SympA'7
Paris, 24 - 27 Avril 2001
7ème Symposium sur les Architectures Nouvelles de Machines

Stratégie de placement de réseaux linéaires sur circuits FPGA

Erwan Fabiani et Dominique Lavenier

IRISA, Campus de Beaulieu, 35042 Rennes Cedex
efabiani,lavenier@irisa.fr

Résumé

Cet article présente une méthode pour placer des réseaux linéaires de processeurs sur des composants FPGA. Cette méthode est basée sur la notion de placement régulier : les propriétés de régularité et de localité sont conservées pendant l'implémentation physique. Nous montrons comment cette méthode permet de placer rapidemment un nombre maximum de processeurs en respectant la proximité physique entre processeurs voisins, et ce, quelque soit la position des extrémités du réseau linéaire sur le circuit FPGA. Ce placement évite la phase de placement des outils commerciaux et réduit la durée de la phase de routage.

1 Introduction

Dans la majorité des applications de calcul intensif, telles que le traitement de signal ou d'image, le temps est principalement passé dans l'exécution des boucles (usuellement à 90 %). Une méthode pour accélérer ces applications consiste à en dériver une implémentation matérielle qui exploite directement le parallélisme inhérent aux boucles. L'architecture résultante est un réseau régulier, souvent systolique, fait de processeurs élémentaires dédiés à l'exécution efficace du corps des boucles internes [1]. La structure peut être en 1 ou 2 dimensions, mais par la suite nous nous restreindrons aux réseaux réguliers unidimensionnels. Cette restriction est justifiée par la possibilité de transformation par sérialisation d'un réseau bidimensionnel en réseau unidimensionnel. De plus, l'implémentation d'un réseau bidimensionnel sans sérialisation peut entraîner des besoins excessifs en débit d'entrées/sorties.

Implémenter de tels nids de boucles sur des composants FPGA présente plusieurs avantages. D'abord, la nature régulière des composants FPGA (une matrice de blocs de calcul de niveau bit) est en parfaite adéquation avec les architectures que nous manipulons : une réplication de processeurs identiques interconnectés régulièrement. Ensuite, la meilleure utilisation des cartes FPGA (d'un point de vue performances) a été démontrée pour de nombreuses applications de calcul intensif, comme par exemple sur les cartes PAM [2]. Enfin, les nouvelles architectures de processeurs tendent à incorporer des ressources reconfigurables dans leur chemin de données. Paralléliser des boucles sur ces composants spécifiques est un moyen très attractif d'exploiter efficacement le calcul reconfigurable.

Nous pensons que même si l'implémentation d'architectures haute performance sur plate-forme reconfigurable est actuellement techniquement faisable, la principale restriction vient des outils de programmation. En effet, le niveau de programmation est comparable à celui du langage assembleur utilisé pour les premières machines. De plus, réaliser une implémentation efficace demande une très bonne connaissance de la structure du composant FPGA, ainsi qu'une longue expérience pour piloter la chaîne complète d'outils nécessaires à la création d'un circuit. Sans dévaloriser la recherche de nouvelles architectures reconfigurables, qui doit continuer à être imaginative, nous estimons que

leur succès est aussi lié à l'aptitude à fournir des outils de programmation de haut niveau. Notre travail est axé dans cette direction.

Plus précisément, le projet de recherche en calcul reconfigurable de l'équipe COSI de l'IRISA vise l'automatisation de la parallélisation de nids de boucles sur carte FPGA[3]. Cela passe par trois étapes majeures :

- **Parallélisation :** cette étape consiste à dériver des architectures de réseaux réguliers (systoliques ou semi-systoliques) à partir de spécifications de boucles ou de description formelle équivalente tels que les systèmes d'*équations récurrentes affines*. Le langage ALPHA, basé sur ce modèle et développé à l'IRISA, permet au programmeur d'explorer les transformations nécessaires à la dérivation systématique de réseaux réguliers pour la parallélisation automatique [4] [5].
- **Partitionnement :** puisque les ressources reconfigurables disponibles sont physiquement limitées et ne peuvent pas contenir la totalité du réseau régulier, une transformation de l'architecture est nécessaire. Elle consiste à découper le réseau en sous-réseaux, ou à regrouper des processeurs dans une même entité. L'automatisation de cette tâche n'est pas encore totalement résolue.
- **Implémentation physique :** cette dernière étape implémente l'architecture sur le support reconfigurable. A partir d'une description RTL (issue des étapes précédentes), on doit trouver la meilleure implémentation en termes de vitesse et de surface occupée. C'est en fait une étape très coûteuse en temps, qui tend à s'allonger avec la complexité croissante des composants FPGA.

Le travail présenté dans cet article traite de la dernière partie. Il se focalise sur la réduction du temps d'exécution du processus de placement-routage, en tirant partie de la nature régulière des réseaux linéaires.

La partie suivante expose d'abord les principes et les avantages du placement régulier. La partie 3 explicite notre stratégie pour placer des réseaux réguliers sur circuit FPGA. La partie 4 conclue.

2 Principes et avantages du placement régulier

Le processus de placement-routage est généralement effectué avec les outils commerciaux associés à la famille du composant FPGA ciblé, car l'accès aux détails bas-niveau de l'architecture est restreint. L'entrée des outils est souvent une description architecturale RTL à plat (i.e. sans structure hiérarchique), et le but est de trouver la meilleure adéquation entre la structure du composant FPGA et le circuit à implémenter. Cela consiste à répartir d'abord les équations booléennes et les registres dans les blocs logiques élémentaires du composant FPGA (dénommés BL par la suite) de telle sorte que les longueurs des connexions soient minimisées. Puis vient la phase de routage pour connecter ces blocs logiques entre eux. Généralement, la rapidité du routage est proportionnelle à la qualité du placement.

Cependant ces deux étapes sont très coûteuses en temps, particulièrement avec les plus gros composants FPGA. Cela est essentiellement dû aux algorithmes (tels que le recuit simulé) utilisés pour trouver des solutions satisfaisantes. L'avantage de ces techniques est leur généralité : elles obtiennent de relativement bons résultats quelque soit la structure des circuits. Mais lorsqu'il s'agit d'une structure régulière, elles sont incapables d'exploiter les informations de localité et de régularité qui seraient utile à l'implémentation. Ainsi, quand dans le meilleur des cas elle parviennent à retrouver un semblant de régularité c'est au prix de longues heures de calcul. Au contraire, il faut profiter des informations structurelles intrinsèques aux réseaux réguliers pour minimiser le temps de conception.

Un des moyens pour limiter ce temps est de fournir le meilleur placement en vue de réduire l'étape de routage. La méthodologie que nous avons développée pour implémenter des réseaux linéaires sur composant FPGA est principalement basée sur cette idée. Notre thèse est que placer

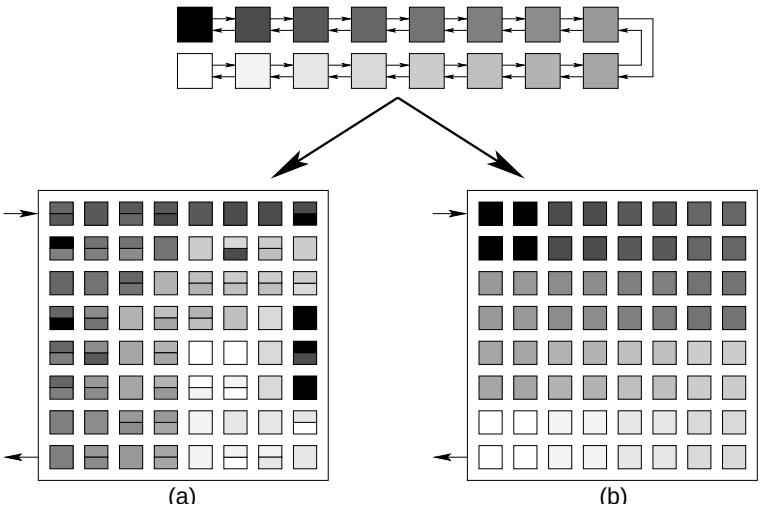

FIG. 1 – *Comparaison de 2 placements d'un réseau linéaire* (haut de la figure) *:* (a) *sans placement prédéfini ;* (b) *avec placement prédéfini. Un processeur nécessite 4 blocs logiques. En* (a)*, la régularité intrinsèque du réseau est partiellement respectée, mais quelques processeurs sont répartis dans des blocs logiques distants et les processeurs voisins dans le réseau ne sont pas physiquement proches. En* (b) *un processeur est implémenté par 4 blocs logiques adjacents, et la proximité physique entre les processeurs voisins est scrupuleusement respectée.*

un réseau linéaire de processeurs en respectant ses propriétés de régularité et de localité amène trois avantages majeurs par rapport aux techniques de placement-routage usuelles :

1. le temps de placement est considérablement réduit;
2. le temps de routage est optimisé;
3. la fréquence d'horloge est augmentée;

Notre stratégie de placement pour exploiter ces avantages est basée sur les principes suivants :

1. Les signaux qui sont issus d'un même processeur ont leurs sources placées dans une même surface restreinte.
2. Les processeurs identiques ont un placement identique : le placement se focalise uniquement sur le placement d'un processeur qui est ensuite répliqué sur tout le composant FPGA. Le temps de placement est donc indépendant du nombre de processeurs.
3. Les processeurs voisins sont physiquement proches.

La figure 1 illustre notre stratégie de placement : en plus de refléter la régularité du réseau linéaire, chercher le meilleur placement d'un processeur et le répliquer sur le FPGA est évidemment plus rapide que de chercher un placement global.

Quelques expérimentations ont été menées pour valider cette idée. Elles sont résumées dans les figures 2 et 3. Nous comparons les temps d'exécution pour placer-router un circuit avec ou sans directives de placement. Les estimations de fréquence d'horloge sont aussi comparées.

Dans la figure 2, l'unité de comparaison est donnée par le temps de placement-routage d'un circuit sans directives de placement. Dans la figure 3, on compare les accélérations induites par l'ajout de directives de placement. Cette comparaison est faite pour le temps de placement, le temps de routage, le temps de placement-routage et la fréquence d'horloge.

Les circuits sont de structure très régulière et issus d'applications réelles :

– **Lyon** est une implémentation sous forme de réseau systolique du multiplieur bit-série de Lyon [6]. La longueur du réseau est égale à la taille des chiffres manipulés.

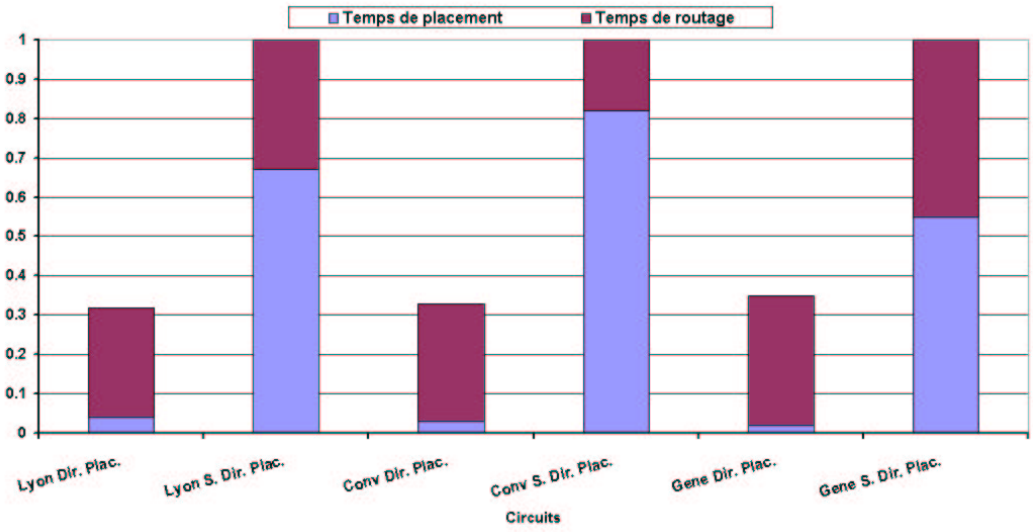

FIG. 2 – *Comparaison sur une échelle d'unité normalisée du temps moyen pour placer-router un circuit avec (Dir. Plac.) ou sans (S. Dir. Plac.) directives de placement.*

- **Conv** est une implémentation systolique d'une pseudo-convolution sur 16 bits dans laquelle le multiplieur est remplacé par un AND.
- **Gene** implémente la partie "mutation" d'un algorithme génétique systolique [7]. Un processeur élémentaire sur 8 bits est composé d'un registre à décalage, d'un accumulateur, d'un comparateur, et de trois bancs de registres.

Les circuits *Lyon, Conv, Gene* sont implémentés sur un Xilinx XC4020 [8], en utilisant l'outil de placement-routage PPR de Xilinx. Les valeurs indiquées sont des valeurs moyennes : pour chaque circuit, plusieurs placement-routage ont été effectués avec différentes options d'effort.

Dans ces exemples, on observe que la phase de placement est plus coûteuse en temps que la phase de routage, et que réduire cette étape entraîne un gain significatif, même si la phase de routage est augmentée dans certains cas.

Cependant, ces résultats expérimentaux ne doivent pas être considérés comme des valeur définitives, mais plutôt comme des tendances confirmant notre idée de départ. Ils indiquent qu'une phase de pré-placement est intéressante dans le cas des architectures régulières, et que cela n'entraîne pas de dégradation de la fréquence d'horloge.

3 Stratégie de placement régulier

3.1 Le placeur de réseaux réguliers sur FPGA

L'environnement de placement de réseaux réguliers sur circuits FPGA est détaillé dans la figure 4. La description du réseau linéaire en entrée est de niveau RTL sans aucune directive de placement, typiquement un tableau d'instances de processeurs connectés entre eux par des signaux communs. Le placement régulier est effectué par notre outil (FRAP) en 3 étapes :

1. Toutes les formes possibles pour un processeur sont générées en combinant les formes disponibles pour les sous-composants du processeur.
2. Un placement régulier en forme de serpentin est déterminé, en utilisant les formes possibles d'un processeur précédemment calculées.

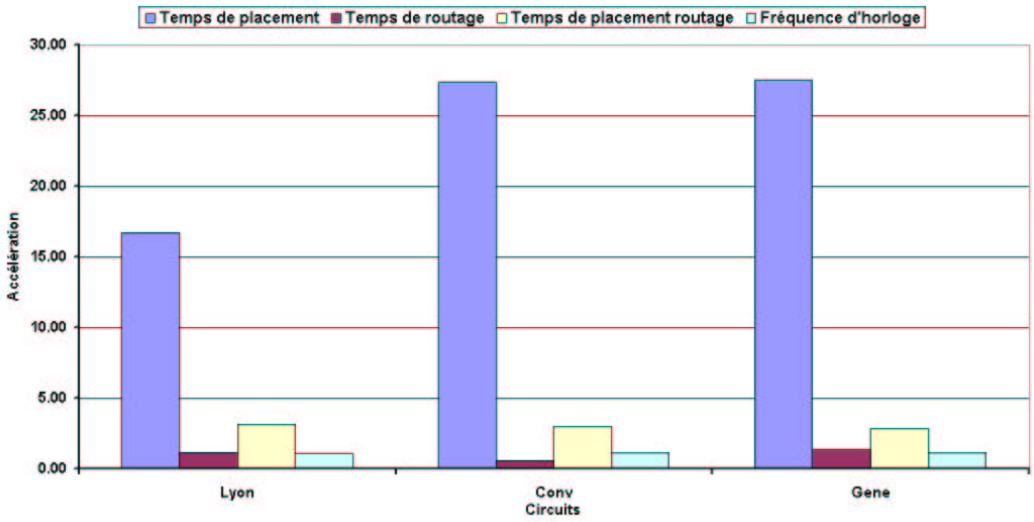

FIG. 3 – *Comparaison des accélérations en temps d'exécution et en fréquence d'horloge induites par l'ajout de directives de placement régulier à la description du circuit.*

3. Le placement interne des processeurs est calculé en fonction des formes choisies.

Dans la première étape, il s'agit de trouver toutes les formes possibles pour un processeur. Comme les réseaux réguliers implémentés sont linéaires, c'est à dire avec un flot de données unidimensionnel, on distingue les deux dimensions d'un processeur : la dimension parallèle au flot de données (hauteur) et la dimension perpendiculaire au flot de données (largeur). Chaque opérateur utilisé par le processeur a, en fonction de sa taille, différentes formes possibles : par exemple un opérateur de 12 blocs logiques peut être implémenté sous la forme d'une colonne de 12 blocs logiques, ou de 2 colonnes de 6 blocs logiques. Ce nombre de formes possibles est cependant limité : on privilégie la hauteur à la largeur, on ne sélectionne pas les formes qui laissent des blocs logiques inutilisés, et certains opérateurs on des formes fixes (par exemple si ils utilisent des mécanismes de retenue rapide). Pour générer les formes possibles d'un processeur, on fait varier sa hauteur et on sélectionne les formes d'opérateurs inférieures ou égales à cette hauteur. Puis on calcule la largeur correspondante. Pour chaque largeur possible d'un processeur on sélectionne la forme de largeur minimale afin de minimiser la surface.

L'étape 3 est essentiellement liée au placement d'un processeur dans une surface restreinte. Nous supposons que les processeurs sont composés d'un petit nombre d'opérateurs instanciés d'une bibliothèque. Comme le placement à trouver est au niveau opérateur et non pas bloc logique, le temps nécessaire pour trouver ce placement n'est pas critique. La principale différence par rapport à un problème classique de *floorplanning* de modules est qu'il faut prendre en compte la longueur des connexions entre processeurs voisins lors de la minimisation des délais du chemin critique.

L'étape 2, plus originale, est détaillée dans la partie suivante.

La sortie de FRAP est une description RTL du circuit avec des directives de placement, dont la structure est éventuellement modifiée (création de sous-tableaux de processeurs) pour faciliter l'expression du placement, mais reste équivalente à la description initiale. Un fichier EDIF, généré à partir de cette description, sert de point d'entrée vers les outils commerciaux de placement-routage. Comme le placement y est complètement spécifié, le temps d'exécution est pratiquement réduit au temps nécessaire pour router le circuit.

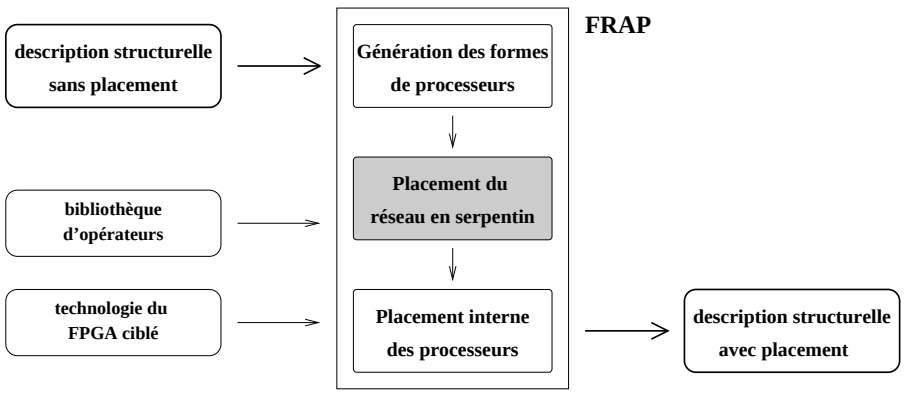

FIG. 4 – *Le placeur de réseaux réguliers (*FRAP*): la description structurelle en entrée est placée et sort annotée de directives de placement.*

3.2 Stratégie de placement du réseau linéaire en serpentin

Le problème ici est de placer un réseau linéaire de processeur sur la matrice de blocs logiques d'un composant FPGA. Le seul moyen pour conserver physiquement la proximité de 2 processeurs voisins est de placer le réseau linéaire selon une forme de serpentin. Le placement en serpentin est fait en 2 phases : (1) la surface de blocs logiques à utiliser est divisée en sous-surfaces, que l'on appellera *surface de base,* et (2) un maximum de processeurs est placé dans chaque surface de base selon une forme de serpentin.

Partitionnement en surfaces de base :

Cette étape est nécessaire pour prendre en compte les contraintes d'entrées/sorties du support reconfigurable. En effet, les premier et dernier processeurs du réseau linéaire doivent être proches de blocs d'entrées/sorties situés sur les bords du circuit FPGA. L'utilisateur indique la position souhaitée des premier et dernier processeurs en associant à chacun d'eux les coordonnées d'un bloc logique.

Une surface de base est définie comme une surface rectangulaire dans laquelle les positions des premier et dernier processeurs sont situées dans deux coins différents du rectangle. Si initialement la surface disponible pour le placement du réseau linéaire n'est pas une surface de base (c'est à dire que les contraintes données par l'utilisateur ne permettent pas le positionnement des processeurs extrêmes dans des coins), alors cette surface doit être divisée en surfaces de base.

Le partitionnement en surfaces de base est effectué en analysant les positions des premier et dernier processeurs, puis en appliquant l'opération de découpage correspondante. Trois cas sont distingués :

- **cas 0 :** les processeurs extrêmes sont positionnés dans deux coins différents. C'est une surface de base donc aucun découpage n'est nécessaire.
- **cas 1 :** un seul processeur extrême est positionné dans un coin. Appliquer le découpage D1.
- **cas 2 :** aucun processeur extrême n'est positionné dans un coin. Appliquer le découpage D2.

Une opération de découpage (de type D1 ou D2) consiste à diviser la surface en deux nouvelles sous-surfaces. Dans chacune de ces nouvelles sous-surfaces, au moins un des processeurs extrêmes est situé dans un coin. Pour chaque nouvelle sous-surface un processeur extrême virtuel est créé, toujours dans un coin. Il symbolise le premier ou dernier processeur du fragment de réseau qui sera implémenté dans la surface. Le processus est appliqué récursivement aux sous-surfaces jusqu'à n'avoir plus que des surfaces de base. La figure 5 donne quelques exemples représentatifs mais non exhaustifs du processus. Puisqu'une surface identifiée au cas 2 est divisée en sous-surfaces de cas 0

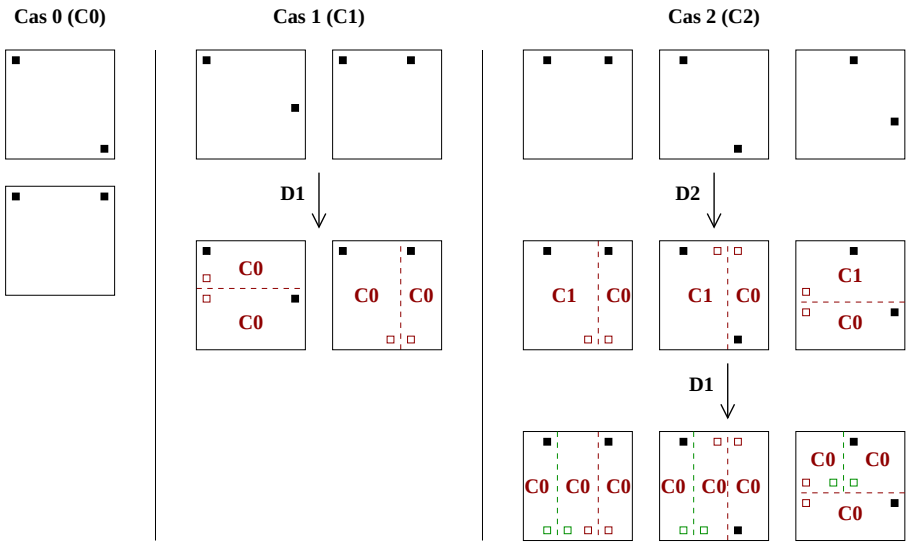

FIG. 5 – *Partitionnement en surfaces de base. Trois cas sont considérés selon la position des processeurs extrêmes. Un découpage (D1 ou D2) divise la surface en deux sous-surfaces de telle sorte qu'au moins un des processeurs extrêmes soit situé dans un coin d'une sous-surface.*

ou 1, et qu'une surface de cas 1 est divisée en sous-surfaces de cas 0, le partitionnement en surfaces de base est réalisé en deux itérations au plus.

Placement en serpentin:

Un serpentin est défini comme un empilement de segments. Un segment est une ligne ou une colonne de processeurs si l'orientation du serpentin est respectivement horizontale ou verticale. La parité du nombre de segments implique que les processeurs extrêmes soient situés sur un même bord ou sur des bords opposés.

Dans un premier temps, l'orientation et la parité du nombre de segments sont déterminés en fonction de la position des processeurs extrêmes (voir figure 6). Ainsi le serpentin est caractérisé par ces deux paramètres. Le problème est alors d'implanter un nombre maximal de processeurs dans la surface de base en respectant ces paramètres. Pour une plus grande flexibilité et pour limiter la surface inutilisée, les segments d'un même serpentin peuvent être composés de processeurs de forme différente, une seule forme étant autorisée par segment.

Ce problème d'optimisation est résolu en utilisant l'analogie avec le problème du sac à dos : pour un sac à dos d'une capacité pondérale fixée et pour un ensemble d'objets caractérisés par leur poids et leur profit, trouver le sous-ensemble d'objets à emporter pour maximiser le profit sans excéder la capacité du sac à dos [9].

Dans notre cas, les objets sont les segments, le poids d'un objet est la hauteur d'un segment (la dimension perpendiculaire au flot de données), le profit correspond au nombre de processeurs d'un segment, et la capacité du sac à dos est la hauteur de la surface de base (voir figure 7.b). La diversité des objets (les segments) est garantie par la possibilité d'avoir différentes formes possibles pour un processeur.

Le problème du sac à dos est habituellement résolu par programmation dynamique, avec une fonction $f(j,k)$ calculée récursivement [10]. La variable j représente la capacité (la hauteur de la surface de base) et k le nombre d'objets (segments). Nous avons modifié la formule générale pour rejeter certaines solutions (comme une mauvaise parité du nombre de segments) et pour favoriser celles qui minimisent le nombre de segments de type différent : en effet, moins il y a de formes de processeurs utilisées et moins de temps il faudra pour trouver le placement interne des processeurs.

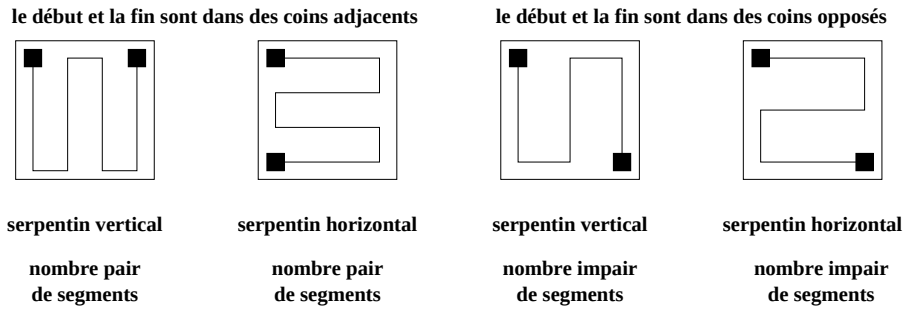

FIG. 6 – *Détermination de l'orientation du serpentin et de la parité du nombre de segments en fonction de la position des processeurs extrêmes.*

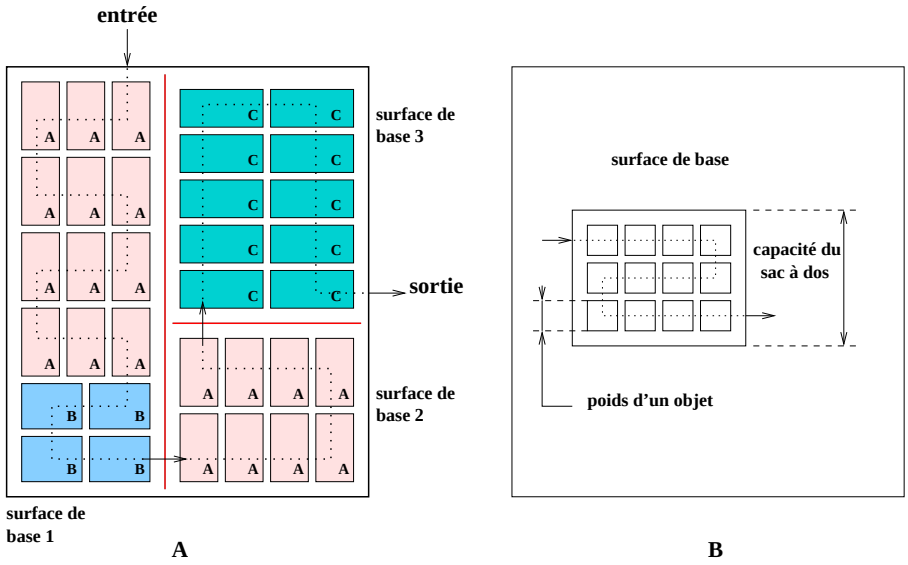

FIG. 7 – *A: Exemple de placement trouvé par* FRAP. *B: Analogie avec le problème du sac à dos.*

La figure 7.a montre un résultat du placement trouvé par FRAP. La surface totale du FPGA a été découpée en trois surfaces de base. Dans la surface de base 1 est placé un serpentin horizontal constitué de deux types de segments différents, les segments de forme A et de forme B. Dans la surface de base 2 est également placé un serpentin horizontal, mais uniquement constitué de segments de forme A. Dans la surface de base 3 est placé un serpentin vertical, fait de segments de type C. La phase suivante de placement interne des processeurs nécessitera donc de trouver un placement pour trois formes de processeurs.

3.3 Mise en oeuvre

FRAP est implémenté en JAVA. Il prend en entrée une description structurelle d'un réseau linéaire sans placement et produit une description équivalente avec un placement en serpentin.

Bien que FRAP soit toujours en développement, la version actuelle a permis de faire quelques tests et expériences. Notre technique de placement par serpentin produit une solution optimale dans un temps très court. Par exemple, le placement d'un réseau linéaire de 152 processeurs avec des processeurs de taille importante (75 BLs) sur une surface équivalente à celle d'un Virtex 1000

de Xilinx prend moins de 2 secondes sur une machine de type Sun UltraSparc à 295 mHz.

4 Conclusion

Nous avons présenté une stratégie pour placer des réseaux linéaires sur composants FPGA. Cette stratégie utilise la technique de résolution du problème du sac à dos et permet un placement rapide comparativement aux outils commerciaux. Ce gain de temps provient essentiellement de la nature régulière de l'architecture à implémenter, c'est à dire un réseau linéaire de processeurs identiques sur lequel on applique un placement sur 2 niveaux: (1) un placement au niveau processeur et (2) un placement au niveau réseau. Un outil de CAO, appelé FRAP, est en développement pour implémenter cette stratégie.

Cependant il s'avère qu'un placement régulier n'entraîne pas nécessairement d'accélération significative du temps de routage. Cela limite l'accélération totale du temps de placement-routage. Ainsi, en supposant que le placement d'un réseau régulier par notre outil soit instantané, l'accélération totale est directement proportionnelle au ratio du temps de placement par le temps de routage d'un circuit non placé régulièrement. Pour que la viabilité future de notre méthode ne soit pas uniquement liée à l'évolution de ce ratio en fonction des technologies et outils, il faudrait également appliquer le principe de réplication à la phase de routage. En effet, dans notre méthode, le placement d'un processeur est répliqué, mais le routage est fait globalement par l'outil commercial, sans tenir compte de cette régularité. Théoriquement, si des processeurs sont placés de manière identique, leur routage devrait aussi être identique, et donc réplicable. La méthode de "routage régulier" consisterait donc à faire router un processeur par l'outil commercial, puis à extraire ce routage et le répliquer pour tous les processeurs de même forme. Pour arriver à cet objectif, trois conditions doivent être réunies: (1) Le routage d'un groupe de blocs logiques doit être physiquement réplicable en tout point du FPGA; (2) la réplication du routage ne doit pas provoquer de conflits d'utilisation des ressources de routage; (3) disposer d'un mécanisme pour pouvoir répliquer un routage.

La première condition est liée à l'architecture de routage du composant FPGA. Cette architecture de routage doit être uniforme, c'est à dire que le type et le nombre de canaux de routage accessibles par un bloc logique doivent être identiques pour tous les blocs logiques. Cette contrainte est vérifiée pour les composants FPGA Xilinx actuels (exception faite des canaux de routage transversaux).

La deuxième condition implique que deux processeurs voisins ne doivent pas partager des canaux de routage. La seule solution pour garantir cela est de confiner le routage d'un processeur à l'intérieur d'un périmètre donné (typiquement le rectangle qui contient les blocs logiques d'un processeur). Les outils commerciaux permettent de contraindre un placement dans un périmètre donné, mais n'offrent pas de telles contraintes pour le routage.

Actuellement il n'existe aucun mécanisme simple pour répliquer un routage. Cependant, deux méthodes sont envisageables:

- Programmer directement la configuration des canaux de routage: l'outil Jbits[11] de Xilinx permet de manipuler directement la configuration d'un FPGA et donc de définir un routage sans avoir à passer par le routeur. Cependant cet outil ne permet d'agir qu'au plus bas niveau (*bitstream*). Cela signifie qu'il faudrait également définir le placement directement au plus bas niveau, sans pouvoir s'aider de contraintes de placement, ce qui aurait pour effet de complexifier aussi cette étape.
- Utiliser des macros: il est possible, par exemple avec l'outil *fpga_editor* de Xilinx, de sauvegarder un circuit sous la forme d'une *hard macro*, qui conserve le placement et le routage du circuit. Ainsi la solution consiste à sauvegarder un processeur sous la forme d'une macro pré-placée et pré-routée, puis à remplacer dans le réseau linéaire toutes les instances des processeurs de même forme par des instances de la macro correspondante.

Il apparaît donc que le principal obstacle au "routage régulier" est l'impossibilité de contraindre le routage pour éviter les conflits.

Cependant, malgré les limitations dues au temps de routage, nous croyons que les gains apportés

par un placement régulier vont croître avec l'évolution de la complexité des composants FPGA. D'abord, il est admis qu'avec cette évolution, les délais associés au routage diminuent moins vite que les délais liés à la logique. Les délais de routage vont prendre une importance croissante dans le chemin critique, donc un mauvais placement sera de plus en plus pénalisant pour la fréquence d'horloge. De plus, avec l'augmentation considérable du nombre de portes logiques, et donc l'augmentation de la taille des réseaux linéaires implémentables, le placement régulier semble être la meilleure solution pour maîtriser la complexité de tels circuits.

Bibliographie

1. P. Quinton et V. V. Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing,* 1(2), 1989.
2. J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati et P. Boucard. Programmable Active Memories: Reconfigurable Systems Come of Age. *IEEE Transactions on VLSI Systems,* 4(1), 1996.
3. E. Fabiani, D. Lavenier et L. Perraudeau. Loop Parallelization on a Reconfigurable Coprocessor. In *WDTA'98: Workshop on Design, Test and Applications,* Dubrovnik, Croatia, 1998
4. C. Mauras. Alpha : un langage équationnel pour la conception et la programmation d'architectures systoliques. *PhD thesis, université de Rennes 1,* 1989.
5. P. Le Moenner, L. Perraudeau, P. Quinton, S. Rajopadhye, T. Risset Generating Regular Arithmetic Circuits with ALPHARD. *MPPS'96: Massively Parallel Computing Systems,* Ischia, Italy, 1996.
6. R.F. Lyon, Two's complement pipeline multipliers. *IEEE Transaction on Communications*, April 1976.
7. G.M. Megson et I.M. Bland. Generic systolic array for genetic algorithms. *IEEE Proceeding - Computers and Digital Techniques,* vol. 144(2): 107-121, 1997.
8. Xilinx. *The Programmable Logic Data Book,* 1997.
9. S. Martello et P. Toth. Knapsack Problems: Algorithms and Computer Implementation. *John Wiley and Sons,* 1990.
10. R. Bellman. Dynamic Programming. *Princeton University Press,* Princeton, NJ, 1957.
11. S. A. Guccione et D. Levi. XBI: A java-based interface to FPGA hardware. *Configurable Computing: Technology and Applications*, Proc. SPIE 3526: 97-102, 1998.