SMAL-X 31 une architecture parallele de granularite fine pour le traitement d'images et l'emulation neuro-mimetique

Belkacem Zerrouk

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° 1146

Programme 2
Structures Nouvelles d'Ordinateurs

SMAL-X 31
UNE ARCHITECTURE PARALLELE DE GRANULARITE FINE POUR LE TRAITEMENT D'IMAGES ET L'EMULATION NEURO-MIMETIQUE

Belkacem ZERROUK

Décembre 1989

RR_1146

SMAL_X31
UNE ARCHITECTURE PARALLELE DE GRANULARITE FINE POUR LE TRAITEMENT D'IMAGES ET L'EMULATION NEURO_MIMETIQUE

A FINE GRAINED PARALLEL ARCHITECTURE FOR IMAGE PROCESSING AND NEURAL NETWORKS

Belkacem ZERROUK

INRIA Rocquencourt
Projet Archi
78150 LE CHESNAY CEDEX FRANCE

Résumé:

L'unique solution en vue répondant aux exigences topologiques et temporelles d'une exécution performante d'applications réparties est une structure multiprocesseurs. Nombreuses sont les applications nécessitant une répartition de tâches fines et identiques sur l'ensemble des processeurs. Les architectures SIMD à granularité fine sont très adaptées à ce genre d'applications. Nous introduisons dans cet article une architecture régulière basée sur un réseau matriciel de cellules synchrones. Chaque cellule est un processeur élémentaire de conception nouvelle, simple et flexible.

Abstract:

The best known solution that satisfies an improved distributed time consuming tasks execution is a multi-processor structure. A large class of highly parallel algorithms can be processed efficiently on regular fine grained processor array. SIMD processor array based architecture is a cost effective alternative, in comparison with usual Von-Neuman processors, efficiently used to perform regular and recursive algorithms. This paper introduces the design of a fine grained SIMD array based architecture whose main features are originality, simplicity and flexibility at processor level.

1 Introduction

Le développement des technologies micro-électroniques acquis durant la dernière décennie a relancé l'intérêt pour certains concepts qui, il y a encore quelques années, trouvaient des difficultés pour leur concrétisation réaliste. Un de ces concepts auquel s'attachent plusieurs équipes de recherche dans le monde est le parallélisme massif [1]. Ce terme est usité pour définir des réseaux réguliers de plusieurs milliers de processeurs élémentaires chacun muni d'une mémoire locale.

Les réseaux de processeurs peuvent être classés, selon l'architecture adoptée, en deux catégories au sens de Flynn: les architectures MIMD, de Multiple Instruction Multiple Data, et les architectures SIMD, de Single Instruction Multiple Data. Dans le cas des architectures MIMD, les processeurs opèrent de manière asynchrone chacun exécutant une tâche propre sur un ensemble de données locales ou globales. Par contre, les processeurs d'un réseau SIMD sont synchrones et exécutent simultanément la même instruction.

Le principal avantage des réseaux MIMD est leur adaptation aux algorithmies parallèles irrégulières, alors que les réseaux SIMD répondent efficacement aux algorithmies régulières et récursives. Cependant la complexité de conception et les problèmes d'interblocage, lors des communications inter_processeurs, et de synchronisation de tâches affectent de façon défavorable les architectures MIMD en comparaison avec les architectures SIMD [23].

Certaines architectures SIMD sont des réseaux matriciels de processeurs sériels 1 bit identiques. Les opérateurs de ces processeurs se ramènent à de simples tranches 1bit (1 bit slice). Une fonction complexe doit être décomposée en un ensemble d'opérations sérielles réalisables par ce type d'opérateurs.

Cette simplicité de conception du processeur élémentaire est un point fondamental dans la perspective de réaliser une architecture massivement parallèle.

Plusieurs réseaux à base de processeurs sériels 1 bit ont déjà vu le jour et démontrent la faisabilité des "nouvelles" architectures massivement parallèles. Citons pour mémoire: le MPP de la NASA, la Connection Machine (CM) de Thinking Machine Corporation et DAP de International Computer Limited (U.K) ... [1] [19] [21] [23]

L'objectif primordial dans la réalisation d'une machine massivement parallèle est de pouvoir intégrer, en fonction de la technologie VLSI utilisée, le

plus de processeurs élémentaires sur une même tranche de silicium (chip). En effet, plus le nombre de processeurs intégrés est grand moins on a de soucis sur l'encombrement du système final et moindre serait le coût de ce même système (cost effective design). En exemples, la puce élémentaire MPP contient 8 processeurs, une puce CM en contient 16 et une puce GAPP intègre 72 processeurs avec 128 bits de mémoire locale pour chacun. Un chiffre plus récent est celui atteint dans le projet BLITZEN [22] où une puce, de plus du million de transistors, intègre 128 processeurs chacun muni d'une mémoire de 4x256 bits.

Les architectures massivement parallèles, en particulier les machines SIMD caractérisées par le nombre important de processeurs sériels 1 bit, offrent une grande capacité de calcul à un coût relativement faible. Les performances d'un réseau SIMD sont fonction du cycle instruction et du nombre de processeurs qu'il contient: si une opération *OP* se décompose en K opérations binaires du répertoire de base, le réseau peut réaliser jusqu'à *N.K/cycle opération OP par seconde* où N est le nombre de processeurs. Par exemple, avec 65536 processeurs, la Connection Machine CM peut atteindre 200 MFLOPS, et avec 16384 le système MPP peut faire 300 MFLOPS [1].

Notons sur les exemples ci-dessus que malgré le nombre plus important de processeurs chez la CM, le MPP est le plus performant d'après les chiffres donnés. En réalité, ce genre de comparaison n'est pas valable. En effet, plusieurs raisons de conception et de réalisation entrent en jeu, voir la topologie, la distribution d'horloge etc... , qui font que le cycle instruction du MPP est largement inférieur au cycle instruction CM.

Le parallèlisme massif et en ce qui nous concerne lorsqu'il est régulier et fait appel aux machines SIMD, couvre divers domaines d'application dont: la vision, l'intelligence artificielle et, avec une actualité pertinente, les réseaux de neurones. Le lecteur est invité à consulter l'ouvrage de Potter [21] pour avoir une idée sur les applications possibles.

Nous introduisons dans la suite de cet article les éléments architecturaux d'une machine parallèle SIMD. Nous y définirons le processeur élémentaire qui forme la base matérielle de la conception.

Un simulateur fonctionnel de réseaux maillés et toriques formés de plusieurs processeurs nous a permis de valider notre conception sur des exemples d'applications tels ceux donnés dans la dernière partie de l'article.

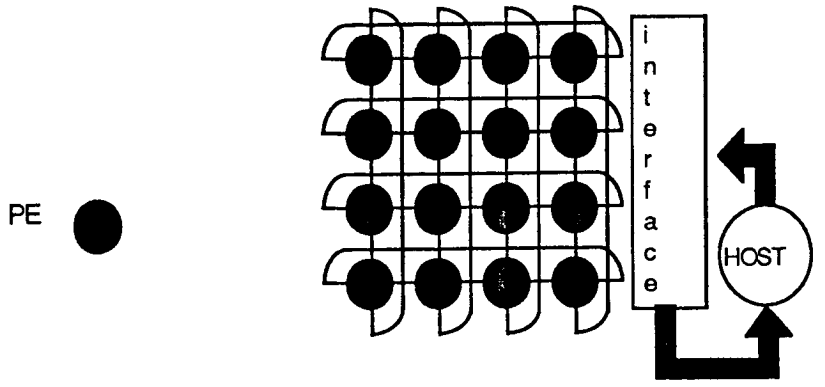

Figure 1: Topologie générale.

2 Les choix fondamentaux

Le réseau est constitué d'un ensemble de processeurs élémentaires identiques appelés PE. Chaque processeur est muni d'une mémoire locale. La conception d'un PE est fortement liée à certains choix effectués de prime abord:

1. Adoption d'un modèle SIMD pour des raisons de simplicité de conception et de programmation, de la diversité des applications possibles et de sa flexibilité.

2. Choix d'une granularité fine : la complexité des opérateurs locaux à un PE est de 1 bit. Nous visons ainsi l'intégration de plusieurs processeurs dans un seul chip ceci afin de minimiser le coût de conception et de réalisation.

3. Assurer une flexibilité dans l'autonomie des processeurs, par rapport à un fonctionnement SIMD classique, avec la mise en oeuvre d'un schéma d'exécution conditionnel souple.

4. Choix d'une topologie régulière simple et non coûteuse: réseau matriciel.

3 Autonomie des processeurs

L'autonomie locale des processeurs est un point important dans la mise en oeuvre d'une programmation flexible d'un réseau de processeurs SIMD. Le

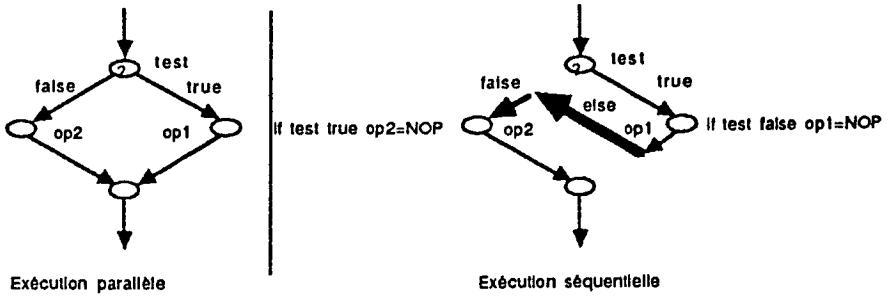

Figure 2: Exécution d'un if-then-else.

plus souvent, un minimum d'autonomie est assuré pour permettre l'exécution d'instructions conditionnelles. Dans le cas le plus simple, un registre d'un bit est employé spécialement pour cela. C'est le cas de la Connection Machine ou du MPP.

La solution généralement adoptée pour les schémas conditionnels et en particulier le IF-THEN-ELSE est donnée par la reformulation suivante:

if (condition true) op00,op01,..
else No operation;
if (condition false)op10,op11,..
else No operation;

Selon l'état de son registre condition, un processeur peut être soit actif soit inactif. L'opération à effectuer n'est effective que si l'expression conditionnelle est évaluée VRAIE. Ce type d'implantation sur une machine SIMD engendre une perte de temps.

Une première solution peut être apportée à ce problème d'oisiveté avec une restriction à des classes d'opérations. Le plus simple est de regrouper en couples des opérations que l'on retrouve dans certains calculs où elles jouent des rôles complémentaires (voir projet BLITZEN). Ainsi, si une condition est vérifiée l'opération spécifiée dans un champ donné de l'instruction initiale est réalisée, sinon l'opération complémentaire prend place moyennant un mécanisme de décision interne au processeur.

La solution ci-dessus n'est intéressante que si le nombre d'opérations offertes par l'ALU est petit, par exemple juste l'addition, la soustraction, la mise à 1 et la mise à 0. Dans le cas contraire une solution envisageable

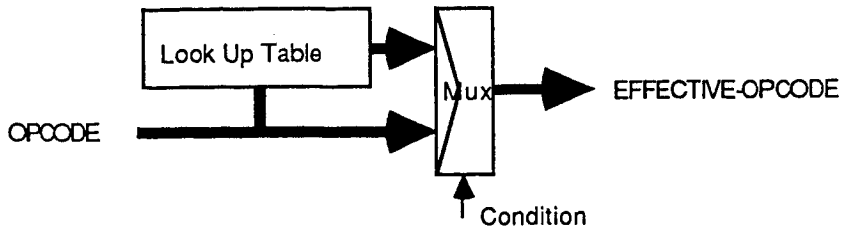

Figure 3: Table de correspondance.

serait l'utilisation d'une table de correspondance locale ou LUT, cependant certains points restent à préciser, principalement:

1. La taille de la table et son implantation.

2. L'influence de l'accès à la table sur le temps de cycle.

La taille de la table dépend directement du codage des opérations et du choix de l'ensemble des couples d'opérations à mettre en correspondance. L'implantation sous forme d'une RAM ou bien d'une ROM pose des problèmes. Dans le cas d'une implantation RAM, un téléchargement est indispensable.

Le dispositif de téléchargement rendrait complexe la conception mais en revanche on dispose d'un moyen de reprogrammabilité. Contrairement à cette première solution, avec la solution ROM le choix des couples d'opérations est figé dès l'implantation ce qui la rend non flexible.

Dans les deux cas la "perte" en surface est préoccupante; en effet à chaque processeur élémentaire nous devons associer une table et dans l'hypothèse ou nous visons l'intégration de plusieurs processeurs sur une même tranche de silicium , cela constituerait un handicap. Quand au cycle de l'instruction et particulièrement au temps nécessaire à la phase de décision, il est évident qu'il doit être augmenté du temps d'accès à la table. Ce point peut être sans importance dans le cas de tables de petites tailles.

Le choix d'une petite taille pour la table constituerait d'ailleurs un bon compromis espace-vitesse.Cependant, en fonction du jeu d'opérations cela peut mener à des restrictions pour répondre au dernier facteur.

La solution apportée dans notre conception,regroupe flexibilité et simplicité. Elle se base sur l'idée originale de réaliser un processeur élémentaire bi-opérateurs, pouvant être vu comme l'adjonction de deux processeurs ordinaires,chacun muni d'un opérateur de base 1bit seriel,identifiés par MSB et LSB. Ces processeurs fonctionnent de manière parallèle et partagent des ressources locales.

Le principe de fonctionnement du processeur élémentaire repose principalement sur deux modes:

1. Fonctionnement en conditions séparées.

2. Fonctionnement en condition commune.

Dans le mode "conditions séparées", les processeurs MSB et LSB sont totalement indépendants et peuvent réaliser des opérations différentes. A chacun des processeurs est associé un registre condition sur un bit.

Dans le mode "condition commune", la même opération doit être réalisée sur chacun des deux opérateurs (ALUs) MSB et LSB. En spécifiant les deux opérations possibles, le choix de l'opération à réaliser, dépend de l'état d'un registre de condition unique. Ce mécanisme de décision peut être un simple multiplexage des champs opérations de l'instruction.Ainsi nous pouvons former n'importe quel couple d'opérations réalisables par les deux opérateurs.

Le second point lié à l'autonomie des processeurs est celui de l'adressage des opérandes. Vu le choix que nous avons fait sur l'implantation de la mémoire locale nous n'allons pas nous attarder sur ce sujet. Le lecteur peut consulter l'ouvrage de Fontain [1] ainsi que [7] pour plus de précisions.

Avec ce type d'autonomie, l'adresse d'un opérande peut être évaluée localement, de manière à ce que les processeurs puissent opérer sur différentes adresses. L'évaluation peut être complète ou partielle. Dans le premier cas tous les bits du champ d'adressage peuvent être calculés, par contre, dans le second cas l'évaluation porte sur un ensemble de bits bien défini, les bits de poids faible par exemple.

Il est important de noter que l'implantation de ce genre d'autonomie n'est intéressante que dans le cas où la mémoire locale est intégrée dans la même puce que le processeur correspondant. Dans le cas où la mémoire est externe à la puce on a besoin, en conséquence, du nombre de broches nécessaire pour sa connexion. Ce nombre peut être important si la mémoire locale est de grande taille et si l'évaluation locale d'adresses est complète.

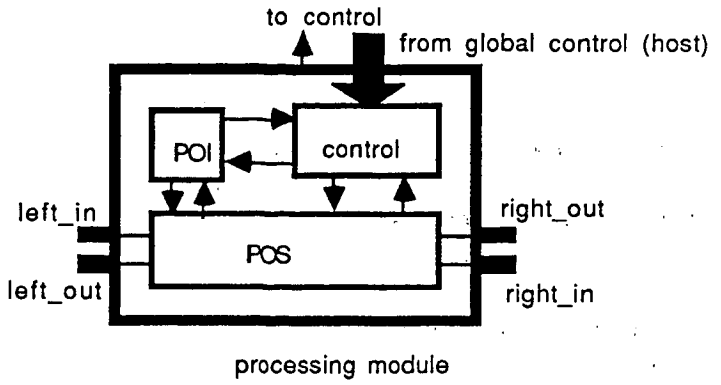

Figure 4: Synoptique du module de traitement.

4 Architecture

Un processeur élémentaire est composé de deux modules: le module de communication et le module de traitement . L'interface entre ces deux modules est composée de deux ports et de deux files. La partie traitement peut envoyer des résultats à ses voisins via la partie communication, à l'aide des deux ports en sortie et en recevoir des données par les deux files d'entrée.

L'architecture interne du module de traitement, est composée de trois parties :

- Une partie contrôle chargée du décodage des instructions,
- Une partie opérative inférieure.
- Une partie opérative supérieure.

  La partie opérative inférieure est aussi appelée unité de masques d'exécution.

4.1 Topologie et interconnexion

Le choix de la topologie d'un système à plusieurs processeurs dépend des primitives matérielles offertes par l'indispensable partie de communication qui sert de support au routage de messages inter-processeurs. Cette communication peut être globale; un processeur peut communiquer avec n'importe quel autre processeur du réseau, ou bien locale;

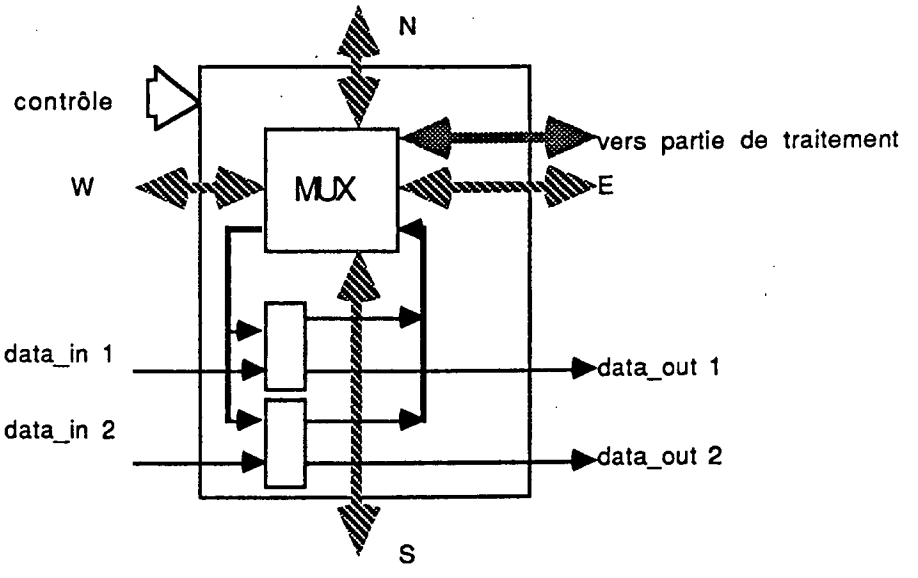

Figure 5: Le module de communication.

chaque processeur ne peut communiquer qu'avec ses proches voisins au sens géométrique.

La réalisation d'une communication globale est très coûteuse et forme une barrière importante pour l'intégration de plusieurs processeurs sur une même puce. L'exemple que l'on peut donner est celui de la Connection Machine: une puce de base de cette machine comporte 16 processeurs élémentaires. La communication locale à une puce est réalisée par une partie locale de routage de messages. Une communication globale, sur un hypercube (de puce) de dimension 12, est reportée à un organe externe.

La circuiterie de communication locale occupe un pourcentage très significatif de la surface de silicium. Cela limite le nombre de processeurs. Cette raison fait qu'une topologie simple et régulière est en général retenue.

Les topologies engendrées par un réseau maillé forment un choix très apprécié aussi bien par sa simplicité et son faible coût que par la possibilité de mettre en oeuvre diverses applications.

Ayant choisi une topologie matricielle, nous proposons de munir le module de communication d'un processeur élémentaire de quatre liens bidirectionnels vers les cellules voisines: Haut, Bas, Gauche et Droite.

Du fait que nous ne disposons que de deux liens bidirectionnels entre la partie de communication et la partie de traitement, le routage de données établies,à un instant donné, est soit vertical soit horizontal.

Si le processeur (PE) de base est vu comme bi-processeurs (MSB,

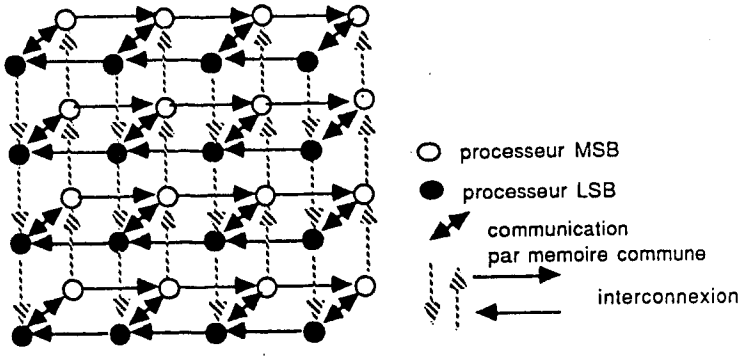

Figure 6: Topologie cubique 2*N*M.

LSB), communiquant à l'aide d'une mémoire locale commune, une topologie en réseau maillé N*M de PEs peut être vue en topologie cubique 2*N*M où les sommets sont les processeurs MSB et LSB appartenant à chaque PE.

La partie communication est aussi chargée, dans notre conception, de l'échange avec le calculateur hôte ; les entrées/sorties effectuées depuis ou vers la machine hôte sont assurées dans cette partie indépendamment du traitement. Ces entrées/sorties se font par plans "binaires" moyennant un plan d'entrées et sorties formé par l'ensemble des couples de registres, 1bit chacun, appartenant à une partie de communication, et des éléments d'échanges des modules de communication avec les modules de traitement correspondants.

D'autres solutions aux entrées et sorties peuvent être rencontrées. Sans doute, la plus usitée se base sur une architecture permettant au système hôte d'accéder directement aux différentes mémoires locales. Cette solution vise à augmenter le débit dans les deux sens. En effet, les mémoires locales, indépendamment du caractère sériel du processeur élémentaire, peuvent avoir des mots de plusieurs bits (4 par exemple) pour permettre un gain en débit. Un mécanisme local doit être prévu pour assurer l'accès sériel tel qu'on le sous entend. Cette architecture est coûteuse. Aussi, les processeurs ne peuvent pas accéder lors de ces échanges aux données-mémoires. Ceci peut impliquer la mise en inactivité des processeurs élémentaires, selon l'architecture du processeur élémentaire.

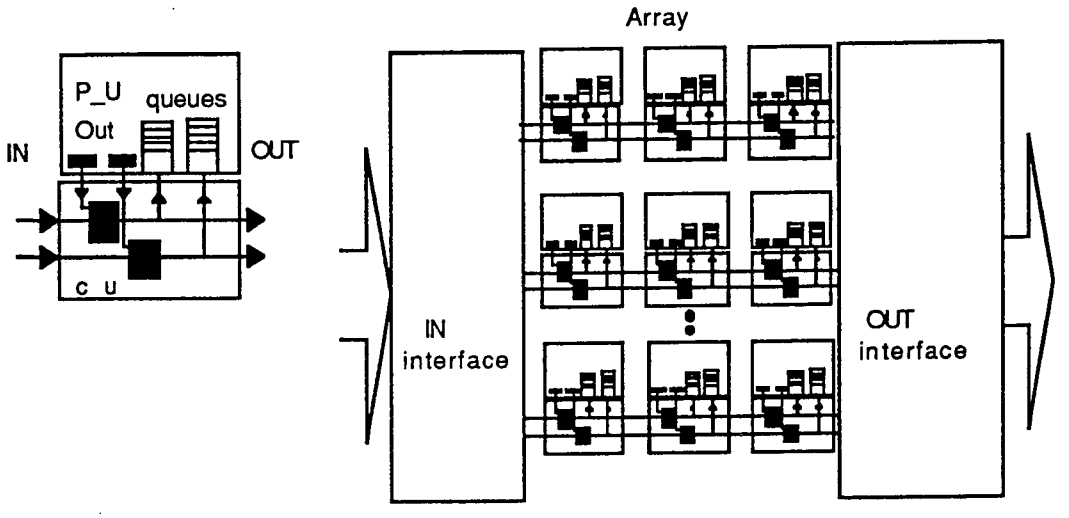

Figure 7: Les entrées sorties.

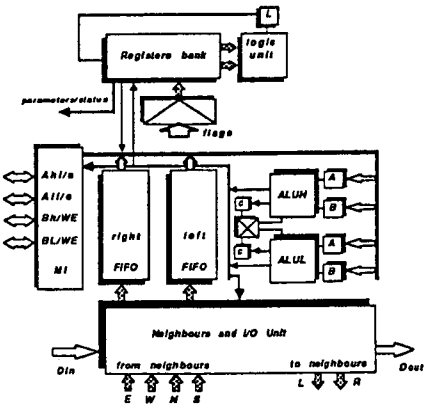

Figure 8: Le module de traitement.

La partie communication peut être commandée depuis l'extérieur de manière à assumer toutes les fonctionnalités envisagées lors de la conception.

4.2 La partie opérative supérieure: POS

Elle est constituée de deux opérateurs arithmétiques et logiques (ALUs), d'une mémoire locale, de deux files fifo d'entrées et de deux ports de sorties.

Nous distinguons deux sous-parties opératives l'une de poids fort et l'autre de poids faible : POSH et POSL. Deux opérations différentes peuvent être exécutées en parallèle moyennant cette subdivision.

Les deux opérateurs sont identiques chacun ayant deux registres d'entrées d'opérandes (A,B) et un registre retenue C. Pour des raisons de maniabilité, nous les avons choisis symétriques.

Les sources d'opérandes vers les ALUs et les destinations des résultats peuvent être de plusieurs types et sont explicitement définies par des champs de l'instruction à exécuter.

Deux modes d'opérations sont prévus pour les deux ALUs: le fonctionnement 1 bit et le fonctionnement 2 bits. Dans ce dernier cas les deux ALUs sont cascadées pour former un opérateur sur 2bits. La cascade se fait moyennant les entrées et sorties retenues.

Les files permettent la circulation aisée des résultats et des données: lors d'un calcul sériel, le résultat peut être directement acheminé vers une file d'entrée d'un processeur voisin sans recourir à une première sauvegarde mémoire.

L'autre fonction importante des files est le stockage de données d'entrée depuis une machine hôte. Cela nous permet de ne pas interrompre le traitement sur des opérandes mémoire lors des entrées.

La taille d'une file est fixée pour cette conception à 16 bits. A chaque file est lié un signal de défilement et d'enfilement. Chaque port de sortie est accompagné d'un signal donnée valide servant de signal d'enfilement dans le cas de la communication avec les proches voisins. L'état d'une file est donnée par deux signaux Vide et Pleine.

La mémoire locale est divisée en deux plans HM et LM. Pour chaque plan, l'accès est double en lecture, il est unique pour l'écriture. L'opération d'écriture peut être inhibée sur chacun des plans.

Pour des raisons d'espace d'intégration et de flexibilité, une mémoire locale externe au processeur est souhaitable. Cela nous permettra d'avoir plus de PEs dans un seul boîtier.

4.3 La partie opérative inférieure : POI

Cette partie est constituée de deux éléments: une unité de traitement logique à deux entrées d'opérandes (a,b) et un feuillet de 16 registres de 1 bit. L'unité logique peut effectuer les 16 opérations réalisables sur deux données booléennes.

R0(0)	bit de mise en cascade ou non des ALU de la POS
R0(1)	indéfini
R0(2)	sortie OU-global vers la machine hôte
R0(3)	indéfini
R1(0)	bit condition_commune (POSH ou POSH et POSL)
R1(1)	bit condition sur l'opération de la POSL
R1(2):	bit d'écriture du résultat de ALU de la POSH
R1(3):	bit pouvant être lu vers l'entrée B de l'ALU de la POSH

Table 1: Les bits des zones R0 et R1 du feuillet.

CH:	Carry de l'ALU POSH.
CL:	Carry de l'ALU POSL.
OH:	Résultat de l'ALU POSH.
OL:	Résultat de l'ALU POSL.
LQE:	File de gauche vide.
LQF:	File de gauche pleine.
RQE:	File de droite vide.
RQF:	File de droite pleine.

Table 2: Les indicateurs.

Le feuillet est organisé en 4 zones de 4bits chacun : R0,R1,R2 et R3. R0 est défini comme zone de paramètres pour la POS et de sortie vers un OU global. R1 est defini comme zone de conditions et d'échange avec la POS. Les zones R2 et R3 sont à usage général. La table 1 donne la signification des différents bits des zones R0 et R1 du feuillet.

Les deux registres R1[0] et R1[1] servent aussi de stockage pour la lecture parallèle de deux indicateurs/résultats de la POS (voir la table 2).

Le OU global sert de test pour un contrôle externe de l'état d'un réseau de PEs. Ce signal est pratiquement évalué par un arbre de Ou-logique dont les entrées sont les bits R0(2) des feuillets de tous les PEs appartenant au réseau. Son utilisation est indispensable pour la détection d'états stables ou encore de points de convergence, dans beaucoup d'applications: réseaux de neurones, squelétisation ...

S:	Champ de définition des sources d'opérandes des ALUs de la POS
D:	Champ de définition des destinations des résultats des ALUs de la POS
CC:	Champ condition
MSB:	Opération à effectuer dans la POSH
Ah:	Adresse mémoire de l'Opérande A de l'ALU de la POSH Peut être aussi l'adresse de destination du résultat
Bh:	Adresse mémoire de l'Opérande B de l'ALU de la POSH
LSB:	Opération à effectuer dans la POSL
Al:	Adresse mémoire de l'Opérande A de l'ALU de la POSL. Peut être aussi l'adresse de destination du résultat
Bl:	Adresse mémoire de l'Opérande B de l'ALU de la POSL
Op:	Opération à effectuer dans la POI
AA:	Adresse du premier Opérande de l'unite logique POI
BB:	Adresse du second Opérande de l'unite logique POI

Table 3: Les différents champs d'une instruction.

5 L' instruction

L'instruction de SMAL_X31 est du type long. Elle est composée de plusieurs champs. La table 3 donne leurs significations.

Les opérations POS et POI sont effectuées en parallèle dans le même cycle d'exécution. Une opération POS (LSB ou MSB) peut être soit une opération ALU soit une opération vide NOP.

La présence d'une opération NOP inhibe les signaux entraînant une modification du contenu d'une zone mémorisante, selon le contenu des champs S et D. Ces signaux sont essentiellement l'écriture mémoire, l'enfilement et le défilement des files.

Les champs sources d'opérandes et destinations de résultats, respectivement S et D, ne concernent que les opérations POS. Ils sont définis selon leur contenu dans ce qui suit.

-Sources d'opérandes .

Nous avons regroupé les sources d'opérandes selon trois types princi-

mnémoniques	source de l'opérande B	source del'opérande A	opérateur
MF	RQ LQ	HM(Bh) LM(Bl)	ALUH ALUL
FM	RQ LQ	LM(Bl) HM(Bh)	ALUL ALUH
RM	R LM(Bl)	HM(Ah) LM(Al)	ALUH ALUL
MM	HM(Bh) LM(Bl)	HM(Ah) LM(Al)	ALUH ALUL
XM	LM(Bl) HM(Bh)	HM(Ah) LM(Al)	ALUH ALUL

Table 4: Les sources d'opérandes.

mnémoniques	chemins de sauvegardes ou de sorties	
WP	ALUH → RP	ALUL → LP
IWP	ALUH → LP	ALUL → RP
WM	ALUH → HM(Ah)	ALUL → LM(Al)
IWM	ALUH → LM(Ah)	ALUL → HM(Al)
WR	ALUH → R	ALUL → LM(Al)
IWR	ALUH → R	ALUL → HM(Al)

Table 5: Les destinations des résultats.

paux: sources mémoire (MM,XM), sources mixtes de type mémoirefiles (FM,MF) et sources mixtes de type mémoire-feuillet de la POI (RM). Le tableau 4 donne les détails nécessaires.

-Destinations des résultats .
De la même manière que ci-dessus, nous regroupons les destinations principalement en trois types différents: sauvegardes mémoire (WM, IWM), sorties sur les ports (WP,IWP) et sauvegardes mixtes mémoire/-feuillet POI (WR, IWR). Nous les précisons dans le tableau 5.

Les mnémoniques ALUH et ALUL désignent respectivement l'ALU de poids fort et l'ALU de poids faible. R est la banque de registres de la POI, RP et LP sont les ports de sorties (gauche,droit) et LM et HM sont les deux plans de la mémoire. RQ et LQ désignent les deux files d'entrées de la partie traitement.

Nous pouvons définir plusieurs types d'instructions POS en combinant

les différents types de sources et de destinations.

Les opérations peuvent être du type:

var1 ← *var1 opération var2*

ou encore du type:

var1 ← *var2 opération var3*

où *var1*,*var2* et *var3* sont des variables données.

Pour certains cas particuliers, nous pouvons aussi bien définir des opérations du type:

var1 ← *var2 op1 var3 op2 (var4 op3 var5).*

Cela est possible avec la mise en cascade des opérateurs ALUH et ALUL.
Le champ Op spécifie une opération logique ou bien une opération de lecture d'indicateurs/résultats (ReadF), vers les registres adéquats de la zone R1, à réaliser dans la POI. L'opération POI est toujours de la forme R1 ← (R1) opération (R2),sauf bien sûr l' opération spéciale ReadF. R1 et R2 sont des zones quelconques du feuillet.

Dans le cas d'une opération ReadF, les champs AA et BB sont utilisés pour adresser les indicateurs/résultats à mémoriser respectivement dans R1(0) et R1(1). Les indicateurs concernés ont été donnés précédemment.

Le Champ condition (CC) indique le type de conditionnelle :

CC==0	Si le contenu du registre R1(0) est à 1 alors le contenu du champ LSB est remplacé par le contenu du champ MSB sinon l'opération inverse
CC==1	Si R1(0)=1 remplacer le contenu du champ MSB par un NOP Si R1(1)=1 remplacer le contenu du champ LSB par un NOP
CC==2	aucun changement sur les opérations indiquées initialement par les champs MSB et LSB

Nous pouvons donner quelques équivalents C d'instruction SMAL_X31 dans ce qui suit:

Soient: c1 et c2 les valeurs logiques respectives des contenus de R1(0) et R1(1).

1.A= A ops1 B ;

// C=C ops2 D ;
// opération OPI;
2.A= B ops1 C ;
// D= E ops2 F ;
// opération OPI;
3.A=(c1==1)? A ops1 B : A ops2 B;
// C=(c1==1)? C ops2 D : C ops1 D ;
// opération OPI;
4.if(c1==0) A= A ops1 B;
// if(c2==0) C =C ops2 D;
// opération OPI;

L'opération POI peut être n'importe quelle opération logique de deux zones du feuillet ou bien une opération de lecture d'indicateurs READF.

6 Exemples d'applications

Nous donnons brièvement dans cette partie quelques exemples d'applications très simples.

6.1 Réseaux de Hopfield

Nous prenons comme modèle de neurone un automate à seuil dont l'état, dans le cas simple, est le signe de la somme pondérée de ses entrées [13]. On définit deux états représentés par 1 et -1.

Un réseau de Hopfield est un réseau de N neurones complètement interconnecté. Il peut être représenté par un graphe orienté complet à N sommets.

Nous nous intéressons à la relaxation parallèle du réseau: ayant initialisé les états des différents neurones, le réseau évolue dynamiquement vers un état, stable ou chaotique. L'évaluation des différents états transitoires du réseau est telle que tous les neurones calculent simultanément leurs états à un instant donné. Donc, si $S_i(t)$ est l'état du neurone i à l'instant t,

$$S_i(t+1) = Signe(\sum_{j=1}^{n} S_j(t) w_{ij})$$

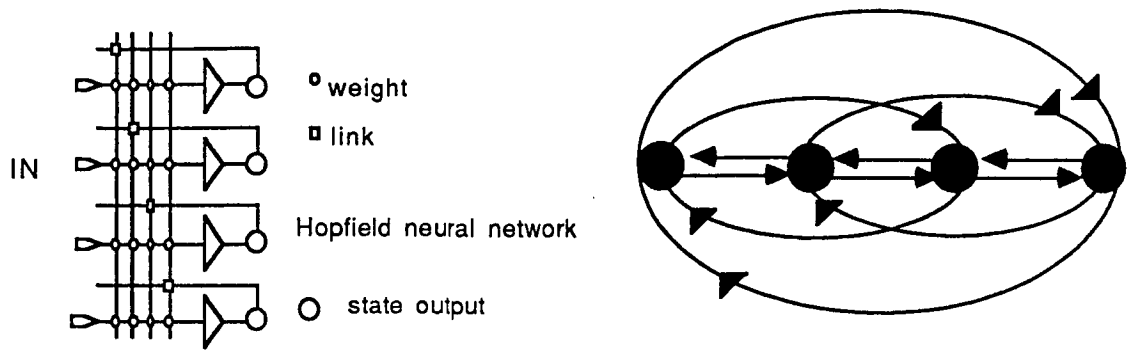

Figure 9: Réseau de Hopfield.

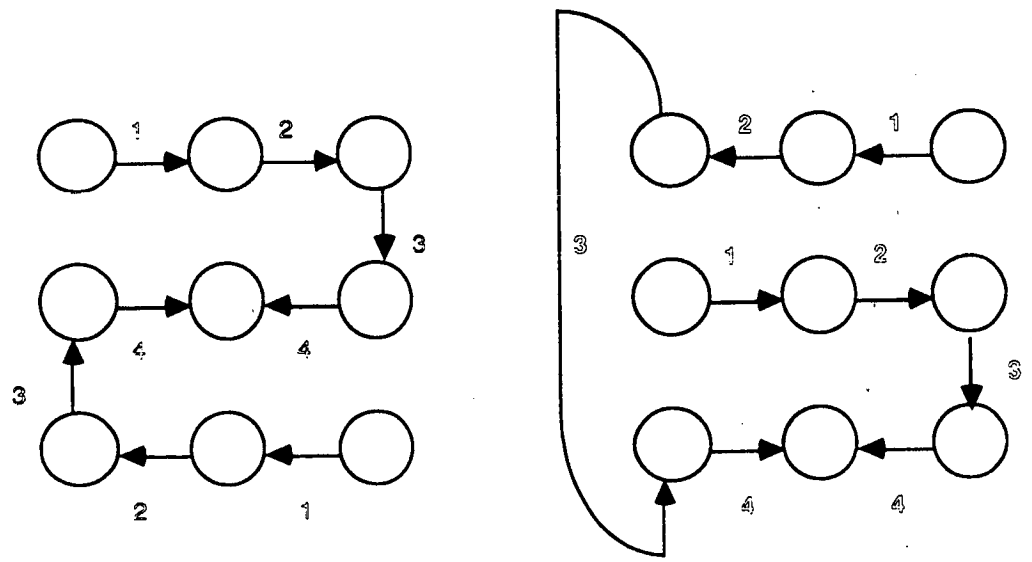

Figure 10: Schéma de calcul.

où w_{ij} est le poids de la connexion reliant la sortie du neurone j au neurone i.

Considérons une matrice torique de N PE. Pour émuler un réseau de Hopfield, nous pouvons dans un cas simple associer un neurone à un PE. Chaque PE mémorise les poids des connexions le reliant aux autres PE répartis convenablement sur les deux plans mémoires. Dans le cas où une convergence est certaine, deux bits du feuillet de la POI suffisent pour sauvegarder deux états successifs permettant sa détection.

Durant une itération d'évaluation, à chaque neurone i sont associés deux messages qui circulent selon un schéma synchrone bien défini de manière à balayer tout le réseau (figure 10). Chaque message balaie la moitié du réseau. Au passage par un neurone j, le message correspondant est augmenté ou diminué de la quantité w_{ij} selon l'état du bit du feuillet représentant l'état courant du neurone j. Le mode condition commune permet le choix de l'opération à effectuer.

A la fin du parcours, les deux messages arrivent simultanément au neurone i et l'on procède à l'évaluation de son nouvel état.

6.2 Jeu de la vie

Le jeu de la vie (Conway Game) est un exemple typique que l'on retrouve souvent dans les ouvrages traitant de la théorie des automates cellulaires.

Dans l'hypothèse de ce jeu on considère une population (matrice) de cellules donnée comme génération initiale. Durant les générations futures, une cellule peut prendre trois états distincts: Mort, survie, naissance. L'état pris par une cellule dépend de l'état des 8 cellules voisines.

Pour cet exemple nous avons choisis un réseau en Mesh où à chaque processeur est associée une cellule. Quatre transferts suffisent pour que chaque cellule prenne connaissance de l'état de ses huit voisins. Une décision, moyennant une instruction conditionnelle, est prise sur l'état futur de la cellule.

La théorie des automates cellulaires ne se limite pas à ce genre de problèmes. En effet, plusieurs démonstrations actuelles (ou non) prouvent l'utilité de cette théorie dans plusieurs domaines et notamment

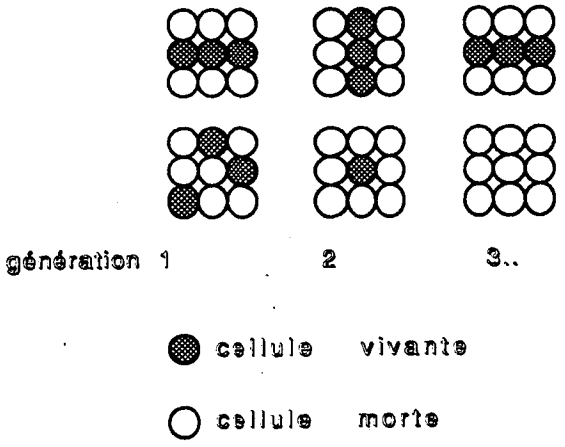

Figure 11: Jeu de la vie.

en modélisation de phénomènes physiques. Les réseaux de processeurs SIMD supportent aisément ce genre d'applications.

6.3 Algorithme de squelétisation

Une autre classe d'algorithmes parallèles facilement exécutables sur un réseau en Mesh sont les algorithmes de squelétisation. Ces algorithmes servent de primitive de base pour un système de reconnaissance de forme et plus particulièrement les caractères manuscrits.

Plusieurs algorithmes existent de nos jours. Le plus souvent, on retrouve des techniques dites de masques ou encore à LUT (look up table). Contrairement à cela, l'algorithme que nous avons choisi (décrit dans [6]), a été directement translaté en programme SMAL_X31 en utilisant la formulation de base.

Remarques:

Plusieurs autres applications peuvent être réalisées sous SMAL_X31: classe d'algorithmes liés au mesh et au tore.

Le problème d'affectation des processeurs, peut être simplement résolu par des techniques similaires à celle du pliage. De cette façon nous pouvons répondre à des problèmes utilisant des matrices de données (images ...) de dimensions supérieures à celle de la matrice physique du réseaux de processeurs.

7 Conclusion

Nous avons présenté dans ce qui a précédé les éléments de base d'une architecture SIMD, SMAL_X31, qui touchera à plusieurs applications. La parallélisation et la granularité fine sont les points importants que nous avons pris en compte durant la conception.

Nous espérons réaliser un prototype de réseau que l'on interfacera à une station SUN.

Le projet entre dans sa dernière phase d'étude.

Une compilation CMOS (ASIC) d'une puce de PEs sera réalisée, en utilisant les moyens matériels du laboratoire: Outils de conception ES2.

Références Bibliographiques.

1. T.Fountain ,"Processor Arrays, Architecture and Applications", Academic Press 1987.

2. K.Preston Jr and J.B.Duff,"Modern Cellular Automata-Theory and Applications",Plenum Press,New York 1984.

3. J.Kittler and J.B.Duff," Image Processing,System Architectures", John Willey & Sons inc 1987.

4. O.A.McBryan,"The Connection Machine:PDE solution on 65 536 processors", in Parallel Computing 9 (1988/89), North-Holland.

5. R.E.Cypher and J.L.C.Sanz, "The Hough transform has O(n) complexity on SIMD MxN mesh array architecture", Research Report, RJ 5709 (57688) 6/18/87, IBM research division.

6. Y.S.Chen and W.H.Hsu," A modified fast parallel algorithm for thinning digital patterns", in Pattern Recognition Letters7 (1988), North-Holland.

7. T.J.Fountain," Introducing Local Autonomy to Processor Arrays", in Machine Vision,Academic Press Inc 1988.

8. J.J.Little,"Integrating Vision Modules on a Fine-Grained Parallel Machine", in Machine Vision,Academic Press Inc 1988.

9. V.Cantoni and S.Levialdi," PAPIA: A case History", in Parallel Computer Vision, Academic Press Inc 1987.

10. J.Signorini," Interfacing Real-Time Data Streams For Neural Architectures",in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris ,1989.

11. B.Faure and G.Mazaré," A VLSI Asynchronous Cellular Architecture Dedicated to Multilayered Neural Networks",in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris ,1989.

12. F.Blayo," A Systolic Architecture Dedicated to Neural Networks", in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris ,1989.

13. L.Personnaz and all,"Toward a Neural Network Chip: A performance assessment and A Simple Example",in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris ,1989.

14. M.Duranton,J.Gobert and N.Mauduit," A Digital VLSI Module For Neural Networks", in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris, 1989.

15. H.P.Graf and Wayne Hubbard," VLSI Neural Network For Fast Pattern Matching", in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris, 1989.

16. A.Guerin,C.Jutten and J.Herault," Neurocomputer: CRASY A Cost-Effective Solution",in Neural Networks from Models to Applications by L.Personnaz and G.Dreyfus I.D.S.E.T, Paris, 1989.

17. Y.Ansade and all," Algorithms Dedicated To Network Of Asynchronous Cells", in Parallel Algorithms and Architectures edited by M.Cosnard, P.Quinton ..., North-Holland 1986.

18. A.Guerin et C.Jutten," VLSI et Machines Pour Le connexionisme", Nancy 1989.

19. W.D.Hillis, "The Connection Machine", The MIT Press 1985.

20. K.S.Eo and C.M.Kyung,"A Hardware Accelerator for Two Dimentionnal Image Analysis", in INTEGRATION the VLSI journal, North Holland Septembre 1988.

21. J.L.Potter,"The Massively Parallel Processor", the MIT press 1985.

22. "The BLITZEN Project", recueil de certaines notes lors de la conférence donnée par E.W.Davis a l'IRISA, RENNES, 23 juin 1989.

23. S.Y.Kung, "VLSI Array Processors", Prentice Hall 1988.

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