4. Jahrgang · Heft 1 · Februar 1990 · 14,– DM · 124,– öS · 14,– sfr

Februar/März

VMEbus

Das Magazin für Boards, Systeme und Software

1/90

Der VMEbus in den 90er Jahren

Auch im nächsten Jahrzehnt ist mit weiteren Steigerungsraten im VMEbus-Markt zu rechnen. Zunehmend wird jedoch der Futurebus+ (VFEAbus) an Bedeutung gewinnen, der teilweise über eine „Brücke" den VMEbus als Ein-/Ausgabe-Bussystem verwenden wird.

TREND
Xputer: Parallelität auf unterster Ebene

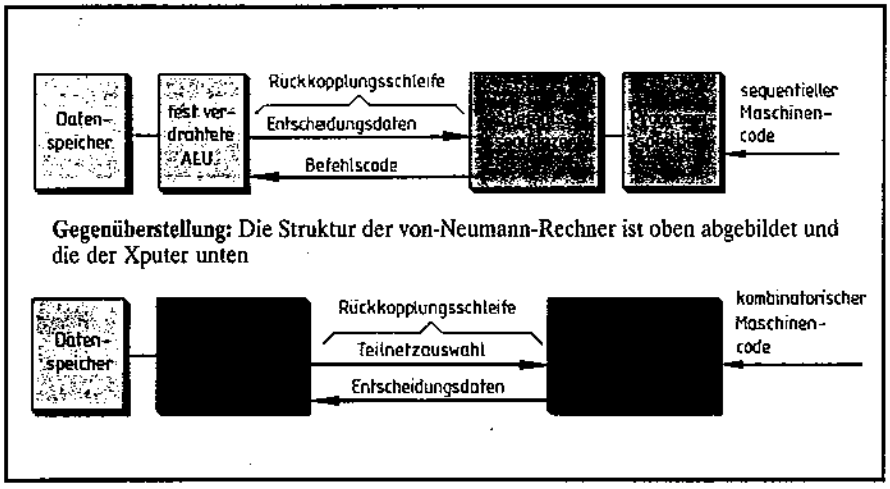
BILDVERARBEITUNG
VMEbus-System mit zwei Video-Kanälen für getrennte Kameras

SOFTWARE
X-Window, MGR und OS-9-Window für Echtzeit

„Rekonfigurierbare ALU erlaubt Parallelisierung auf unterster Ebene“

R. W. Hartenstein, A. Hirschbiel und M. Weber über Xputer

Prof. Dr.-Ing. R. W. Hartenstein ist Leiter der Forschungsgruppe für Rechnerstrukturen und Technische Informatik an der Universität Kaiserslautern. Dipl.-Inform. A. Hirschbiel und Dipl.-Inform. M. Weber sind zwei seiner Mitarbeiter.



Gegenüberstellung: Die Struktur der von-Neumann-Rechner ist oben abgebildet und die der Xputer unten

Viele VMEbus-Anwender und andere Computer-Anwender sind ständig auf der Suche nach immer leistungsfähigeren Systemen mit immer schnelleren Prozessoren. Insbesondere für eine Reihe von Echtzeit-Anwendungen (z. B. Bildverarbeitung, Signalverarbeitung und Computer-Grafik) werden Leistungen von vielen GIPS (Giga Instructions per Second) gewünscht und dies möglichst zu Preisen von wenigen Hundert Dollars zwecks Einsatz in Massenprodukten der Fahrzeugtechnik, Industrie-Elektronik oder Konsum-Elektronik. Bei traditionellen Computersystemen hemmen in zunehmendem Maße Limitierungen grundsätzlicher Art den Fortschritt (vgl. auch Colin Davies in VMEbus 1989, H. 5, S. 70 ff).

Das von-Neumann-Prinzip: seine Dominanz und seine Schwächen

Die heute im praktischen Einsatz befindlichen Computer basieren durchweg auf dem von-Neumann-Prinzip. Es gibt eine Reihe von alternativen Ansätzen, die teilweise davon abweichen, aber meist in ihrem Kern dennoch einen von-Neumann-Computer enthalten. Radikale Alternativen konnten kommerziell nicht Fuß fassen. Die Frage, warum gerade das von-Neumann-Prinzip eine derart monopolhafte Verbreitung gefunden hat, läßt sich wie folgt beantworten. Zu jener Zeit als dieses Konzept entstand (vor fast 50 Jahren), war es aus technologischen Gründen nicht möglich, komplizierte Hardwarestrukturen technisch zu realisieren. Deshalb hatte gerade ein so elegantes Konzept mit der technischen Einfachheit, wie es durch von Neumann vorgestellt wurde, diesen durchschlagenden Erfolg. Auf erste Rechner mit fest verdrahteten Programmen folgten hiervon immer wieder neue verbesserte Versionen mit im gemeinsamen Speicher abgelegten Programmen und Daten. Durch den Fortschritt der Technologie, wie etwa durch schnellere Prozessoren und durch viel höhere Speicherkapazitäten, offenbarten sich im Laufe der Zeit mehrere Durchsatz-Engpässe (auch „von Neumann bottlenecks“ genannt), die gewisse Schwächen dieses Konzepts deutlich machten.

Um nun trotz dieser „bottlenecks“ einen höheren Durchsatz zu erreichen, wurde im Laufe der Jahrzehnte das dominierende Konzept von Neumann's immer mehr verfeinert. Einen Raum für grundsätzlich neue Architekturprinzipien gab (oder gibt) es jedoch kaum, von wenigen Forschungsprojekten abgesehen. Der Grund hierfür liegt hauptsächlich darin, daß es aus Marketing-Gründen für Neuentwicklungen wichtig ist, auf bereits existierende Bestandteile zurückgreifen zu können. Diese Bestandteile dabei sind nicht nur die Hardware-Komponenten, sondern auch Software wie beispielsweise das Betriebssystem, Compiler und auch Anwendersoftware. Eine neue, von dem von-Neumann-Konzept abweichende Rechnerarchitektur zu entwickeln, bedeutet denn ein sehr weitreichendes und umfangreiches Ziel: sehr viele Teile eines umfangreichen Gesamtsystems müssen neu durchdacht und entwickelt werden. Und selbst wenn dies erfolgreich geschafft wurde, stünde der Markteinführung noch ein sehr schwieriges Problem im Wege: das Anwender-Personal (Programmierer usw.) hat eine Ausbildung hinter sich, die ausschließlich auf dem

von-Neumann-Konzept beruht – dies gilt auch für die Hardware-Entwickler. Die mit Abstand am meisten verbreiteten Sprachen sind von-Neumann-Sprachen und die Programmierer implementieren Programme, die auf dem virtuellen von-Neumann-Modell basieren. Neue Denkweisen können sich hier nur sehr schwer durchsetzen, und hängen auch davon ab, ob bekannte oder noch bessere komfortable Benutzeroberflächen zur Verfügung stehen, und ob auch hier eine Integration in existierende Anwendungsumgebungen ohne Umwege möglich ist.

Aus diesen Gründen konnten sich immer wieder nur auf dem von-Neumann-Prinzip basierende Neuentwicklungen durchsetzen. Um nun trotz der von-Neumann-Engpässe höhere Durchsätze zu erreichen, wurde um diese Architektur herum ein ganzes Arsenal von Tricks und Hilfs-Konzepten entwickelt. Beispiele dafür sind schnelle Pufferspeicher (Caches) für Programme und Daten, Prozessoren mit Pipeline-Verfahren, Einsatz von Coprozessoren für optimalen Speicherzugriff (DMA, MMU), für Grafik und für numerische Berechnungen (Gleitkomma-Arithmetik), und zwar stets als Coprozessoren (Slaves), so daß die von-Neumann-Eigenschaften des Masters unberührt bleiben.

Eine andere Möglichkeit die Leistungsfähigkeit zu steigern liegt darin, mehrere von-Neumann-Computer gleichzeitig an einem Problem parallel arbeiten zu lassen. Solche Parallelrechner-Systeme haben jedoch den Nachteil, daß erheblicher Aufwand für die Synchronisation der parallel laufenden Teilprozesse getrieben werden muß, so daß oft die erwartete Leistungssteigerung bei Weitem nicht erreicht wird. Teilprozesse können nur sehr schlecht und auf ineffiziente Weise Daten austauschen, weshalb die partitionierenden Compiler für solche Rechner nach Daten-*Un*abhängigkeiten suchen müssen. Nicht zuletzt deshalb sind viele Probleme nur schlecht parallelisierbar. Es wird immer einen größeren rein sequentiellen Anteil geben. Sind z. B. 50 % eines Problems parallelisierbar (was hier meist eine außergewöhnlich optimistische Annahme ist) während die anderen 50 % sequentiell ausgeführt werden müssen, dann ist klar, daß durch Einsatz auch noch so vieler Prozessoren die Durchsatzverbesserung höchstens den Faktor 2 erreicht (Amdahl's Gesetz), wobei die vom Betriebssystem für die Parallelisierung zusätzlich verbrauchte oft ganz erhebliche Rechenzeit (overhead) noch nicht einmal berücksichtigt ist. Eine Parallelisierung auf dieser hohen Ebene (Prozeßebene) wird auch als grobkörnig bezeichnet.

Compiler-freundliche Hardware

Ein wichtiger Aspekt in Hinsicht auf Durchsatzsteigerung ist die Code-Erzeugung durch die Compiler. Eine leistungsfähige Hardware steht und fällt mit den zur Verfügung stehenden Compilern. Bekannt ist dieses Problem auch von den RISC-Prozessoren her: wegen des primitiven Befehlssatzes ist deren effizienter Einsatz ohne gut optimierende Compiler kaum möglich. Es hat sich gezeigt, daß für Compiler um so effizientere Optimierungs-Verfahren angewandt werden können, wenn Daten-*Ab*hängigkeiten ausgenutzt werden können (anstatt Daten-*Un*abhängigkeiten), was von der Hardware allerdings außergewöhnlich schnelle Kommunikationskanäle verlangt, die bei klassischen Parallelrechnersystemen nicht vorliegen. ▷

Tabelle 1: Vergleich zwischen von-Neumann-Rechnern und Xputern

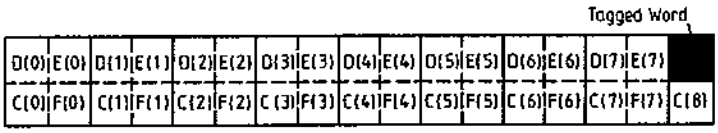
Teil der Maschine	Computer	Xputer
ALU	einfache Operationen (Befehlssatz) (Engpaß1: nur1 Operation zu einem Zeitpunkt)	mächtige Verbundfunktionen hoher Grad an Parallelität
	festverdrahtet	rekonfigurierbar
Befehls-sequenzer	Engpaß2: mehrere Lese-schritte pro Rechenoperationen	hat keine Befehlssequenzer
Programm-Speicher	Engpaß3: langsame Zugriffszeit	hat keinen Programm-Speicher
Daten-speicher	Engpaß4: sequentieller und langsamer Zugriff	Xputer: Anzahl der Zugriffe auf die Daten drastisch reduziert

Solche Optimierungsverfahren lassen sich um so besser anwenden, je feinkörniger die in der Hardware mögliche Parallelität ist. Dann können trotz sehr engmaschiger Datenabhängigkeiten selbst allerkleinste Teile eines Algorithmus parallelisiert werden bis hinunter zu einzelnen Befehlen. In dieser Hinsicht sind bei moderneren Rechnerarchitekturen, wie beispielsweise bei sogenannten VLIW-Architekturen deutliche Verbesserungen zu beobachten. (VLIW steht für „very long instruction word"). Doch auch hier handelt es sich noch um von-Neumann-*Com*puter mit all ihren Schwächen. Die wichtigsten Ergebnisse der letzten zwei Jahrzehnte Forschung und Entwicklung in der Computer-Architektur sind die folgenden Erkenntnisse:

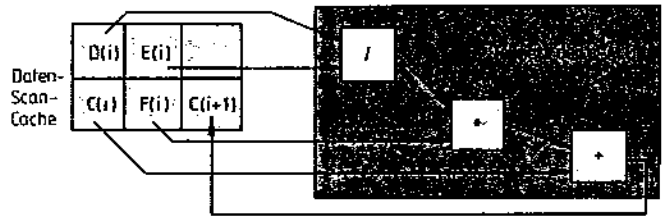
- O Für eine große wichtige Klasse von Algorithmen sind sehr gute Strategien zur Parallelisierung bekannt.
- O Deren Anwendung scheitert an mangelnder Flexibilität und ungenügender Bandbreite der Kommunikationsmechanismen in der Hardware (Compiler-feindliche Hardware).
- O Dabei wird im Betriebssystem und im Anwenderprogramm obendrein sehr viel zusätzliche Rechenzeit verbraucht (Overhead).
- O Deutlich bessere Ergebnisse sind nur dann zu erwarten, wenn die Hardware gemäß den Bedürfnissen der Compiler maßgeschneidert wird (Compiler-freundliche Hardware).
- O Hierzu muß die Hardware vor allem Parallelität auf möglichst niedriger Ebene gestatten – etwa weit unterhalb der Prozeß-Ebene („feinkörnige" Parallelität).

Die Parallelität von Xputern wird unterhalb der Befehls-Ebene realisiert und ist deshalb ganz besonders „feinkörnig". Xputer folgen dabei zur Erreichung von Compiler-Freundlichkeit einem völlig neuartigen Ansatz, dessen Prinzipien wesentlich von denen der von Neumann-Prinzipien abweichen. Nicht zuletzt wegen der besonders feinkörnigen Parallelität haben Xputer u. a. folgende Vorteile:

- O Xputer sind ebenso universell wie (von-Neumann-)Computer.
- O Xputer haben prinzipiell eine wesentlich höheren „Wirkungsgrad" als von Computer – ohne einen größeren Hardwareaufwand zu erfordern.
- O Bereits ein einziger Xputer-Prozessor bringt für eine wichtige Klasse von



Funktionsweise von Xputern: Oben ist durch den Xpiler vorbereitete Speicherbelegung des nebenstehenden Beispiels dargestellt; unten sieht man die Konfiguration des Daten-Scan-Caches und das Teilnetz der r-ALU (die Cachegröße ist 3 mal 2 und der „Walk" ist linearer Art und umfaßt acht Schritte nach rechts, wobei die Schrittweite 2 ist)



Beispiel: Der oben abgebildete Algorithmus wird durch den Datenabhängigkeitsgraphen (a) dargestellt; zum Vergleich die Partionierungsbeispiele für Vektormaschinen (b) und einen MIMD-Rechner mit vier Prozessoren (c)

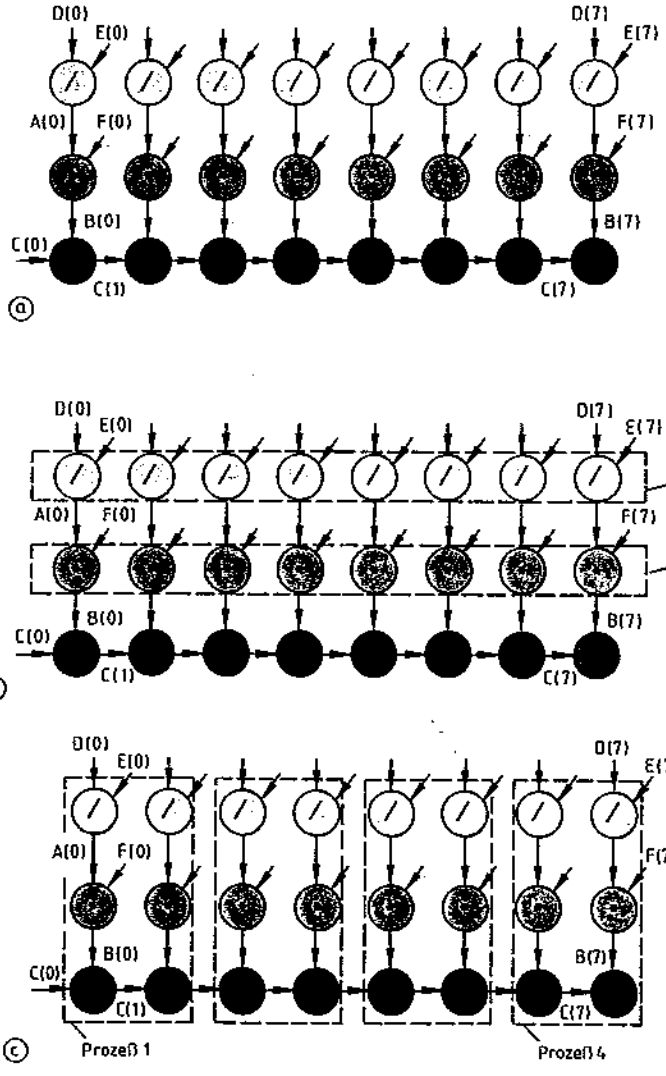


Tabelle 2: Leistungsvergleich verschiedener Rechner-Architekturen

Operations-Beispiel	MoM	VAX 11/750		68000	
	Sek.	Sek.	Akzel.-Faktor	Sek.	Akzel.-Faktor
Diverse einzelne Bildvorverarbeitungs-Operationen **	0,06	7,8	>130	14,7	>240
Bildverarb.: Erosion, Dilation, Kanten-Entdeckung **	0,21	38	>180	64	>300
Design Rule Check f. integr. Schaltungen (λ-basiert)	0,26	300	>1000	600	>2000
Lee-Routing-Algorithmus	0,15	ca. 20	>130	40	>160
10×10-Matrix-Multiplikation:					
* systolic array-Emulation	0,016	0,04	(>2)	0,15	(>9)
* mit 2 Scan-Caches	0,001	0,3	>40	0,15	150
Sortier-Algorithmus	0,100	4,2	>40	8,0	80

* konservative TTL-Demo-Hardware ** (512×512-Bildpunkte)

Algorithmen einen Durchsatz, der sich selbst mit denen von modernen parallelen Computersystemen zumindest vergleichen läßt, oder diese sogar übertrifft.

- Der (nicht programmierbare) Standardteil der Xputer-Hardware ist einfacher als der der RISC-Prozessoren.
- Die Technologie der rekonfigurierbaren Teile von Xputern ist verfügbar von einem neuen rasch wachsenden Markt (1990: 1 Mrd. US-Dollar mit weltweit ca. 20 PLD-Anbietern).

Computer sind abhängig vom Programmcode, der durch das Leitwerk sequentiell aus dem Speicher gelesen wird. Die Programmierbarkeit von Xputern hingegen basiert auf einem völlig neuen Paradigma: auf konfigurierbarer Hardware, basierend auf programmierbaren Verbindungen elektrisch rekonfigurierbarer Hardwarestrukturen. Diese werden unter der Verwendung von programmierbaren Logik-Schaltungen realisiert. Beispiele dafür sind: PLDs (Programmable Logic Devices), MAX (Multiple Array Matrix, Altera), DPLAs (Dynamically Programmable Logic Arrays), die an der Universität Kaiserslautern entwickelt wurden, sowie PGAs (Programmable Gate Arrays) wie z. B. LCAs (Logic Cell Arrays, Xilinx), CALs (Configurable Array Logic, Algotronix), AGA (Alterable Gate Arrays, UMA).

Die mit solchen Technologien erzielbare Rekonfigurierbarkeit der Hardware erlaubt eine sehr feinkörnige, aber auch sehr flexible Form der Parallelität auf niedrigster Ebene. Auf dieser Basis liegt das Geheimnis der außergewöhnlich hohen Leistungsfähigkeit von Xputern in einer extremen Compiler-Freundlichkeit der Hardware.

Jenseits systolisierbarer Algorithmen

Xputer sind in der Lage einfach alles zu berechnen, was berechenbar ist, weil sie genauso universell wie von-Neumann-Computer sind. Es gibt jedoch viele wichtige Problemstellungen, bei welchen Xputer vom Prinzip her in einem ganz außergewöhnlichen Maße leistungsfähiger sind als Computer. Solche Problemstellungen liegen vor, wenn regelmäßige Datenabhängigkeiten äußerst effizient auf einen zwei- oder mehrdimensional organisierten Speicher abgebildet werden können. Wir können hier prinzipiell zwei Klassen von Algorithmen angeben:

- systolisierbare Algorithmen (systolische oder in solche konvertierbare Algorithmen)
- Algorithmen mit generischen Datenabhängigkeiten (G-Algorithmen)

Systolisierbare Algorithmen (S-Algorithmen) sind eine Teilmenge der G-Algorithmen. Systolische Algorithmen (z. B. Konvolution, Korrelation, Algorithmen für Filter, Bildvorverarbeitung, lokale Pattern-Matching-Verfahren usw.) haben eine hohe Dichte von lokalen und regelmäßigen Datenabhängigkeiten, so daß eine Hardware lediglich die lokale Kommunikation wirksam unterstützen muß, wie dies zum Beispiel bei VLSI-Array-Prozessoren (VAPs) der Fall ist (sogenannten systolischen Arrays). VAPs haben aber den Nachteil nicht programmierbar zu sein, weshalb hier ein teurer und zeitraubender ASIC-Entwurf nötig wäre. Bei den meisten S-Algorithmen sind Xputer gegenüber allen bekannten Parallelrechnern konkurrenzfähig oder sogar überlegen – mit nur einem einzigen Prozessor.

Unsystolisierbare G-Algorithmen (NSG-Algorithmen) besitzen lediglich globale regelmäßige Datenabhängigkeiten. NSG-Algorithmen, wie FFT-Algorithmen oder die Mehrzahl der Sortieralgorithmen und andere, sind sehr viel schwerer parallelisierbar als systolische Algorithmen. Keines der bisher bekannten Computer-Hardwarekonzepte ▷

unterstützt Strategien zur effizienten Parallelisierung von NSG-Algorithmen – auch nicht VLIW-Architekturen, noch VAPs. Xputer bieten durch ihre Hardware die einzig bekannte Ressource, welche wirklich hochoptimierten „Code" auch aus NSG-Algorithmen akzeptiert.

Xputer-Prinzipien

In diesem Abschnitt werden die Xputer-Prinzipien in einer mehr vergleichenden Art und Weise kurz zusammengefaßt (Details sind anderweitig publiziert worden). Xputer-Prinzipien und ihr Unterschied zu den von-Neumann-Prinzipien werden beleuchtet. Des weiteren werden Unterschiede gegenüber den Kommunikations- sowie Synchronisationsmechanismen in parallelen Rechensystemen, Datenflußmaschinen und VLSI-Array-Prozessoren betrachtet.

Die ALU von Computern ist eine Einheit mit geringer Bandbreite: sie kann lediglich eine einzige Operation zu einem Zeitpunkt ausführen. Xputer benutzen eine rekonfigurierbare r-ALU, welche durch mehrere hoch parallelisierte Datenpfade, die einen mächtigen sogenannten „Compound Functor" bilden, ihre Gestalt erhält. Die Parallelität dieser „Compound Functors" in Xputern (innerhalb der r-ALU) ist auf einer niedrigeren Ebene (sehr feinkörnige Parallelität) anzusiedeln, als beispielsweise die „Compound"-Funktionen (Verbundfunktionen) in VLIW-Maschinen (Very Long Instruction Word). Diese Funktionen befinden sich auf Befehlsebene und erfordern teure interprozessorale Kommunikation.

Bei den Xputern wurden drei von vier von-Neumann-Engpässen eliminiert. Der vierte Engpaß wurde aufgrund der hochoptimierenden Compilation durch Xpiler (Compiler für Xputer) ganz erheblich reduziert. Durch die Xpiler-Übersetzung erreicht man grundlegend bessere Optimierungsergebnisse (speziell bei G-Algorithmen) als dies bei Compilern für konventionelle parallele Prozessorsysteme der Fall ist.

Parallelität auf niedrigster Ebene. Xputer haben keine fest verdrahteten Befehle. Vielmehr konfiguriert der Xpiler (neuartiger Compiler) innerhalb der r-ALU mehrere Teilnetze, die dem Format des „Data Scan Cache" (neuartiger Cache) angepaßt sind.

In der Rekonfigurierbarkeit der r-ALU liegt eines der Geheimnisse der sehr hohen Leistungsfähigkeit von Xputern. Es kann durch den Compiler Parallelität auf einer weit niedrigeren Ebene genutzt werden, als dies bei allen bisher bekannten Arten programmierbarer Hardware der Fall ist. Für Kommunikation und Synchronisation sind keine zeitraubenden Konzepte notwendig, wie z. B. die Anwendung von Bus-Systemen. Es entsteht praktisch kein „Overhead".

Data Sequencing. Weil der kombinatorische Maschinencode direkt in die r-ALU geladen wird, benötigen Xputer weder einen Programmspeicher noch einen Befehls-Sequenzer (Leitwerk bzw. „controller"). Stattdessen stellt der Daten-Sequenzer des Xputers ein fest verdrahtetes parametrisiertes Repertoire an generischen Datenzugriffsmustern (generic data scans, Adreßsequenzen wie beispielsweise lineare Sequenzen, Sprünge, Video Scans, Shuffles, Butterflies usw.) zur Verfügung, welche wir „Walks" (walk = laufen durch den Speicherraum) nennen. Eine einfache Hardware erzeugt die nötige Folge von Datenadressen für das Durchlaufen (walk through) eines Pfades im Adreßraum eines gegebenen Datenspeicher-Segmentes. Für ein bestimmtes r-ALU-Teilnetz und seiner damit verbundenen Scan-Cache-Konfiguration stellt der Xpiler einen passenden „Walk" zur Verfügung. Die Adreßfolge, die sich aus einem solchen „Walk" ergibt, wird auf optimale Weise durch eine Hardware ohne „Overhead" generiert.

Die Reihenfolge der ausgeführten Datenoperationen wird also nicht aus Steuerungsanweisungen („nächster Befehl", goto usw.) abgeleitet, ist also nicht „control-driven" wie beim herkömmlichen Computer. Vielmehr wird die Reihenfolge aus der Folge der anzulaufenden Datenspeicherplätze abgeleitet, ist also „data-driven". Dieses Prinzip der Ablaufsteuerung nennen wir „Data Sequencing". Dies ist ein weiteres Geheimnis der hohen Leistungsfähigkeit von Xputern.

Am Ende eines „Walks" wird ein markiertes Wort (tagged word) empfangen, welches den aktuellen „Walk" mit dem Nachfolgenden verbindet. Natürlich stellen Xputer aus Universalitätsgründen Verzweigungen zur Verfügung, welche Entscheidungsdaten auswerten, um den folgenden „Walk" zu ermitteln (für datenabhängige „Walks", wie z. B. Kurvenverfolgung) oder den nächsten Schritt innerhalb eines „Walks" zu beeinflussen. Das ist auch letztendlich der Grund warum Xputer genauso universell wie gewöhnliche Computer sind.

Xputer können in einem Stand-alone-Modus arbeiten. Jedoch hat der Coprozessormodus zusammen mit einem Computer als Hostrechner den Vorteil, sowohl der weichen Integration in bereits bestehende Anwendungsumgebungen, als auch der bequemen Nutzung von vorhandenen Programmierwerkzeugen.

Xputer im Vergleich zu Parallel-Rechnern

Innerhalb einer von-Neumann-Maschine können mehrere wichtige Durchsatzengpässe ausgemacht werden. Dies ist ein Grund warum von-Neumann-Prozessoren um exponentielle Größenordnungen schwächer fungieren als Lösungen, welche maßgeschneiderte ASICs mit einer vergleichbaren Transistoranzahl benutzen. Da bei Xputern die meisten dieser Engpässe umgangen werden, kommen sie der Leistung von ASIC-Hardware wesentlich näher, obwohl auch Xputer programmierbare Prozessoren sind, die auf einfachen Maschinenprinzipien – genauso einfach wie die Prinzipien eines von-Neumann-Computers – beruhen. Da es mittlerweile auf dem Markt an innovativen parallelen Computersystemen sowie Supercomputern nicht mangelt und diese sehr mächtige Hardware-Ressourcen zur Bewältigung massiver paralleler Problematik besitzen, drängt sich die Frage auf: Benötigen wir Xputer? Aus diesem Grund untersuchen und diskutieren wir im nächsten Abschnitt einige Vergleichsdaten und Gütemaße von Hochleistungscomputern. Weil Xputer datengesteuert arbeiten, betrachten wir in unserer Erörterung auch Datenflußmaschinen.

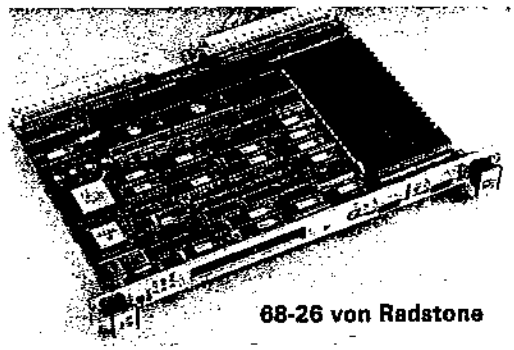
Will man Xputer in die Diskussion mit einbeziehen, so ist es äußerst schwierig objektiv zu bleiben, weil eine sehr große Bandbreite der Ebenen der Parallelität betrachtet werden muß: die Prozeßebene (hohe Ebene), die Befehlsebene (mittlere Ebene) und die darunterliegende Ebene (niedrige Ebene). Begriffe wie *Granularität des Parallelismus* und *Grad der Parallelität* müssen neu definiert werden, weil sie gewöhnlicherweise prozessorbasiert verwendet werden. Wegen dieser Nichtangemessenheit der Abgrenzungskriterien, die üblicherweise in der Literatur verwendet werden, ist es äußerst schwierig die Parallelität auf niedrigster Ebene der Xputern mit der Parallelität auf der hohen und mittleren Ebene zu vergleichen.

Der Schlüssel für optimalen Parallelismus liegt im „Partitioning" und

Mit Huckepack-Moduln erweiterbar

Die Doppeleuropakarte 68-26 mit einer 25-MHz-Version des 68020 kann mit Aufsteckplatinen an die jeweilige Anwendung angepaßt werden.

Die optionell mit den Coprozessoren 68881 oder 68882 ausrüstbare Zentraleinheit bietet Platz für bis zu 16 MByte DRAM mit Paritätsprüfung. Sie verfügt über Jedec-Sockel für EPROMs und SRMAs. Außerdem sind auf dem Board zwei serielle Schnittstellen, ein Parallel-Interface, ein 24-Bit-Timer sowie eine Echtzeituhr untergebracht. Über das APEX-Interface kann man Huckpack-Module anschließen, die wahlweise eine Ethernet-, SCSI-, Floppy-Disk- oder IEC-Bus-Controller enthalten.



68-26 von Radstone

„Scheduling" (Steuerung des Zeitablaufs). Gewöhnliche prozedurale Programmnotationen verlangen ein voreiliges „Partitioning" und „Scheduling" (der Befehle) und verhindern somit optimale Parallelität. Dies ist schlechthin das Hauptproblem der optimierenden Compilertechniken. Aus Korrektheitsgründen muß man lediglich die Datenabhängigkeiten aufrechterhalten. Die ersten beiden Operationen in der Schleife (‚/' und ‚*') sind unabhängig, wobei die dritte Anwendung (‚+') eine Rekurrenz beinhaltet und daher sequentiell ausgeführt werden muß (vgl. Beispiel).

Die Optimallösung hängt von beidem ab, den Möglichkeiten des Compilers und den Eigenschaften der Hardware. Falls der Compiler eine exzellente „Partitioning" – und „Scheduling"-Strategie besitzt, stellt sich immer noch die Frage: Kann die Lösung auf die Zielhardware abgebildet werden! Falls der Compiler nach Datenabhängigkeiten sucht, wird die Verständigungsbandbreite innerhalb der Hardware zu einem kritischen Punkt. Falls der Compiler nach Unabhängigkeiten unter den Daten forscht wird die Anzahl der verfügbaren Prozessoren zum Kritikum.

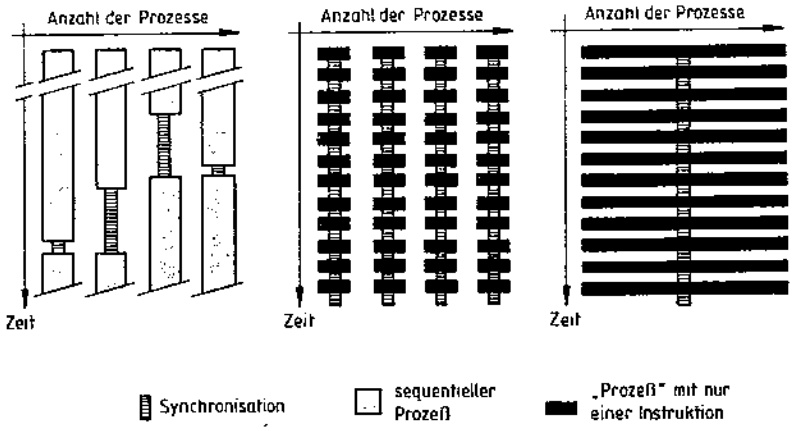
Von der Hardware wird wegen der leistungsschwachen Kommunikations-Hardware eine ausgeglichene Balance zwischen Verarbeitungsleistung der Prozessoren und der Kommunikationsbandbreite verlangt. Viel Prozessorkapazität kann nicht genutzt werden, wenn die benötigte Kommunikations-Kapazität nicht vorhanden ist. Diese Balance hängt auch von dem zu implementierenden Algorithmus ab. Eine wichtige Klasse von Algorithmen mit massiven Echtzeit-Anforderungen besitzen in hohem Maße Datenabhängigkeiten, so daß die einzige Möglichkeit darin besteht, diese Datenabhängigkeiten zu suchen. Die Hardware muß hier an erster Stelle eine hohe Kommunikationsbandbreite bieten.

Vektormaschinen haben fundamentale Durchsatzengpässe. Deren tatsächlich erreichte durchschnittliche Leistung ist Größenordnungen kleiner, als die in den Datenblättern angegebenen Spitzenleistungen. Dies ist auch der Fall, wenn kreatives und intelligentes Programmieren dem Compiler hilft, Code zu generieren, welcher gewisse Organisationsengpässe der Maschine umgeht. Der Grund dafür ist, daß Vektormaschinen eine sehr spezielle Art von Parallelismus (auf niedriger Ebene) mit einem sehr speziellen Kommunikationsmechanismus haben. Division und Multiplikation können wegen der Datenunabhängigkeit vektorisiert werden. Die Addition jedoch ist rekurrent und muß deshalb in einer Pipeline bearbeitet werden. Der Parallelismus auf dieser niedrigen Ebene ist schnell aber extrem inflexibel, so daß Compiler nicht viel mehr tun können als im Algorithmus entsprechende Schleifen auszutauschen.

MIMD-Parallelprozessorsysteme (Multiple Instruction Multiple Data) verwirklichen Parallelismus auf höchster Ebene (Prozeßebene). Die Kommunikation ist sehr langsam, da die Synchronisation der einzelnen Transaktionen und die Zugriffsblockierungsmechanismen (Monitor, Semaphor usw.) durch das (sequentielle) Betriebssystem zur Laufzeit erfolgen. Compiler partitionieren hier, indem sie nach Datenunabhängigkeiten suchen, wodurch die Prozesse groß werden. Die unvorhersagbare Laufzeit jeden Prozesses bewirkt, daß der langsamste die schnelleren Prozesse warten läßt. Der Optimierungserfolg der Compiler ist speziell für S- und G-Algorithmen extrem niedrig.

VLIW-Architekturen (Very Long Instruction Word) verwirklichen Parallelismus auf mittlerer Ebene (Befehlsebene). Prozessoren können bei jeder Befehlsausführung synchron über prozessorexterne Busse kommunizieren. Wegen der hohen Kommunikationsbandbreite wird eine datenabhängige Partitionierung unterstützt. Gute Ergebnisse werden durch Systolisierung von S-Algorithmen erzielt, da passende Befehlsfolgen für die Hardware gefunden werden können. Für nicht systolisierbare G-Algorithmen wird eine Strategie benutzt, die „Trace Scheduling" genannt wird. Diese ist probabilistisch und folglich von beschränkter Effizienz.

VLSI-Array-Prozessoren (VAPs) verwenden Intra-Chip-Kommunikation über dedizierte Verbindungen die schneller als Busse und extrem platzsparend sind, wodurch mehr Fläche für mehr Prozessorelemente frei wird, um den Parallelitätsgrad zu erhöhen. Deshalb erlauben VAPs den höchstmöglichen Parallelitätsgrad auf der niedrig- ▷



Klassen von Parallelität: Nebenläufige Prozesse bei Parallelrechnern (links), parallele Prozesse durch Verbundfunktionen in VLIW-Architekturen (Mitte) und Intra-ALU-Parallelität durch „Compound Functors" in Xputern (rechts)

sten Ebene. Diese ermöglicht extrem gute Leistungsergebnisse bei systolisierbaren Algorithmen, wobei die Optimierung vollständig auf Datenabhängigkeiten basiert. Dies zeigt, daß die besten Optimierungsergebnisse erzielt werden, wenn alle Prozessorelemente auf den gleichen Chip plaziert werden und die niedrigst mögliche Ebene der Parallelität gewählt wird. Genau dies ist das Konzept des Xputers: dessen r-ALU – die intern die niedrigste Ebene von Parallelität verwendet – kommuniziert On-chip direkt mit dem „Daten-Scan-Cache". VAPs sind jedoch nicht frei programmierbar. Ein anderer wesentlicher Nachteil der VAPs ist die Tatsache (häufig in der Literatur nicht erwähnt), daß periphere Konversionen zwischen Datenfeldern und Datenströmen („scrambling" und „unscrambling") einen erheblichen zusätzlichen Aufwand verursachen, der obendrein noch rapide weiter eskaliert, falls ein Problem nicht vollständig auf einen einzigen Chip paßt.

Datenflußmaschinen sind extrem compiler-feindlich, da sie hochgradig nichtdeterministisch arbeiten. Optimierende Compiler-Strategien sind hier nicht anwendbar, da diese eine detaillierte Vorausplanung der Ausführungs-Reihenfolge verlangen, also eine deterministische Maschine. Zudem ist zur Laufzeit keine Optimierungsstrategie vorhanden. Datenflußmaschinen neigen dazu unter Datenzugriffskonflikten zu leiden und haben einige neuartige Flaschenhälse. Herkömmliche höhere Programmiersprachen können nicht automatisch für den Einsatz bei Datenflußmaschinen übersetzt werden. Der Code wird häufig sehr lang, enthält viel Redundanz und verursacht einen enormen Adressierungsaufwand.

Xputer vermeiden den Kommunikations- und Synchronisationsaufwand von Vektorprozessoren, MIMDs und VLIW-Maschinen durch ihre festverdrahteten generischen Datenzugriffsfolgen. Das abgebildete Beispiel zeigt: die Sequenz nimmt acht Schritte nach rechts mit einer Schrittweite von zwei. Die r-Alu vereinigt alle Operatoren einer Schleifeniteration in einem einzigen „Compound Functor", so daß exakt ein einziger Speicherlesezyklus und ein einziger Schreibzyklus pro Datum gebraucht wird, um das Datenfenster zu verändern; kein Befehlszyklus wird benötigt. Die Kommunikation findet innerhalb dieses „Fensters" statt ohne zusätzliche Zeit zu benötigen (das Fenster benutzt einen internen Schiebemechanismus). Synchronisation ist nicht notwendig, da der ganze Vorgang in einem einzigen Prozessor stattfindet. Bei VLIW-Maschinen, Vektorprozessoren und speziell bei MIMDs, jedoch, führt die Rekurrenz zu einem großen Mehraufwand wegen der durchgereichten Werte der Variablen C(i). Ein Vergleich mit der Verarbeitung durch systolische Arrays ist eine gute Illustration der Verarbeitungsprinzipien des Xputers. Statt eines Feldes von Prozessorelementen haben Xputer ein einziges Prozessorelement (die r-ALU). Es benutzt den „Daten-Scan-Cache" als ein Fenster, das sich über das Datenfeld bewegt, so daß Daten nicht in Ströme und aus Strömen konvertiert werden müssen. Deshalb wird ein Mehraufwand vermieden, selbst derjenige, der auftritt, wenn ein Problem nicht in einen einzelnen VAP paßt. Megawortspeicher und zukünftige Gigawortspeicher sind kein Problem für Xputer, so daß Datenverarbeitungsabläufe nicht auf verschiedene Hardwarekomponenten partitioniert werden müssen.

Xpiler haben eine optimale Ziel-Hardware um – verglichen mit von-Neumann-Computern – moderate Beschleunigungsfaktoren für alle Arten von Berechnungen zu erzielen, jedoch zwischen 50 und mehr als 1000 für G-Algorithmen. Deshalb können Xpiler hochgradig deterministische Datenzugriffsabläufe finden und Parallelismus in einer voll deterministischen Art vordefinieren, was eine enorme Aufwandsreduktion bedeutet. Gleichzeitig ergibt sich ein deterministischer Speichermechanismus des Cachespeichers bei Xputern, was ebenfalls neu ist. Der probabilistische Cachemechanismus von (traditionellen) Computern ist ein Glücksspiel, das das Problem des Hase-und-Igel-Rennens zwischen Entwurf und Technologie, wie es bei RISC, CISC und CRISPs typisch ist, nicht löst.

Wegen der Eigenschaften ihrer Zielhardware können konventionelle Compiler für parallele Rechner nur eingeschränkte Optimierungserfolge erzielen. Compiler für Vektorprozessoren erreichen nur unwesentlich mehr Verbesserungen als sie durch Schleifenaustausch von geschachtelten Schleifen gewinnen. VLIW-Compiler erreichen ermutigende Resultate hauptsächlich für S-Algorithmen, jedoch nicht für G-Algorithmen. VLSI-Array-Compiler überlassen einen Teil des Aufwandes dem sequentiell arbeitenden Wirtsrechner. Deshalb glauben wir, daß für einen Großteil von Anwendungsgebieten Xputer ein wettbewerbsfähiges Konzept über alle anderen Ansätze hat.

Xputer am VMEbus

Xputer-Prinzipien bieten ein großes Feld für verschiedene Architekturansätze. So können beispielsweise auf bestimmte Anwendungsgebiete optimierte Xputer-Architekturen entwickelt werden, wie etwa mit dem Ziel einer noch billigeren Massenproduktion. Sonst programmierbare Teile von Xputern lassen sich bei Bedarf nach der Entwicklungsphase auch in Gate Arrays oder andere ASIC-Strukturen „gießen". Die Fortentwicklung von Technologie und Architektur der PLD-Schaltungen werden in Zukunft zu noch leistungsfähigeren Xputern führen. Ein neues interessantes Forschungsgebiet ist eröffnet

worden, zumal hier Hardware-Architektur und Compiler-Technik (Xpiler-Technik) wesentlich enger miteinander verzahnt sind als üblich, und Xpiler sozusagen Hybriden zwischen traditionellen Compilern und Silicon-Compilern sind.

Ein Beispiel einer Xputer-Architektur ist die in Kaiserslautern realisierte experimentelle Version des MoM-Xputers (MoM = Map oriented Machine). Sie verfügt mit dem VMEbus über eine Normschnittstelle zur Ankopplung an konventionelle Computer. Diese Architektur benutzt als Hostrechner ein VMEbus-System der Firma Eltec. Die Entwicklungsumgebung beinhaltet die Sprache MoPL (Map-oriented Programming Language) sowie einen syntaxgesteuerten Editor. Bestandteil von MoPL ist PaDL (Pattern Definition Language) mit einem grafischen Pattern-Editor, welcher das Programmieren von Pattern-Matching-Applikationen effizient unterstützt. □

VMEbus-Karte mit Transputer

Archipel, Hersteller von Transputer-Systemen, bietet mit der Karte Volvox-1/V eine Schnittstelle zu VMEbus-Systemen an.

Die wahlweise mit einem Transputer T425, T800 oder T805 bestückbare Doppeleuropakarte eignet sich insbesondere zur Ankoppelung von VMEbus-Rechnern (z. B. Workstations von Sun oder Cetia) an Transputer-Subsysteme. Der lokale Speicher der VMEbus-Platine ist nach Informationen des französichen Herstellers bis auf 32 MByte DRAM ausbaubar.

Risc-Platinen von Tadpole

Der britische VMEbus-Hersteller kündigte Zentraleinheiten mit dem 88000 von Motorola und dem 80960 von Intel an.

Tadpole hatte als einer der ersten eine VMEbus-Karte mit dem Risc-Chipsatz 88000 angekündigt. Die seit kurzem verfügbare Mehrprozessorversion TP881V verwendet die Hypermodule von Motorola. Der aus zwei Baugruppen bestehende Kartensatz, für den eine Unix-Implementierung verfügbar ist, hat einen VSB-Anschluß, einen Ethernet-Controller und bis zu 64 MByte DRAM.

Außerdem zeigte die in Cambridge beheimatete Firma auf der Buscon im letzten Herbst eine VMEbus-Karte mit dem 80960. Diesen Risc-Prozessor von Intel benutzt auch Heurikon auf einem VMEbus-Board (siehe VMEbus 1989, H. 6, S. 18).

Array-Prozessor

Die Doppeleuropakarte VAP-AP von DSP Systems enthält einen Array-Prozessor, für den eine Bibliothek mit Signalverarbeitungsroutinen verfügbar ist.

Die VMEbus-Karte VAP-AP kann pro Sekunde 10 Mio. Operationen ausführen; eine 15-Mops-Version ist ebenfalls lieferbar. Der kalifornische Hersteller bietet als Ergänzung noch eine Cachespeicher-Platine mit Dual-ported-RAMs (VAP-32K) und eine 32-KWorte-Baugruppe (VAP-FS) an. Für Anwender, die den Array-Prozessor in eine Sun-Workstation integrieren wollen, ist ein Unix-Treiberprogramm verfügbar. DSP³ Systems liefert außerdem A/D- und D/A-Karten mit einer Auflösung von 12 Bit bei 20 MHz. Die VMEbus-Palette der 1981 gegründeten Firma wird von einer Platine mit DR11W- oder DRV11W-Schnittstelle abgerundet.

Risc-Board im Dreifacheuropaformat

Mips Computer Systems, Entwickler der Risc-Prozessoren R2000 und R3000, bietet auch VMEbus-Karten mit diesen Chips an.

Die Niederlassung in München vertreibt neben den VMEbus-Platinen auch noch die Komplettsysteme, die auf den CPU-Karten im Dreifacheuropaformat basieren. Als Betriebssystem steht eine Unix-Portierung unter der Bezeichnung Risc/OS zur Verfügung. Sie entspricht System V.3.0 und hat einige Berkeley-Erweiterungen. Die GSE, München, vertreibt als „Value Added Reseller" diese Systeme mit verschiedenen Datenbanken und CAD-Programmen.

An die VMEbus-Platine mit dem R3000 kann man bis zu vier Speicherkarten R3250 anschließen, die jeweils eine Kapazität von 32 MByte aufweisen. Der Zugriff auf ein 32-Bit-Datum erfolgt in 40 ns. Neben dem R3000 befinden sich auf der Karte noch der Coprozessor R3010 sowie zwei serielle Schnittstellen. Boards mit dem R2000 sind ebenfalls lieferbar.

Mehrprozessorsystem

Der von Delta t, Hamburg, entwickelte Computer „Muck" basiert auf dem Risc-Prozessor RTX-2000 von Harris, der speziell für Forth-Anwendungen konzipiert wurde.

Das echtzeitfähige Mehrprozessorsystem im Einfacheuropaformat ist über Mailboxes in einem Dual-ported-RAM an VMEbus-Systeme ankoppelbar. Die Speicherkarte verfügt über eine VMEbus-Schnittstelle. Das System Muck hat ein eigenes Bussystem, an dem digitale und analoge E/A-Baugruppen anschließbar sind. Als Programmiersprache und Betriebssystem liefert der Hersteller eine Forth-83-Implementierung (Compiler und Interpreter).