Les architectures reconfigurables

Stéphane Rubini* **et Dominique Lavenier****

** Laboratoire d'Informatique de Brest,*
Université de Bretagne Occidentale, BP 809, 29285 Brest Cedex
Adresse électronique : Stephane.Rubini@univ-brest.fr
*** IRISA,*
Campus de Beaulieu, 35042 Rennes Cedex
Adresse électronique : Dominique.Lavenier@irisa.fr

RÉSUMÉ. *La technologie FPGA actuellement disponible permet de construire des machines à matériel adaptable ou configurable. Cet article dresse un panorama de ces machines. Les schémas architecturaux proposés diffèrent par le statut des ressources configurables, par leur degré de couplage avec les autres composants. Ce critère sert de base à une classification en trois groupes : (1) machines à unité interne configurable, (2) machines à co-processeur configurable et (3) machines à unité périphérique configurable. Une analyse des outils de programmation associés termine l'article.*

ABSTRACT. *The FPGA technology is available to build custom computing machines. This paper gives a survey of theses machines. The proposed architectural schemes are different by the status of reconfigurable resources, and by the coupling level with the other components. This criterion is used to classify machines into three groups : (1) machines with a configurable internal unit, (2) machines with a co-processing configurable unit, and (3) machines with a peripheral configurable unit. An analysis of the programming tools ends the paper.*

MOTS-CLÉS : *FPGA, machines à matériel configurable, architecture spécialisée, synthèse de circuits*

KEYWORDS: *FPGA, FPGA-based custom computing machines, dedicated architectures, circuit synthesis*

Introduction

« Toujours plus vite, toujours plus performant » telle est la devise de chaque nouveau calculateur. Pour atteindre cet objectif, les concepteurs agissent principalement sur deux volets : l'accroissement des fréquences d'horloge et l'exploitation du parallélisme. Dans cet article, nous nous concentrons sur le deuxième aspect en introduisant une technique en pleine expansion, l'intégration de la logique reconfigurable dans les calculateurs.

Le parallélisme est exploité de diverses manières ; d'abord au sein des processeurs par deux grandes techniques : le superpipeline et le superscalaire (les micro-processeurs modernes) ; ensuite par la multiplication des processeurs pour former une structure parallèle programmable (les calculateurs parallèles) ; ou encore, par la spécialisation des architectures lorsque des volumes de calcul importants sont à effectuer dans une espace restreint.

Dans un processeur, le superpipeline repose sur la décomposition des instructions en une suite d'étapes, chaque étape utilisant une ressource distincte à un instant donné. S'il y a régularité dans le flot d'instructions toutes les ressources sont utilisées en permanence et on exécute, ainsi, plusieurs instructions en parallèle. Le superscalaire, par contre, s'appuie sur la duplication des unités fonctionnelles. Le contrôle du processeur demeure centralisé, mais plusieurs instructions peuvent être exécutées simultanément en leur attribuant dynamiquement des unités fonctionnelles distinctes. Ces deux techniques sont intéressantes, car transparentes pour le programmeur. Le processeur est considéré comme une machine purement séquentielle, et le résultat se doit de rester cohérent avec celui fourni par une exécution *réellement* séquentielle.

Cependant, le niveau de parallélisme escompté sur ce type de machine est limité : typiquement sur des applications courantes et en considérant une hiérarchie mémoire correcte, on peut considérer un débit de 1,5 à 2 instructions par cycle (ou encore, à un instant donné, plusieurs dizaines instructions en cours d'exécution). Les dépendances de données et les ruptures du flot séquentiel d'instructions en sont les principaux responsables. Ensuite, le modèle de programmation utilisé s'appuie sur une séquentialisation de l'exécution des actions, qui tend à « étouffer » le parallélisme intrinsèque du programme.

Ainsi, l'accroissement du nombre d'unités fonctionnelles disponibles ne suffit pas à étendre significativement le niveau de parallélisme extrait dynamiquement des codes séquentiels. Seules des machines parallèles sont adaptées pour exploiter ce potentiel. La multiplicité des ressources est considérée en amont de l'exécution matérielle, soit au niveau des compilateurs, soit au niveau du modèle de programmation. Malgré une complexité de programmation et un coût financier importants, les machines parallèles sont utilisées pour résoudre certains problèmes, car elles sont seules en mesure de produire des puissances de calcul suffisantes.

Une troisième approche, non exclusive des deux précédentes, s'appuie sur une *adaptation de matériel* aux problèmes à traiter. En général, ces architectures dites dédiées exhibent d'excellentes performances sur des classes restreintes d'applications caractérisées. Hélas, les temps de développement, le coût et le manque de souplesse en limitent la portée effective.

Une opportunité technologique récente, celle des composants logiques programmables ou FPGA (Field Programmable Gate Array), permet d'étendre le domaine d'intérêt de l'adaptation matérielle. Une machine qui utilise la technologie FPGA se situe à mi-chemin entre un système dédié et un système programmable. Les circuits FPGA offrent la possibilité de spécialiser une architecture, mais cette spécialisation n'est pas immuable dans le temps : par simple reconfiguration des composants d'autres architectures peuvent être mises en œuvre.

Les premiers composants FPGA ont été développés pour rassembler sous un même

boîtier toute la logique qui, dans un système numérique, était physiquement dispersée dans plusieurs circuits intégrés. Aujourd'hui, la capacité d'intégration ne limite plus ces composants à ce seul rôle : leurs ressources internes leur permettent d'implanter des circuits logiques complexes. Par exemple, des architectures d'une complexité équivalente à celle des micro-processeurs du début des années 80 peuvent maintenant être contenus dans les composants les plus récents.

Ainsi, de par l'évolution technologique, sont nés de nouveaux usages de la logique programmable. On peut citer le prototypage rapide qui consiste à câbler une architecture dans des composants FPGA pour la tester *in situ* avant une réalisation VLSI irréversible ; l'accélération des simulations de langages de description matérielle sur des plates-formes FPGA dans lesquelles la description est directement mise en œuvre sur le matériel programmable ; l'intégration, dans des architectures existantes de parties reconfigurables pour accélérer certains calculs spécifiques. Dans ce dernier cas, et c'est ce qui nous intéresse ici, on mise à la fois sur la spécificité des opérateurs de calcul et sur la parallélisation du traitement.

Les *machines à matériel configurable* incluent donc une zone configurable, sans fonctionnalités pré-établies. La définition de son comportement est laissée à l'appréciation de l'utilisateur, qui détermine l'implantation des fonctions matérielles des parties du traitement à accélérer.

Les architectures de ces machines sont très diverses et peuvent se caractériser par leur couplage avec la partie reconfigurable. On trouve ainsi des processeurs qui possèdent leurs propres unités fonctionnelles reconfigurables (i.e. intégrées dans le chemin de données interne), d'autres qui accèdent à la partie reconfigurable via les bus externes du processeur, ou, enfin, des processeurs qui leur confèrent seulement un statut de périphérique.

Se pose ensuite le problème de configurer ces éléments, l'idéal étant bien sûr de confier cette tâche à un compilateur qui, à partir du code source, extrait automatiquement les parties consommatrices de calcul et les traduit *in fine* en matériel. C'est là l'enjeu majeur de cette technologie. Du degré d'automatisation et de la qualité des résultats dépendra le succès de ces machines. À l'heure actuelle, la programmation de telles machines relève plus de la micro-électronique que de l'informatique, les outils de programmation étant directement issus de la CAO électronique.

Cet article propose de faire le point sur ces différents aspects. La section suivante expose d'abord le concept de la logique reconfigurable. La section 2. dresse ensuite un panorama des diverses architectures et l'illustre sur trois exemples de machines reconfigurables. La section 3. est consacrée aux outils de programmation ; elle montre les efforts engagés par plusieurs équipes de recherche dans ce domaine. Enfin, en conclusion, nous constatons les réussites et les insuccès des machines à matériel configurable dans la situation actuelle, situation qui sera sans doute remise en cause par le développement des possibilités de reconfiguration partielle.

1. Les circuits logiques configurables

Le concept de matériel adaptable peut s'appuyer sur des technologies et des architectures multiples. Ce paragraphe présente la structure générale des composants lo-

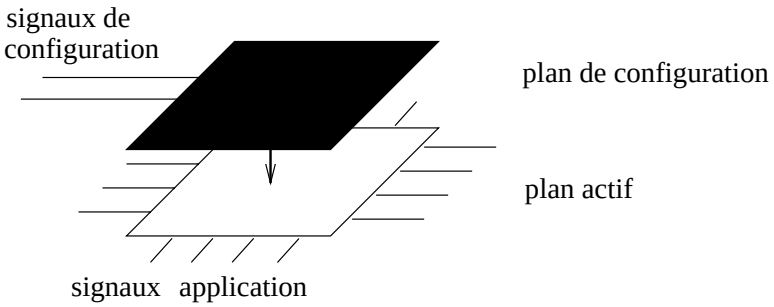

FIG. 1 - *Principe d'un circuit logique programmable*

giques configurables, les technologies utilisées pour les réaliser, et introduit la problématique du choix des ressources de calcul et de communication. Enfin, les outils permettant la production des configurations sont décrits.

1.1. *Principe*

L'idée commune à tous les circuits configurables est de proposer, non pas une fonction standard figée dont le champ d'application est le plus large possible, mais une structure adaptable que l'utilisateur peut spécialiser pour une utilisation donnée. Plus exactement, la fonction fondamentale de tout circuit logique configurable est de permettre la définition de la topologie d'un réseau de cellules élémentaires simples et universelles, afin d'obtenir un comportement spécialisé du circuit dans sa globalité.

Schématiquement, un circuit logique configurable se présente comme la superposition de deux plans : un plan supérieur de programmation et un plan actif (figure 1). Le plan actif se compose d'une matrice d'opérateurs logiques simples et d'un ensemble de ressources d'interconnexion. Le plan de programmation est constitué d'éléments de mémorisation. Le comportement du niveau actif est déterminé selon les données écrites dans le plan supérieur : chaque élément de mémorisation valide des interconnexions entre les cellules et forme un réseau quelconque d'opérateurs logiques. C'est la topologie de ce réseau qui définit le comportement du circuit.

L'augmentation des capacités d'intégration a permis de développer des circuits logiques configurables de grande taille, les circuits FPGA. Ces circuits sont destinés à des problèmes nécessitant plusieurs centaines, voire plusieurs milliers de portes logiques. Ils incluent des ressources de mémorisation et disposent de nombreuses broches d'entrées/sorties. La figure 2 montre l'évolution rapide de la capacité d'intégration fournie par les circuits FPGA.

1.2. *Configuration*

Les circuits FPGA se configurent *in situ* sur les cartes auxquelles ils sont destinés ou à l'aide de programmateur de bureau simple. Certains ne sont configurables qu'une seule fois, d'autres sont re-configurables à l'aide d'un matériel externe ou directement

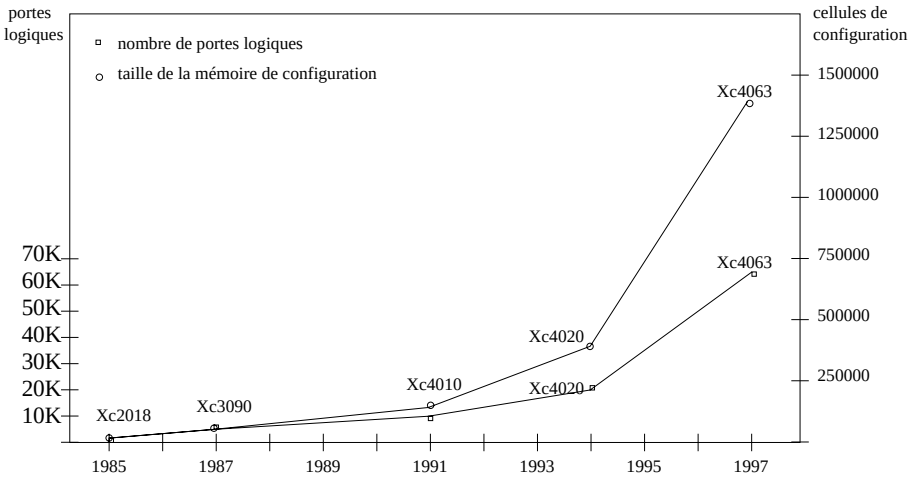

FIG. 2 - *Capacité des circuits* FPGA *et taille de leur mémoire de configuration.*

sur site à la mise sous tension du système. Ces différences s'expliquent par la technologie choisie pour la fabrication des éléments de mémorisation. Trois technologies sont principalement proposées : (1) les anti-fusibles, (2) les transistors à grille flottante, et (3) les cellules de RAM statique (bascule bistable implantée à l'aide de 4 transistors). Ces technologies ont hérité des développements effectués pour les composants mémoire, volatile ou non [RESV93].

Dans le cadre des machines à matériel configurable, les cellules de mémorisation de type SRAM sont très majoritairement employées. Elles admettent un nombre illimité de phase d'écriture, sans requérir de tension ou de courant élevé, avec des temps d'accès rapides, de l'ordre de la centaine de nano-secondes ; les circuits FPGA utilisant cette technologie sont entièrement reconfigurables en quelques centaines de milli-secondes, et sont faciles à interfacer avec les autres composants d'un système à micro-processeur. La figure 2 montre la taille des mémoires de configuration en regard de la capacité d'intégration pour quelques circuits FPGA.

Pour la plupart des circuits FPGA, les phases de configuration se traduisent par la définition complète de l'ensemble des valeurs enregistrées dans le plan de configuration. Certains circuits disposent d'un système d'adressage utilisé pour désigner une ou un groupe de cellules ; les fonctionnalités implantées dans le circuit FPGA peuvent alors être *partiellement* modifiées.

1.3. *Ressources internes*

La structure interne des circuits FPGA doit s'adapter à des problèmes de nature variée. Une organisation en une matrice de blocs logiques distincts, reliés par des ressources publiques d'interconnexions, constitue la solution la plus générale. La figure 3 montre une architecture de type *Manhattan* où les ressources logiques, les blocs lo-

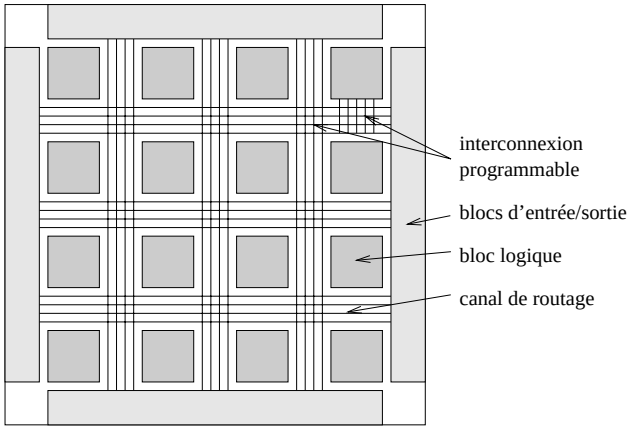

FIG. 3 - *Architecture Manhattan*

giques, sont séparées par des groupes de segments d'interconnexion, appelés canaux de routage.

La conception d'une architecture de composant FPGA repose en grande partie sur le choix des ressources intégrées dans chaque bloc logique, de même que le nombre et la taille des segments d'interconnexion. Ces choix résultent d'un compromis entre le taux d'utilisation des ressources logiques, qui diminue lorsque le grain des blocs augmente, et le besoin en ressources d'interconnexions, qui augmente lorsque le grain diminue. Par exemple, les circuits FPGA Xilinx de la famille 4000E disposent de blocs logiques composés de :

– deux tables pré-calculées capables d'évaluer deux fonctions logiques quelconques de 4 variables d'entrées,

– deux bascules D flip-flop,

– d'une circuiterie dédiée à la propagation de retenue.

Dans ce même composant, 18 segments d'interconnexion sont disponibles par canal, répartis en 8 segments de longueur 1, 4 segments de longueur 2 et 6 lignes qui parcourent la matrice d'un bord à l'autre. D'autres architectures sont fondées sur des blocs logiques plus simples, seulement 2 transistors par bloc pour les circuits FPGA de la société Crosspoint, mais avec une structure d'interconnexion plus dense.

1.4. *Génération des données de configuration*

Des outils logiciels permettent de générer l'ensemble de valeurs qui définissent la configuration d'un circuit FPGA, c'est à dire les fonctionnalités des blocs logiques et le routage des liaisons entre ceux-ci. À partir d'une description de l'architecture à im-

planter sous forme d'un ensemble d'équations logiques, plusieurs traitements sont nécessaires pour obtenir un fichier de configuration :

1. Partitionnement : les équations logiques sont partitionnées en un ensemble équivalent d'équations. Chaque équation de ce nouvel ensemble peut être implantée dans un seul bloc logique du composant FPGA cible.
2. Placement : des blocs logiques sont sélectionnés dans la matrice et affectés au calcul des nœuds du réseau booléen.
3. Routage : les ressources d'interconnexion sont affectées à la communication de l'état des nœuds du réseau vers les différents blocs logiques qui en ont besoin.
4. Génération des données numériques de configuration : les informations abstraites de routage, de placement, et la définition des équations implantées dans les blocs sont transformées en un ensemble de valeurs numériques, qui seront chargées dans le plan de configuration du composant FPGA.

Ces algorithmes sont caractérisés par une complexité importante. De plus, de meilleurs résultats sont prévisibles si les phases de partitionnement, de placement et de routage sont menées interactivement. La taille des problèmes à résoudre impose l'utilisation d'heuristiques afin de limiter le temps de calcul ; malgré cela, plusieurs minutes restent nécessaires pour obtenir un fichier de configuration pour une application d'une complexité équivalente à quelques milliers de portes logiques.

Les temps de calcul nécessaires restent importants, même s'ils paraissent acceptables dans une philosophie d'outils de CAO électronique. L'amélioration de cette caractéristique passe par une réflexion concomitante de l'architecture du composant FPGA et des outils de programmation.

Les composants FPGA, introduits dans les architectures de machine, apportent une ressource matérielle adaptable. D'un autre point de vue, ils constituent des systèmes massivement parallèles, support de réseaux d'opérateurs de topologie et de fonctionnalité programmables. La seconde partie de cet article montre leur position et leur statut architectural dans les projets actuels de machines à matériel configurable.

2. Les machines à matériel configurable

Les premières propositions d'utilisation de circuits logiques configurables dans les architectures de calcul datent de 1988 environ, avec les projets ARMEN, SPLASH et PERLE. Depuis, de nombreux projets explorent les possibilités de cette technologie [1].

L'apport attendu de l'adjonction de ressources configurables est lié à l'adaptation des unités opératives aux programmes et aux données à traiter [Rub95]. La spécialisation se traduit à plusieurs niveaux :

– adaptation de la taille des chemins de données à la dynamique réelle des données pour l'application considérée ; les bus et les opérateurs ne sont pas surdimensionnés par rapport aux valeurs effectivement traitées.

1. Une liste est disponible en consultant l'URL http://www.io.com/∼guccione.

– spécialisation des opérations effectuées ; la contrainte de généralité liée aux circuits de traitement figés disparaît.

– limitation du nombre d'échange avec la hiérarchie mémoire ; la notion de registre d'état peut être étendue afin de mémoriser un état interne complexe de l'unité de traitement.

Toutes les formes de parallélisme peuvent être exploitées dans les unités opératives configurables. Il est difficile de définir, dans le cadre général, quelles sont les formes les mieux adaptées, puisque, par définition, l'intérêt de la configurabilité est pouvoir implanter une architecture spécifique pour chaque application. Il n'en demeure pas moins que l'environnement architectural de la ressource configurable détermine les caractéristiques des échanges de données avec les autres composantes de la machine, et ce faisant, il restreint l'éventail des implantations acceptables et efficaces.

Nous proposons ici une description et une classification des architectures actuellement mises en œuvre pour les machines configurables. Les critères de caractérisation retenus se basent sur la structure et la réalisation effective des unités reconfigurables, et sur le degré de couplage avec les autres ensembles fonctionnels des machines.

2.1. *Structure des unités configurables*

La capacité des circuits logiques configurables actuels est encore insuffisante pour certaines applications. Le nombre de circuits FPGA physiquement mis en œuvre pour réaliser une unité reconfigurable procède d'un choix préalable d'utilisation. En règle générale, les systèmes de grande capacité sont plutôt dédiés à l'implantation de circuits faiblement couplés aux autres éléments de la machine, avec un flot de contrôle autonome, ou structurés en pipeline. Les unités reconfigurables de petite taille réalisent, au contraire, des opérations plus élémentaires à partir d'un couplage serré avec le reste du système.

La figure 4 donne une vue schématique des principales solutions expérimentées. Trois catégories de réalisation peuvent être distinguées :

1. Les systèmes les plus simples se composent d'un nombre très restreint de circuits FPGA (moins de 4) associés à un processeur (figure 4a). Les systèmes HARP1 [LLP93], *Spyder* [IS94], PRISM [AS93] constituent des exemples de ce type de machines. Ils sont conçus soit comme support pour des activités de conception concurrente matérielle-logicielle (co-design) [GCM94], soit comme accélérateur de traitement. Dans le premier cas, la généralité et la visibilité de l'unité reconfigurable sont prioritaires, en même temps qu'un accès direct aux entrées/sorties. Dans le second cas, la simplicité et la rapidité du processus de synthèse sont particulièrement recherchées. La démonstration du concept de *cache fonctionnel* [FT93] est l'objectif de ces projets ; le matériel nécessaire à un instant donné est disponible jusqu'à ce qu'il devienne inutile, ou moins prioritaire qu'une autre fonction matérielle qui le remplace.

2. La deuxième catégorie d'accélérateurs reconfigurables se présente comme un assemblage linéaire de circuits FPGA (figure 4b). Les systèmes SPLASH [GHK+91],

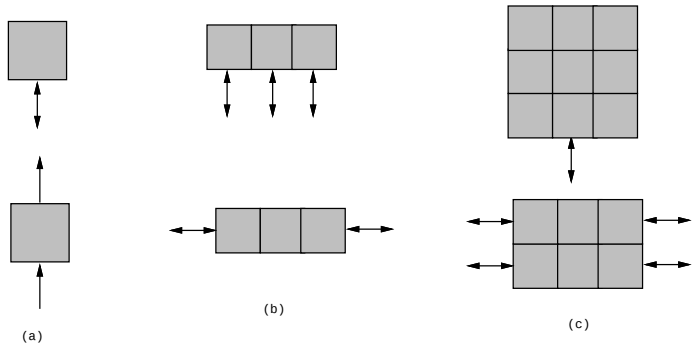

FIG. 4 - *Structure des unités configurables : les carrés grisés représentent les composants* FPGA*, les flèches les liaisons avec le ou les processeurs. Deux carrés adjacents schématisent des liaisons broches à broches directes entre 2* FPGA.

ARMEN [Pot91] ou CHAMP constituent des exemples de ce type de machines. Ces architectures sont adaptées à l'implantation de pipelines de traitement unidimensionnels de grande taille et permettent la mise en œuvre de modèles systoliques de calcul. Les principales différences dans cette catégorie de machines proviennent des techniques d'alimentation en données et de la disponibilité de ressources de mémorisation entre les étages des pipelines.

3. La troisième catégorie d'unité reconfigurable est constituée de tableaux bidimensionnels de circuits FPGA (figure 4c). Les machines PAM [BRV89], Enable++ [HKL+95] ou *Virtual Computer* représentent cette classe de machines. Chaque tableau est constitué de plus d'une dizaine de circuits FPGA. L'utilisateur se voit ainsi proposer une surface de composants logiques vierges, que Bertin et al. qualifient dans [BRV89] de « fonderie logicielle de silicium ». Ces architectures diffèrent par l'accès aux ressources de mémorisation, et par les capacités de routage interne à la surface reconfigurable. Certaines cartes disposent de structures de bus et de circuits FPIC (Field Programmable Interconnect Circuit) spécialisés pour les interconnexions. D'autres utilisent des interconnexions bord à bord simples entre les circuits FPGA, les éventuels besoins de routage étant remplis par le réseau d'interconnexion interne des composants FPGA. La complexité des circuits réalisables dans ces unités de traitement est élevée, et le couplage avec le processeur peut être faible.

La complexité des architectures mises en œuvre pour l'implantation d'unités reconfigurables diffère donc de façon importante. Les solutions retenues induisent une granularité variable des tâches accélérées.

2.2. *Alimentation en données des unités reconfigurables*

L'alimentation des unités de traitement reconfigurable est un point fondamental pour l'obtention de performances optimales. De nombreux projets montrent des ac-

célérations limitées par le débit d'entrées/sorties. Dans [GHK+91], Gokhale et al. indiquent que les performances de la machine SPLASH I étaient ralenties, sur certains problèmes, d'un ordre de magnitude par des entrées/sorties trop lentes.

La position et le degré de couplage des unités reconfigurables dans les systèmes actuels dépendent à la fois de choix architecturaux et de contraintes de réalisation. Cependant, quelle que soit la solution retenue, il est primordial que la granularité des tâches effectuées dans la logique reconfigurable et la granularité des transferts de données soient harmonisées. Le débit et la latence déterminent le grain optimal des communications avec les unités configurables. La visibilité des signaux de contrôle et le degré de couplage entre les éléments de la machine constituent aussi des caractéristiques importantes qui influent sur la finesse des événements de synchronisation.

La figure 5 représente schématiquement les solutions mises en œuvre. Par ordre décroissant du degré de couplage, les circuits configurables peuvent se situer (1) sur les chemins de données internes d'un processeur, (2) sur les bus externes avec un statut de co-processeur ou (3) à l'extrémité d'une mémoire tampon.

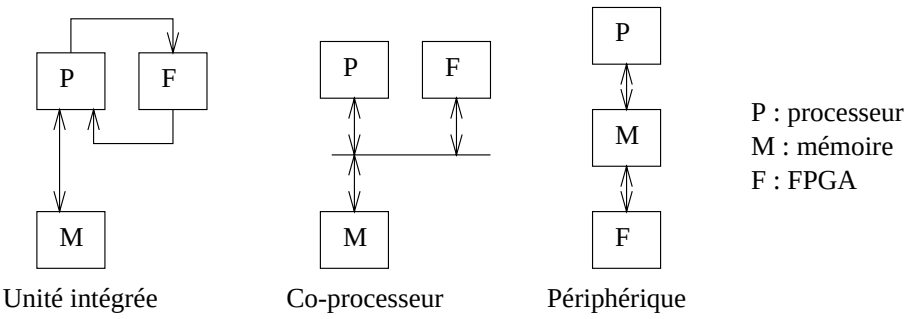

FIG. 5 - *Couplage processeur/*FPGA. *La latence et le débit des communications processeur/*FPGA *conditionnent la structure et les fonctionnalités potentielles de la ressource configurable.*

1. Unité de traitement intégrée : l'unité reconfigurable est située sur les chemins de données internes du processeur, au même titre qu'une unité entière ou flottante. Son alimentation est alors réalisée efficacement par des transferts avec les registres généraux du processeur. Le temps de propagation des signaux dans les circuits reconfigurables étant plus important que dans les unités fixes, le contrôle du processeur doit considérer des *opérations configurables* multicycles et gérer les aléas éventuels. Les problèmes de reprise de pipeline et de commutation de tâches semblent, dans ce contexte, délicats à résoudre.

   Les processeurs développés dans le cadre des projets BRASS, OneChip [Wit95] ou Spyder intègrent une ressource configurable de ce type.

2. Co-processeur : l'unité configurable est située sur les bus externes du processeur. Le co-processeur est alimenté par des accès externes du processeur, ou il peut

éventuellement obtenir le contrôle du bus pour réaliser des opérations d'accès direct sur la mémoire. Certains processeurs disposent d'instructions dédiées aux échanges avec les co-processeurs, ce qui permet de disposer d'un couplage plus serré. Par exemple, la machine PRISM II utilise des instructions de ce type, disponibles sur le processeur AMD29K, pour alimenter efficacement l'unité configurable par mots de 64 bits (le bus d'adresses et le bus de données sont logiquement fusionnés pour cela). Les performances de ces implantations sont néanmoins limitées par le temps de propagation *inter-boîtiers* et par les retournements de bus nécessaires à l'obtention des résultats.

Les machines PRISM, ARMEN, HARP1 utilisent des circuits FPGA en position de co-processeurs reconfigurables.

3. Périphérique : dans ce dernier cas, les unités configurables présentent un statut de périphérique. Les échanges avec la carte mère sont réalisés par l'intermédiaire d'accès DMA sur des bus d'entrées/sorties. Une mémoire FIFO permet de réguler le flux d'information entre les deux entités. Le temps de latence est élevé, mais la bande passante permet une alimentation efficace de l'accélérateur configurable. De plus, les calculs peuvent s'effectuer en recouvrement avec les instructions du processeur.

   Les machines PERLE et Virtual Computer possèdent un statut de ce type. Elles communiquent avec le processeur associé par l'intermédiaire des bus d'entrées/sorties de stations de travail. En particulier, les machines précitées utilisent le bus SBUS pour les stations Sun et le bus *TurboChannel* pour les stations DEC.

Les différentes solutions décrites induisent des difficultés de réalisation très variables. Le développement d'une unité reconfigurable présentant un statut de périphérique peut s'appuyer sur des corps de machines existantes, alors que l'intégration dans les processeurs demande une redéfinition complète de celui-ci.

Le paragraphe suivant présente une instance de machine pour chacune des catégories définies ci-dessus.

2.3. *Exemples de machines à matériel configurable*

Unités internes configurables : le processeur Garp Le processeur *Garp* est développé dans le cadre du projet BRASS de l'université de Berkeley [BRA96]. La première implantation du *Garp* regroupera un corps de processeur MIPS-II, un cache d'instructions, un cache de données, et un réseau logique reconfigurable dans le même circuit intégré (voir la figure 6).

La matrice de blocs reconfigurable est organisée en ligne ; chacune est capable d'effectuer des opérations logiques quelconques sur des données d'une taille maximum de 46 bits. Une configuration partielle, ligne par ligne, est possible, en moins d'une vingtaine de cycles bus.

La compilation totalement automatique du langage C ANSI est visée dans le cadre de ce projet. Le rôle alloué à la ressource reconfigurable est l'accélération des boucles internes des programmes. Les tâches spécifiques liées à la disponibilité d'une ressource reconfigurable, sont l'identification des sections de code candidates, et leur synthèse

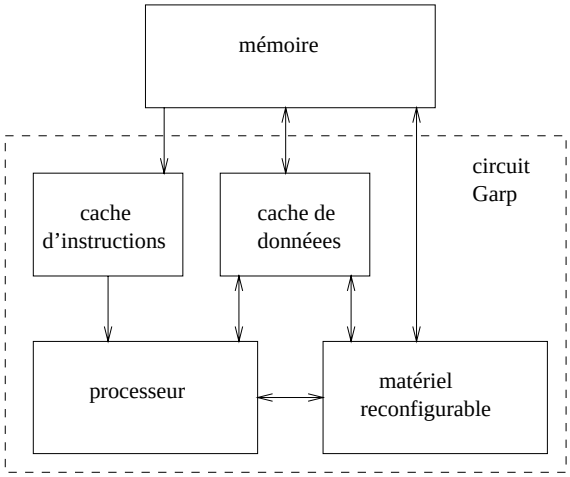

FIG. 6 - *Architecture du processeur* Garp. *La ressource reconfigurable, fortement couplée au chemin de données du processeur, possède un statut identique à celui des unités arithmétiques du processeur. Elle dispose également de ports d'échanges avec le cache de données et la mémoire principale.*

matérielle. Les outils de synthèse s'appuient sur un graphe de flot de données dérivé du programme. La rapidité du processus de compilation est un objectif majeur des concepteurs.

Le projet BRASS a débuté récemment et à notre connaissance, aucune performance n'a encore été publiée.

Co-processeur : le système PRISM II La machine PRISM II (*Processor Reconfigurable through Instruction Set Metamorphosis*), développée par l'université de Brown, utilise un processeur AMD29050 et une plate-forme matérielle reconfigurable composée de 3 circuits Xilinx 4010 [WAL+93]. La figure 7 montre l'architecture simplifiée de cette machine.

Les échanges avec la plate-forme matérielle sont effectués à l'aide des instructions co-processeur du processeur AMD. Une largeur de bus de 64 bits est disponible et un échange avec le co-processeur peut être réalisé en 30 ns.

La machine PRISM se programme en langage C. Le frontal du compilateur est celui du logiciel GNU gcc, le générateur de code produit soit du code assembleur pour le processeur, soit une configuration de circuit FPGA. La synthèse matérielle s'appuie sur un graphe de flot de données et un graphe de flot de contrôle ; un réseau d'opérateurs est déduit du flot de données, et un contrôleur gère les entrées/sorties et les itérations.

Les résultats rapportés dans [WAL+93] montrent des temps d'exécution diminués dans un facteur compris entre 6 et 86 pour un jeu test de programmes C. Les algorithmes considérés nécessitent des calculs sur des données de taille non standard ou des

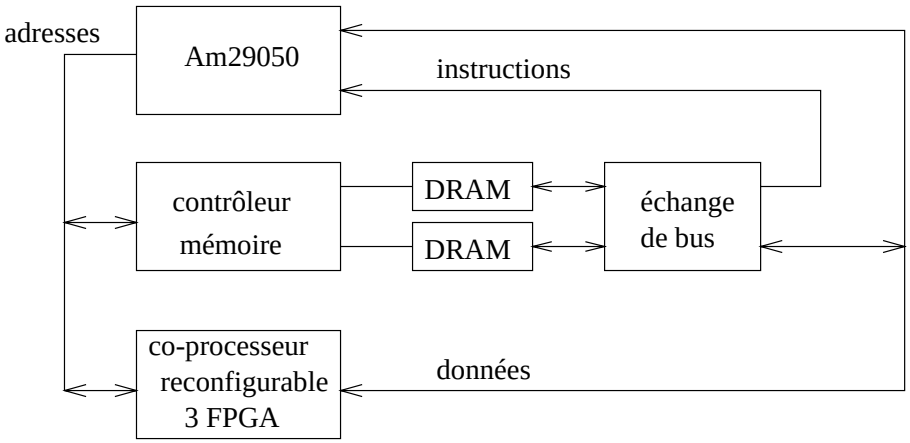

FIG. 7 - *Structure simplifiée de la machine* PRISM II *: une ressource composée de 3 circuits* FPGA *est accessible sur le bus mémoire du processeur. Les échanges de données processeur-*FPGA *sont effectués par l'intermédiaire des instructions co-processeur disponibles dans le jeu d'instruction du processeur* AMD.

manipulations de bits. L'implantation d'un algorithme génétique sur un nœud PRISM [SWSS95] a montré une accélération d'un facteur 3 par rapport à une station Sun SS20-61, malgré la lenteur relative du processeur AMD considéré isolément.

Accélérateur périphérique : la carte PERLE-1 L'architecture PAM (Programmable Active Memory), développée par le laboratoire PRL de DEC [VBR+96], a été conçue en 1988 dans l'un des premiers projets visant à l'utilisation de circuits FPGA comme accélérateur de calcul. Une PAM est une surface reconfigurable régulière composée d'une matrice de composants FPGA. Un bus d'entrées/sorties (TurboChannel pour les stations DEC) supporte les communications avec la machine hôte. Afin de réduire le nombre d'échanges sur ce bus, dont la latence est élevée, les architectures PAM disposent localement d'une quantité importante de mémoire RAM rapide.

La machine PERLE-1 (voir la figure 8) est une implantation de l'architecture PAM : une carte est composée d'un tableau de 16 circuits FPGA Xilinx 3090, de 4 MOctets de RAM statique, et de 7 circuits FPGA utilisés pour gérer les échanges avec la station hôte, la configuration du système et les accès aux mémoires.

Considérant la justification de la PAM en termes de performance de calcul, ses concepteurs ont basé les outils de programmation sur une description architecturale de bas niveau. Le langage C++ est utilisé pour cela ; il permet l'utilisation de librairies de composants génériques, et admet des directives relatives de placement à l'intérieur de la matrice de blocs logiques du composant FPGA.

Un effort applicatif important caractérise ce projet, il a permis de démontrer l'intérêt de l'architecture PAM sur des problèmes variés. Des applications dans les domaines

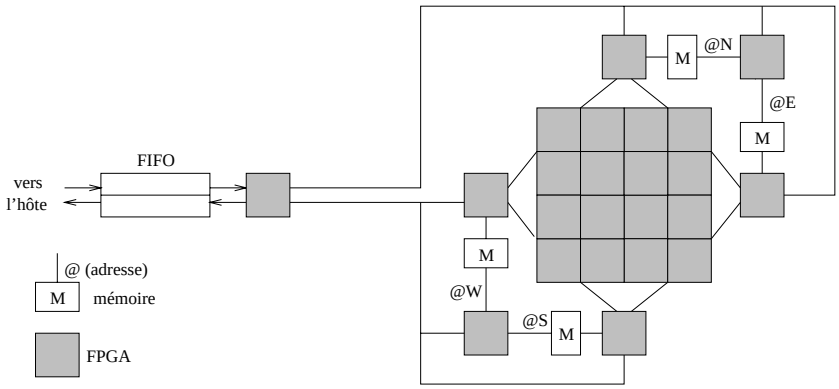

FIG. 8 - *Structure de la machine* PERLE-1. *La* PAM *dispose de plusieurs ports d'accès vers des bancs mémoires indépendants. Une mémoire* FIFO *assure la régulation des échanges entre le processeur et le tableau de circuits* FPGA.

du traitement d'image, de l'arithmétique non-standard, la cryptographie, etc. ont montré des performances du même ordre que celles fournies par des super-calculateurs [BRV93].

3. Les outils de programmation

Si le support matériel des architectures configurables a atteint aujourd'hui une certaine maturité, il n'en va pas de même pour les outils de programmation.

Les outils de CAO électronique et les langages traditionnellement utilisés pour décrire et implanter des circuits logiques, n'ont pas été développés dans une optique adaptée aux besoins des machines configurables. La plupart des programmeurs « informaticiens » ne sont pas préparés à raisonner par rapport aux « boîtes à outils » de l'électronique numérique. Les méthodes de développement employées sont différentes de celles pratiquées dans un projet informatique classique : plus longue et moins souple, la conception de matériel ne permet pas une élaboration incrémentale à l'image de celle d'un logiciel.

À l'heure actuelle, il n'existe pas encore d'outils standard ayant réellement fait leurs preuves pour générer automatiquement, à partir d'une description de haut niveau, des implantations pour des machines reconfigurables. Il s'agit encore d'un domaine de recherche en pleine croissance et de nombreuses voies sont explorées, eu égard à la diversité des architectures reconfigurables proposées. Sur une thématique relativement nouvelle, cette exploration tout azimut est incontournable et nécessaire. Elle reflète la vitalité et le dynamisme de la communauté scientifique sur cette activité, et devrait permettre, au fil du temps, de dégager quelques principes de base qui serviront de fondements aux travaux futurs.

Dans ce foisonnement d'idées et de propositions, nous regroupons les propositions

de langage de programmation des machines reconfigurables suivant quatre directions : les formalismes de haut niveau, les langages parallèles, les langages impératifs ou les langages de description matérielle. Le but est, bien sûr, de dériver une architecture qui exploite au mieux les caractéristiques des machines reconfigurables et des composants FPGA associés.

Formalismes de haut niveau

À ce stade, le comportement d'une application est défini à travers un formalisme qui met en avant le parallélisme potentiel, sans en décrire l'organisation spatiale ni temporelle. Par exemple, le modèle d'automates cellulaires mis en œuvre sur la machine ARMEN [CPR92] spécifie localement les calculs et les dépendances sur les nœuds reconfigurables. À partir de cette description, un réseau d'opérateurs est automatiquement synthétisé et implanté sur la ressource reconfigurable de la machine.

L'implantation du formalisme GAMMA sur une PAM [BLV94] est un autre exemple. Le modèle GAMMA s'illustre par une métaphore basée sur des réactions chimiques : la structure de données est un multi-ensemble et les calculs sont représentés comme une suite de réactions consommant et produisant des éléments nouveaux en fonction de règles spécifiques. L'architecture résultante est un ensemble d'opérateurs fonctionnant en pipeline. Ils sont synthétisés directement à partir des règles élémentaires de GAMMA.

Ces deux exemples ne sont pas issus de projets isolés, il en existe bien d'autres qui adoptent cette solution. Ils sont cependant assez représentatifs pour mesurer les avantages et les inconvénients de cette approche. La spécification est précise et l'architecture cible est fixée d'avance, ce qui facilite grandement la synthèse et permet d'aboutir à des implantations fiables et efficaces. D'un autre coté, la gamme d'applications est restreinte.

Langages parallèles

À travers les langages parallèles, on cherche à organiser le parallélisme des applications. La partie reconfigurable de la machine est vue comme un système parallèle à grain fin ; les langages et les paradigmes développés pour ce type de système peuvent alors être adaptés. Des extensions du langage C pour la programmation de machines systoliques (spC) ou des machines SIMD (dbC [Kim96]) sont proposées. D'autres compilateurs travaillent à partir de programmes flot de données.

Les langages choisis favorisent l'expression du parallélisme de données de l'application. Ce faisant, la synthèse de l'accélérateur matériel se fonde sur la duplication spatiale d'une même cellule de calcul. Ce schéma est particulièrement adapté aux architectures comme SPLASH ou ARMEN, composés de réseaux linéaires de circuits FPGA.

Langages impératifs

Dans cette catégorie (PRISM [AS93], Garp [BRA96], Spyder [IS95]) on admet un langage séquentiel de programmation commun au processeur et au matériel configurable. Il s'agit d'une approche de conception concurrente (co-design). Les langages C, C++ ou des sous-ensembles sont les plus souvent proposés.

Le flot de données et le flux de contrôle de sections de code C sont extraits et servent

de guide à la synthèse matérielle : des modules correspondant aux opérateurs utilisés sont implantés dans le matériel configurable et validés en fonction du flot de contrôle.

Cette approche est justifiée par l'unicité du modèle de programmation, qui est identique pour le processeur et la ressource configurable. Elle semble cependant limitée par le parallélisme effectif du circuit logique dont la synthèse est réalisée à partir de sections du code séquentiel.

Langages de description matériel

À ce niveau, on rejoint le monde de la CAO électronique, même si une démarche de « sociabilisation » de ces outils est tentée en maquillant les descriptions matérielles derrière des langages de programmation. Dans tous les cas, on décrit un circuit numérique, ce qui exige de la part du programmeur de sérieuses connaissances en architecture de machines informatiques.

Ces langages (PAM programming tools [BT94], Ruby [Sin95], LoLa [Wir95], ...), même si certains visent également la synthèse de circuits VLSI, incluent des propriétés indispensables pour exploiter au mieux les ressources des composants FPGA. Par exemple, pouvoir diriger le placement-routage limite considérablement les temps de calcul consacrés à cette tâche. Dans un contexte de compilation automatique, limiter la durée de cette action est certainement tout aussi important que d'optimiser l'usage des ressources internes des composants FPGA.

Programmer un composant FPGA à l'aide de ces langages est du même ordre de complexité que la programmation en assembleur. Aujourd'hui, faute de mieux, et pour être efficace, ils sont largement employés. Demain, ils constitueront sans doute un niveau intermédiaire et obligé, produits à partir d'outils de compilation de plus haut niveau.

À terme, le but est de fournir un environnement de programmation qui rende transparent l'usage de la partie reconfigurable. C'est ici que se situe le véritable enjeu : l'architecture dérivée ne doit pas seulement être efficace (accélérer notablement le calcul), elle doit aussi être compilée rapidement. C'est à ce prix que les machines reconfigurables trouveront leur place.

Conclusion

L'étude des implications de l'association processeur-FPGA reste encore un sujet ouvert ; la multiplicité des projets en cours en témoigne. Nous avons résumé, dans ce panorama, l'éventail des solutions envisagées. Périodiquement, de nouvelles équipes de recherche débutent le développement d'architectures et d'environnements de programmation pour ce type de machines. La diversité des projets signale un domaine de recherche dynamique, mais elle est aussi la marque d'un manque de maturité. Très peu de produits industriels sont commercialisés, et leurs systèmes de développement ne sont pas satisfaisants du point de vue énoncé dans ce papier.

Les machines à matériel configurable actuelles ne constituent pas encore une alternative crédible aux stations de travail construites autour de processeurs « classiques ».

Le potentiel de puissance de calcul lié aux ressources configurables est important, mais son exploitation exige une spécialisation poussée des circuits synthétisés en fonction des programmes qui s'exécutent sur la machine. Si cette caractéristique n'est pas remplie, les structures de traitement hautement optimisées des processeurs continueront à imposer leur supériorité. Le véritable défi pour la diffusion des unités de calcul configurable est donc celui des outils de synthèse. Ils doivent répondre à des contraintes antagonistes liées au niveau du langage de programmation, à la vitesse de la compilation matérielle, et à la forme du circuit synthétisé.

En revanche, les machines disposant d'un tableau de circuits FPGA ont déjà démontré leur utilité et leur potentiel ; elles peuvent produire des puissances de calcul du même ordre que celles des super-calculateurs, à un coût nettement inférieur. La taille des circuits logiques à créer en font des machines semi-dédiées, et la problématique des outils de programmation s'en trouve changée. Pour des applications nécessitant des phases de calcul intensif sur un grand nombre de données, la ressource configurable leur permet (1) un parallélisme de traitement efficace, par exemple sous la forme de pipelines très profonds, et (2) un débit important d'échange avec la mémoire. Ces machines peuvent, dans certaines situations, remplacer des super-calculateurs parallèles.

Ainsi, la puissance de calcul produite par les circuits FPGA provient de leur spécialisation par rapport aux besoins des programmes exécutés. Cependant, ces besoins changent selon les phases algorithmiques en cours. De plus, il est très courant que plusieurs programmes utilisent en temps partagé les mêmes ressources matérielles. Dans une machine à matériel configurable, la notion de *contexte matériel* apparaît. Le temps de changement du contexte matériel doit être pris en compte : ce contexte représente un volume d'informations de plusieurs centaines de KOctets pour les gros circuits FPGA, comme nous l'avons montré dans la première partie de l'article. De ce point de vue, la disponibilité de mécanismes de configuration partielle constitue une caractéristique primordiale. La configuration partielle est une technique permettant de modifier une partie du circuit implanté, tout en conservant une autre partie dans un état fonctionnel. Avec ce mécanisme, le chargement d'une nouvelle configuration peut s'effectuer en recouvrement, sans interrompre l'activité opérationnelle du circuit.

La configuration partielle ouvre, de plus, des perspectives de spécialisation très poussée, par une adaptation dynamique des circuits en fonction des données traitées dans le programme. Par exemple, un algorithme de recherche de motif peut être transformé en une machine dédiée à la recherche d'un motif particulier ; le motif à rechercher est alors « intégré » dans le circuit lui-même. Les possibilités offertes restent encore à explorer ; les schémas de pensée traditionnels de la conception matérielle doivent être revus, eu égard aux nouvelles hypothèses qu'impliquent la configuration partielle. Le support technologique pour de telles recherches est disponible commercialement, notamment par la famille Xilinx 6200 ; en revanche, les méthodologies de conception et d'intégration système, nécessaires à l'utilisation effective de cette technologie, ne sont pas encore définies.

Références

[AS93] Athanas (P. M.) et Silverman (H. F.). – Processor Reconfiguration Through Instruction-Set Metamorphosis. *IEEE Computer*, mars 1993.

[BLV94] Banâtre (J.P.), Lavenier (D.) et Vieillot (M.). – From High Level Programming Model to FPGA Machines. *In : IEEE workshop on FPGAs for Custom Computing Machines*. – Napa, CA, avril 1994.

[BRA96] BRASS (Projet). – Garp: Combining a processor with a reconfigurable gate array. – http://www.cs.berkeley.edu/projects/brass, novembre 1996.

[BRV89] Bertin (P.), Roncin (D.) et Vuillemin (J.). – Introduction to Programmable Active Memories. *In : Systolic Array Processors*, éd. par McCanny (J.), McWhirter (J.) et Swartzlander (E.), pp. 301–309. – Prentice/Hall, 1989.

[BRV93] Bertin (P.), Roncin (D.) et Vuillemin (J.). – *Programmable Active Memories: a Performance Assessment*. – Rapport technique n° 24, DEC PRL, mars 1993.

[BT94] Bertin (P.) et Touati (H.). – PAM Programming Environments: Practice and Experience. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Buell (D.A.) et Pocek (K.L.). – NAPA, avril 1994.

[CPR92] Champeau (J.), Pottier (B.) et Rubini (S.). – Synthèse d'automates cellulaires sur la machine ArMen. *In : Workshop Synthèse et compilation d'architectures : méthodes de spécification et de compilation*. AFCET, GCIS et PRC ANM, p. 15. – Grasse, France, Janvier 1992.

[FT93] French (P. C.) et Taylor (R. W.). – A Self-reconfiguring processor. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Buell (D. A.) et Pocek (K. L.). – Napa, avril 1993.

[GCM94] Gupta (R. K.), Coelho (C. N.) et Micheli (G. De). – Program Implementation Schemes for Hardware-Software Systems. *IEEE Computer*, vol. 27, n° 1, janvier 1994, pp. 48–55.

[GHK+91] Gokhale (M.), Holmes (W.), Kopser (A.), Lucas (S.), Minnich (R.), Sweely (D.) et Lopresti (D.). – Building and Using a Higly Parallel Programmable Logic Array. *Computer*, vol. 24, n° 1, 1991, pp. 81–88.

[HKL+95] Högl (H.), Kugel (A.), Ludvig (J.), Männer (R.), Noffz (K.H.) et Coz (R.). – Enable++: A second generation FPGA processor. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Athanas (P.) et Pocek (K. L.). – Napa, avril 1995.

[IS94] Iseli (C.) et Sanchez (E.). – Augmentation du parallélisme par la reconfigurabilité. *In :* $6^{ièmes}$ *Rencontres Francophones du Parallélisme*, éd. par Bougé (L.), Cosnard (M.) et Fraigniaud (P.), pp. 3–6. – Lyon, France, juin 1994.

[IS95] Iseli (C.) et Sanchez (E.). – A C++ Compiler for FPGA Custom Execution Units Synthesis. *In : Proceeding of IEEE Workshop FPGAs for Custom Computing Machines*, éd. par Athanas (P.) et Pocek (K.). – Napa, California, avril 1995.

[Kim96] Kim (J.D.). – Implementing Data Parallel C on a Custom Computing Machine. *In : Proceedings of MASPLAS'96*.

[LLP93] Luk (W.), Lok (V.) et Page (I.). – Hardware Acceleration of Divide-and-Conquer Paradigms: a Case Study. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Buell (D. A.) et Pocek (K. L.). – Napa, avril 1993.

[Pot91] Pottier (B.). – ArMen : une machine parallèle intégrant un réseau de circuits logiques programmables. – Thèse de l'Université de Rennes I, Juin 1991.

[RESV93] Rose (J.), El Gamal (A.) et Sangiovanni-Vincentelli (A.). – Architecture of Field-Programmable Gate Arrays. *Proceedings of the IEEE*, vol. 81, n° 7, juillet 1993.

[Rub95] Rubini (S.). – Intégration des FPGA dans les architectures MIMD. – Thèse de l'université de Rennes 1, juillet 1995.

[Sin95] Singh (S.). – Architectural Descriptions for FPGA Circuits. *In : Proceeding of IEEE Workshop FPGAs for Custom Computing Machines*, éd. par Athanas (P.) et Pocek (K.). – Napa, California, avril 1995.

[SWSS95] Sitkoff (N.), Wazlowski (M.), Smith (A.) et Silvermen (H.). – Implementing a Genetic Algorithm on a Parallel Custom Computing Machine. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Athanas (P.) et Pocek (K. L.). – Napa, avril 1995.

[VBR+96] Vuillemin (J), Bertin (P.), Roncin (D.), Shand (M.), Touati (H.) et Boucard (P.). – Programmable Active Memories: Reconfigurable Systems Come of Age. *IEEE Transactions of VLSI Systems*, vol. 4, n° 1, mars 1996.

[WAL+93] Wazlowski (M.), Agarwal (L.), Lee (T.), Lam (S. E.), Athanas (P.), Silverman (H.) et Ghosh (S.). – PRISM-II Compiler and Architecture. *In : IEEE Workshop on FPGAs for Custom Computing Machines*, éd. par Buell (D. A.) et Pocek (K. L.). – Napa, avril 1993.

[Wir95] Wirth (N.). – *Digital Circuit Design. An Introductory Textbook*. – Springer, 1995.

[Wit95] Wittig (R. D.). – *OneChip : An FPGA Processor With Reconfigurable Logic*. – Canada, Thèse, University of Toronto, 1995.