Réseaux systoliques spécifiques à base du processeur API15C

Patrice Frison, Eric Gautrin, Dominique Lavenier, Jean-Luc Scharbarg

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1227

Programme 2
Structures Nouvelles d'Ordinateurs

RESEAUX SYSTOLIQUES SPECIFIQUES A BASE DU PROCESSEUR API15C

Patrice FRISON
Eric GAUTRIN
Dominique LAVENIER
Jean-Luc SCHARBARG

Mai 1990

RR-1227

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Réseaux systoliques spécifiques à base du processeur API15C

Designing specific systolic arrays with the API15C chip

Patrice Frison, Eric Gautrin, Dominique Lavenier, Jean-Luc Scharbarg

Mars 1990
Publication Interne n° 529 - 26 Pages

résumé

API15C est un processeur systolique VLSI étudié et réalisé à l'IRISA. Cette puce de 45.000 transistors (CMOS $2\mu m$) a été conçue comme une brique de base pour être exploitée par différentes architectures systoliques (linéaires, bidimensionnelles, ...) de type SIMD.

C'est un processeur 16 bits disposant de 3 ports d'entrées/sorties parallèles. Etant prévu pour fonctionner dans un environnement SIMD, il n'intègre pas de séquenceur interne. Par contre, son jeu d'instructions comporte quelques opérations spécifiques permettant d'exécuter, d'une manière élégante, un certain nombre d'instructions conditionnelles. API15C est piloté par une horloge à 2 phases non recouvrantes et exécute une instruction par cycle à une fréquence maximum de 10 MIPS.

Ce papier montre comment API15C peut être utilisé dans diverses architectures systoliques. Nous montrons, en particulier, qu'il est très bien adapté pour résoudre des problèmes tels que la transmission de valeurs initiales aux processeurs périphériques d'un tableau de processeurs. Nous décrivons également 2 réalisations à base du processeur API15C. La première est un réseau linéaire de 18 éléments dont la vocation est de couvrir une large gamme d'applications. La seconde est un réseau bidimensionnel tronqué, spécialement adapté à un type de calcul particulier : la comparaison de chaînes de caractères.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

abstract

API15C is a VLSI systolic processor which has been designed at IRISA. This programmable CMOS 2 μm chip (45.000 transistors) is a building block for either linear or bi-dimensional SIMD systolic architectures.

This chip is a 16-bit processor with 3 parallel I/O ports. It has been designed to operate in a SIMD environment and does not incorporate an instruction sequencer. However, its instruction set includes some special instructions allowing the processor to execute conditional operations in a fast and easy way. API15C is driven by a 2-phase nonoverlapping clock and can execute one instruction per phase at a rate of 10 MIPS.

In this paper we show how to use API15C to design various systolic arrays. In particular, we show that it is very convenient to solve tricky problems such as the transmission of initial values for the boundary processors of a systolic array. Finally, we describe 2 prototype arrays designed with API15C chips. The first one is a linear array of 18 elements used for multiple applications. The second one is a truncated bi-dimensional systolic structure for intensive string comparison computation. In particular, an interface board used to control both arrays is described.

1 Introduction

Pour des domaines d'application tels que le traitement d'image, le traitement du signal ou le calcul numérique, les architectures systoliques présentent, dans de nombreux cas, d'excellentes solutions [8, 11]. De plus, la régularité des calculs mis en jeu est adaptée à une implémentation sur des architectures de type SIMD. Cette dernière propriété facilite grandement le contrôle général du réseau, en particulier au niveau des communications entre processeurs qui se réalisent alors de manière totalement synchrone.

Notre objectif principal a été de concevoir un processeur pouvant être utilisé comme une brique de base permettant de réaliser des architectures systoliques de topologies diverses mais dont le mode commun de fonctionnement est exclusivement de type SIMD. Afin de supporter une gamme d'applications relativement conséquente, l'élément de base se devait de posséder les propriétés suivantes :

- *mono-puce :* indispensable pour atteindre des performances en vitesse appréciables et pour que le réseau puisse tenir sur une seule carte,
- *programmable :* la diversité des applications visées l'exige,
- *exécution SIMD :* le contrôle de la machine et sa programmation s'en trouvent grandement simplifiés,
- *jeu d'instructions dédié :* nécessaire pour faciliter la programmation dans un environnement purement SIMD,
- *plusieurs ports d'entrées/sorties :* incontournable pour être en mesure de réaliser différentes architectures (1D, 2D, ...),
- *communications rapides entre processeurs :* essentiel pour ne pas ralentir le calcul, en particulier lorsqu'il s'agit de parallélisme à grain très fin.

En fait, il n'existe pas de processeurs du commerce regroupant toutes ces propriétés. Le transputer, par exemple, fonctionne uniquement en mode MIMD et est limité par la bande passante de ses liens dès qu'un parallélisme très fin est requis. Le processeur VLSI programmable, PSC, conçu à l'université de Carnegie Mellon [6] est plus proche de nos objectifs. C'est un processeur 8 bits avec 6 ports d'entrées/sorties parallèles. Il possède une UAL, 64 registres et un multiplieur avec accumulation du résultat. Ce processeur exécute son propre programme stocké dans une mémoire de micro-code de 60 bits. La principale différence avec notre approche réside donc dans la mode de fonctionnement d'un réseau construit à partir de ce processeur puisque chaque élément est autonome (mode MIMD). D'autres réalisations, telles que le Warp [7], ne correspondent pas aux spécifications précitées à cause de leur mode de fonctionnement, leur complexité ou leur taille. Soulignons que la version intégrée du Warp, le iWarp annoncé par intel [2], inclut la plupart des propriétés énoncées mais reste essentiellement tourné vers un mode de fonctionnement asynchrone,

même s'il existe la possibilité de faire travailler tous les processeurs d'un même réseau de manière synchrone.

Ce papier présente le processeur API15C. C'est un processeur 16 bits étudié pour concevoir différentes architectures systoliques. Il constitue, en fait, une brique de base pour des réseaux de type SIMD avec des communications synchrones et rapides entre processeurs. Ces deux derniers points sont relativement importants car ils permettent de simplifier notablement le processeur (pas de séquenceur interne, pas de FIFO, ...). Mais ces simplifications ne restreignent pas les possibilités du processeur : il a été montré que dans des domaines tels que le traitement du signal, le traitement d'image ou le calcul matriciel, par exemple, beaucoup d'applications peuvent être traitées sur des architectures sytoliques de type SIMD.

API15C est un circuit programmable avec un jeu d'instructions réduit incluant les opérations arithmétiques et logiques standard et les opérations de transfert usuelles. Conçu dans un but de recherche et pour des raisons de simplicité de réalisation, ce circuit n'intègre pas d'opérateurs flottants. Notre but principal est de montrer l'efficacité et les possibilités d'un tel élément de base.

Le circuit possède une UAL, 32 registres, un multiplieur, une mémoire de 256 mots de 16 bits et trois ports d'entrées/sorties parallèles sur 16 bits. Cette dernière caractéristique permet une communication rapide entre processeurs ainsi que différents schéma d'interconnexion.

Le jeu d'instructions a été étendu à des instructions plus spécifiques, appelées instructions *SIMD*. Elles procurent des solutions efficaces pour résoudre des problèmes simples tels que la gestion du contrôle, l'initialisation des processeurs périphériques ou l'émulation de connexions logiques. Elles apportent une souplesse de programmation appréciable dans un environnement SIMD composé d'un réseau de processeurs API15C. Ce dernier point est largement explicité dans ce papier.

La section suivante donne une brève description de l'architecture interne du processeur API15C. Nous décrivons ensuite comment le processeur peut être utilisé efficacement dans différentes architectures. Nous montrons, à l'aide d'un exemple adapté, la souplesse de programmation offerte par les instructions *SIMD*. Dans la section 4, deux machines systoliques réalisées à l'IRISA et à base de processeurs API15C sont présentées. Nous décrivons également de quelle manière les structures réalisées peuvent être connectées à un système hôte. Nous terminons sur quelques conclusions et perspectives.

2 Le processeur API15C

Le processeur API15C est une brique de base pour élaborer différents réseaux systoliques de type SIMD. Dans ces réseaux, le transfert de données entre processeurs revêt une importance considérable. C'est pourquoi, une attention toute particulière a été réservée à la gestion des entrées/sorties pendant la conception du circuit. De plus, le mode d'exécution SIMD pouvant être un facteur restrictif lors de la programmation d'une application, de nouvelles instructions ont été introduites de façon à s'affranchir des contraintes liées au mode SIMD.

Cette partie expose brièvement l'architecture interne du processeur API15C, le mécanisme d'entrées/sorties et le jeu d'instructions. Une description plus détaillée peut être trouvée dans [5, 10].

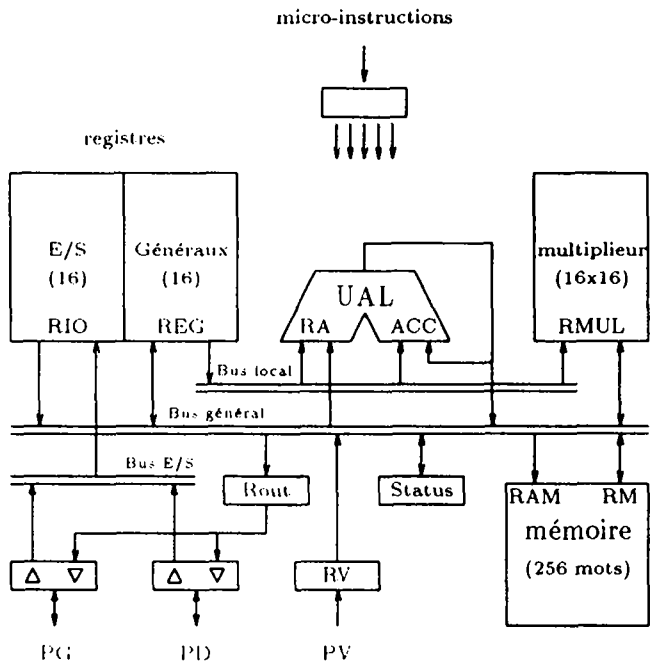

Figure 1 : architecture interne du processeur API15C

2.1 Architecture interne

Le circuit est organisé autour de 3 bus (16 bits) sur lesquels viennent se connecter différents modules fonctionnels. La figure 1 indique les chemins de données du processeur API15C. Un bus principal est chargé de fournir une connexion globale entre toutes les unités, un bus local favorise plus particulièrement la liaison entre les registres, l'UAL et le multiplieur, et un dernier bus (bus E/S) est dédié aux entrées/sorties. Nous donnons maintenant une description succinte des unités fonctionnelles qui composent le circuit.

L'Unité Arithmétique et Logique offre 16 instructions incluant les opérations arithmétiques élémentaires (addition, soustraction, incrémentation, ...) et les opérations logiques usuelles (NON, ET, OU, ...).

Les Registres sont divisés en deux bancs de 16 registres: le premier est dédié aux opérations d'entrées/sorties; le second est utilisé comme un banc de registres banalisés à double accès en lecture. Ce dernier est connecté à la fois au bus principal et au bus local pour générer simultanément deux opérandes vers l'UAL ou vers le multiplieur.

La Mémoire contient 256 mots de 16 bits. Elle est dynamique avec un mécanisme de

rafraîchissement automatique. Elle inclut également un système d'auto-incrémentation pour accéder à des données stockées de manière contigüe en mémoire.

Le Multiplieur effectue des multiplications en complément à deux [1]. Le résultat est obtenu sur 32 bits et lu en deux cycles, les poids faibles en premier, puis les poids forts.

2.2 Mécanisme d'entrées/sorties

Les communications entre processeurs sont très importantes au sein d'un réseau systolique. Le mécanisme associé à chaque processeur doit donc être particulièrement efficace et rapide. La solution retenue a été d'effectuer les transferts en parallèle sur 16 bits. Trois ports de données sont disponibles :

- un port d'entrée sur 16 bits, noté PV,
- deux ports bidirectionnels sur 16 bits (PG et PD). Quand l'un est positionné en entrée, l'autre l'est en sortie.

Lors d'un transfert d'une unité fonctionnelle à l'autre, le port sélectionné en sortie présente systématiquement le contenu du bus principal. Ce mécanisme permet de transférer une donnée vers un processeur voisin sans adjonction de cycles supplémentaires.

2.3 Jeu d'instructions

Puisqu'API15C a été conçu pour fonctionner dans un environnement SIMD, il n'intègre pas de séquenceur interne et, bien évidemment, ne dispose pas d'instructions de contrôle. Ces instructions sont générées par un séquenceur externe et diffusées à tous les processeurs du réseau. Le jeu d'instructions est divisé en quatre classes principales :

- les instructions de transfert entre unités,
- les instructions UAL et de multiplication,
- les instructions d'accès à la mémoire,
- les instructions d'entrées/sorties.

Toutes les instructions sont exécutées en un seul cycle. De plus, grâce à un certain parallélisme au niveau du jeu d'instructions, le circuit peut effectuer simultanément une opération UAL et un accès mémoire. A ces opérations de base viennent s'ajouter de nouvelles instructions, appellées *instructions SIMD*, qui permettent en un seul cycle de résoudre des problèmes de test simples tels que :

si *condition* **alors** x:=y **sinon** x:=x+y

La condition est issue d'un registre particulier, le *registre d'état*, positionné à chaque opération effectuée par l'UAL.

3 Conception de réseaux systoliques

3.1 Introduction

Cette partie a pour but de montrer comment le circuit API15C peut être utilisé pour concevoir différentes architectures systoliques. Pour illustrer notre propos, nous essayons, tout au long de cette partie, de décrire l'implémentation d'un même algorithme sur les différentes topologies proposées.

L'exemple choisi concerne le calcul de la distance de Levenshtein entre deux chaînes de caractères [12]. Il s'agit de comparer une chaîne test avec une chaîne référence, sachant que des erreurs d'insertions, d'omissions ou de substitutions sont éventuellement présentes.

Plus formellement, le problème se présente de la manière suivante: soit $T = (t_1, t_2, ..., t_i, ..., t_n)$ et $R = (r_1, r_2, ..., r_j, ..., r_m)$ les deux chaînes à comparer. La distance de Levenshtein est obtenue par la relation de récurrence:

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d(t_i, r_j) \\ D(i-1,j) + K_a \\ D(i,j-1) + K_o \end{cases} \quad (1)$$

avec les conditions initiales:

$D(0,0) = 0$
$D(i,0) = D(i-1,0) + K_a$ *pour* $1 \leq i \leq n$
$D(0,j) = D(0,j-1) + k_o$ *pour* $1 \leq j \leq m$

où $D(t_i, r_j)$ représente le coût de substitution de r_j par t_i, K_a le coût d'insertion de t_i et K_o le coût d'omission de r_j.

Dans une application typique, telle que la correction de fautes, par exemple, ce calcul doit être répété un grand nombre de fois puisqu'un mot erroné (la chaîne test) doit être comparé à toutes les références d'un dictionnaire. La somme de calculs à effectuer est donc très importante et limite l'usage des processeurs conventionnels à de petits dictionnaires. Cependant, le calcul de cette distance particulière présente des propriétés de régularité intéressantes qui sont avantageusement exploitées sur des architectures systoliques.

Dans un premier temps, nous décrivons le calcul de la distance de Levenshtein sur un réseau 2D complet. Nous montrons, ensuite son implémentation sur un réseau linéaire puis sur un réseau spécialement "taillé" pour ce problème particulier. A chaque étape, les différentes transformations sont expliquées et l'algorithme de chaque processeur est donné. Ceci constitue la première partie.

Dans un second temps, nous cherchons à raffiner la mise en œuvre pour obtenir de meilleures performances. L'idée consiste à répartir le calcul de base sur 2 processeurs en introduisant un pipeline. Les mêmes architectures sont reprises et les modifications logicielles étudiées. Ces différentes implémentations montrent, d'une part, les possibilités d'interconnexion du processeur API15C et, d'autre part, confirment l'efficacité des instructions SIMD proposées dans le jeu d'instructions du processeur.

3.2 Modèle systolique de base

3.2.1 Calcul d'une distance

La distance $D(i,j)$ peut être représentée graphiquement comme l'intersection de la $j^{ième}$ ligne et de la $i^{ième}$ colonne d'une matrice $(n \times m)$. L'idée de base est d'associer un processeur au calcul de chacun des $D(i,j)$ et de construire un réseau de $(n \times m)$ processeurs, comme le montre la figure 2. La chaîne test T est diffusée verticalement (un caractère par colonne), tandis que la chaîne référence R circule horizontalement (un caractère par ligne). Chaque processeur (i,j) dispose de cinq ports logiques d'entrée PEH, PED, PEV, PER et PET, qui reçoivent respectivement les distances $D(i-1,j)$, $D(i-1,j-1)$, $D(i,j-1)$ et les caractères R_j et T_i. Il dispose également de cinq ports logiques de sortie PSH, PSD, PSV, PSR et PST pour diffuser la distance $D(i,j)$ à ses trois voisins et propager les caractères R_j et T_i.

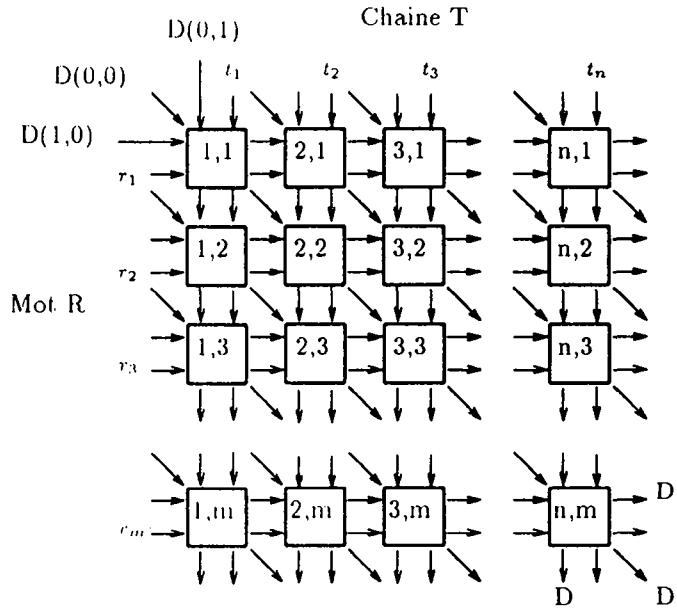

Figure 2 : graphe de calcul de la distance

3.2.2 Enchaînement des calculs

Le réseau systolique décrit au paragraphe précédent n'est pas efficace, en raison des dépendances de données entre les processeurs. Il est en effet aisé de montrer que la diagonale de processeurs (i,j), telle que $i+j-1=t$, est la seule active au cycle systolique t. Cependant, les traitements de plusieurs mots référence peuvent être enchaînés, de telle sorte que le calcul d'une nouvelle distance débute à chaque cycle systolique [3]. Le processeur (n,m) délivre

ainsi une nouvelle distance à chaque cycle systolique. Dans cette approche, les processeurs exécutent la boucle suivante (appelée cycle systolique) :

début { **cycle systolique fonctionnel** }
$écrire(PSD, Dt)$; $lire(PED, Dd)$;
$écrire(PSV, D)$; $lire(PEV, Dv)$;
$écrire(PSH, D)$; $lire(PEH, Dh)$;
$écrire(PSR, r)$; $lire(PER, r)$;
$écrire(PST, t)$; $lire(PET, t)$;
$Dt = D$;
$D = min(Dd + d(t, r), Dh + K_a, Dv + K_o)$;
fin { **cycle systolique fonctionnel** }

Dd, Dh et Dv représentent respectivement $D(i-1, j-1)$, $D(i, j-1)$ et $D(i-1, j)$. Dt est une variable intermédiaire qui insère un retard d'un cycle systolique pour le transfert diagonal de D. Ce programme est directement déduit de l'équation 1 et il suppose que des valeurs initiales correctes sont envoyées aux processeurs des bords (cf équation 1).

Deux simplifications peuvent être apportées aux entrées/sorties de ce programme. D'abord, il n'est pas nécessaire de transmettre les caractères test t_j. En effet, dans une application réelle, une chaîne test est comparée à un grand nombre de mots référence. En conséquence, chaque processeur reçoit sa valeur de t au début du traitement de la chaîne test, et la mémorise. La deuxième simplification concerne les ports diagonaux (PED et PSD). Un retard devant être introduit entre PSD et PED, il n'est pas nécessaire de mettre en œuvre une connexion diagonale directe entre les processeurs. La connexion diagonale peut être remplacée "gratuitement" par une connexion horizontale, suivie d'une connexion verticale (ou vice-versa).

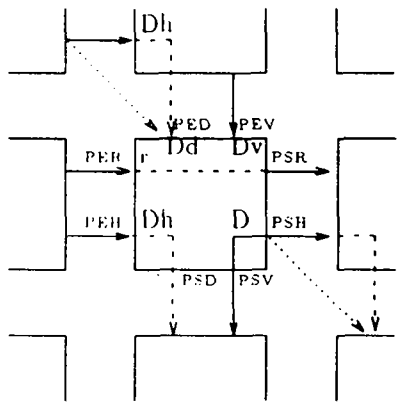

Figure 3 : connexions d'un processeur élémentaire

La figure 3 montre comment une connexion diagonale est obtenue. Une valeur D devant être transférée diagonalement est d'abord envoyée horizontalement et reçue sous le nom

de Dh, puis transmise verticalement et reçue sous le nom de Dd. La variable Dt est ainsi remplacée avantageusement par la variable Dh. Le cycle systolique devient donc :

début { cycle systolique sans connexion diagonale }

```
1   écrire(PSD, Dh);     lire(PED, Dd);
2   écrire(PSV, D);      lire(PEV, Dv);
3   écrire(PSH, D);      lire(PEH, Dh);
4   écrire(PSR, r);      lire(PER, r);
5   Rv = Dv + K_v;
6   Rh = Dh + K_h;
7   D  = (Rv < Rh ? Rv : Rh);
8   Rd = MEM[r];
9   Rd = Dd + Rd;
10  D  = (D < Rd ? D : Rd);
```

fin { cycle systolique sans connexion diagonale }

Dans ce programme, le coût $d(t, r)$ est lu dans un tableau bidimentionnel stocké dans la mémoire de chaque processeur, au début du traitement d'une chaîne test. En fait, t étant constant pour chaque processeur, seule la colonne du tableau correspondant à t est stockée dans la mémoire du processeur, d'où la notation $MEM[r]$. Par ailleurs, les lignes 7 et 10 du programme illustrent l'utilisation d'instructions conditionnelles pour le calcul de minimum.

3.2.3 Réseau bidimensionnel

La mise en œuvre effective du réseau systolique avec API15C (figure 4) s'avère plus simple que celle de la figure 2 notamment pour ce qui concerne le schéma de connexion. En effet, différents types de données peuvent transiter sur une même connexion physique, grâce à un multiplexage temporel. API15C utilise trois ports PG, PD et PV. PG émule les ports logiques PEH et PER, PD émule PSH, PSV, PSR et PSD, et enfin PV émule PEV et PED.

La mise en œuvre d'un réseau systolique bidimensionnel pose un problème délicat de gestion d'entrées/sorties de données. En effet, une grande quantité de données doit être fournie au réseau pour ne pas ralentir le calcul. Un mécanisme d'entrées/sorties performant est donc nécessaire. API15C offre une alternative logicielle pour réduire la quantité de données à envoyer au réseau.

Pour le calcul de la distance de Levenshtein, les processeurs des bords doivent en effet recevoir des valeurs initiales (cf equation 1) en lieu et place des distances calculées par leurs voisins (ces voisins n'existent pas). Il existe donc différents types de processeurs, caractérisés par la nature des valeurs qu'ils reçoivent (initiales ou calculées). Trois booléens sont utilisés pour chaque processeur : V, H et O indiquent que les valeurs reçues respectivement verticalement, horizontalement et diagonalement ont été calculées, comme le montre la figure 4. Le cycle systolique devient :

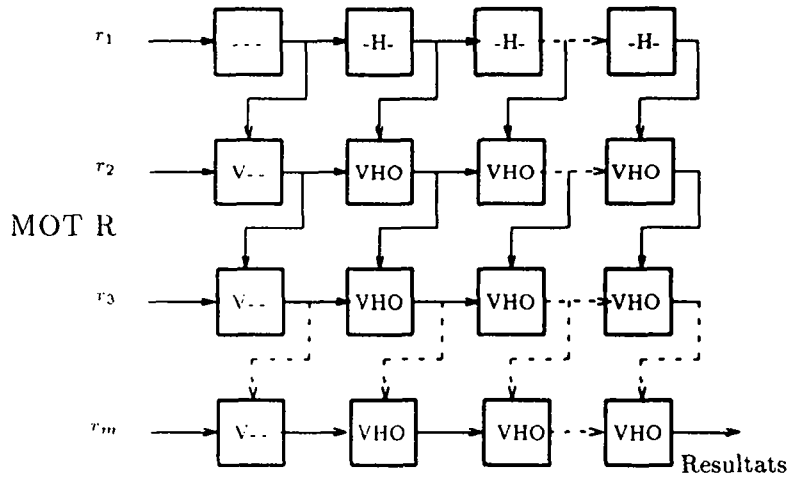

Figure 4 : connexions d'un processeur élémentaire

début { cycle systolique bidimentionnel }

```
1    écrire(PD, Dh);      lire(PV, Dd);
2    écrire(PD, D);       lire(PV, Dv);
3    écrire(PD, D);       lire(PG, Dh);
4    écrire(PD, r);       lire(PG, r);
5    Rv = (V ? Dv + Kv : Iv + Kv);
6    Rh = (H ? Dh + Ka : Ih + Ka);
7    D  = (Rv < Rh ? Rv : Rh);
8    Rd = MEM[r];
9    Rd = (O ? Dd + Rd : Id + Rd);
10   D  = (D < Rd ? D : Rd);
```

fin { cycle systolique bidimensionnel }

Iv, Ih et Id sont des constantes. Elles contiennent une valeur initiale qui devrait être envoyée au réseau. Le type du processeur est codé dans le *registre d'état,* qui est utilisé par les instructions SIMD. Les lignes 5, 6 et 9 contiennent un chargement conditionnel (le processeur choisit, parmi deux registres, celui qui doit être transféré à l'UAL). Dans le cas présent, l'instruction choisit entre un registre d'entrée et un registre général contenant une valeur d'initialisation.

Cet exemple montre simplement comment le flot de données nécessaire à un calcul systolique peut être fortement réduit si des valeurs initiales doivent être envoyées au réseau. Le mécanisme logiciel proposé est très efficace, puisqu'il n'augmente pas la durée du cycle systolique.

3.2.4 Réseau linéaire

API15C peut aussi être utilisé pour construire des réseaux linéaires. L'alimentation du réseau est beaucoup plus simple. En contre-partie, les algorithmes mis en œuvre sur de telles structures doivent résoudre divers problèmes, comme les réinitialisations partielles.

La distance de Levenshtein peut se calculer sur une structure linéaire. En effet, lors du calcul d'une distance, une seule diagonale du tableau de processeurs de la figure 4 est active à un instant donné. Chaque processeur d'une structure linéaire émule donc une colonne de processeurs de la structure bidimensionnelle. Chaque processeur mémorise un caractère test, tandis que les caractères référence circulent de gauche à droite (un nouveau caractère référence est injecté dans le réseau à chaque cycle systolique), comme le montre la figure 5.

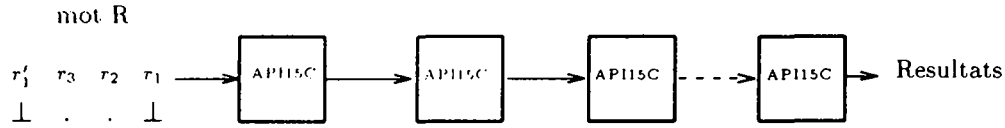

Figure 5 : réseau linéaire avec API15C

Lorsque le premier processeur a traité le dernier caractère d'un mot référence, il est disponible, après réinitialisation, pour le traitement d'un nouveau mot référence. Dans cette optique, un symbole m est associé à chaque caractère référence r. m est appelé un jalon et vaut $\perp$ lorsqu'il est associé au premier caractère d'une chaîne référence. Il est utilisé pour réinitialiser le processeur pour le calcul d'une nouvelle distance. Le cycle systolique devient :

début { **cycle systolique linéaire** }
1 $Dd = Dh$;
2 $Dv = D$;
3 $écrire(PD, D)$; $lire(PG, Dh)$;
4 $écrire(PD, r)$; $lire(PG, r)$;
4.1 $écrire(PD, m)$; $lire(PG, m)$;
5 $Rv = (m \neq \perp\ ?\ Dv + K_v\ :\ Iv + K_v)$;
6 $Rh = Dh + K_h$;
7 $D = (Rv < Rh\ ?\ Rv\ :\ Rh)$;
8 $Rd = MEM[r]$;
9 $Rd = (m \neq \perp\ ?\ Dd + Rd\ :\ Id + Rd)$;
10 $D = (D < Rd\ ?\ D\ :\ Rd)$;
fin { **cycle systolique linéaire** }

La numérotation des lignes de ce programme est conservée par rapport à celle du cycle systolique bidimensionnel. De petits changements sont à remarquer. Les transferts verticaux des lignes 1 et 2 sont remplacés par des affectations de variables, et la ligne 4.1 effectue le transfert du jalon.

Le processus de réinitialisation consiste à affecter aux variables concernées des constantes quand le jalon vaut $\perp$. Cela est obtenu de la même manière que sur la structure bidimensionnelle, en multiplexant les opérandes suivant la valeur du *registre d'état*. La condition dépend des données et doit être calculée à chaque cycle systolique (lignes 5 et 9).

3.2.5 Réseau diagonal

Cette section présente l'utilisation d'API15C pour réaliser une machine bidimensionnelle spécialisée pour l'algorithme de correction. Dans certaines applications, un mot test contient au plus une insertion ou une omission de caractères. La structure bidimensionnelle peut donc être réduite à trois diagonales, comme le montre la figure 6. Le schéma de connexion de ce réseau est différent de celui de la structure bidimensionnelle. Les connexions verticales, horizontales et diagonales sont directement mises en œuvre, grâce aux ports bidirectionnels PD et PG. Un transfert vertical est réalisé en envoyant une donnée sur le port gauche PG (configuré en sortie) et en la recevant sur le port vertical. Le transfert diagonal est immédiat (la donnée est envoyée sur le port droit et reçue sur le port vertical).

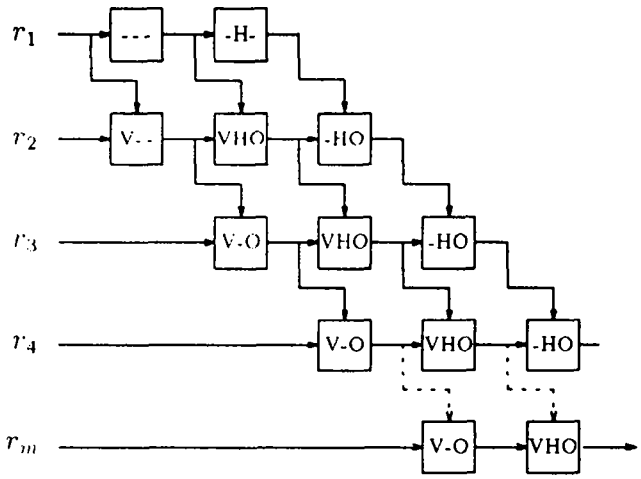

Figure 6 : réseau diagonal

Dans ce type de réseau, la connexion diagonale est nécessaire. En effet, un transfert diagonal ne peut pas être systématiquement remplacé par un transfert horizontal, suivi d'un transfert vertical. Un tel chemin n'existe pas pour les processeurs de la diagonale supérieure.

Pour réduire la quantité de données à envoyer au réseau, en particulier les valeurs d'initialisation des processeurs des bords, le mécanisme décrit à la section 3.2.3 est utilisé. Les processeurs sont de différents types, en fonction du type des valeurs qu'ils reçoivent (initiales ou calculées). Trois booléens V, H et O indiquent respectivement pour chaque processeur que les valeurs reçues verticalement, horizontalement et en oblique sont calculées. Le cycle systolique devient :

début { **cycle systolique diagonal** }

```
1.1  écrire(PD, Dt);        lire(PV, Dd);
1.2  Dt = D;
2    écrire(PG, D);         lire(PV, Dv);
3    écrire(PD, D);         lire(PG, Dh);
4    écrire(PD, r);         lire(PG, r);
5    Rv = (V ? Dv + K_o : Iv + K_o);
6    Rh = (H ? Dh + K_a : Ih + K_a);
7    D  = (Rv < Rh ? Rv : Rh);
8    Rd = MEM[r];
9    Rd = (O ? Dd + Rd : Id + Rd);
10   D  = (D < Rd ? D : Rd);
```

fin { **cycle systolique diagonal** }

Ce programme est identique au **cycle systolique bidimensionnel** à ceci près que la ligne 1 a été remplacée par les lignes 1.1 et 1.2. En effet, une connexion diagonale est maintenant immédiate. Un délai doit donc être introduit entre le transfert de la valeur D et son utilisation en tant que Dd (cf section 3.2.2). Cela est mis en œuvre par les instructions 1.1 et 1.2, la deuxième gérant le délai grâce à la variable Dt. Ce programme montre à nouveau que le jeu d'instructions d'API15C est bien adapté à la gestion de calculs dépendants des données. Ce programme contient essentiellement des opérations d'entrées/sorties et des affectations conditionnelles.

3.3 Modèle systolique pipeline

Dans le modèle systolique de base, un processeur est associé au calcul de chaque valeur $D(i,j)$ (un tableau de $(n \times m)$ processeurs est ainsi construit). Etant donné que le calcul d'un $D(i,j)$ nécessite plusieurs cycles de base, le modèle systolique pipeline consiste à répartir ce calcul sur plusieurs processeurs. Cette répartition est intéressante dans deux cas :

- lorsque le temps de calcul doit être réduit et qu'il est par conséquent nécessaire d'augmenter le nombre de processeurs de la machine,
- lorsque la taille du réseau est supérieure à la taille du problème. Il est alors intéressant d'utiliser la machine à pleine puissance.

Le calcul de la distance de Levenshtein peut être représenté par le graphe de la figure 7. Ce calcul nécessite trois additions, deux minimisations, un accès mémoire et deux constantes.

Dans l'optique d'un pipeline des calculs, ce graphe peut être divisé en deux sous-graphes A et B, comme le montre la figure 8. De légères modifications sont apportées au graphe original pour que les deux sous-graphes soient identiques :

- Un additionneur est ajouté dans le sous-graphe B. Il ne modifie pas le calcul, étant utilisé pour ajouter 0 à la valeur Min calculée par le sous-graphe A.

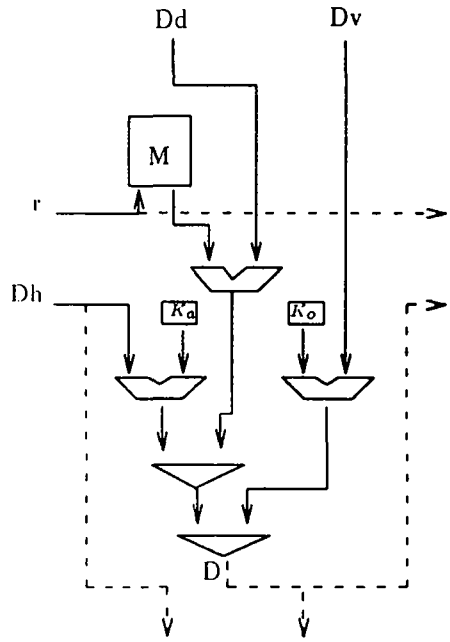

Figure 7 : graphe de calcul de la distance de Levenshtein

- Dans le sous-graphe B, le registre contenant K_o est remplacé par une mémoire. Cette mémoire est adressée par r et rend systématiquement la valeur K_o.
- Un multiplexeur, noté MUX, est ajouté aux deux sous-graphes, à cause de la nature systolique de l'application. En effet, la structure bidimensionnelle de la figure 4 montre qu'un D calculé est reçu sous trois noms différents, suivant la direction du transfert :
  - verticale : reçu sous le nom de Dv,
  - horizontale : reçu sous le nom de Dh,
  - diagonale : reçu horizontalement sous le nom de Dh au premier cycle systolique et verticalement sous le nom de Dd au deuxième cycle systolique.

  Le multiplexeur est nécessaire pour prendre en compte la différence entre les deux étages A et B du pipeline. En effet, tandis que l'étage B délivre D verticalement, l'étage A doit transférer verticalement la valeur Dh qu'il a précédemment reçue (pour émuler le transfert diagonal).

Ces deux sous-graphes ont été conçus de telle sorte qu'un même programme API15C permette de calculer la distance de Levenshtein en mode pipeline sur deux processeurs différents.

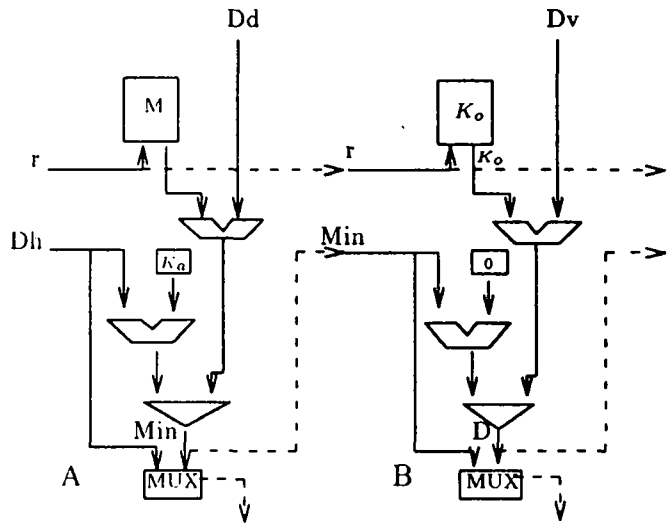

Figure 8 : transformation du graphe de calcul de la distance de Levenshtein

3.3.1 Réseau bidimensionnel

La mise en œuvre de la structure pipeline sur une architecture bidimensionnelle est simple. Il suffit en effet de remplacer un élément de calcul par deux processeurs. Un booléen P indique pour chaque processeur s'il exécute le premier étage du pipeline, comme le montre la figure 9.

Le cycle systolique devient :

```
début { cycle systolique 2-D pipeline }
                                          /* étage 1        étage 2      */
 1   écrire(PD, mux);   lire(PV, u);      /* u ≡ Dd         u ≡ Dv       */
 2   écrire(PD, min);   lire(PG, l);      /* l ≡ Dd         l ≡ min      */
 3   écrire(PD, x);     lire(PG, x);      /* x ≡ r          x ≡ r        */
 4   R1  = MEM[x];
 5   R1  = u + R1;
 6   R2  = l + K;                         /* K ≡ K_a        K ≡ 0        */
 7   min = (R1 < R2 ? R1 : R2);           /* min ≡ Min      min ≡ D      */
 8   mux = (P ? l : min);
fin { cycle systolique 2-D pipeline }
```

Dans ce programme, de nouvelles variables (u, l, x, K, min) apparaissent. La variable leur correspondant à chaque étage du pipeline est indiquée en commentaire. Dans un souci de simplicité, le problème des valeurs initiales des processeurs des bords n'est pas traité dans ce programme. Il peut être résolu de la même manière que pour le programme systolique bidimensionnel.

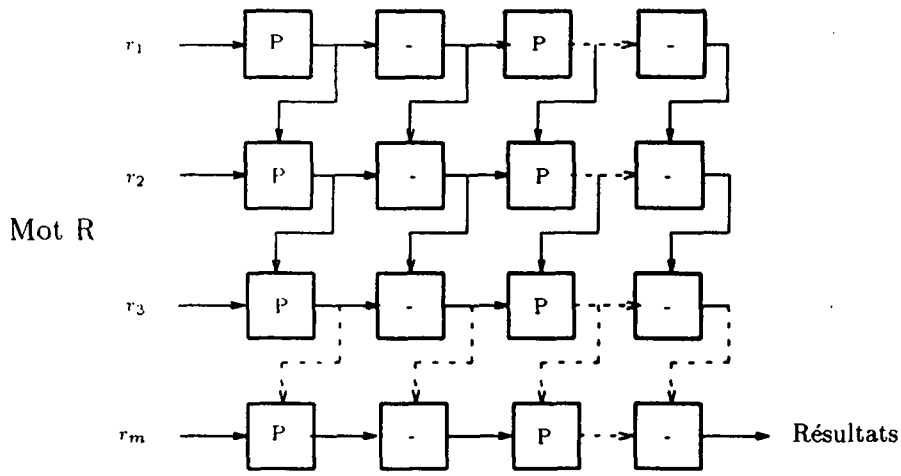

Figure 9 : structure bidimensionnelle pipeline

3.3.2 Réseau linéaire

Un réseau linéaire peut émuler le programme systolique bidimensionnel pipeline. En effet, lors d'un calcul d'une distance, une seule diagonale de la structure bidimensionnelle est active à un instant donné. Chaque processeur de la structure linéaire émule donc une colonne de processeurs de la structure bidimensionnelle. La figure 10 représente le réseau linéaire. Le booléen P indique l'étage du pipeline exécuté par le processeur. Le cycle systolique devient :

début { **cycle systolique linéaire pipeline** }

1 $u = mux$;
2 $écrire(PD, min)$; $lire(PG, l)$;
3 $écrire(PD, x)$; $lire(PG, x)$;
3.1 $écrire(PD, m)$; $lire(PG, m)$;
4 $R1 = MEM[x]$;
5 $R1 = (m \neq \perp \, ? \, u + R1 \, : \, Iu + R1)$;
6 $R2 = l + K$;
7 $min = (R1 < R2 \, ? \, R1 \, : \, R2)$;
8 $mux = (P \, ? \, l \, : \, min)$;

fin { **cycle systolique linéaire pipeline** }

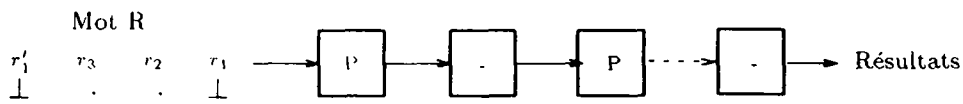

Figure 10 : structure linéaire pipeline

Ce programme est très voisin du programme systolique bidimensionnel pipeline. A la ligne 1, une affectation remplace un transfert vertical. A la ligne 3.1, un jalon est utilisé (comme pour le programme systolique linéaire).

3.3.3 Réseau diagonal

La mise en œuvre du modèle pipeline sur la structure diagonale est immédiate. L'idée de base consiste à remplacer chaque processeur du réseau diagonal original par deux processeurs, comme le montre la figure 11.

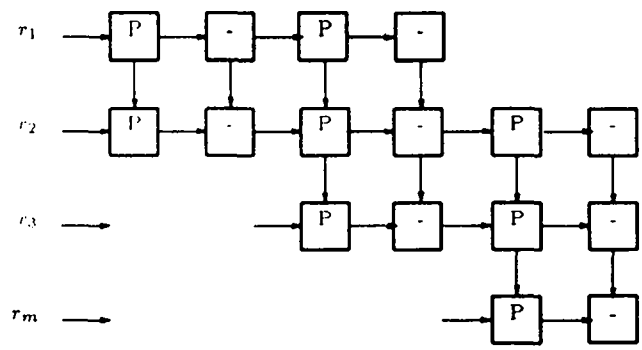

Figure 11 : structure diagonale pipeline

Une autre solution intéressante consiste à utiliser le réseau diagonal décrit à la section 3.2.5 pour mettre en œuvre le modèle pipeline. L'idée est de "tordre" le réseau de la figure 11 pour obtenir le réseau de la figure 12.

Il faut remarquer que, si la géométrie du réseau est modifiée, la topologie reste la même. La mise en œuvre de ce réseau sur la structure diagonale de la section 3.2.5 n'est pas immédiate. La direction de certains transferts varie en fonction du type du processeur (en diagonal pour les processeurs de type P et horizontalement pour les autres). Le mode de fonctionnement du réseau étant SIMD, ces transferts doivent donc être effectués dans les deux directions. Le cycle systolique devient :

début { cycle systolique diagonal pipeline }

1	$écrire(PG, mux)$;	$lire(PV, u)$;
2.1	$écrire(PD, min)$;	$lire(PG, l1)$;
2.2	$écrire(PD, min)$;	$lire(PV, l2)$;
2.3	$l = (P\,?\,l1 : l2)$;	
3.1	$écrire(PD, x)$;	$lire(PG, x1)$;
3.2	$écrire(PD, x)$;	$lire(PV, x2)$;
3.3	$x = (P\,?\,x1 : x2)$;	
4	$R1 = MEM[x]$;	
5	$R1 = u + R1$;	
6	$R2 = l + K$;	

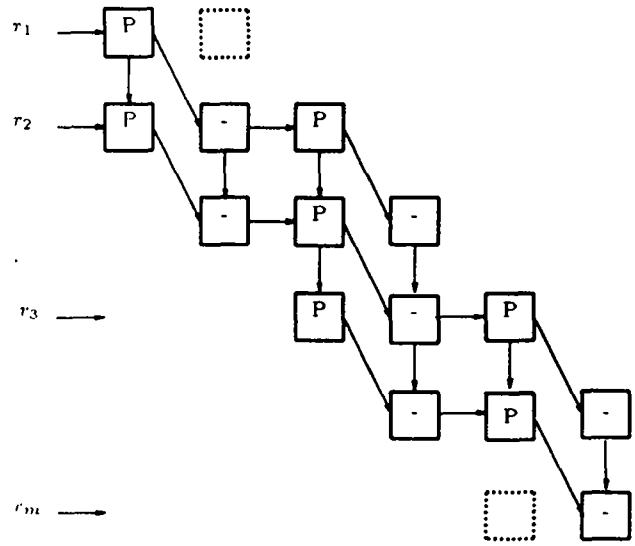

Figure 12 : structure diagonale pipeline transformée

7 $min = (R1 < R2\ ?\ R1\ :\ R2)$;
8 $mux = (P\ ?\ l\ :\ min)$;
fin { cycle systolique diagonal pipeline }

Ce programme ressemble au programme systolique bidimensionnel pipeline décrit précédemment. En raison du mode d'exécution SIMD, une donnée devant être transférée en diagonal ou horizontalement est transmise dans les deux directions (instructions 2.1 et 2.2 pour *min* et 3.1 et 3.2 pour x). Le booléen P, indiquant l'étage de pipeline exécuté par le processeur, est utilisé par les instructions 2.3 et 3.3 pour choisir respectivement les valeurs des variables l et x.

4 Mise en œuvre matérielle

Afin de valider l'architecture du circuit API15C et d'obtenir des mesures de vitesse précises, deux structures systoliques différentes ont été réalisées. La première est un réseau linéaire de 18 processeurs API15C qui peut être considéré comme une machine générale supportant de nombreux algorithmes [4]. La seconde est un réseau bidimensionnel tronqué de 28 processeurs API15C. Ce réseau est dédié à une application de reconnaissance automatique d'adresses postales.

Cette section présente d'abord comment est résolu le problème de la connexion avec le calculateur hôte, indépendamment de la topologie du réseau systolique. Ensuite, le module d'interface correspondant est décrit. Nous terminons en donnant quelques résultats sur les performances obtenues sur les algorithmes présentés dans la section précédente.

4.1 Présentation du module d'interface

Tout d'abord, lors de la conception du module d'interface, aucune restriction sur la topologie du réseau n'a été faite. Il doit être très versatile pour une utilisation avec des réseaux systoliques variés tels que les structures classiques (linéaires ou 2-D) ou bien des réseaux plus spécifiques pour des applications particulières. Ces spécifications impliquent la conception d'un module entièrement programmable afin de tester différentes implémentations d'un algorithme sur les processeurs API15C.

En second lieu, sachant qu'un réseau systolique nécessite une alimentation en données importante, une liaison directe avec le calculateur hôte est exclue afin de ne pas ralentir les calculs. Ainsi, les fonctions principales de l'interface sont d'alimenter le réseau de processeurs à la bonne cadence et de contrôler une structure systolique SIMD.

Ces propriétés conduisent à considérer 3 tâches principales allouées au module d'interface, appelé **Macs** :

1. générer des instructions pour les processeurs API15C,
2. gérer les transferts de données de/vers le réseau systolique,
3. communiquer avec le calculateur hôte.

Ces 3 fonctions sont toujours nécessaires quel que soit le réseau systolique considéré. Il apparaît que cette approche est très intéressante puisque ce module complexe est utilisable pour piloter de nombreuses structures systoliques. De plus, le temps nécessaire à la réalisation matérielle, ainsi que le test, d'un nouveau réseau dans un environnement réel est très court puisque seule la conception de la carte systolique est nécessaire. Généralement, en raison de la grande régularité inhérente aux structures systoliques, la conception d'une telle carte ne présente aucune difficulté et peut être réalisée rapidement.

4.2 Principe du module d'interface

En fait, on distingue deux processus dans l'exécution d'une application : le premier produit les instructions exécutables par les processeurs tandis que le second gère les échanges de données avec le réseau ou l'hôte. Ces processus ont à peu près la même structure de contrôle mais peuvent légèrement différer suivant les données à traiter. En fait, le premier processus exécute un programme indépendant des données alors que le second exécute un programme dépendant des données.

L'architecture de ce module de contrôle est directement issue de ce découpage en 2 processus : deux contrôleurs implémentent les deux processus. Le premier est chargé de fournir les instructions aux circuits API15C. C'est une machine très simple construite avec un séquenceur (AMD 2910) et une mémoire de micro-programmes. Un seul séquenceur est nécessaire puisque tous les processeurs exécutent la même instruction au même instant.

L'autre contrôleur représente une machine plus complexe et plus puissante. Il contient également un séquenceur et une mémoire de micro-programmes. En plus, il inclut une mémoire assez grande pour conserver les informations à envoyer ou reçues par le réseau. Cette mémoire est pilotée par un générateur d'adresses qui gère de façon efficace l'adressage.

Puisque les deux contrôleurs exécutent leur programme indépendamment, ils doivent se synchroniser en permanence afin que les échanges de données entre le réseau et la mémoire puissent s'effectuer convenablement. L'envoi d'une donnée sur le réseau correspond à une instruction d'entrée API15C, ce qui implique que les deux contrôleurs doivent se synchroniser sur cet événement. Un mécanisme matériel performant a été mis en œuvre pour gérer efficacement les synchronisations.

Ce module d'interface est connecté actuellement à un IBM PC/AT. Il est équipé d'un port d'instruction sur 16 bits délivrant une instruction toutes les 100 ns et d'un bus de données bidirectionnel sur 16 bits sur lequel plusieurs ports peuvent être connectés en fonction de la topologie du réseau. Une donnée peut être reçue ou émise toutes les 100 ns.

4.3 Tableaux systoliques réalisés avec API15C

La figure 13 montre l'organisation du réseau 2-D tronqué et ses connexions avec le module d'interface. Celui-ci gère 2 ports, un en sortie pour transmettre des données aux processeurs de la colonne de gauche et un en entrée pour récupérer les résultats du processeur inférieur droit. Quand une donnée est envoyée, les données précédentes sont décalées d'une position vers le bas. Ainsi, la fourniture de données pour le réseau nécessite n opérations de sorties si n est le nombre de lignes de processeurs.

Un réseau linéaire de 18 processeurs API15C a également été réalisé. Le réseau est connecté au module Macs au moyen de 3 ports : 2 ports bidirectionnels pour envoyer ou recevoir des données de/vers les processeurs extrêmes et un port de sortie connecté au port d'entrée PV de chaque processeur API15C, permettant ainsi de diffuser la même donnée (d'une manière non systolique) à tous les processeurs.

4.4 Performances

La table 1 montre les performances des réseaux systoliques décrits dans la section précédente en terme de nombre d'instructions exécutées par cycle systolique. Ces résultats ont été obtenus par macro-expansion des programmes correspondants. Ceci signifie que les valeurs obtenues donnent une borne supérieure puisqu'aucune optimisation n'a été appliquée. La table montre que le jeu d'instructions du circuit API15C est bien adapté pour l'exécution du modèle de base sur les 3 structures puisque le nombre d'instructions est similaire. Les résultats du modèle pipeline suggèrent 2 remarques. D'une part, que le pipeline à 2 étages ne permet pas une réduction de 50% du temps de calcul mais seulement de 22% et de 31.5% respectivement pour les cas 2-D et linéaire, ce qui est un résultat intéressant. D'autre part, que le pipeline n'apporte aucune amélioration dans le cas diagonal. En fait, ceci est dû surtout au surcoût associé aux transferts de données complexes entre les processeurs plutôt qu'à

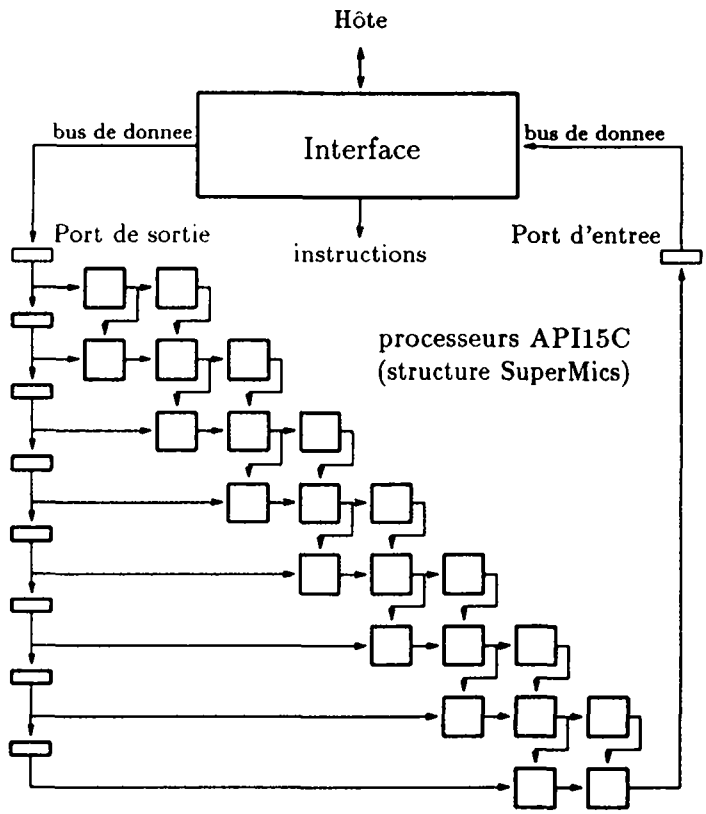

Figure 13 : réseau systolique 2-D tronqué

la structure interne du processeur API15C.

Réalisation	2-D	linéaire	diagonal
modèle de base	36	38	36
pipeline	28	26	36

Table 1 : nombre d'instructions API15C exécutées pour différentes implémentations

5 Conclusion

Le processeur API15C est un bloc de base pour réaliser différentes structures systoliques. Il est conçu exclusivement pour travailler dans un mode d'exécution SIMD (ainsi, il ne contient pas de séquenceur). Afin de supporter efficacement ce mode de fonctionnement, le jeu d'instructions du processeur inclut des instructions de contrôle spéciales. Trois ports d'entrées/sorties parallèles sont disponibles pour réaliser différents schémas d'interconnexion.

Le circuit API15C a été conçu dans une technologie CMOS 2 μm. Il contient 45.000 transistors sur une surface de silicium de 6 mm par 6,2 mm. Le circuit a fonctionné correctement dès la première fabrication. Il exécute une instruction par phase d'horloge de 100 ns, soit une puissance de 10 MIPS.

Afin de valider ce processeur élémentaire en tant que bloc de base pour structures systoliques, une interface programmable et 2 machines mono-carte ont été développées. La première carte contient un réseau linéaire de 18 processeurs capable d'exécuter une large classe d'algorithmes. La seconde est constituée par une structure bidimensionnelle de 28 processeurs conçue pour une application spécifique de comparaison de chaînes.

Nous avons montré que le jeu d'instructions est particulièrement bien adapté au mode d'exécution SIMD. Par exemple, dans le réseau bidimensionnel, les problèmes complexes liés aux processeurs des bords, sont résolus d'une façon élégante.

Ces premières investigations démontrent la puissance d'API15C. Cependant, de nouvelles améliorations peuvent être apportées :

- comme indiqué précédemment, ce processeur comporte 3 ports d'entrées/sorties parallèles (2 ports bidirectionnels et 1 port d'entrée) qui permettent une grande variété de topologies. Une extension possible consiste à implémenter 3 (ou plus) ports bidirectionnels afin de supporter des schémas d'interconnexions plus complexes.

- la vitesse de communication des entrées/sorties est d'une importance capitale dans les architectures systoliques lorsqu'un parallélisme à grain fin est nécessaire. API15C est capable de réaliser quelques opérations d'E/S de façon transparente mais dans certains cas, ces opérations utilisent des ressources internes (bus) qui pénalisent le temps de cycle systolique. L'ajout d'un mécanisme d'E/S spécifique devrait apporter de meilleures performances.

- il est clair que des instructions conditionnelles efficaces sont importantes dans un environnement SIMD. Le jeu d'instructions d'API15C inclut seulement des instructions conditionnelles de base. De nouvelles investigations doivent être entreprises sur ce genre d'instruction.

Pour certaines applications, comme la convolution par exemple, une structure constituée de processeurs API15C est plus lente qu'une solution entièrement intégrée. Ceci est dû principalement à la nature programmable du processeur API15C. En contre-partie, API15C exécute une large gamme d'applications. Nous pensons qu'une approche utilisant des processeurs API15C représente une bonne solution de compromis qui peut être suivie par une solution intégrée. Des applications complexes, telles que la correction de chaînes, décrite dans ce papier, peuvent d'abord être émulées sur une structure composée de circuits API15C. Puis, après une validation complète, une solution mono-circuit peut être envisagée par une simple duplication soit du dessin du coeur d'API15C, soit, si nécessaire, de modules d'API15C adaptés. Une telle solution est réaliste dès à présent puisque la technologie VLSI permet d'intégrer plus d'un million de transistors par circuit, à comparer aux 45.000 transistors du circuit API15C.

Bibliographie

[1] C. R. Baugh, B. A. Wooley, "A Two Complement Parallel Array Multiplication Algorithm", *IEEE Transactions on Computer,* vol. C-22 *N°* 12, pp. 225-261, 1973.

[2] S. Borkar, R. Cohn, G. Cox, T. Gross, H.T. Kung, M. Lam, M. Moore, C. Peterson, J. Pieper, J. Rankin, P.S. Tseng, J. Sutton, J. Urbanski, and J. Webb. "iWarp: An Integrated Solution to High-Speed Parallel Computing", *Proceedings of Supercomputing '88,* Nov 1988.

[3] P. Frison, D. Lavenier, "A VLSI Systolic Machine for String Correction", *ICS 88: Third International Conference on Supercomputing,* proceedings vol. 3, pp. 19-24 Mai 1988.

[4] P. Frison, D. Lavenier, H. Leverge, P. Quinton, "MicMacs: A VLSI Programmable Systolic Architecture," *International Conference on Systolic Arrays,* Juin 89.

[5] P. Frison, E. Gautrin, D. Lavenier, J.L. Scharbarg, "API15C, a Programmable Chip for Systolic Architecture," *IFIP Workshop on Parallel Architectures on Silicon*, Grenoble, Nov 1989.

[6] H.T. Kung, "Programmable systolic chip," *Proc. NATO ASI on Microarchitecture of VLSI computers*, Sogesta, Italie, Juillet, 1984.

[7] M. Annaratone & al, "Warp Architecture and Implementation," *Proc. IEEE ISCA,* Tokyo, pp. 346-356, 1986.

[8] S.Y. Kung, "VLSI Array Processors," *Prentice Hall,* 1988.

[9] D. Lavenier, J. L. Scharbarg, P. Frison, "Architecture Systolique pour la Correction Automatique de Libellé d'Adresse," *rapport de recherche INRIA*,n° 995, Mars 1989.

[10] D. Lavenier, " MicMacs : un réseau systolique linéaire programmable pour le traitement de chaînes de caractères," *Thèse de l'université de Rennes 1,* Juin 1989.

[11] P. Quinton, Y. Robert, "Algorithmes et Architectures Systoliques," *ed. Masson,* 1989.

[12] R. A. Wagner, M. J. Fisher, "The string to string correction problem," *J. ACM,* vol. 21, pp. 168-173, 1976.

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 518 **MULTISCALE SYSTEM THEORY**
Albert BENVENISTE, Ramine NIKOUKHAH, Alan S. WILLSKY
Février 1990, 30 Pages.

PI 519 **PANDORE : A SYSTEM TO MANAGE DATA DISTRIBUTION**
Françoise ANDRE, Jean-Louis PAZAT, Henry THOMAS
Février 1990, 14 Pages.

PI 520 **SCHEDULING AFFINE PARAMETERIZED RECURRENCES BY MEANS OF VARIABLE DEPENDENT TIMING FUNCTIONS**
Christophe MAURAS, Patrice QUINTON,
Sanjay RAJOPADHYE, Yannick SAOUTER
Février 1990, 14 Pages.

PI 521 **COMPUTABILITY OF RECURRENCE EQUATIONS**
Yannick SAOUTER, Patrice QUINTON
Février 1990, 28 Pages.

PI 522 **PROGRAMMING BY MULTISET TRANSFORMATION**
Jean-Pierre BANATRE, Daniel LE METAYER
Mars 1990, 26 Pages.

PI 523 **GOTHIC MEMORY MANAGEMENT : A MULTIPROCESSOR SHARED SINGLE LEVEL STORE**
Béatrice MICHEL
Mars 1990, 20 Pages.

PI 524 **ORDER NOTIONS AND ATOMIC MULTICAST IN DISTRIBUTED SYSTEMS : A SHORT SURVEY**
Michel RAYNAL
Mars 1990, 18 Pages.

PI 525 **MULTI-SCALE AUTOREGRESSIVE PROCESSES**
Michèle BASSEVILLE, Albert BENVENISTE, Alan S. WILLSKY
Mars 1990, 136 Pages.

PI 526 **TRANSFORMATIONS PYRAMIDALES D'IMAGES NUMERIQUES**
Nadia BAAZIZ, Claude LABIT
Mars 1990, 102 Pages.

PI 527 **LE LANGAGE SIGNAL : UN EXEMPLE EN SEGMENTATION AUTOMATIQUE DE LA PAROLE CONTINUE**
Claude LE MAIRE
Mars 1990, 112 Pages.

PI 528 **CONDITIONAL REWRITE RULES AS AN ALGEBRAIC SEMANTICS OF PROCESSES**
Eric BADOUEL
Mars 1990, 46 Pages.

PI 529 **RESEAUX SYSTOLIQUES SPECIFIQUES A BASE DU PROCESSEUR API15C**
Patrice FRISON, Eric GAUTRIN, Dominique LAVENIER,
Jean-Luc SCHARBARG
Mars 1990, 26 Pages.

ISSN 0249 - 6399