Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 79 (1988)

Heft: 7

Artikel: Synchroner Datenflussrechner zur Echtzeitbildverarbeitung

Autor: Gunzinger, A. / Mathis, S. / Guggenbühl, W.

DOI: https://doi.org/10.5169/seals-904017

Synchroner Datenflussrechner zur Echtzeitbildverarbeitung

A. Gunzinger, S. Mathis, W. Guggenbühl

Ein an die Bildverarbeitung angepasstes Datenflussrechnerkonzept wird vorgestellt. Dabei wird der statische Datenflussgraph auf einem programmierbaren Mehrprozessorsystem nachgebildet. Dank einem am Institut für Elektronik entwickelten verteilten Netzwerk kann die Zuordnung der Knoten des Datenflussgraphen zu den einzelnen Prozessorelementen automatisiert werden. Der Rechner kann in der Echtzeitbildverarbeitung wie z. B. in Fahrzeugsteuerungen oder in der Robotertechnik eingesetzt werden.

Le concept d'un calculateur à circulation de données adapté au traitement d'images est présenté. Le graphe du flux de données statique est reproduit sur un système à multiprocesseur programmable. Grâce à un réseau développé à l'Institut d'électronique il est possible d'automatiser l'affectation des nœuds du graphe aux divers éléments du processeur. Le calculateur peut s'utiliser dans le traitement d'images en temps réel, par exemple dans les commandes de véhicules ou dans la robotique.

Adresse der Autoren
Dipl. Ing. ETH *Anton Gunzinger*,
Dipl. Ing. ETH *Severin Mathis*
und Prof. Dr. *Walter Guggenbühl*,
Institut für Elektronik, ETH-Zentrum,
8092 Zürich.

Ziel dieses Projekts sind die Konzeption und der Aufbau eines Echtzeitbildverarbeitungssystems, das einerseits dieselbe Rechenleistung erbringt wie problemspezifische Hardware, anderseits aber programmierbar ist und in weiten Grenzen an Probleme der Echtzeitbildverarbeitung angepasst werden kann. Der Anwender sollte dabei stets auf einer ihm angepassten Softwareebene mit dem System kommunizieren können. Die Arbeit umfasst neben der Konzeption der Rechnerstruktur auch den Bau eines Funktionsmusters. Dabei ist darauf zu achten, dass sich das vorgeschlagene Konzept gut für die Implementation mit VLSI-Schaltkreisen eignet.

In einer weiteren Phase soll eine an die Bedürfnisse der Echtzeitbildverarbeitung angepasste Anwendersprache definiert sowie die notwendige Systemsoftware konzipiert und realisiert werden. Anhand von realen Anwendungen (z. B. Tracking, autonome Fahrzeugführung) sollen die Einsatzmöglichkeiten des Systems demonstriert werden.

Im Bereich Robotertechnik und Fahrzeugsteuerung besteht ein wachsendes Bedürfnis nach optisch geführten Systemen. Bei einem solchen System liegt das Ausgangssignal des Prozesses als optisches Signal vor (z. B. als Bild der Fahrbahn). Dieses Signal wird zunächst aufbereitet und daraus die interessierende Information durch ein Bildverarbeitungssystem extrahiert (z. B. die Position der Leitlinie). In einem Regler wird ein Vergleich mit dem Sollwert (Gewünschte Ablage zur Mittellinie) durchgeführt und daraus das Steuersignal für den Prozess (z. B. Lenkung des Fahrzeugs) berechnet. Da das Bildverarbeitungssystem im Regelkreis eingebettet ist, muss es die Information sehr schnell (in Echtzeit) und möglichst ohne Totzeit verarbeiten.

1. Problemstellung bei der Echtzeitbildverarbeitung

1.1 Die klassische Struktur von Echtzeitbildverarbeitungssystemen

In einem «klassischen» Bildverarbeitungssystem durchlaufen die Bilddaten nach der Aufnahme und Digitalisierung die Stufen Vorverarbeitung (Verbesserung des Datenmaterials, Normierung, Kodierung), Segmentierung (Einteilung des Bildes in Gebiete mit «gleichen» Eigenschaften), Nachverarbeitung (Korrektur von Segmentierungsfehlern) und Merkmalextraktion (Berechnung von statistischen Merkmalen wie Fläche, Umfang, Momente). Der nachfolgende Block Klassifikation/Szenenanalyse ordnet Teilgebiete zu vorgegebenen Klassen und interpretiert die Szene. Die systemrelevante Information dieser Analyse wird an das Regelsystem weitergegeben. Ein Steuerungsteil kann die Parameter sämtlicher Verarbeitungsstufen verändern, wobei die Parametersätze nur in den Verarbeitungspausen geändert werden.

1.2 Datendurchsatz und Instruktionsrate

Eines der Hauptprobleme bei der Verarbeitung von Videobildern in Echtzeit ist die hohe Datenrate des Videosignals [1]. Bei der Aufnahme können durch die Verwendung mehrerer Kanäle (z. B. Stereovision, Farbbildanalyse) bis zu 100 Millionen Abtastwerte pro Sekunde entstehen. Diese Datenrate wird für die Szenenanalyse

auf einige 1000 Abtastwerte pro Sekunde reduziert. Geht man davon aus, dass die Vorverarbeitung bis zu 100 Instruktionen pro Datenwert beinhaltet und dass sich diese Anzahl für die Klassifikation etwa um den Faktor 10 erhöht, so werden für die Szenenanalyse/Klassifikation Rechenleistungen von etwa 0,1 bis 1 MIPS (Millionen Instruktionen pro Sekunde) benötigt. Die Instruktionsrate für die vorangehende Aufnahme und Merkmalextraktion (100 Millionen Abtastwerte) ist jedoch beträchtlich höher und beträgt bis zu 10 GIPS (Milliarden Instruktionen pro Sekunde). Während für die Szenenanalyse/Klassifikation bereits Rechnerstrukturen mit genügender Rechenleistung erhältlich sind (z.B. Signalprozessoren, Transputer), stehen programmierbare Systeme für die Bildaufnahme/Merkmalextraktion noch aus.

1.3 Anwendersicht

Da der Endverbraucher immer höhere Anforderungen an ein Computersystem stellt, steigen die Anwenderprogrammkosten stetig an. Bereits beim Entwurf eines neuen Systems sollte nach Möglichkeiten gesucht werden, wie die Erstellungskosten von Anwendersoftware verringert werden können. Die folgenden Massnahmen können zur Reduktion dieser Kosten führen:

- anwendungsnahe Benutzersprache,
- klar strukturiertes Systemmodell,
- hochwertige Übersetzungsprogramme (Compiler),
- automatische Zuordnung der Teilaufgaben an die Prozessorelemente bei Mehrprozessorsystemen
- hochwertige Testhilfsmittel auf der Ebene der Benutzersprache.

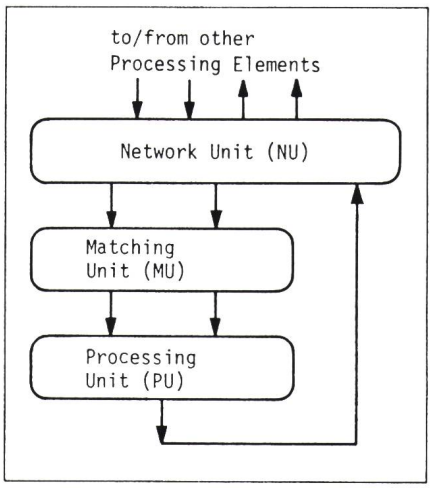
Um diesen Anforderungen gerecht zu werden, sollte der Anwender das Verhalten des Systems (die Algorithmen) in einem ihm vertrauten mathematischen Formalismus beschreiben können. Für die Kommunikation des Systems mit seiner Umgebung (Videosignal-Ein- und -Ausgänge, Tastatur, Maus, Bildschirm) muss die Benutzersprache die entsprechenden Elemente zur Verfügung stellen.

2. Rechnerkonzept

2.1 Grundidee

Die Grundidee des hier vorgestellten Konzeptes besteht in der direkten Nachbildung von Software durch Hardware. Dabei wird aus der formalen Systemspezifikation zuerst ein statischer Datenflussgraph erzeugt. Dieser wird dann anschliessend durch Hardware nachgebildet, indem jeder Knoten bzw. jede Knotengruppe durch ein Prozessorelement ersetzt wird. Jedes Prozessorelement (Fig. 1) besteht aus einer Verarbeitungseinheit (Processing Unit, PU), aus einer Datensynchronisationseinheit (Matching Unit, MU) und aus einer auf die einzelnen Prozessoren verteilten Kommunikationseinheit (Network Unit, NU).

Im folgenden wird dieses Konzept anhand eines einfachen Beispiels, nämlich des Roberts-Operators (eines nichtlinearen, zweidimensionalen Kantendetektors), näher erläutert. Die Figur 2 zeigt die Wirkungsweise des Roberts-Operators: Vom hellen Rechteck (hohe Zahlenwerte) auf dunklem Hintergrund (niedrige Zahlenwerte) werden durch die Bearbeitung mit dem Roberts-Operator im wesentlichen die Objektkanten hervorgehoben.



Figur 1 Struktur eines Prozessorelementes

Dieses besteht aus der eigentlichen Verarbeitungseinheit (Processing Unit, PU), aus einer Datensynchronisationseinheit (Matching Unit, MU) und aus einer auf die einzelnen Prozessoren verteilten Kommunikationseinheit (Network Unit, NU).

a

3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3
3	3	3	10	10	10	10	3	3	3
3	3	3	10	10	10	10	3	3	3
3	3	3	10	10	10	10	3	3	3
3	3	3	10	10	10	10	3	3	3
3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3

b

	0	0	0	0	0	0	0	0	0
	0	0	7	14	14	14	7	0	0
	0	0	14	0	0	0	14	0	0
	0	0	14	0	0	0	14	0	0
	0	0	14	0	0	0	14	0	0
	0	0	7	14	14	14	7	0	0
	0	0	0	0	0	0	0	0	0

Figur 2 Wirkungsweise des Robertsoperators

Vom hellen Rechteck (hohe Zahlenwerte) auf dunklem Hintergrund (niedrige Zahlenwerte) werden durch die Bearbeitung mit dem Robertsoperator im wesentlichen die Objektkanten hervorgehoben.

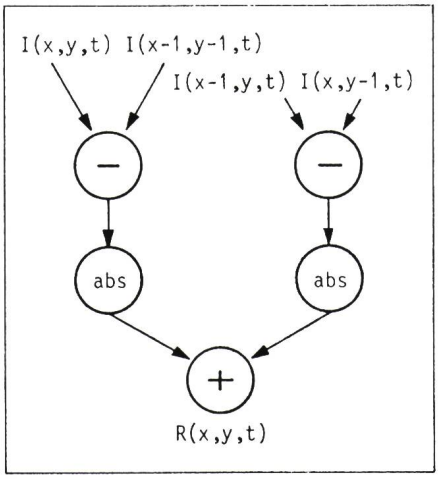
$R(x, y, t) = \text{abs}\,|\,I(x, y, t) - I(x-1, y-1, t)\,| + \text{abs}\,|\,I(x-1, y, t) - I(x, y-1, t)\,|$

a Ursprüngliches Bild $I(x, y, t)$

b Bearbeitetes Bild $R(x, y, t)$

2.2 Systemsoftware

Die Funktion wird zuerst in der Sprache IPL (Image Processing Language), einer an die Bedürfnisse der Echtzeitbildverarbeitung angepassten, für den Anwender leicht verständlichen, formalen Sprache formuliert. Ein besonderer Vorteil dieser Sprache ist, dass sich der in einer Aufgabe vorhandene Parallelismus leicht erkennen lässt, was für die klassischen Programmiersprachen wie Fortran, Pascal, Modula 2 nur eingeschränkt gilt [2; 3]. Die Sprache IPL wurde - ebenfalls im Rahmen dieses Projektes - am Institut für Elektronik entworfen.

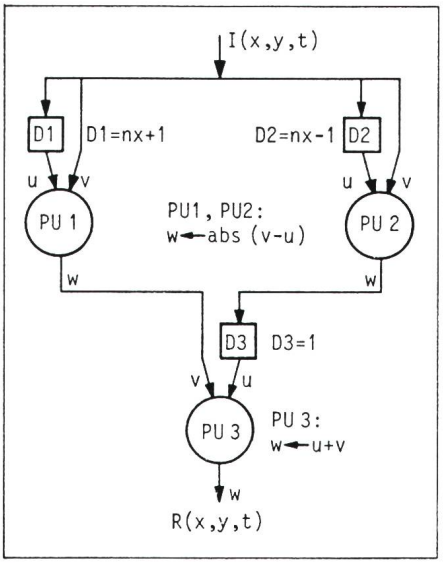


Figur 3 Statischer Datenflussgraph des Robertsoperators

Der Compiler übersetzt das Programm zuerst einmal in einen statischen Datenflussgraphen (Fig. 3). Dieser wird dann in Subgraphen mit zwei Eingängen und einem Ausgang unterteilt, da die Rechenwerke ebenfalls zwei Eingänge und einen Ausgang besitzen, wobei, wie im nächsten Abschnitt beschrieben wird, die Komplexität der Operation keine Rolle spielt. Zuletzt wird jeder dieser Subgraphen durch ein Prozessorelement ersetzt.

2.3 Rechenwerk

Das Rechenwerk oder die Verarbeitungseinheit besteht aus einer Look-up-Tabelle: Für zwei Operanden von beispielsweise 8 Bit Breite wird zur Compilationszeit das Ergebnis jeder möglichen Eingangskombination berechnet und in einem statischen Speicher abgelegt [4]. Bei der Verarbeitung werden nun die zwei Operanden als Adresse interpretiert und an den Speicher angelegt, worauf am Speicherausgang das Ergebnis der Operation erscheint. Diese Technik hat verschiedene Vorteile: Jede beliebige Funktion von zwei Operanden kann berechnet werden, und die «Ausführung» der Operation dauert unabhängig von ihrer Komplexität immer gleich lang. Allerdings benötigt eine Look-up-Tabelle einen relativ grossen Speicherraum (in diesem Fall 64 KByte), weshalb damit keine beliebige Auflösung erreicht werden kann. Die 8-Bit-Genauigkeit ist für viele Anwendungen in der Bildverarbeitung glücklicherweise ausreichend [4].



Figur 5 Vollständige Darstellung des Robertsoperators in Datenflussgraphen

D1, D2, D3	Einstellung der digitalen Schieberegister
PU1, PU2, PU3	Look-up-Tabelle
u, v, w	lokale Datenströme

2.4 Datensynchronisation

Die Kamera wird im sogenannten Line-Scan-Verfahren abgetastet, und die Daten gelangen in gleicher Reihenfolge in das System. Müssen nun zwei Operanden miteinander verknüpft werden, z. B. $I(x,y,t)$ mit $I(x-1,y-1,t)$, so geschieht dies durch digitales Verzögern des Datenstromes in einem einstellbaren Schieberegister. In obigem Beispiel müssen die Daten um eine Zeile und einen Punkt verzögert werden (Fig. 4). Die benötigte Verzögerung kann bereits zur Compilationszeit bestimmt werden. Die Figur 5 zeigt für das Beispiel des Roberts-Operators die Einstellung der digitalen Schieberegister sowie die Look-up-Tabellen mit den zugehörigen Funktionen.

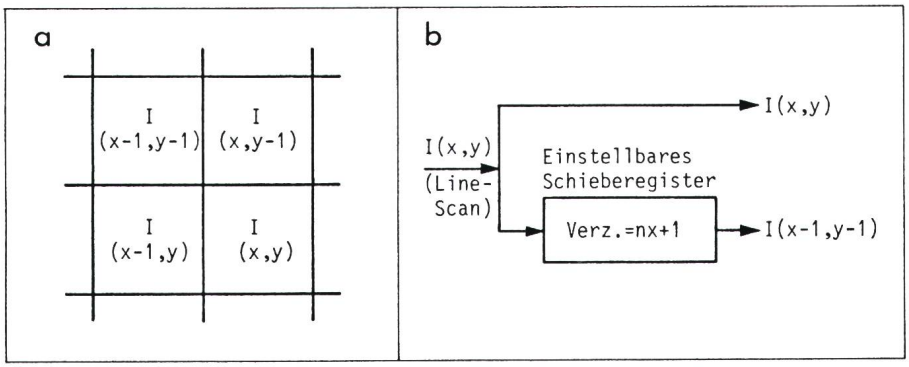
2.5 Prozessorkommunikation

In einem enggekoppelten Mehrprozessorsystem werden die Hardwarekosten durch das Kommunikationsnetzwerk dominiert. Damit der Preis eines Systems linear mit seiner Leistungsfähigkeit (Anzahl Prozessoren) wächst, muss das Kommunikationsnetzwerk auf die verschiedenen Prozessoren verteilt werden. Eine weitere Forderung ist, dass die Leistungsfähigkeit des Kommunikationsnetzwerkes durch Hinzufügen einer beliebigen Anzahl weiterer Prozessorelemente erhöht werden kann.

Die Figur 6 zeigt am Beispiel des Roberts-Operators die Kommunikationspfade. Dabei wird einfachheitshalber angenommen, dass das Ergebnis wieder zu einem (hier neu eingeführten) Prozessorelement PE0 zurückgeführt wird, z. B. zur Darstellung auf dem Bildschirm. Das universellste Kommunikationsnetzwerk ist der Kreuzschienenverteiler (Fig. 7). Leider wächst der Aufwand (Anzahl Knotenpunkte) für dieses Netzwerk proportional mit der Ordnung $O(n^2)$. Für Anwendungen mit sehr vielen Prozessorelementen muss deshalb nach einer anderen Lösung gesucht werden.

In Figur 8 ist der Kreuzschienenverteiler etwas anders gezeichnet; die einzelnen Schaltelemente sind jetzt lokal einem Prozessor zugeordnet. Jeder Prozessor kann dabei seinen Ausgang auf einen beliebigen Bus schalten. Bei genauer Betrachtung fällt auf, dass z. B. der zweite Bus (von unten) nur zur Übertragung von Daten von PE 2 auf PE 3 benötigt wird. Wenn es möglich wäre, den Bus zu unterbrechen, so könnte er gleichzeitig mehrmals zur Übertragung von Daten ausgenutzt werden. Von dieser Möglichkeit wurde in Figur 9 Gebrauch gemacht. Anstelle von 4 werden für die Nachbildung desselben Netzwerks jetzt nur noch 2 Busse benötigt. Wird dieses Netzwerk weiter untersucht, so wird für bekannte Algorithmen minimal die in Tabelle I angegebene Anzahl Busse benötigt.

Das bedeutet, dass sich viele Algorithmen der digitalen Signalverarbeitung mit Hilfe einer solchen Busstruktur implementieren lassen. Anstelle der in Figur 9 gezeichneten Schalter

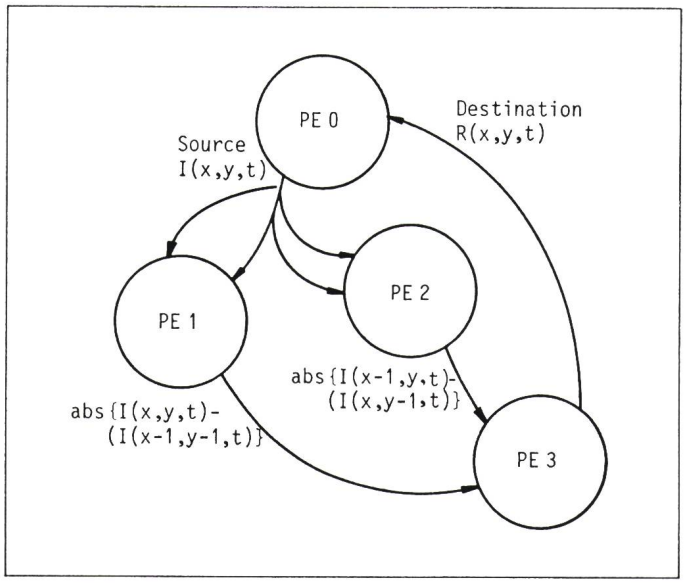


Figur 4 Datensynchronisation

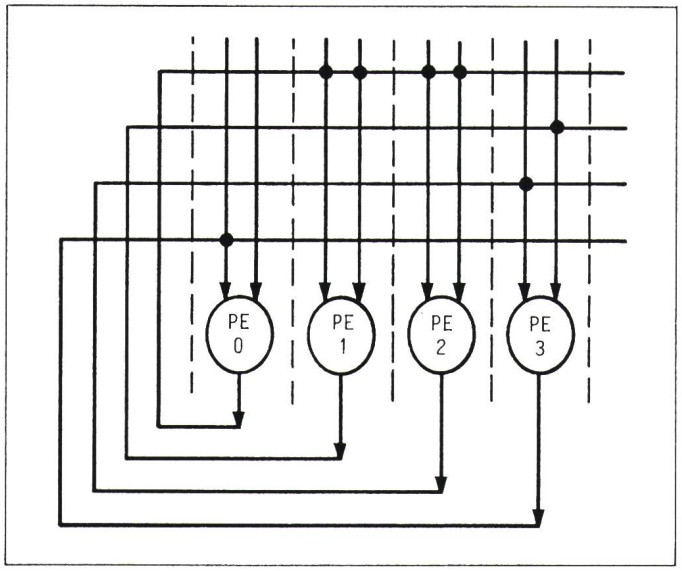
Durch Verzögern des Datenstroms in einer digitalen Verzögerungsleitung (b) der Länge $nx+1$ (nx: Punkte pro Zeile) wird erreicht, dass die Operanden $I(x,y)$ und $I(x-1,y-1)$ gleichzeitig an die Recheneinheit gelangen.

a Bildfeld

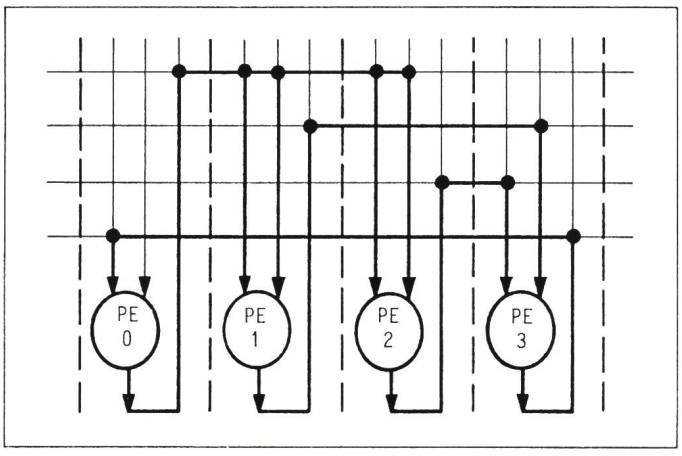
b Verzögerungsschaltung



Figur 6
Kommunikationsgraph des Robertsoperators
PE Processing Element



Figur 7
Implementation des Robertsoperators auf einem universellen Koppelnetzwerk (Kreuzschienenverteiler)
Der Aufwand für das Koppelnetzwerk wächst mit der Ordnung $O(n^2)$



Figur 8
Implementation des Robertsoperators auf verteiltem Netzwerk
Ein solches System kann maximal nur soviele Prozessoren enthalten, wie Busse zur Verfügung stehen. Jede Verbindung blockiert einen ganzen Bus.

werden (als programmierbare Schalter) Multiplexer mit nachgeschaltetem Latch verwendet. Damit können die Daten synchron von einem Prozessorelement zum andern verschoben werden (Pipeline). Weil die Daten auf genau definiertem Pfad vom Sender (Ausgang eines PE) zum Empfänger (Eingang des nächsten PE) weitergeleitet werden, kann eine viel höhere Taktrate als in einem System mit parallelem Bus und einer unbekannten Anzahl von Empfängern erzielt werden. Die sich durch den Transport ergebende Verzögerung kann durch die digitalen Verzögerungsleitungen (Abschnitt 2.4) kompensiert werden, da diese sinnvollerweise mit derselben Taktrate arbeiten.

2.6 Konfiguration

Die Anzahl benötigter Busse ist nicht nur von der Form des Kommunikationsgraphen, sondern auch von der Zuordnung der Aufgaben (Tasks) an die verschiedenen Prozessorelemente abhängig. Unter der Annahme, dass jedes Prozessorelement gleichwertig ist, ergeben sich für einen Algorithmus mit n Prozessoren $(n-1)!$ mögliche Anordnungen. Für das Beispiel Roberts-Operator ist $n = 4$; es gibt also insgesamt 6 mögliche Anordnungen, wie Tabelle II zeigt.

Aus dieser Tabelle geht hervor, dass die Anzahl benötigter Busse stark von der Verteilung der Aufgaben auf die Prozessorelemente abhängt. Da ein System aber für eine definierte Anzahl Busse ausgelegt werden muss, kann es möglich sein, dass eine zufällig gewählte Anordnung nicht implementiert werden kann, weil die Anzahl Busse nicht ausreicht, obwohl es eine andere Konfiguration gibt, deren Implementation keine Probleme macht.

Es ist nun die Aufgabe eines Programms, des sogenannten *Konfigurators*, eine geeignete Zuordnung zu finden, die auf dem System implementiert werden kann. Eine Überprüfung aller Anordnungen ist für Algorithmen, die viele Prozessorelemente benötigen, nicht möglich, da die Anzahl der Möglichkeiten mit $(n-1)!$ zunimmt. Deshalb werden heuristische Verfahren verwendet, die den Aufwand stark reduzieren können. Beispiele solcher Verfahren sind z. B. Subblock-, Traveling-Salesman- und Simulated-Annealing-Optimierung. Es sind dieselben Verfahren, die auch in Print-Layout-Systemen für das Routing verwendet werden.

Sobald eine Konfiguration gefunden ist, die auf dem System implementiert werden kann, muss diese in die Hardware geladen werden, d.h. die Bussschalter, Verzögerungsleitungen und Look-up-Tabellen müssen entsprechend programmiert werden. Anschliessend kann das Programm ausgeführt werden.

3. Realisation

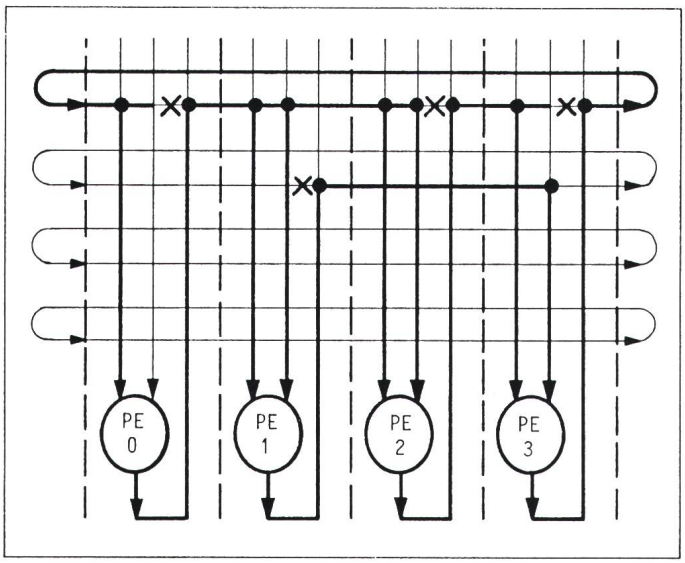
Ein erster Testaufbau erfolgte im Rahmen einer Diplomarbeit am Institut für Elektronik. Dabei wurden herkömmliche LS-TTL-Bausteine verwendet. Ein einzelnes Prozessorelement fand dabei auf einer Dreifach-Europakarte Platz, die Stromaufnahme einer Karte betrug etwa 4 A.

Diese Schaltung wurde überarbeitet und teilweise durch programmierbare Gatearrays (Logic Cell Array, LCA) ersetzt. Die Datensynchronisationseinheit (Matching Unit) wurde in SMD-Technologie aufgebaut. Dadurch konnte die Packungsdichte um den Faktor 3 erhöht und die Stromaufnahme um den Faktor 4 reduziert werden. Eine Integration in Custom-VLSI-Chips könnte das Volumen nochmals um den Faktor 4 bis 8 bei gleichzeitiger Reduktion der Stromaufnahme verringern. Die technischen Daten des Prototyps sind in der Tabelle III zusammengestellt.

Figur 9
Reduktion des Koppelnetzwerkes

● Verbindung
X Unterbruch

Die zu den einzelnen Prozessoreinheiten gehörenden Busanteile können entsprechend der jeweiligen Aufgabe unterbrochen oder durchgeschaltet werden. Für die Berechnung des Robertsoperators zum Beispiel werden damit anstelle von 4 nur noch 2 Busse benötigt.



Algorithmus	Minimale Anzahl Busse
Lineares Array	2
Binärer Baum mit 2^n Eingängen	n
2 n × n-Matrix-Multiplikation	2^n-1
Rekursive Struktur *n*-ter Ordnung	n + 1

Tabelle I Minimale Anzahl Busse für verschiedene Algorithmen

Anordnung der Teilaufgaben	Anzahl benötigter Busse
0 1 2 3	2
0 1 3 2	3
0 2 1 3	2
0 2 3 1	3
0 3 1 2	4
0 3 2 1	4

Tabelle II Die Anzahl benötigter Busse ist von der Zuordnung der Tabellenaufgaben auf die Prozessoren abhängig

4. Anwendungen

Aus Platzgründen muss auf eine ausführliche Beschreibung von Anwendungsbeispielen verzichtet werden. Es sind aber nachfolgend einige der möglichen Algorithmen, die auf dem Rechner implementiert werden können, zusammengestellt.

- ortsabhängige Korrekturen (Shading),
- Lokaloperationen (linear, nichtlinear),
- temporale Operationen (linear, nichtlinear),
- multispektrale Verarbeitung (Farbbildverarbeitung),
- Erzeugung von Fensterfunktionen (Rechteck, Rahmen, Kreis, Kreisring),
- Erzeugung von Verteilungsfunktionen als Funktion des Ortes und der Zeit,
- Bildsegmentierung,
- Nachverarbeitung (Kantendetektion, Erosion, Dilatation, nichtlineare Rauschunterdrückung),

Tabelle III
Technische Daten des Datenflussrechnerprototyps

Verarbeitungsleistung:	50 Bilder/Sekunde, 256×256 Bildpunkte
Maximale Anzahl Prozessorelemente:	256
Kommunikationsnetzwerk:	12 umlaufende Verbindungspfade mit 8-Bit-Auflösung, 10-MHz-Datenrate
Datensynchronisation:	Verzögerung von 1...4096 Bildpunkten einstellbar. Damit können beliebige Daten, die innerhalb von 16 Bildzeilen liegen, miteinander verknüpft werden.
Prozessorelementtypen:	A/D- und D/A-Wandler, Filterprozessor, Korrelator, «Look-Up-Table»-Prozessor, Binärprozessor, Statistikprozessor
Klassifikationsrechner:	IBM-XT oder AT, Signalprozessor TMS 320C25
Programmentwicklungssystem:	IBM-XT- oder -AT-kompatibler Rechner
Programmentwicklungsumgebung:	Compiler, Konfigurator, Loader, Interpreter, Hardwaremonitor, Debugger und Simulator

- Merkmalsextraktion (Fläche, Umfang, Momente 1. und 2. Ordnung, Histogramme).

Diese Algorithmen oder beliebige Kombinationen davon können selbstverständlich in Echtzeit (50 Bilder pro Sekunde) abgearbeitet werden.

5. Ausblick

Das vorgestellte Konzept ist vielversprechend für Anwendungen mit relativ wenig Instruktionen und grossen Datenmengen bzw. grossem Datendurchsatz. Die Echtzeitbildverarbeitung ist eine solche Anwendung. Bekannte Bildverarbeitungsalgorithmen können mit diesem System 100- bis 1000mal schneller abgearbeitet werden als auf herkömmlichen Systemen. Beispielsweise benötigt der Farbbildklassifikator zur autonomen Fahrzeugführung auf dem WARP-Rechner [5] (gilt in den USA als sehr erfolgversprechende Architektur) 5 Sekunden zur Auswertung eines einzelnen Bildes. Derselbe Algorithmus kann auf dem vorgestellten Rechner in 20 ms, also rund 250mal schneller, abgearbeitet werden.

Das vorgestellte Konzept ist jedoch auch von der Programmentwicklungsumgebung aus gesehen äusserst interessant: Der Anwender kann die Aufgaben an das System auf sehr hoher Ebene definieren (Spezifikationsebene). Die Systemsoftware übersetzt die Spezifikation nicht nur in ein lauffähiges Programm, sondern verteilt dieses automatisch auf die zur Verfügung stehende Hardware. Ein Debugger sollte die Untersuchung von Programmen ebenfalls auf der Spezifikationsebene erlauben. Auf diese Aspekte kann aus Platzgründen nicht eingegangen werden, doch sei hier auf weitere Publikationen über diese Rechnerarchitektur verwiesen [6; 7; 8].

Da das vorgestellte Konzept von der Datenbreite und den verwendeten Rechenwerken unabhängig ist, liesse sich unter Zuhilfenahme entsprechender Rechenelemente auch ein äusserst leistungsfähiges System für 32/64-bit-Gleitkomma-Arithmetik realisieren. Damit könnten auch Algorithmen aus dem Gebiet der Computersimulation und des CAD (Computer-Aided Design) effizient bearbeitet werden.

Schliesslich könnten durch die Integration des Systems in Custom-VLSI-Chips die Stromaufnahme, das Volumen und auch die Kosten beträchtlich gesenkt werden. Das System würde damit auch im industriellen Umfeld Interesse finden.

Literatur

[1] *S. Yalamichili* a.o.: Image processing architectures: A taxonomy and survey. In: Progress in pattern recognition. Volume 2. Amsterdam, North-Holland, 1985.

[2] *E.J. Lerner:* Data-flow architecture. IEEE Spectrum 21(1984)4, p. 57...62.

[3] *A.L. Davis* and *R.M. Keller:* Data flow program graphs. IEEE Computer 15(1982)2, p. 26...41.

[4] *H.J. Keller, A. Favre* and *A. Comazzi:* VAP a video array processor using cascaded look-up tables and its applications in biomedicine. Proceedings of SPIE (The international Society of Optical Engineering). Vol. 397: Applications of digital image processing; p. 406...414.

[5] *H.T. Kung:* Systolic algorithms for the CMU Warp processor. Proceedings of the seventh International Conference on Pattern Recognition, Quebec, July 30–August 2, 1984; vol. 1, p. 570...577.

[6] *T. Gunzinger:* Synchroner Datenflussrechner zur Echtzeitbildverarbeitung. Mustererkennung 1986. 8. DAGM-Symposium (Deutsche Arbeitsgemeinschaft für Mustererkennung). Informatik-Fachberichte 125. Berlin u. a, Springer-Verlag, 1986; S. 123...128.

[7] *A. Gunzinger, S. Mathis* and *W. Guggenbühl:* Datenflussrechner Echtzeitbildverarbeitung: Softwareentwicklungsumgebung. Mustererkennung 1987. 9. DAGM-Symposium (Deutsche Arbeitsgemeinschaft für Mustererkennung) Informatik-Fachberichte 149. Berlin u. a., Springer-Verlag, 1987; S. 34...39.

[8] *A. Gunzinger, S. Mathis* and *W. Guggenbühl:* A reconfigurable systolic array for real-time image processing. Proceedings of the IEEE International Conference in Acoustics, Speech and Signal Processing (ICASSP) 1988.