Seminar "Innovative Prozessor-Architekturen"

AG Hartenstein
FB Informatik, Universität Kaiserslautern
Wintersemester 1987/88

Teil 3

Vortragsliste:

	Thema	Vortragender	Datum
1.:	Transputer	A. Schaffer	13.1.88
2.:	Coprozessoren und intelligente Controller	H.-J. Franz	20.1.88
3.:	Connection Machine	H. Reinig	20.1.88
4.:	Data Flow Computers	T. Blüthner	27.1.88
5.:	RISC Architekturen	J. Kreiselmaier	27.1.88
6.:	Systolic Arrays / WARP	F. Faber	3.2.88
7.:	Map Oriented Machine	A. Hirschbiel/M. Weber	3.2.88

Seminar

"Innovative Prozessor Architekturen"

Vortrag

Systolische Arrays und WARP

von Fred Faber

U E B E R S I C H T

A) S Y S T O L I S C H E A R R A Y S

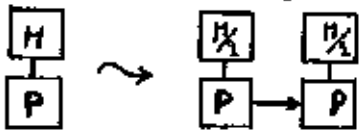
B) W A R P

I D E E U N D D E F I N I T I O N

* PIPELINING

Das Wort 'pipeline' lässt sich hier wahrscheinlich am besten mit 'Fliessband' übersetzen.Beim pipelining geht es im Prinzip darum,die Arbeit eines Prozessors auf eine Kette von Prozessoren zu verteilen.Jeder dieser Prozessoren führt nur einen kleinen Teil der Gesamtarbeit durch und übergibt das Resultat dem nächsten Prozessor in der Kette. Somit kann man einen relativ hohen Grad an Parallelität erreichen falls alle Prozessoren gleichmässig ausgelastet werden.

-Beispiel:Ersetzen eines Prozessors durch eine '2-Prozessoren-Pipeline'.



Man kann so eine Verdreifachung der Rechengeschwindigkeit erreichen,da man zwei Effekte ausnutzt:

1) 2 Prozessoren ⇒ grössere Rechengeschwindigkeit
2) Aufteilung des Speichers ⇒ schnellerer Speicherzugriff

-Nachteile:

1) Die Aufteilung einer Aufgabe in mehrere Teilaufgaben ist nicht immer möglich und nicht immer effizient.
2) Pipelining erfordert eine sehr starre Kommunikationsdisziplin zwischen den Prozessoren der Kette.

-Entwurfskosten:

Da die Hardwarekosten immer mehr fallen,werden die Designkosten immer wichtiger.Man kann also beim Pipelining die Kosten niedrig halten,indem man nur wenige Typen gleichartiger Prozessoren benutzt.
Gerade weil man nur ein paar Typen einfacher Zellen benutzt ist es möglich allgemeine Richtlinien für den Entwurf von Systemen mit Pipelining zu entwerfen.
So wird es möglich sein Wissen weiterzuleiten und die gleichen Fehler beim Entwurf von Systemen zu vermeiden.

-Anwendungsgebiete:

Da es nicht möglich ist alle Probleme für parallele Verarbeitung umzuformen,werden solche 'pipelines' hauptsächlich zur Entlastung herkömmlicher Rechner von besonders rechenaufwendigen Aufgaben verwendet.

* SYSTOLISCHES SYSTEM

-Ein systolisches System ist ein Netzwerk von einfachen gleichartigen Prozessoren,in denen die Daten rhytmisch und gleichmässig zirkulieren.
(Zirkulation:Vom Speicher durch die Prozessoren zurück in den Speicher.)

-'systole' : Zusammenziehung des Herzmuskels
(Hier wird ein systolisches System mit dem Blutkreislauf verglichen: Herz ⇒ Körpertiele ⇒ Herz)

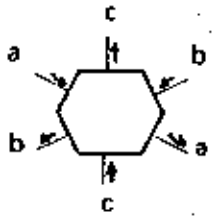
-Das Konzept der systolischen Architektur ist eine allgemeine Methode Berechnungen höherer Ebene in der Hardware zu implementieren. ⇝ VLSI-Algorithmen

B E I S P I E L : MATRIZEN-MULTIPLIKATION

* Gegeben : $A=(a_{ij})$ und $B=(b_{ij})$ zwei n x n Bandmatrizen mit den respektiven Bandbreiten w_1 und w_2.

* Behauptung : Das Produkt $C=(c_{ij}) = A \times B$ kann in $3n + \min(w_1,w_2)$ Zeiteinheiten berechnet werden, wenn man $w_1 \cdot w_2$ hexagonal angeordnete IPS-Prozessoren benutzt.
(Zeitaufwand für einen 'normalen' Rechner mit Standartalgorithmus : $O(w_1 \cdot w_2 \cdot n)$)

* Aufbau einer benutzten Zelle:



a ← a
b ← b
c ← c + ab

Inner Product Step Processor

-In einem Zeitzyklus werden drei Aktionen durchgeführt :
1) Eingang a wird an Ausgang a übergeben.
2) Eingang b wird an Ausgang b übergeben.
3) Zum Eingang c wird das Produkt a.b dazuaddiert und an Ausgang c übergeben.

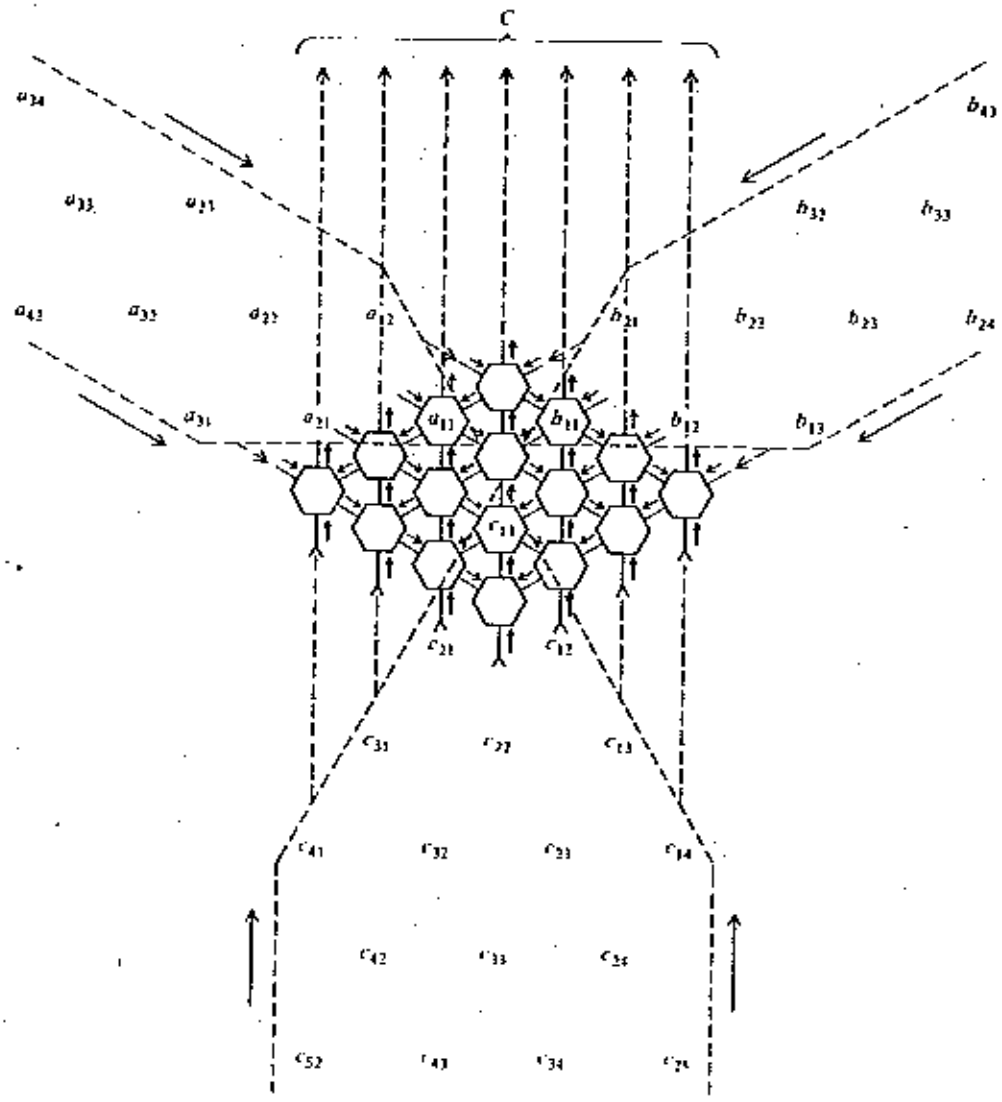
-Wieso man gerade eine solche Zelle benutzt wird leicht ersichtlich, wenn man die Rekurrenzgleichung für die Matrizenmultiplikation betrachtet:

$$\begin{cases} c_{ij}^{(1)} = 0 & \text{(Initialisierung)} \\ c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik} \cdot b_{ik} & \text{(Berechnung)} \\ c_{ij} = c_{ij}^{(n+1)} & \text{(Ausgabe)} \end{cases}$$

-Bandmatrizenmultiplikation:

$$\begin{bmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & \ddots & & \\ 0 & & & & \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & \\ & b_{32} & b_{33} & b_{34} & b_{35} \\ & & b_{43} & \ddots & \\ 0 & & & & \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & \ddots & & \\ 0 & & & & \end{bmatrix}$$

A B C

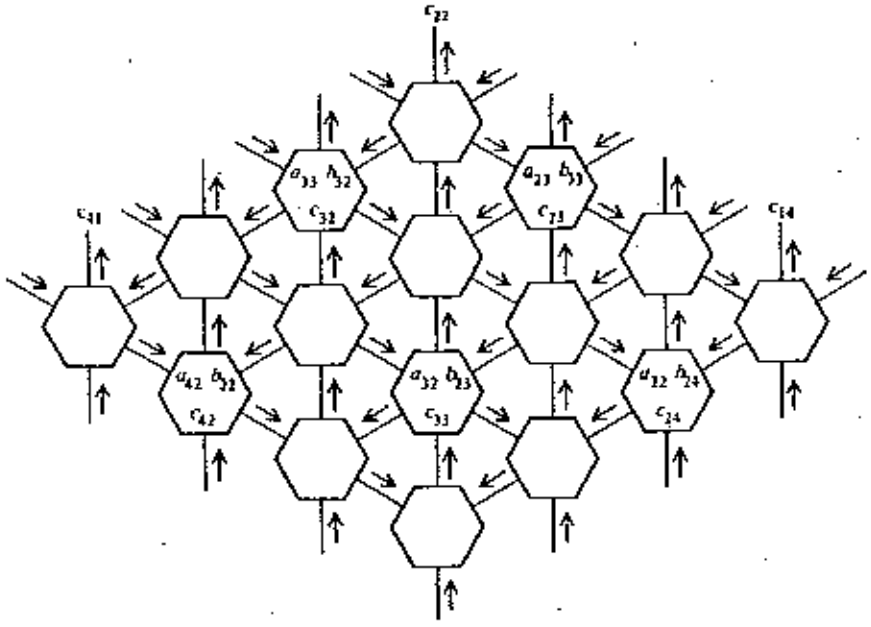


Hexagonales Array zur Berechnung der
Band-Matrizen-Multiplikation

* Die Pfeile geben den Datenfluss an: Die Elemente der drei Matrizen A,B,C wändern synchron in drei verschiedenen Richtungen durch das Array. Jedes c_{ij} wird mit dem Wert 0 initialisiert, wenn es in das Array^{ij} eintritt. (Für die allgemeinere Berechnung von C=AB+D sollen alle Werte c_{ij} mit dem zugehörigen Wert d_{ij} initialisiert werden.)
Man kann leicht nachprüfen, dass jedes c_{ij} den richtigen Termen a.b begegnet, ehe es das Array 'verlässt'.
Zur Veranschaulichung kann man auf der nächsten Seite drei Schritte während der Berechnung von c_{33} verfolgen.

* Bemerkung :

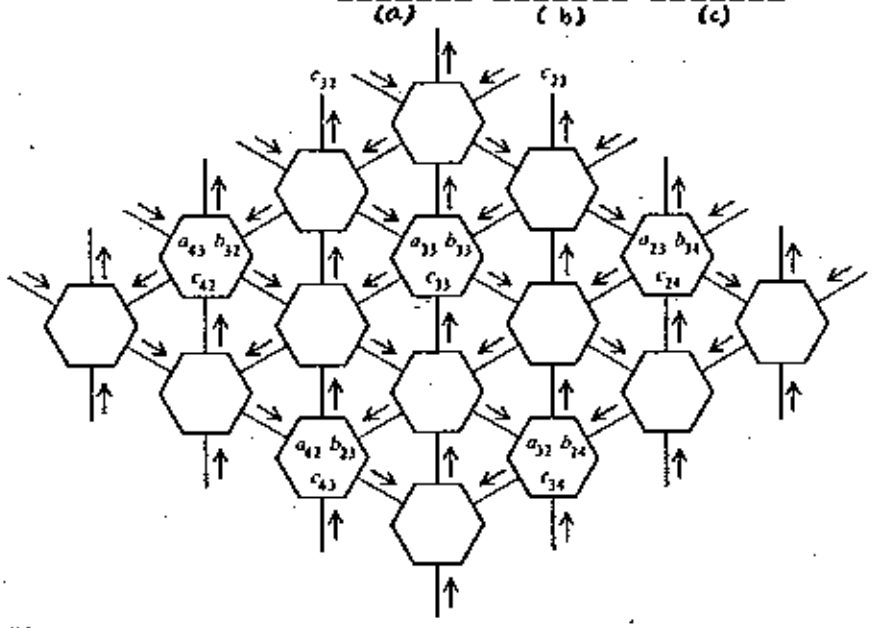
Wie man sieht, ist von jeweils drei aufeinanderfolgenden Zellen in jeder Reihe oder Kollonne nur eine Zelle aktiv. Bei der Multiplikation von zwei Bandmatrizen ist es also möglich ein Drittel der $w_1 . w_2$ Zellen gleichzeitig auszulasten.
(Man kann das Array auch ganz auslasten, indem man drei Matrizenmultiplikationen gleichzeitig durchführt.)



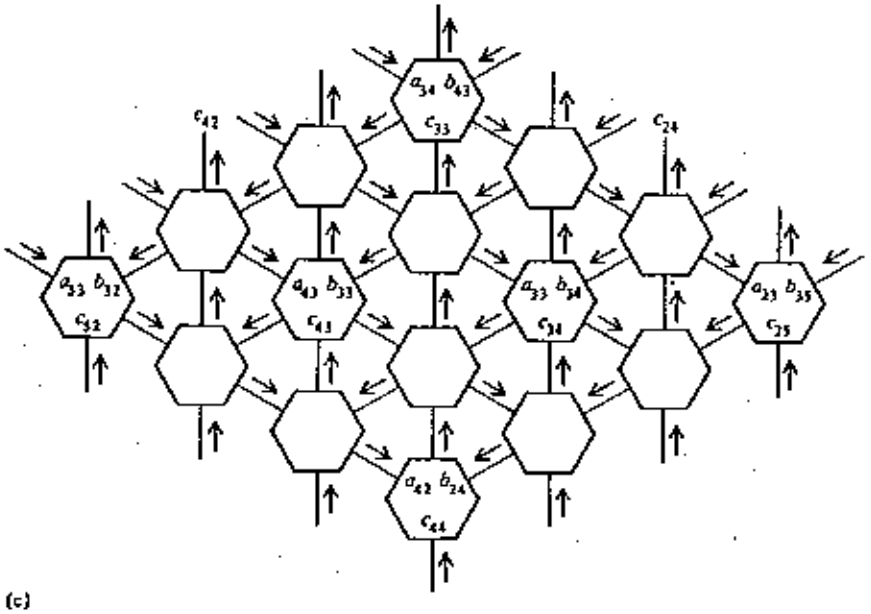
(a)

$$c_{33}=a_{31}\cdot b_{13}+a_{32}\cdot b_{23}+a_{33}\cdot b_{33}+a_{34}\cdot b_{43}$$

(a) (b) (c)



(b)



(c)

A L L G E M E I N E S T R U K T U R

1) FUNKTION DES EINZELNEN PROZESSORS

-Ein Prozessor soll eine Fûktion durchführen,die konstante Zeit braucht (z.B. Produkt/Vergleich&Vertauschen/Daten durchschleusen)

-Ein Prozessor soll einen geringen Speicher- und Platzbedarf haben,damit man möglichst viele Rechenzellen auf einem Chip unterbringen kann.

-Die einzelnen Zellen sollen uniform sein.

-Ein Array soll so wenig wie möglich I/O-Prozessoren haben, damit die Anzahl der Pins eines Chips begrenzt bleibt. (Man nennt I/O-Prozessoren die Prozessoren,die sich am Rand des Arrays befinden und die zur Ein- und Ausgabe der Daten benötigt werden.)

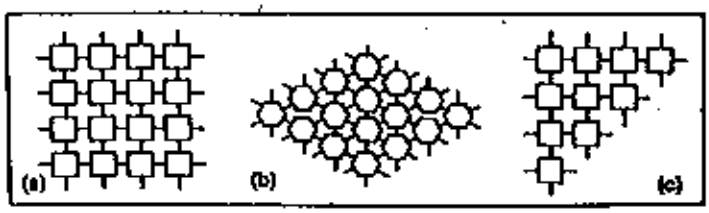
2) KOMMUNIKATIONS-GEOMETRIE

(z.B. Matr.-Mult.:hexagonales Array-Nẗzwerk)

-Chip-Fläche,Zeit und Stromverbrauch hängen zum grossen Teil von der Kommunikations-Geometrie ab.
Man soll diese also einfach und regelmässig gestalten, um eine grösstmögliche Dichte und ein modulares Design zu erreichen.

-Es gibt hauptsächlich drei einfache und regelmässige Strukturen :

1) Quadrat
2) Hexagon
3) gleichschenkliges Dreieck



3) DATENFLUSS

-In systolischen Arrays wird Pipelining benutzt.

-Der Datenfluss ist normalerweise synchron(obschon ein asynchroner Datenfluss auch realisierbar ist.)

-Der Datenfluss in systolischen Arrays wird durch drei Grössen charakterisiert:

1) Richtung
2) Geschwindigkeit
3) Timing

·Die Daten können in verschiedene Richtungen mit verschiedenen Geschwindigkeiten fliessen;das Timing muss dann dafür sorgen,dass die Daten zur richtigen Zeit am richtigen Ort sind.
(z.B. Matr.-Mult.:3Datenströme (A,B,C) / 3 verschiedene Richtungen / gleiche Geschwindigkeit / die Daten in jeder Diagonalen des Arrays sind jeweils 3Zeiteinheiten auseinander.)

-Der Datenfluss soll einfach,regelmässig und uniform sein, um die Kontrollkomplexität niedrig zu halten.

E I N - D I M E N S I O N A L E L I N E A R E A R R A Y S

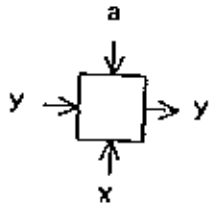
-Lineare Arrays stellen die einfachste Form eines syst. Arrays dar und bilden die Basis für andere Kommunikationsgeometrien.Sie sind zugleich die natürlichste Struktur für Pipeline-Berechnungen.

-Die Daten in linearen Arrays können in eine oder in zwei Richtungen gleichzeitig fliessen (abhängig vom Algorithmus)

1) EIN-WEG PIPELINE

-BEISPIEL:Matrix-Vektor Multiplikation

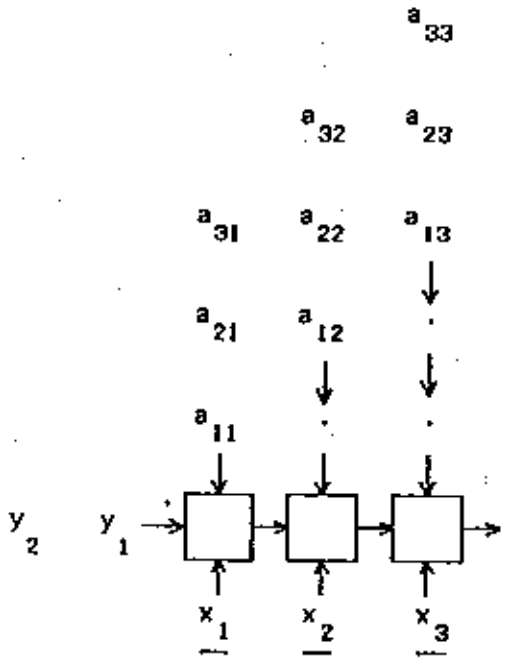
-Benutzte Prozessoren:



$y \leftarrow y + ax$

-Problemstellung und Lösung:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$



-Die a_{ij}'s und y_i's bewegen sich synchron.

- $x_{1,2,3}$ bleiben konstant und können in der jeweiligen Zelle gespeichert bleiben.
- y_1 sammelt seine 3 Einzelresultate im 1.,2.,3.Schritt
- y_2 sammelt seine 3 Einzelresultate im 2.,3.,4.Schritt

≡ BEMERKUNGEN :

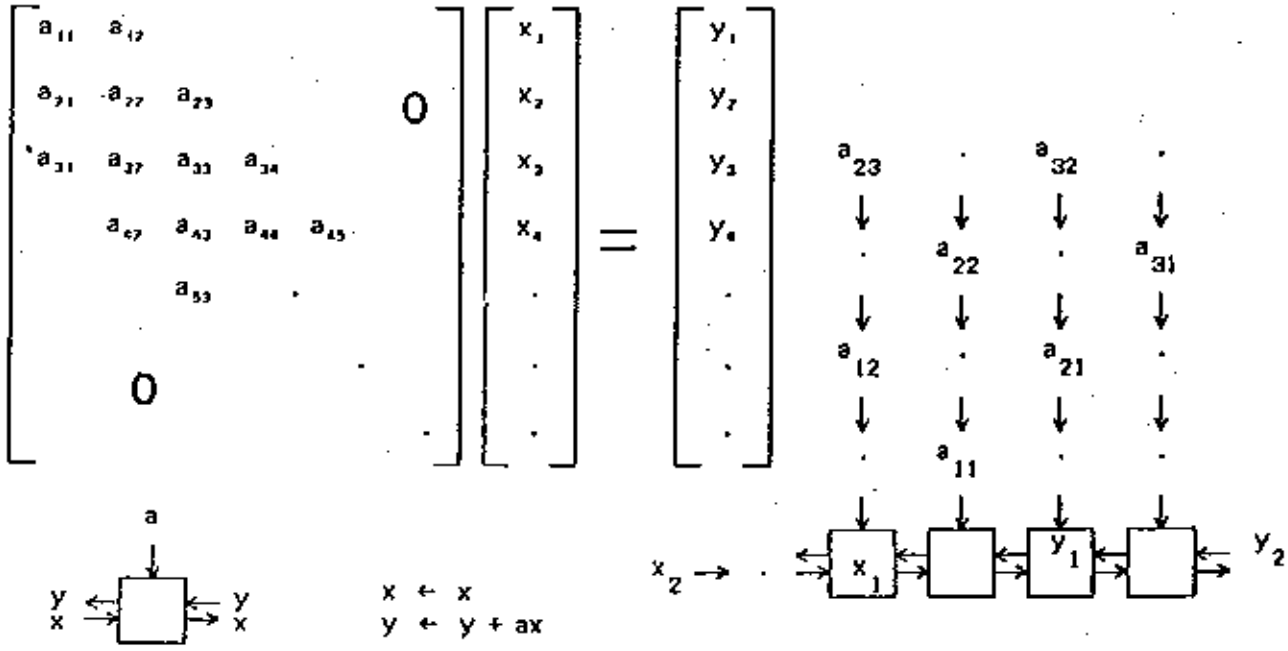
-Hier können Matrix und Resultate unbegrenzt lang sein, während die Länge des Vektors festgelegt sein muss.

-Man kann die Funktion des Arrays ändern, indem man die Funktion der einzelnen Zellen ändert: $y \leftarrow F(a,x,y)$

-anderes Beispiel: carry-pipelining für Addierer oder arithmetische Einheit mit pipeline-Arch.

2) ZWEI-WEGE PIPELINE

-BEISPIEL: Bandmatrix-Vektor Multiplikation



-HIER: Bandmatrix, x und y können verschiedene Länge haben
⇒ alle drei Grössen müssen während der Berechnung bewegt werden.
- x_i und y_i werden in entgegengesetzter Richtung bewegt. Damit aber alle x_i allen y_i begegnen können, müssen alle Elemente eine Zeiteinheit voneinander getrennt sein (Timing).
- Alle y_i's werden mit 0 initialisiert und sammeln ihre Zwischenresultate, bis sie den linken Prozessor wieder verlassen.
- Theoretisch ist immer nur die Hälfte des Arrays ausgelastet; damit man keine Zeit verschwendet, kann man mehrere Matrizen gleichzeitig berechnen.

-SCHLUSSFOLGERUNG:
Ist die Länge aller Eingabe- und Ausgabedaten grösser als die Länge der Pipeline, so müssen alle Eingaben und Zwischenresultate während der Berechnung bewegt werden.

⇒ Datenfluss in 2 Richtungen gleichzeitig

-Andere Beispiele: -kartesisches Produkt
-diskrete Fourier-Transformation

A U S W E R T U N G VON R E K U R R E N Z - RELATIONEN

-Rekurrenz-Problem der k-ten Ordnung:
Seien $x_0, x_{-1}, x_{-2}, \ldots, x_{-k+1}$

berechne $x_1, x_2, \ldots$ mit $x_i = R_i(x_{i-1}; \ldots; x_{i-k})$ für $i>0$

-Auch wenn ein solches Problem sich am besten für sequentielle Berechnungsmethoden zu eignen scheint, gilt die Behauptung :

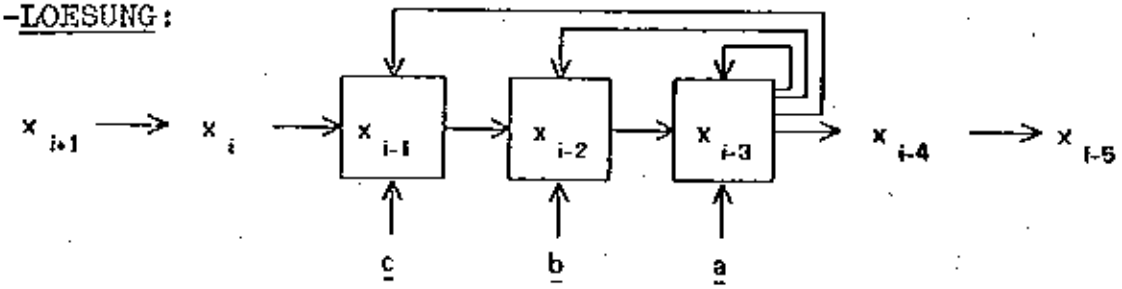
> Ein Rekurrenz-Problem k-ter Ordnung kann in Echtzeit mit k linear verbundenen Prozessoren berechnet werden.

-BEISPIEL: $x_i = a.x_{i-1} + b.x_{i-2} + c.x_{i-3} + d$

Dieses Problem hat dritte Ordnung und kann also mit drei linear verbundenen Prozessoren gelöst werden.

-Für die Lösung des Problems werden Rückkopplungen benötigt, da jedes neue Resultat für weitere Berechnungen später benutzt wird.

-LOESUNG:



-Nur der rechte Prozessor benötigt mehr als einen Ausgang. Alle x_i werden mit d initialisiert und sammeln die Zwischenresultate während sie das Array durchlaufen:

Zeiteinheit:	0	1	2	3
x_i =	d	$+c.x_{i-3}$	$+b.x_{i-2}$	$+a.x_{i-1}$

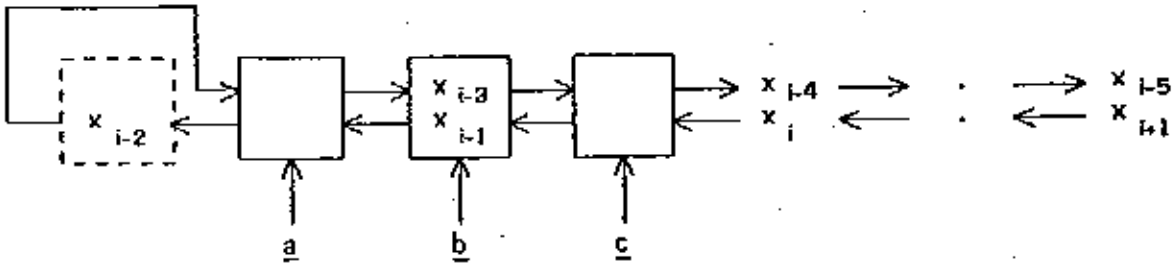
-In der vierten Zeiteinheit wird das Resultat ausgegeben.

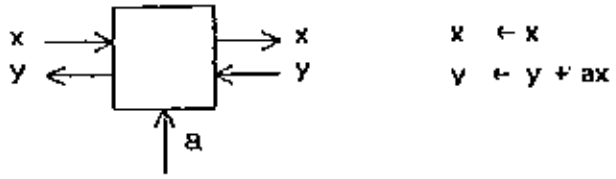
-PROBLEME:

- Der Datenfluss ist unregelmässig.
- Der Aufbau ist nicht modular: Wenn man eine Rekurrenzrelation höherer Ordnung auswerten will, muss man nicht nur eine Zelle anhängen, sondern man muss auch die rechte Zelle durch eine andere ersetzen, die die entsprechende Anzahl Ausgänge hat.

-LOESUNG:

Benutzung einer 2-Wege-Pipeline mit regelmässigem Datenfluss:



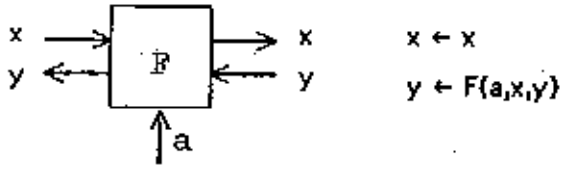


-Jeder Prozessor kann rechnen und Daten durchschieben. Der zusätzliche Prozessor (links) kann nur Daten durchschieben und wird als Zeitverzögerung benutzt (Timing).
-Genau wie vorher sammelt x_i seine Bestandteile zu den Zeitpunkten t=1,2,3. Bei t=4,5,6,7 wird x_i durchgeschoben und zur Berechnung der Nachfolger verwendet. Die Ausgabe erfolgt bei t=8. Danach erfolgt bei jeder zweiten Zeiteinheit eine Ausgabe.

-GENERALISIERUNG:

Zur Lösung allgemeinerer Rekurrenzrelationen genügt es, die Prozessoren entsprechende Funktionen berechnen zu lassen.

$$x_i = F_1(a_{i-1}, x_{i-1}, F_2(b_{i-2}, x_{i-2}, F_3(c_{i-3}, x_{i-3}, d_{i-4})))$$



-BEISPIEL:

$$x_i = 3 \cdot x_{i-1}^2 + x_{i-2} \cdot \sin(x_{i-3}+4)$$

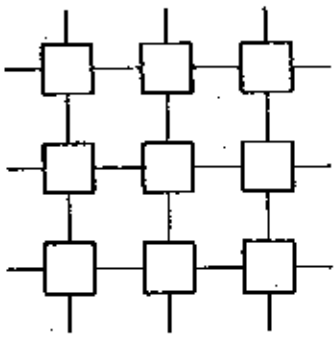
$$\begin{cases} F_3(x,y,z) = \sin(y+z) \\ F_2(x,y,z) = y \cdot z \\ F_1(x,y,z) = 3 \cdot y^2 + z \end{cases}$$

-SCHLUSSFOLGERUNG:

2-Wege-Pipelining ist ein nützliches Hilfsmittel, da es hilft störende Rückkopplungen zu entfernen.

Z W E I D I M E N S I O N A L E A R R A Y S

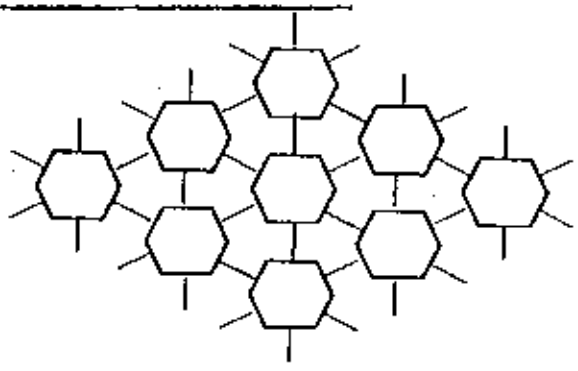
1) QUADRATISCHE ARRAYS



-Quadratische Arrays wurden wahrscheinlich als erstes betrachtet,als es darum ging eine geeignete Kommunikationsstruktur für parallele Prozesse zu finden.
Von Neumann hat sich 1966 bereits mit ähnlichen Strukturen beschäftigt.
-Quadratische Arrays bieten die natürlichste Struktur zur Lösung von Matrizenproblemen;trotzdem ist es effizienter hexagonale Arrays für diese Aufgaben zu benutzen.
-Anwendungsbeispiele:

- Graphenprobleme und Adjazenzmatrizen
- Pattern Recognition
- Schnelle Fourier-Transformation
- Sortieralgorithmen(Mischsort)

2) HEXAGONALE ARRAYS



-Hexagonale Arrays haben die Eigenschaft der Symmetrie in drei Richtungen:
-Ein Resultat und zwei Eingaben können zur gleichen Zeit ein- und ausgelesen werden.(siehe Matr.-Mult.)
-Vorteil gegenüber quadratischen Arrays:
Separate Ein- und Ausgabephasen bleiben erspart.
-Anwendungsbeispiele:

- Matrizen-multiplikation
- LU-Zerlegung

BEMERKUNGEN

-Wenn eine aufwendige Aufgabe auf einem kleinen Netzwerk gelöst werden soll,kann man entweder die Aufgabe in Teilaufgaben zerlegen oder den VLSI-Algorithmus in Einzelteile zerlegen.Man muss dann hauptsächlich darauf achten, dass man die Ein- und Ausgabe beschränkt,oder die I/O-Bandbreite so gross macht,dass die Geschwindigkeit wegen der Zerlegung nicht herabgesetzt wird.

-Mit multi-direktionellem Datenfluss kann man komplexe Berechnungen durchführen,ohne die Einfachheit und Regelmässigkeit des Konzepts zu zerstören.Ausserdem erspart man sich so separate Ein- und Ausgabephasen.

-Man kann davon ausgehen,dass hexagonale Verbindungen den quadratischen Verbindungen überlegen sind,weil hexagonale Verbindungen einen Datenfluss in mehr Richtungen erlauben, und trotzdem ungefähr den gleichen Implementierungsaufwand haben,wie die quadratischen Verbindungen.

-Folgende Probleme müssen noch gelöst werden:

- Es gibt noch keine Notation zur Spezifikation der Geometrie des Datenflusses.
- Es gibt noch keine Methode Korrektheitsbeweise von Algorithmen in solchen Netzwerken durchzuführen.
- Es gibt noch keine allgemeine Richtlinien für den Entwurf von VLSI-Algorithmen.

-Es besteht ein enger Zusammenhang zwischen der definierenden Rekurrenzrelation eines Problems und dem VLSI-Algorithmus,der das Problem lösen soll.Man wird versuchen, die Implementierung von Rekurrenzrelationen als VLSI-Algorithmen in Zukunft grösstenteils automatisch durchführen zu lassen.

-Wir haben gesehen,dass man immer versucht,ein Array aus 'wenigen Typen einfacher Zellen' aufzubauen; was aber eine 'einfache Zelle' ist,muss von Fall zu Fall unterschieden werden:•Bei den bisherigen Beispielen wollte man immer möglichst viele Zellen auf einem Chip unterbringen.Daher besteht eine Zelle nur aus einfachen logischen Schaltungen mit ein paar Wörtern Speicher.

•Es gibt aber auch Arrays,wo eine einzige Zelle aus einem ganzen Board besteht (siehe WARP). Dann braucht man eine leistungsfähige Recheneinheit und ein paar tausend Wörter Speicher.

Man muss also bei der Auswahl der Grösse einer Zelle immer einen Kompromiss zwischen Einfachheit und Flexibilität machen.

V O R T E I L E

-Das Design benutzt alle Eingaben mehrfach.

-Das Design benutzt viele Prozessoren gleichzeitig.

-Für ein Problem kann es ein- oder zweidimensionale Lösungen geben.Man hat also die Auswahl und kann das Design der Aufgabe anpassen. Ist zum Beispiel die Speichergeschwindigkeit grösser als die Ein- Ausgabekapazität,so benutzt man am besten ein zwei-dimensionales Array,da man so alle Ausgänge voll auslasten und somit das System besser ausnutzen kann.

-Der Datenfluss ist einfach und regelmässig,deswegen ist es möglich:
- Lange Verbindungswege für die Datenkommunikation zu vermeiden.
- Nur eine einzige globale Verbindung zu benutzen:den Taktgeber.
- Adressangaben und Adressberechnungen für Daten und Programme einzusparen.

-Leistung und Kosten erhöhen sich proportional (falls das Problem gross genug ist.).Andere parallele Systeme sind selten günstig für mehr als nur ein paar einzelne Prozessoren.

-Systolische Arrays sind einfach im Gebrauch:
Im Prinzip braucht man nur Eingabedaten einzulesen und dann das Resultat entgegenzunehmen.

U E B E R G A N G

-Aktueller Gebrauch: Implementierung von Spezialrechnern für Einzelaufgaben.

- Ziel : Benutzung von systolischen Prozessoren in generellen Rechnern um diese von rechenaufwendigen Aufgaben zu entlasten.

- Forschung :
1)Integration leistungsfähiger Prozessoren in einem kompakten System,das man zur Benutzung nur auf dieser Ebene betrachten muss.
2)Spezifizierung und Entwurf einfacher,programmierbarer Zellen.Solche Arrays kann man dann ohne viel Aufwand für viele verschiedene Aufgabe benutzen.

W A R P

W A R P E I N L E I T U N G

1) DEFINITION

-Warp : Kette

-Warp : Computer auf Basis von systolischen Arrays, der für rechenintensive Anwendungen entworfen wurde.

2) AUFBAU

-lineares systolisches Array
-10 oder mehr identische Zellen
-10 MFLOPS/Zelle ==> Spitze von 100 MFLOPS
(MFLOPS = Million Floating-Point Operations per Second)
-Das Warp-Array wird an einen Host-Rechner angeschlossen, der mit UNIX betrieben wird.
-Das Warp-Array wird in der höheren Programmiersprache W2 programmiert.
-Das Warp System besitzt einen interaktiven,programmierbaren Kommandointerpreter : Warp-Shell.
Ein Warp-Programm wird ähnlich wie eine Prozedur aufgerufen : Die Shell ruft das Warp Programm auf und schickt Ein- und Ausgabedaten zwischen Benutzerprogramm und Warp-Programm hin und her.

3) PROJEKT

-1984 : Start des Warp-Projekts

-Juni 1985 : Der erste zwei-Zellen Prototyp wird an der Carnegie Mellon Univerität hergestellt.

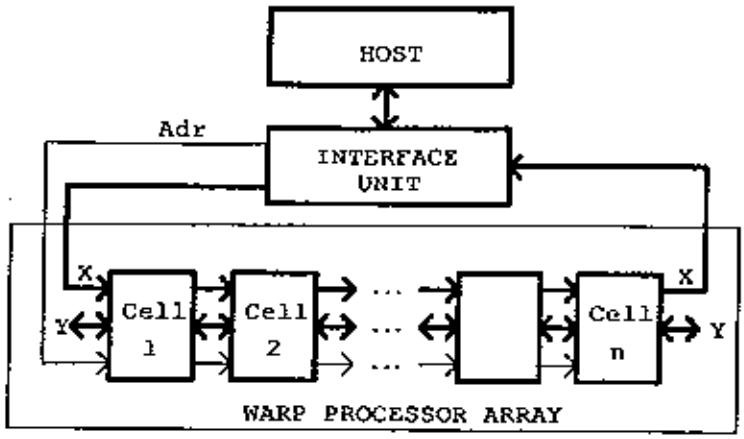
-Februar 1986 : Zwei zehn-Zellen Prototypen werden hergestellt;einer bei Honeywell und einer bei General Electric. Diese Prototypen sind noch auf wire-wrapped boards gefertigt.

-April 1987 : Eine verbesserte 10-Zellen Version wird hergestellt und von General Electric in Produktion genommen.(Preis:Ø350000)

4) ANWENDUNGEN

-Bild-und Signalverarbeitung.
-Grobes Sehen für die Steuerung von Roboter-Fahrzeugen.
-Wissenschaftliche Berechnungen.
-Algorithmen für Graphen.

W A R P S Y S T E M U E B E R B L I C K



1) AUFGABEN

- -Warp Array : führt rechenaufwendige Routinen durch.
- -Interface Unit : führt den Datentransfer zwischen Warp-Array und Host durch.
  generiert Adressen und Steuersignale für das Warp-Array.
- - Host : liefert Eingabedaten und erhält Resultate.
  führt Programmteile aus, die nicht im Warp-Array berechnet werden können.

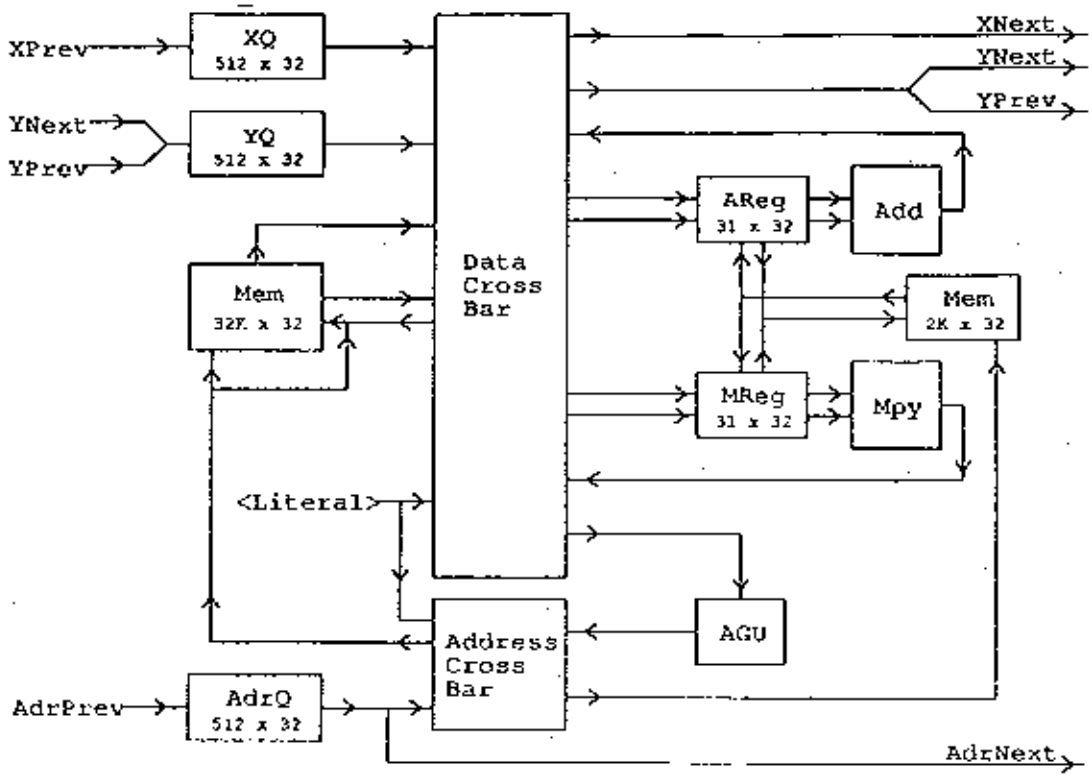
2) VERBINDUNGEN

- Zwei Kommunikationskanäle (X & Y) für Daten.
- Ein Adresskanal für lokale Zellenspeicheradressen und Kontrollsignale.

3) Warp-Zellen IMPLEMENTIERUNG

-Die einzelnen Warp-Zellen sind durch Mikro-Rechner mit 8k Programmspeicher für 272-Bit-Befehle implementiert.

W A R P Z E L L E : A R C H I T E K T U R



- Jede Warp-Zelle enthält einen 32-Bit Multiplizierer und Addierer mit lokalem 2k Wortspeicher für temporäre und stationäre Datenmengen. Vor dem Addierer und dem Multiplizierer befindet sich jeweils ein 31 x 32 Bit Register als Puffer.
- Jeder Kommunikations- und Adresskanal ist durch eine 512 Worte lange Warteschlange gepuffert.
- Auf jedem Board befindet sich ein weiterer 32k Wortspeicher.
- Auf jedem Board (der Produktionsmaschine) befindet sich eine Adress-Generierungs-Einheit (AGU).
  Also können die Adressen für die Speicher entweder in der AGU berechnet werden oder vom Adresskanal übernommen werden.
- Alle diese Einheiten werden durch die Crossbar (Querbalken) miteinander verbunden.
- Dimensionen eines Boards : 15" x 17"

1) CROSSBAR

In programmierbaren systolischen Zellen kann die interne Datenbreite oft zu Engpässen führen:
In diesem Fall enthält die Recheneinheit 4 Eingänge und 2 Ausgänge. Die Recheneinheit wird von zusätzlichen Speicherblocks unterstützt ==> insgesamt: {6 Eingänge / 8 Ausgänge
In jedem Zeitzyklus muss die Möglichkeit bestehen, jeden der 6 Eingänge mit jedem der 8 Ausgänge zu verbinden. Durch die Crossbar kann dieses Problem ohne die Konflikte gelöst werden, die bei der Benutzung von mehrfach verwendeten Bussen auftreten würden.

2) DATENSPEICHER

- Lokaler Datenspeicher: 32k-Wortspeicher
  - enthält Eingabedaten oder Zwischenresultate
  - kann in jedem Zyklus (200ns) beschrieben und gelesen worden.
- Register-Speicher: 2 x 31 Worte
  - Datenpuffer mit 6 Ports
  - kann in jedem Zyklus {2 Daten vom Crossbar annehmen / 2 Daten an den Crossbar weiterleiten.
- Zwischenspeicher: 2k Worte
  - sorgt für schnellen Transfer zwischen den Registern
  - enthält Fliesspunktkonstanten, kleine arrays, usw.
  - bietet einen grossen Vorteil beim Rechnen mit lokalen Daten:
    Die Recheneinheit kann zwar in einem Zyklus 4 Daten einlesen und ausgeben, aber der lokale Datenspeicher kann nur 1 read und 1 write durchführen. Durch Verwendung des Zwischenspeichers kann man diesen Fehler grösstenteils beheben.

3) ADRESS-GENERIERUNG

- AGU : Rechenwerk für Integer-Zahlen mit 64 Registern
  - kann eine read und eine write-Adresse pro Zyklus berechnen.
  - kann bedingte Verzweigungen durchführen.
- Auf dem Prototyp-Warp war die AGU nur einmal auf dem IU-Board implementiert, da kein Platz mehr auf den Array-Boards war; das brachte folgende Nachteile mit sich :
  (1) Es kann nur eine Adress-Menge effektiv im IU berechnet werden ==> alle Zellen müssen das gleiche Programm ausführen.
  ==> Bedingungen und Loops werden auch bereits vorher vom IU berechnet.
  (2) Die Möglichkeiten für Pipelining werden beschränkt, da der Adress-Puffer nur eine beschränkte Länge hat und zur gleichen Zeit Adressen und Kontrollsignale aufnehmen muss.
- Auf der Produktionsmaschine ist die AGU auf jedem Board enthalten; so kann der Compiler die Adressen lokal generieren. (Für Spezialfälle bleibt die AGU auch auf dem IU-Board enthalten.)

K O M M U N I K A T I O N ZWISCHEN DEN ZELLEN

- 2 Datenkanäle
- 1 Adress- und Kontrollkanal
- 3 Puffer : Warteschlangen für 512 Worte.
  Diese Puffer sind gerade gross genug für eine oder zwei Zeilen eines Bildes,deren Auflösung oft 512x512 oder 256x256 punkte ist. ==>Das Array ist gut geeignet für Bildverarbeitung.

-Kontrolle der Kanäle durch Hardware:

* Wenn eine Zelle versucht Daten aus einem leeren Puffer zu lesen,wird die Zelle blockiert,bis Daten vorhanden sind.
* Wenn eine Zelle versucht Daten in den vollen Puffer einer benachbarten Zelle zu schreiben,wird die schreibende Zelle blockiert,bis Daten aus dem Puffer entnommen werden.
* Blockierung : Alle rechnenden Einheiten einer Zelle werden blockiert,nur die Puffer können noch Daten empfangen,weil die Zelle sonst nicht entblockt werden könnte.
* Nur die betroffene Zelle wird blockiert,alle anderen Zellen arbeiten normal weiter.
* Wegen dieser Blockierungs-Strategie werden zwei Takt-Generatoren benötigt:-einer für die rechnenden Einheiten
  -einer für die Puffer
* Eine Zelle kann wegen seinem eigenen Puffer und wegen den Puffern seiner benachbarten Zellen blockiert werden. Damit der Status der Nachbarzellen so schnell wie möglich übertragen wird,ist es nötig Steuersignale zwischen den Zellen hin und her fliessen zu lassen.

- Schlussfolgerungen:

* Die Puffer erlauben es den Sende- und Empfangszellen mit eigener Geschwindigkeit zu arbeiten.(Jedenfalls solange nicht aus einem leeren Puffer gelesen oder in einen vollen Puffer geschrieben wird.)
* Je grösser der Puffer ist,desto weniger Zeit verliert man wegen der Kommunikation zwischen den Zellen.

W A R P A R R A Y A R C H I T E K T U R

*MERKMALE

-1- einfache Topologie eines linearen Arrays

-2- leistungsfähige Zellen mit lokaler Programm-Kontrolle.

-3- grosse Datenspeicher.

-4- grosse Kommunikationsbandbreite zwischen den Zellen.

*VORTEILE

- Lineare Arrays sind verhältnismässig einfach zu benutzen.
- Lineare Arrays sind einfach zu implementieren.
- Ein lineares Array benötigt nur eine schmale I/O-Bandbreite,da nur zwei Zellen mit der Aussenwelt verbunden sind.
- Wegen der Leistungsfähigkeit der einzelnen Zellen kann man mehrdimensionale Topologien simulieren: z.B. kann eine einzelne Zelle durch Multiplexen die Funktion einer ganzen Kolonne von Zellen übernehmen ==> 2-dimensionales syst. Array.

*Gut geeignet für:

- systolische Vorgehensweise,wegen der grossen Kommunikationsbandbreite:
  20 Millionen 32-Bit Wörter pro Sekunde
  10 Millionen 16-Bit Wörter pro Sekunde
- aufwendige Berechnungen innerhalb der Zellen.
- globale Operationen (Operationen,wo jede Ausgabe von vielen Eingaben abhängt)
- lokale Operationen (Operationen,wo jede Ausgabe von wenigen Eingaben abhängt)

I N T E R F A C E U N I T

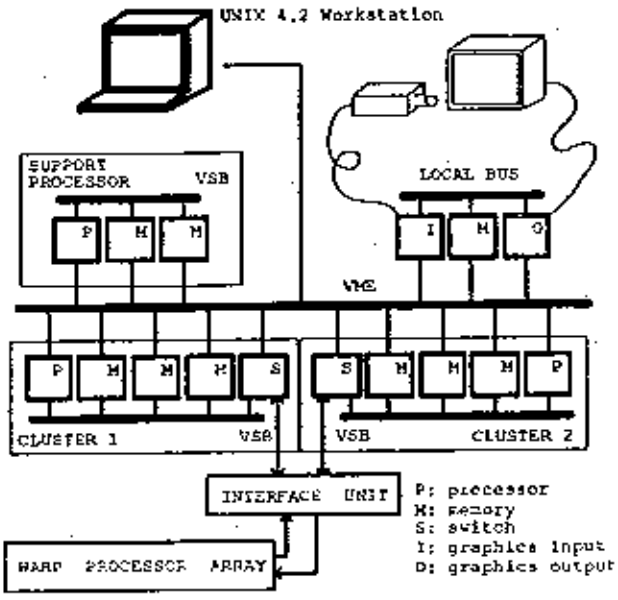
- Implementierung auf einem 15" x 17" Board.
- IU kann Adressen generieren.
- IU enthält Kontrollregister,die dem Host zugänglich sind.So kann der Host das IU und das Warp-Array ansteuern.
- IU kann den Takt des Warp-Array kontrollieren und so die Programme im Schrittmodus laufen lassen.
- IU kann jede Zelle einzeln überprüfen und erlaubt so unabhängige Programm-Korrekturen.

H O S T S Y S T E M

* Aufgaben

-1- Daten-Transfer vom und zum Warp-Array
-2- Programme durchführen,die nicht für das Warp-Array geeignet sind.
-3- Peripherie kontrollieren (Digitizer/graph.Display/usw)
-4- Interface zu anderen Rechnern und I/O-Geräten.

* Aufteilung : externer Host und Standart UNIX-Workstation



- Zwei Cluster-Prozessoren arbeiten parallel
- Ein Support-Prozessor bearbeitet Interrupts vom Warp-Array.
- Die 8M-Byte Speicher in den Cluster-Prozessoren können über den VME-Bus auf 56M-Bytes erweitert werden.

Die Workstation hat UNIX Programmierumgebung und kontrolliert den externen Host.

Der externe Host kontrolliert das Warp-Array und sorgt über die Cluster-Prozessoren für einen schnellen Daten-Transfer zum Warp-Array.
Ausserdem führt der externe Host die oben genannten Teilprogramme aus.

P R O G R A M M I E R S P R A C H E W 2

- W2 ist eine algol-ähnliche Programmiersprache und ist die einzige Programmiersprache für das Warp-Array.
- Der Compiler kümmert sich um die Parallelität auf System- und Zellenebene.
- Dem Benutzer liegt folgendes Modell des Warp-Array vor:
  - Das Warp-Array entspricht einem linearen Array identischer konventioneller Prozessoren, die asynchron mit ihren Nachbarn kommunizieren können.
  - Für jede Zelle muss ein Programm geliefert werden und man muss auf Blockierungen achten.
  - Pipelining und Parallelität bleiben dem Benutzer verborgen.
- W2 enthält zwei Spezialbefehle: send und receive
  Diese beiden Befehle schicken Daten zwischen benachbarten Zellen hin und her: send (Richtung,Kanal,Quelle)
  receive(Richtung,Kanal,Resultat,Quelle)
- Beispiel-Programm : 10 x 10 Matrix-Multiplikation

```
cellprogram (cid : 0 : 9)
begin
    function mm
    begin
        float col[10]; /* stores a column of the b matrix */
        float row;     /* accumulates the result of a row */
        float element;
        float temp;
        int i,j;

        /* first load a column of b in each cell */
        for i := 0 to 9 do begin
            receive (L, X, col[i], B[i,0]);
            for j := 1 to 9 do begin
                receive (L, X, temp, B[i,j]);
                send (R, X, temp);
            end;
            send (R, X, 0.0);
        end;

        /* calculate a row in each iteration */
        for i := 0 to 9 do begin
            /*  each cell computes the dot product
                between its column and the same row of A */
            row := 0.0;
            for j := 0 to 9 do begin
                receive (L, X, element, A[i,j]);
                send (R, X, element);
                row := row + element * col[j];
            end;

            /*  send the result of each row out */
            receive (L, Y, temp, 0.0);
            for j := 0 to 8 do begin
                receive (L, Y, temp, 0.0);
                send (R, Y, temp, C[i,j]);
            end;
            send (R, Y, row, C[i,9]);
        end;
    end
    call mm;
```

A L L G E M E I N E S C H L U S S F O L G E R U N G E N

* Hauptanwendungsgebiete:
  - Sehen mit Computern
  - Roboter-Navigation: ° Weg verfolgen
    ° Hindernissen ausweichen
    ° Weg-Planung
  - Signalverarbeitung
  - Wissenschaftliches Rechnen
* Warp ist in seinen Anwendungsgebieten anderen Maschinen ähnlicher Kosten und Programmierfähigkeit weit überlegen.
* Einfache lineare Topologie verbunden mit grosser Rechenleistung ersetzen den Mangel an mehrdimensionaler Verknüpfung.
* Die grosse Rechenleistung der Zellen wird angepasst mit grossen internen und externen Kommunikationsbandbreiten.
* Für die Entwicklung von parallelen Rechnern mit Pipeline-Technik ist ein leistungsfähiger Compiler unbedingt notwendig: -1- Die systematische Analyse deckt Fehler und Probleme auf, die beim Entwerfen der Programme sehr schwer zu entdecken oder zu beheben wären.
  -2- Der Compiler ist ein gutes Hilfsmittel zum Vergleich verschiedener Architekturen.
* Es ist wichtig die Hauptanwendungsgebiete so früh wie möglich festzulegen, damit man die Maschine schon beim Entwurf auf die späteren Anwendungen vorbereiten kann.
* Der Entwurf eines Prototyps ist wichtig:
  - um eine realistische Vorstellung der auftretenden Probleme zu bekommen.
  - um Software und zusätzliche Hardware entwickeln zu können.
  - um die Realisierbarkeit weiterer Projekte zu Testen.
  - um Verbesserungen und Veränderungen vornehmen zu können.

Seminar

"Innovative Prozessor Architekturen"

MOM - MAP ORIENTED MACHINE
AN INNOVATIVE COMPUTING ARCHITECTURE

von

Alex Hirschbiel

Michael Weber

ABSTRACT

In this paper we describe an innovative computing architecture, called *Map Oriented Machine* (MOM). Concering speed, cost and flexibility today there mainly exist two extreme solution: the totally flexible, but slow von Neumann computer and the very fast, but expensive and inflexible fully parallelized solution directly implemented on customized silicon. With some applications running on the MOM we show that it not only fills this gap, but also is a very good instrument to implement algorithms which have a map-oriented organization such as image processing. The basic idea of speeding up the algorithms is to parallelize the program access by combinational hardware, whose development is supported by some CAD tools.

INTRODUCTION

The traditional approach in computer architectures is the well-known concept based on John von Neumann's work. Also well-known is the bottleneck between the processor and the memory of these computers. All data is accessed sequentially and furthermore the program access and program execution is sequential. Despite these drawbacks, the programming of von Neumann computers is very easy and also highly flexible.

Many efforts were and are done to improve the performance of this architecture. Special coprocessors were implemented to speed up various tasks in the computer. There are numeric processors like floating point processors to accelerate parts of a program, or there are I/O-processors to perform direct memory access, or graphic processors to speed up the display of immense amount of graphical data. Huge primary memories are installed to minimize the access to secondary storage. Reduced instruction sets are used to have faster processing units available.

Another approach is it to extend the basic von Neumann concept into a multicomputer concept. These systems parallelize the programs by using special arrangements (arrays, hypercubes, vectors, and others) of a few computers. The programming of these systems is very difficult and they also have expensive operating systems and a large administrative overhead (PU82), (YOFU86).

Totally different to the approaches above (all based on the von Neumann concept) is the use of special processing elements implemented in a special hardware. A certain number of these uniform processing elements (PEs) is arranged in linear or two-dimensional arrays. All these PEs perform the same operations in parallel and are connected for intercommunication purposes. So the degree of parallelism is direct proportional to the amount of PEs. The data is pulsed through these PEs, if the implementation is based on a systolic array concept (KUN79), or the complete data is stored in a PE square array. All these special-purpose architectures are very fast, but also very inflexible. If the special hardware is not capable of a certain algorithm, high design cost and a long turn-around time is necessary to implement new processing elements and their arrangement.

So we have the classical von Neumann concept on the one side and the special-purpose hardware solutions on the other. Both have their advantages and disadvantages concerning speed, cost and flexibility (BLA84), (SJGS85). Between these two main approaches is a remarkable gap, which is filled by the new concept presented here. The *Map-Oriented Machine* (MOM) is a medium speed solution at low cost. The basic idea of accelerating algorithms is to parallelize the program access by combinational hardware, where the data access remains sequential. The data itself is stored in a two-dimensional map-oriented memory and is accessed via a special window cache. The system can be personalized in a highly mechanized way to be adapted to a surprisingly wide range of

applications. A CAD environment to develop applications includes the programming language MOPL (Map Oriented Programming Language) which supports the 'programming' of the MOM.

THE ARCHITECTURE OF THE MAP-ORIENTED MACHINE

The Map-oriented Machine consists of four main parts (see figure 1), the map-oriented data memory, the data cache, the move control unit and the programmable part, the problem oriented logic units. All these parts are connected to each other via a VME-bus.

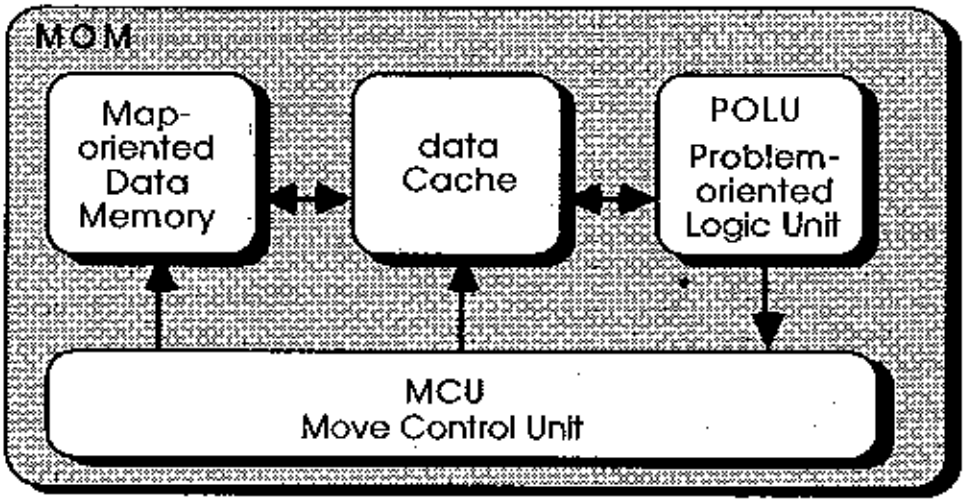


Figure 1: MOM Architecture

The map-oriented data memory has a two-dimensional organization and is addressable in x and in y direction. At each address a pixel is located (z direction), which has a variable size and a variable format (figure 2).

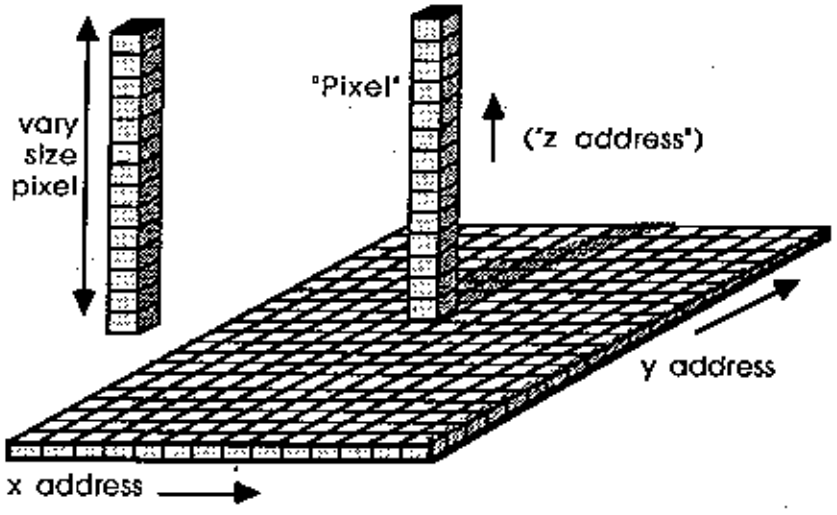


Figure 2: Map-oriented Data Memory

The pixels can be used to store bit vectors, numeric values, they may have a tagged format or any mixed representation. In the whole memory all these types of pixel formats may be used at the same time to have many different types of variables and values available.

The data cache is a window to the map-oriented data memory located at the position which is addressed by the x and y address in the memory. The cache then holds a copy of a few pixels in the memory to make them accessable for the execution. All the pixels in the cache may be accessed in parallel and sent to the actual problem-oriented logic unit. The size of the cache is variable to adapt it to different applications (figure 3).

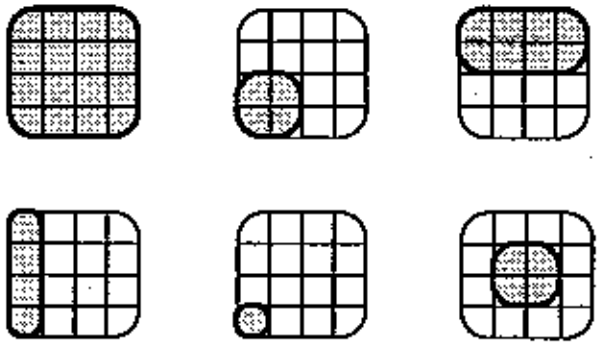


Figure 3: Window Cache Principle

In the move control unit (MCU) the addresses for the cache are generated to provide a controlled cache movement over the data in the memory. Two major move strategies are available. Schematic move strategies are independent of the data in the cache. There are single steps to neighbour positions possible, as well as gotos and relative jumps possible. A combination of these steps may result in linear step sequences, videoscan, shuffle sequences and other schematic move sequences. Non-schematic strategies are dependent on the data currently provided by the cache. These data driven movements are important to trace for example the contour of an object in an image. A, not complete, summary of possible movements is shown in figure 4.

All these movement strategies are supported by memory space limit registers to determine the boundary of the certain move area.

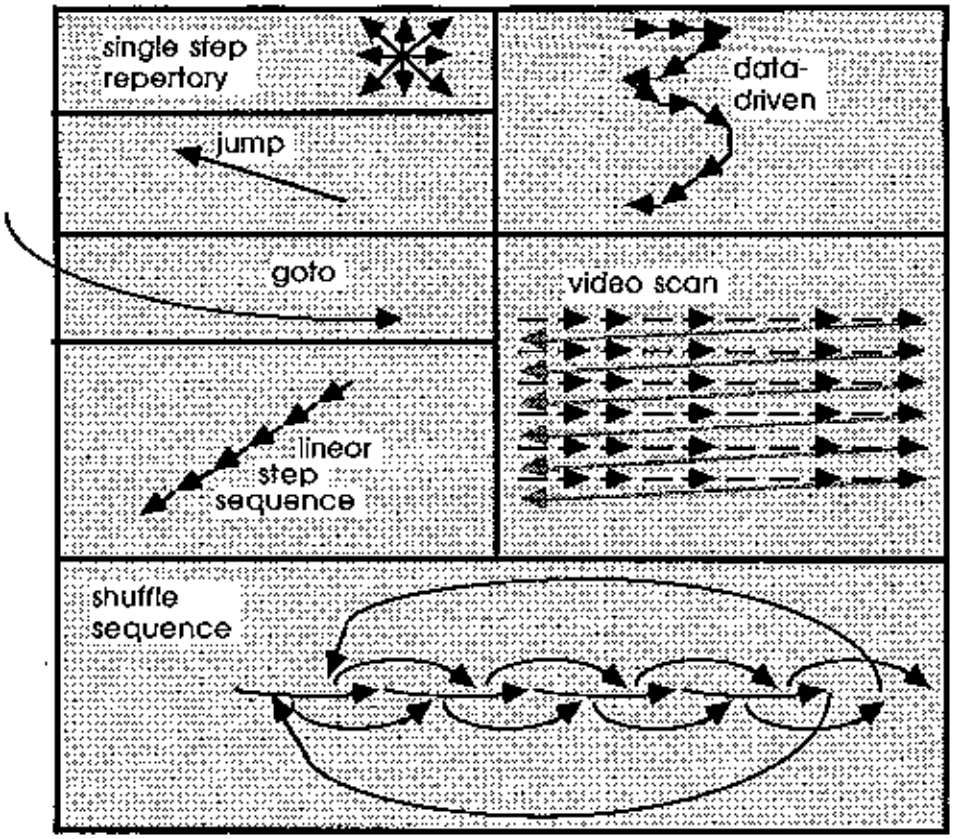


Figure 4: Cache Move Examples

The programmable part of the MOM consists of several problem-oriented logic units (POLUs), one for each application. They can be selected by a special application select code (see figure 5). Such a POLU does not hold a sequential program, but is a CAD generated combinational hardware, which is obviously faster than a sequential program. The hardware contains a combinationally stored set of reference patterns, which is matched with the cache's contents in parallel. As a result of this pattern match a different pattern can be written back into the cache to cause the change of data in the cache and with this, in the memory. A second result, if the application requires data dependent cache movement, is the move code to be sent to the move control unit. To support numeric processing a traditional arithmetic logic unit (ALU) may be added as one of the POLUs. Then the cache is used as the register file for this ALU.

To implement new POLUs, that means to 'program' the MOM, a CAD toolbox is used. A comfortable editor to specify the reference patterns and the information necessary to interpret this is available. A translator generates the program code to store this reference data in EPROMs, PLAs, RAMs or ROMs. A high level language MOPL (Map Oriented Programming Language) may also be used to specify the patterns for new POLUs or to use existing POLUs as sort of standard functions and to control the movement of the cache.

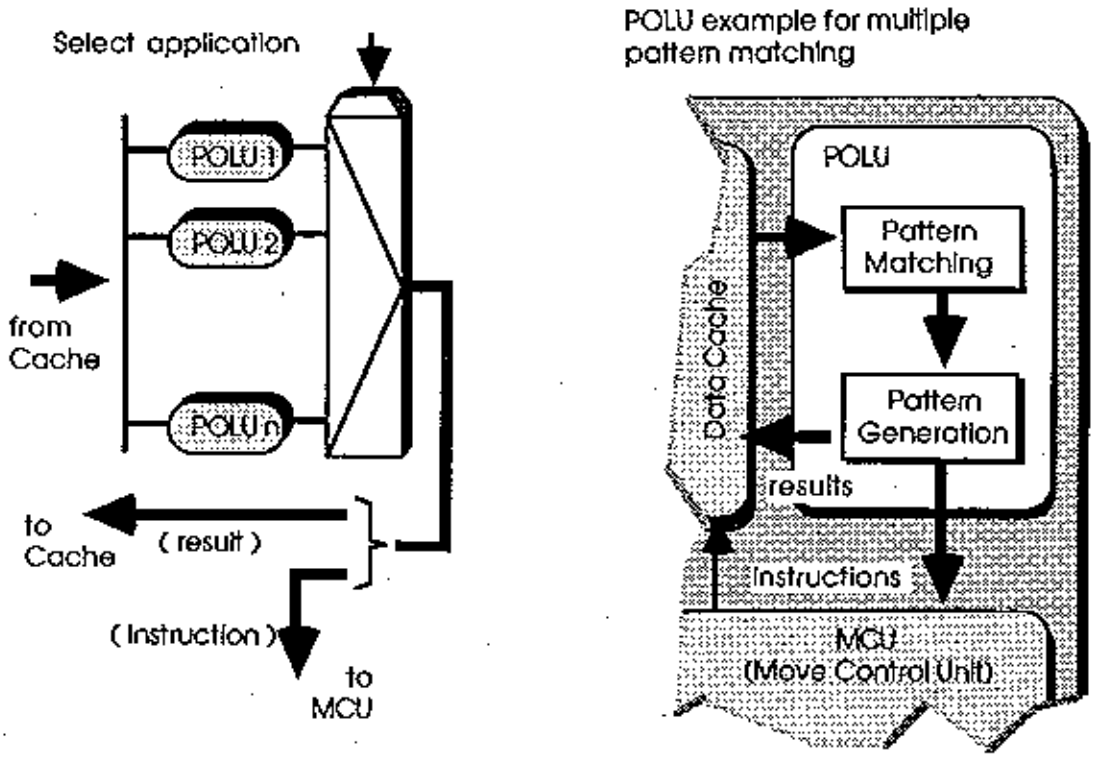


Figure 5: The Problem-Oriented Logic Units

The map-oriented machine is connected via a high speed bus to a workstation serving as a host. This bus (1 Mbyte per second) is used to load and unload the data memory, to have peripheral I/O equipment available and to provide the user interface to the MOM (WES87).

APPLICATION EXAMPLES

The MOM is capable to execute almost any kind of data processing. Anyhow there are problems more or less feasible for MOM execution. Classes of problems with their algorithms showing the power of the MOM are described.

Image Preprocessing

Since the pixel memory is a multilayer bit map, it obviously is an excellent representation medium for two-dimensional images. This fact opens the wide range of image preprocesses for MOM execution (see also STERN81). The simplest problems in this area are bit operations (boolean operations applied to different layers in a pixel) and set operations. Sets can be stored as areas of set bit in different pixel layers, and boolean instructions are applied to compute functions like *contained, intersection, union,* or *difference*. These operations require just an one-pixel cache to be moved over the memory. The movement strategy is to visit each pixel exactly once and to match it with the set of reference patterns. This is done by a videoscan, i.e. the cache is moved from the left bottom corner, row by row, to the right top corner of the pixel memory. To provide the required operation each reference pattern represents a possible input-instance of the entire boolean operation, the corresponding result pattern represents the result of the operation applied to the input-instance. Figure 7 gives an example of the operation 'intersection'.

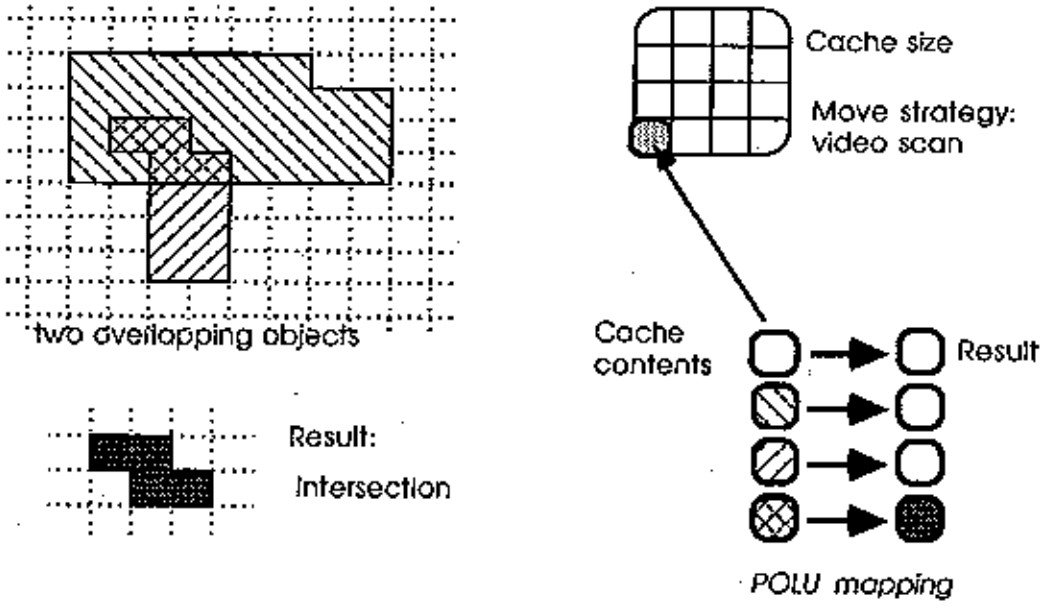


Figure 7: Set operation 'intersection'

These bit and set operations can be combined to perform functions on colour graphics data (colours stored in different layers), e.g. to mix colours or to make colours invisible.They may also be used to implement topological algorithms for image analysis.

The next slightly more complex level in image preprocessing needs the evaluation of neighbourhood information. To meet this requirement a larger cache is used. In most cases a 3-by-3 pixels wide cache is sufficient to get the necessary local informations. Usually the eight neighbours of the center pixel determine the change of the cache's contents. Shrinking and expansion of image structures are examples for this type of applications. To expand an object it is necessary to know whether a neighbour of the center pixel is already blocked or whether it is still free to expand the object (see Figure 8). The two reference patterns and their result patterns shown in Figure 8 are the first two encountered when performing a videoscan from the left bottom to the right top of the pixel memory section in the example of Figure 8. Naturally there are more different patterns necessary to provide a complete expansion.

The expansion algorithm may also be implemented using a varied move strategy. The cache may follow the outline of the object and add new pixels at the edge of the existing object. In this case the moves of the cache are dependent of the matching reference patterns. Now the POLU provides the MCU with the new move commands.

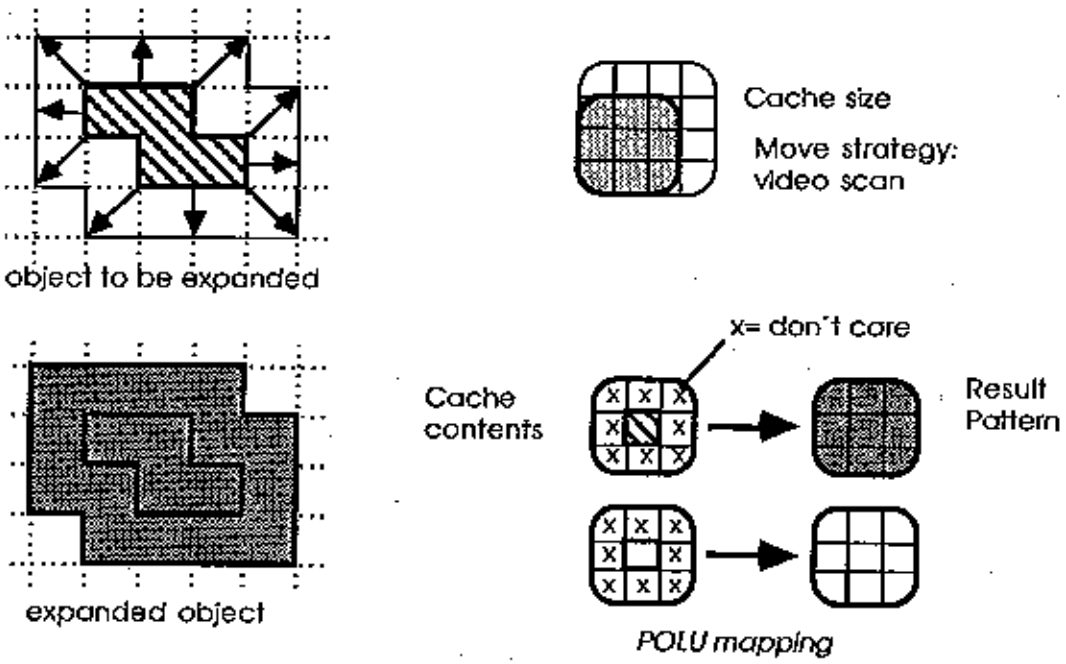


Figure 8: Expansion of an object

This explains that problems can be realized in different MOM algorithms, either in different reference patterns, or in different move strategies, or in both. The user has to decide whether to have a more complex set of reference patterns, usually resulting in less execution time, or whether to have a simple move strategy, resulting in an easier programming of the reference patterns.

Layout Processing

Another suitable field for MOM executions is VLSI layout processing, which can be regarded as image processing too. Since layout can be excellently stored in multi-coloured bit maps the ability to solve problems by scanning a local window over the layout data is obvious.

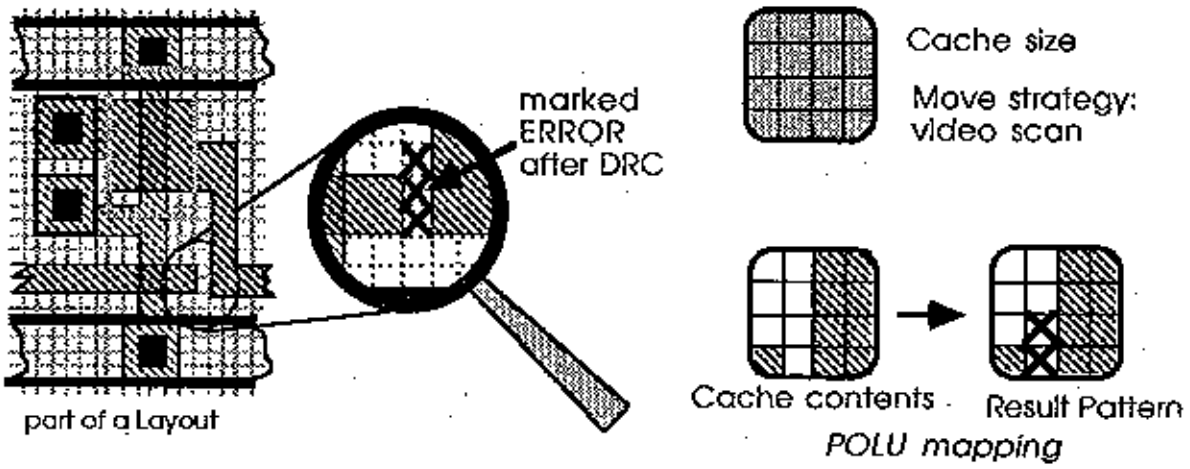


Figure 9: Design rule check example

To perform a design rule check of the layout of integrated circuits it is necessary to have a cache, which is one pixel larger than the largest design rule, e.g. if the rule "minimum metal spacing is three lambda" has the biggest 'lambda factor' of all design rules, the cache has to have a size of 4-by-4 pixels (one pixel is equivalent to one lambda unit) in order to cover this rule, and, at least one of its direct neighbour pixels (WE885). The entire design rule check is done by a single videoscan over the layout. The reference patterns represent all possible design rule violations. A match of one or more of these patterns indicates a violation. For all reference patterns there is only one result pattern directly marking the location of a design rule violation in using a special error layer at the position where a reference pattern has matched. Figure 9 shows an example of a design rule violation and its detection by a reference pattern. The number of possible design rule violations is rather limited compared to the number of designs possible in an area as large as the cache. A

Mead-&-Conway (MECO80) design rule check for example requires only 258 reference patterns and takes 1 second for a layout size of 1,000-by-1,000 lambdas. Clearly only orthogonal layout structures can be processed, because the underlying pixel memory allows only orthogonal structures to be stored.

Minimum-cost Path

To find a minimum-cost path using the Lee routing algorithm (LCY61), (BS81) is another MOM application example in CAD for VLSI. Starting from a specially marked pixel (the start cell), the minimum-cost path to a target cell is searched by propagating arrows from the start cell, thus propagating a wavefront to the target cell. By backtracking the path, back along the arrows, the shortest connection between the two points is achieved (see figure 10).

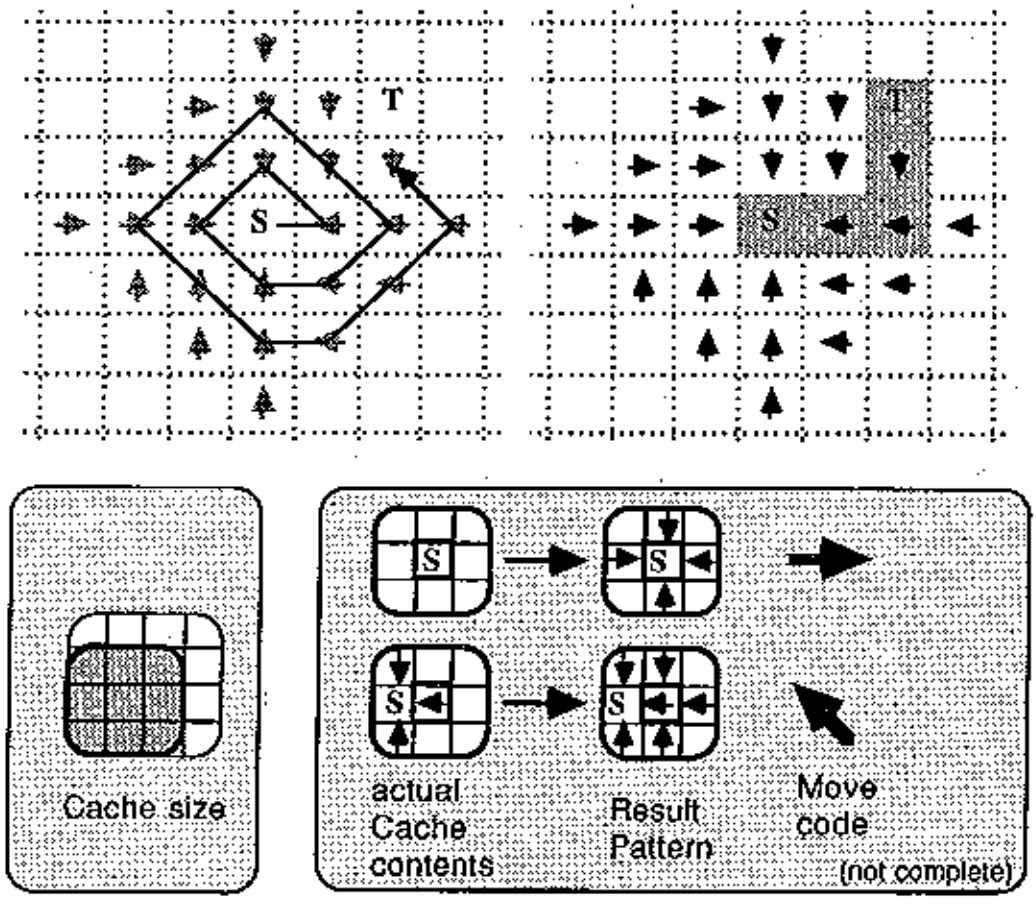
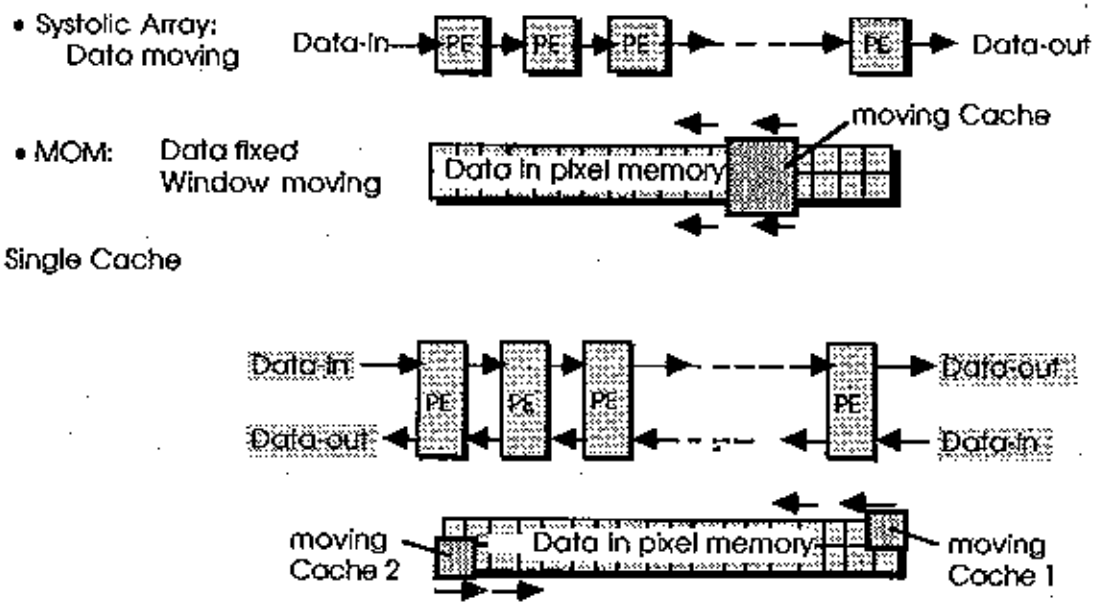


Figure 10: Lee Routing example

This routing process requires the combination of different sets of reference patterns and different movement strategies. First a single pixel cache is videoscanned over the image to find the start cell. The waveform expansion of arrows requires a 3-by-3 pixel cache to get

programmed or personalized in another way. In the latter application MOM would be used as a CAD tool (like a programmable systolic array compiler) within a toolbox for the design of systolic architectures. So, as a matter of fact, everything which fits onto a systolic array implementation can be directly mapped onto the MOM system.



a) Single Cache

b) Multiple Caches

Figure 11: Aping of systolic arrays

CURRENT WORK

To cover a wider area of data processing problems the MOM having been introduced here, can be extended to have two or more caches running simultaneously. This leads to two data selections at a time, which can be interleaved, and it leads to two parallel data accesses. E.g. to multiply two matrices one cache scans the first matrix row by row, the second cache the second matrix column by column. The two caches exchange information via the POLU to provide a multiplication of the two matrices (see also Figure 11).

A main topic of our present work is to design a MOM programming language (MOPL) as an aid to program MOM easily, safely and quickly. This language includes features for the description of the shape of reference patterns and result patterns as well as the behavioural description of the cache movements. Via existing CAD-tools and a language interpreter the algorithms are transformed into new POLUs to extend MOM. This language is not only used to install desired new POLUs in the MOM, it also serves as a user's programming language to run and control the MOM using the existing POLUs.

Currently the MOM prototype based on standard TTL-circuits is reassembled using self-designed full custom NMOS circuits to get a higher degree in integration and in speed. E.g. an extendible 4-by-4 pixel cache on a single chip has been designed and manufactured to substitute a whole board of TTL-circuits. (This work was funded by the German Ministry of Research and Technology within the E.I.S.-project.)

REFERENCES

(BHN85) K. Bastian, R. Hartenstein, W. Nebel: *VLSI-Algorithmen: Innovative Schaltungstechnik statt Software - Shuffle Sort,* VDI-Berichte Nr. 550, 1985

(BLA84) T. Blank: *A Survey of Hardware Accelerators used in Computer-Aided Design,* IEEE Design and Test of Computers, August 1984

(BS81) M.A. Breuer and K. Shamsa: *A Hardware Router,* In: Journal Of Digital Systems, Vol.4, Issue 4, 1981.

(FOKU80) M.J. Foster, H.T. Kung: *The Design of Special-Purpose VLSI Chips,* Computer, January 1980

(GHHO86) R. Gebhardt, R. Hartenstein, R. Hauck, D. Oelcke: *Functional Extraction from Personality Matrices of MOL (Matrix Oriented Logic) Circuits,* report, Kaiserslautern University, 1986

(HIR85) A. Hirschbiel: *PISA Maschine, Eine spezielle Hardware für pixel orientierte Bildverarbeitung,* report, Kaiserslautern University, 1985

(HNW84) R.W. Hartenstein, R. Hauck, A.G. Hirschbiel, W. Nebel, M. Weber : *PISA, A CAD Package And Special Hardware For Pixel-Oriented Layout Analysis.* In: ICCAD-84, Digest Of Technical Papers, Santa Clara, 1984.

(KUN79) H.T. Kung: *Let's Design Algorithms for VLSI,* Caltech Conference on VLSI, 1979

(KUN82) H.T. Kung: *Why Systolic Architectures?,* Computer, January 1982

(LCY61) C.Y. Lee: *An Algorithm For Path Connections And Its Applications.* In: IEEE Trans. on Electronic Computers, vol EC-10, pp. 346-365, Sep. 1961.

(LOLA80) A. Lopez, H. Law: *A Dense Gate Matrix Layout Method for MOS VLSI,* IEEE Journal of solid-state circuits, Vol. SC-15, No. 4, Aug. 1980

(MECO80) C. Mead, L. Conway: *Introduction to VLSI Systems,* Addison-Wesley, 1980.

(PU82) K. Preston jr., L. Uhr: *Multicomputers and Image Processing.* Academic Press, 1982.

(SJGS85) L. Snyder, L. H. Jamieson, D. B. Gannon and H.J. Siegel: *Algorithmically Specialized Parallel Computers.* Academic Press, 1985.

(STER81) S. R. Sternberg, *Parallel architectures for Image processing,* in "Real/Time Parallel Computers" (M. Onoe, K. Preston, Jr., and A. Rosenfeld, eds.) pp. 347-359. Plenum, New York 1981.

(VEL87) I. Velten: *Implementierung des Lee-Algorithmus auf MOM* (in german), report, Kaiserslautern University, 1987

(WEB85) M. Weber: *Ein Patterngenerator zur automatischen Referenzmustererzeugung* (in german), report, Kaiserslautern University, 1985

(WES87) J. Westphal: MOM Interface (in german), report, Kaiserslautern University, 1987

(WEI67) A. Weinberger: *Large Scale Integration of MOS Complex Logic: A Layout Method,* IEEE Journal of Solid-State Circuits, Vol. SC-2, No. 4, Dec. 1967

(YOFU86) T. Young, K. Fu: *Handbook of Pattern Recognition and Image Processing,* Academic Press, 1986