# Une carte massivement parallèle SIMD programmable

Denis Archambaud, François Dromard, Alain Greiner, Pascal Faudemay, Ivan Saraiva Silva et Laurent Winckel

Laboratoire MASI / Université Pierre et Marie Curie
4 Place Jussieu, 75252 Paris cedex 05, France

{archambaud, winckel, faudemay }@masi.ibp.fr

**Résumé**

Dans cet article nous présentons la carte massivement parallèle RAPID-2 et son modèle de programmation. RAPID-2 est une carte massivement parallèle SIMD (un seul flot d'instructions, plusieurs flots de données), destinée à être utilisée comme carte accélératrice sur des PC. Elle utilise deux types de composants VLSI qui ont été développés dans le laboratoire à l'aide de la chaîne de CAO Alliance.

Un langage de programmation, L1, a été développé pour cette carte. Il s'appuie sur un modèle de programmation que nous décrivons brièvement dans cet article. Le langage est une extension du C. A partir d'un programme écrit dans ce langage, un compilateur le sépare en un programme d'application en C destiné à être exécuté sur le PC, et un microprogramme destiné à être exécuté sur la carte accélératrice. Le compilateur permet donc le codéveloppement du code C et du micro-code.

**Introduction**

La puissance de calcul des stations de travail et des micro-ordinateurs augmente régulièrement. Cependant de nombreuses applications demandent un gain de plusieurs ordres de grandeur dans le rapport performances / coût, dont l'amélioration est le but principal des architectes de machines et de circuits. Parmi ces applications on peut noter le calcul scientifique, comme par exemple la comparaison de séquences génétiques pour la biologie moléculaire, la reconnaissance de formes en temps réel, le codage de données vidéo, etc...

L'une des approches les plus prometteuses au cours des prochaines années pour accroître ce rapport performances / coût est constitué par le parallélisme massif SIMD. Ce parallélisme repose sur la répétition, dans une ou plusieurs cartes, de *petites* unités d'exécution relativement simples à réaliser, et sur un séquencement global. Celui-ci facilite la programmation de la machine selon un modèle de programmation simple.

Parmi les principaux modèles de parallélisme SIMD, le parallélisme de données et le modèle systolique correspondent à différents degrés de dépendance entre les données. Avec les réseaux d'interconnexion les plus courants actuellement, ces deux modèles peuvent être utilisés simultanément ou alternativement sur la même machine physique [3]. Cette possibilité est offerte par plusieurs types de processeurs associatifs ou vectoriels (Array Processors, tels que Cnaps ou IMAP [2, 6]).

Cependant l'approche du parallélisme massif à travers des machines centralisées de coût élevé, comme Maspar ou la CM-5, a conduit dans certains cas à un relatif échec industriel, en raison de l'étroitesse du marché et parfois de la relative difficulté de programmation par des utilisateurs ordinaires. Dans le même temps, des solutions utilisant des cartes accélératrices de coût comparable à celui d'un PC semblent rencontrer un succès croissant. Ce succès dépend pour une grande part de la disponibilité d'un langage de programmation simple et de haut niveau.

Dans cet article, nous présentons une carte accélératrice pour PC comprenant deux types de composant VLSI développés au laboratoire MASI. Nous présentons d'abord l'architecture de notre système, puis le modèle de programmation en L1, qui repose sur une extension du C. Nous présentons ensuite l'état d'avancement du projet et les gains de performance obtenus. Nous concluons en montrant pour quels types d'applications cette architecture apparaît comme la plus prometteuse.

## Architecture

Notre équipe développe actuellement une carte destinée à l'accélération des programmes du génome tournant sur les machines couramment utilisées par les généticiens pour la comparaison des séquences. Notre but est de fournir une puissance de calcul supplémentaire et dédiée à des machines existantes comme les PC ou compatibles. Cette puissance additionnelle devrait être suffisante pour résoudre les problèmes actuels à un prix largement inférieur aux solutions parallèles générales de type Maspar ou Connection-Machine, tout en restant entièrement programmable au niveau le plus bas, ce qui donne une flexibilité supérieure à toute solution strictement dédiée à un algorithme. Dans la gamme des machines applicables aux opérations génome pour un coût de fabrication modeste, la complexité de Rapid-2 se situe entre celle de P-NAC (dont la réalisation est maintenant ancienne, [1]) et celle de Cnaps (fabriqué par *Adaptative Solutions*, [2]); elle se rapproche de celle de Samba (système systolique dédié au génome en cours de conception à l'IRISA) bien que son concept de programmabilité soit très différent.

Notre carte se compose de deux types de circuit. Le circuit d'exécution, *brique* de base cascadable qui réalise le parallélisme, et le circuit de contrôle chargé d'envoyer les ordres aux circuits d'exécution et de gérer l'interface avec l'ordinateur hôte. Globalement (c'est à dire sans tenir compte du partitionnement introduit par les circuits), le système se compose d'un micro-séquenceur, d'une unité de gestion de l'interface et de N processeurs élémentaires (N est le facteur de parallélisme). L'implémentation de 256 processeurs élémentaires répartis sur 16 circuits en technologie CMOS 0.8 µm est prévue. La figure 1 présente les communications possibles entre ces blocs. Les connexions entre les divers éléments sont les suivantes :

- Un bus de micro-instruction qui provient du micro-séquenceur et qui est connecté à tous les processeurs élémentaires (PE). Chaque processeur reçoit donc simultanément la même instruction que les autres. Une telle organisation SIMD permet une grande simplicité dans l'implémentation de la partie contrôle qui peut être unique alors que la partie exécutive est répliquée.

- Un bus de données bidirectionnel qui provient de l'interface avec l'hôte et qui est connecté à tous les PEs. Un tel bus permet d'accéder à l'espace mémoire formé par les PEs ainsi que de fournir des opérandes communs à tous les PEs pour certaines opérations.

- Un réseau de communication de type systolique monodimensionnelle. Il s'agit d'un bus qui relie chaque processeur à ses deux voisins. Celui-ci sera appelé *« mesh »* par la suite. Les PEs sont topologiquement organisés en une rangée de processeurs, chacun ayant un *voisin* *de gauche* et un *voisin de droite*.

Le mesh a un coût d'implémentation très faible comparé aux bus globaux cités précédemment. Il permet une utilisation du circuit telle que les données peuvent être facilement déplacées d'un PE à l'autre en mode parallèle. Le mesh permet un fonctionnement systolique, offrant une utilisation maximale du parallélisme pour de faibles coûts relatif en entrées/sorties [4].

- Un arbre d'arbitrage qui permet de sélectionner un PE parmi ceux qui répondent à un critère donné. Physiquement l'arbre a une architecture dichotomique qui lui permet d'être facilement partagé entre les différents circuits d'exécution et le circuit de contrôle [5]. Cet arbre est utilisé dans la situation suivante : une micro-instruction a sélectionné un certain nombre de PEs, l'utilisateur désire lire le contenu d'un registre de chacun d'entre eux. Leurs contenus doivent donc être émis sur le bus de données séquentiellement, l'arbre d'arbitrage sert alors à sélectionner les PEs l'un après l'autre pour que seul un PE émette à un instant donné.

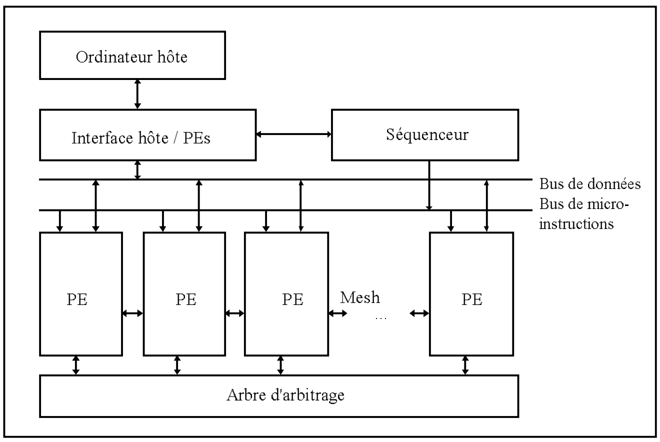


Figure 1 : Architecture de la carte Rapid-2.

L'interface avec l'ordinateur hôte est composée de trois registres dédiés à la communication entre l'hôte et la carte. Le registre de données est utilisé pour transférer les données, opérandes et résultats. Le registre d'instruction est utilisé pour transmettre à la carte le *codop* de l'instruction accompagné de ses paramètres. Le registre d'état informe l'hôte de l'état du circuit, un mécanisme de sémaphores y est implanté, qui permet une gestion asynchrone des lectures et écritures dans les registres de l'interface.

Le micro-séquenceur se compose d'une mémoire de microprogramme réinscriptible depuis l'hôte. Il s'agit d'une RAM dont l'adressage est assuré par un mécanisme de séquencement qui permet 8 types de branchement (4 branchements conditionnels, branchement à des sous-programmes etc...). Le microprogramme se compose de micro-instructions dont les champs décrivent exactement les commandes des ressources des PEs. Aucun codage/décodage n'est nécessaire, de telle sorte que le séquencement des PEs peut être rapide. Le stockage des microprogrammes en RAM permet une grande souplesse dans l'implémentation d'algorithmes variés et susceptibles d'évoluer comme ceux concernant la comparaison des séquences génétiques.

Le processeur élémentaire se compose d'une mémoire de 64 mots de 38 bits (32 bits de données et 6 bits de marquage interne) et d'une ALU capable d'effectuer des opérations sur des mots de 32 bits signés (+, -, décalages, opérateurs logiques), ainsi que sur des octets non signés. L'ALU 32 bits est partitionnée en 4 ALUs 8 bits de telle sorte que les opérations 8 bits sont effectuées en parallèle avec le même matériel. Globalement le facteur de parallélisme est multiplié par 4 en mode 8 bits. Les PEs n'ont pas d'adresse. Un mode de fonctionnement paginé est rendu possible par l'utilisation d'un même registre dans chaque PE (nommé page). Les pages peuvent être adressées par le circuit de contrôle et manipulées individuellement.

Le système peut être utilisé comme un réseau systolique linéaire grâce à ses communications entre voisins. Il peut aussi être utilisé comme une mémoire associative paginée par ensembles où les requêtes sont effectuées en parallèle dans tous les PEs. L'avantage d'une telle versatilité est l'adaptation plus grande aux algorithmes par nature divers dans leur besoins calculatoires. Par exemple, la comparaison des séquences génétiques demande une phase très calculatoire dont seule une implémentation systolique est efficace, puis une phase de récupération de certains résultats par la machine hôte, qu'une implémentation associative résout mieux que la vidange du réseau systolique.

L'aspect associatif du système implique la gestion de résultats booléens qui indiquent l'état des PEs. Des registres d'un bit (nommés tokens) ont donc été implémentés dans chaque PE pour le stockage et le traitement de ces résultats de requêtes. On en distingue quatre :

- **R-token** : token stockant les résultats de l'ALU. Par exemple l'égalité ou les inégalités entre les opérandes, la retenue d'une addition ou d'une soustraction, la sortie d'un décalage etc. L'arbitrage peut être effectué sur la page des R-tokens pour ne choisir qu'un seul PE parmi plusieurs qui auraient *répondu* à une requête.
- **B-token** : il s'agit de la même fonctionnalité appliqué aux PEs configurés en mode 8 bits, on a donc un token par octet plutôt qu'un token par PE.
- **I-token** : ce token peut être affecté en fonction du contenu du R-token, il a pour effet d'empêcher un ordre émanant du contrôleur d'être exécuté par le PE. Ce token d'inhibition permet d'effectuer des traitements particuliers sur des sous-ensembles de PEs donc dans des parties de pages seulement.
- **E-token** : ce token contrôle l'émission des PEs sur le bus de diffusion (bus de données). Un seul PE peut émettre à la fois sur ce bus sinon des courts-circuits peuvent endommager les circuits, il a donc été installé une page de tokens particulière afin de s'assurer que seule un PE émette à la fois.

### Le modèle de programmation L1

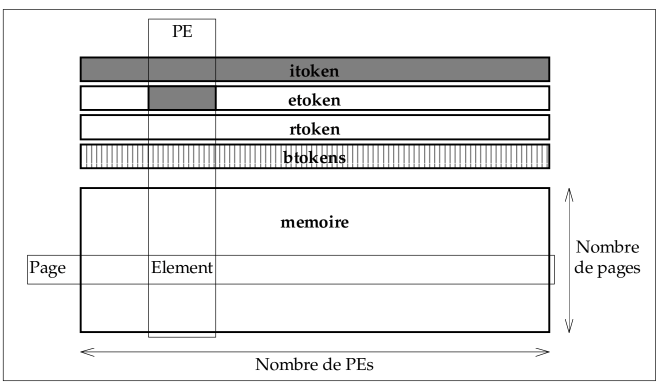


Figure 2 : Modèle des ressources de la carte.

La figure 2 montre un représentation abstraite des ressources de la carte accélératrice Rapid-2. Toutes ces ressources sont réparties dans les PEs, cela permet les traitements parallèles. Nous utilisons deux types principaux de données : « Page » et « Token ». Les types « Token » servent à définir un certain nombre de variables globales utilisées implicitement durant les opérations sur les variables de type « Page ».

**Sémantique associée aux variables pages**

**Le type « Page »**

Les variables de ce type sont implicitement des vecteurs d'éléments. Une variable scalaire de ce type correspond à une matrice à une dimension d'éléments. Chaque élément de ce vecteur appartient à un PE. La taille de ce vecteur est une constante définie par le nombre de PEs de la carte accélératrice. La déclaration de la variable suivante dans le langage L1 est :

```
Page var1;
```

Qui est équivalante à la déclaration suivante en langage C :

```
Element var1[PE_NUMBER];
```

Où *PE_NUMBER* est le nombre de PEs de la carte accélératrice. Une opération sur une variable page, s'applique implicitement sur tous les éléments de la page. Cela permet d'intégrer le mode de fonctionnement SIMD où une même instruction est exécutée simultanément dans tous les PEs. La sémantique des opérations sur les variables page sera présentée plus loin.

**Les tokens**

Plusieurs types « Token » sont utilisés pour modéliser le comportement des variables tokens. Tous ces types sont implicitement des vecteurs de variables Booléennes manipulés comme des scalaires. Les variables suivantes sont des variables globales prédéfinies dans le langage L1 :

```
IToken itoken;
EToken etoken;
RToken rtoken;
BTokens btokens;
```

Ces déclarations correspondent aux suivantes en langage C :

```
Boolean itoken[PE_NUMBER];
Boolean etoken[PE_NUMBER];
Boolean rtoken[PE_NUMBER];
Boolean btokens[PE_NUMBER][4];
```

Dans les déclarations précédantes, le type *Boolean* est défini par :

```
typedef enum Boolean{false=0, true=1};
```

Les types « Token » n'existent que pour permettre la déclaration de quelque variables globales. Aucune autre variable de ce type ne peut être déclarée, car ces variables sont associées à des ressources matérielles spécifique de la carte.

**Opérations sur les pages**

En L1, toutes les opérations utilisant des variables page sont des opérations vectorielles conditionnées. Voici un exemple d'opération sur des variables pages :

```
page_d=op(page_s1, …, page_sn, scalar_s1, …, scalar_sm);
```

Cette opération peut-être convertie en langage C comme suit :

```
for(index_pe=0;index_pe<PE_NUMBER;index_pe++)
 if(itoken[index_pe])
  page_d[index_pe]=op(page_s1[index_pe], …, page_sn[index_pe], scalar_s1,
   …, scalar_sm);
```

En L1, la boucle sur les PEs est masquée et l'utilisation du I-token pour le contrôle de l'affectation de la variable destinataire est implicite.

**Eléments de la mémoire**

Les éléments de la mémoire sont des variables structurées de 38 bits. Comme le montre la figure 3, elles sont composées de deux parties, une première partie de 32 bits de données (*data*) et une seconde partie de 6 bits pour des marqueurs (*tag*). La partie donnée peut contenir quatre entier de 8 bits ou un entier de 32 bits. Cette structure correspond à la définition en langage C suivante :

```
struct Element{Data data; Tag tag;};
union  Data{unsigned char bytes[4]; signed long   slong;};
struct Tag{Boolean tag[5]; Boolean used;};
```

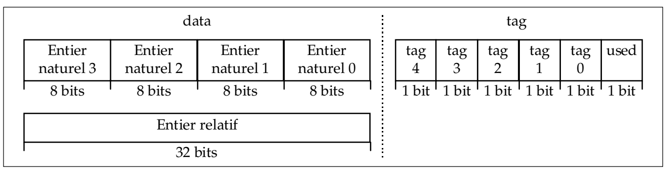


Figure 3 : Structure d'un élément de la mémoire.

**Opérations 8 bits sur les pages**

La partie opérative du PE permet l'exécution de la même opération en parallèle au niveau des octets. Les opérations 8 bits sont groupées par quatre dans chaque PE 32 bits, l'affectation de chaque octet est conditionnée par les B-tokens. Voici un exemple d'opération 8 bits sur des variables pages :

```
page_d=op(page_s1, …, page_sn, scalar_s1, …, scalar_sm);
```

Cette opération peut-être convertie en langage C comme suit :

```
for(index_pe=0;index_pe<pe_number;index_pe++)
 if(itoken[index_pe])
  for(index_byte=0;index_byte<4;index_byte++)
   if(btokens[index_pe][index_byte])
    page_d[index_pe].data.bytes[index_byte]=
     op(page_s1[index_pe].data.bytes[index_byte], …,
     page_sn[index_pe].data.bytes[index_byte], scalar_s1, …, scalar_sm);
```

Comme pour les opérations 32 bits, L1 masque la boucle sur les PEs et utilise implicitement les I-tokens et les B-tokens pour contrôler l'affectation des octets. On peut voir que L1 simplifie l'écriture d'application pour la carte accélératrice Rapid-2.

**Fonctions spéciales.**

En plus des opérations arithmétique et logique habituelles, il existe certaines opérations spéciales. Les opérations décrites précédemment s'appliquaient indépendamment dans tous les

PEs. Dans certains cas des opérations prédéfinies où la sémantique dépend du numéro d'ordre du PE sont nécessaires. Voici quelques exemples de telles fonctions :

**Fonctions sur les pages**

Les éléments de la mémoire n'étant pas adressable individuellement, les échanges quelconque entre éléments de la mémoire ne sont pas possible. Il existe en L1 un certain nombre de fonctions permettant des échanges réguliers. Ces fonctions utilisent le lien connectant chaque PE à ses deux voisins immédiats dans le vecteur de PEs. Ces fonctions sont :

- **rl_mesh(Page)** : (*Rotate Left through the MESH*), chaque élément de la page est envoyé au PE de gauche. L'élément le plus à gauche est envoyé au PE le plus à droite.
- **rr_mesh(Page)** : (*Rotate Right through the MESH*), chaque élément de la page est envoyé au PE de droite. L'élément le plus à droite est envoyé au PE le plus à gauche.

Pour ces fonctions, les éléments sont enregistrés uniquement dans les PEs actifs (les PEs ayant leur I-token mis à un). Des fonctions similaires existent pour les tokens.

**Fonction globale sur le R-token**

- **propagate(rtoken)** : met à un uniquement le R-token des PEs à droite du premier PE en partant de la gauche ayant son R-token mis à un.
- **arbitrate(rtoken)** : met à un uniquement le R-token du premier PE en partant de la gauche ayant son R-token mis à un. L'arbitrage défini un ordre sur les éléments contenu dans les PEs.
- **unmark(etoken)** : met à zéro le R-token du PE ayant sont E-token mis à un.

**Fonctions de communications**

Ces fonctions permettent les communications entre l'ordinateur hôte et la carte accélératrice :

- **spread(value)** : envoie la valeur à tous les PEs.
- **extract(page)** : récupère la valeur contenue dans la page indiquée du PE ayant son E-token mis à un.

**Environnement de Développement L1**

Un programme L1 peut-être soit émulé dans un environnement C++, soit compilé en un programme pour l'ordinateur hôte et un microprogramme Rapid-2. Il est très utile de pouvoir émuler et compiler les programmes L1. L'émulation permet de contrôler le comportement d'un programme sans avoir à passer par le compilateur L1. Cela permet de comparer les résultats issus de l'émulation et de l'exécution du code compilé et donc de valider l'environnement de développement.

L'application une fois compilée peut-être utilisée soit avec la carte Rapid-2 soit avec le simulateur ROLLS. ROLLS est un simulateur cycle à cycle de la carte Rapid-2, écrit en C++. La carte Rapid-2 étant en cours de réalisation, nous utilisons ROLLS pour simuler la carte.

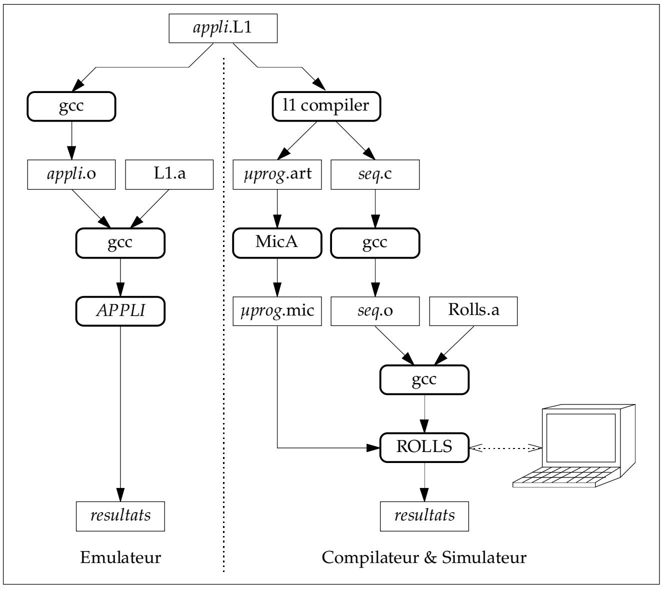


Figure 4 : L'environnement de développement.

Dans la figure 4, *gcc* est un compilateur C++, *MicA* est un micro-assembleur pour la carte Rapid-2, *L1.a* est une librairie de fonctions en C++ d'émulation du comportement des opérations L1 et *Rolls.a* est une librairie en C++ contenant le simulateur de la carte Rapid-2.

### L'émulateur

L'émulateur est composé d'une librairie en C++ contenant la définition de plusieurs classes. Pour chaque type spécifique au langage L1, une classe décrit sa structure et le comportement de toutes les opérations concernant ce type. Cela permet d'obtenir un exécutable, qui émule le programme L1, en compilant et liant celui-ci avec la librairie *L1.a*. Cette méthode ne permet toutefois pas de vérifier toute les erreurs de syntaxes du programme L1, elle donne seulement une émulation du comportement du programme à condition que celui-ci soit correct au niveau de la syntaxe. Pour la vérification complète de la syntaxe le compilateur L1 est nécessaire.

### Le compilateur

Le compilateur doit accomplir deux tâches principales. En premier lieu, il doit séparer le programme L1 en deux processus, un processus pour l'ordinateur hôte et un pour la carte accélératrice. Le processus de l'ordinateur hôte sera un programme séquentiel en langage C, il sera extrait du programme L1 et complété par des appels à des fonctions de communication de bas niveau. Le processus de la carte accélératrice est composé de toutes les opérations concernant les types spéciaux de L1 (« Page », « IToken », « EToken », …). Le compilateur devra alors convertir le processus de la carte accélératrice en un microprogramme.

### Etat d'avancement

Les deux composants VLSI de la carte ont été développés au laboratoire MASI à l'aide de la chaîne de CAO du Laboratoire, la chaîne Alliance. Cette chaîne de CAO complète comprend des outils de synthèse logique et un compilateur de chemin de données, qui ont été utilisés dans la conception des circuits. Elle est distribuée dans plus de 200 Universités à travers le monde, suivant les modalités de licence de la *Free Software Foundation*. Le projet

RAPID-2 est l'un des divers projets d'architecture du laboratoire MASI, qui sont utilisés de façon interne au laboratoire pour expérimenter cette chaîne de CAO.

Une version d'essai du circuit d'exécution a été réalisée, elle comporte 4 processeurs élémentaires 38 bits et 170 000 transistors. Le circuit a été fabriqué au CMP et mesure 60 mm² en technologie CMOS 0.8 microns. Il a été testé et il est valide. Le circuit de contrôle a également été réalisé dans la même technologie. Il contient 20 Kb de mémoire de microprogramme. Il a un mécanisme de résistance aux pannes utilisant une redondance de 20% de la RAM. Il mesure 55 mm² pour 160 000 transistors et il est en cours de test. Une carte PCI supportant un circuit de contrôle et plusieurs circuits d'exécution est en cours de développement au sein du laboratoire.

Le simulateur cycle par cycle de la carte et l'émulateur de L1 sont disponibles. Le compilateur est en cours de développement.

La carte RAPID-2 a été simulée de façon précise pour une application de comparaison de séquences génétiques (chaînes d'ADN ou de protéines), selon l'algorithme de Smith et Waterman [7]. Cette application est très calculatoire et peut nécessiter plusieurs années de temps CPU [8]. La taille des bases de séquences avec lesquelles doit se faire la comparaison augmente exponentiellement, à un rythme au moins égal à celui des progrès dans les performances des stations de travail. Cette application a été entièrement microprogrammée.

Pour cette application, le gain de performances par rapport à une exécution séquentielle sur une station de travail Sparc-10 peut atteindre un facteur 110. Ce gain de performances de deux ordres de grandeur est très important pour une carte dont l'ordre de grandeur du coût est celui d'un micro-ordinateur de type PC.

| **Nom** | **Bisp** | **Bioscan** | **Bioccelerator** | **Splash2** | **Rapid-2** |
|---|---|---|---|---|---|
| **Architecture** | 1 carte de 16 circuits de 16 UE | 1 carte de 16 circuits de 812 UE 1 bit | 1 tower de 16 FPGA X4008 avec RAM | 16 cartes de 16 FPGA X4010 512 Ko RAM | 1 Carte de 16 ASICs de 16 UE 32 bits |
| **Fréquence et bus** | 12,5 MHz DMA 3 Mo/s | 32 MHz VME | 10 MHz VME | 54 Mo/s SBUS | 33 MHz PCI |
| **Algorithme supporté** | **Smith & Waterman** | Algo. simplifié | GCG Profile_search | **Smith & Waterman** | **Smith & Waterman** |
| **Performances** | 768 MCE/s | 26 000 MCE/s | 320 MCE/s | 56 MCE/s | 50 MCE/s |
| **Program-mabilité** | néant | néant | par synthèse | VHDL, par synthèse | langage haut niveau |

Figure 5 : Performance de diverses machines pour la comparaison des séquences génétiques.

La figure 5 montre les performances de plusieurs machines dédiées au génome, comparées à notre machine. MCE/s signifie Méga Comparaisons Elémentaire par seconde. Cette mesure donne une idée des performances crête des machines utilisées. Rapid-2, pour un coût moindre, offre des performances comparables aux machines qu'utilisent les généticiens. La différence des performances crêtes ne se retrouve pas forcement dans les temps de réponse, en raison du coût des entrées sorties. Si c'est le cas, elle représente le prix d'une programmabilité complète. Celle-ci permet aux biologistes de choisir l'algorithme le plus approprié.

## Conclusion

Le projet RAPID du laboratoire MASI a développé une carte accélératrice pour PC, fondé sur une architecture massivement parallèle SIMD. Les composants VLSI de la carte ont entièrement été développés à l'aide de la chaîne de CAO Alliance du laboratoire. Le composant d'exécution a d'ores et déjà été testé avec succès. La carte est également en cours de développement, ainsi que l'environnement de développement. Le simulateur cycle par cycle et l'émulateur du langage L1 sont disponibles.

La carte a été microprogrammée et simulée pour une application de calcul scientifique, destinée à la comparaison de séquences génétiques. Le gain de performances est de plus de deux ordres de grandeur par rapport à une station de travail Sparc-10. Des gains de performances plus élevés peuvent être obtenus avec des machines dédiées ou plus coûteuses, mais ce gain de performances est très élevé pour une carte accélératrice dont le coût est comparable à celui d'un PC. D'autre part cette carte est entièrement programmable dans un langage de haut niveau, ce qui n'est pas le cas d'autres architectures.

L'expérience de RAPID-2 confirme l'intérêt des cartes additionnelles massivement parallèles, comme Cnaps et IMAP développées par d'autres équipes. Au cours des prochaines années, ces cartes pourraient augmenter de façon très importante le ratio performances / coût pour des applications ne nécessitant que du calcul entier, une taille de mémoire modérée et un débit d'entrées-sorties relativement limité. Ces propriétés paraissent particulièrement réunies pour des applications de calcul scientifique sur des entiers, ou pour le filtrage de données en temps réel (flots d'images, flots de textes).

## Références


[1] Lopresti, D.P., "P-NAC : A systolic Array for Comparing Nucleic Acid Sequences", Computer, vol 20, pp 98-99, july 1987

[2] Skinner T., "Harness Multiprocessing Power for DSP Systems", Electronic Design 2/7/94 issue

[3] Herman G., Sodini C., "A dynamic associative processor for machine vision application", IEEE Micro, June 1992, pp 31-41

[4] Kung, H. T. "Why Systolic Architectures ?" Computer, Vol. 15, pp 37 - 46, January 1982.

[5] Archambaud D., Faudemay P., "An arbitration tree adapted to object-oriented associative memories", ICCD'94, Cambridge - Massachus-sets, pp 306-310, October 1994

[6] Yamashita N. et al., "A 3.84 Gips integrated memory array processor with 64 processing elements and a 2-Mb SRAM", IEEE Solid-state Circuits, Nov. 1994, vol 94 n°11, pp 1336-1342

[7] Waterman, M. S., Smith, T. F., and Beyer, W. A. "Some biological sequences metrics". Adv. in Math. 20, pp. 367 - 387. 1976.

[8] Gonnet G.H. et al., "Exhaustive matching of the entire protein sequence database" in Science vol. 256, 5 June 1992