Du micro-processeur au circuit FPGA

une analyse sous l'angle de la reconfiguration

Raphaël David* — **Dominique Lavenier**** — **Sébastien Pillement***

* *LASTI - ENSSAT / INRIA - R2D2*
6, rue de Kérampont
F-22300 Lanion
Raphael.David@enssat.fr, Sebastien.Pillement@enssat.fr

** *IRISA - CNRS*
Campus de Beaulieu
F-35042 Rennes
Dominique.Lavenier@irisa.fr

RÉSUMÉ.

ABSTRACT.

MOTS-CLÉS : *micro-processeur, architectures reconfigurables, circuit FPGA, reconfiguration, reconfigurable computing, système reconfigurable*

KEYWORDS: *microprocessor, reconfigurable architectures, FPGA, reconfiguration, reconfigurable computing, reconfigurable systems*

2e soumission à *Technique et science informatiques*, le 15 juillet 2003.

1. Introduction

Les micro-processeurs et les circuits FPGA partagent la faculté de s'adapter, par programmation, à un traitement numérique particulier. La manière de les programmer est certes différente, mais ils ont en commun de posséder un support (une mémoire) leur permettant de stocker temporairement la spécification de la tâche à accomplir. Du contenu de cette mémoire dépend la nature du traitement.

Les micro-processeurs modernes, issus du modèle Von Neuman, sont plus enclins à l'exécution séquentielle d'un traitement, même s'ils intègrent une batterie d'unités fonctionnelles et de longs pipelines pour extraire au mieux le parallélisme présent au niveau des instructions. A l'opposé, les circuits FPGA présentent un modèle d'exécution spatial répartissant le traitement sur un assemblage d'unités fonctionnelles spécifique à ce traitement. Dans le premier cas, chaque cycle machine spécifie quelles sont les unités fonctionnelles actives, quels types d'opérations doivent être effectués, et comment les données transitent des espaces de stockages aux unités fonctionnelles. Dans le second cas, pour un traitement donné, l'architecture est figée : les données sont traitées suivant un schéma pré-établi et transitent via un réseau déterminé d'opérateurs.

Entre ces deux extrêmes, est apparue, peu après la mise sur le marché des composants FPGA par la société Xilinx en 1985, une classe d'architectures baptisée plus ou moins consensuellement *architecture reconfigurable*. Au départ fortement lié à la technologie FPGA, le concept a évolué vers un statut autonome, à la fois motivé par une recherche intense et par les progrès prodigieux des technologies d'intégration. On parle aujourd'hui de calcul reconfigurable (ou *reconfigurable computing*) pour faire référence à ces architectures et aux modèles d'exécution associés.

L'argumentaire classique est que le concept d'architecture reconfigurable allie à la fois les avantages des circuits FPGA, voire des circuits VLSI, (rapidité obtenue par une spécification spatiale des algorithmes) et des processeurs (flexibilité de la mise en oeuvre par programmation des ressources). On reconnaît néanmoins une difficulté de programmation évidente par rapport aux processeurs. Cette dichotomie grossière fait cependant abstraction de toutes les richesses et des potentialités propres au calcul reconfigurable : du fameux *bitstream* des circuits FPGA jusqu'aux jeux d'instructions RISC des micro-processeurs, il existe une multitude d'alternatives architecturales pour programmer sur silicium un traitement numérique. Les avantages, tout comme les inconvénients, sont multiples et ne s'évaluent pas par rapport à un critère unique ; ils se mesurent plutôt par rapport à une variété de caractéristiques telles que la flexibilité, le coût, la consommation, les performances, les domaines d'applications, les outils de programmation, etc.

Dans cette article nous analysons les circuits programmables sous l'angle de leur flexibilité. Celle-ci est mesurée par le potentiel de leur **reconfiguration**, c'est à dire leur capacité à adapter leurs ressources en fonction d'un traitement. Il convient donc, dès lors, de préciser ce que nous entendons par reconfiguration. Une architecture matérielle est généralement constituée d'une collection d'éléments de calcul et de sto-

ckage, d'un dispositif d'échange d'information entre ces éléments et, éventuellement, d'un contrôleur pour gérer l'ensemble. Une configuration précise à la fois la fonctionnalité des éléments et les chemins de données entre ces éléments, voire le contrôle. Le terme *reconfiguration* apporte, quant à lui, une notion supplémentaire de dynamicité. Il exprime le fait qu'une configuration n'est pas fixée définitivement (comme pour les circuits ASIC), mais qu'il existe un mécanisme qui, au cours du temps, puisse modifier à la fois la fonctionnalité des éléments de calcul et leur interconnexion.

Ainsi, un micro-processeur possède une capacité de reconfiguration dictée par une liste pré-définie et bornée d'opérations arithmétiques et logiques, et par un choix restreint de transferts de données entre opérateurs, registres et mémoires. Une configuration étant réduite, elle peut alors être codée de manière concise et mise en oeuvre extrêmement rapidement, e.g. à chaque cycle machine. Typiquement, elle tient sur 32 bits et représente le jeu d'instructions d'un processeur. À l'autre extrémité, un circuit FPGA offre une liberté totale sur le choix des opérateurs, leur nombre, et leur interconnexion. Mais le prix est dans la description de cette organisation puisqu'il faut, pour simplifier, tout spécifier à partir d'équations booléennes. La taille des *bitstream* des derniers composants FPGA peut atteindre plusieurs millions de bits, ce qui, de fait, interdit une reconfiguration fréquente du composant. Les architectures reconfigurables, quant à elles, essaient de trouver un juste milieu pour autoriser plus de flexibilité tout en limitant les pénalités dues à la reconfiguration. Les recherches dans ce domaine visent essentiellement à déterminer cet équilibre en explorant ce vaste espace de conception.

L'objectif de cet article est de balayer cet espace avec, comme critère principal, la reconfiguration. Ce critère sert de point de repère et, dans une certaine mesure, de critère de classification pour comparer la malléabilité des architectures et leur capacité à résoudre efficacement un traitement donné. Cette efficacité est principalement évaluée dans cette étude sous l'angle des performances et de la consommation énergétique.

La suite de l'article est structurée de la manière suivante : la section 2 analyse la flexibilité des circuits programmables en fonction de leur potentiel de reconfiguration et de la complexité de leur réseau d'interconnexions. La section 3 passe en revue les différentes architectures des circuits programmables, des processeurs jusqu'aux circuits FPGA. Elle examine à partir du critère de reconfiguration, les ressources de calcul, le réseau d'interconnexion et leur usage. La section **??**, quant à elle, évalue l'efficacité énergétique et les performances des différents paradigmes d'exécution. Enfin, la section **??** conclue l'article.

2. Flexibilité des circuits programmables

Dans cette section, la flexibilité des circuits programmables est analysée par rapport à deux critères : le potentiel de reconfiguration des ressources de calcul et la structure du réseau d'interconnexion. Ce sont deux points essentiels car l'adaptation

des ressources matérielles à un traitement donné repose à la fois sur la versatilité des opérateurs de base et leur aptitude à s'agencer dans l'espace.

2.1. *Le potentiel de reconfiguration*

Techniquement, la granularité de reconfiguration est un indice concret sur lequel on peut s'appuyer pour envisager une première analyse [RAB 97]. Ainsi, trois niveaux de reconfiguration sont-ils proposés : système, fonctionnel et logique. Ils reflètent directement les trois types d'architectures déjà mentionnés : les micro-processeurs, les architectures reconfigurables et les circuits FPGA.

2.1.1. *Niveau système*

Typiquement, ce niveau de reconfiguration est celui des processeurs programmables. Il concerne essentiellement la fonctionnalité des unités de calcul et l'orientation du chemin de données (de/vers la mémoire et/ou les registres). Pour être efficace, la quantité de données à lire en mémoire d'instructions doit être faible, ce qui induit des possibilités d'optimisation relativement limitées, notamment lorsque le traitement possède un fort potentiel de parallélisation.

Ces architectures sont extrêmement flexibles et maintiennent un même modèle de programmation quelle que soit l'application à exécuter. Par exemple, un processeur RISC (Reduce Instruction Set Computer) traite exclusivement des opérations registre-registre, un processeur de traitement du signal (DSP : Digital Signal Processor) des instructions mémoire-mémoire, etc. Dans la mesure où le modèle de calcul est invariant, cette limitation est transparente et le passage d'une application à une autre n'entraîne aucune perte d'efficacité.

Ici, chaque cycle machine entraîne une nouvelle configuration du matériel : schématiquement, une instruction est décodée pour activer certaines unités fonctionnelles et indiquer les transferts avec les unités de stockage (registres ou mémoire). La reconfiguration joue à la fois sur les unités fonctionnelles (effectuer une addition ou une soustraction, un ET ou un OU logique, etc.) et sur l'aiguillage des données via les bancs de registres.

2.1.2. *Niveau fonctionnel*

Pour augmenter le potentiel d'optimisation des processeurs standards, la rigidité des interconnexions des ressources (opérateurs arithmétiques ou logiques, ressources de mémorisation, générateur d'adresses, etc.) peut être assouplie. On définit alors un nouveau grain dit fonctionnel ou flot de données. Dans ce cas, les distributions du contrôle et des données sont réalisées via des chemins différents. Dans la mesure où ces distributions sont réalisées concurremment, la redirection des connexions peut être dynamique dès lors que la quantité de données à transmettre est réduite.

Par rapport au niveau précédent, le maintient d'un même modèle de programmation est ici beaucoup plus délicat. En effet, si les applications qui manipulent un

grand nombre de données sont exécutées efficacement avec des opérations mémoire-mémoire, leur implémentation sur des architectures centrées sur de larges bancs de registres est très sérieusement perturbée par les incessants transferts entre la mémoire et les registres. Dans le même ordre d'idée, il est pénalisant de systématiser des accès (lents) aux mémoires de données, à la manière des DSPs, lorsque ce sont toujours les mêmes données qui sont manipulées. Ces deux exemples mettent en avant la complexité de la tâche qui consiste à définir un modèle de programmation commun à un ensemble d'algorithmes. La redéfinition des interconnexions permet de s'affranchir de cette difficulté en autorisant le passage d'un modèle de programmation de type RISC à un modèle de type DSP, par simple transformation du pipeline d'exécution.

2.1.3. *Niveau logique*

Les architectures qui se situent à ce niveau sont qualifiées de grain fin en raison de la faible largeur de leurs chemins de données. La programmation opère au niveau logique sur des primitives de calcul manipulant des données binaires et sur leurs interconnexions. Étant donnée la quantité de ressources nécessaires à la définition de fonctions évoluées, ces réseaux sont très complexes et nécessitent un très grand nombre de données pour spécifier à la fois le traitement réalisé sur chacune des primitives de calcul et l'ensemble de leurs interconnexions.

En conséquence, la fréquence de programmation de ces circuits est faible et, en général, reste stable pendant un nombre de cycles conséquent pour ne pas perturber l'exécution de l'application par des phases de *gel* trop fréquentes. Cette limitation est contournée, dans certains composants, par la mise en place de chargement partiel (e.g. on ne modifie qu'une partie de la mémoire de programmation du composant).

En programmant une architecture au niveau logique, il est ainsi possible de définir n'importe quel type de chemin de données en synthétisant les opérateurs requis par l'application. Cette caractéristique est intéressante pour les traitements logiques pour lesquels chaque bit est susceptible d'être traité indépendamment du reste d'un flot de données. Si cette flexibilité facilite la spécialisation des chemins de données disposant d'un fort parallélisme au niveau bit, elle est en revanche défavorable aux traitements arithmétiques. En effet, l'emploi de cellules standards limite fortement les possibilités d'optimisation, et donc l'efficacité des opérateurs synthétisés sur l'architecture, par rapport à des solutions *full-custom* où ces opérateurs sont dédiés au traitement d'une opération unique (e.g. addition, multiplication).

2.2. *Réseaux d'interconnexion*

Au sein d'un composant programmable, le réseau d'interconnexion assure la distribution des données entre les différentes unités fonctionnelles. Sa topologie a un impact de tout premier ordre sur les performances, la consommation et la flexibilité du système. Trois grandes catégories de réseaux [ZHA 99] peuvent être distinguées,

chacune étant caractérisée par un temps de traversée, une complexité et une consommation.

2.2.1. *Les réseaux d'interconnexion globaux*

Les réseaux d'interconnexion globaux garantissent une totale connectivité des ressources, au sens où chaque élément (calcul, mémorisation, contrôle, etc.) peut potentiellement communiquer avec tous les autres. Les deux principaux motifs de ce type sont les réseaux de type *crossbar* et *multi-bus*.

La flexibilité de ces deux réseaux est identique et leur nature extrêmement orthogonale simplifie le développement de l'architecture. Cependant, ces motifs sont complexes à mettre en oeuvre de part leurs nombreux points d'interconnexion. De plus, leur consommation et leur temps de traversée sont proportionnels à leur taille et au nombre d'éléments connectés. Aussi, sachant que la surface croît comme le carré du nombre de modules connectés et que le temps de cycle décroît linéairement dans le même temps [ZHA 99], ce type de réseau est généralement réservé aux systèmes comportant peu d'éléments à connecter.

2.2.2. *Les réseaux d'interconnexion point-à-point*

Dans le monde des systèmes multiprocesseurs, il existe pléthore de topologies relatives aux réseaux point-à-point [VAR 94] dont le seul but est de fournir des connexions locales efficaces. Parmi ces motifs, on trouve notamment les réseaux en anneaux et les réseaux *mesh* 2-D généralisés.

Ce type de motif optimise les communications locales en temps et en surface tout en limitant le nombre de connexions. Ceci se fait cependant au prix de communications longue distance très nettement détériorées par le fait que les communications entre deux modules nécessitent la traversée d'un nombre de boîtes de commutations (*SB : Switch Box*) proportionnel à la distance entre la source et la destination. Étant donné que les délais des commutations sont bien supérieurs à ceux de la propagation des signaux le long des fils, les connexions longues distances sont lentes et consommatrices d'énergie. Par ailleurs, contrairement aux réseaux totalement connectés, les réseaux point-à-point sont très peu orthogonaux et donc plus complexes à exploiter. Les étapes de placement et de routage ont donc une importance toute particulière dans le flot de conception des architectures centrées sur de tels motifs.

2.2.3. *Les réseaux d'interconnexion hiérarchiques*

Une notion de hiérarchie peut être adjointe aux topologies précédentes pour pallier leurs insuffisances. Ceci conduit alors à la définition de motifs tels que les réseaux hiérarchiques segmentés, les réseaux en arbre ou encore les réseaux *mesh* hiérarchiques. Pour ces derniers, l'architecture est décomposée en *clusters* à l'intérieur desquels les ressources sont interconnectées en favorisant les communications locales. Pour les connexions longues distances, un second niveau d'interconnexions, global, est défini.

Celui-ci permet de proposer des communications *inter-cluster* efficaces en terme de performance et de consommation.

L'inconvénient majeur de ces topologies est la possibilité d'être confronté à des conflits lors des communications *inter-cluster*. Ceci impose une décomposition en *clusters* consciencieuse. Plus encore que pour les topologies précédentes, le placement est donc ici une étape cruciale puisqu'il détermine la forme et la surface des *clusters*. De même, le placement des ports d'interface des *clusters* influe de manière significative sur l'efficacité de la connectique. En terme de performance, l'efficacité s'obtient finalement au prix d'un développement fastidieux.

3. Classification et caractérisation

Compte tenu de la dimension de l'espace de conception, il apparaît nécessaire de définir un critère principal de classification autorisant une rapide évaluation des architectures proposées dans la littérature. À ce titre, nous considérons le grain de reconfiguration et les trois niveaux définis précédemment sont repris. Pour chacun d'eux nous discutons des ressources de calcul et du réseau d'interconnexion en nous appuyant sur de nombreux exemples de projet de recherche et/ou de produits commerciaux. Nous évoquons par ailleurs les différents modes d'utilisation de ces architectures et étudions l'impact du mode de reconfiguration sur les performances du système.

3.1. *Le niveau système*

Les architectures appartenant à ce niveau sont étudiées depuis de nombreuses années et sont plus communément appelées les processeurs programmables (Fig. 1). Elles se déclinent sous de

nombreuses formes en fonction de l'architecture de leur chemin de données (RISC, CISC, DSP) et de la méthode employée pour exploiter le parallélisme de l'application (superscalaire, VLIW).

3.1.1. *Les ressources de calcul*

Les ressources de calcul intégrées dans les processeurs sont classiquement des unités généralistes telles que des UALs, capables de traiter les opérations les plus courantes. Certaines architectures disposent également d'unités plus spécifiques, adaptées à des domaines applicatifs bien ciblés. L'exemple le plus connu concerne les multiplieurs et les unités de décalages intégrées dans les chemin de données des DSPs.

Récemment sont apparues des solutions architecturales plus évoluées, distribuées sous la forme d'IP, qui offrent la possibilité aux utilisateurs de spécialiser leur architecture en intégrant des unités de calcul spécialisées [BUR 00]. Pour exploiter ces nouvelles ressources de calculs, ces coeurs de processeurs doivent pouvoir modifier leur jeu d'instructions. Les constructeurs proposant ces solutions doivent donc disposer

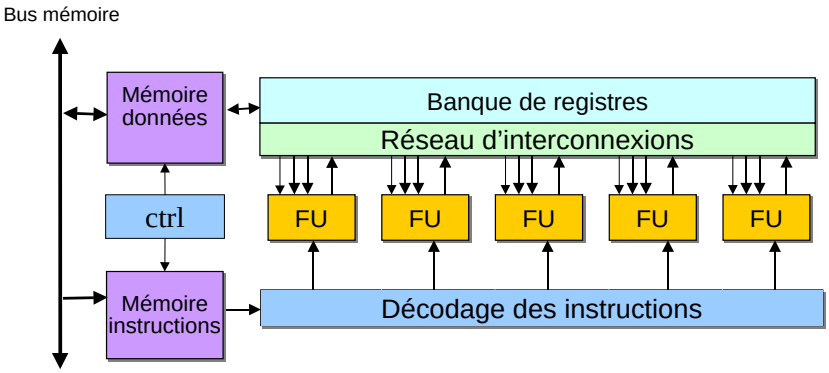

Figure 1. Architecture générique des composants reconfigurables au niveau système. Elle est constituée d'un ensemble de ressources travaillant sur une banque de registres centrale. Les opérations sont ordonnancées par un contrôleur unique.

d'outils de conception performants, capables d'adapter le flot de conception à ces nouvelles instructions. Les processeurs CARMEL [EYR 98] et ReAL [KIE 98, HOR 98], proposés respectivement par Infineon technologies et Philips, sont des exemples d'architecture supportant l'intégration de fonctionnalité dédiées. Ces architectures sont bien souvent citées sous le terme de *processeurs configurables* [DAV 00].

Le nombre d'unités de calcul intégrées dans ces architectures est particulièrement limité. Depuis les premiers processeurs qui ne disposaient que d'une unique ressource de calcul, un certain nombre de progrès ont cependant été réalisés. Aujourd'hui, les processeurs les plus performants en intègre huit. La gestion de ces ressources de calcul est complexe et nécessite l'exploitation de modèles de programmation nettement plus évolués que celui proposé par John Von Neuman [HEN 96]. À l'heure actuelle, deux approches sont utilisées.

– L'approche superscalaire qui se traduit par l'intégration d'un support matériel à l'exécution concurrente de plusieurs instructions.

– L'approche VLIW qui se base sur des outils de développement performants pour extraire un maximum de parallélisme au niveau instructions (ILP : Instruction-Level Parallelism) dans l'application.

Cette exploitation du parallélisme au niveau instructions permet d'augmenter sensiblement les performances des processeurs. Cependant, cette augmentation s'accompagne typiquement d'une sous-utilisation des ressources. Malgré les nombreux mécanismes architecturaux développés et mis en œuvre dans le cadre des processeurs hautes performances (prédiction de branchement, exécution conditionnelle, buffer de préchargement, etc.), la levée des dépendances de données et l'extraction du parallélisme d'instructions (dans des descriptions procédurales) sont des tâches aujourd'hui encore trop mal maîtrisées pour exploiter pleinement le potentiel de ces architectures.

3.1.2. *Les interconnexions*

Dans ce type d'architecture, les différents éléments constituant un processeur sont statiquement connectés. Les implémentations réalisées étant nécessairement de type temporel, tout transfert de données se traduit par le stockage de résultats intermédiaires dans une file de registres.

L'absence de flexibilité des interconnexions se traduit donc par de lourdes pénalités en performances. En contre-partie, très peu de bits suffisent à orienter les données. Les chemins sont précisés directement dans l'instruction et donc spécifiés à chaque cycle. De ce point de vue, la flexibilité est optimale puisque le processeur est capable d'exécuter n'importe quelle application.

3.1.3. *Usages*

L'un des principaux atouts des processeur est de supporter l'exécution de structures de contrôle complexes (branchements conditionnels, gestion d'interruptions, etc.). En effet, ce type de traitement nécessite une mise en oeuvre temporelle puisque d'un cycle à l'autre, le comportement de l'architecture doit pouvoir être adapté à l'environnement extérieur. Contrairement aux architectures qui vont être décrites dans les sections suivantes, et qui privilégient des implémentations spatiales plus efficaces lors des traitements réguliers, les processeurs programmables ne souffrent, dans ce contexte séquentiel, d'aucune concurrence.

La flexibilité des processeurs programmables leur ouvre une quantité impressionnante de domaines d'utilisation. Par voie de conséquence, ils sont aujourd'hui intégrés dans la quasi-totalité des systèmes numériques. Que ce soit à des fins de gestion du système ou de traitement, le concepteur dispose d'une palette de solutions lui permettant de répondre à la quasi-totalité des contraintes qu'il est susceptible de rencontrer. Les processeurs restent cependant aujourd'hui encore incapables de répondre conjointement à des contraintes de haute performance et de faible consommation. La principale explication à cet état de fait est qu'il est extrêmement délicat de limiter les gaspillages d'énergie associés à la lecture et au décodage des instructions, de même que ceux associés aux accès aux données dans de larges bancs mémoire.

3.2. *Le niveau fonctionnel*

Les architectures de ce niveau peuvent être schématisées par la figure 2. L'architecture générique consiste en un ensemble d'opérateurs arithmétiques dédiés ou programmables, reliés par un réseau d'interconnexions, qui communiquent vers l'extérieur au moyen de blocs d'entrées/sorties configurables. En fonction de l'aspect programmable ou dédié des ressources de calcul, les modèles de programmation peuvent être très diversifiés.

Par la suite, nous caractérisons cet espace de conception en se basant sur des réalisations concrètes issues de différents projets (Pléiades [RAB 00], RaPiD [CRO 99],

Chameleon [TAN 00], Morphosys [LEE 00], FPFA [1] [SMI 02], PACT XPP [BAU 01], Systolic Ring [SAS 01]), DART [DAV 03].

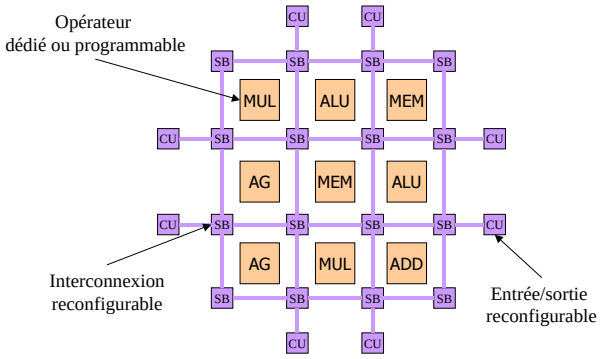

Figure 2. Architecture générique des composants reconfigurables au niveau fonctionnel. Elle consiste en un ensemble de ressources de calcul dédiées ou programmables, reliées par un réseau d'interconnexions reconfigurable.

3.2.1. *Les ressources de calcul*

Ici, deux approches sont utilisées pour définir les ressources : elles découlent des modèles de programmation de ces architectures. En effet, dans le cadre de certains projets, les concepteurs ont jugé bon de distribuer le contrôle dans l'architecture, et de définir une hiérarchie dans laquelle le plus bas niveau à pour but de définir la fonctionnalité de l'opérateur. Dans ce cas, les unités de calcul sont relativement complexes et capables de supporter plusieurs opérations. Elles sont typiquement constituées de plusieurs opérateurs (e.g. multiplieur, UAL, registres à décalages, etc.) et disposent d'un registre de configuration ou d'un contrôleur local. Ce type d'approche peut par exemple être illustrée par la figure 3 représentant la primitive de calcul du Systolic Ring.

L'exploitation de ressources de calcul complexes, faite dans des projets comme Morphosys ou Chameleon, augmente sensiblement la flexibilité de l'architecture mais se paie par une éventuelle sous-utilisation des ressources et par l'intégration d'un support local de configuration (un registre pour Morphosys ou Chameleon, un contrôleur local pour Systolic Ring).

D'autres architectures, telles que Pléiades, XPP, DART ou RaPiD, se basent en revanche sur des primitives de calcul câblées dont les fonctionnalités sont déterminées en fonction du domaine applicatif. On trouve ainsi dans ces architectures des opérateurs tels que des multiplieurs, des UALs, des générateurs d'adresses, des mémoires,

1. Pour des raisons stratégiques, FPFA a récemment été renommé MONTIUM (Mountain Chameleon) [P.H 03].

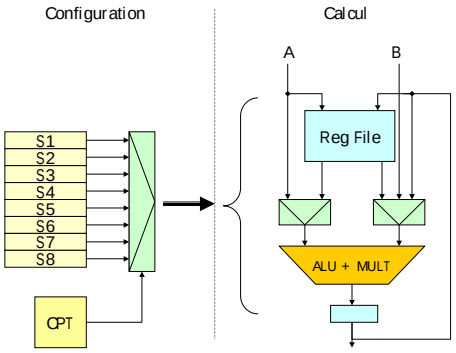

Figure 3. Architecture de la ressource de calcul du Systolic Ring, le D-Node : elle consiste en un multiplieur et une UAL câblés travaillant sur des données stockées dans une file de registre. Elle peut être contrôlée localement par le biais d'une mémoire de configuration pouvant stocker jusqu'à huit instructions.

etc., et éventuellement certaines fonctions plus spécifiques destinées à accélérer certains traitements, e.g. opérateur ACS (Add-Compare-Select) pour le codage de Viterbi ou l'opérateur SAD (Sum of Absolute Difference) pour l'estimation de mouvement. À l'extrême, certaines plateformes, comme Pléiades, définissent tous les opérateurs de l'architecture en fonction du domaine applicatif [ABN 96].

3.2.2. *Les interconnexions*

Compte-tenu du nombre important de ressources de calcul qu'elles intègrent, ces architectures exploitent typiquement des réseaux linéaires ou des réseaux hiérarchiques.

Les réseaux linéaires

Les réseaux linéaires permettent de définir des réseaux d'interconnexion relativement simples dans lesquels les communications locales sont très fortement privilégiées. À titre d'exemple, l'architecture RaPiD peut être citée (Fig. 4). Dans cette architecture, les communications entre les ressources de calcul se font par le biais d'un réseau segmenté. Les réseaux linéaires permettent de faire communiquer les ressources de calcul voisines de manière très flexible et efficace. En revanche, les communications entre cellules distantes nécessitent de faire transiter les signaux via plusieurs couches de bus, d'une manière très inefficace. La longueur et la charge de ces connexions se traduisent typiquement par une consommation énergétique élevée.

Ce type de topologie est particulièrement bien adapté aux applications fortement pipelinées. En effet, dans ces applications, l'essentiel des communications se fait entre des cellules voisines, en exploitant pleinement les connexions locales des réseaux linéaires. En revanche, du fait de l'inefficacité du réseau dans les communications longues distances, l'exploitation des architectures bâties sur ce type de topologie se

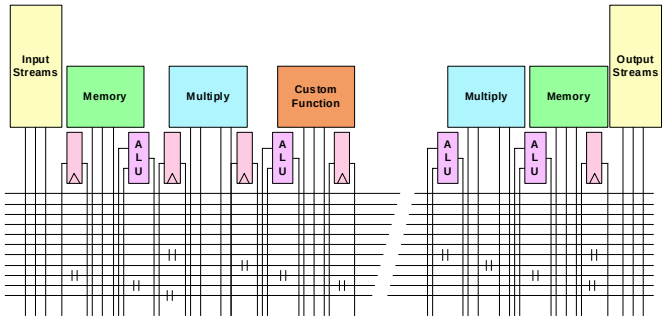

Figure 4. Architecture de RaPiD : les unités de calcul communiquent par le biais d'un réseau segmenté. Chaque opérateur peut communiquer avec ses plus proches voisins de manière très flexible. Les communications longues distances sont en revanche beaucoup plus coûteuses et nécessitent la traversée de plusieurs niveaux d'interconnexions.

limite à l'implémentation de pipelines profonds. Les applications disposant d'un parallélisme de grain plus épais (e.g. thread) seront implémentées de manière inefficace.

Les réseaux hiérarchiques

Chameleon, commercialisé depuis 2000 par *Chameleon Systems*, est un représentant relativement charismatique des architectures (Fig. 5) qui incluent ce type de réseau. Ses 108 unités arithmétiques sont d'abord réunies par groupe de neuf au sein de tuiles, puis regroupées par tranche. Le passage d'un niveau de hiérarchie à un autre se traduit par une réduction de la flexibilité des interconnexions. Dans le même ordre d'idée, les architectures DART, Pléiades et FPFA adoptent également une organisation hiérarchique de leurs ressources de calcul, en se limitant cette fois à deux niveaux de hiérarchie.

3.2.3. *Usages*

Bon nombre de projets associent dans un même système le bloc reconfigurable à un contrôleur chargé de gérer ses reconfigurations. Ce contrôleur est typiquement un processeur généraliste de type RISC. L'exploitation du bloc reconfigurable passe ici par une extension du jeu d'instructions pour gérer des communications explicites entre la ressource reconfigurable et le processeur hôte.

La définition d'un système autonome implique la possibilité de traiter tous les types d'applications (et pas seulement les calculs intensifs). Aussi, un soin tout particulier doit être apporté à la modification du bloc reconfigurable. En effet, celle-ci doit être suffisamment rapide pour ne pas perturber l'exécution d'un algorithme. Les recherches menées sur ce thème sont à l'heure actuelle très actives et se traduisent par une grande variété de mise en oeuvre.

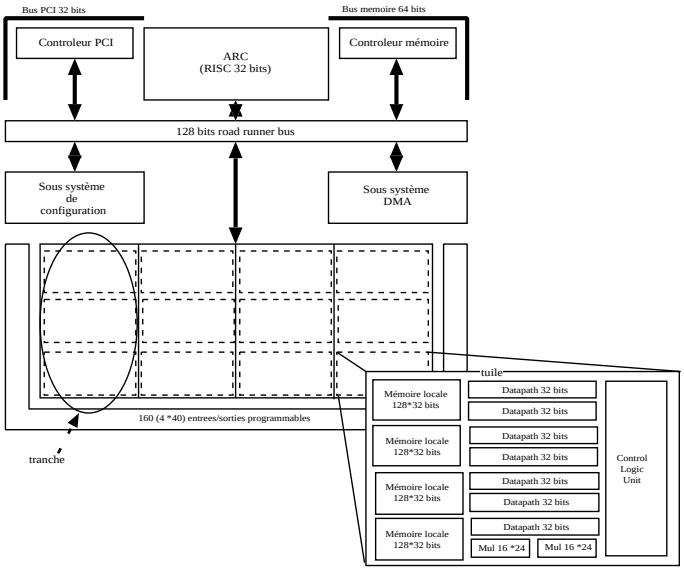

Figure 5. Architecture du Chameleon : cette architecture est décomposée en quatre tranches, elles même constituées de 3 tuiles. Au plus bas niveau de la hiérarchie, diverses unités de calcul sont interconnectées avec un réseau totalement flexible. Cette flexibilité s'amenuise à mesure que l'on remonte les différents niveaux de hiérarchie.

Une première solution, étudiée dans le cadre des FPGAs dits *multi-contextes* [MAE 00] et implémentée dans Chameleon [TAN 00], consiste à définir plusieurs plans de configuration. Cette méthode autorise la modification partielle et dynamique du bloc reconfigurable puisqu'un plan de configuration peut être modifié par le contrôleur pendant qu'un second spécifie la fonctionnalité du bloc. Bien qu'extrêmement performante (la reconfiguration ne prend que le temps d'une commutation sur l'entrée d'un multiplexeur), cette solution est très critiquable en terme de surface puisqu'elle nécessite la présence sur le circuit de plusieurs mémoires de configuration. Son coût énergétique est par ailleurs très élevé.

Une autre solution destinée à accélérer les temps de reconfiguration consiste à distinguer la configuration matérielle de l'architecture, décrivant la structure du chemin de données, et la configuration logicielle qui la contrôle. La première nécessite un nombre relativement important de données de configuration (quelques centaines de bits, voire plusieurs milliers) qui sont maintenues pendant toute la durée d'un traitement. La seconde est quant à elle caractérisée par un volume de données de configuration limité (quelques dizaines de bits) mais est amenée à évoluer très souvent. Elle permet par exemple de spécifier un *reset* ponctuel des accumulateurs ou d'initialiser certaines variables avant de rentrer dans une boucle. Cette solution, exploitée notamment dans DART, RaPiD et FPFA, se traduit par une optimisation des ressources mais

se paie par une augmentation du nombre de modèles de programmation du système [CRO 98]. L'architecture XPP autorise également un mode de reconfiguration similaire. Sur cette architecture, il est en effet possible de reconfigurer ponctuellement certains blocs particuliers de l'architecture, sans altérer le fonctionnement du reste du circuit. Ce mécanisme, défini sous le terme de *differential configuration* [PAC 02], cible principalement le chargement de constantes au sein du chemin de données. Il peut par ailleurs être utilisé afin de réduire le temps de reconfiguration.

Enfin, une dernière solution consiste à distribuer le contrôle au même titre que les ressources de calcul. Chaque bloc de calcul doit alors disposer d'un contrôleur gérant les chargements des configurations et les communications avec l'extérieur. Bien qu'efficace, cette solution pose de sérieux problèmes de synchronisation entre les différents éléments de l'architecture. Ce problème est intelligemment réglé dans Pléiades par l'emploi de mécanismes de communication de type Globalement Asynchrone Localement Synchrone (GALS) [WAN 99].

3.3. *Le niveau logique*

Dans cette section nous ne retiendrons que les circuits FPGA au détriment des circuits de type PAL (Programmable Array Logic) ou CPLD (Complex Programmable Logic Device) qui ne disposent pas d'une complexité suffisante pour implémenter des systèmes complets. Par ailleurs, la recherche sur les circuits FPGA ayant débuté dès le milieu des années 80, il existe aujourd'hui une très grande variété de produits et les passer exhaustivement en revue nécessiterait un chapitre complet. Aussi, ne seront mentionnés ici que deux grandes familles de composants reconfigurables au niveau logique, à savoir les produits des sociétés Xilinx et Altera, leaders mondiaux du marché. En dehors de celles-ci, le lecteur pourra vérifier dans la littérature [TAV 96, DUT 97] que c'est dans la réalisation des cellules et de leurs interconnexions que chaque constructeur apporte ses spécificités.

L'architecture générique de ces composants est représentée sur la figure 6. Elle consiste en une matrice d'éléments configurables, reliés par un réseau d'interconnexions, qui communiquent vers l'extérieur au moyen de blocs d'entrées/sorties configurables. La taille, le type et le nombre de cellules, de même que leurs interconnexions varient suivant les familles et les constructeurs.

3.3.1. *Les ressources de calcul*

La cellule de base est un élément mémoire associé à un module combinatoire. Construire un système consiste à associer ces cellules par le biais d'un réseau d'interconnexions programmable. La logique combinatoire est implantée sous forme de LUTs. Ce module combinatoire est associé à un élément de mémorisation au sein de la primitive de calcul : le bloc logique configurable (CLB) pour la famille Xilinx et l'élément logique (LE) pour Altera. Ces briques de base intègrent également des fonctionnalités intéressantes telles que le chaînage de retenue pour accélérer l'opéra-

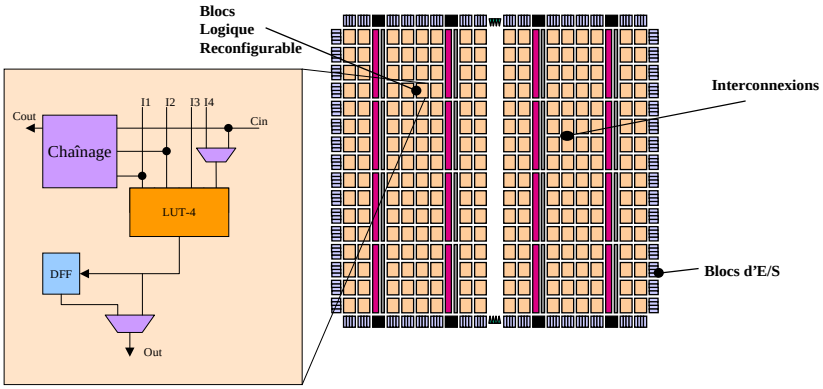

Figure 6. Architecture générique des composants reconfigurables au niveau logique. Ils sont constitués d'éléments de calcul reconfigurables, assemblés par le biais d'un réseau d'interconnexions complexe véhiculant des signaux binaires.

tion d'addition ou de petits modules mémoire (Fig. 6). Les dernières générations de FPGA, le VIRTEX pour Xilinx [XIL 01] et l'APEX pour Altera [ALT 01], offrent de très fortes capacités d'intégration (4,000,000 portes équivalentes) et une fréquence de fonctionnement élevée.

Dans les nouvelles architectures, les fabricants ont rajouté un niveau de hiérarchie en regroupant les cellules logiques au moyen d'une matrice d'interconnexions locales qui accélère les communications entre proches voisins. Les *versa-blocs*, pour Xilinx, regroupent 4 cellules de bases alors que les *LAB* d'Altera intègrent de 8 à 10 *LE*. Les architectures Altera intègrent par ailleurs un autre type de ressources connectées au reste du composant à la manière d'un *LAB*. Ces ressources, baptisées *EAB* (Embedded Array Block) ou *ESB* (Embedded System Block), exploitent un module mémoire piloté par de la logique séquentielle programmable. Elles peuvent être utilisées soit comme de simples mémoires, soit comme de grosses tables de scrutation pour spécifier des fonctions complexes telles qu'un multiplieur 4×4.

Certaines recherches, principalement menées dans les milieux universitaires, étudient par ailleurs l'intérêt de solutions dans lesquelles les ressources de calcul sont de granularité plus épaisse. À titre d'exemple, les projets PipeRench [CHO 00, GOL 99], Chess [MAR 99], DP-FPGA [CHE 94] ou DReAM [BEC 00] peuvent être cités. Les modules combinatoires constituant la cellule de base ne sont plus implémentés sous la forme de LUT à 3 ou 4 entrées mais sous la forme de composants multi-fonctions travaillant sur des données de plusieurs bits. Les opérateurs arithmétiques intégrés au sein de l'architecture sont obtenus par le biais de l'association de ces primitives de calcul. La réduction du nombre de cellules nécessaires à la construction d'opérateurs complexes permet ainsi d'accroître les performances du système, par rapport aux solutions basées sur de la logique reconfigurable, lors des traitements arithmétiques.

3.3.2. *Les interconnexions*

Les composants FPGA exploitent des réseaux d'interconnexion très complexes, où chaque bit doit pouvoir être orienté individuellement. Pour les composants Xilinx les interconnexions transitent à travers différents types de ligne suivant la distance, le temps de propagation et le nombre de portes connectées. Des lignes prédéfinies, dites directes, connectent efficacement les proches voisins. Des lignes, dites générales, parcourent l'ensemble du circuit et sont organisées suivant un motif de type *mesh* généralisé.

Pour les signaux dont la propagation doit être uniforme (signal d'horloge par exemple), des lignes basses impédances les distribuent simultanément à tous les blocs du circuit. Ces dernières disparaissent dans le VIRTEX au profit de lignes routant les signaux tous les 3 ou 6 blocs, l'horloge disposant quant à elle d'un réseau de routage dédié organisé en arbre. Les réseaux d'interconnexions des circuits Altera ont relativement peu évolués depuis le Flex8K [ALT 99] et se basent sur un maillage de pistes horizontales et verticales.

3.3.3. *Usages*

Historiquement, les composants FPGA ont tout d'abord été exploités pour remplacer des composants discrets ou valider des systèmes destinés à être intégrés dans des circuits de type ASIC [PIL 98]. Dans ce contexte, ces circuits sont configurés à l'initialisation et cette configuration est maintenue pendant toute la phase de validation. L'aspect statique de la reconfiguration n'implique dès lors aucune perte d'efficacité et les contraintes de couplage avec le système hôte sont très limitées. Les circuits FPGAs commerciaux sont, aujourd'hui encore, principalement utilisés à cet effet.

Ces composants peuvent, par ailleurs, être utilisées à des fins d'augmentation de performance en déchargeant le système hôte de tout ou partie des traitements intensifs pour lesquels il est mal adapté. De nombreuses applications en traitement du signal, de l'image, en cryptographie, en biologie moléculaire, etc., ont donné lieu à des architectures massivement parallèles, et ont démontré que des facteurs d'accélération de plusieurs dizaines, voire plusieurs centaines, d'ordres de grandeur peuvent être obtenus par le biais de cette approche. Dans ce cadre, trois modes d'utilisation peuvent être distingués :

Les cartes accélératrices

Ces cartes, contenant un ou plusieurs FPGAs et de la mémoire vive, peuvent se connecter à un processeur via des bus périphériques (e.g. PCI, VME). La machine PeRLe-1, fût l'une des premières réalisations de ce type. Celle-ci se base sur l'architecture PAM (Programmable Active Memory) [VUI 96] et est composée de 16 circuits FPGA Xilinx 3090, de mémoire vive et de 7 circuits FPGAs utilisés pour gérer les échanges avec la station hôte. Outre ce type d'assemblage bi-dimensionnels de circuits FPGAs, d'autres systèmes considèrent des assemblages linéaires tels que les systèmes SPLASH-2 [ARN 93] ou ArMen [POT 91]. Compte tenu de l'augmentation des capacités d'intégration, les recherches sur ce type de machines sont devenues net-

tement moins attractives. Désormais, la plupart de ces machines peuvent être intégrées au sein d'un seul et même composant.

Les Co-Processeurs :

Plus récemment, de nombreuses études ont été menées afin d'étudier des couplages plus étroits entre la ressource reconfigurable et le processeur hôte. À titre d'exemple, le projet GARP proposé par le groupe de recherche BRASS (Berkeley Reconfigurable Architectures, Systems and Software) [HAU 00], et le projet NAPA proposé par National Semiconductor [RUP 98] peuvent être considérés comme représentatifs, sachant que des projets industriels visent aussi à étudier ce type de couplage [BOR 02]. Les deux premières solutions exploitent des blocs reconfigurables conçus au sein même des laboratoires alors que la troisième exploite un composant commercial, le FPGA embarqué M2000 [M20].

Ce type d'architecture vise à décharger le processeur hôte des traitements pour lequel il n'est pas adapté. Il ne s'agit plus ici de configurer, à l'initialisation, le bloc reconfigurable pour un traitement particulier mais de modifier sa configuration à chaque fois que le processeur hôte rencontre un traitement critique. La reconfiguration du FPGA intervient donc au cours de l'exécution du programme sur le processeur hôte. La distinction entre configuration et reconfiguration apparaît donc ici clairement, de même que l'aspect critique du temps nécessaire à la reconfiguration du bloc reconfigurable.

Dans ce type d'architecture, un processeur de type RISC se charge de l'exécution des traitements de faible complexité. Lorsque celui-ci rencontre un traitement critique (généralement reconnu par une directive assembleur), il fait appel au bloc configurable afin de se décharger de ce traitement. Certains processeurs reprennent alors l'exécution du programme, sous réserve qu'il n'y ait pas de dépendances de données, et laissent une complète autonomie au bloc configurable, qui doit dès lors disposer d'un accès direct à la mémoire du processeur. C'est par exemple l'un des modes de fonctionnement privilégié de NAPA qui autorise ainsi une exécution de type *multi-thread*. Dans d'autres projets en revanche, tel que GARP, l'approvisionnement en données du bloc configurable est effectué sous la forme de transferts explicites, sous le contrôle du processeur hôte. Dans ce cas, les exécutions du programme sur le processeur et sur le bloc configurable sont mutuellement exclusives.

Un des facteurs critique pour évaluer la qualité de la solution proposée devient dès lors le temps nécessaire aux configurations. En conséquence, de nombreux mécanismes ont été envisagés dans le cadre de sa réduction. À ce niveau de granularité les deux principales options sont l'utilisation de circuits multi-contextes et la reconfiguration partielle. La configuration multi-contextes consiste à mémoriser plusieurs configurations. Le passage d'une configuration à une autre ne nécessite alors qu'une commutation entre les différents plans de configuration du système. Les performances sont alors très sensiblement améliorées mais ceci se paie par une explosion du coût du circuit. Afin de limiter ce surcoût, une autre solution, utilisée dans le cadre de GARP, consiste à stocker un certain nombre de plans de configuration dans des mémoires

caches. Le rapprochement entre les données de configuration et la ressource reconfigurable permet ainsi de réduire les temps de configuration. À titre d'exemple, le projet GARP qui utilise cette technologie ne consomme que quelques cycles pour spécifier une configuration de taille moyenne (500 blocs logiques).

La seconde méthode permettant de limiter les temps de configuration consiste à autoriser une modification partielle de la ressource configurable. Cette méthode, exploitée dans le cadre des projets GARP et NAPA, nécessite un découpage fonctionnel et logique de l'architecture mais ouvre en contrepartie des perspectives d'optimisation très intéressantes. Dans le même ordre d'idée, une autre option d'implémentation consiste à considérer plusieurs ressources configurables. Cette technique a été exploitée dans le cadre du projet ARDOISE, dans lequel deux ressources configurables dynamiquement (ATMEL AT40K[ATM 02]) sont modifiées suivant une méthode de "ping-pong", i.e. un FPGA est configuré pendant qu'un autre exécute un calcul [DEM 99].

Les unités fonctionnelles configurables

Afin d'améliorer plus encore la qualité du couplage entre le processeur et la ressource configurable, certains projets étudient la possibilité d'insérer des blocs configurables au sein même du chemin de données du processeur. L'alimentation en données de la ressource est alors extrêmement efficace et assurée par le biais des registres généraux du processeur. L'intérêt principal est que l'extension du jeu d'instructions du processeur hôte autorise l'utilisation de méthodes classiques de compilation. Ainsi l'utilisateur n'a pas à se soucier de l'architecture ciblée. L'utilisation de langages procéduraux tels que le C est satisfaisante. En revanche, la latence des opérations exécutées sur la ressource configurable impose l'emploi d'instructions multi-cycles et limite les gains qui peuvent être attendus.

Les études menées par le passé confirment cette tendance. À titre d'exemple, des expériences menées autour de l'intégration d'unités de calcul reconfigurables supportant aussi bien les traitements entiers que flottants n'ont mis en évidence que des gains en performance de l'ordre de 20% [SOL 01]. Dans le même ordre d'idée, l'extension du jeu d'instructions d'un processeur RISC par le bloc logique reconfigurable CHIMAERA ne se traduit que par un même gain moyen en performance de 20% [YE 00].

Ces résultats mitigés illustrent la difficulté de la tâche consistant à intégrer un accélérateur matériel avec un processeur hôte. Dans [RIZ 02], les auteurs mettent en avant les aspects devant être pris en compte afin de définir un système performant. Leur étude extrait deux principaux critères, a priori antinomiques, influençant l'efficacité du système.

– Les traitements implémentés doivent être relativement complexes : dédier un bloc reconfigurable à des instructions particulières semble donc inadapté. Les auteurs de [RIZ 02] préconisent l'exécution de traitements critiques nécessitant, au minimum, plusieurs centaines de cycles d'exécution sur le processeur hôte.

– Le couplage entre le processeur hôte et l'accélérateur doit être très étroit. Idéalement, des connexions directes doivent permettre un approvisionnement en données

efficace.

Associer ces deux aspects semble dès lors paradoxal puisqu'il nécessite à la fois des connexions directes entre le processeur hôte et l'accélérateur et des bandes passantes extrêmement importantes pour assurer une distribution efficace des données et des configurations. Ces deux paramètres influent donc directement sur le coût de la connectique entre le processeur hôte et la ressource reconfigurable. Bien qu'amorcées depuis de nombreuses années, les recherches sur le couplage entre processeur hôte et accélérateur matériel (qu'il soit dédié ou reconfigurable) sont aujourd'hui encore très attractives.

4. Energie et performances

Dans le cadre des applications émergentes, deux contraintes doivent nécessairement être considérées : la consommation et les performances. Cette section évalue ces deux aspects pour les architectures décrites précédemment, toujours en différenciant les trois niveaux de reconfiguration.

4.1. *Efficacité énergétique des architectures reconfigurables*

Concevoir une architecture faible consommation n'est pas une chose facile. Cette tâche nécessite en effet la prise en compte de ce critère à tous les niveaux de la conception, depuis la définition du système d'exploitation jusqu'à la réalisation physique du circuit. Dans les paragraphes qui suivent, nous présentons les atouts associés aux différents paradigmes de reconfiguration [PIL 03]. Il appartient alors aux concepteurs de fournir l'effort (très important) nécessaire à l'exploitation de l'éventuel potentiel de réduction d'énergie de ces architectures.

4.1.1. *Niveau système*

Les micro-processeurs sont réputés comme étant inefficaces d'un point de vue énergétique. Une des principales explications à cet état de fait est que le surcoût associé à la distribution du contrôle est considérable. En effet, à chaque cycle, un processeur se doit de lire une instruction et de la décoder pour contrôler les unités fonctionnelles. Il est tout de même à noter que certains processeurs embarqués, tels que le DSP16xx de Lucent Technologies [TEC] ou l'ADSP 21xx d'Analog Devices [DEV 96], limitent le coût énergétique de cette opération par l'insertion de buffer mémorisant les dernières instructions lues.

Un autre inconvénient associé aux architectures Harvard ou Von Neuman est l'exploitation de larges mémoires centrales. Dans ces architectures, le coût des accès aux données est prohibitif et la dissipation d'énergie est dominée par la mémoire centrale. Les processeurs souffrent finalement de la quantité limitée de parallélisme pouvant être traitée. Bien que beaucoup de micro-processeurs exploitent certains niveaux de

parallélisme (données [FRI 00], instructions [FAR 98] ou tâches [KRI 98]), leur degré de parallélisme est réduit afin de limiter le nombre d'unités fonctionnelles. En effet, un des aspects les plus importants lors de la conception d'un processeur est le coût du circuit. De ce fait, pour avoir une bonne densité de calcul (i.e. ratio entre les performances et la surface), le concepteur doit limiter le degré de parallélisme. En conséquence, puisque ces architectures ne peuvent exploiter que partiellement le parallélisme de l'application, elles doivent fonctionner à des fréquences plus élevées, nécessitant dès lors des arbres d'horloges plus gros et des tensions d'alimentation plus grandes.

4.1.2. *Niveau fonctionnel*

Limiter les cibles de la reconfiguration aux seules interconnexions peut avoir un impact très favorable sur la consommation. Se concentrer sur ce point réduit très sensiblement le volume de données de configuration, par rapport aux solutions reconfigurables au niveau logique . Dans le même temps, si les phases de reconfigurations n'interviennent qu'entre les phases de calcul intensif, les étapes de recherche et de décodage de configuration sont très occasionnelles. En effet, la règle des 80/20 affirme que 80% du temps d'exécution est consommé par seulement 20% du code, ce qui signifie que certaines portions du programme sont exécutées pendant de longues périodes [STI 02, VIL 02]. Ainsi, il est possible de réduire le nombre de lectures et de décodages de configuration en ne réalisant ces opérations que lors des appels de fonction (la configuration doit alors permettre l'exécution de toute la fonction). En réduisant à la fois le nombre et le coût des reconfigurations, on peut minimiser le gaspillage d'énergie associé à la distribution du contrôle dans le circuit.

Un autre aspect intéressant, sous-jacent à la capacité à reconfigurer le réseau d'interconnexions, est la possibilité d'adapter le parallélisme de l'application à celui de l'algorithme. En exploitant pleinement le parallélisme intrinsèque de l'algorithme, la fréquence de fonctionnement peut être réduite de même que la tension d'alimentation et par voie de conséquence, la consommation de puissance et d'énergie du circuit.

Malheureusement, le potentiel de reconfiguration est bien souvent inexploité dans les projets de recherche. En effet, hormis DART et Pléiades, la plupart des études menées se concentrent uniquement sur le gain en performance au prix d'une consommation d'énergie non maîtrisée.

4.1.3. *Niveau logique*

De nombreux travaux ont porté sur l'analyse de la distribution de la consommation dans les circuits FPGA [POO 02, SHA 02, KUS 97, GAR 00]. Toutes aboutissent à la constatation que la principale source de consommation est le réseau d'interconnexions. À titre d'exemple, dans une étude du FPGA Xilinx Virtex, K. Poon, dans [POO 02], estime le pourcentage d'énergie consommée dans le réseau d'interconnexions à 62%, contre seulement 18% dans les re ssources de calcul.

Un autre inconvénient associé à la faible granularité de ces architectures vient du volume de données de configuration qui leur est associé. En effet, les phases de reconfiguration nécessitent la transmission d'une grande quantité de données qui doivent nécessairement être stockées dans de larges mémoires centrales. Lorsque le système nécessite des reconfigurations du bloc reconfigurable, ces phases se traduisent donc par des pénalités énergétiques très importantes.

Varghese George, à Berkeley, a travaillé avec un certain succès à la réduction de la consommation des FPGAs [GEO 00], notamment en limitant la consommation des interconnexions [ZHA 98]. Cependant, à l'heure actuelle, ces architectures restent très inefficaces d'un point de vue énergétique.

4.2. *Performances*

Le critère de performance est sans aucun doute le plus important pour valoriser une architecture. Quelle que soit l'efficacité d'une architecture en consommation ou en surface, elle ne sera en effet jamais utilisée si elle ne dispose pas de la puissance de calcul nécessaire à l'exécution d'une application.

4.2.1. *Niveau système*

Les processeurs implémentent leurs applications de manière temporelle. Dès lors, ils sont mal adaptés à des calculs intensifs et répétitifs où il est possible d'extraire un fort parallélisme. Ceci peut quand même être modulé par l'exploitation d'un certain degré de parallélisme via l'intégration, dans les derniers micro-processeurs, de quelques instructions SWP ou SIMD.

Dans [MEN 02], Daniel Menard met en évidence un parallélisme d'instructions relativement important (entre 2.8 et 7.5) dans le cadre d'algorithmes de filtrage sur le TMS320C64x (supportant un ILP maximum de 8). Dans le même ordre d'idée, Emmanuel Gaudry, dans [GAU 01] met en avant un ILP variant de 2.3 à 3.4 pour le traitement d'un *Rake Receiver* [GRO 00] sur un processeur développé à ST microelectronics, le Lx [FAR 00] (disposant d'un ILP maximum de 4). Ceci démontre bien la capacité des processeurs à proposer de hautes performances. Ceci se fait cependant au prix d'un contrôle très complexe.

Ainsi, définir une architecture de processeur combinant à la fois haute performance et haute densité de calcul est un exercice extrêmement délicat. Compte-tenues des contraintes de coût inhérentes aux marchés ciblés par les processeurs programmables, les concepteurs de ces systèmes privilégient généralement la densité de calcul au détriment des performances.

En revanche, ces architectures sont très efficaces pour des traitements irréguliers, caractérisées par un très faible niveau de parallélisme et par des comportements non déterministes. En effet, il est difficile de prédire leur comportement à un cycle donné, celui-ci dépendant bien souvent du résultat des traitements réalisés au cycle précédent

ou de l'environnement extérieur (e.g. arrivée d'interruption). Si les processeurs programmables sont peu performants pour les calculs intensifs, ils sont donc parfaitement adaptés aux traitements irréguliers et aux applications orientées contrôle.

4.2.2. *Niveau fonctionnel*

En limitant la reconfiguration aux seules interconnexions d'opérateurs arithmétiques, les architectures reconfigurables au niveau fonctionnel disposent d'un potentiel d'optimisation très élevé. La reconfiguration autorise en effet la définition d'un pipeline d'exécution totalement adapté au traitement à réaliser en éliminant, par exemple, la nécessité de stocker les variables temporaires en mémoire ou en registre grâce au chaînage des opérateurs. Cette mise en adéquation du pipeline d'exécution au motif de calcul à traiter permet par conséquent l'optimisation de la fréquence de fonctionnement.

La fréquence de fonctionnement peut par ailleurs être accrue en exploitant au mieux le parallélisme de l'application. La possibilité de modifier l'agencement des opérateurs dans ces architectures facilite cette exploitation. Outre le parallélisme d'opérations obtenu par l'intégration de nombreuses unités fonctionnelles, il est en effet possible de définir plusieurs chemins de données pour profiter d'un parallélisme de tâche. Par l'intégration d'opérateurs supportant des traitements SWP, un parallélisme de données peut finalement être exploité sans introduire de pénalité trop importante en terme de coût.

En revanche, ces architectures n'ont pas la possibilité d'exploiter le parallélisme de niveau bit des applications. En effet, dans ce contexte, l'emploi d'opérateurs travaillant au niveau arithmétique, même s'ils autorisent les traitements logiques, est très inefficace puisque avant de pouvoir traiter un bit particulier dans un mot de N bits, il est nécessaire de préalablement l'extraire en lui appliquant un masque. Un traitement logique nécessite donc deux opérations sur une ressource arithmétique.

Pour pallier à ce problème, un certain nombre d'architectures, telles que Pléiades, DART ou le Chameleon, intègrent parmi leurs unités de calcul des opérateurs logiques capables de traiter plus efficacement ces opérations.

4.2.3. *Niveau logique*

De par leur structure, les circuits FPGA sont prédestinés à implémenter spatialement leurs applications. Le niveau de performance est alors directement lié à la quantité de ressources qu'ils intègrent. L'efficacité de ces architectures dans le cadre de la manipulation de données binaires, voire de petite taille (< 4 bits), est indéniable et ces architectures ne disposent d'aucune réelle concurrence dans ce domaine.

Le problème est cependant tout autre pour les traitements arithmétiques. Comme il a déjà été dit, construire un opérateur arithmétique nécessite l'utilisation et l'interconnexion d'un grand nombre de ressources de calcul. La quantité de ressources de routage devant être traversées devient alors prohibitive et les performances des opérateurs ainsi synthétisés sont très nettement dégradées par rapport à des solutions

full-custom. Bien que l'introduction d'architectures à grain plus épais limite cette tendance, les résultats obtenus restent insatisfaisant pour des traitements critiques. Ainsi, pour atteindre un niveau de performance comparable aux solutions dédiées, les implémentations doivent accroître leur niveau de parallélisme, ce qui se traduit par des solutions inefficaces du point de vue de la densité de calcul. Pour réduire l'importance de ce problème, les architectures les plus récentes intègrent au sein de leur structure reconfigurable des opérateurs câblés travaillant au niveau arithmétique (e.g. multiplieurs du VIRTEX-II [XIL 01], blocs DSP du Stratix [ALT 02]).

5. Conclusion et perspectives

Dans la littérature [HAR 01], le terme reconfigurable qualifie une très large gamme de systèmes. Ceci s'explique par la variété de formes que revêt cette propriété. En effet, une architecture est qualifiée de reconfigurable dès lors qu'elle dispose d'un support lui permettant de s'adapter aux traitements qui lui sont assignés. L'ambiguïté des termes "support" et "traitement" dans cette définition fait de son interprétation un exercice délicat et explique la relative obscurité du concept de *reconfigurable* ainsi que la quantité de systèmes qui peuvent justifier cette dénomination.

Dans cette article, nous avons balayé l'espace de conception de ces architectures avec comme critère principal la reconfiguration. Par l'examen de ce critère, nous avons tout d'abord analysé la flexibilité de ces architectures en fonction de leur potentiel de reconfiguration. Nous avons par la suite balayé l'espace de conception de ces architectures et illustré les différents concepts relatifs aux différents paradigmes de reconfiguration par le biais d'exemples issus de la littérature. Finalement, nous avons évalué ces architectures suivant les critères de performance et d'efficacité énergétique.

Dans le cadre des architectures reconfigurables, des efforts conséquents ont été produits afin de réduire le coût de la reconfiguration et ainsi augmenter leur flexibilité. Ainsi, depuis l'échec cuisant du FPGA Xilinx Xc6200 [XIL 96] ayant amorcé ces recherches, de nombreux mécanismes architecturaux ont été introduits pour simplifier la gestion de la reconfiguration tout en réduisant son surcoût. Ainsi, grâce à la reconfiguration partielle, à la définition de grain de reconfiguration plus épais, etc., les performances des architectures reconfigurables permettent désormais de supporter de très hauts niveaux de performance.

À l'heure actuelle en revanche, la reconfiguration dynamique n'est pas exploitée dans le cadre de la maîtrise de la consommation. Cette contrainte, bien qu'unanimement reconnue comme l'une des plus critiques pour les systèmes embarqués, n'est quasiment jamais prise en compte lors de la conception des architectures reconfigurables. DART et Pléiades font parties des rares architectures pouvant justifier cette qualification. L'exploitation du potentiel des architectures reconfigurables dans le cadre de la maîtrise de l'énergie est à l'heure actuelle un des thèmes de recherche des plus attractif [PIL 03].

À la manière de la spécialisation des processeurs programmables, qui s'est traduite par la définition de nombreuses classes d'architectures (DSP, micro-contrôleur, processeur embarqué, etc.), la tendance actuelle, en matière de conception d'architectures reconfigurables, est l'adaptation des caractéristiques de l'architectures au besoin applicatifs. La conception orientée plate-forme devient donc nécessairement l'un des enjeu majeurs relatifs à la valorisation des circuits reconfigurables. Dans ce cadre, de nombreux projets sont en cours.

Une des difficultés majeur concerne ici la définition de métriques suffisamment discriminent pour obtenir, à partir d'une plateforme reconfigurable, l'instance qui répondra au mieux aux besoins de l'application. La diversité des paramètres devant être évalués pour juger de la pertinence d'une solution interdit en effet, dans ce contexte, la définition de *benchmarks* performants. Dans le cadre de la caractérisation des architectures reconfigurables, l'heure est donc à l'analyse qualitative et non à l'analyse quantitative.

La diversité des concepts mis en œuvre pour réduire le coût de la reconfiguration se traduit par ailleurs par une explosion du nombre de modèles de programmation relatifs aux architectures reconfigurables.La conséquence directe de cet état de fait est une difficulté évidente à concevoir des outils de programmation simples et efficaces. Si à l'heure actuelle la plupart des constructeur sont en mesure de fournir une chaîne de développement complète, la conception de celle-ci est pénalisée par l'absence de méthodologie suffisamment souple pour répondre à des modèles de programmation nombreux et bien souvent mal identifié.

Outre l'optimisation des architectures du point de vue des performances, du coût ou encore de la consommation, de nombreux travaux sont actuellement menés dans le cadre de l'utilisation de ces ressources à des fins d'accélération de calcul. Dans ce cadre, l'une des principales difficultés devant être levée est relative au couplage de la ressource reconfigurable au reste du système. Bien qu'amorcée depuis de nombreuses années, ces recherches restent très actives [WAL 02, MEI 01].

6. Bibliographie

[ABN 96] ABNOUS A., RABAEY J., « Ultra Low-Power Specific Multimedia processors », 459–468, novembre 1996, VLSI Signal Processing IX.

[ALT 99] ALTERA, « Flex 8000 Programmable Logic Device Family », juin 1999.

[ALT 01] ALTERA, « APEX20k Programmable Logic Device Family », août 2001.

[ALT 02] ALTERA, « Stratix FPGA Family », Datasheet ver. 3.0, décembre 2002.

[ARN 93] ARNOLD J. M., « The Splash 2 Software Environment », *The Journal of Supercomputing*, vol. 9, n° 3, 1993, p. 277-290, Kluwer Academic Publishers.

[ATM 02] ATMEL, « 5K -50K Gates Coprocessor FPGA with FreeRAM », Technical report 0896c-rev 04/02, avril 2002, ATMEL.

[BAU 01] BAUMGARTE V., MAY F., NÜKEL A., VORBACH M., WEINHARDT M., « PACT XPP - A self-Reconfigurable Data Processing Architecture », *International Conference on*

Engineering of Reconfigurable Systems and Algorithms (ERSA 01), Las Vegas, USA, juin 2001.

[BEC 00] BECKER J., PIONTECK T., GLESNER M., « DReAM : A Dynamically Reconfigurable Architecture for Future Mobile Communication Applications », *international Workshop on Field Programmable Logic and Applications (FPL 00)*, Villach, Austria, août 2000, Lecture Notes in Computer Science 1896, p. 312–321.

[BOR 02] BORGATTI M., LERTORA F., FORÊT B., CALÍ L., « A Reconfigurable System featuring Dynamically Extensible Embedded Microprocessor, FPGA and Customisable I/O », *Custom Integrated Circuits Conference (CICC)*, Orlando, USA, mai 2002.

[BUR 00] BURSKY D., « Upgraded DSP Core Tackles future Communication needs », *Electronic design*, vol. 48, n° 8, 2000, p. 66 - 68.

[CHE 94] CHEREPACHA D., LEWIS D., « A Datapath Oriented Architecture for FPGAs », *International Symposium on Field Programmable Gate Arrays (FPGA 94)*, Monterey, USA, février 1994.

[CHO 00] CHOU Y., PILLAI P., SCHMIT H., SHEN J., « PipeRench Implementation of the Instruction Path Coprocessor », *International Symposium on Microarchitecture (MICRO-33)*, Monterey, USA, décembre 2000, p. 147–158.

[CRO 98] CRONQUIST D. C., FRANKLIN P., BERG S. G., EBELING C., « Specifying and Compiling Applications for RaPiD », *Symposium on Field-Programmable Custom Computing Machines (FCCM 98)*, Los Alamitos, USA, avril 1998, p. 116–125.

[CRO 99] CRONQUIST D. C., FRANKLIN P., FISHER C., FIGUEROA M., EBELING C., « Architecture Design of Reconfigurable Pipelined Datapath », *Advance Research in VLSI (ARVLSI 99)*, Atlanta, USA, mars 1999, p. 23–40.

[DAV 00] DAVID R., « Analyse qualitative des nouvelles architectures embarquées et outils associés : étude du cas des processeurs ARM et DSP », Mémoire DEA, Univeristé de Rennes1/ INSA/ SUPELEC, juillet 2000.

[DAV 03] DAVID R., « Architecture reconfigurable dynamiquement pour applications mobiles », PhD thesis, Université de Rennes I, juillet 2003.

[DEM 99] DEMIGNY D., PAINDAVOINE M., WEBER S., « Architecture à reconfiguration dynamique pour le traitement temps réel des images », *Technique et Science de l'Information Numéro Spécial Architectures Reconfigurables*, vol. 18, n° 10, 1999, p. 1087–1112.

[DEV 96] DEVICES A., « ADSP-2100 Family DSP microcomputers », Datasheet rev b., 1996, Analog Devices.

[DUT 97] DUTRIEUX L., DEMIGNY D., *Architecture des FPGA et CPLD, méthode de conception, le langage VHDL*, Eyrolles, 1997.

[EYR 98] EYRE J., BIER J., « Carmel enables Customizable DSP », *Microprocessor report*, vol. 12, n° 17, 1998.

[FAR 98] FARABOSHI P., DESOLI G., FISHER J. A., « The Latest Word in Digital and Media Processing », *IEEE Signal Processing Magazine*, , 1998, p. 59 – 85.

[FAR 00] FARABOSHI P., BROWN G., FISHER J., DESOLI G., « Lx : A technology Platform for Customizable VLIW Embedded Processing », *International Symposium on Computer Architecture (ISCA 00)*, Vancouver, Canada, juin 2000.

[FRI 00] FRIDMAN J., « Sub-Word Parallelism in Digital Signal Processing », *IEEE Signal Processing Magazine*, vol. 17, n° 2, 2000, p. 27–35.

[GAR 00] GARCIA A., « Etude sur l'estimation et l'optimisation de la consommation de puissance des circuits logiques programmables du type FPGA », PhD thesis, Ecole Nationale Supérieure des Télécommunications, 2000.

[GAU 01] GAUDRY E., « Estimation de la complexité algorithmique d'une chaîne de traitement WCDMA en télécommunication mobile : application au processeur Lx », Master's thesis, DEA STIR, ENSSAT-Université de Rennes 1, juin 2001.

[GEO 00] GEORGE V., « Low Energy Field-Programmable Gate Array », PhD thesis, University of California, Berkeley, 2000.

[GOL 99] GOLDSTEIN S., SCHMIT H., MOE M., BUDIU M., CADAMBI S., « PipeRench : A Coprocessor for Streaming Media Acceleration », *International Symposium on Computer Architecture (ISCA 99)*, Atlanta, USA, mai 1999.

[GRO 00] GROE J., LARSON L., *CDMA Mobile Radio Design*, Harteck House Publishers, 2000.

[HAR 01] HARTENSTEIN R., « A Decade of Reconfigurable Computing : A Visionary retrospective », *Design Automation and Test in Europe (DATE 01)*, Munich, Germany, mars 2001.

[HAU 00] HAUSER J., « Augmenting a microprocessor with reconfigurable hardware », PhD thesis, University of California, Berkeley, 2000.

[HEN 96] HENNESSY J. L., PATTERSON D. A., *Architecture des ordinateurs : une approche quantitative (deuxième édition)*, International Thomson Publishing, 1996.

[HOR 98] VAN DER HORST E., KLOOSTERHUIS W., VAN DER HEYDEN J., « A C Compiler for the Embedded R.E.A.L. DSP », *International Conference on Signal Processing (ICSPAT 98)*, Toronto, Canada, septembre 1998.

[KIE 98] KIEVITS P., LAMBERS E., MORMAN C., WOUDSMA R., « R.E.A.L. DSP Technology for Telecom Baseband Processing », rapport, 1998, Philips Semiconductors, ASIC Service Group.

[KRI 98] KRISHNAN V. S., « Speculative Multithreading Architectures », PhD thesis, University of Illinois, 1998.

[KUS 97] KUSSE E., « Analysis and Circuit Design for Low Power Programmable Logic Modules », Master's thesis, University of California, Berkeley, 1997.

[LEE 00] LEE M., SIGNH H., LU G., BAGHERZADEH N., KURDAHI F., « Design and Implementation of the MorphoSys Reconfigurable Computing Processor », *Journal of VLSI and Signal Processing-Systems for Signal, Image and Video Applications*, vol. 24, n° 2, 2000, p. 147–164.

[M20] M2000, « Flexeos family technical manual », www.m2000.fr.

[MAE 00] MAESTRE R., FERNANDEZ M., KURDAHI F., BAGHERZADEH N., SINGH H., « Configuration Management in Multi-Context Reconfigurable Systems for Simultaneous Performance and Power Optimization », *International Symposium on System Synthesis*, septembre 2000, p. 107–113.

[MAR 99] MARSHALL A., VUILLEMIN J., STANSFIELD T., KOSTARNOV I., HUTCHINGS B., « A Reconfigurable Arithmetic Array for Multimedia Applications », *International Symposium on Field Programmable Gate Arrays (FPGA 99)*, février 1999, p. 135–144.

[MEI 01] MEI B., VERNALDE S., MAN H. D., LAUWEREINS R., « Design and Optimization of Dynamically Reconfigurable Embedded System », *International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, USA, juin 2001.

[MEN 02] MENARD D., QUEMERAIS P., SENTIEYS O., « Influence of Fixed-point DSP architecture on Computation Accuracy », *European Signal Processing Conference (ESIPCO 02)*, Toulouse, France, septembre 2002.

[PAC 02] PACT, « The XPP White Paper : A Technical Perspective », Release 2.1, mars 2002, PACT.

[P.H 03] P. HEYSTERS G. S., « Mapping of DSP Algorithms on the MONTIUM Architecture », *International Reconfigurable Architecture Workshop (RAW 03)*, Nice, France, avril 2003.

[PIL 98] PILLEMENT S., « Méthodologies d'évaluation et de prototypage des systèmes numériques intégrés », PhD thesis, Université de Montpellier II, décembre 1998.

[PIL 03] PILLEMENT S., DAVID R., SENTIEYS O., « Architectures reconfigurables : Opportunités pour la faible consommation », *papier invité aux journées Faible Tension Faible Consommation (FTFC)*, Paris, France, mai 2003.

[POO 02] POON K., « Power Estimation for Field Programmable Gate Arrays », Master's thesis, University of British Columbia, 2002.

[POT 91] POTTIER B., « ArMen : une machine parallèle intégrant un réseau de circuits logiques programmables », PhD thesis, Université de Rennes 1, juin 1991.

[RAB 97] RABAEY J. M., « Reconfigurable Processing : the Solution to Low-Power Programmable DSP », *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, avril 1997.

[RAB 00] RABAEY J., « A low-energy heterogeneous reconfigurable DSP IC », *Design Automation Conference (DAC 00)*, Los Angeles, USA, juin 2000.

[RIZ 02] RIZZO D., COLAVIN O., « A Video Compression Case Study on a Reconfigurable VLIW Architecture », *Design Automation and Test in Europe (DATE 02)*, Paris, France, mars 2002.

[RUP 98] RUPP C., LANDGUTH M., GRAVERICK T., GOMERSALL E., HOLT H., « The NAPA Adaptative Processing Architecture », *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 98)*, avril 1998, p. 28–37.

[SAS 01] SASSATELLI G., TORRES L., GALY J., CAMBON G., DIOU C., « The Systolic Ring : A Dynamically Reconfigurable Architecture for Embedded Systems », *International Workshop on Field Programmable Logic and Applications (FPL 01)*, Lecture Notes in Computer Science 2147, 2001, p. 409–419.

[SHA 02] SHANG L., KAVIANI A., BATHALA K., « Dynamic Power Consumption in Virtex-II FPGA family », *International Symposium on Field Programmable Gate Arrays (FPGA 02)*, Monterey, USA, février 2002, p. 157–164.

[SMI 02] SMIT G., HAVINGA P., HEYSTERS P., ROSIEN M., « Dynamic Reconfiguration in Mobile Systems », *International Conference on Field Programmable Logic and Applications (FPL 02)*, Montpellier, France, septembre 2002, Lecture Notes in Computer Sciences 2438, p. 171–181.

[SOL 01] SOLIHIN Y., CAMERON K., LUO Y., LAVENIER D., GOKHALE M., « Mutable Functional Units and their Applications on Microprocessors », *International Conference on Computer Design (ICCD 01)*, Austin, Texas, USA, 2001.

[STI 02] STITT G., GRATTAN B., VILLARREAL J., VAHID F., « Using On-Chip Configurable Logic to Reduce System Software Energy », *Symposium on Field-Programmable Custom Computing Machines (FCCM 02)*, Napa, California, septembre 2002.

[TAN 00] TANG X., AALSMA M., JOU R., « A compiler directed aproach to hiding configuration latency in chameleon processors », *International Workshop on Field Programmable Logic and Applications (FPL 00)*, Villach, Autriche, avril 2000, Lecture Notes in Computer Science 1896.

[TAV 96] TAVERNIER C., *Circuits logiques programmables*, Microcontrôleur et environnement, Dunod, 1996.

[TEC] TECHNOLOGIES L., « DSP 16xx », rapport, Lucent Technologies.

[VAR 94] VARMA A., RAGHAVENDRA C., *Interconnection Networks for multiprocessors and multicomputers : Theory and Practice*, IEEE Computer Society Press, 1994.

[VIL 02] VILLARREAL J., SURESH D., STITT G., VAHID F., NAJJAR W., « Improving Software performance with Configurable Logic », *Design Automation for Embedded Systems*, vol. 7, n° 4, 2002, p. 325–339, Kluwer Academic Publishers.

[VUI 96] VUILLEMIN J., BERTIN P., SMITH A., SILVERMEN H., « Programmable Active Memories : Reconfigurable Systems Come of Age », *IEEE Transaction of VLSI Systems*, vol. 4, n° 1, 1996.

[WAL 02] WALSTROM J. D., « The Design of the Amalgam Reconfigurable Cluster », PhD thesis, Master's Thesis, University of Illinois at Urbana-Champaign, 2002.

[WAN 99] WAN M., ZHANG H., BENES M., RABAEY J., « A Low-Power Reconfigurable Data-Flow Driven DSP System », *Workshop on Signal Processing Systems (SIPS 99)*, Taipei, Taiwan, octobre 1999.

[XIL 96] XILINX, « Xilinx 6200 Preliminary Data Sheet », San Jose, CA, 1996.

[XIL 01] XILINX, « VIRTEX2 1.5V Series Field Programmable Gate Arrays », juillet 2001.

[YE 00] YE Z. A., MOSHOVOS A., HAUCK S., BANERJEE P., « CHIMAERA : a high-performance architecture with a tightly-coupled reconfigurable functional unit », *ACM SIGARCH Computer Architecture News*, vol. 28, n° 2, 2000, p. 225–235.

[ZHA 98] ZHANG H., RABAEY J., « Low-Swing Interconnect Interface Circuits », *International Symposium on Low Power Electronics and Design (ISLPED 98)*, New York, USA, août 1998, p. 161–166.

[ZHA 99] ZHANG H., WAN M., GEORGE V., RABAEY J., « Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs », *International Workshop on VLSI*, avril 1999.