Actes du 4ème Symposium sur les Architectures Nouvelles de Machines,
Rennes, 7–9 février 1996

Conception d'un réseau systolique à partir de C-stolic
Application à la biologie moléculaire

D. Lavenier - C. Wagner
IRISA
Campus de Beaulieu
35042 Rennes cedex
lavenier@irisa.fr, wagner@irisa.fr

Résumé

Cet article décrit la méthodologie de conception mise en œuvre pour réaliser un ASIC constituant la brique de base de la machine SAMBA, une machine systolique dédiée à la comparaison des séquences biologiques. Cette méthodologie s'appuie sur le langage C-stolic, outil qui a permis la spécification, la validation et le test du circuit. A travers cette expérience nous soulignons les points sur lesquels nos efforts doivent porter pour automatiser complètement le processus de conception.

1. Introduction

La conception d'un machine informatique spécifique est un processus qui requiert de nombreuses étapes, notamment lorsque des circuits intégrés spécifiques (ASIC) doivent être développés. La première phase consiste généralement à spécifier la fonction à réaliser ; la dernière vérifie, dans son environnement réel, le bon fonctionnement de la machine. Le rôle de l'architecte de machines informatiques est, entre autre, d'assurer un enchaînement cohérent entre toutes ces étapes.

D'autre part, le nombre de transistors intégrables sur une puce double chaque année. La complexité des circuits développés, et des temps de mise sur le marché de plus en plus courts, ont conduit à automatiser certaines étapes, voire l'ensemble du processus. La synthèse d'architecture est donc, aujourd'hui, une activité de recherche très active où différentes méthodologies sont proposées pour assister l'architecte tout au long de la conception.

Cet article présente la méthodologie de conception qui a été suivie pour réaliser, une machine systolique dédiée à la comparaison des séquences biologiques, la machine SAMBA. Plus précisément, nous montrons comment nous avons spécifié, réalisé et testé un ASIC (le circuit API 256) constituant la brique de base de la machine. SAMBA

SAMBA se compose de trois parties : une station de travail, une carte à base de FPGA (la carte Perle-1 [3]) et un réseau systolique linéaire de processeurs cablés. La figure 1 indique les connexions entre ces 3 éléments. La carte FPGA réalise le lien entre la partie logicielle d'une application (implantée sur la station de travail) et la partie matérielle (implantée sur le réseau systolique).

Même si le réseau systolique est constitué de processeurs cablés – donc, par définition, non programmables – SAMBA n'est pas restreinte à l'exécution d'un algorithme unique :

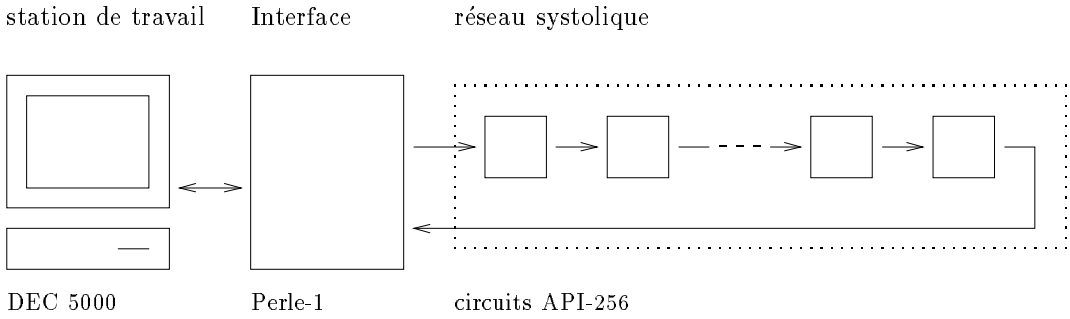

FIG. 1 - . synoptique de la machine SAMBA

plusieurs registres d'état permettent de configurer les processeurs pour exécuter plusieurs variations d'un algorithme principal.

Le pivot central sur lequel s'appuie notre méthodologie de conception est le langage C-stolic [8] [9] [7], langage initialement développé pour la simulation des algorithmes systoliques s'exécutant sur un réseau linéaire de processeurs programmables. Outre cet usage initial, nous avons exploité ses caractéristiques pour nous rapprocher au maximum d'une description matérielle et obtenir ainsi une transcription naturelle en VHDL.

La méthodologie qui a été suivie nécessite donc une intervention manuelle pour passer d'une description C-stolic à une description VHDL. L'équivalence entre les deux descriptions a été obtenue par simulation et confrontation des résultats, tout en sachant les limites de ce mode de vérification. Cette stratégie a été mise en place parce qu'elle représentait la seule possibilité de vérification dans notre environnement au moment où SAMBA a été conçue.

Il est clair que dans l'optique d'une automatisation complète, cette intervention doit être supprimée. Aussi, à partir de l'expérience acquise pendant la conception de SAMBA, cet article tente également de préciser la nature des nouveaux outils à mettre en œuvre pour tendre vers une conception plus automatisée.

Le paragraphe suivant expose d'abord le domaine d'applications de SAMBA et introduit le problème algorithmique. Le paragraphe 3 présente succintement les particularités de C-stolic. Nous décrivons ensuite les différentes étapes qui ont abouties à la réalisation concrète du circuit intégré API 256. Nous terminons sur diverses perspectives relatives à ce travail en indiquant, notamment, les points sur lesquels notre effort doit se poursuivre pour améliorer l'automatisation du processus de conception.

2. Domaine d'applications

La machine SAMBA (Systolic Accelerator for Molecular Biological Applications) est entièrement dédiée à la comparaison de séquences biologiques [2]. Cette tâche est un des traitements de base de la biologie moléculaire. La dénomination "*comparaison de*

séquences" regroupe, en fait, plusieurs opérations qui ont en commun de manipuler de gros volumes de données (les séquences biologiques) et de leur appliquer des traitements informatiques particuliers. Parmi les opérations traditionnellement citées on distingue :

- la recherche de segments similaires, opération qui consiste à partir d'une séquence particulière – la séquence test – à rechercher dans une banque quelles sont les séquences qui partagent une ou plusieurs zones *ressemblantes* ;
- la classification d'un ensemble de séquences : il s'agit d'établir une distance entre toutes les séquences pour les situer les unes par rapport aux autres ;
- l'élimination des redondances : actuellement, les banques contiennent de quelques dizaines à quelques centaines de milliers de séquences fournies par l'ensemble de la communauté biologiste. Il est fréquent qu'une même séquence soit enregistrée sous deux noms différents, parce qu'elle a été séquencée indépendemment par deux laboratoires travaillant sur des thèmes identiques.

Ces exemples sont représentatifs de ce qu'on entend habituellement par "*comparaison de séquences*", mais ne constitutent, en aucune manière, une liste exhaustive.

Le point commun de ces opérations est l'estimation d'une distance entre deux séquences (ou deux sous-séquences) afin de juger de leur *ressemblance*. Le calcul de cette distance n'est pas unique : différentes méthodes existent et apportent chacune leur lot d'avantages et d'inconvénients. On en distingue deux grandes familles : la première détermine une distance entre deux séquences (ou sous-séquences) de même taille. L'avantage réside à la fois dans la faible complexité algorithmique et par un modèle mathématique sous-jacent qui permet d'estimer la validité des résultats. L'inconvénient majeur est que ce mode de calcul est inadapté lorsque l'on doit mesurer une distance entre deux séquences (ou morceau de séquences) de tailles différentes.

La seconde famille pallie cet inconvénient au prix d'une complexité algorithmique accrue et au détriment d'un support mathématique quasi inexistant. Les méthodes mises en jeu font essentiellement appel à des techniques de programmation dynamique qui impliquent des volumes de calculs importants et conduisent, par conséquent, à des temps de calculs très longs. Aussi, ces méthodes sont elles très peu employées aujourd'hui lorsqu'il s'agit, par exemple, d'explorer des banques volumineuses ou de faire de la classification sur un ensemble de séquences conséquent.

La machine SAMBA a comme objectif principal de proposer une solution pour réduire fortement les temps de calcul relatifs aux traitements appartenant à la deuxième famille[1], c'est à dire aux traitements qui sont en général laissés de coté par ce que trop onéreux en puissance de calcul. Le gain en vitesse, par rapport à une station de travail (un SPARC-2, par exemple), varie de 100 à 1000 suivant les applications considérées.

Les algorithmes considérés (programmation dynamique) sont des algorithmes *réguliers* qui s'implémentent bien sur des structures systoliques. Si l'adéquation algorithme-architecture s'exprime mieux sur une structure bi-dimensionnelle, ils peuvent néanmoins être efficacement mis en œuvre sur une structure linéaire. Cette dernière à l'avantage de nécessiter une alimentation en données raisonnable et de bénéficier, à l'IRISA, d'un environnement de programmation adéquate (C-stolic).

1. cela n'exclue pas l'accélération des algorithmes appartenant à la première fammille

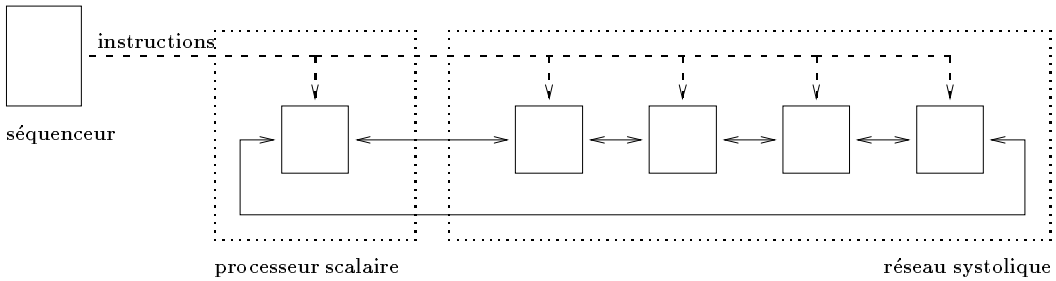

FIG. 2 - . modèle de programmation de C-stolic

3. C-stolic

C-stolic a été conçu pour fournir au concepteur d'applications systoliques un outil de programmation simple et efficace. La simplicité est atteinte par l'extension d'un langage largement répandu, le langage C. Cette extension porte sur très peu de mots clés et respecte la philosophie du langage originel. L'efficacité provient de ce que l'utilisateur contrôle directement les éléments les plus critiques : les communications et le partitionnement.

Le modèle de programmation du langage C-stolic représente une machine systolique comme un système composé de deux éléments distincts : un processeur scalaire et un réseau de processeurs régulièrement connectés dont les extrémités sont reliées au processeur scalaire (figure 2). La tâche du réseau est d'effectuer les calculs de façon systolique ; celle du processeur scalaire est de contrôler le flot de données et de gérer les entrées/sorties du réseau.

Les objets manipulés par le programmeur sont localisés soit sur le processeur scalaire, soit sur le réseau. Un objet localisé sur le processeur scalaire est mémorisé dans une variable *scalaire*. Un objet localisé sur le réseau est un tuple dont chaque processeur mémorise un élément dans une variable de type *systolique*. Une opération portant sur des variables systoliques provoque l'exécution simultanée de cette opération dans chaque processeur du réseau. Les seules opérations qui manipulent à la fois des variables scalaires et des variables systoliques sont les opérations de communication.

Il n'existe qu'un seul séquenceur qui envoie simultanément une instruction au processeur scalaire et une instruction à l'ensemble des processeurs du réseau. Les conditions de rupture dans le flot de contrôle proviennent principalement du processeur scalaire. Cependant, un contrôle local plus limité est possible par l'affectation conditionnelle. Par exemple, l'instruction `A = (K>0) ? B : C`, si elle ne fait intervenir que des variables de type systolique, n'interfère pas sur le flôt de contrôle principal.

Les communications sont synchrones : toute émission d'une donnée dans une direction implique une réception dans la direction opposée. Les opérations de communication réalisent ainsi des décalages de données à travers l'ensemble du réseau. Au cours de cette opération, le comportement du processeur scalaire est le suivant : dans le cas d'un décalage vers la gauche, par exemple, une donnée est émise vers le processeur situé le plus à

gauche dans le réseau, tandis que la donnée produite par le processeur le plus à droite est récupérée par le processeur scalaire.

Le compilateur C-stolic produit un ou plusieurs fichiers C suivant le type de machine visée pour l'exécution. De manière générale, l'application principale est développée en C et appelle des procédures écrites en C-stolic pour les parties parallélisées. L'exécution de ces procédures s'effectue sur la machine parallèle spécifiée lors de la compilation.

Un simulateur, s'exécutant sur une machine séquentielle est également disponible pour aider à la mise au point des programmes C-stolic. Il incorpore un déboggeur qui permet de connaitre à tout instant l'état de l'ensemble des variables (scalaires et systoliques). Cet outil est indispensable dès lors que des programmes conséquents sont mis en oeuvre.

4. Methodologie de conception

Ce paragraphe montre l'usage de C-stolic dans les différentes étapes de conception qui ont abouties à la réalisation du circuit intégré API 256. La méthodologie que nous avons suivie se résume par la figure 3. On distingue plusieurs phases :

- description en C-stolic de l'algorithme à implanter ;
- simulation C-stolic ;
- description en VHDL du même algorithme ;
- simulation VHDL et comparaison avec C-stolic ;
- synthèse à partir de VHDL ;
- test du circuit.

Les paragraphes suivants explicitent chacune de ces phases.

Description en C-stolic : l'algorithme qui a été câblé est basé sur l'algorithme de Smith et Waterman [10] [5]. C'est un algorithme stable et unanimement reconnu dans le domaine de la biologie moléculaire. Cependant, pour être utilisé dans les principales applications, il doit autoriser "certaines" variations.

Un des premiers buts de notre étude a donc été de cerner l'ampleur des ces variations, tout en sachant que le produit final serait un ASIC non programmable. Plusieurs applications cibles ont alors été étudiées, puis le traitement de base relatif à la comparaison de séquences parallélisé et simulé en C-stolic. L'étude comparative des codes C-stolic a ensuite généralisé l'algorithme par l'introduction de paramètres qui, suivant leurs valeurs, déterminent l'algorithme à exécuter. Cette étude nous a permis :

- de déterminer un algorithme paramétré couvrant un large spectre d'applications dans lesquelles la comparaison de séquences intervient de manière intensive ;
- d'être certain que cet algorithme était parallélisable (puisque décrit directement en C-stolic) ;
- d'imaginer (en parallèle) un dispositif interne aux processeurs pour acheminer les résultats vers les extrémités du réseau ;
- de prendre conscience que chaque variation algorithmique demande un contrôle du réseau différent (initialisation, alimentation en données et collecte des résultats).

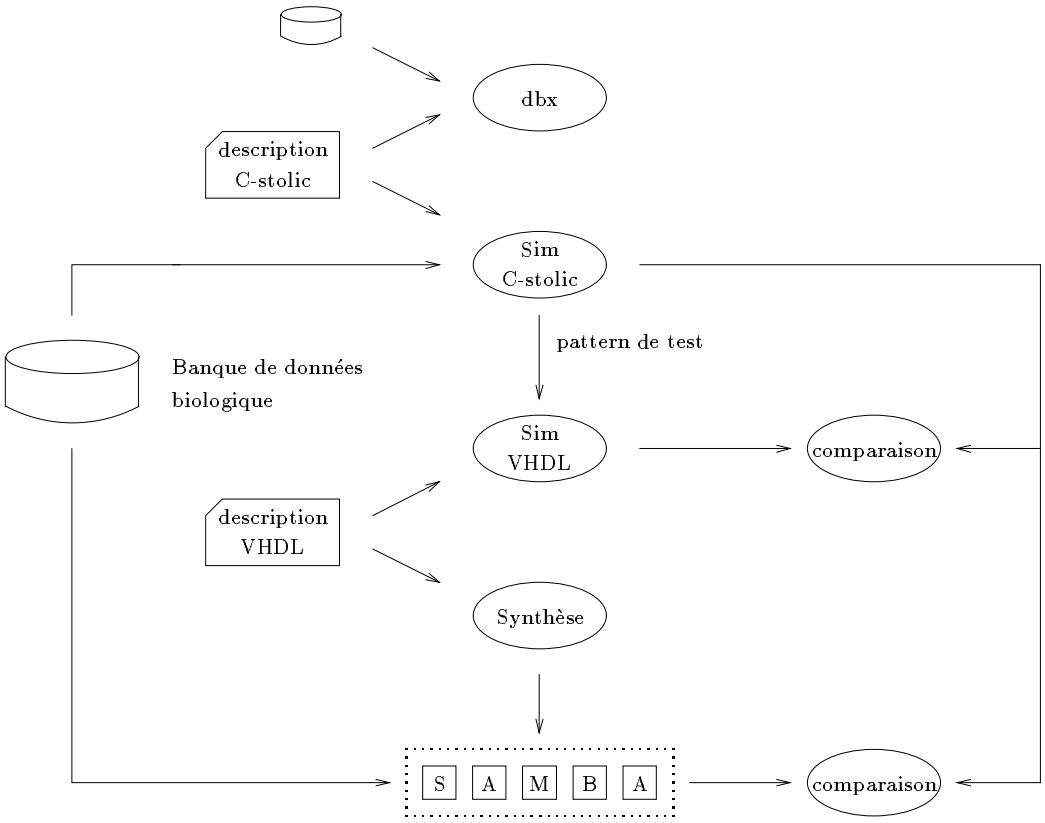

FIG. 3 - . méthodologie de conception de SAMBA

Ce dernier point est particulièrement important car sa mise en œuvre conditionne grandement les performances globales de la machine [6]. Le contrôle ne peut être définitivement cablé puisqu'il est différent suivant l'algorithme exécuté par le réseau ; il ne peut être exécuté par un processeur séquentiel, trop lent pour gérer (à chaque top d'horloge) l'émission et la réception simultanées des données. C'est pourquoi, nous avons opté pour une structure à base de FPGA, structure qui permet d'allier les contraintes temporelles à une architecture modulable.

Simulation C-stolic : contrairement à l'étape précédente qui s'est satisfaisaite de données réduites pour mettre au point le programme C-stolic, cette étape a eu pour but de valider les choix réalisés sur des données grandeur nature. En effet, la mise au point algorithmique s'accomode mal avec la gestion de gros volumes de données et des temps d'exécution qui en découlent. Aussi, pour tester rapidement le bien fondé d'une modification, par exemple, les vérifications sont réitérées sur un petit jeu de données représentatif. Mais, aussi representatif soit il, ce jeu de données ne peut prétendre couvrir la majorité des cas, notamment les cas particuliers qui n'ont pas été imaginés et qui posent souvent

problème.

Dans le cas de la réalisation d'un ASIC, il est indispensable de garantir sa fonctionnalité dans tous les cas de figure : après passage en fonderie, une remise en cause n'est plus possible, contrairement à un logiciel qui peut toujours être corrigé ultérieurement. La première étape ayant stabilisé l'algorithme – donc l'architecture du processeur résultant – cette seconde étape le valide par le biais de simulations intensives.

Cependant, la simulation C-stolic est une opération coûteuse en temps de calcul dès lors que l'on envisage le test sur de vraies banques de séquences biologiques. Cette étape peut toutefois être réduite en exécutant cette simulation sur une machine parallèle sur laquelle il existe un compilateur C-stolic.

Description VHDL : l'écriture de l'agorithme en C-stolic s'est faite en ayant en tête qu'elle devait servir de base à une transcription en VHDL. Elle a alors consisté à mettre en évidence ce qui devait être exécuté pendant un cycle systolique, c'est à dire par chaque processeur à chaque coup d'horloge. En C-stolic, cela est représenté par l'exécution d'un corps de boucle. Pour fixer les idées, une description partielle du corps de boucle est donnée ci-dessous avec, en correspondance, la traduction en VHDL.

Code C-stolic	Code VHDL
...	...
	Fv_beta <= Fv - beta;
	Eh_beta <= Eint - beta;
	Hv_alpha <= Hv - alpha;
	Hh_alpha <= Hint - alpha;
tmp5 = (Hh-alpha) >? (Eh-beta);	if signed_vector(Hh_alpha) > signed_vector(Eh_beta)
	then tmp5 <= Hh_alpha;
	else tmp5 <= Eh_beta;
	end if;
F = (Hv-alpha) >? (Fv-beta);	if signed_vector(Hv_alpha) > signed_vector(Fv_beta)
	then F <= Hv_alpha;
	else F <= Fv_beta;
	end if;
tmp1 = tmp5 >? F;	if signed_vector(tmp5) > signed_vector(F)
	then tmp1 <= tmp5;
	else tmp1 <= F;
	end if;
tmp2 = Hd+SUB[S];	tmp2 <= Hd + subs;
tmp3 = delta >? tmp2;	if signed_vector(tmp2) > signed_vector(delta)
	then tmp3 <= tmp2;
	else tmp3 <= delta;
	end if;
tmp4 = tmp1 >? tmp3;	if signed_vector(tmp3) > signed_vector(tmp1)
	then tmp4 <= tmp3;
	else tmp4 <= tmp1;
	end if;
tmp10 = (tmp4>=Mh) ? tmp4 : Mh;	if signed_vector(Mint) > signed_vector(tmp4)
	then tmp10 <= Mint;
	else tmp10 <= tmp4;
	end if;
...	...

L'expression C-stolic `A = B >? C` est une compression de `A = (B>C) ? B : C`. A chaque expression C-stolic a été substituée une expression VHDL équivalente pour obtenir le fichier VHDL synthétisable. La correspondance a pu être réalisée aisément parce que le programme C-stolic a été écrit en ayant en tête la structure matérielle sous-jacente.

Simulation VHDL : cette simulaton a eu pour but de vérifier que les descriptions C-stolic et VHDL étaient équivalentes. Les patterns d'entrée ont été générés en introduisant dans le code C-stolic une fonction d'impression des variables systoliques représentant les connexions inter-processeur. Cette fonction, placée en tête de la boucle principale, donne à chaque cycle systolique l'état de toutes les liaisons. Ces données peuvent ensuite être reprises à la fois comme stimuli d'entrée du simulateur VHDL et comme données de vérification par rapport à celles produite par le simulateur.

Comme nous l'avons déjà souligné, cette méthode a ses limites : elle n'assure pas une équivalence totale à cause de la non exhaustivité des situations. On peut seulement prédire que plus les tests sont nombreux, plus la chance de laisser passer une erreur est faible.

Synthèse : cette étape, composée d'une multitude de sous-étapes, produit les masques du circuit à partir de la description VHDL. L'outil de CAO utilisé a été COMPASS et la méthodologie de conception celle préconisée par cet outil.

La partie synthèse, proprement dite, est immédiate. Le code VHDL est analysé par un synthétiseur et délivre une *netlist* composée d'éléments de bibliothèque. La synthèse peut cependant être partiellement dirigée en décorant le code VHDL de directives telles que le partitionnement en plusieurs blocs arithmétiques, le choix d'opérateurs optimisés (en surface ou en vitesse), etc. La suite des opération est classique et nécessite l'intervention d'un expert en conception de circuits intégrés.

Le circuit final contient 4 processeurs. La nature systolique du réseau permet de cascader plusieurs puces pour obtenir un réseau taille de quelconque.

Test : Les tests ont été réalisés en deux temps : les premiers ont consisté à vérifier le bon fonctionnement des circuits seuls. Puis les seconds ont été consacrés au réseau, c'est à dire à l'assemblage de plusieurs puces. Pour chaque test, les patterns ont été générés à partir de C-stolic et les résultats confrontés à ceux produits par le simulateur.

5. Conclusion

Le langage C-stolic a servi de support tout au long de la conception du circuit API 256, que ce soit pour la définition de l'algorithme à implanter, sa validation par des simulations intensives ou pour la production de patterns de test.

Cette expérimentation a également permis de relever les faiblesses de cette approche sachant, qu'initialement C-stolic n'a pas été développé pour cet usage. Néanmoins, il nous semble intéressant d'explorer plus en avant cette méthodologie et de l'appliquer, non plus sur la conception d'un seul élément, mais sur une machine complète. Cela conduit à réfléchir sur les modifications éventuelles à apporter à C-stolic ainsi que sur d'autres outils pour améliorer l'automatisation du processus de conception.

Nous avons relevé trois principaux points sur lesquels il conviendrait de concenter nos

efforts :

- la traduction automatique de C-stolic en VHDL ;
- l'extension de C-stolic pour prendre en compte certaines contraintes matérielles ;
- l'accélération des simulations C-stolic.

Traduction C-stolic en VHDL

Ce point est particulièrement important car, pour l'instant, il constitue une rupture dans l'automatisation du processus de conception. Posséder un tel outil permettrait de s'affranchir des opérations de simulation que nous avons été contraint de mettre en place pour s'assurer de l'équivalence des deux codes.

A partir du code C-stolic, le traducteur devra considérer 2 types de structure : la première a trait au réseau, c'est à dire à l'architecture des processeurs. La seconde, qui peut être déduite du code C-stolic, est le mécanisme de gestion des données. Sur SAMBA, cette seconde structure correspond à l'interface à base de FPGA qui assure la liaison entre la station de travail et le réseau systolique.

Le compilateur actuel réalise déjà cette séparation entre le calcul parallèle (celui réalisé par le réseau) et le calcul scalaire (gestion du réseau). Aussi, au lieu de produire un code s'exécutant sur des machines parallèles programmables, l'idée est de générer deux descriptions matérielle, l'une correspondant à l'architecture d'un processeur du réseau, l'autre à l'architecture mettant en œuvre le contrôle et l'alimentation du réseau, la collecte des résultats, et la liaison avec le processeur hôte.

A partir de ces deux descriptions distinctes on peut ensuite choisir la cible technologique adéquate : logique reconfigurable (FPGA), circuit intégré spécifique (ASIC) ou, comme dans le cas de SAMBA, un panachage de deux technologies (ASIC pour le réseau et FPGA pour le reste).

Extension de C-stolic

L'usage initial de C-stolic est la simulation d'algorithmes systoliques sur des machines parallèles programmables. Dès lors que l'on s'intéresse à une implémentation sur du matériel spécialisé, plusieurs points doivent être précisés. Citons, entre autre :

- la spécification de la largeur des chemins de données : sur une machine programmable les entiers sont, par exemple, codés sur 32 bits. Sur une machine spécifique, il n'y a souvent pas lieu de posséder une telle dynamique. On cherche plutôt à optimiser les ressources de manière à implémenter ce qui est juste nécessaire. De plus, pour tester les débordements éventuels, la simulation doit en tenir compte. Actuellement, C-stolic permet ce type de contrôle à l'aide d'opérations logiques qui effectuent différents masquages et simulent les opérations sur une dynamique reduite. Ces opérations doivent être explicitées lors de l'élaboration du programme et constituent une tâche fastidieuse, voire une source d'erreurs.
- le choix des opérateurs matériels : l'optimisation d'une architecture est souvent soumise au compromis vitesse/surface. Plus on veut aller vite, plus la taille des opérateurs est importante ; plus on cherche à réduire les ressources, plus la vitesse des opérateurs est lente. En général, c'est du ressort l'architecte de machine d'ajuster au mieux cet équilibre. Il doit donc posséder les moyens adéquats pour y parvenir.

Ceci ne constitue pas une liste exhaustive des points qui méritent d'être étudiés pour orienter C-stolic vers le design d'accélérateurs matériel, mais donne une idée des extensions à apporter. Ces dernières peuvent être mises en œuvre de 2 manières : soit en modifiant le langage, soit en ajoutant au code existant des directives de compilation introduites par le biais de commentaires.

Accélération des simulations

L'éxécution d'un programme C-stolic sur une machine parallèle programmable engendre une accélération qui, à l'usage, s'est révélée insuffisante. Les raisons sont multiples : faible nombre de processeurs disponibles (iWarp à 8 processeurs), communications inter processeurs non adaptées à la granularité "systolique" (iPSC/2), communication hôte/réseau trop lente (MasPar), etc.

Une autre solution consiste à implémenter l'algorithme sur une structure à base de logique reconfigurable telle que les machines *WildFire* (la version commerciale de *SPLASH-2*) [1], *Virtual Computer* (réseau de 52 Xilinx 4010) [4], META 100 (une machine française dédiée à l'accélération de programme VHDL), ou tout autre structure équivalente.

Cette approche présente d'une part l'avantage d'être très proche de la réalisation finale et, d'autre part, de bénéficier directement des travaux de synthèse envisagés à partir de C-stolic.

Bibliographie

1. J.M. Arnold, D.A. Buell, and E. G. Davis. SPLASH 2. In *4th Annual ACM Symposium on Parallel Algorithms and Architecture*, 1992.
2. L. Audoire, JJ. Codani, D. Lavenier, and P. Quinton. Machines spécialisées pour la comparaison de séquences biologiques. *TSI*, 14(1):9–22, January 1995.
3. P. Bertin, D. Roncin, and J. Vuillemin. Programmable active memories : a performance assessment. In F. Meyer auf der Heide, B. Monien, and A.L. Rosenberg, editors, *Parallel Architectures and their efficient use*, pages 119–130, Lecture notes in Computer Science, Springer-Verlag, oct 1992.
4. S. Casselman. Virtual Computing and The Virtual Computer. In *FPGAs for custom computing machines*, pages 43–48, IEEE Computer Society Press, apr 1993.
5. O. Gotoh. An Improved Algorith for Matching Biological Sequence. *J. Mol. Biol.*, (162):705–708, 1982.
6. D. Lavenier, F. Raimbault, and P. Frison. I/O and Computation Overlap on SIMD Systolic Arrays. *Journal of VLSI Signal Processing*, 9(3), April 1995.
7. F. Raimbault. *Etude et réalisation d'un environnement de simulation parallèle pour les algorithmes systoliques.* PhD thesis, Université de Rennes 1, jan 1994.
8. F. Raimbault and D. Lavenier. Relacs for Systolic Programming. In *ASAP 93*, pages 132–135, IEEE Computer Society Press, Venice, Italy, Oct 1993.
9. F. Raimbault, P. Quinton, and D. Lavenier. Architectures Systoliques et Parallélisme de données; l'environnement de programmation ReLaCS. *TSI*, 12(5):597–620, 1993.
10. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol*, (147):195–197, 1981.