Snowbird, Utah, June 2001.

[Balasa95] F. Balasa, F. Cathoor, H. De Man. *Bakground Memory Area Estimation for Multidimensional Signal Processing Systems*. In IEEE Transactions on VLSI Systems, Vol. 3, No. 2, pp. 157-172, June 1995.

[Baumgarte01] V. Baumgarte, F. Mayr, A. Nückel, M. Vorbach and M. Weinhardt. PACT-XPP - *A Self-reconfigurable Data Processing Architecture*. In proceeding of the International Conference of Engineering of Reconfigurable Systems and Algorithms, ERSA 01, Las Vegas, Nevada, USA, June 2001.

[Benoit02] P. Benoit, L. Torres, M. Robert, G. Cambon, G. Sassatelli, T. Gil. *Caractérisation d'Architectures Reconfigurables. Un Exemple : Le Systolic Ring.* Journées Francophones Adéquation Algorithme Architecture, JFAAA'02, Monastir, Tunisie, 16-18 décembre 2002.

[Benoit03] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, G. Cambon. *Metrics for Reconfiguration Architectures Characterization : Remanece and Scalability*. In 10th Reconfigurable Architectures Workshop, RAW03, Workshop of IPDPS 03, Nice, France, April 22, 2003.

[Berry00] G. Berry. *The Esterel v5 Language Primer, Version v5-91*. Techical Paper, INRIA, Sophia-Antipolis, France, July 27, 2000.

[Betz00] V. Betz. *VPR and T-Vpack User's Manual (Version 4.30)*. Toronto University, March 27, 2000.

[Betz99] V. Betz, J. Rose, A. Marquart. *Architecture and CAD for Deep Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[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.

[Bilavarn03] S. Bilavarn, G. Gogniat, J.L. Philippe, L. Bossuet. *Fast Prototyping of Reconfigurable Architectures From a C Program*. In IEEE International Symposium on Circuits and Systems, ISCAS'03, Bangkok, Thailand, 25-28 May, 2003.

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

[Bjuréus02] P. Bjuréus, M. Millberg, A. Jantsch. *FPGA Resource and Timing Estimation from Matlab Execution Traces*. In International Symposium on Hardware/Software CoDesign, CODES 02, Estes Park, Colorado, USA, May 6-8, 2002.

[Blodget03] B. Blodget, P. James-Roxby, E. Keller, S. McMillan and P. Sundararajan. *A Self-reconfiguration Platform*. In proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL'03, Lisbon, Portugal, September 2003.

[Bondalapati02] K. Bondalapati, V. K. Prasanna. Reconfigurable Computing Systems. Invited Paper, Proceedings of the IEEE, July, 2002

[Bossuet04] L. Bossuet, G. Gogniat, W. Burleson. *Dynamically Configurable Security for SRAM FPGA Bitstreams*. 11th IEEE Reconfigurable Architectures Workshop, RAW 2004, Santa Fé, New Mexico, USA, 26-17 avril 2004.

[Bossuet03a] 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.

[Bossuet03b] L. Bossuet, G. Gogniat, J.L. Philippe. *Fast Design Space exploration Method for Reconfigurable Architectures*. In Proceedings of International Conference of Engineering of Reconfigurable Systems and Algorithms, ERSA'03, Las Vegas, Nevada, USA, June 23-26, 2003.

[Bossuet03c] L. Bossuet, W. Burleson, G. Gogniat, V. Anand, A. Laffely, J.L. Philippe. *Targeting Tiled Architectures in Design Exploration*. In 10th Reconfigurable Architectures Workshop, RAW03, Workshop of IPDPS 03, Nice, France, April 22, 2003.

[Bossuet02a] L. Bossuet, G. Gogniat, J.P. Diguet, J.L. Philippe. *A Modeling Method for Reconfigurable Architectures*. In IEEE International Workshop on System-on-Chip for Real-Time Applications, IWSOC'02, Banff, Canada, July 6-7, 2002.

[Bossuet02b] L. Bossuet, Guy Gogniat. *Reconfigurable Architecture Modeling. The THF Model, Application to the aSoC Architecture*. Technical Report, Université de Bretagne Sud, Lorient, Septembre 2002.

[Bossuet02e] L. Bossuet. *Exploration de l'espace de conception de l'architecture des circuits intégrés, ou comment construire une ville nouvelle ?*. Doctoriales de Bretagne 2002, Lorient, France, du 24 au 29 novembre 2002.

[Bossuet01] L. Bossuet. *Modélisation d'architectures reconfigurables embarquées*. Master Thesis, Université de Bretagne Sud, Lorient, Juin 2001.

[Burleson01a] W. Burleson, R. Tessier, D. Goeckel, S Swaminathan, P. Jain, J Euh, S. Venkatraman, V. Thyagarajan. *Dynamically Parameterized Algorithms and Architectures to Exploit Signal Variations for Improved Performance and Reduced Power*. In International Conference on Acoustic, Speech, and Signal Processing, ICASSP 01, 2001.

[Burleson01b] W. Burleson, P. Jain, S. Venkatraman. *Dynamically Parameterized Architecture for Power-Aware Video Coding Motion Estimation and DCT*. Second USF International Workshop on Digital Computational Video, DCV 01, 2001.

[Butts02] M. Buts. *Survey of Nanoscale Digital System Technology*. In IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM O2, April 24, 2002.

[Bruni01] D. Bruni, A. B. L. Benini. *Statistical Design Space Exploration for Application Specific Unit Synthesis*. In 38th Design Automation Conference, DAC 01, pp. 641-646, 2001.

[Cambonie03] J. Cambonie, ST Microelectronics. *Reconfigurable Architecture for WLAN Application*. Séminaire POMMARD, ENST Paris, December 2003.

[Celoxica03] Celoxica. *DK2 - Handel-C*. Language Reference Manual. 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.

[Chow99a] Paul Chow, Soon Ong Seo, Jonathan Rose, Gerard Paez-Monzon and Immanuel Rahardja. *The Design of an SRAM-Based Field-Programmable Gate Array - Part I: Circuit Design and Layout*. IEEE Transactions on VLSI Systems, June 1999 , Vol. 7, No.2.

[Chow99b] Paul Chow, Soon Ong Seo, Jonathan Rose, Gerard Paez-Monzon and Immanuel Rahardja. *The Design of an SRAM-Based Field-Programmable Gate Array - Part II*: Architecture. IEEE Transactions on VLSI Systems, Septembre 1999 , Vol. 7, No.3.

[Clifford00] P. Clifford, S. J.E.Wilton. *Architecture of Cluster-Based FPGAs with Memory*. IEEE Custom Integrated Circuits Conference, CICC 00, May 2000.

[Compton99] Katherine Compton. *Programming Architectures For Run-Time Reconfigurable Systems*. Master's Thesis, Dept of ECE, Northwestern University, Evaston, IL USA. December 1999

[Compton00] Katherine Compton and Scott Hauck. *Configuration Caching Techniques for FPGA*. IEEE Symposium on FPGAs for Custom Computing Machines , FCCM 2000.

[C.T.I. 98] C.T.I. COMETE (CENT, LIRMM, TIMA, IRESTE, IRISA, LAMI). *CODESIGN, conception conjointe logiciel-matériel*. Collection technique et scientifique des télécommunications (CTST), édition Eyrolles, Juin 1998.

[Dandalis00] A. Dandalis, K. Prasanna, J. D. P. Rolim. *A Compartive Study of Performances of the AES Final Candidates Using FPGA*. Workshop on Cryptographic Hardware and Embedded Systems, August 2000.

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

[deDinechin99] F. de Dinechin. *Le prix du routage dans les FPGAs*. 5ème Symposium sur les Architectures nouvelles de machines (SympA'5).Rennes, Juin 1999.

[Dehon99] A. Dehon. *Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you dont't really want 100% LUT utilization)*. In Proceedings of the International Symposium of Field Programmable Gate Array, FPGA 1999, pp. 125-134, Monterey, Californie, USA, February 1999.

[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.

[Delahaye03] J-P Delahaye. *Systèmes Radio Dynamiquement Reconfigurables sur des Architectures Hétérogènes*. Master Thesis, Université de Paris XI, Faculté d'Orsay. Spetember 2003.

[Demigny02] D. Demigny, N. Boudouani, O. Sentieys, and D. Chillet. *La rémanence des architectures reconfigurables dynamiquement*. In 7ème SYMPosium en Architectures nouvelles de machines, SYMPA'7, pages 23-32, Paris, France, Avril 2001.

[Diguet00] J.P. Diguet, G. Gogniat, P. Danielo , M. Auguin, J.L. Philippe. *The SPF model*. Forum on Design Langage, FDL 00, Tübingen, Germany, 2000.

[Ebeling96] C. Ebeling, D. C. Cronquist, P. Franklin. *RaPiD - Reconfigurable Pipelined Datapath*. In International Workshop on Field Programmable Logic and Applications, FPL'96. Darmstadt, Germany, September 23 - 25, 1996.

[Elbirt00] A. J. Elbirt, W. Yip, B. Chetwynd, C. Paar. *An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists*. In proceeding of the third Advanced Encryption Standard candidate conference, AES3, New York, New York, USA, April 12-14, 2000.

[Elleouet03] D. Elleouet. *Caractérisation et modélisation de la consommation de puissance des mémoires sur FPGA*. Master Thesis, Université de Bretagne Sud, Lorient, Septembre 2003.

[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.

[Fauth95] A. Fauth, J. Van Praet, M. Freericks. *Describing Instruction Set Processors Using nML*. In European Design and Test Conference, ED&TC, pp. 503-507, March 1995.

[Gaj00] K. Gaj, P. Chodowiec. *Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware*. In proceeding of the third Advanced Encryption Standard candidate conference, AES3, New York, New York, USA, April 12-14, 2000.

[Garcia00] A. Garcia. *Estimation et optimisation de la consommation de puissance des circuits logiques programmables du type FPGA*. PhD Thesis, Ecole Nationale Supérieure des Télécommunications, Paris, Août 2000.

[Garcia99] A. Garcia, W. Burleson, J-L. Danger. *Power Modelling in Field Programmable Gate Arrays (FPGA)*. In Proceeding of the 9th International Workshop on Field Programmable Logic and Applications, FPL 1999, Glasgow, Scotland, 1999.

[George99] V. George, H. Zhang, J. Rabaey. *The Design of a Low Energy FPGA*. In Proc. Int. Symp. On Low Power Electronics and Design, ISLPED 1999, pages 188-193, 1999.

[Goldstein99] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, R. Laufer. *PipeRench : A Coprocessor for Streaming Multimedia Acceleration*. 26th International Symposium on Computer Architecture, ISCA 99, Atlanta, Georgia, USA, May 2-4, 1999.

[Gouyen03] C. Gouyen, L. Lagadec. *Outils génériques pour le reconfigurable : Application à deux architectures commerciales*. RenPar'15/CFSE'3/SympAAA 2003, Le Colle sur Loup, France, October 15-17, 2003.

[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.

[Hadjiyiannis97] G. Hadjiyiannis, S. Hanono, S. Devadas. *ISDL : An Instruction Set Description Language for Retargetability*. In Design Automation Conference, DAC 97, Anaheim, California, USA, 1997.

[Hartenstein03] R. Hartenstein. *Are we Realy Ready for the Breakthrough ?*. In 10th Reconfigurable Architectures Workshop, RAW 03, Workshop of IPDPS 03, Nice, France, April 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.

[Hartenstein00] R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger. *KressArray Xplorer : A New CAD Environment to Optimize reconfigurable Datapath Array Architectures*. In 5th Asia and South Pacific Design Automation Conference, ASP-DAC 00, Pacifico Yokohama, Yokohama, Japan, January 25-28, 2000.

[Hartenstein95] R. Hartenstein, R. Kress. *A Datapath Synthesis System for the Reconfigurable Datapath Architecture*. In Asia and South Pacific Design Automation Conference, ASP-DAC 95, Nippon convention Center, Makuhari, Chiba, Japan, August 29 –Spetember 1, 1995.

[Hauck98] Scott Hauck. *The Future of Reconfigurable Systems*. 5th Canadian Conference on Field Programmable Devices, Montreal, June 1999

[Hauck97] S. Hauck, T.W. Fry, M.M. Hosler, J.P. Kao. *The Chimarea Reconfigurable Functional Unit*. In Proceeding of the IEEE symposium on FPGAs for Custom Computing Machines, FCCM 97, pp. 87-96, 1997.

[Haulbert03] C.Haulbert, J. Teich. *Accelerating Design Space Exploration*. In Proceedins of 5th IEEE International Conference on ASIC, ASICON 03, Beijing, China, October 21-24, 2003.

[Hauser00] J. R. Hauser. *Augmenting a Microprocessor with Reconfigurable Hardware*. PhD Thesis, University of Californie, Berkeley, USA, 2000.

[Hauser97] J. R. Hauser, J. Wawrzynek. *Garp : A MIPS Processor with a Reconfigurable Coprocessor*. The 5th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 96, Napa, CA, USA, April 16 - 18, 1997.

[Havinga00] P. J.M. Havinga and G. J.M. Smit. *Design Techniques for Low Power Systems*. Journal of Systems Architecture Vol 46. Iss 1, 2000.

[Helaihel97] R. Helaihel and K. Olukotum. *Java as a Specification Language for Hardware-Software Systems*. In Proceeding of International Conference on Computer Aided Design, ICCAD 97, pp. 690-697, November 1997.

[Heysters00] P. M. Heysters, J. Smit, G. J.M. Smit, P. J.M. Havinga. *Mapping of DSP Algorithms on Field Programmable Function Arrays*. In International Workshop of Field Programmable Logic and Application, FPL 00, Austria, August 2000.

[Jerraya99] A. Jerraya at al. .*Multilanguage Specification for System Design and CoDesign*. Chapter in « System-Level Synthesis », NATO ASI 1998 edited by A. Jerraya and J. Mermet, Kluwer Academic Publishers, 1999.

[Kandel92] A. Kandel. *Fuzzy Expert Systems*. CRC Press, Boca Raton, F1, USA, 1992.

[Kathail02] V. Kathail, S. Aditya, R Schreiber, B. R. Rau, D. Cronquist, M. Sivaraman. *PICO : Automatically Designing Custom Computers*. In IEEE Computer, n° 35 vol 9, pp. 39-47, September 2002.

[Kessal00] L. Kessal, D. Demigny, R. Bourguiba, N. Boudouani. *Architecture reconfigurable méthodologie et modélisation VHDL pour la mise au point d'applications*. 6ème Symposium en Architectures Nouvelles de Machines, SympA'6, Besançon, France, 19-22 juin 2000.

[Kienhuis97] B. Kienhuis, E. Deprette, K. Vissiers, P. van der Wolf. *An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures*. In Application Specific Systems, Architectures and Processors, ASAP 03, July 2003.

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

[Kress95] R. Kress et al. *A Datapath Synthesis System for the Reconfigurable Datapath Architecture*. In Asia and South Pacific Design Conference, Asp-DAC 95. Chiba, Japan, Auguste 29 –Sepetember 1, 1995.

[Krusse98] E. Krusse, J. M. Rabaey. *Low-Energy Embedded FPGA Structures*. In Proceeding of the International symposium on Low Power Electronics and Design, pp. 155-160, Monterey, California, USA, August 1998.

[Kulkarni02] D. Kulkarni, W. A. Najjar, R. Rinker, F. J. Kurdahi. *Fast Area Estimation to Support Compiler Optimizations in FPGA-based Reconfigurable Systems*. In IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 02, Napa, California, USA, April 21-24, 2002.

[Laffely03a] A. Laffely, J. Liang, R. Tessier, W. Burleson. *Adaptative System on a Chip (aSoc) : A Backbone for Power-Aware Signal Processing Cores*. In IEEE International Conference on Image Processing, ICIP 03. Barcelona, Spain, September 14-17, 2003.

[Laffely03b] A. Laffely, J. Liang, R. Tessier, C. A. Moritz, W. Burleson. *Power-Aware System on a Chip*. In Boston Area Architecture Workshop. Boston, Massachusetts, USA, January, 2003.

[Laffely01] A. Laffely, J. Liang, P. Jain, N. Weng, W. Burleson, R. Tessier. *Adaptative System on a Chip (aSoc) for Low-Power Signal Processing*. Thirty-Fith Asilomar Conference on Signals, and Computers, Nov 4-7, 2001.

[Lagadec00] Loïc Lagadec. *Abstraction, modélisation et outils de CAO pour les circuits intégrés reconfigurables*. PhD Thesis, Université de Rennes1, 2000.

[Lagadec01] Loïc Lagadec. *Outils génériques pour les architectures reconfigurables*. 7ème Symposium sur les Architectures nouvelles de machines (SympA'7), Paris, Avril 2001.

[Lahiri01] K. Lahiri, A. Raghunathan, S. Dey. *Evaluation of the Traffic-Performance Characteristics of System_onChip Communication Architectures*. In International Conference on VLSI Design, VLSI 01, pp. 21-35, January 2001.

[Lahiri00a] K. Lahiri, A. Raghunathan, S. Dey. *Performance Analysis of Systems With Multi-Channel Communication Architectures*. In 13th International Conference on VLSI Design, pp. 530-537, Calcutta, India, January 2000.

[Lahiri00b] K. Lahiri, A. Raghunathan, S. Dey. *Efficient Exploration of the SoC Communication Architecture Design Space*. In International Conference on Computer Aided Design, ICCAD 00, pp. 424-430, November 2000.

[Laticce04] *ispXPGA Family*. Data Sheet, January 2004.

[Lee61] C. Y. Lee. *An Algorithm for Path Connections and its Applications*. IRE Transactions on Electronic Computers, 10 : 346-365, 1961.

[LeMoullec03a] 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.

[LeMoullec03b] Y. Le Moullec, N. Ben Amor, J.P. Diguet, J.L. Philippe and M. Abid. *Multi-granularity Metrics for the Era of Strongly Personalized SoCs*. In Design Automation and Test in Europe, DATE 2003, Munich, Germany, March 3-7, 2003.

[LeMoullec03c] Y. Le Moullec, P. Koch, J.P. Diguet, J.L. Philippe. *Design Trotter : Building and Selecting Architectures for Embedded Multimedia Applications*. In IEEE International Symposium on Consumer Electronics, ISCE 03, Sydney, Australia, December 3-5, 2003.

[LeMoullec03d] Y. Le Moullec, D. Heller, J.P. Diguet, J.L. Philippe. *Estimation du parallélisme au niveau système pour l'exploration de l'espace de conception de systèmes enfouis*. Technique et Science Informatiques, RSTI-TSI, Vol. 22, n°3/2003, pp. 315-349, Lavoisier Hermes-Science publications.

[Leventis98] Paul Leventis. *Using edif2blif Version 1.0*. Toronto University, June 30, 1998.

[Liang02] J. Liang, A. Laffely, S. Swaminathan, R. Tessier. *An Architecture for Saclable On-Chip Communication*. Technical Report, University of Massachusetts, Amherts, USA, September 2002.

[Liang00] J. Liang, S. Swaminathan, R. Tessier. *aSoC : A Saclable, Single-Chip Communications Architecture*. In the IEEE International. Conference on Parallel Architectures and Compilation Technique, Philadelphia, USA. October 2000.

[Lockwood03] J. W. Lockwood, C. Neely, C. Zuver, J. Moscola, S. Dharmapurikar, D. Lim. *An Extensible, System-on-Programmable-Chip, Content-Aware Internet Firewall*. In Proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL'2003, Lisbon, Portugal, September 2003.

[Marshall99] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, B. Hutchings. *A Reconfigurable Arithmetic Array for Multimedia Applications*. In ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, FPGA'99, Monterey, California, USA February 21-23, 1999.

[McMurchie95] L. McMurchie, C. Ebeling. Pathfinder : *A Negotiation_based Performance-driven Router for FPGAs*. In FPGA, pp. 111-117, February 1995.

[Messé99] V. Messé. *Production de compilateurs flexibles pour la conception de processeurs programmables spécialisés*. Ph. D. Thesis, Université de Rennes, March 1999.

[Mirsky96] E. Mirsky, A. DeHon. *MATRIX : A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Ressources*. In IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'96, Napa, California, USA, April 17-19, 1996.

[Miyamori98] T. Moyamori, K. Oluktun. *REMARC : Reconfigurable Multimedia Array Coprocessor*. In ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'98, Monterey, California, USA, February 1998.

[Model03] Model Technology. *ModelSim, Tutorial, Version 5.8*. December 16, 2003.

[Mohanty02] S. Mohanty, V.K. Prasanna, S. Neema, J. Davis. *Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-*

Granular Simulation. In Workshop on Languages, Compilers, and Tools for Embedded Systems, LCTES 2002. Berlin, Germany, June 2002.

[Moritz98] C. A. Moritz, D. Yeung, A. Agarwal. *Exploring Optimal Cost-Performance Designs for Raw Microprocessors*. Proceedings of the International IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 98, April 1998.

[Mudge01] T. Mudge. *Power : A First-Class Architectural Design Constraint*. IEEE Computer, Volume 34, pp. 52-57, April 2001.

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

[Nayak02] A. Nayak, M. Haldar, A. Choudhary, P. Banerjee. *Accurate Area and Delay Estimators for FPGAs*. In Design and Test in Europe, DATE 02, Paris, France, March 4-8, 2002.

[Pareto1896] V. Pareto. *Cours d'Economie Politique*. F. Rouge, Lausanne, 1896.

[Pees00] S. Pees, A. Hoffman, H. Meyr. *Retargeting of Compiled Simulators for Digital Signal Processors Using a Machine Description Langage*. In Design, Automation and Test in Europe Confrence, DATE 00, pp 669-938, Paris, France, March 27-30, 2000.

[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.

[Poon02a] K. K. W. Poon, A. Yan, S. J.E.Wilton. *A Flexible Power Model for FPGAs*. In Proceeding of the 12th International Conference on Field-Programmable Logic and Applications, FPL 2002. Montpellier, France, September 2002.

[Poon02b] K. K. W. Poon. *Power Estimation for Field Programmable Gate Arrays*. Master's Thesis, University of British Columbia, Vancouver, Canada, August 1999.

[Rabaey00] J. M. Rabaey. *Silicon Platforms for the Next Generation Wireless Systems - What Role does Reconfigurable Hardware Play* ? In International Workshop of Field Programmable Logic and Application, FPL '00, Austria, August 2000.

[Radumovic98] B. Radunovic, V. Milutinovic. *A Survey of Reconfigurable Computing Architectures*. In Field-Programmable Logic and Applications, R.W. Hartenstein and A. Keevallik (editorts), LCNS 1482, Springer, 1998, pp. 376-385.

[Ramtron] http://www.ramtron.com

[Riordan] M. Riordan. *The Lost History of the Transistor*. In IEEE Spectrum, pp36-41, May 2004.

[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.

[Salefski01] B. Salefski, L. Caglar. *Re-Configurable Computing in Wirless*. In 38th Design Automation Conference, DAC 2001, Louisana, USA, June, 2001.

[Sassatelli01] G. Sassatelli, L. Torres, G. Cambon, J. Galy. *Architectures Reconfigurables Dynamiquement pour les Systèmes Embarqués*. In GRESTI'01, Toulouse, France, Septembre 2001.

[Sassatelli02a] G. Sassatelli. *Architectures Reconfigurables Dynamiquement pour les Systèmes sur Puce*. Ph.D. Thesis, Université de Montpellier, France, April 2002.

[Sassatelli02b] G. Sassatelli, L. Torres, P. Benoit, T. Gil, C. Diou, G. Cambon, J. Galy. *Highly Scalable Dynamically Reconfigurable Systolic Ring-Architecture for DSP Applications*. In IEEE Design Automation and Test in Europe, DATE 02, pp. 553-557, Paris, France, March 2002.

[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.

[Sidsa02] Sidsa. *FIPSOC Mixed Signal System-on-Chip*. Technical document, 2002.

[Singh00] H. Singh, M.H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. Filho. *MorphoSys : an integrated reconfigurable system for data-parallel and computation-intensive applications*. IEEE Trans. On Computers 49, pp. 465-481, 2000.

[Shang02] L. Shang, A. S. Kaviani, K. Bathala. *Dynamic Power in Virtex™-II FPGA Family*. In ACM/SIGDA Symposium of Field Programmable Gate Arrays, FPGA'02, Monterey, California, USA, February 24-26, 2002.

[So03] B. So, P. C. Diniz, M. W. Hall. *Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Spcae Exploration*. In 40th Design Automation Conference, DAC 03. Anaheim, California, USA, June 2-6, 2003.

[So02] B. So, M. W. Hall, P. C. Diniz. *A Compliler Approach to Fasst Hardware Deigsn Space Exploration in FPGA-based Systems*. In Proceeding of the ACM Conference on Programming Language Deisgn and Implementation, June 2002.

[Systolix00] Systolix. *PulseDSP : Synthesisable DSP Co-processor*. Technical Document, 2000.

[Tessier01] R. Tessier, W. Burleson. *Reconfigurable Computing for Digital Signal Processing : A Survey*. Journal of VLSI Signal Processing 28, 7-27, 2001.

[Tredennick03] N. Tredennick, B. Shimamoto. *The Rise of Reconfigurable Systems*. In proceeding of Engineering of Reconfigurable Systems and Application, ERSA 03. June 23-26, 2003, Las Vegas, Nevada, USA.

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

[Triscend01] Triscend Corporation. *Triscend E5 Configurable Sysytem-on-Chip Platform*. Product Description, 2001.

[Triscend02] Triscend Corporation. *Triscend A7S Configurable Sysytem-on-Chip Platform*. Product Description, 2002.

[Ulmann04] M. Ulmann, M. Hübner, B. Grimm, J. Becker. *An FPGA Run-Time System for Dynamical On-Demand Reconfiguration*. 11th IEEE Reconfigurable Architectures Workshop, RAW 2004, Santa Fé, New Mexico, USA, 26-17 avril 2004.

[Verdoscia96] L. Verdoscia. *ALFA fine grain dataflow machine*. International Programming, M.A. Orgun and E.A. Ashcroft edition, 1996.

[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.

[Weaver00] N. Weaver, J. Wawrzynek. *A Comparison of the AES Candidates Amenability to FPGA Implementation*. In proceeding of the third Advanced Encryption Standard candidate conference, AES3. April 12-14, 2000, New York, New York, USA.

[Wollinger03] T. Wollinger, C. Paar. *How Secure Are FPGAs in Cryptographic Applications*. In proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL 03, Lisbon, Portugal, September 2003.

[Wilton97] S. J.E. Wilton. *Architectures and Algorithms for Field Programmable Gate Arrays with Embedded Memory*. Ph.D Thesis, University of Toronto, Canada, 1997.

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

[Witting97] R.D. Witting and P. Chow. *One Chip : An FPGA Processor with Reconfigurable Logic*. In Proceedings of the IEEE Symposium on FPGA for Custom Computing Machines, FCCM 96, pp. 126-135, 1996.

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

[Xilinx04b] Xilinx. *Software Manual and Documentation for ISE6.1i* Technical Document, 2004.

[Xilinx03a] Xilinx. *Revolutionary for the Next Generation Platform FPGAs*. December 8, 2003.

[Xilinx03b] Xilinx. *Virtex-II Platform FPGAs*. Technical Document, August 2003.

[Xilinx02] Xilinx. *Micro Blaze RISC 32 Bits Soft Processor*. Technical Document, August 2002.

[Yeung94] D. Yeung, W. J. Dally, A. Agarwal. *How to Choose the Grain Size of a Parallel Computer*. MIT/LCS Technical Report, MIT-LCS-TR-739, February 1994.

[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.

[Zissulecu03] C. Zissulescu, T. Stefanov, B. Kienhais, E. Deprettere. *Laura : Leiden Architecture Research and Exploration Tool*. In the 13th International Conference Field Programmable Logic and Applications, FPL'03, Lisbon, Portugal, September 1-3, 2003.

[Zivkovic03] V. D. Zivkovic, P. van der Wolf. *From High Level Application Specification to System-Level Architecture, Definition: Exploration, Design and Compilation*. In Proceeding of the International Workshop on Compilers for Parallel Computers, CPC 03, pp. 39-49, Amsterdam, the Netherlands, January 8-10, 2003.

Lexique

ACG : **A**verage **C**ommunication **G**raph (terme LESTER)

AES : **A**dvance **E**nrcyption **S**ystem

ALM : **A**daptive **L**ogic **M**odule (terme Altera)

ALU : **A**rithmetic and **L**ogic **U**nit

AR : **A**rchitecture **R**econfigurable

ASIC : **A**pplication **S**pecific **I**ntegrated **C**ircuit

aSoC : **a**daptative **S**ystem on a **C**hip (terme Université du Massachusetts)

CISC : **C**omplex **I**nstruction **S**et **C**omputer

CLB : **C**onfigurable **L**ogic **B**lock (terme Xilinx)

DRAM : **D**ynamic **RAM**

DPR : Data**P**ath **R**econfigurable (terme LASTI-ENSAT)

DSP : **D**igital **S**ignal **P**rocessor

FPGA : **F**ield **P**rogrammable **G**ate **A**rray

FRAM : **F**erroelectric **RAM**

GPP : **G**eneral **P**urpose **P**rocessor

HCDFG : **H**ierarchical **C**ontrol **D**ata **F**low **G**raph (terme LESTER)

IP : **I**ntellectual **P**roperty

LAB : **L**ogic **A**rray **B**lock (terme Altera)

LE : **L**ogic **E**lement (terme Altera)

LUT : **L**ook **U**p **T**able

MAC : **M**ultiplieur **Ac**cumulateur

MRAM : **M**agnetoresistive **RAM**

MOPS : **M**illion d'**O**pération **P**ar **S**econde

OUM : **O**vonic **U**nified **M**emory

SoC : **S**ystem **o**n a **C**hip

rDPA : **r**econfigurable **D**ata**P**ath **R**econfigurable (terme Université Kaiserlautern)

rDPU : **r**econfigurable **D**ata**P**ath **U**nit (terme Université Kaiserlautern)

RSoC : **R**econfigurable **S**ystem **o**n a **C**hip

RAM : **R**andom **A**cess **M**emory

ROM : **R**ead **O**nly **M**emory

SRAM : **S**tatic **RAM**