Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses

Band: 73 (1982)

Heft: 3

Artikel: Zweidimensionale Rechner : Theorie und Beispiele

Autor: Egbersen, A. P. J.

DOI: https://doi.org/10.5169/seals-904928

Zweidimensionale Rechner – Theorie und Beispiele

A. P. J. Engbersen

681.323:519.763;

Will man in der Bildverarbeitung mit Echtzeitbetrieb arbeiten, so müssen dazu zweidimensional konzipierte Rechner verwendet werden. Dabei wird sehr oft «Parallelismus» angewendet. Es soll hier die Theorie der parallelen und zweidimensionalen Rechnersysteme vorgestellt werden.

Pour faire du traitement d'image en temps réel, il est necessaire de développer un processeur conçu pour opérer en deux dimensions. Le concept de «parallélisme» est alors très souvent utilisé. On présente ici la théorie des processeurs qui travaillent en parallèle et en deux dimensions.

1. Einleitung

Mit diesem Artikel soll versucht werden, den Leser in die Theorie der zweidimensionalen Rechnersysteme einzuführen und ihm gleichzeitig einen kleinen Überblick zu verschaffen. Er soll zudem anhand einiger kleinerer Systeme, welche in den vergangenen Jahren entwickelt worden sind, die hier auftretenden speziellen Probleme und deren Lösungsversuche kennenlernen. Den Hauptteil des Artikels bilden jedoch die Erklärungen der Begriffe Vektorprozessor, Paralleler Prozessor, sowie Assoziativer Prozessor und Feldrechner (Array Prozessor).

2. Klassifikation

Die Gliederung der verschiedenen Rechnertypen nach topologischen Gesichtspunkten ist aus Komplexitätsgründen unmöglich. Ein Beispiel: Wird im Rechner eine Änderung vorgenommen, etwa um den Datenfluss effizienter zu gestalten, so könnte sich seine Klasse in der topologischen Gliederung ändern.

Aus diesem Grunde hat Flynn [1] bereits im Jahre 1972 ein Gliederungsschema entwickelt, in welchem der Rechner die Funktion eines Verarbeitungsinstrumentes für Daten- und Befehlsflüsse übernimmt, welche nach seiner Darstellung je eine einfache oder mehrfache (= parallele) Struktur aufweisen. Somit ergeben sich die im folgenden beschriebenen vier Klassen:

– *Einfacher Befehlsfluss, Einfacher Datenfluss (Single Instruction, Single Data;* SISD*):* Bei solchen Maschinen wird pro Befehl immer genau ein Datenelement in ein neues umgewandelt. Es werden keine zusätzlichen Funktionen ausgeführt. Makroskopisch betrachtet, arbeiten fast alle heute verfügbaren (einfachen) Mikroprozessoren nach diesem Prinzip. Mikroskopisch gesehen, überlappen sich der Befehlsabholzyklus eines Befehls und die (interne) Verarbeitung des vorhergehenden Befehls, dies aber entspricht dann einer MISD-Struktur.

– *Mehrfacher Befehlsfluss, Einfacher Datenfluss (Multiple Instruction, Single Data;* MISD*):* In einer derartigen Maschine werden in einer Zeitspanne T durch n verschiedene Prozessoren auf n sequentiellen Datenelementen n (im Prinzip) verschiedene Befehle ausgeführt. Grundlegend dabei ist, dass in der nächsten Zeitspanne T der Prozessor i ($1 \leq i \leq n$) wieder den genau gleichen Befehl wie in der vorhergegangenen Zeitspanne ausführt, diesmal aber auf dem Datenelement $(i-1)$. Es werden also alle Daten immer wieder um einen Prozessor weiter verschoben, ähnlich der Fliessbandarbeit in einer Fabrik. Diese Struktur nennt man auch «Pipelining».

– *Einfacher Befehlsfluss, Mehrfacher Datenfluss (Single Instruction, Multiple Data;* SIMD*):* Zu dieser Klasse zählen unter anderem die Feldrechner und vor allem Rechner, die in der Bildverarbeitung eingesetzt werden. Das diesen Maschinen zugrundeliegende Prinzip ist folgendes: Auf verschiedenen Datenelementen führen mehrere Rechner in einer Zeitspanne T gleichzeitig jeweils den gleichen Befehl aus – oder sie belassen die Datenelemente gänzlich unverändert. Beispiele zu dieser Struktur werden später noch behandelt.

– *Mehrfacher Befehlsfluss, Mehrfacher Datenfluss (Multiple Instruction, Multiple Data;* MIMD*):* Maschinen dieser Art bestehen meist aus vielen zusammengesetzten, aber unabhängigen Prozessoren, wobei jeder einzelne gemäss seinem eigenen Programm seine Daten verarbeitet. Es gibt viele Möglichkeiten, diese Prozessoren zu synchronisieren und untereinander kommunizieren zu lassen, um damit ein zusammenhängendes System zu erhalten. Im allgemeinen sind MIMD-Maschinen durch einen gewaltigen zusätzlichen Arbeitsaufwand für die Interprozesskommunikation gekennzeichnet und deshalb in ihrer Leistung beschränkt.

Es liegt auf der Hand, dass die vier beschriebenen Klassen eine jeweils andere Speicher- und Datenflußstruktur aufweisen müssen (Fig. 1). So genügt bei den SISD-Maschinen ein Speicher, der sowohl der Daten- als auch der Befehlsspeicherung dient, während SIMD-Maschinen nur mit einem mehrfach zugänglichen (multiple access) Speicher oder mit je einem Arbeitsspeicher für die Datenspeicherung pro Prozessor wirtschaftlich sind. Bei den MIMD-Maschinen verhält es sich wie

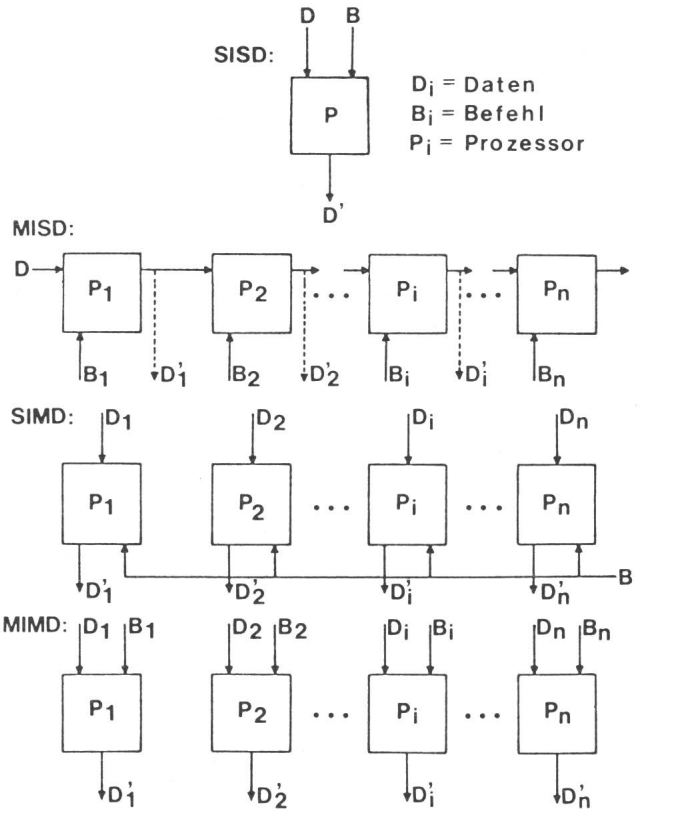

Fig. 1 Klassifikation nach Flynn

bei den SIMD-Maschinen, wobei zusätzlich die Befehlsspeicherung wie die Datenspeicherung behandelt werden muss.

Die Probleme, die auftreten, wenn mehrere Prozessoren in einer derartigen Struktur die gleichen Daten benötigen, erfordern ausgeklügelte und oft komplizierte Verfahren zur Bewältigung der Datenflußsteuerung. Gemäss einer kürzlich aufgestellten Behauptung von *M. Duff* ist die Datenflußsteuerung und die Datenverteilung das grösste Problem der heutigen Bildverarbeitung [2].

Die nachfolgende Liste stellt eine kurze Übersicht der verschiedenen Arten von Rechnerstrukturen dar. Sie ist nicht vollständig und soll vor allem dem interessierten Leser ermöglichen, in der einschlägigen Literatur nachzuschlagen.

1. Vektor-Befehlsrechner

a) Speicher zu Speicher (CDS STAR–100)
b) Register zu Register (Cray Research: Cray–1)

2. Ensembles (Zusammenschaltungen von zwei oder mehreren Rechnern)

(Burroughs: ILLIAC, PEPE)
(ICL: Distributed Array Processor DAP)

3. Assoziative Feldrechner

(Goodyear Aerospace Corporation: STARAN)
(Sanders Associates: OMEN)

4. Algorithmische Feldrechner

(IBM 3829)

5. Algorithmische Minifeldrechner

(CSP Inc.: Macro Arithmetic Processor)

6. Spezielle Entwürfe

(CLIP4: Cellular Logic Image Processor)
(LPPP: Local Parallel Pattern Processor, TOSHIBA)
(FLIP: Flexible Image Processor)

3. Rechnerstrukturen

Mit der vorstehenden Einteilung als Leitfaden können die verschiedenen Arten von Rechnertypen erläutert werden. Ausgehend vom einfachsten Rechner, dem Vektorprozessor, wird durch sukzessive Erweiterung des Systems, der Parallele Prozessor, der Assoziative Prozessor und schliesslich der Array Prozessor erklärt.

3.1 Vektorprozessor

Wenn man die Vektorrechnung betrachtet, fällt sofort auf, dass viele Berechnungen aus der elementweisen, unabhängigen Manipulation von Vektorkomponenten bestehen. Unter der Voraussetzung, dass diese Manipulationen genügend unabhängig sind, können diese auch in einem speziell dafür konzipierten Rechner durchgeführt werden: Wenn man nämlich für jedes Datenelement des Vektors einen kleinen Prozessor vorsieht, kann eine solche Manipulation gleichzeitig auf allen Elementen durchgeführt werden. Dabei werden sowohl Prozesszeit als auch Instruktionen eingespart. Der Rechner wird wegen seiner Operationsweise Vektorprozessor genannt.

Im allgemeinen besteht ein Vektorrechner aus mehreren parallel arbeitenden Prozessoren. Dabei berechnet jeder Prozessor aus zwei «Ausgangsvektoren» gemäss einer vorgegebenen Instruktion (z.B. Multiplikation) einen neuen Vektor. Unabhängig von diesem Teilergebnis kann das Endergebnis noch abhängig sein von einem sog. Maskenvektor.

In Figur 2 ist eine Vektorinstruktion des STAR-100-Rechners zusammen mit den Ausgangs-, Ergebnis- und Maskenvektoren dargestellt. In der Instruktion sind Felder für Operationscode, Vektoradressen und Maske vorgesehen. Die Funktionsweise sollte anhand dieser Darstellung klar sein.

Es gibt drei Gründe für die Einführung von Vektorrechnern: Erstens erlaubt ein derartiger Rechner kürzere Programme, weil viele Jump-, Lade-, Speicher- und Testinstruktionen eingespart werden können. So braucht man z.B. auf «Vektorebene» keine Schleifen mehr zu programmieren. Dies führt zur Verringerung der notwendigen Speicherkapazität und der Bandbreite des Instruktionspfades (weniger Instruktionen pro Zeiteinheit). Zweitens erlaubt diese Struktur eine parallele Verarbeitung, die eine Beschleunigung der Programme zur Folge hat, und drittens ist die Vektornotation für gewisse wissenschaftliche Berechnungen besonders geeignet.

3.2 Paralleler Prozessor

Der parallele Prozessor ist hinsichtlich der Struktur dem Vektorrechner nahe verwandt. Derartige Prozessoren sind sowohl nach SIMD- als auch nach MIMD-Prinzipien gebaut. Im Gegensatz zum Vektorrechner zeichnet sich der parallele Prozessor vor allem durch die Verknüpfung der Nachbarelemente aus. Solche Verknüpfungen ermöglichen z.B., dass Prozesse in den Rechnerelementen voneinander abhängig werden oder dass Daten «durchgegeben» werden können.

In SIMD-Strukturen läuft im Prinzip in jedem Element dasselbe Programm ab. Dies geschieht jedoch auf Daten, die entweder von Nachbarn oder von einer externen Quelle stammen. Ob eine bestimmte Instruktion von einem Element ausgeführt wird, kann unter anderem vom Ergebnis der Nachbarelemente abhängig gemacht werden. Damit ist gegenüber dem Vektorrechner eine zusätzliche Dimension gewonnen. MIMD-Strukturen können zusätzlich in jedem Element einen andern Algorithmus ausführen.

Bei der Besprechung des CLIP-4-Rechners wird ein Beispiel einer parallelen Struktur angeführt werden, bei welcher die Berechnungen in einem Element vom Ergebnis von dessen Nachbarn abhängig sind.

Parallele Prozessoren bewähren sich im allgemeinen nur, wenn die auszuführenden Algorithmen ausreichend unabhängig sind. Es ist klar, dass durch die parallele Ausführung in günstigen Fällen sehr viel Zeit eingespart werden kann. Wenn jedoch die Unabhängigkeit der Prozesse nicht gewährleistet ist, wird die zusätzliche Arbeit für die Interprozesskommunikation sehr schnell zum Engpass.

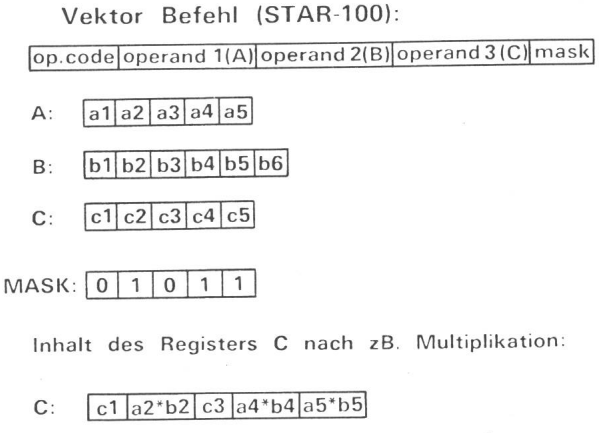

Fig. 2 Vektorbefehl (STAR-100)

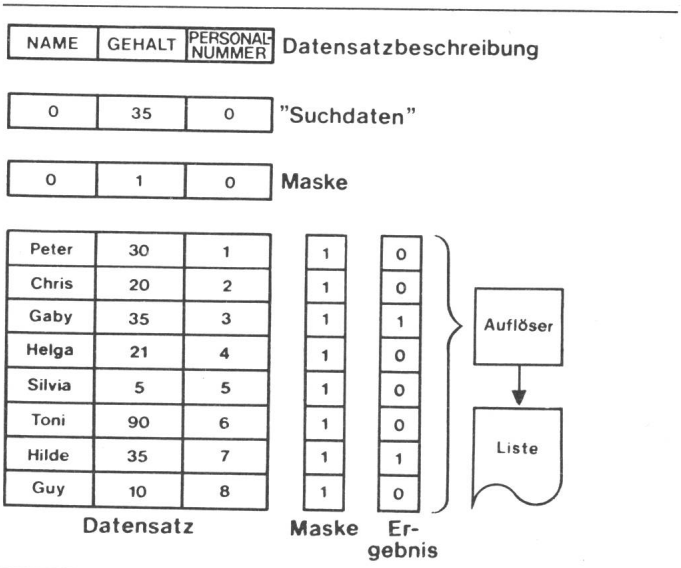

Fig. 3 Logik eines Assoziativen Prozessors

3.3 Assoziativer Prozessor

Der assoziative Prozessor ist eigentlich eine spezielle Ausführung des parallelen Prozessors: Eine Gruppe von N Datenelementen (z.B. Datensätze oder Records) wird N unabhängigen Prozessorelementen zugeführt. Alle Prozessorelemente testen gemäss einem Kriterium die gleichen Felder der ihnen zugeführten Datenelemente und führen ihre Ja- oder Nein-Entscheidung einem sog. «Match Resolver» (Koinzidenzauflöser) zu. Dieser erstellt aus den Entscheidungen mit Hilfe eines Adressauflösers (Address Resolver) eine Adressliste derjenigen Datenelemente, die das Kriterium erfüllen. In Figur 3 ist dieses System schematisch dargestellt.

Der assoziative Prozessor ist für die elektronische Datenverarbeitung ein sehr nützliches Instrument, wenn man bedenkt, dass bei Suchprozessen in Datenbanken sehr viele Datenelemente auf ihren Inhalt hin überprüft werden müssen. Stellt ein assoziativer Prozessor die Adressen der benötigten Daten für den Hauptrechner bereit, funktioniert das ganze System wesentlich effizienter.

Da die Elemente eines assoziativen Rechners relativ einfach sind, werden viele Versuche unternommen, Speicher zu bauen, mit pro Speicherzelle eingebauten, assoziativen Eigenschaften [3; 4; 5].

3.4 Feldrechner (Array Processor)

Der Feldrechner ist aus dem parallelen Prozessor entstanden. An der richtigen Stelle eingesetzt, ist der Feldrechner ein sehr leistungsstarkes Instrument.

Im Gegensatz zur linearen Struktur des Parallelen Prozessors besitzt der Feldrechner eine zweidimensionale Struktur. Die Elemente des Rechners sind wie jene einer Matrix angeordnet. Verknüpfungen mit den umliegenden vier oder acht Nachbarn verleihen dem Rechner seine grosse Leistung: Ergebnisse können weitergegeben werden; Berechnungen der einzelnen Elemente auch von Ergebnissen der Nachbarelemente abhängig sein.

Feldrechner können sowohl eine SIMD- als auch eine MIMD-Struktur aufweisen. Dies hängt häufig nur von der Programmierung ab. Solche Rechner finden hauptsächlich im Bereich der Bildverarbeitung (Mustererkennung) sowie bei der geophysischen Datenverarbeitung und in komplizierten wissenschaftlichen Berechnungen Verwendung.

Figur 4 zeigt z.B., wie eine Matrixmultiplikation in nur drei Zyklen mit neun Prozessorelementen ausgeführt werden kann.

4. Programmierung

Die Programmierung dieser Prozessorstrukturen bildet bis heute die grösste Schwierigkeit und ist sehr wahrscheinlich der Hauptgrund dafür, dass Feldrechner und parallele Prozessoren noch immer kein Gemeingut in den heutigen Rechenzentren sind.

Bis jetzt hat die Programmierung sich nur für spezielle Anwendungen bewährt, und Versuche, allgemein verwendbare Programme zu schreiben, sind immer an der Tatsache gescheitert, dass der grösste Teil der Programme nicht in einen für Parallele Rechner optimalen Code umgewandelt werden kann.

5. Spezielle Prozessoren für die zweidimensionale Signalverarbeitung

Zum Abschluss dieses Aufsatzes sollen noch zwei spezielle Strukturen für die Signalverarbeitung geführt werden. Beide können als MIMD-Maschinen betrachtet werden, obwohl bei der ersten Maschine (CLIP-4) eine SIMD-Betrachtungsweise sinnvoller erschiene.

Beispiel:

Nehmen wir einen Rechner mit einer Matrix von 3 x 3 Prozessoren, die je vier Register aufweisen: areg, breg, creg, treg. In diesem Fall koennen wir eine Matrixmultiplikation in drei Schritte durchfuehren:

Algorithmus:

```
Set:    creg:=o;
        breg(i,j)=b(i,j);
        areg(i,j)=a(i,j);                    | Initiation
Shift:  Ith-row of 'A' left I-1 columns;     |
        Jth-column of 'B' up J-1 rows;       |

do k=1 to 3;
  Multiply  : treg=areg*breg;
  add       : creg=creg+treg;
  shift     : areg right one row;
              breg down one column;
enddo;
```

Let:

A:				B:			
	a1	a2	a3		b1	b4	b7
	a4	a5	a6		b2	b5	b8
	a7	a8	a9		b3	b6	b9

Nach der Initiation, ist der Inhalt der Register:

areg:				breg:			
	a1	a2	a3		b1	b5	b9
	a5	a6	a4		b2	b6	b7
	a9	a7	a8		b3	b4	b8

Nach der Ausfuehrung einer Schleife:

creg:			
	a1*b1	a2*b5	a3*b9
	a5*b2	a6*b6	a4*b7
	a9*b3	a7*b4	a8*b8

areg:				breg:			
	a3	a1	a2		b3	b4	b8
	a4	a5	a6		b1	b5	b9
	a8	a9	a7		b2	b6	b7

Fig. 4 **Beispiel einer Matrixmultiplikation eines parallelen Prozessors**
Die Initiation in Shift bedeutet: I-te Zeile von A um (I − 1) Kolonnen nach links verschieben; J-te Kolonne von B um (J − 1) Zeilen nach oben verschieben

Die erste Maschine ist sehr geeignet für Mustererkennung in digitalen Bildern; sie ist eine spezielle Ausführung des Feldrechners.

Die zweite Maschine ist geeigneter für rekursive Prozesse und ist entsprechend sehr flexibel und umstrukturierbar. Beide Maschinen sind tatsächlich schon gebaut worden, und einige Resultate sind bereits vorhanden [6; 7].

6. CLIP-4 (Cellular Logic Image Processor)

Figur 5 stellt die Logik der CLIP-4-Maschine dar. Für den Aufbau eines Rechners, der ein Bild von $N \times M$ Bildelementen verarbeitet, braucht man im Prinzip $N \times M$ Prozessorelemente. Unterteilt man jedoch das ganze Bild in kleinere Bilder, so kann man diese auch sukzessiv verarbeiten. Dabei besteht ein Prozessorelement aus einem programmierbaren Bitprozessor (BP), einem 32-Bit-Speicher (D) und verschiedenen I/O-Leitungen.

Ein Array von diesen Elementen kann im Speicher entweder bis zu 32 Ein-Bit-Bilder oder bis zu vier Grauwertbilder (mit je acht Bit pro Bildpunkt) abspeichern. Ein Dateneingang (A) (entweder von externem oder internem Speicher) und ein Eingang (P) (von internem Speicher oder bestimmt durch eine Funktion der Nachbarn) sind als Eingänge vorhanden. Als Ausgänge sind Leitungen zum internen Speicher (D) und zum Nachbarn (N) vorgesehen. Beide Ausgänge sind mittels zweier Operationscode-Eingänge (Fd, Fn) separat programmierbar. Eine zusätzliche Programmierung für arithmetische Operationen ist über Anschluss R möglich.

Figur 6 zeigt eine Anwendung des Rechners für Bildverarbeitung: Eine schwarze Figur (dargestellt durch «1») auf weissem Hintergrund («0») wird so verarbeitet, dass nur die Grenzelemente der Figur übrigbleiben (Contour Extraction). Diese Operation benötigt für das ganze Bild vier Schritte. Der Prozessor eignet sich für Ein-Bit-Bilder ziemlich gut, ist jedoch für Grauwertbilder nicht optimal. Es hat bis jetzt an einer einigermassen guten Implementation für ein ganzes Bild gefehlt, da ein solches ohne weiteres eine Million Bildpunkte umfassen kann.

7. FLIP (Flexible Image Processor)

Dieses System besteht aus 16 Mikroprozessoren, die mittels eines ausgeklügelten Bussystems miteinander verbunden werden können. Jeder Prozessor hat zwei Eingangsgatter, die entweder an einem von zwei Speicherausgängen oder an einem beliebigen Ausgang eines anderen Prozessors angeschlossen werden können. Um eine optimale Geschwindigkeit zu erreichen, hat jeder Prozessor einen privaten Ausgangsbus, um Buszuteilungen und die damit verbundenen Probleme ausklammern zu können.

Diese Struktur ermöglicht die Verarbeitung der Daten entsprechend dem «Rechengraphen» (Fig. 7). Eine Geschwindigkeitserhöhung wird durch Parallelismus in den verschiedenen Pfaden des Graphen erreicht.

Die grosse Flexibilität des Rechners wird durch die komplizierte und wiederholte Adressierung für den Speicher bei Bildverarbeitungsaufgaben leider beeinträchtigt.

8. Schlussbemerkung

Anhand zweier Beispiele sind die Strukturen und ihre Probleme erläutert worden. Der Datenfluss und die Adressierung des Speichers bilden die momentan grössten Schwierigkeiten

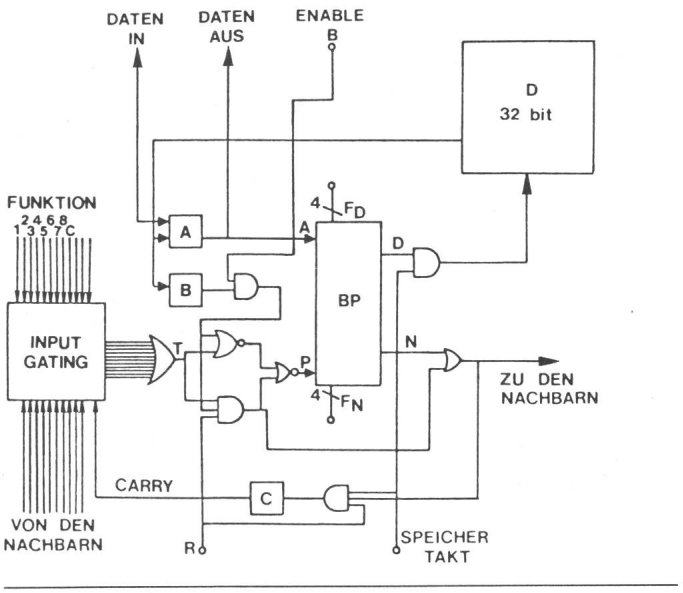

Fig. 5 Ein CLIP-4-Prozessorelement

contour extraction:

picture 'in' as data: (A)	Bn=¬A (N)	'or' of all neighbors (P)	Bd=P.A (D)
0 0 0 0 0 0 0	1 1 1 1 1 1 1	1 1 1 1 1 1 1	0 0 0 0 0 0 0
0 1 0 0 1 0 0	1 0 1 1 0 1 1	1 1 1 1 1 1 1	0 1 0 0 1 0 0
0 1 1 1 1 1 0	1 0 0 0 0 0 1	1 1 1 1 1 1 1	0 1 1 1 1 1 0
1 1 1 1 1 1 0	0 0 0 0 0 0 1	1 1 0 0 1 1 1	1 1 0 0 1 1 0
0 1 1 1 1 0 0	1 0 0 0 0 1 1	1 1 1 1 1 1 1	0 1 1 1 1 0 0
1 0 0 0 0 0 0	0 1 1 1 1 1 1	1 1 1 1 1 1 1	1 0 0 0 0 0 0

Fig. 6 Contour Extraction

Die Figur umfasst einen Ausschnitt aus einem Bild. Für jedes Bit steht ein Prozessor zur Verfügung. In einem *ersten Schritt* wird jeder Prozessor über seinen Dateneingang A mit «seinem» Bit geladen. Über Ausgang D wird das Bit in den Speicher D geschrieben, zwecks weiterer Verwendung. Am Ausgang N wird das inverse Bit erzeugt. Im *zweiten Schritt* wird über den Block «input gating» mittels der logischen «Oder»-Funktion am Eingang P des Bitprozessors BP ein «Oder» aller Nachbarn des Prozessors erzeugt. Im *dritten Schritt* wird dieser Eingang P mittels der logischen «Und»-Funktion mit dem Originalbit (das seit dem ersten Schritt im Speicher steht) verknüpft und das Ergebnis in den Speicher geschrieben. Im *vierten Schritt* wird das Ergebnis aus dem Speicher zum Datenausgang geleitet

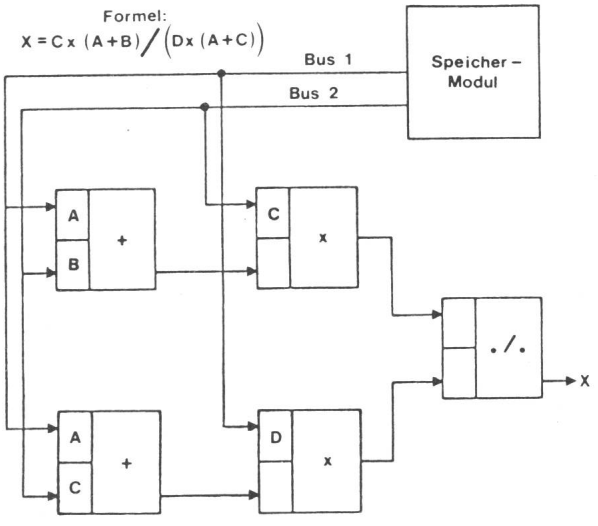

Fig. 7 Aufbau von fünf FLIP-Elementen gemäss Rechengraph

und stehen einer effizienten Echtzeitverarbeitung bei noch tragbaren Kosten im Wege.

Der Verfasser arbeitet zurzeit daran, unter anderem durch tiefgehende Analyse der erwünschten Datenflüsse, einige Verbesserungen zu erreichen. Interessenten, die sich weiter orientieren möchten, sei der Artikel von *K. J. Thurber* [8] empfohlen.

Literatur

[1] *M. J. Flynn:* Some computer organizations and their effectiveness. IEEE Trans. C 21(1972)9, p. 948...960.

[2] *M. J. B. Duff* a.o.: Special computer architectures for pattern recognition and image processing. Proceedings of the 5th International Conference on Pattern Recognition, Miami Beach, Florida, 1...4 december 1980; vol. 2, p. 510...515.

[3] *W. H. Kautz:* An augmented content-addressed memory array for implementation with large-scale integration. Journal of the Association for Computing Machinery 18(1971)1, p. 19...33.

[4] *L. D. Wald:* An associative memory using large-scale integration. Proceedings of the National Aerospace Electronics Conference, New York, May 18...20, 1970; p. 277...281.

[5] *R. R. Kessler* a.o.: Development of an LSI associative processor. US National Technical Information Service, Air Force Report, No. AFAL-TR-70-142, 1970.

[6] *M. J. B. Duff:* Review of the CLIP image processing system. National Computer Conference 1978, Anaheim/California, June 5...8, 1978. AFIPS Conference Proceedings 47(1978).

[7] *P. Gremmar, I. Ischen* and *K. Luetjen:* FLIP: A multiprocessor system for image processing. Karlsruhe, Forschungsinstitut für Informationsverarbeitung und Mustererkennung.

[8] *K. J. Thurber* and *L. D. Wald:* Associative and parallel processors. Computing Surveys 7(1975)4, p. 215...255.

Adresse des Autors

A. P. J. Engbersen, IBM Zürich Forschungslaboratorium, 8803 Rüschlikon, und Institut für Automatik, ETH Zürich, 8092 Zürich.