Untersuchungen zu Spezialhardware für ausgewählte Algorithmen der Bildverarbeitung

Günter Kemnitz
Institut für Technische Informatik
Technische Universität Dresden
kemnitz@ite.tu-dresden.de

Kurzfassung

Am Beispiel linearer Nachbarschaftoperationen, der schnellen Fouriertransformation und der Histogrammberechnung wird ein sehr schnelles Rechenwerk konzepiert, daß nach den Prinzipien eines Vektorrechners – Hardware-Steuerung der inneren Schleifen und Ausführung der Basisoperationen in einer schnellen Pipeline – arbeitet. Das Ergebnis ist eine speicherprogrammierbare Matrix aus Vektorregistern, multifunktionalen Arithmetikbausteinen, Multiplizieren, programmierbaren Logikblöcken und einer speziellen Verbindungsarchitektur. Das Konzept stellt einen interessanten Lösungsansatz für zukünftige Graphikprozessoren dar.

0 Einleitung

Der Bedarf an Rechenleistung für die Echtzeitbildverarbeitung liegt mit mehreren Milliarden Operationen pro Sekunde noch um Größenordnungen über der nutzbaren Leitung heutiger Prozessoren [1]. Nur Mehrprozessorsysteme und Spezialschaltkreise erreichen diese Leistung. Beide Konzepte sind sehr aufwendig. Mehrprozessorsysteme verlangen einen apriori hohen Schaltungsaufwand, sind schwer zu programmieren und in vielen Fällen nur für ausgewählte Algorithmen geeignet [2]. Algorithmenspezifische Schaltkreise arbeiten in der Regel synchron zur Bildsignalbereitstellung und verlangen vereinfacht einen Baustein je Operation [3].

Die Vision des zukünftigen Bildverarbeitungsrechners ist eine universelle Architektur, die auf einem kleinen Chipsatz integriert werden kann. Eine Prognose in [1] geht davon aus, daß es sich um ein heterogenes Mehrprozessorsystem handelt, in dem jeder Prozessor eine sehr hohe Rechenleistung besitzt und für spezielle Algorithmenklassen optimiert ist. Die hier vorgestellte Studie entwickelt den Kern eines solchen Prozessors.

Algorithmenspezifische Befehle und Verarbeitungswerke erlauben, die Rechenleistung eines Prozessors auf Kosten der Universalität zu erhöhen. Die Spezialisierung von Bildverarbeitungsbefehlen reicht z.B. von kombinierten Multiplikations-/Additionsbefehlen für die Konstruktion von Filter- und Transformationsfunktionen bis zur kompletten Filterfunktion, Fourier- oder Kosinustransformation.

Ein interessanter Kompromiß sind Rechenwerke mit rekonfigurierbarem Datenfluß. Ein Satz von Grundfunktionen z.B. Addition, Multiplikation und Tabellenumrechnung (LUT, look-up table) wird mit einem programmierbaren Verbindungsnetzwerk zu einer Spezialoperation verschalten [3]. In Bild 1 ist das erste Datenflußelement Q die Informationsquelle oder die HOST-Schnittstelle. Es stellt die Grauwerte eines Bildfensters von 2×2 Punkten bereit. Die übrigen Elemente sind programmierte Tabellenfunktionen, die in der gezeigten Konfiguration die Subtraktion mit Betragsbildung und die Addition nachbilden. Das Ergebnis gelangt zurück zur HOST-Schnittstelle. Ohne die Flexibilität eines elementaren Befehlsvorrates aufzugeben, wird der potentielle Rechenzeitgewinn stark spezialisierter Befehle genutzt.

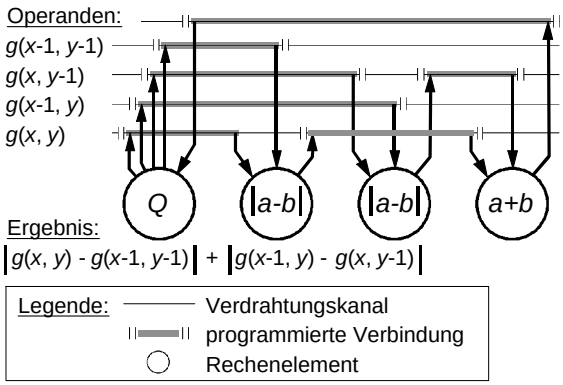


Bild 1: Programmierbare Verarbeitungseinheit [3]

Neue Möglichkeiten zur Realisierung programmierbarer Verarbeitungswerke bieten speicherprogrammierbare Logikbausteine. Funktion und Anschlußbelegung dieser Bausteine werden von einem Konfigurationsprogramm festgelegt [4].

Die vorliegende Studie untersucht ein an die Technik der programmierbaren Verarbeitungswerke angelehntes Konzept. Es werden zusätzlich Vektorregister und Elemente für ein freiprogrammierbares Steuerwerk in das Rechenwerk aufgenommen. Das erlaubt, einfache Prozeduren, insbesondere die inneren Schleifen von rechenzeitintensiven Algorithmen, im Rechenewerk in einer Pipeline auszuführen. Die Idee ist den Vektorrechnern entlehnt. Vektorrechenwerke besitzen, wie im folgenden Abschnitt dargelegt, für

ausgewählte Aufgabenstellungen eine sehr hohe Rechenleistung, die sonst nur mit Mehrprozessorsystemen erreicht wird. Der Gegenstand der Studie, ist diese potentielle Rechenleistung auch für die Bildverarbeitung nutzbar zu machen.

Das Rechenwerk wird in mehrere Iterationen entwickelt. Am Beispiel linearer Nachbarschaftsoperationen entsteht eine erste Architektur. Die sogenannte Butterfly-Operation der schnellen diskreten Fouriertransformation (vgl. Bild 8 und Bild 9) zeigt, daß sich die Hardware wesentlich vereinfachen läßt, wenn die Ablaufsteuerung den Datenfluß dynamisch verändern kann. Zur Histogrammbrechnung ist eine datenabhängige Umschaltung von Datenpfaden erforderlich. Aspekte der Layout-Gestaltung und der Verdrahtung führen zu einer Array-Struktur. An dieser Stelle bricht die Studie ab. Im Ausblick wird darauf verwiesen, daß das Rechenwerk für wesentlich mehr als die drei untersuchten Algorithmen geeignet ist.

1 Rechengeschwindigkeit

Eine Kette ist nicht besser als ihr schwächstes Glied. Das schwächste Glied in Bezug auf die Geschwindigkeit eines Rechner ist der Speicher, in dem die abzuarbeitenden Befehle und Daten stehen. Typisch für heutige Rechner ist, daß der Prozessor mehr als zehnmal so schnell Befehle abarbeiten könnte als die dRAM-Schaltkreise des Hauptspeichers den Befehlskode bereitstellen. Diese Kluft liegt in der Architektur des von Neumann Rechners und im Aufbau der Speicherschaltkreise begründet.

Schnelle Prozessoren können ihre Verarbeitungsgeschwindigkeit nur im Zusammenhang mit Cache-Speichern nutzen (Bild 2). Der Cache, ein schneller, kleinerer assoziativer sRAM-Speicher, kann etwa 90% der Speicheranforderungen ohne Verzögerung erfüllten [5]. Für die verbleibenden 10% muß der Prozessor auf den Hauptspeicher zugreifen. Während dieser Zeit von angenommen 10 Befehlsholezyklen wird der Prozessor angehalten. Bei den hier unterstellten Zahlen würde der Cache die Auslastung des Prozessors von 10% auf 50% erhöhen.

Der Anteil der Zeit, die der Prozessor auf seinen Speicher wartet, wächst tendentiell. Dieser ungünstige Trend kann nur durch grundlegende Neuerungen in der Rechner-, Speicher- und Befehlsarchitektur aufgehalten werden [6].

Ein wesentliches Merkmal aller schnellen Prozessoren ist die Pipeline-Technik. Im Idealfall beginnt mit jedem Takt, in dem der Prozessor nicht auf seinen Speicher wartet, eine neue Operation. Die vorherige Operation wird weitergeführt und die um p Takte zuvor gestartete Operation abgeschlossen (Bild 2). Aufgrund von Daten-, Steuer- und Ressourcenkonflikten kann nur ein Teil der Zeitscheiben einer Verarbeitungs-Pipeline genutzt werden [5]. Unter der Annahme, daß 50% der Zeitscheibe durch Konflikte verloren gehen und die Wartezeit auf den Speicher auch 50% beträgt, ist praktisch nur ein Viertel der theoretisch verfügbaren Rechenleistung nutzbar.

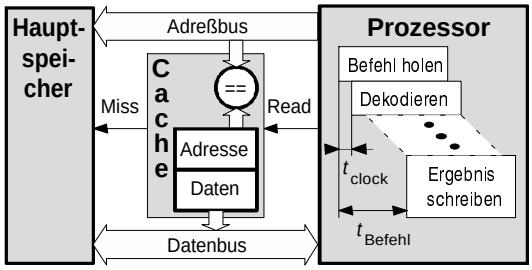


Bild 2: Kern eines schnellen Mikrorechners

Vektorrechenwerke vermeiden Wartezeiten und Konflikte, und sie führen Operationen der Kategorie "Wiederhole für alle Vektorelemente ..." wesentlich schneller aus als ein vergleichbarer skalarer Prozessor. Das ist auf folgende Prinzipien zurückzuführen:

- Realisierung der Schleife "Für alle Vektorelemente ..." in Hardware
- Prozessorregister für Vektoren oder Teilbereiche aus Vektoren (Vektorregister)
- feine Stufung der Verarbeitungs-Pipeline und hohe Taktfrequenz
- gute Pipeline-Auslastung.

Die Hardware Schleife "Wiederhole für alle Vektorelemente ..." reduziert die Anzahl der nacheinander auszuführenden Verarbeitungsschritte. Das Weiterzählen von Indexregistern, das Laden von Operanden, das Abspeichern von Ergebnissen, bedingte Sprünge zur Bildung von Schleifen, etc. werden von der Hardware zeitgleich zur eigentlichen Verarbeitung ausgeführt. Die hierfür notwendigen Befehlsholezyklen und Zeitscheiben im Pipeline-Fluß entfallen. Auch Wartezeiten auf den Hauptspeicher, wenn der Code zum ersten Mal aufgerufen und in den Cache geladen wird, und Verluste durch Daten-, Steuer- oder Ressourcenkonflikte werden vermieden.

Das zweite Prinzip – Vektorregister mit einer ausreichenden Kapazität – ist Voraussetzung für eine unterbrechungsfreie Arbeit des Vektorrechenwerkes. Bei der Verarbeitung von Vektoren und Feldern mit einem skalaren Prozessor stehen die Vektorelemente im Hauptspeicher bzw. im Cache. Es kann zu einem Zeitpunkt nur ein Operand gelesen oder eine Ergebnis geschrieben werden. Eine Operation mit o Operanden und e Ergebnissen benötigt mindestens $o+e$ Cache- bzw. Hauptspeicherzyklen.

Für die Vektorverarbeitung werden die Operanden und Ergebnisse in den prozessorinternen Vektorregistern gespeichert. Es können zeitgleich mehre-

re Operanden gelesen und gegebenfalls mehrere Ergebnisse geschrieben werden. Damit kann auch aus der Sicht der Datenbereitstellung in jedem Takt eine Operation begonnen und eine abgeschlossen werden.

Die Wartezeiten eines Vektorrechenwerks auf den Hauptspeicher lassen sich sich vergleichsweise gering halten. Der Transfer erfolgt in Blöcken. Das bietet zahlreiche Ansatzpunkte für eine beschleunigte Übertragung (z.B. parallele Übertragung mehrerer Daten, überlappte Adressierung und Pipeline-Stufen im Übertragungsweg) [8]. In den Vektorregistern werden Zwischenergebnisse gespeichert, so daß der Datentransfer mit dem Hauptspeicher insgesamt geringer ist. Das Laden und Speichern der Vektoren kann ohne Ressourcen-Konflikte zeitgleich zur eigentlichen Verarbeitung erfolgen.

Die Verfeinerung der Pipeline erlaubt, die Taktfrequenz des Rechenwerkes zu erhöhen. Eine Aufteilung einer Register-Transfer-Funktion mit einer Laufzeit t_{v_max} in zwei Register-Transfer-Funktionen durch ein Pipeline-Register erlaubt bei einer optimalen Wahl der Trennstelle die doppelte Taktfrequenz. Die Ausführungszeit je Operation bleibt zwar gleich oder sie erhöht sich geringfügig. Aber es können nun zwei statt vorher nur eine Operation zeitgleich bearbeitet werden (Bild 3).

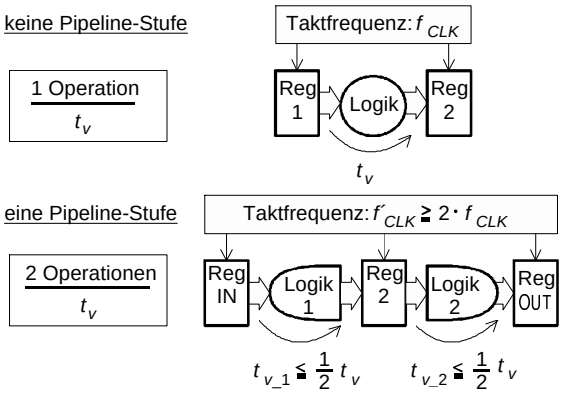


Bild 3: Erhöhung der Rechenleistung durch zusätzliche Pipelinestufen

Dieses Prinzip läßt sich mit den beiden Teilfunktionen fortsetzen. Ein Rechenwerk mit *p* Pipeline-Stufen und der *p*-fachen Taktfrequenz gestattet nahezu den selben Verarbeitungsdurchsatz, wie *p* parallel betriebene Rechenwerke ohne Pipeline [7]. Nur ist der Schaltungsaufwand gegenüber einer Parallelverarbeitung mit mehreren Werken geringer.

Das Potential zur Parallelverarbeitung mit einer Pipeline kann nur von Algorithmen mit einem entsprechend hohen Maß an feinkörniger Parallelität genutzt werden. Die Merkmale feinkörniger Parallelität sind im Vergleich zur Pipeline-Tiefe lange Befehlssequenzen ohne Steuer- und Datenkonflikte. Skalare Programme rechtfertigen nach zahlreichen Untersuchungen nur Pipeline-Tiefe von 4 bis 8 [5], [7]. Längere Befehlssequenzen ohne bedingte Verzweigungen und ohne Datenkonflikte sind untypisch.

Vektoroperationen sind im Gegensatz dazu so aufgebaut, daß innerhalb der Schleife "Wiederhole für alle Vektorelemente ..." keine Konflikte auftreten. Mit Ausnahme einer Anlaufzeit, die gleich der Pipeline-Tiefe *p* ist, wird in jedem Takt eine Operation abgeschlossen. Die Pipeline-Auslastung beträgt in Abhängigkeit von der Anzahl der Basisoperationen einer Vektoranweisung *n*:

$$\eta = \frac{n}{n+p} \qquad (1)$$

Daraus leitet sich eine wichtige Regel für die Vektorverarbeitung ab. Die Anzahl der Basisoperationen einer Vektoranweisung *n* soll die Pipeline-Tiefe *p* um mindestens eine Größenordnung übersteigen.

Im Vergleich zur Rechenleistung eines skalaren Prozessors ergibt sich für die Vektorverarbeitung folgender Überschlag. Allein die Vermeidung von Konflikten und Wartezeiten während der Befehlsholezyklen erhöht die nutzbare Rechenleistung um den Faktor 4. Mit einer verdoppelten Taktfrequenz ergibt sich die achtfache Rechenleistung. Die Basisoperation einer Vektoroperation (z.B. einer Vektoraddition) entspricht 6 RISC-Befehlen (Zählen des Schleifenzählers, zwei Load-Operationen, die Addition, eine Store-Operation und ein bedingter Sprung). Multipliziert mit 8 ergibt sich die 48-fache Geschwindigkeit eines skalaren RISC-Prozessors mit vergleichbarer Schaltungstechnik..

In den weiteren Abschnitten wird gezeigt, daß für die Bildverarbeitung komplexere Basisoperationen benötigt werden, die nicht nur 6, sondern bis zu 20 oder 30 RISC-Befehlen entsprechen. Dadurch läßt sich insgesamt ein Beschleunigung bis um zwei Größenordnung erzielen.

Eine vektorielle Verarbeitung im weiteren Sinne verlangt daß die *n*-mal zu wiederholende Operationsfolge einschließlich der Schleifensteuerung in einer Pipeline ausgeführt wird. Die Operationsfolge ist in aufeinanderfolgende und nebenläufige Register-Transfer-Funktionen zu zerlegen. Die Basisoperationen sind durch Hardware-Funktionen und ihre Verkettung durch ein entsprechend rekonfiguriertes Verbindungsnetzwerk nachzubilden. In den folgenden Abschnitten wird diese Zerlegung exemplarisch für lineare Nachbarschaftoperationen, die diskrete Fouriertransformation und die Histogrammberechnug durchgeführt. Dabei kristallisieren sich Prinzipien für den Aufbau eines rekonfigurierbaren Rechenwerks heraus.

2 Lineare Nachbarschaftsoperationen

Der Grauwert eines Punktes im Ergebnisbild $g'_{m\,n}$ ist die Summe der Grauwerte des Punktes und seiner Umgebung im Ausgangsbild $g_{m+j\,n+k}$ gewichtet mit den Koeffizienten $c_{j\,k}$:

$$g'_{m\,n} = \sum_{j=x_a}^{x_e} \sum_{k=y_a}^{y_e} c_{j\,k} \cdot g_{m+j\,n+k} \qquad (2)$$

Die Wichtungskoeffizienten $c_{j\,k}$ bilden die Filtermatrix. Lineare Nachbarschaftsoperation werden zur Rauschunterdrückung, zur Kantendetektion, zur Bildreduktion und zur Texturanalyse eingesetzt [9], [10].

Die Basisoperationsfolge besteht aus dem Lesen des Wichtungsfaktors auf der Adresse $(m\ \ n)$ im Matrixspeicher und des Grauwertes auf der Adresse $(x+m-m_a\ \ y+n-n_a)$. Beide Werte werden multipliziert und in einem Akkumulator aufsummiert. Zu Beginn jeder Operation wird der Akkumulatorinhalt nicht auf den Addierer zurückgeführt, sondern in den Ergebnisspeicher auf der Adresse $(x\ \ y)$ verzögert um Δp Takte entsprechend der Differenz der Pipeline-Stufen geschrieben (Bild 4).

Die Vektorregister für das Ausgangs- und das Ergebnisbild und im Sinne eines Bausteinkonzeptes auch das Koeffizientenregister sollen eine Datenwortbreite von einem Byte besitzen. Die Multiplikation führt zu einer Verdoppelung des Wertebereiches. Die zu akkumulierenden Produkte besitzen einen Wertebereich von 2 Byte. Vor der Abspeicherung ist die Summe aller Produkte wieder auf 1 Byte zu reduzieren. Im einfachsten Fall wird das niederwertigste Byte abgeschnitten. Dazu ist eine Skalierung der Summe über die Filterkoeffizient $c_{j\,k}$ erforderlich.

Der Datenfluß von der Adreßbereitstellung über die Speicherzugriffe und das Multiplikationswerk ist frei von Rückführungen. Er kann beliebig durch Pipeline-Register verfeinert werden. Innerhalb der Vektorregister bieten sich die Trennstellen zwischen Dekoder und Speichermatrix an. Ein Multiplizierer kann durch Pipeline-Register in Register-Transfer-Funktionen zur Bildung von Teilergebnissen zerlegt werden.

Der Addierer darf kein Pipeline-Register enthalten, da er sein eigenes Ergebnis als Operand verwendet. Seine Signallaufzeit begrenzt die maximale Taktfrequenz und damit sinnvoller Weise auch die Anzahl der Pipeline-Stufen der anderen Verarbeitungselemente. Es empfiehlt sich, ein sehr schnelles Addierer-Design (z.B. eine Carry-Select-Architektur) und die Zusammenfassung mit dem Akkumulator zu einem gemeinsamen Datenflußbaustein.

Als Vergleichsobjekt für die Abschätzung der Geschwindigkeit soll das Rechnewerk des Alpha-Prozessors dienen [11]. Schnelle Versionen dieses Prozessors benötigen für eine 64-Bit-Addition 5 ns. Die längste Verzögerungszeit in einem Addierer tritt zwischen dem Übertragsein- und -ausgang auf. Mit der gleichen 0,5 µm CMOS-Technologie und der selben Schaltungsarchitektur besitzt ein 16-Bit-Addierwerk voraussichtlich eine entsprechend geringere Verzögerungszeit, angenommen von 2 ns. Das wäre im skizzierten Rechenwerk dann die längste Verzögerungszeit. Das Vektorrechenwerk könnte mit 500 MHz betrieben werden.

Die innere Schleife der Basisoperation würde auf einem RISC-Prozessor mindestens zwei Additionen für die Indexberechnung, zwei Load-Operationen (Grauwert und Koeffizient) der eigentlichen Multiplikation und Addition und einem bedingten Sprung bestehen. Eine Operation des Rechenwerks entspricht 7 herkömmlichen Verarbeitungsschritten bzw. 3,5 Milliarden herkömmlichen Operationen je Sekunde. Wie im letzten Abschnitt gezeigt wird, kann mit den Ressourcen für die Fourietransformation, die auch unterstützt werden soll, vier und mehr Nachbarschaftoperationen gleichzeitig ausführen,

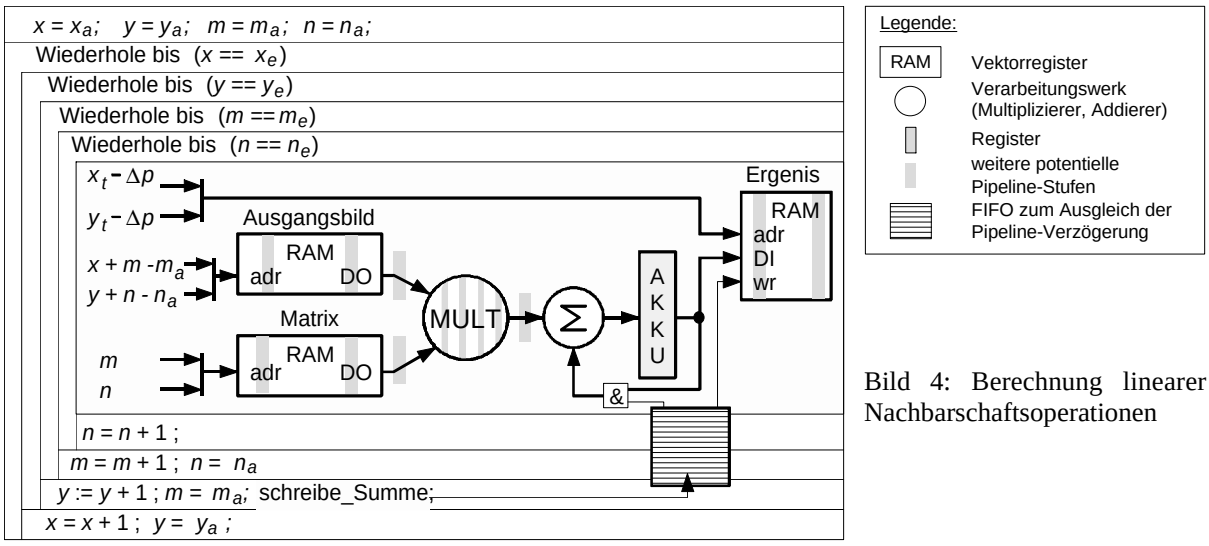


Bild 4: Berechnung linearer Nachbarschaftsoperationen

so daß etwa eine Rechenleistung bis zu 10 Milliarden Operationen je Sekunde erzielbar ist.

Bei dieser Rechnung sind Wartezeiten des Rechenwerkes, insbesondere auf die Bereitstellung der Grauwerte und das Rückschreiben der Ergebniswerte in den Hauptspeicher ausgeklammert. Vernünftige Aussagen hierzu verlangen, den gesamten Rechner einschließlich der Kopplung des Koprozessors mit dem Hauptprozessor und der Übertragung der Bildsignale in den Hauptspeicher zu betrachten. Mit einer geeigneten Architektur sind Konzepte ohne wesentliche Wartezeiten denkbar (schneller sRAMs für Bilddaten, zeitgleiche Verarbeitung und Datenübertragung u.ä.). Tiefere Untersuchungen würde aber den Rahmen dieser Studie sprengen.

Die Schleifensteuerung

Wieviel der ineinander verschachtelten Schleifen soll die Hardware-Steuerung übernehmen? Die Antwort hängt von der minimalen Größe der Filtermatrix ab. Nach Gleichung (1) muß der Vektorbefehl, um eine Pipeline der Tiefe $p \approx 10$ zu $\eta = 90\%$ auszulasten, mindestens $N = 100$ Basisoperationen umfassen. Häufig werden 3×3 oder 5×5 Filter genutzt. Bei diesen Filtergrößen werden in den beiden inneren Schleifen insgesamt nur $N = 9$ bzw. 25 Basisoperationen ausgeführt. Das ist für eine effektive Pipeline-Auslastung zu wenig.

Ein Vektorbefehl für die schnelle Filterberechnung sollte sich deshalb nicht nur auf ein Bildpunkt, sondern auf ein kleines Bildfenster beziehen. Das verlangt eine Hardwaresteuerung für 3 oder alle 4 ineinander verschachtelte Schleifen. Dabei unterliegen nur die inneren Schleifenzähler engen Zeitanforderungen. Die äußeren beiden Schleifen "Für alle Bildpunkte eines Fensters" können, wenn dadurch der Pipeline-Fluß nicht unterbrochen wird, auch vom Programm oder einem Mikroprogramm nachgebildet werden.

Die Schleifensteuerung zur Berechnung einer lokalen Nachbarschaftsoperation für ein Bildfenster zerfällt in die nebenläufigen Register-Transfer-Funktionen:

- Aktualisierung der Zählervariablen *x, y, m, n*:

$$\text{if } \left(m \neq m_e\right)_t \quad \left\{m_{t+1} = m_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n \neq n_e\right)\right)_t \quad \left\{m_{t+1} = m_a;\ n_{t+1} = n_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right) \wedge \left(y \neq y_e\right)\right)_t \quad \left\{m_{t+1} = m_a;\ n_{t+1} = n_a;\ y_{t+1} = y_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right) \wedge \left(y == y_e\right) \wedge \left(x \neq x_e\right)\right)_t \quad \left\{m_{t+1} = m_a;\ n_{t+1} = n_a;\ y_{t+1} = y_a;\ x_{t+1} = x_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right) \wedge \left(y == y_e\right) \wedge \left(x == x_e\right)\right)_t \quad \left\{\text{Befehlsende}\right\}$$

- Berechnung der Bildpunktadressen im Ausgangsbild $xm = x + m - m_a$ und $yn = y + n - n_a$:

$$\text{if } \left(m \neq m_e\right)_{t-1} \quad \left\{xm_{t+1} = xm_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n \neq n_e\right)\right)_{t-1} \quad \left\{xm_{t+1} = x_a;\ yn_{t+1} = yn_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right)\right)_{t-1} \quad \left\{xm_{t+1} = x_a;\ yn_{t+1} = y_a\right\}$$

- Berechnung der um Δp Schritte verzögerten Ergebnisadresse:

$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right) \wedge \left(x \neq x_e\right)\right)_{t-\Delta p} \quad \left\{xv_{t+1} = xv_t + 1\right\}$$
$$\text{if } \left(\left(m == m_e\right) \wedge \left(n == n_e\right) \wedge \left(x == x_e\right) \wedge \left(y \neq y_e\right)\right)_{t-\Delta p} \quad \left\{xv_{t+1} = x_a;\ yv_{t+1} = yv_t + 1\right\}$$

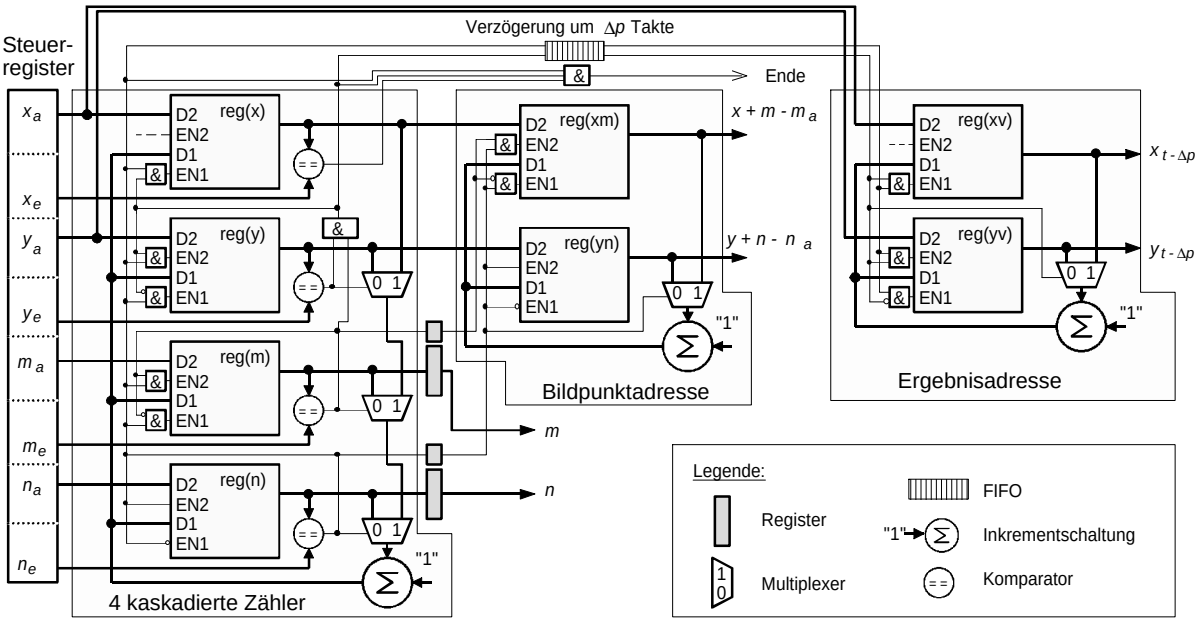


Bild 5: Modularer Entwurf für die Ablaufsteuerung der Filterberechnung

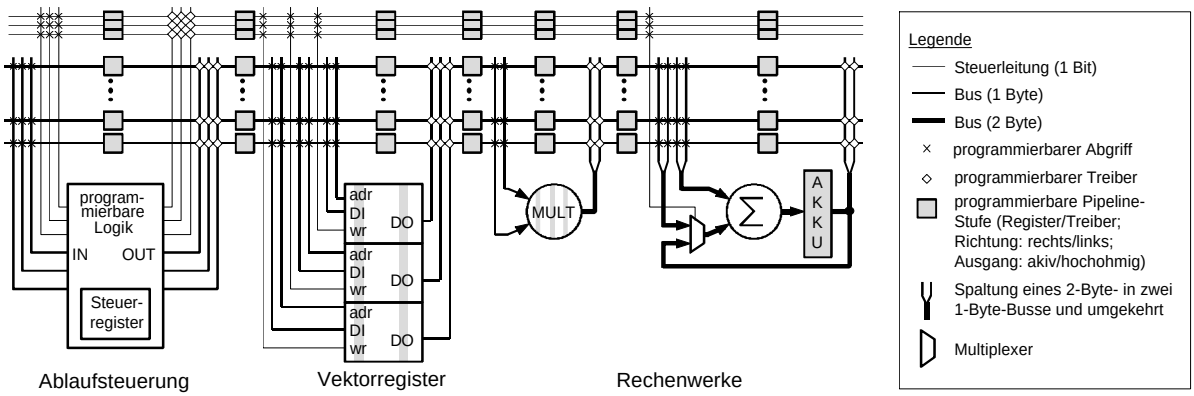


Bild 6: Rekonfigurierbares Vektorrechenwerk (Entwurf 1)

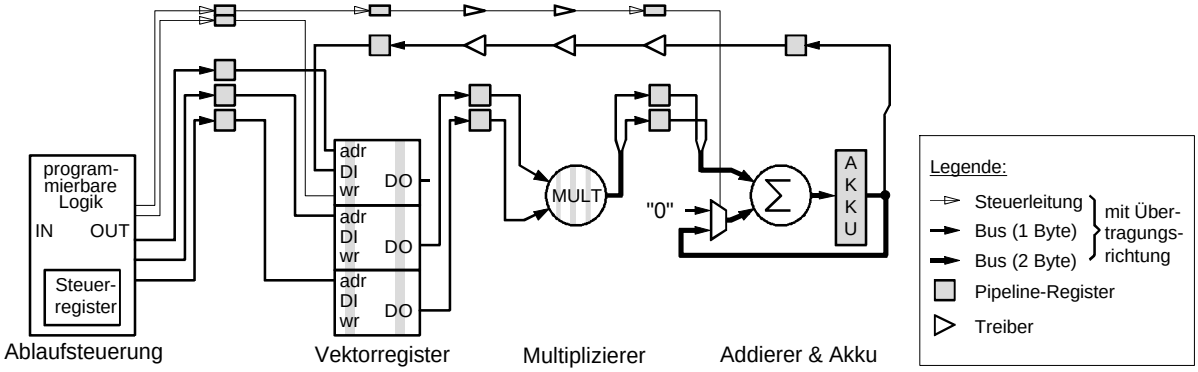


Bild 7: Konfiguration des Vektorrechenwerks für lineare Nachbarschaftsoperationen

In jeder der Register-Transfer-Funktionen wird gleichzeitig nur eine Variable gezählt. Es genügen zusammen drei Inkrement-Werke (Bild 5). Insgesamt besitzt das Steuerwerk eine reguläre, bausteinartige und erweiterbare Struktur. Für die Umsetzung soll eine speicherprogrammierbare Schaltung unterstellt werden.

3 Ein rekonfigurierbares Rechenwerk

Ein rekonfigurierbares Vektorrechenwerk besteht aus den Rechenelementen und einem geeigneten Verbindungsnetzwerk. Die Bausteine für den Vektorbefehl nach Bild 4 sind eine programmierbare Logikschaltung für den Adreßgenerator und die Ablaufsteuerung, drei byte-organisierte Vektorregister, ein 8×8-Bit-Multiplizierer und ein 16-Bit-Addierer mit Akkumulator (Bild 6).

Für das Verbindungsnetzwerk soll wie in [3] ein partitionierbares Mehrbussystem eingesetzt werden. Es bestehen aus einer Menge von Leitungsbündeln, die an ausgewählten Stellen unterbrochen und in mehrere Bündel von Stichleitungen aufgeteilt werden können. Unterschiede zu [3] (Bild 1) ergeben sich aus den sehr hohen Geschwindigkeitsanforderungen. Um das gesamte Rechenwerk mit einer Taktfrequenz von ca. 500 MHz betreiben zu können, ist eine kompakte Verbindungsstruktur erforderlich. Längere Verbindungswege sind durch Pipeline-Register zu unterteilen.

Die Einbeziehung der Ablaufsteuerung und des Adreßgenerators in das Rechenwerk verlangt ein System von Steuerleitungen. Für lineare Nachbarschaftsoperationen werden zwei Steuersignale benötigt. Das Addier-/Akkumulierwerk ist zwischen den Funktionen "Akku laden" und "Akkumulieren" umzuschalten. Im selben Rythmus aber zeitversetzt ist an den Ergebnisspeicher ein Schreibsignal zu senden.

Bild 7 zeigt die Konfiguration des Rechenwerks zur Ausführung des Vektorbefehls nach Bild 4. Die Adreßsignale m, n, $x+m-m_a$, $y+n-n_a$, $x_{t+\Delta p}$ und $y_{t+\Delta p}$ werden von der programmierbaren Logikschaltung bereitgestellt und über eine Pipeline-Stufe auf die Adreßeingänge der Vektorregister geführt. Die Datenausgänge der unteren beiden Vektorregister (Ausgangsbild und Koeffizientenmatrix) sind wiederum über eine Pipeline-Stufe mit den Eingängen des Multiplizierers verbunden. Dessen Ausgänge mit nunmehr doppelter Datenbreite führen auf die Akkumulationsschaltung, in der alle Produkte für einen Ergebnispunkt aufsummiert werden. Nach dem Start der Vektoroperation und anschließend aller $m\times n$ Takte wird die Rückführung des Akkumulationswerkes unterbrochen und damit ein neuer Akkumulationsprozeß

begonnen. Zwei Takte später liegt die zuvor akkumulierte Summe gemeinsam mit der Ergebnisadresse und dem Schreibsignal am Ergebnisspeicher an.

4 Die schnelle Fouriertransformation

Die Fouriertransformation dient zur Umrechnung eines Bildes in den Ortsfrequenzbereich und mittelbar für die schnelle Ausführung linearer Nachbarschaftsoperationen mit großen Masken. Lineare Nachbarschaftsoperationen mit einer Maske der Größe $\Delta m \times \Delta n$ nach Gleichung (2) bilden sich im Frequenzbereich auf $\Delta m \times \Delta n$ komplexe Multiplikationen ab. Hinzu kommt der Aufwand für die Transformation. Dieser ist für große Masken deutlich geringer als die Ausführung der Operation im Ortsbereich [9].

Die Fouriertransformation kann in eine Folge nacheinander auszuführender Transformationen für Teilbilder zerlegt werden. Gebräuchlich ist die Zerlegung der Fouriertransformation einer Fläche in eine Folge eindimensionaler Transformationen. Die eindimensionale Berechnung eines Punktes im Frequenzbereich gehorcht der Gleichung:

$$\hat{f}_u = \frac{1}{M} \sum_{m=0}^{M-1} f_m \cdot \mathrm{e}^{\frac{2 \cdot \pi \cdot \mathrm{i} \cdot m \cdot u}{M}} \qquad (3)$$

(i - imaginäre Einheit; *M* - Anzahl der Bildpunkte; *u* - Punktnummer im Fequenzbereich; *m* - Punktnummer im Ortsbereich). Für die Fouriertransformation eines Vektors, dessen Länge eine Potenz von 2 darstellt, gibt es einen sehr effektiven Algorithmus (Bild 8). Vor der Berechnung werden die Ausgangswerte in bit-gespiegelter Reihenfolge in ein Vektorregister geladen. Die Basisoperationen des Rechenschemas, die sogenannten Butterfly-Operationen, beginnen mit dem Lesen eines Wertepaares aus dem Speicher. Der zweite Wert wird mit einem komplexen Faktor multipliziert. Die Summe aus dem ersten Wert und dem Produkt überschreibt den ersten Wert im Vektorregister und die Differenz den zweiten Wert.

Die Adreß- und Koeffizientenreihenfolge gehorcht einem einfachen binären Algorithmus. Die Operanden einer Butterfly-Operation unterscheiden sich genau in einem Bit: in der 0-ten Berechnungsebene dem Bit 0, in der ersten Ebene dem Bit 1 und allg. in der *i*-ten Ebene dem Bit *i*. Für den ersten Operanden ist dieses Bit Null und für den zweiten Eins. Innerhalb einer Ebene wird die Adresse binär hochgezählt, wobei das *i*-te als nicht vorhanden oder logisch Eins interpretiert wird.

Die komplexen Koeffizienten der Transformation werden vor der Transformation berechnet und in einer Tabelle abgelegt. Die Tabelle besitzt den folgenden Aufbau:

$$\mathrm{LUT}\left((b_n \quad \cdots \quad b_1 \quad b_0)_2\right) = \exp\left(2\pi \cdot \mathrm{i} \cdot (0, b_0 \quad b_1 \quad \cdots \quad b_n)_2\right) \qquad (4)$$

Aus dieser Tabelle (LUT, look-up table) ergibt sich der Koeffizient für eine Butterfly-Operation durch die boolsche Operation:

$$\text{Koeffizent}(adr, \quad level) = \mathrm{LUT}\left(adr \wedge \left(2^{level} - 1\right)\right) \qquad (5)$$

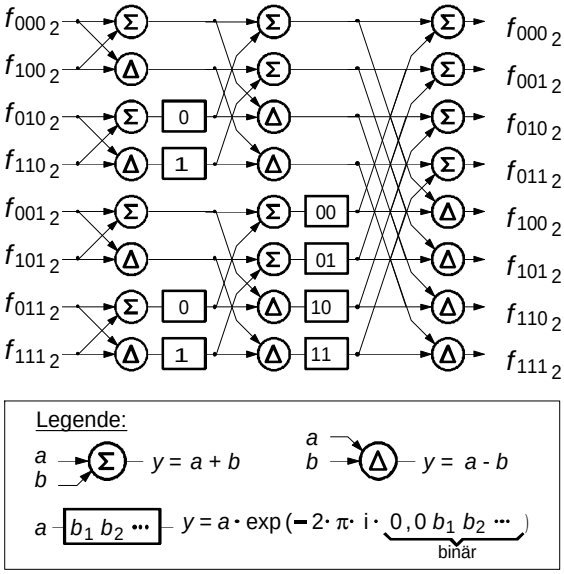
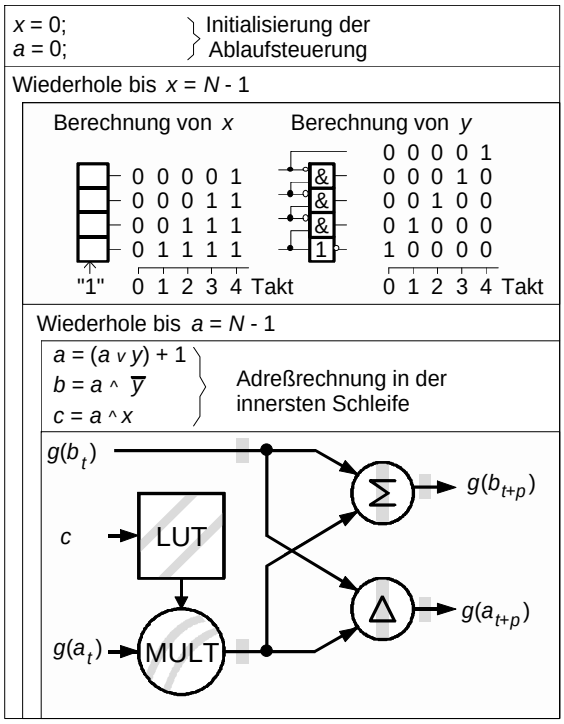


Bild 8: Butterfly-Algorithmus für die diskrete Fouriertransformation eines Vektors



N - Länge des zu transformierenden Vektors

Bild 9: Datenfluß einer Butterfly-Operation

Die Rechnung mit komplexen Zahlen und die Rundungsfehler der schnellen Fouriertransformation verlangen mindestens 4 Byte für die Darstellung eines Bildpunktes (2 Byte für den Real- und 2 Byte für den Imaginärteil). Bild und Koeffizientenspeicher benötigen eine Zugriffsbreite von 4 Byte.

Der Bildspeicher muß ein 4-Port-Speicher sein, in dem zeitgleich auf zwei Adressen gelesen und zwei anderen Adresse geschrieben werden kann. Die komplexe Multiplikation verlangt nach Gleichung:

$$(a+\mathrm{i}\cdot b)\cdot(c+\mathrm{i}\cdot d)=a\cdot c-b\cdot d+\mathrm{i}\cdot(a\cdot d+b\cdot c) \quad (6)$$

vier 16×16-Bit Multiplizierer, ein Subtraktions- und ein Additionswerk. Für die eigendliche Addition und Subtaktion werden weitere 4 Rechenwerke benötigt, so daß die Gesamtbilanz vier 16×16-Bit Multiplizierer und je drei 16-Bit-Addierer und Subtrahierer lautet.

Teilung der Butterfly-Operation in Schritte

Mit der Zerlegung der Butterfly-Operation in Phasen kann die Anzahl der Speicherzugriffe und der Register-Transferfunktionen im Pipeline-Fluß halbiert werden (Bild 10). Im Bildspeicher werden die beiden Operanden nacheinander gelesen und die Ergebnisse nacheinander geschrieben. Es genügt ein Dualport- statt eines 4-Port-Speichers. Die eigentliche Butterfly-Operation zerfällt in zwei identische Berechnungen, einmal für den Real- und zum anderen für den Imaginärteil. Dafür werden nur zwei Multiplizierer, ein Addierer und ein Subtrahierer benötigt. Für den Koeffizientenspeicher genügt, da der Real- und der Imaginärteil nacheinander gelesen werden können, eine Datenwortbreite von 2 Byte.

Der Bildspeicherzugriff und die Verarbeitung besitzen eine unterschiedliche zeitliche Ordnung. Ein Datenstrom der Form "Operand 2, Operand 1, ..." ist innerhalb des Datenpfades in eine Folge der Form "Realteil beider Operanden, Imaginärteil beider Operanden, ..." und umgekehrt umzusetzen. Das erfordert Zwischenspeicher und Multiplexer im Pipeline-Fluß.

5 Dynamisch schaltbarer Pipeline-Fluß

Der Mehrphasenalgorithmus für die schnelle Fouriertransformation trägt einen neuen Aspekt in die Architektur flexibler Pipeline-Rechenwerke. Die Zerlegung Verarbeitungsphasen, die in unterschiedlichen Zeitscheiben der Pipeline ausgeführt werden, verringert die Anzahl der Rechenwerke und die Anzahl der zeitgleichen Zugriffe auf ein Vektorregister. Auf diese Weise lassen sich die Ressourcen des Rechenwerkes besser ausnutzen, bzw. es lassen sich komplexere Basisoperationen abarbeiten.

Die Zerlegung verlangt, daß Daten in Pipeline-

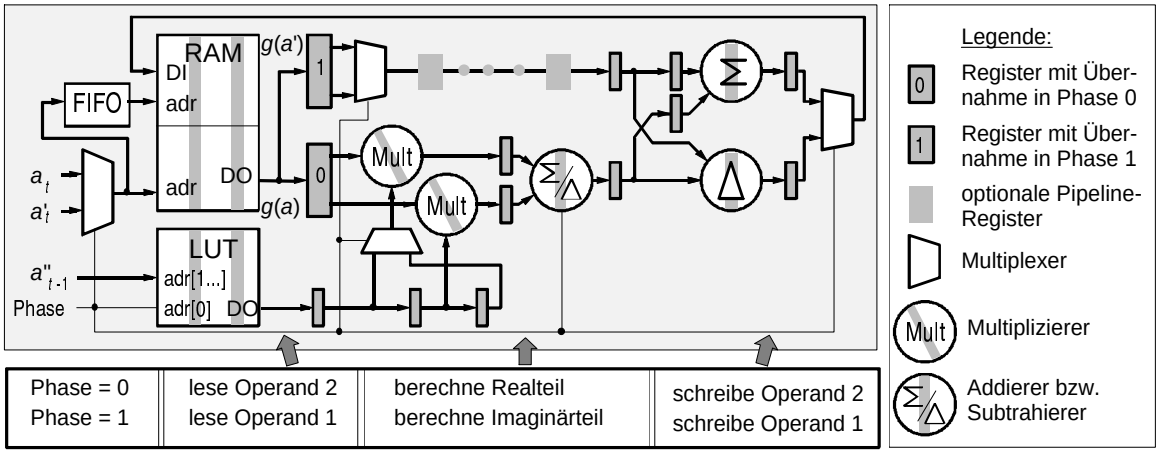


Bild 10: Vereinfachung der Verarbeitungspipeline einer Butterfly-Operation durch Aufteilung in zwei Verarbeitungsphasen

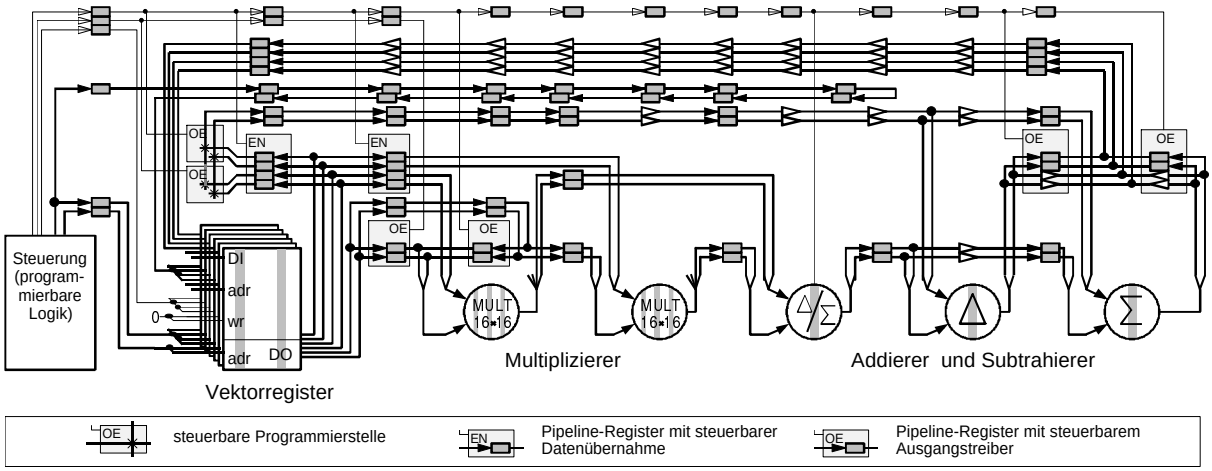


Bild 11: Datenfluß für der Zweiphasen-Butterfly-Operation nach Bild 10

Stufen länger als einen Takt aufbewahrt und Datenpfade umgeschaltet werden können. Die Multiplexerfunktionen sind in dem in Abschnitt 3 skizzierten Verbindungsnetzwerk vorbereitet (Bild 6). Jede Busprogrammierstelle besitzt eine Konfigurationsspeicherzelle, über die sie geschlossen oder geöffnet werden kann. Die Byte-organisierten Pipeline-Register zum Aufspalten von Verbindungen mit langen Signallaufzeiten besitzen statische Programmiermöglichkeiten zur Richtungsumschaltung, zur Überbrückung und zum deaktivieren ihrer Ausgänge. Für einen dynamisch schaltbaren Pipeline-Fluß sind die Konfigurationszellen durch Leitungen zu ersetzen, die über andere Programmierstellen optional mit Steuerleitungen verbunden werden können.

Das Speichern von Zwischenergebnissen für mehrere Takte verlangt eine zusätzliche Freigabesteuerung je Pipeline-Register. Die zugehörigen Steuerleitungen sind genau wie die anderen Steuerleitungen über Programmierstellen mit Steuerleitungen zu verbinden.

In Bild 11 zeigt, daß mit der Grundstruktur des Rechenwerkes nach Bild 6 und den hier diskutierten Änderung der Algorithmus nach Bild 10 nachgebildet werden kann. Es ist aber auch zu erkennen, daß die Verdrahtungsstruktur bereits recht kompliziert ist. Dieser Aspekt wird in Abschnitt 8 weiter verfolgt.

6 Berechnung von Histogrammen

Histogramme als diskrete Grauwertverteilungen sind die Grundlage für Punktoperationen (z.B. die Korrektur oder Normierung der Helligkeit und des Kontrastes) [9]. Eine andere Anwendung ist die Berechnung der Entrophie und von Kodeumsetzungstabellen für die redundanzarme Darstellung der Bildinformation. Die Berechnung eines Histogramms besteht darin, daß für jeden Bildpunkt der Grauwert gelesen, mit diesem Wert der Histogrammspeicher adressiert, der adressierte "Histogrammwert" inkrementiert und in den Histogrammspeicher zurückgeschrieben wird [10].

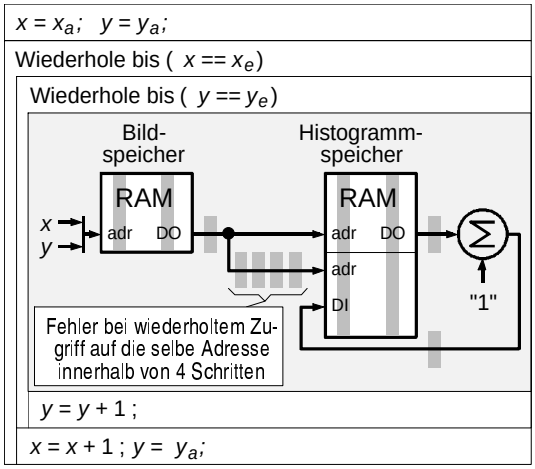
Das Lesen und Inkrementieren erfordert in Bild 12 vier Zeitstufen. Wird innerhalb dieser Zeit mehrmals auf die selbe Adresse im Histogrammspeicher zugegriffen, befindet sich der aktuelle Wert, d.h. der Operand der folgenden Inkrement-Operation noch in der Pipeline. Es tritt ein Datenkonflikt auf.

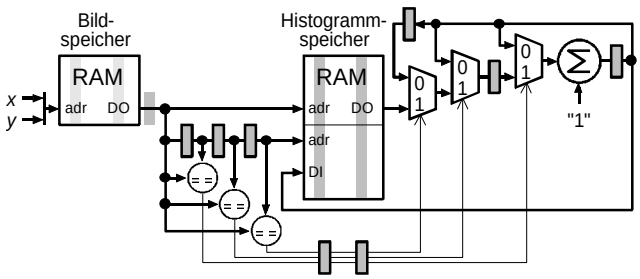
Datenkonflikte werden erkannt, indem die aktuelle Histogrammadresse mit allen 3 Vorgängern verglichen wird. Auf einen Konflikt kann das Rechenwerk auf unterschiedliche Art reagieren. Der Pipeline-Fluß vor dem Histogrammspeicher wird gestoppt. Das ist für ein Rechenwerk, das mit einer Taktfrequenz von mehreren 100 MHz arbeiten soll, schwer zu lösen. Eine andere und bessere Lösung ist, im Konfliktfall den Operanden der Inkrement-Operation statt aus dem Speicher direkt aus der Ergebnis-Pipeline zu übernehmen (Bild 13).

7 Datengesteuerter Pipeline-Fluß

Für die Berechnung linearer Nachbarschaftsoperationen und die Butterfly-Operation wurden alle Steuersignale von einem zentralen Steuerwerk bereitgestellt. Die Histogrammberechnung verlangt, daß auch aus Operanden Steuersignale abgeleitet werden. Das bedeutet für die Architektur des Rechenwerkes, daß auch die Steuersignale von unterschiedlichen Quellen stammen können. Die Steuerleitungen müssen genau wie die Datenleitungen als partitionierbares Mehrbussystem ausgeführt sein.

Für die Datenflußsteuerung der Histogrammberechnung werden Komperatoren benötigt. Jeder Komperator erzeugt ein eigenes Gleichheitssignal. Die Gleichheitssignale aller Komparatoren müssen in Steuersignale für Programmierstellen und programmierbare Pipeline-Register umkodiert werden. Diese Umkodierung verlangt eine programmierbare Logikschaltung.





links Bild 12: Berechnen eines Histogramms

rechts Bild 13: Lösung von Datenkonflikten

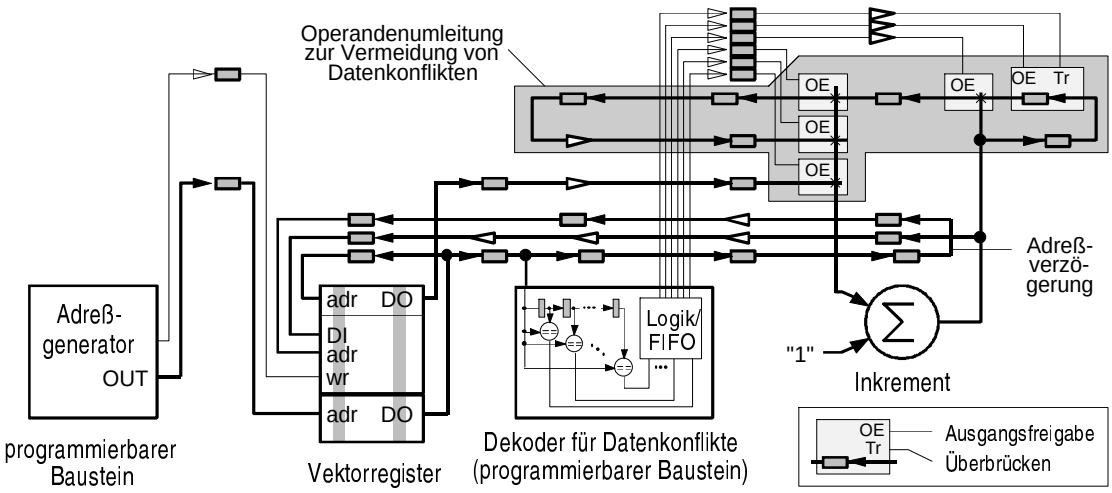


Bild 14: Konfiguration des Vektorrechenwerks zur Histogrammberechnung

In Bild 14 vergrößert das Verbindungsnetzwerk die Pipeline-Tiefe zwischen Auslesen und Rückschreiben des inkrementierten Wertes gegenüber Bild 13 von 4 auf 6 Takte. Die Adreßvergleichsschaltungen sind in einem programmierbaren Logikbaustein zusammengefaßt. Dieser Baustein erzeugt die Steuersignale für den Pipeline-Fluß und verbinden den Eingang des Inkrementier-Bausteins entsprechend erkannter Datenkonflikte entweder mit dem adressierten Speicherinhalt, dem vorhergehenden Ergebnis, dem Ergebnis der vorletzten Operation, ... oder dem vor 6 Takten berechneten Ergebnis.

Es wäre auch denkbar, für die Vergleichsschaltungen busorientierte Arithmetik- oder Logikeinheiten zu verwenden.

8 Dezentralisierung der programmierbaren Logik

Bild 14 legt einen Nachteil des bisherigen Buskonzeptes offen. Die Umschaltung des Pipelineflusses verlangt eine große Anzahl von Steuerleitungen.

Unterschiedliche Algorithmen benötigen ganz unterschiedliche Steuersignale. Für die Datenquelle kommen deshalb nur programmierbare Logikschaltungen in Betracht. Empfänger sind die Steuereingänge der Verarbeitungselemente (die Schreibsteuerung der Vektorregister, die Funktionsauswahl der Rechenwerke, der Programmiereingang für Busverbindungen und die Steuereingänge der Pipeline-Register). Die Kodierung der Steuersignale und die Anzahl der Steuerleitungen wird durch die Empfänger bestimmt. Jeder zu steuernde Treiber benötigt sein eigenes Steuersignal. Für einen k:1-Multiplexer sind das *k* Signale. Es gelingt kaum, mit einer Steuerleitung mehrere Treiber zu versorgen. Der Steuerbus muß sehr viele Punkt-zu-Punkt-Verbindungen erlauben.

In einem rekonfigurierbaren Rechenwerk ist die maximale Anzahl der Steuerleitungen ein Aufwandsfaktor. Jede zusätzlich Steuerleitung verlangt zusätzliche Programmierstellen und zusätzliche programmierbare Pipeline-Register. Ein Mangel an Steuerleitungen innerhalb des Vektorrechenwerks schränkt das Spektrum der Algorithmen, die abgearbeitet werden können, ein.

Die Anzahl der erforderlichen Steuerleitungen läßt sich wesentlich reduzieren, indem die Steuerinformationen redundanzarm übertragen werden. Für die Histogrammberechnung sind 7 Situationen zu unterscheiden (kein Datenkonflikt, gleicher Grauwert wie letzter, vorletzter, usw. Bildpunkt). Zur Unterscheidung genügen 3 Signale. Aus der dicht kodierten Steuerinformation müssen die Signale für die zu steuernden Datenelemente vor Ort zurückgewonnen werden. Das erfordert programmierbare Logikschaltungen in der Umgebung der zu steuernden Treiber und Register (Bild 16).

9 Zweidimensionale Anordnung und austauschbare Rechenelemente

Das bisher unterstellte lineare Bussystem besitzt gravierende Nachteile. Zur Nachbildung eines gegebenen Pipeline-Flusses wird ein Teil der Ressourcen des rekonfigurierbaren Rechenwerkes in einer bestimmten Reihenfolge verkettet. In den bisherigen Konfigurationsentwürfen (Bild 7, Bild 11 und Bild 14) wurden die Basiselemente (Logikbausteine, Vektorregister, Multiplizierer, Addierer) so angeordnet, daß jeweils eine einfache Verbindungsstruktur entsteht. Die Reihenfolge der Basiselemente im Rechenwerk ist jedoch starr und nicht anpaßbar. Viele Algorithmen werden eine wesentlich ungünstigere Verbindungsstruktur hervorrufen.

Bild 15 illustriert das am Beispiel eines Rechenwerks mit 9 Elementen. Bereits eine einfache Kettenstruktur, in der die Elementereihenfolge nicht mit der der Hardware übereinstimmt, führt zu vielen parallelen Bussen im Verdrahtungskanal und langen Verdrahtungswegen. Das Rechenwerk

benötigt eine große Verdrahtungskapazität. Lange Verdrahtungswege verlangen viele Pipeline-Stufen. Nach Gleichung (1) verringert sich die Pipeline-Auslastung der Vektorverarbeitung.

Das einfachste Prinzip, die Signalwege zu kürzen, ist eine zweidimensionale Anordnung der Rechenelemente (Bild 15). Es verkürzen sich dabei nicht nur die Signalwege. Die Verdrahtungsressourcen werden auch gleichmäßiger ausgelastet.

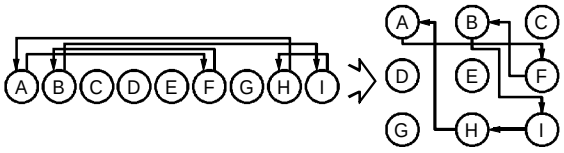


Bild 15: Verkürzung der Verdrahtungswege durch eine zweidimensionale Anordnung der Rechenelemente

Die Schaltungs- und Verdrahtungsressourcen einer Zellenmatrix lassen sich weiterhin besser ausnutzen, wenn Funktionsblöcke an unterschiedlichen Orten des Rechenwerkes angeordnet werden können. Für die bisher entworfenen Konfigurationen könnten z.B. folgende Modifikationen die Verdrahtbarkeit verbessern:

- dezentrale Anordnung der Vektorregister
- Nachbildung der Addition, der Subtraktion, des Vergleichs, der Akkumulation und gegebenenfalls von logischen Operationen durch eine umschaltbare Arithmetikeinheit.

Die einzelnen Blöcke sollten möglichst einheitliche Größe und eine vergleichbare Anzahl von Anschlüssen aufweisen. Komplexe Verarbeitungsfunktionen sind zu zerlegen:

- die 16×16-Bit-Multiplizierer für die Butterfly-Operation in 8×8-Bit-Multiplizierer und mehrere Arithmetikschaltungen
- die Ablaufsteuerungen der drei Algorithmen und die Schaltung zur Erkennung von Datenkonflikten in mehrere busorientierte Arithmetikschaltungen und kleinere freiprogrammierbare Bausteine für Steuersignale.

Bild 16 skizziert ein solches Rechenwerk. Die weitere Verfolgung und Verifizierung dieser Idee würde den Rahmen der Studie sprengen.

10 Zusammenfassung und Ausblick

In der Bildverarbeitung gibt es zahlreiche Aufgaben, die nur mit einem enormen Rechenaufwand gelöst werden können. Hierzu werden sehr schnelle, teilspezialisierte Rechner benötigt. Die Studie untersucht für eine Kombination von Techniken der Vektorverarbeitung mit Elementen der speicherprogrammierbaren Schaltungstechnik, welche Rechengeschwindigkeit und Flexibilität erreicht werden kann. Die erzielbare Rechengeschwindigkeit liegt um Größenordnungen über den Werten skalarer Prozessoren. Ein rekonfigurierbares Rechenwerk ist aus diesem Blickwinkel eine sehr kostengünstige Alternative zu einem Mehrprozessorsystem.

Über die Flexibilität erlaubt die Studie eine optimistische Prognose. Das Konzept ist nicht auf die drei untersuchten Algorithmen beschränkt. Mit den Ressourcen für lineare Nachbarschaftsoperationen

- drei Vektorregister

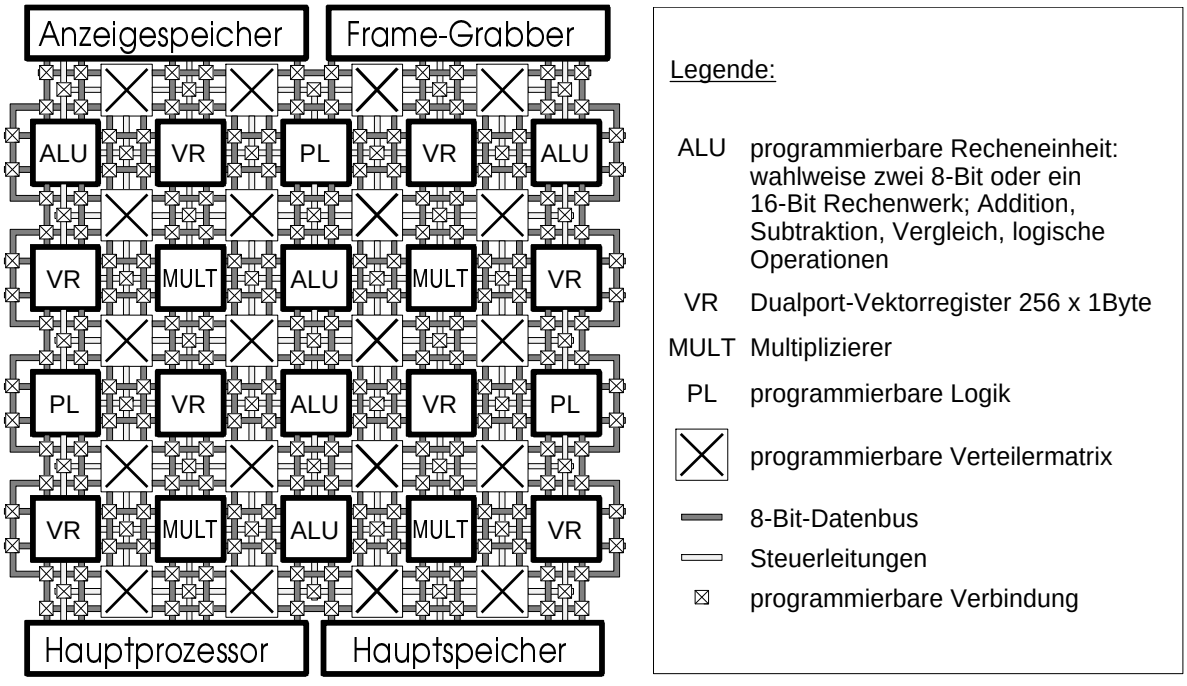


Bild 16: Eine reguläre, programmierbare Struktur für das Vektorrechenwerk

- ein 8×8-Bit-Multiplizierer
- ein 16-Bit-Addierer/Akkumulator

und den zusätzlichen Ressourcen, die nur für die Fourier-Transformation und für lineare Nachbarschaftsoperationen benötigt werden,

- zusätzliche Multiplizierer
- zusätzliche Vektorregister
- zusätzliche Arithmetikschaltungen

lassen sich zeitgleich mehrere Nachbarschaftsoperatioen ausführen (für unterschiedliche Bildausschnitte und/oder mit unterschiedlichen Koeffizientenmatrizen). Unbenutze Vektorregister können auch als programmierbare Funktionen (look-up table) oder Automaten eingesetzt werden. Das erlaubt die Nachbildung beliebiger nichtlinearer Punktoperationen, ortsabhängige Filterumschaltung und vieles mehr. Matrixmultiplikationen werden auch für die Koordinatentransformation von Graphiken z.B. von dreidimensionalen Objekten auf ein zweidimensionales Bildfenster benötigt [12].

Mit Hilfe ungenutzter Arithmektikschaltungen und der Möglichkeit einer datenabhängigen Datenflußsteuerung können Sortier- und Suchfunktionen nachgebildet werden. Sortierfunktionen werden z.B. für Rangordnungsfilter benötigt [9].

Eine wichtige Klasse der Bildverarbeitungsfunktionen sind die orthogonalen und unitären Transformationen. Hier nicht behandelte Transformationen wie die Hadarmadtransformation, die Haartransformation und die Kosinustransformation sind mit der Fouriertransformation verwandt [9], [10]. Auch diese Algorithmen können unterstützt werden.

Insgesammt ist zu erwarten, daß mit dem skizzierten Rechenwerk ein breites Spektrum rechenzeitintensiver Bildverarbeitungsalgorithmen sehr stark beschleunigt werden kann.

Literatur

[1] Grove, R. J.: Architectures for single-chip image computing. Image Processing and Interchange: Implementation and Systems. SPIE Vol. 1659, pp. 30-40

[2] Sunwoo, M. H.; Aggarwal, J. K.: VisTA - An Image Understanding Architecture. In Parallel Architectures and Algorithms for Image Understanding, Edited by Kumar, V. K. P., Academic Press, p. 131-153

[3] Gunzinger, A.: Synchroner Datenflußrechner zur Echtzeitbildverarbeitung. Verlag der Fachvereine an den schweizer Hochschulen und Techniken, 1990

[4] FPGAs als Coprozessor. Markt&Technik 28/95, p.1, 3

[5] Hennessay, J. L.; Patterson, D. A.: Computer Architecture. A Qualitative Approach, Morgan Kaufmann Publishers, 1990

[6] Wulf, W. A.; McKee, S. A.: Hitting the Memory Wall: Implication of the Obvious. Computer Architecture News, Vol. 23, No. 1, 1995

[7] Jouppi, N. P.; Wall, D. W.: Available Instruction-level Parallelism for Superscalar and Superpipelined Machines. 1989, p. 279-82

[8] Fink, T,; Liebig, H.: Mikroprozessortechnik. Springer-Verlag, 1994

[9] Jähne, B.: Digitale Bildverarbeitung. Springer-Verlag, 1993

[10] Ernst, H.: Einführung in die digitale Bildverarbeitung. Franzis-Verlag, 1991

[11] DECs Alpha 21064. c't 12/93, S. 138

[12] Langer, T.: Virtuelle Welten aus Draht. Toobox & Multimedia, 3/95, S. 90-94