Une architecture parallele SIMD a instruction longue

Belkacem Zerrouk, A. Derieux

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

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° 1413

Programme 1
Architectures parallèle, Base de données,
Réseaux et Systèmes distribués

UNE ARCHITECTURE PARALLELE SIMD A INSTRUCTION LONGUE

Belkacem ZERROUK
Anne DERIEUX

Avril 1991

RR_1413

UNE ARCHITECTURE PARALLELE SIMD A INSTRUCTION LONGUE

SIMD MULTIPROCESSOR ARCHITECTURE WITH A LONG INSTRUCTION WORD

Belkacem ZERROUK*
Anne DERIEUX**

*Projet ARCHI, INRIA Roquencourt
**Laboratoire CAO&VLSI_MASI, Paris VI

Résumé

Les progrès de l'éléctronique favorisent de plus en plus la réalisation de machines massivement parallèles à un coût réduit. Certains projets présentent une évolution dans les prototypes conçus, en fonction des nouvelles données technologiques : densité d'intégration, encapsulement, circuits imprimés... Ces progrès ont une influence directe sur les architectures du point de vue de leur faisabilité. C'est ainsi que Blitzen, avec 128 processeurs élémentaires par puce (mémoire inclue), donne un nouvel élan au MPP, et que la Connection Machine CM2, une nouvelle version de la CM1, est dotée d'opérateurs de calcul en virgule flottante améliorant ainsi les performances de la CM1.

Le but de notre projet est de concevoir les éléments de base pour la réalisation une architecture parallèle SMAL: le processeur élémentaire et les éléments de contrôle. Dans ce rapport nous donnons une vue globale de l'architecture et de la mise en oeuvre de la machine SMAL.

Abstract

The advent of new electronic technologies encourages the design of massively parallel architectures. The important benefit of the current development in electronic area, like high density of integration and new packaging and printed circuit technics, is to permit increasing evolution of some existing prototypes with a good cost-effectiveness. As examples, the Blitzen, with 128 on chip PEs (memory included) is probably a good successor for the MPP. And, the CM2, an extended version of the CM1, exhibits more performances using floating point operators.

We are working on the design of the basic elements of a parallel architecture called SMAL (from SIMD Multiprocessor Architecture with a Long instruction word). This report give a global architectural and software view.

1. INTRODUCTION

Le projet SMAL (SIMD Multiprocessor Architecture with a Long instruction word) a déjà été introduit dans une publication antérieure [1]. Notre but principal est de pouvoir réaliser des éléments de base permettant le développement d'architectures massivements parallèles à granularité fine de type SIMD (Single Instruction Stream Single Data Stream). Nous rejoignons ainsi le courant des nouvelles architectures qui s'interesse aux problèmes posés par le parallélisme massif. Les principaux projets de cette voie sont: MPP[2], Blitzen[3], Connection Machine[4], YUPPI[5], le GAPP[6], CLIP[7]...

Une architecture parallèle à grain fin présente généralement un bon rapport coût/performances. En effet, la conception est relativement peu coûteuse et étant donné le grand nombre de processeurs élémentaires, les performances globales sont trés significativement améliorées par rapport à un processeur séquentiel.

Certains projets présentent une évolution dans les prototypes conçus, en fonction des nouvelles données technologiques: densité d'intégration, encapsulement ou packaging, circuits imprimés... Ces progrès ont une influence directe sur les architectures du point de vue de leur faisabilité. C'est ainsi que Blitzen, avec 128 processeurs élémentaires par puce (mémoire inclue), donne un nouvel élan au MPP, et que la Connection Machine CM2, une nouvelle version de la CM1, est dotée d'opérateurs de calcul en virgule flottante améliorant ainsi les performances de la CM1.

Pour des raisons d'intégration, de connectique et d'extensibilité, les architectures en grille (Mesh connected array of processors) sont préférées aux architectures à diamétre logarithmique. De plus, une large classe d'algorithmes nécessite des connexions en grilles. Un argument suplémentaire pour ce genre de choix est l'avènement de la WSI (Wafer Scale Integration) qui favorise les systèmes tolérants aux défaillances tels que ELSA[8].

SMAL fait partie de la classe de machines à réseau en grille. Son architecture est composée essentiellement de processeurs élémentaires SMAL_X31 et d'un contrôleur µSMAL.

Les paragraphes suivants seront consacrés respectivement à l'architecture de SMAL (section 2), à certains exemples d'opérations (section 3), aux communications (section 4), à des exemples d'applications (section 5) et enfin nous terminons par l'état d'avancement du projet (section 6).

2. ARCHITECTURE

Dans cette partie nous décrivons une architecture régulière dont l'extension peut être aussi bien topologique que fonctionnelle. L'extensibilité topologique implique l'ajout de processeurs élémentaires; ce qui est aisé dans le cas d'architecture matricielle. L'extensibilité fonctionnelle est justifiée par le fait que plusieurs modules "réseaux" peuvent être cascadés pour former non pas une simple machine SIMD mais une machine MSIMD.

2.1. LE PROCESSEUR ELEMENTAIRE

Le processeur élémentaire, SMAL_X31, déja décrit dans [1] est repris dans la figure 1 . Il est constitué principalement de deux chemins de données symétriques qui forment la partie opérative supérieure (POS), d'une partie opérative inférieure, et d'une partie contrôle incluant une unité de décision. Des précisions peuvent être trouvées dans [1a] ou [1b].

La partie opérative supérieure (POS) est dédiée au traitement, aux entrées/sorties et aux communications interprocesseurs. Elle est formée essentiellement de deux ALUs, deux files (FIFO) et de deux registres d'entrées/sorties IOPR et IOPL.

La partie opérative inférieure (POI), formée par un banc de 16 registres et d'une unité logique, offre essentiellement une paramétrisation du processeur élémentaire: topologie, taille du grain de calcul, rebouclage des files. Elle permet une évaluation de bits de conditions/masques ainsi que du OU_global et offre une ressource mémoire locale incorporée au circuit.

Nous avons généralisé l'utilisation des conditions/masques [2,3] à toutes les opérations: opérations arithmétiques et logiques, la sauvegarde en mémoire et communications interprocesseurs.

La mémoire locale d'un PE étant externe, l'inhibition de l'écriture se fait par une simple réécriture de l'ancienne valeur. Cela nous a permis l'intégration d'un simple dispositif avec le PE et de gagner ainsi les signaux de validité d'écriture mémoire (Write Enable).

Le processeur élémentaire peut exécuter des instructions de type SIMD [1a]. Parmi ces opérations on trouve la conditionnelle "***if-then-else***" qui est fondamentale pour assurer l'autonomie des processeurs élémentaires.

2.2. LE RESEAU SMAL

La réseau SMAL, à base de processeurs élémentaires SMAL_X31 que nous proposons, est donné par la figure 2a. L'architecture est composée essentiellement de 6 modules:

-Un réseau matriciel de processeurs élémentaires.
-Une unité de contrôle.
-Une mémoire programme.
-Une interface avec la machine hôte.
-Une FIFO d'entrées de données.
-Une FIFO de sorties de données.

Le réseau:

Les différents plans du réseau SMAL sont: Les plans mémoire, les plans registres POI, les plans FIFOs, le plan de registres d'entrées/sorties (IOPR et IOPL), les plans d'entrées UALs (AR,BR,AL et BL) et les plans de retenues (CR et CL).

Le réseau de processeurs est une matrice de (N+P)xN processeurs élémentaires (PEs) SMAL_X31 connectés de sorte que chaque PE puisse communiquer avec ses 4 proches voisins Nord, Sud, Est et Ouest. La liaison d'un PE vers ses voisins Nord et Est est assurée par le port de droite (Right Port) alors que la liaison Sud et Ouest est assurée par le port de gauche (Left Port).

Chaque processeur est muni de deux mémoires locales externes (R et L) à 2 ports chacune : 1 port de lecture et 1 port de lecture/écriture.

Le réseau matriciel (N+P)xN est divisé en deux sous réseaux de processeurs élémentaires : une matrice de traitement de dimensions NxN et une matrice de formatage de données de dimensions PxN.

La matrice de traitement peut être organisée grâce à un paramétrage local à chaque processeur, en différentes formes topologiques matricielles où l'on retrouve typiquement la forme maillée (MESH) et la forme semi-torique (les bords Nord et Sud liés). Ce paramétrage interne permet aussi de déconnecter un processeur élémentaire de ses voisins Ouest et Sud.

La matrice de formatage de données sert à la sérialisation et la désérialisation des données, au transfert depuis ou vers des éléments externes (FIFOs) et au stockage de données. De plus cette même unité assure le routage de messages/données.

La matrice de traitement et la matrice de formatage peuvent former un réseau de (N+P)xN PEs. Les liens existants entre ces deux parties sont donnés par la figure 2b.

Une topologie torique complète (càd bords Sud et Nord liés et bords Est et Ouest liés) peut être établie dans la matrice de traitement. Son implantation est possible moyennant un "court circuit" (By_pass) des processeurs de l'unité de formatage. Ce "by_pass" permet de lier directement les entrées/sorties Est et Ouest de l'unité de formatage.

L'unité de contrôle:

L'unité de contrôle comprend deux types de circuits:

-Un contrôleur µSMAL.
-Deux masques d'instructions.

Le contrôleur µSMAL est un véritable processeur de type LOAD_IMMEDIAT/NO_STORE [16] (voir figure 3). Nous le définissons ainsi du fait qu'il ne fait que lire la mémoire programme. Des registres internes permettent le stockage de données qui généralement sont des adresses de plans mémoire. Des opérations arithmétiques telle que l'incrémentation/décrémentation peuvent être effectuées localement.

L'existence d'éléments comme un temporisateur (external loop register), un compteur de répétition d'instruction, un registre de contrôle externe et d'une pile de programme, font du processeur µSMAL un élément bien adapté au contrôle d'un réseau SIMD.

Le contrôleur permet d'effectuer plusieurs fonctions dont la génération d'adresses mémoire, le contrôle des entrées/sorties depuis ou vers l'interface, le séquencement des instructions pour le réseau et offre une logique de dialogue avec la machine hôte.

Quatre adresses de 8 bits peuvent être calculées et/ou envoyées simultanément au réseau de processeurs. Chacune correspond à un port mémoire donné. Ayant prévu des mémoires locales de 1024 mots, et pour des raisons de conception telles que la normalisation du chemin de données interne de µSMAL et le gain de quelques broches sur le circuit, nous avons opté pour une décomposition des plans mémoire en quatre groupes (ou pages) de 256 plans.

Une instruction PE a un format de 32 bits, mais le circuit lui même ne dispose que de 16 entrées instructions: il nous faut donc prévoir un mécanisme de sérialisation. Le réseau des processeurs se décompose fonctionnellement en deux sous réseaux: le réseau de formatage et le réseau de traitement. Un mécanisme de sérialisation est associé à chaque sous réseau ce mécanisme est commandé par une logique de masquage permettant d'inhiber l'exécution d'une instruction envoyée par le contrôleur. Ceci nous permet l'instruction courante soit sur sur un des réseaux, soit simultanément sur les deux réseaux.

La machine hôte peut initialiser le réseau, charger des données dans la FIFO d'entrée, lire des résultats depuis la FIFO de sortie, et avec des signaux d'interruption et d'état, établir un protocole de dialogue avec le contrôleur. De plus, la machine hôte peut demander à tout moment une suspension au contrôleur afin d'acceder à la mémoire programme.

La mémoire programme peut avoir une taille maximum de 64 Kmots de 48bits. Elle est accessible en lecture par µSMAL et en écriture par la machine hôte.

Les modules SMAL sont facilement cascadables avec l'utilisation de FIFO pour les entrées/sorties. La figure 4 montre un réseau linéaire unidirectionnel de modules SMAL. Une cascade bidirectionnelle est aussi réalisée avec un ajout d'un minimum de composant (FIFOs). Avec cette approche nous pouvons réaliser un module unique de taille acceptable, qui forme la brique de base d'un système évoluant vers une machine massivement parallèle avec la seule adjonction de modules identiques. Le système ainsi obtenu serait une machine de type MSIMD du fait que les modules peuvent fonctionner de manière indépendante et totalement asynchrone.

2.3. L'INSTRUCTION SMAL

L'instruction SMAL a une taille de 48 bits et peut avoir un des trois formats de la figure 5. Les formats 1 et 2 concernent µSMAL et englobent les instructions de contrôle, de paramétrage, de rupture de séquence et de communication avec la machine hôte. Dans le troisième format nous retrouvons, concaténées, une instruction µSMAL et une instruction SMAL_X31. Le résultat de l'opération µSMAL, qui est un calcul d'adresses, est exploité via un mécanisme pipeline pour l'exécution effective de l'instruction PE dans le réseau. Le pipeline d'exécution est donné par la figure 6 avec les différents étages qui sont:

-La recherche de l'instruction.
-Décodage et exécution de l'opération µSMAL.
-Exécution de l'instruction réseau.

Sur la même figure 6 nous montrons aussi la décomposition en deux champs de 16 bits chacun de l'instruction PE.

3. LES OPERATIONS ARITHMETIQUES ET LOGIQUES

Nous pouvons distinguer globalement deux types d'opérations pouvant être effectuées par le processeur élémentaire : des opérations mémoire et des opérations mémoire-file. La taille des opérandes peut être paramètrée et cela moyennant la cascade des ALUs du processeur élémentaire.

La présence de deux ALUs dans le processeur élémentaire permet, de façon simple, soit de réaliser simultanément deux opérations soit d'exécuter une opération unique (pouvant être) accélérée par un facteur deux. Ces deux modes opératoires nécessitent à la fois une vectorisation du code machine et une bonne répartition des opérandes en mémoire.

Une bonne vectorisation consiste à utiliser une séquence contigüe de code machine, c'est à dire sans rupture de séquence, pour réaliser l'opération. En plus de la vectorisation, l'accélération par chainage des ALUs demande une optimisation de la gestion mémoire, à savoir la répartition des champs de bits d'un opérande sur les plans mémoires.

Nous présentons ci-dessous quelques exemples: l'addition de deux nombres non signés, l'addition de trois opérandes et des opérations de transferts mémoires (copies).

3.1. L'ADDITION

Pour éclaircir le principe de la vectorisation et de l'allocation mémoire optimisée nous donnons des exemples d'additions de nombres non signés:

```
additionR;
ren R00   AR; adresse opérande A
ren R30   BR; adresse opérande B
def N     30;
def alu_chain 0;
```

```
        begin add;
add:    {##zero(alu_chain)};
        mover AR BR {mm # na wm };
        nop  AR BR {mm # nxor wm}; addition des bits LSB.
        loop N;
        par inc nop nop inc {mm # add wm};
        par inc nop nop inc {mm # c wm}; le 33eme bit.
        return;
        end;
```

Ce sous programme permet de réaliser une addition de deux opérandes de 32 bits chacun, et délivre un résultat sur 33 bits. L'opération est de type AR<-AR+BR. Les opérandes dans cette exemple sont rangées dans la mémoire locale R. Nous supposons, que les adresses sont passées comme paramètres dans les registres R30 (pour BR) et R10 (AR). De plus; le registre de chaînage de la POI doit être initialisé à 0 avant l'opération.

Le langage utilisé si dessus est l'assembleur SMALASS[15]. Les mnémoniques entre accolades constituent le langage d'assemblage du processeur élémentaire SMALEL[15]. A gauche de l'accolade ouvrante, on retrouve les mnémoniques propres au controleur µSMAL. Nous remarquons la présence d'opérations paralléles au niveau du contrôleur.

L'équivalent algorithmique de ce sous programme est comme suit:

```
AdditionR(ar[],br[],N)
debut
% les opérations utilisées sont:
        le Ou-exclusif ^
        le Et logique  .
        le Ou logique  +
%
entier i;
booleen ar[0:N], br[0:N], rd;
% initialisation de la retenue %
rd=0;
pour i=0 à N-1
faire_parallèle
ar[i]=ar[i] ^br[i] ^rd; rd= ar[i].br[i]+rd.(ar[i]^br[i]);
fin_faire
ar[N]= rd;
fin
```

Moyennant une petite modification, nous pouvons transformer ce sous programme pour réaliser deux opérations d'addition:

```
        additionLR;
        ren R00   AR; adresse opérande AR
        ren R30   BR; adresse opérande BR
        ren R10   BL; adresse opérande AL
        ren R20   AL; adresse opérande BL
        def N     30;
        def alu_chain 0
        begin add;
add:    {##zero(alu_chain)};
        mover AR BR BL AL {mm na na wm };
        nop  {mm nxor nxor wm};
        loop N;
        par inc inc inc inc {mm add add wm};
        par inc inc inc inc {mm c c wm};
        return;
        end;
```

La différence par rapport au premier sous programme est qu'une deuxième opération d'addition s'effectue en parallèle sur deux autres opérandes de la mémoire locale L.

```
AdditionLR(ar[],br[],al[],bl[],N)
debut
entier i;
booleen ar[0:N], br[0:N],al[0:N],bl[0:N], rd,rg;
% initialisation des retenues %
rd=0;
rg=0;
pour i=0 à N-1
faire_parallèle
        ar[i]=ar[i] ^br[i] ^rd; rd= ar[i].br[i]+rd.(ar[i]^br[i]);
        al[i]=al[i] ^bl[i] ^rg; rg= al[i].bl[i]+rg.(al[i]^bl[i]);
fin_faire
ar[N]= rd;
al[N]= rg;
fin
```

Maintenant si l'on considère que chaque opérande est réparti en champs de 16 bits, selon la parité de la position des bits, sur les deux mémoires locales alors, avec la cascade des opérateurs arithmétiques et logiques, l'addition se réécrira comme suit :

```
        addition;
        ren R00   RAM; adresse opérande A bits pairs
        ren R30   RBM; adresse opérande B bits pairs
        ren R10   LAL; adresse opérande A bits impairs
        ren R20   LBL; adresse opérande B bits impairs
        def N     14;
        def alu_chain 0;
        begin add;
add:    {##one(alu_chain)}; chainage des ALUs
        mover RBL RBM LAL RAM {mm na na wm };
        nop  {mm nxor suba wm};
        loop N;
        par inc inc inc inc {mm add add wm};
        par inc inc inc inc {mm c # wm};
        return;
        end;
```

Il est important de remarquer que les trois types d'additions si dessus s'exécutent en un temps identique.

Un des exemples de vectorisation possibles, dans le cas du processeur élémentaire SMALX31, est l'addition de trois opérandes. En effet, nous pouvons répartir 3 opérandes comme suit:

Soient A,B et C les trois variables, une solution de répartition est telle que A doit être en mémoire R (droite) et B et C en mémoire L (gauche). L'opérande A doit être adressé par les lignes correspondant au port AR ainsi que BR alors que B est adressé par les ligne BL et C par AL.

```
        addition3;
        ren R00   AR; adresse l'opérande A
        ren R30   BR; adresse l'opérande A
        ren R10   BL; adresse l'opérande B
        ren R20   AL; adresse l'opérande C
        def N     29;
        def alu_chain 0
        begin add;
add:    {##zero(alu_chain)};
        mover AR BL AL BL {xm na na wm };
        nop  {xm # nxor wm};
        nop nop nop inc inc {xm nxor add wm}
```

```
loop N;
par inc inc inc inc {xm add add wm};
par inc inc inc inc {xm add c wm};
par inc inc nop nop {xm add  # wm};
par inc inc nop nop {xm c  # wm};          le 34 ème bit
return;
end;
```

L'opération est de la forme A<-A+B, C<-A+C. Néanmoins, la seconde opération d'addition est retardée d'une opération sérielle par rapport à la première. Cela revient à pipeliner deux opérations sérielles comme on peut le constater dans le programme. La soustraction peut être obtenue de la même manière moyennant des modifications minimes. L'algorithme correspondant est le suivant :

```
Addition3(ar[],br[],al[],bl[],N)
debut
entier i;
booleen ar[0:N], br[0:N],al[0:N],bl[0:N], rd,rg;
% initialisation des retenues %
rd=0;
rg=0;
faire_parallèle
ar[0]=ar[0] ^bl[0] ^rd; rd= ar[0].bl[0]+rd.(ar[0]^bl[0]);
fin_faire
pour i=1à N-1
faire_parallèle
        ar[i]=ar[i] ^bl[i] ^rd; rd= ar[i].bl[i]+rd.(ar[i]^bl[i]);
        al[i-1]=al[i-1] ^ar[i-1] ^rg; rg= al[i-1].ar[i-1]+rg.(al[i-1]^ar[i-1]);
fin_faire
faire_parallèle
al[N-1]=al[N-1] ^ar[N-1] ^rg; rg= al[N-1].ar[N-1]+rg.(al[N-1]^ar[N-1]);
ar[N]= rd;
fin_faire
al[N]=al[N] ^ar[N] ^rg; rg= al[N].ar[N]+rg.(al[N]^ar[N]);
al[N]= rg;
fin
```

Remarque: Dans les exemples ci dessus nous supposons rajoutés des bits de fort poids supplémentaires nuls.

3.2. LES COPIES MEMOIRE_MEMOIRE

Les copies mémoires sont de trois types : des copies locales à un plan mémoire donné, des copies entre plans mémoires ou encore des permutations. On peut dégager deux cas de copies selon la page source et la page destination : dans le cas de pages identiques un transfert direct mémoire-mémoire peut être effectué alors que dans le cas contraire un passage par les Fifos est obligatoire. Les routines de copies, pour des pages de numéros differents, ont un temps d'exécution double par rapport à celle qui s'effectuent sur des pages ayant le même numéro. Cela découle de l'architecture du prototype SMAL. En effet, rappelons que les adresses générées par le controleur tiennent sur 8 bits auxquels on rajoute un numéro de page sur 2 bits de manière à utiliser des mémoires locales d'une taille de 1024 mots.

Ayant deux Fifos sur 16 bits, le transfert d'une zone à une autre d'une donnée de taille supérieure nécessite un découpage en mots de 16 (ou 32 bits) bits si nécessaire et en particulier au cas où le numéro de page source est différent du numéro de page destination.

4. LES COMMUNICATIONS

Le processus d'échanges de données qui assure la communication entre processeurs élémentaires est implanté d'une manière différente par rapport aux architectures existantes. En effet, nous rejoignons l'idée qu'on retrouve dans SLiM [9] dans le sens ou le processeur élémentaire offre la possibilité de propager des données selon des chemins de calcul adaptés à certains algorithmes. Le calcul de convolution ci dessous en est un exemple typique. Dans ce cas, les files sont trés utiles à la transmission série d'un résultat entre processeurs voisins effectuée en parallèle avec les opérations de calcul.

Les sources d'échanges entre processeurs voisins peuvent être soit les mémoires locales soit les Fifos. Les destinations sont les Fifos. La possibilité de construire des topologies à mémoires partagées entre processeurs distants n'est pas à écarter.

Dans une liaison maillée, un processeur élémentaire peut recevoir deux données de ses deux voisins (Est-Ouest ou Nord-Sud selon la direction) de manière simultanée. Ceci, permet d'améliorer d'un facteur deux la communication avec le voisinage en comparaison avec une grille NEWS (terme désignant une liaison North East West South) habituelle. D'autres solutions existent cependant, comme la liaison directe avec les 8 voisins ou encore une solution en X proposée dans le projet BLITZEN [3] (voir figure 7). En ce qui nous concerne, nous avons préféré améliorer les performances de calcul, avec le dédoublement de l'UAL, à une solution utilisant la connectique.

Dans l'exemple ci dessus nous reprenons l'addition de deux opérandes avec la seule modification du champ destination:

```
        add_sendR;
        ren R00   AR; adresse opérande A
        ren R30   BR; adresse opérande B
        def N     7;
        def alu_chain 0;
        begin add;
add:    {##zero(alu_chain)};
        mover AR BR {mm # na wm };
        nop  AR BR {mm # nxor wp}; addition des bits LSB.
        loop N;
        par inc nop nop inc {mm # add wp};
        par inc nop nop inc {mm # c wp}; le 9eme bit.
        return;
        end;
```

A partir de la première instruction de la boucle, chaque bit du résultat de l'addition d'un processeur donné est directement transféré sur le port de droite et sérialisé dans la file gauche du voisin de droite. Dans cet exemple aucune altération n'est effectuée sur les files droites (Right Fifos) car l'opération gauche est un NOP (No OPeration). Un algorithme explicatif serait comme suit:

```
add_sendR(a[],b[],N)
debut
entier i;
booleen a[0:N-1],b[0:N-1],rd;
% initialiser la retenue %
r=0;
pour i =0 à N-1
faire_parallèle
envoyer_voisin_de_droite(addition_binaire(a[],b[],r); positionner la retenue;
recevoir_résultat_voisin_gauche;
fin_faire
fin
```

5. APPLICATIONS

Dans cette partie, nous reprenons les deux applications décrites dans [9] afin de donner une comparaison entre leur mise en oeuvre respective dans SLiM (où le chemin de donnée PE est de 8bits) et SMAL (PE sériel 1bit). Ces applications sont le calcul de convolutions bidimentionnelles et le filtrage médiane.

5.1. CONVOLUTION

Le calcul de convolutions d'une image, est un exemple que l'on retrouve souvent dans les publications traitant d'architectures parallèles étant donné son importance en traitement d'images [10],[11],[12]. Une publication concernant ce sujet, et spécialement la convolution 2D, a été faite par Lee et Aggarwal [13]. Ce dérnier a en effet introduit la notion de chemin hamiltonien. L'existence d'un chemin hamiltonien selon la fenêtre, permet de gagner aussi bien en performance de calcul sur le rapport calcul/communication, qu'en espace mémoire pour la sauvegarde des résultats intermédiaires.

Le calcul d'une convolution revient à une multiplication/ accumulation, de la forme $y_i=y_{i-1}+w_i*x_i$. Cependant pour un calcul bidimensionel deux approches sont possibles, tenant compte de l'existence d'un chemin hamiltonien:

Si l'on considère l'architecture SLiM, la première méthode consiste en la propagation de pixels ce qui donnerait l'algorithme suivant [9]:

```
faire_parallèle
T<-s*w0; s<- pixel voisin;
fin_faire
pour i=1 à window_size-1
faire_parallèle
        T<-T+s*wi;
        s<- pixel voisinant;
fin_faire
```

Où T représente un registre interne à un processeur élémentaire et s un plan glissant (sliding plan). Toute l'image est contenue dans ce plan glissant. Les calculs et les transferts sont dans cet exemple parallélisés.

La seconde méthode, séquentielle (par rapport à un processeur élémentaire), consiste à propager les résultats intermédiaires:

```
T<-w0*p;
pour i=1 à window_size-1
faire
        s<- T_voisin;
        T<-s+wi*p;
fin_faire
```

Où p est cette fois-ci la valeur du pixel de l'image, correspondant à un processeur donné et s ne sert ici qu'en simple registre.

Le premier algorithme est plus adéquat à l'architecture SLiM. D'ailleurs, cette dernière a été spécialement conçue pour résoudre ce genre de problème.

Le deuxième algorithme est plus adéquat pour SMAL. En effet, avec la sérialisation des opérations, et une communication à base de files (FIFOs), l'absorption des communications par le calcul s'obtient, comme nous l'avons précisé dans l'exemple de l'addition ci-dessus.

Dans SMAL, il est possible de parler d'algorithmes de calcul à double chemins hamiltoniens géométriquement symétriques par rapport au centre d'une fenêtre, comme le montre la figure 8. Cette symétrie est adoptée pour utiliser les liaisons bidirectionnelles et les deux axes de communications (horizontalement ou verticalement selon le contrôle) présents dans SMAL_X31.

Dans le cas ou il n'existe pas de chemins hamiltoniens, des étapes de communications supplémentaires, montrées en pointillés sur la figure 8, apparaissent. Quant au nombre de ces étapes, il dépend de la construction topographique de la fenêtre.

Une fenêtre peut être vue comme le résultat d'une suite d'applications successives d'opérateurs d'extension sur une partition noyau [14]. C'est ainsi qu'une fenêtre en diamant impaire [13] d'ordre 2n+1, peut être obtenue par 2n applications de l'opérateur 5-points sur un point (voir figure 9). Pour ce même exemple le nombre d'étapes supplémentaires est de 2n.

5.2. FILTRE MEDIANE

Le filtrage par médiane consiste à remplacer un point d'une image (pixel) par la médiane de son voisinage. La médiane d'un ensemble de nombres est un nombre choisit de telle sorte que la moitié des nombres de l'ensemble lui soit inférieure et l'autre moitié lui soit supérieure. Le calcul de la médiane d'un pixel neccessite alors un tri sur le voisinage défini par une fenêtre. Le tri local à chaque processeur élémentaire d'une machine SIMD est une tâche coûteuse en temps surtout pour des machines ne disposant pas d'autonomie d'adressage.

La méthode décrite dans [9], consiste à construire une liste unidirectionnelle ordonnée sous la forme d'une zone contigüe. Avec le mécanisme de l'image glissante (chemin hamiltonien unique) nous obtenons l'algorithme suivant:

```
pour i=1 à window_size-1
faire_parallèle
        insertion(s,l);
        s<- pixel_voisin;
fin_faire
```

où **l** est la liste (chaque PE a sa propre liste locale). Le résultat est tel que dans tout processeur élémentaire nous avons un tableau de valeurs ordonnées. La médiane en chaque point serait alors la valeur se trouvant au milieu de ce tableau. Si on considère que les temps d'un décalage (pour l'opération d'insertion) et d'une comparaison sont équivalents alors dans le pire des cas nous obtenons un temps en $O(n^2/2)$, **n** étant le nombre de point de la fenêtre.

La construction d'une liste nécessite un espace mémoire proportionnel au nombre de voisins. Dans le cas de SliM par exemple, 9 mots de 8bits sont nécessaire pour une fenêtre 3x3 sachant qu'un pixel est codé sur 8bits. Pour une architecture à base de PEs 1bit, cela demanderait 72 bits.

Nous décrivons ci dessous un algorithme qui tient compte de l'espace mémoire: Le principe de l'algorithme est comme suit:

Soient **E** un ensemble de **2n** valeurs $\{e_1,..,e_{2n}\}$, **A** et **B** deux parties de **E** de taille **n** tel que **E=A+B**. Considérons $E'=\{r_1,..,r_{2n}\}$ comme résultat du tri de **E**. Alors nous cherchons à trouver les éléments successifs r_n et r_{n+1}. La recherche de ces éléments est équivalente à décomposer **E** en deux partitions **A** et **B** tel que tout élément de **A** est inférieur à tout élément de **B**:

a<-max(A);b<-min(B);
***tq**(a>b)*
faire
A<-A-a+b;B<-B-b+a;
a<-max(A);b<-min(B);
fin_faire

Le filtrage par médiane, sur une image, doit tenir compte de la répartition spaciale des données, sachant que l'on a un pixel par processeur élémentaire, nous proposons l'algorithme suivant:

faire_parallèle
a<-max(A);b<-min(B);
fin_faire
***tq**(il existe a>b)*
faire
faire_parallèle
A<-A-a;
B<-B-b; % phase élimination %
fin_faire
faire_parallèle
a<-max(max(A),b);
b<-min(min(B),a);
fin_faire
fin_faire

L'opération d'élimination d'un élément d'un ensemble n'est qu'un simple marquage. Chaque processeur élémentaire a une zone de taille **window_size-1** bits ou chaque bit correspond à un voisin. L'élimination d'un point par rapport au voisin revient à positionner à une valeur donnée le bit correspondant.

Nous constatons dans l'algorithme que l'on doit parcourir deux fois un ensemble à chaque pas de boucle: une fois pour la procédure d'élimination et une autre pour la recherche du "**min**" ou du "**max**". A la sortie de la boucle, il ne reste qu'à comparer l'élément central à **a** et **b** pour en déduire la médiane.

N'ayant pratiquement que des opérations de comparaisons cet algorithme a une complexité temporelle de $O(n^2/2)$. De plus, il ne consomme pas beaucoup d'espace mémoire.

La décomposition en deux ensembles est parfaitement adaptée au réseau SMAL. Le principe de double chemins hamiltoniens disjoints et les capacités d'autonomie du PE répondent directement aux besoins de la mise en oeuvre de l'algorithme. La figure 10 montre un exemple sur une fenêtre 3x3.

6. ETAT DU PROJET

Un processeur SMALX31 a été réalisé lors d'un projet de DEA [1c] au laboratoire CAO&VLSI de l'Université Pierre et Marie Curie. Les masques ont été envoyés au run du 15.IX.90 du CMP (Circuit MultiProjet) pour la fabrication. Le processeur élémentaire présente les caractérisques suivantes:

-technologie CMOS 2µm, 2 niveaux d'ALu.
-nombre de transistors: 6764.
-surface occupée:32.672mmsq.
-temps critique simulé: inférieur à 17ns.

La méthodologie "standard cell" a été adoptée pour la réalisation de ce premier prototype. Les étapes d'implantation se résument dans ce qui suit:

a-description fonctionnelle (GHDL[17]).
b-description structurelle (GHDL).
c-placement et routage du coeur (NSAR,BASIL).
d-placement et routage des plots d'E/S (manuelle).

Chaque étape est suivie d'une verification par rapport aux spécifications de départ et cela en utilisant comme outils de simulation HISIM[17]. La vérification est soit directe (étapes a et b) ou bien précédée d'une extraction (étapes c et d). Pour passer de l'étape b à c, nous avons introduit une interface GHDL/NSAR.

NSAR et BASIL sont respectivement le générateur de netlists et le routeur (canal global) de la chaine ALLIANCE développée au laboratoire CAO&VLSI de Paris 6. Ces outils ont été conçus pour des architectures régulières (opérateurs nbits...), ce qui ne nous permet pas une optimisation en surface dans le cas de notre processeur.

La figure 11 , montre le dessin de masques du processeur élémentaire SMAL_X31. Les pertes en surface y sont visibles.

Remarque: Pour faciliter le test du processeur élémentaire, plus de 1200 transistors ont été nécessaires à l'introduction d'un chemin de test.

Pour réaliser un réseau de 2x2 nous serons amenés à utiliser des outils mieux appropriés à notre architecture.

En ce qui concerne les autres éléments, le contrôleur µSMAL et les filtres ont été entiérement décrits en langage GHDL[17] et simulés sous HISIM[17]. Nos estimations en nombre de transistors de µSMAL est de l'ordre de 80000 et le nombre de broches est de 110.

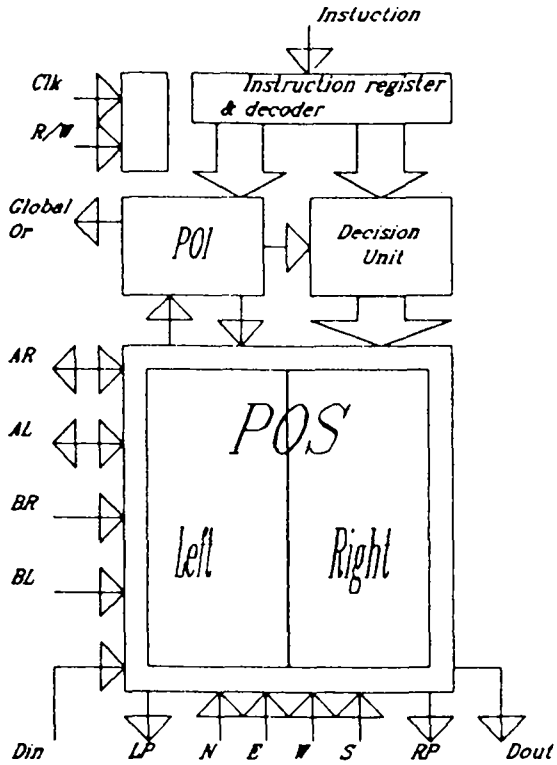
Une nouvelle version du processeur élémentaire, SMAL_A31, apportant des améliorations impliquant des performances plus élevées est en phase de spécification. Notre objectif et de permettre un meilleur recouvrement des entrées/sorties avec les communications, et d'étendre l'autonomie de fonctionnement du processeur élémentaire aussi bien sur le plan d'exécution d'opérations que sur le plan des communications.

7. CONCLUSION

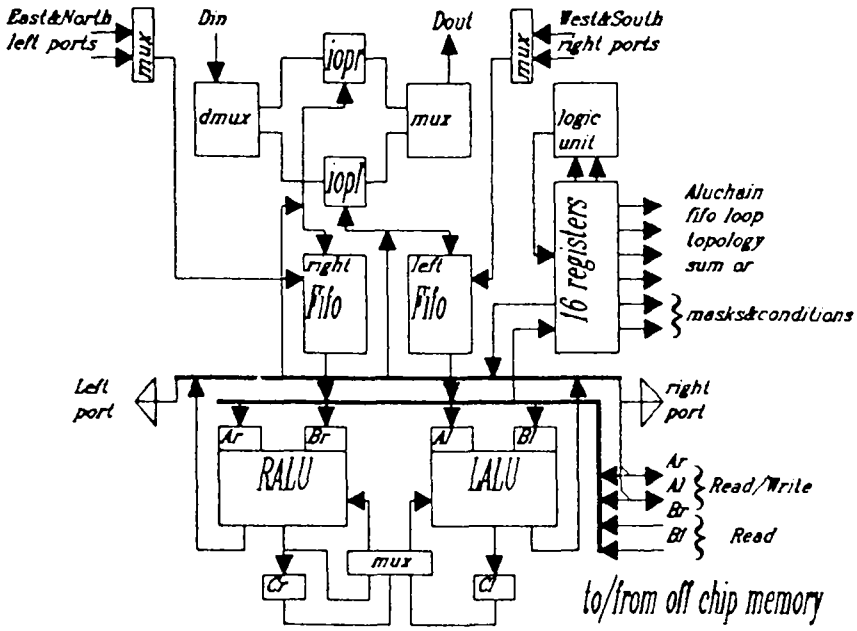
L'architecture parallèle SMAL conçue à base d'éléments nouveaux, qui sont le processeur élémentaire SMAL_X31 et le contrôleur µSMAL, a été décrite. Le processeur SMAL_X31 a déjà fait objet d'une réalisation qui nous servira comme donnée importante pour nos prochaines recherches.

Le processeur élémentaire est doté d'un mécanisme de décision lui donnant une meilleure efficacité pour l'exécution d'instructions conditionnelles. Le système d'échanges entre processeurs élémentaires est à base de Files (FIFOs) répondant ainsi au fonctionnement à caractère sériel. De plus, avec un double chemin d'échanges et avec l'adoption de chemin de calcul, nous pouvons atteindre un recouvrement calcul/communication optimal et ainsi nous diminuons sensiblement la charge temporelle des communications qui constituent un point important dans les réseaux de type "MESH".

Nos prochaines recherches porteront, principalement, sur la réalisation de µSMAL et l'élaboration d'une nouvelle version du processeur élémentaire, SMAL_A31.

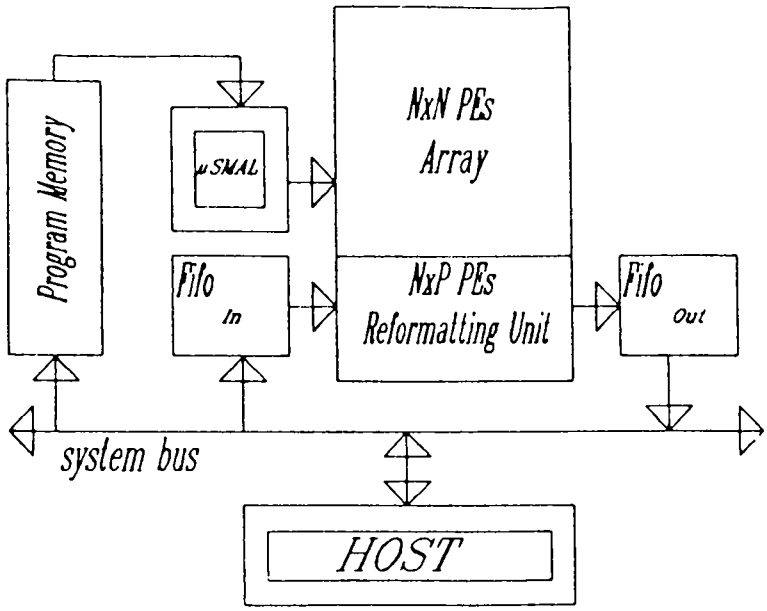


a) L'interface et le découpage fonctionel de SMAL_X31.

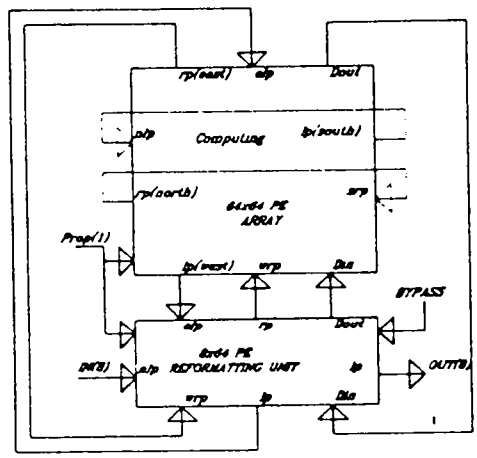


b) Les chemins de données de SMAL_A31.

Figure 1: Architecture de SMAL_X31.



a) Les différentes partie de SMAL et intégration dans une machine hôte.



b) Les matrices de processeurs élementaires.

Figure 2: Architecture type d'un réseau SMAL.

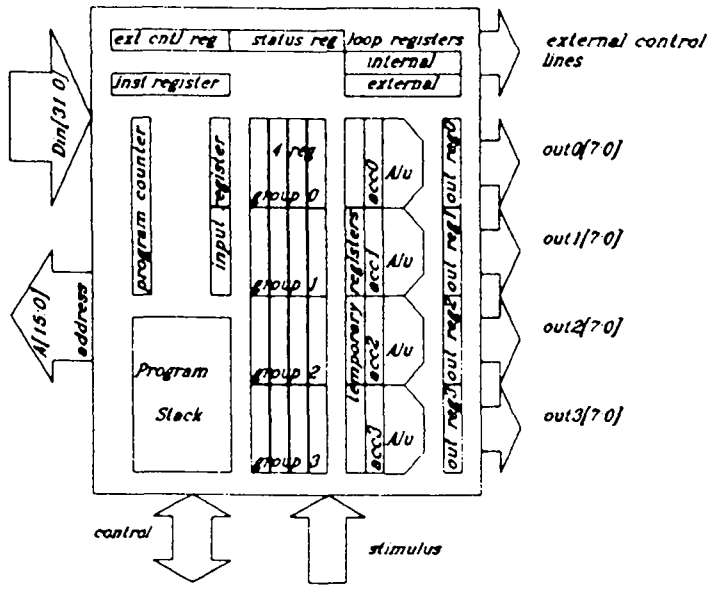


Figure 3: Synoptique de l'architecture du contrôleur µSMAL.

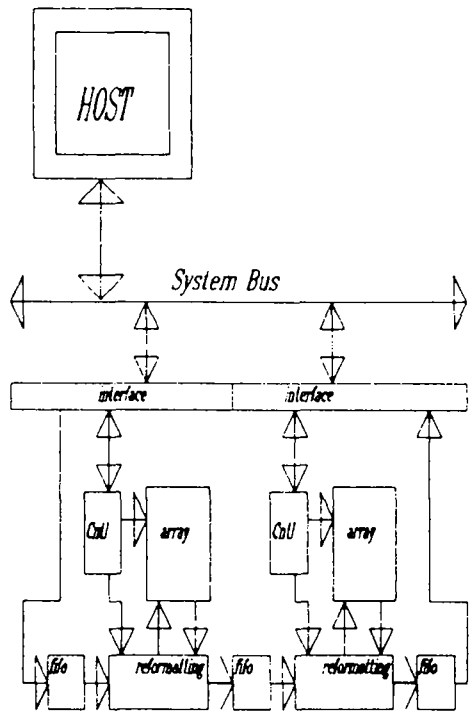


Figure 4: Cascade de modules SMAL.

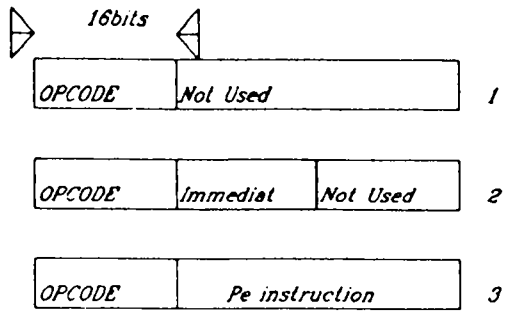


Figure 5: Formats de l'instruction.

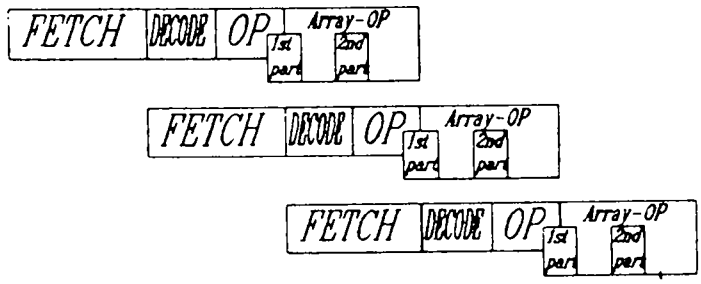


Figure 6: Pipline d'exécution.

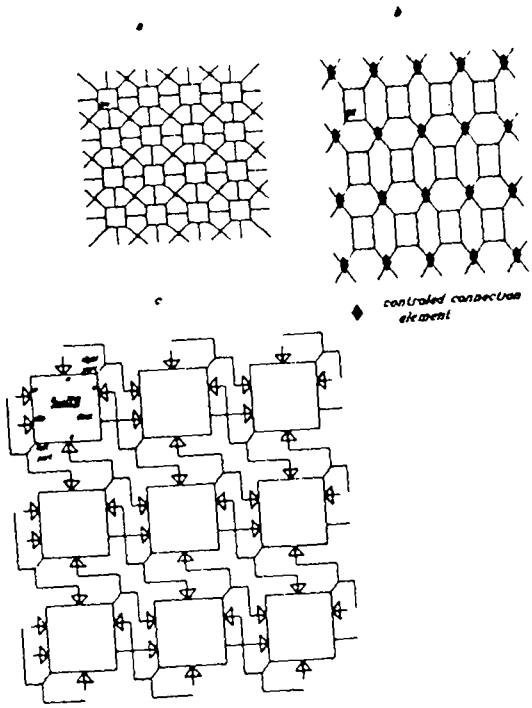


Figure 7: Interconnexions matricielles:
a) huit voisins.
b) X (Blitzen).
c) SMAL.

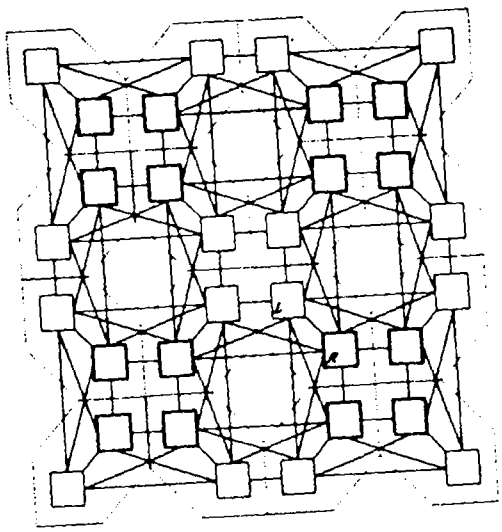


Figure 7d: Interconnexions logiques dans SMAL tenant compte des parties gauche et droite.

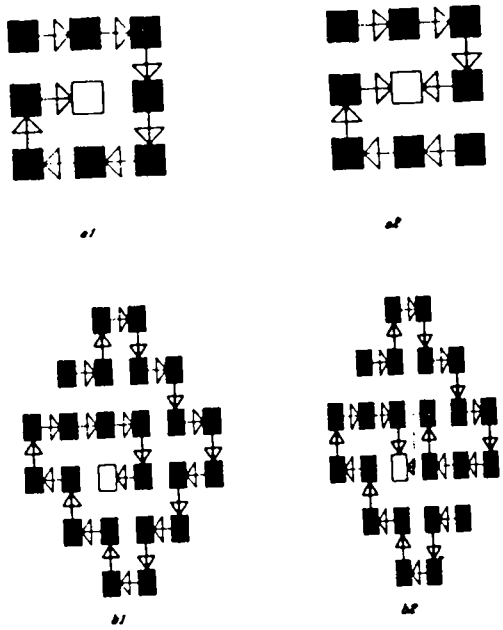


Figure 8: Chemins de convolution:

a) fenêtre carré 3x3 a1)SLiM a2)SMAL.
b) fenêtre en diamant 6x6 b1)SLiM b2)SMAL.

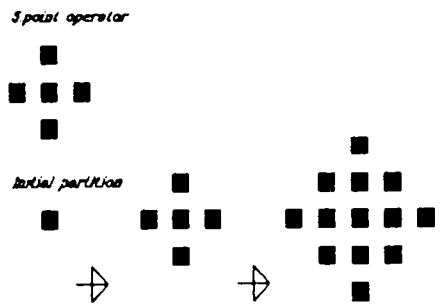


Figure 9: Extension d'une partition par un opérateur 5_points.

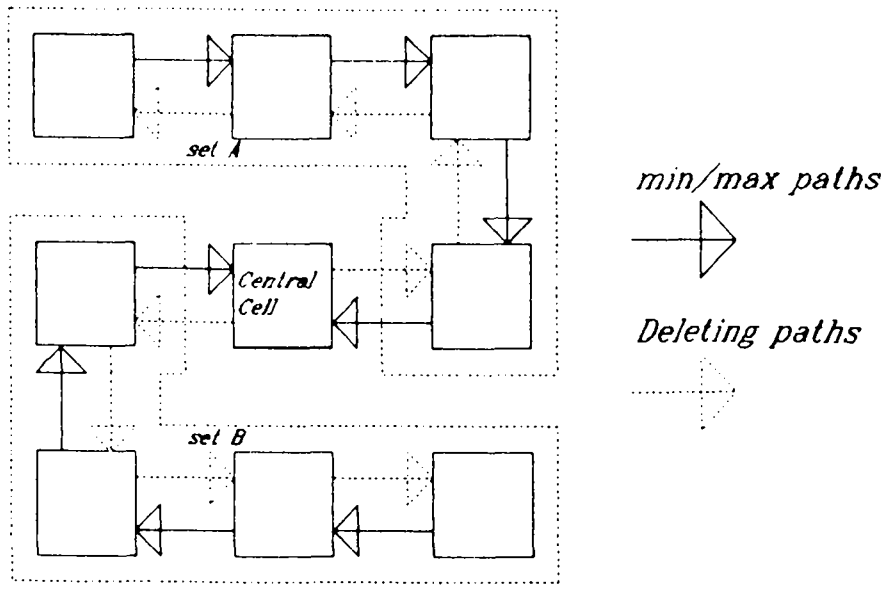


Figure 10: Chemins hamiltoniens pour le calcul de la médiane.

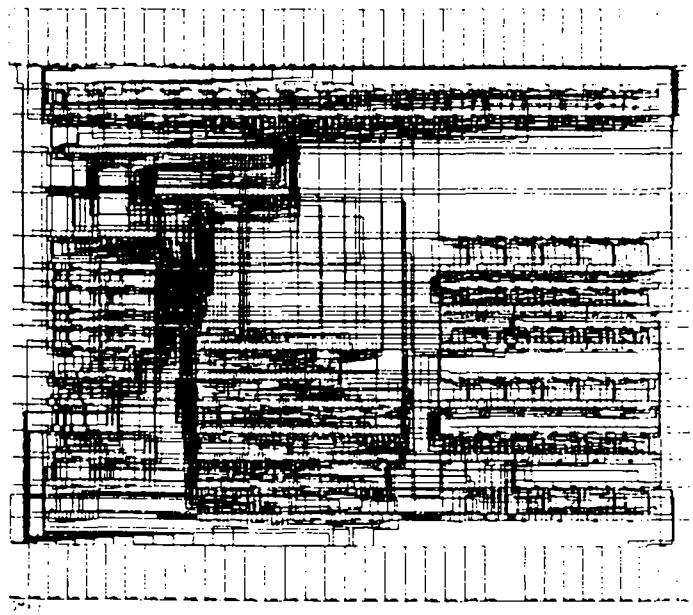


Figure 11: Dessin de masques du processeur SMAL_X31.

BIBLIOGRAPHIE

[1a].B.Zerrouk, "SMAL_X31, une architecture de granularité fine pour le traitement d'images et l'émulation neuro-mimétique", Rapport de recherche, INRIA Décembre 1989.

[1b].B.Zerrouk, "A SIMD Multiprocessor architecture based on a fine grain processor for image processing and neural networks emulation", IFIP Workshop On Parallel Architecures On Silicon", Grenoble, december 1989.

[1c].I.SIROT et A.ZOLFAGHARI,"Projet SMAL_X31, processeur 1bit", Rapport de DEA CAO&VLSI Paris 6, Juillet 1990.

[2].J.L.Potter,"The Massively Parallel Processor", the MIT Press 1985.

[3].D.W.Blevins et al,"BLITZEN: A highly integrated massively parallel machine", Journal of Parallel and Distributed Computing 8,1990.

[4].W.D.Hillis,"The Connection Machine",The MIT Press 1985.

[5].M.Maresca and H.Li,"Connection autonomy in SIMD computers: A VLSI Implementation", Journal of Parallel and Distributed computing 7,1989.

[6].E.L.Cloud,"The Geometric arithmetic Parallel Processor", IEEE Second Symposium On frontiers of massively parallel computations, 1988.

[7].T.J.Fountain,"Processor Arrays, Architecture and Applications", Academic Press 1987.

[8].A.Boubeker et al,"Reconfiguration in ELSA", IFIP Workshop On Parallel Architecures On Silicon", Grenoble, december 1989.

[9].M.H.Sunwoo and J.K.Aggarwal,"A Sliding Memory Plan Array Processor", IEEE Second Symposium On frontiers of massively parallel computations, 1988.

[10].P.Quinton et Y.Robert,"Algorithmes et Architectures Systoliques", édition Masson, 1989.

[11].P.M.Dew et al,"Parallel Processing for Computer Vision and Display", Addison Wesley 1989.

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

[13].S.Y.Lee and J.K.Aggarwal,"Parallel 2-D Convolution on Mesh Connected Array Processor", IEEE Trans. On Patern Analysis And Machine Intelligence,Vol.PAMI-9,NO.4, July 1987.

[14].F.F.Lee," Partitioning of Regular Computation on Multiprocesseor Systems", Journal of Parallel and Distributed Computing 9,1990.

[15].B.Zerrouk,"L'assembleur SMALASS", rapport interne.

[16].B.Zerrouk,"μSMAL: architecture et description GHDL", rapport interne.

[17].Manuels HILO,HISIM,GHDL de GenRad.

Imprimé en France
par
.l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399