KFA

FORSCHUNGSZENTRUM JÜLICH GmbH

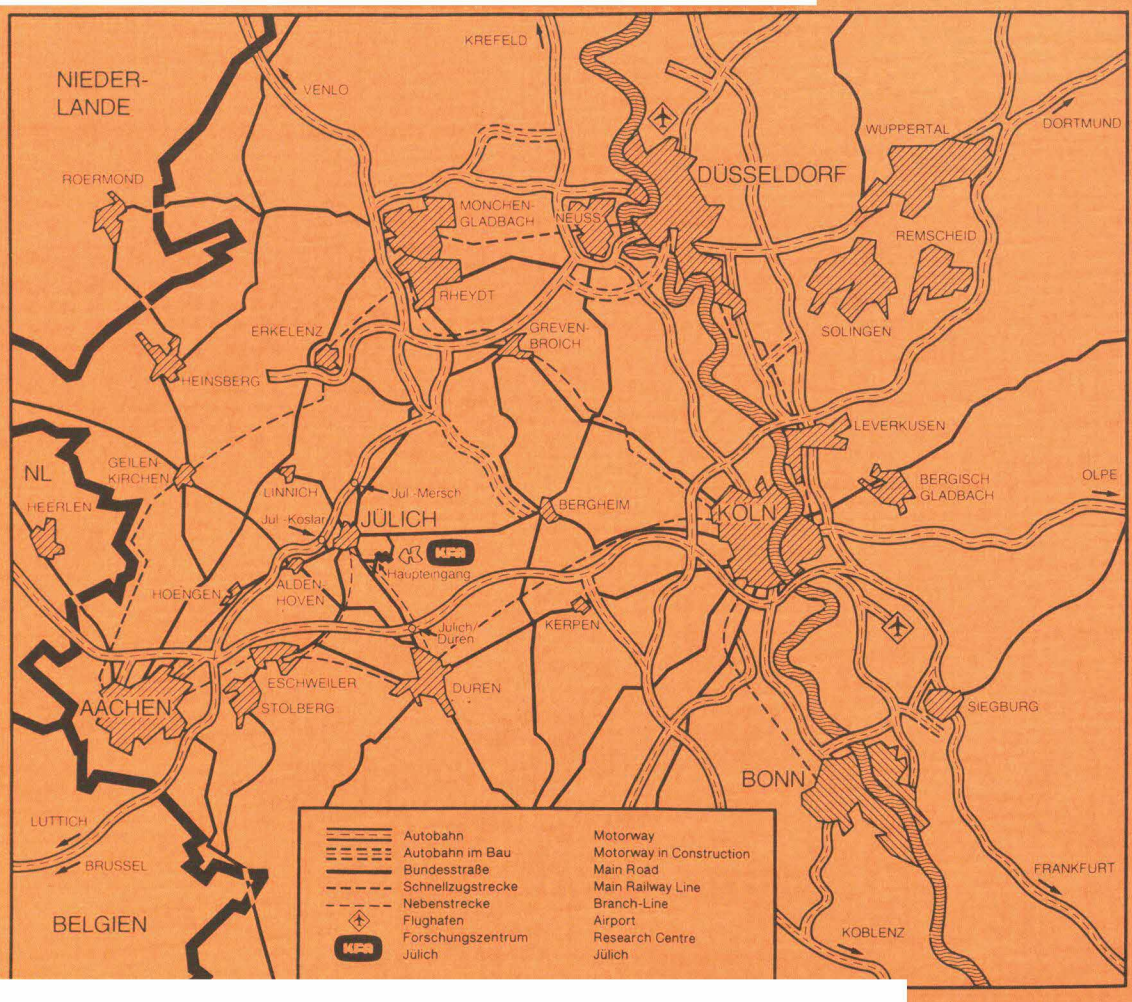
Zentralinstitut für Angewandte Mathematik

Vergleich der Speicherarchitekturen der Mehrprozessor-Vektorrechner CRAY X-MP und CRAY Y-MP anhand vektorisierender Algorithmen

von

Gerd Hofemann

Jül-Spez-555
März 1990
ISSN 0343-7639



Forschungszentrum Jülich: Spezielle Berichte Nr. 555

Zentralinstitut für Angewandte Mathematik Jül-Spez-555

Zu beziehen durch: ZENTRALBIBLIOTHEK · Forschungszentrum Jülich GmbH

Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)

Telefon: 02461/61-0 · Telex: 833556 kf d

Vergleich der Speicherarchitekturen der Mehrprozessor-Vektorrechner CRAY X-MP und CRAY Y-MP anhand vektorisierender Algorithmen

von

Gerd Hofemann

Inhaltsverzeichnis

1.0 Einleitung

Höchstleistungsrechner sind in der Wissenschaft inzwischen ein eingeführtes Hilfsmittel zur Lösung rechenintensiver Probleme geworden. Trotz der hohen Leistungsfähigkeit dieser Rechner gibt es eine Vielzahl von Aufgabenstellungen, die zu komplex sind, um mit derzeit verfügbaren Rechnern bearbeitet zu werden [Hoßf,88]. Die Klasse der Problemstellungen, die erst durch die hohe Leistungsfähigkeit neuester Supercomputer sinnvoll bearbeitet werden kann, stellt für die Höchstleistungsrechner ein wichtiges Aufgabenfeld dar.

Die Grenzen der Leistungsfähigkeit von Supercomputern werden entscheidend von der vorhandenen Software-Entwicklungsumgebung und hier speziell durch die verfügbaren Compiler mitbestimmt. Moderne Compiler sind in der Lage, ohne spezielle Erweiterungen üblicher Programmiersprachen die Leistungsfähigkeit der Rechner in einem hohen Maße verfügbar zu machen. Für darüber hinausgehende Programmoptimierungen, wie sie gerade die oben beschriebene Klasse von Problemen erfordert, ist eine eingehende Analyse der Rechner-Architektur und der Eigenschaften der Software-Umgebung unerläßlich. Diese Analyse kann Hinweise zur Optimierung des Gesamtsystems sowohl für den Entwickler und den Betreiber als auch für den Anwender des Rechners geben. In der vorliegenden Arbeit wird besonders der Anwenderstandpunkt berücksichtigt.

Als typische Vertreter der aktuellen Supercomputer werden in der vorliegenden Arbeit die beiden Mehrprozessorrechner CRAY X-MP/416 und CRAY Y-MP8/832 untersucht. Die einzelnen Prozessoren der hier betrachteten CRAY-Systeme sind als Vektorrechner ausgelegt und bieten bei der Vektorverarbeitung die Möglichkeit, mehrere Zugriffe gleichzeitig auf den Hauptspeicher durchzuführen. Der Aufbau der Funktionseinheiten für die Verarbeitung von Gleitkommazahlen als Pipelines hat in dieser Architektur das Gewicht der Arithmetik bei der Ausführungszeit vermindert und zusätzlich hin zu den Auswirkungen einer intensiveren Nutzung des Speichers verschoben. Im Gegensatz zu Mehrprozessorrechnern mit lokalem Speicher, bei denen die Interprozessorkommunikation den Engpaß im Datentransfer zwischen den CPUs und dem verteilten Speicher darstellt [SaSc,89], hat sich bei Mehrprozessorsystemen mit gemeinsamem Hauptspeicher gerade dieser als kritische Ressource des Systems erwiesen. Der Aufbau des Gesamtsystems als Mehrprozessorrechner aus Prozessoren mit jeweils mehrfachem Speicherzugriff stellt hohe Anforderungen an den Aufbau des Hauptspeichers und die Auflösung von Konflikten bei Speicherzugriffen. Deshalb ist es wichtig, Untersuchungen im Bereich der Speichersysteme durchzuführen. Dies ist besonders dann der Fall, wenn, wie es beim Übergang von der CRAY X-MP zur CRAY Y-MP geschehen ist, grundlegende Veränderungen in wichtigen Bereichen einer Architektur (erhöhte Anzahl der Prozessoren, veränderte Struktur des Hauptspeichers) vorgenommen worden sind ([CRAY,85],[CRAY,88b]).

Zum Vergleich der Speicherarchitekturen bieten sich zwei Gesichtspunkte an:

- der Vergleich der Architekturmerkmale,
- der Vergleich der Auswirkungen der veränderten Architektur auf die Ausführungszeit von Programmen.

Der Vergleich der Auswirkung der Speicherarchitekturen auf das Laufzeitverhalten von Programmen beschränkt sich nicht nur auf die Gegenüberstellung der Laufzeitmessungen, sondern besteht auch in der Zurückführung des Zeitverhaltens auf die ausschlaggebenden Systemkomponenten. Hierzu kann man Methoden der Leistungsanalyse (Performance Analysis) verwenden. Diese lassen sich in drei Kategorien einteilen [Schn,86]:

1. Die Messung am realen System mit Hilfe spezieller Software- und Hardware-Einrichtungen,
2. die Simulation, d.h. die Nachbildung des zu untersuchenden Systems und des Ablaufgeschehens mit Hilfe eines Programms in einer Datenverarbeitungsanlage,
3. die mathematischen Methoden, insbesondere die graphentheoretischen und die wahrscheinlichkeitstheoretischen Methoden.

Eine Zusammenstellung der Ergebnisse einiger Arbeiten im Bereich der mathematischen Methoden findet sich in [Oed,85]. Darüber hinaus beschäftigt sich [Bail,87] mit der gleichen Problematik. Beispiele von Simulationsmodellen für Speichersysteme in CRAY-Rechnern und deren Ergebnisse finden sich in [ChSm,86], [CaEl,85] und [Oed,85]. Messungen des Zeitverhaltens der beiden Rechner sind bereits in [Dete,89] und [FaSc,89] vorgestellt worden und werden in der vorliegenden Arbeit vertieft.

Die für eine Leistungsanalyse grundlegende Beschreibung der beiden CRAY-Rechner X-MP/416 und Y-MP8/832 wird in Kapitel 2 durchgeführt. Neben der Beschreibung der einzelnen Prozessoren werden der Aufbau und die Konfliktauflösung der Speichersysteme ausführlich vorgestellt. Dabei werden besonders die Unterschiede zwischen den Speichersystemen der CRAY X-MP und CRAY Y-MP herausgearbeitet.

In Kapitel 3 wird das Zeitverhalten von Programmkernen betrachtet, die wichtige Anwendungsfelder von Vektorrechnern abdecken. Aufbauend auf die in Kapitel 2 gefundenen Systemeigenschaften sind diese Programmkerne so gewählt, daß sie gezielt einzelne Komponenten des Rechners belasten und deren Zeitverhalten beschreiben. Dabei werden vektorisierende Programme untersucht, da nur mit solchen Programmkernen eine starke Belastung des Speichers zu erreichen ist. Als Bewertungsmaßstab werden die von Hockney [HoJe,88] eingeführten Leistungskenngrößen r_∞ und $n_{1/2}$ bestimmt und diese zwischen

- den beiden Rechnern,
- den verschiedenen Kernen und
- dem Ein- und Mehrprozessorbetrieb

verglichen. Die Kenngrößen bilden ein analytisches Hilfsmittel zur Beschreibung der Rechnerleistung in Abhängigkeit von der betrachteten Problemgröße. Bei der Ermittlung der Kenngrößen wird der Weg von der Untersuchung sehr maschinennaher Programmkerne hin zu immer anwendungsbezogeneren Programmen gewählt.

In Kapitel 4 werden die an den Programmkernen gewonnenen Erfahrungen zur Beurteilung der Rechnerleistung bei der Bearbeitung von Algorithmen herangezogen. Da Vektorrechner eine besonders hohe Leistungsfähigkeit bei der Verarbeitung von Vektoren,

Matrizen oder ähnlich regelmäßigen Datenstrukturen erreichen, werden zwei vektorisierende Algorithmen untersucht, die Matrizen verwenden :

- als Beispiel aus der linearen Algebra verschiedene Programmvarianten der Matrixmultiplikation,
- aus der Graphentheorie der Algorithmus zur Bestimmung kürzester Wege nach Warshall-Floyd.

Für beide Algorithmen werden sowohl Einprozessor- als auch Mehrprozessormessungen betrachtet. Kapitel 5 faßt die gewonnenen Ergebnisse zusammen.

2.0 Der Aufbau der betrachteten CRAY-Systeme

Die Rechner CRAY X-MP und CRAY Y-MP weisen viele Gemeinsamkeiten in der Architektur auf. Im folgenden wird zuerst ein Abriß der Entwicklung der CRAY-Rechner bis zur CRAY Y-MP gegeben. Es schließt sich eine kurze Übersicht über die beiden hier betrachteten Rechnersysteme an. Diese wird durch eine Beschreibung der Architektur einer einzelnen CPU ergänzt. Danach werden in einem eigenen Abschnitt der Aufbau und die spezifischen Eigenschaften des Hauptspeichers vorgestellt. Der letzte Abschnitt dieses Kapitels faßt dann die Unterschiede der beiden hier betrachteten CRAY-Systeme zusammen. Die Bezeichnung **CRAY-Rechner** steht dabei synonym für CRAY X-MP und CRAY Y-MP; CRAY-2 Systeme werden hier nicht betrachtet. Als Referenz für technische Details der CRAY-Rechner sind für dieses gesamte Kapitel [CRAY,85] und [CRAY,88b] verwendet worden.

2.1 Historische Entwicklung

Dieser Abschnitt beschreibt die Entwicklung der Konzepte der CRAY-Architektur und die unterschiedlichen CRAY-Rechnersysteme. Der erste CRAY-Rechner ist die CRAY-1. Die CPU dieses Einprozessorsystems besitzt bereits die meisten prinzipiellen Merkmale der CPUs der Nachfolgesysteme X-MP und Y-MP, nämlich

- Register-Architektur,
- Aufteilung in Adreß-, Skalar- und Vektorteil,
- Vektorverarbeitung,
- unabhängige Funktionseinheiten,
- Vektorfunktionseinheiten als Pipeline.

Bezüglich der unabhängigen Funktionseinheiten und der Instruktionsverarbeitung ist die CRAY-1 der logische Nachfolger der CDC Cyber 7600 [IbTo,89]. Die Einführung der Vektorverarbeitung löst dabei das Problem, daß die Instruktionsverarbeitung eine Ausführung einer Gleitkommaoperation pro Takt nicht zuläßt.

Ein Hauptmerkmal des Speichers bei der CRAY-1 ist seine Aufteilung in Speicherbänke. Das Prinzip der Bankaufteilung des Hauptspeichers zur Erhöhung der Speicherbandbreite ist bereits in den 50-er Jahren entwickelt worden und z.B. im Rechner IBM 709411 implementiert [Ston,80]. Ein System, das das Prinzip der **Speicherverschränkung** (Interleaving) zusammen mit Vektorverarbeitung benutzt, ist der Rechner CDC STAR-100 [Gilo,81]. Eine interessante Komponente des Speicherzugriffssystems dieses Rechners ist die Datenstromeinheit. Diese organisiert eigenständig die Datentransporte vom und zum Speicher und ist als mikroprogrammierte Einheit ausgeführt. Diese Datenstromeinheit erinnert an die „Ports" der CRAY-Architektur. Im Gegensatz dazu fehlen beim STAR-100 und seinem Nachfolger, dem Vektorrechner Cyber 205 [IbTo,89], die Vektorregister als schneller Zwischenspeicher im Prozessor. Ein Engpaß der Speicherarchitektur bei der CRAY-1 ist, daß nur ein bidirektionaler Weg zum Speicher zur Verfügung steht.

Die Architektur der CRAY X-MP bildet den Schritt von einem Vektorrechner mit einem Prozessor zum Mehrprozessorrechner mit gemeinsamem Hauptspeicher. Hierbei wird

die Bankaufteilung des Hauptspeichers außer für die Erhöhung der Zugriffsgeschwindigkeit auch auf zwei Arten für unabhängigen Zugriff genutzt:

- für mehrere parallele Datentransporte zu einem Prozessor:
  Erstmals stehen jeder CPU vier Wege zum Speicher zur Verfügung, wovon drei für Operanden benutzt werden können.
- für parallele Datentransporte zu mehreren Prozessoren:
  Jeder Prozessor kann auf den Speicher zugreifen, wenn der Zugriff auf nicht belegte, voneinander verschiedene Speicherbänke erfolgt.

Eine weitere Verbesserung der Architektur der CRAY X-MP gegenüber der CRAY-1 findet sich in der automatischen Verkettung von Vektoroperationen (Auto-Chaining) [HoJe,88]. Eine anschauliche Beschreibung der Design-Ideen der CRAY X-MP findet man in [AuBr,89].
Der Vierprozessorrechner CRAY X-MP enthält als Neuerung die Möglichkeit der indizierten Adressierung durch Vektoren, das **Gather/Scatter**. Diese Funktion ist bereits vorher in der Cyber 205 implementiert worden.

Bei der Weiterentwicklung der CRAY-Systeme von der CRAY-1 zur CRAY X-MP wurde die Zykluszeit der Zentralprozessoren verringert, wobei die Bank-Zykluszeit der Speicherbänke mit dieser Entwicklung Schritt gehalten hat. Außerdem ist die Anzahl der Bänke von 8 bzw. 16 Bänken bei der CRAY-1 auf 64 bei der CRAY X-MP mit 4 Prozessoren erhöht worden.

Die CRAY Y-MP bildet den nächsten Schritt in Richtung auf mehr Prozessoren und unterstützt mehr Speicherbänke (256) und kürzere Prozessor- und Speicherzykluszeiten. Die Gegenüberstellung von CRAY Y-MP und CRAY X-MP bildet den Kern der folgenden Abschnitte.

2.2 Die Systeme im Überblick

Die hier betrachteten CRAY-Rechner sind die X-MP/416 und die Y-MP8/832 (Abb. 1), wie sie bei der KFA Jülich installiert sind [ZAM,89]. Die CRAY X-MP/416 verfügt über vier Prozessoren und 16 Megaworte Hauptspeicher, die CRAY Y-MP8/832 über acht Prozessoren und 32 Megaworte Hauptspeicher. Die Wortgröße beträgt bei beiden Rechnern 64 Bit. Der Hauptspeicher ist bei beiden Systemen als gemeinsamer Hauptspeicher für alle Prozessoren aufgebaut. Ein Netzwerk verbindet die Prozessoren mit dem Hauptspeicher. Die CPUs eines Systems sind gleichartig aufgebaut und werden mit einer Zykluszeit von 8.5 Nanosekunden bei der CRAY X-MP gegenüber 6.0 Nanosekunden bei der CRAY Y-MP getaktet (Tab. 1).

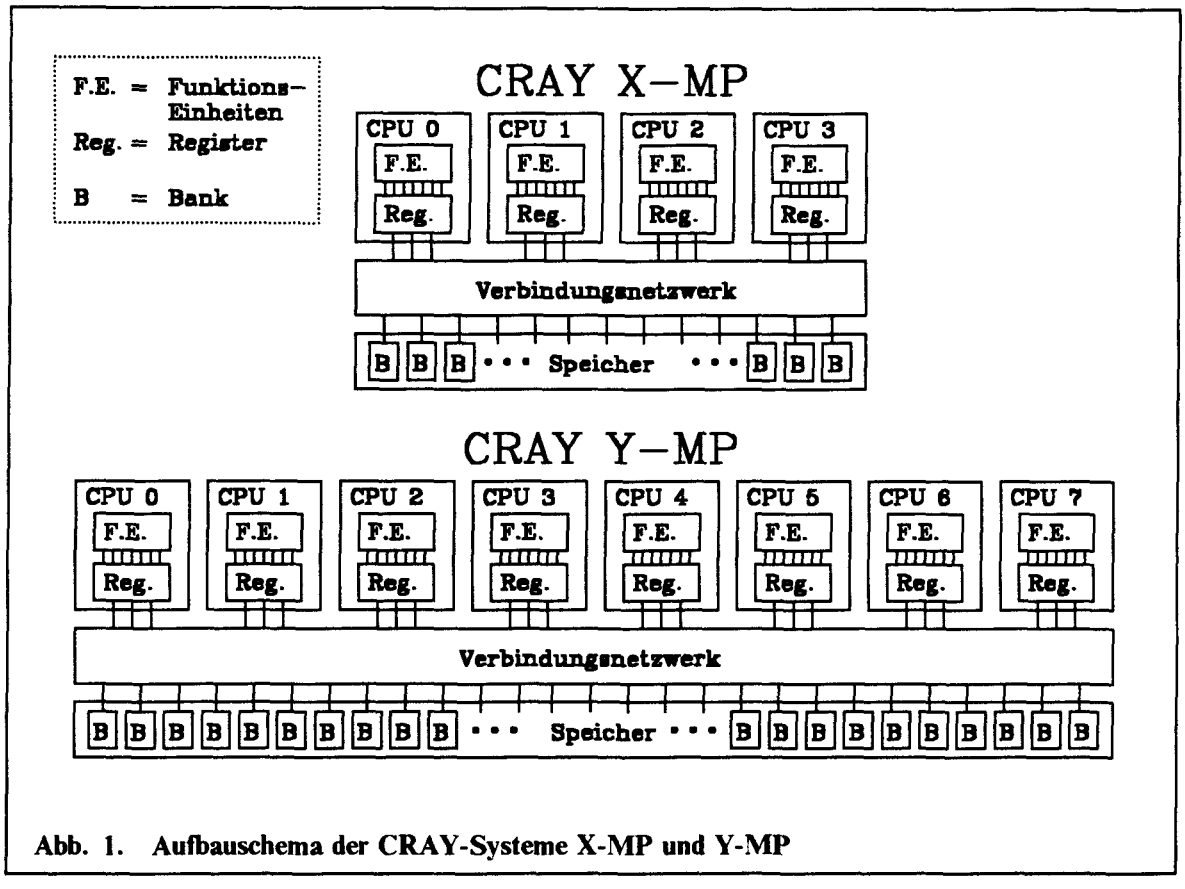


Abb. 1. Aufbauschema der CRAY-Systeme X-MP und Y-MP

Eigenschaft	CRAY X-MP	CRAY Y-MP
Anzahl Prozessoren	4	8
Prozessorzykluszeit (Taktzeit)	8,5 ns	6,0 ns
Speicher	16 MWorte	32 MWorte
- Bänke	64	256
- Sektionen	4	4
- Untersektionen	-	8 pro Sektion
- Zykluszeit	34 ns	30 ns
- Zykluszeit in Takten	4	5

Tab. 1. Kenndaten der CRAY-Systeme der X-MP/416 und Y-MP8/832 in der KFA Jülich

2.3 Der Aufbau eines Prozessors

Bei der Betrachtung der Speicherarchitekturen ist zu berücksichtigen, in welcher Weise der Speicher durch die CPU angesprochen wird. Ein wichtiger Aspekt ist z.B. die registerorientierte Verarbeitung bei CRAY-Rechnern, die die Art und Menge der Datentransporte vom und zum Speicher stark beeinflußt. Deshalb wird in diesem Abschnitt zuerst der Aufbau einer CPU aus Adreßteil, Skalarteil, Vektorteil, Instruktionsverarbeitung und Meßeinrichtungen beschrieben (Abb. 2). Weitere Abschnitte beschreiben die gleichzeitige Nutzung der Prozessorkomponenten und die Verbindung der CPUs eines Systems über gemeinsame Register.

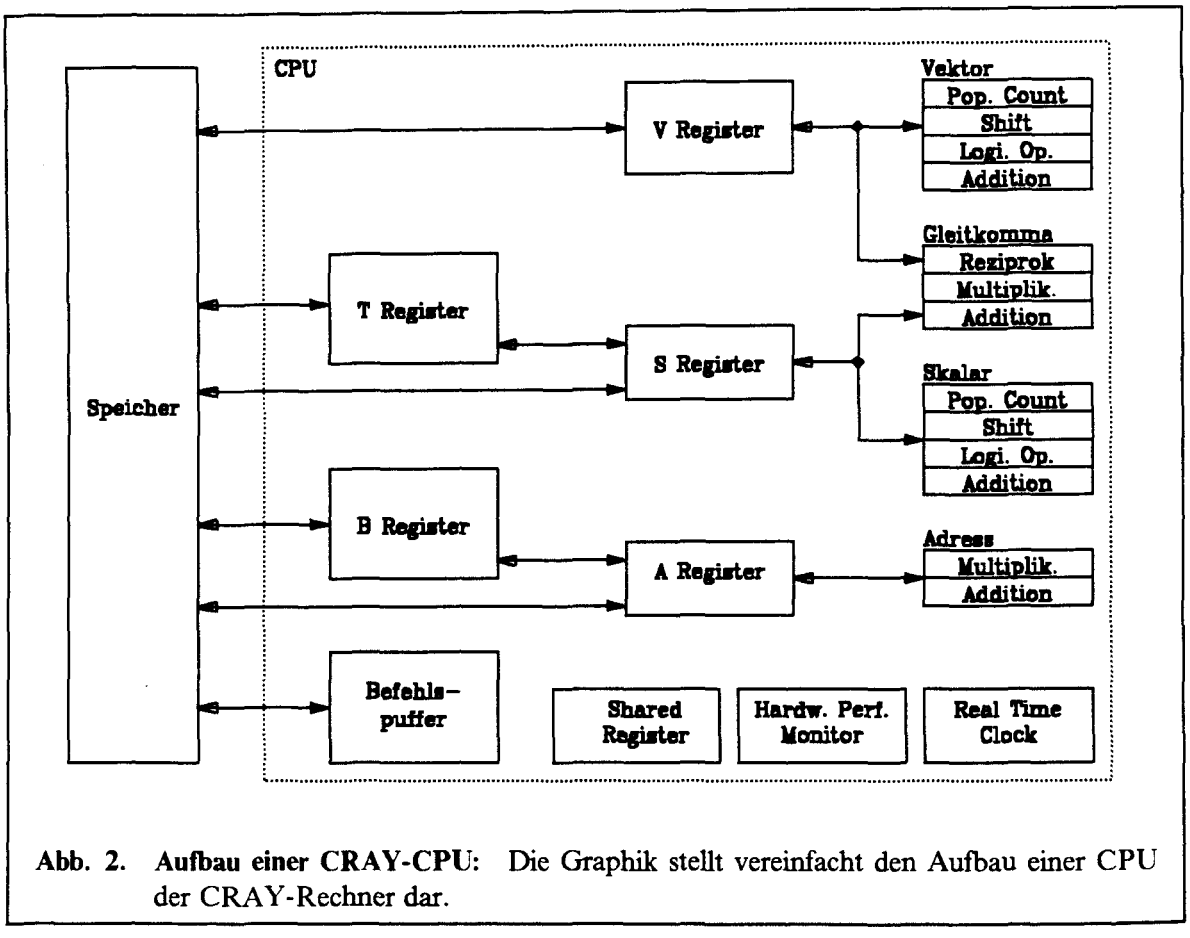


Abb. 2. Aufbau einer CRAY-CPU: Die Graphik stellt vereinfacht den Aufbau einer CPU der CRAY-Rechner dar.

2.3.1 Der Adreßteil

Die Adressen bei CRAY-Systemen sind Wortadressen. Der Adreßraum der CRAY-Rechner hat sich mit der Entwicklung hin zu größerem Speicher strukturell verändert. Ursprünglich hatte diese Architektur eine 22-Bit-Adressierung. Die sich ergebende Beschränkung auf vier Megaworte Speicher wird schon bei der neueren Generation der Rechner vom Typ X-MP durch segmentbezogene Adressierung des Hauptspeichers überwunden (EMA = extended memory address mode). Die längere physikalische Adresse wird dabei durch eine Verlängerung der Basisadresse ansprechbar, die Adreßregister bleiben unverändert. Bei der Architektur der Y-MP ist die Breite der Adreßregister und auch der Adreßanteil einiger Befehle auf 32 Bit erweitert worden. Dies

hat Folgen für die Länge des Assembler-Codes. Eine Erhöhung der Ausführungszeit der Befehle ist durch ein zusätzliches paralleles Register in der Instruktionseinheit verhindert worden. Um eine möglichst hohe Kompatibilität zu der CRAY X-MP zu erhalten, kann die CRAY Y-MP aber auch die Adreßrechnung der X-MP Maschinen emulieren (X-Mode).

Für Adreßrechnungen besitzt jede CRAY-CPU acht Adreßregister (A-Register) und als zugehörige Funktionseinheiten Adreßaddierer und Adreßmultiplizierer. Adressen können in der CPU in 64 B-Registern zwischengespeichert werden. Die Inhalte mehrerer B-Register können auch mit einem Befehl vom und zum Hauptspeicher übertragen werden (Blocktransfer).

2.3.2 Der Skalarteil

Bei skalaren Berechnungen wird mit einer Operation jeweils ein Ergebniswert berechnet. Als Operationen stehen arithmetische und logische Grundoperationen zur Verfügung. Arithmetische Operationen können auf ganzzahligen Operanden oder Gleitkommaoperanden durchgeführt werden. Wie beim Adreßteil gibt es hier eine Zweiteilung der Register in acht S-Register für Operanden der skalaren Operationen und 64 T-Register als Zwischenspeicher. Der Inhalt mehrerer T-Register kann auch als Block vom und zum Hauptspeicher übertragen werden. Für ganzzahlige Berechnungen gibt es eigene Funktionseinheiten für die Addition, die logischen und Verschiebe-Operationen, aber nicht für die Multiplikation. Die Multiplikation muß entweder in andere Operationen zerlegt werden oder mit geringerer Genauigkeit (46-Bit) in Gleitkommaarithmetik berechnet werden.

2.3.3 Der Vektorteil

Ein Hauptmerkmal aller CRAY-Rechner ist die Verarbeitung von Operanden als Vektoren. Ein Vektor ist eine Folge von Operanden oder Ergebnissen, die bei der wiederholten Ausführung der gleichen Operation auftritt. Die Operanden werden in Registern zusammengestellt, die bei CRAY-Rechnern 64 Worte umfassen und von denen es acht gibt. Diese Register heißen Vektorregister. Sie dienen als Quelle und Ziel von Vektoroperationen, für die es eigene Funktionseinheiten in jeder CPU gibt.

Für Gleitkommaarithmetik gibt es Funktionseinheiten für Addition, Multiplikation und Reziprokwertbildung. Die Division wird mit Hilfe der Reziprokwertbildung und der Multiplikation durchgeführt. Diese Funktionseinheiten werden auch für skalare Gleitkommaberechnungen benutzt. Weitere Funktionseinheiten zur Vektorverarbeitung führen ganzzahlige Addition, logische Operationen, Bit-Zähloperationen (Population Count, Leading Zero Count) sowie Verschiebeoperationen (Shift) aus. Bei der Operation „Population Count" wird die Anzahl der Bits eines Wortes gezählt, die den Wert Eins besitzen. Die Operation „Leading Zero Count" zählt die Anzahl der führenden Bits eines Wortes, die den Wert Null besitzen.

Die Funktionseinheiten für Vektoroperationen sind als **Pipeline** ausgeführt. Das Pipelineprinzip dient der Erhöhung der Ausführungsgeschwindigkeit einer (aufwendigen) Operation, die auf eine Vielzahl von Operanden angewendet werden soll. Diese Opera-

tion wird in mehrere Teiloperationen zerlegt, die nacheinander ausgeführt die ursprüngliche Operation ergeben. Diesen Vorgang nennt man Segmentierung, die Teiloperationen Segmente. Die Aneinanderreihung der Segmente, insbesondere die hardwaremäßige Realisierung der Segmentsequenz, nennt man Pipeline; die Anzahl der Segmente heißt Länge der Pipeline. Die Segmente der Pipeline können unabhängig voneinander entsprechend der zeitlichen Abfolge verschiedene Operanden bearbeiten. Deshalb können sich soviele Operanden gleichzeitig in Bearbeitung befinden, wie die Pipeline Segmente besitzt. Der Vorteil der Segmentierung liegt darin, daß die einzelnen Teiloperationen der Segmente der Pipeline sehr viel schneller ausgeführt werden können als die Gesamtoperation. Ist das Ergebnis der Verarbeitung des ersten Operanden(paares) am Ende der Pipeline angelangt, so erreicht jedes weitere Ergebnis aus den folgenden Operanden das Ende in jedem weiteren Takt. Die mögliche minimale Taktzykluszeit der gesamten Pipeline wird dabei von der Berechnungszeit des langsamsten Segments bestimmt. Es ergibt sich eine Zeitdauer zwischen Einbringen der Operanden und Erhalten des ersten Ergebnisses, die durch die Länge der jeweiligen Pipeline bestimmt ist. Diese Zeitdauer wird mit **Start-Up-Zeit** bezeichnet. Die Zerlegung der mathematischen Funktionen ist ausführlich am Beispiel CDC STAR-100 in [Gilo,81] beschrieben.

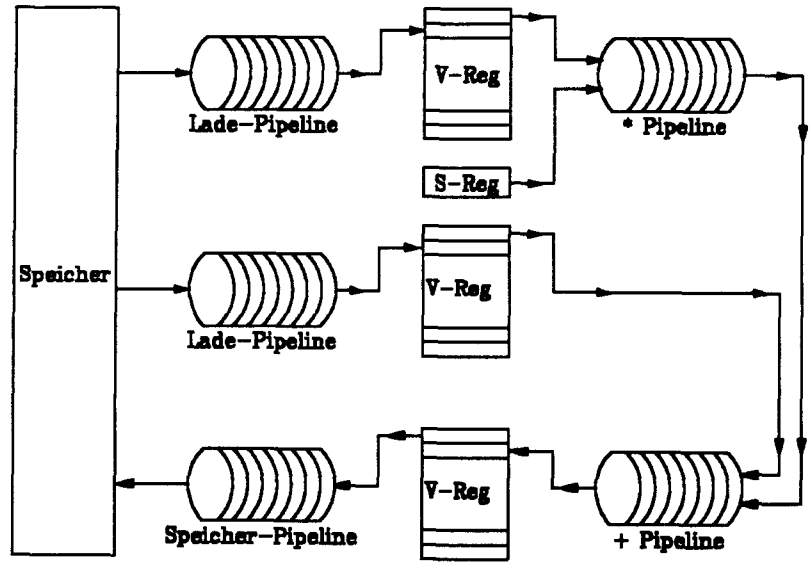


Abb. 3. Chaining: Beispiel für den Einsatz von Chaining bei einer typischen Berechnung.

Bei Funktionseinheiten, die nach dem Pipelineprinzip arbeiten, ist es sinnvoll, die Pipelines zu verketten, wenn die Ergebnisse einer Pipeline die Operanden einer anderen Pipeline bilden (Abb. 3). Verketten bedeutet dabei, die einzelnen Elemente des Ergebnisvektors einer Pipeline direkt in die nächste Pipeline einzuspeisen und nicht erst die Pipeline-Operation für alle Vektorelemente durchzuführen, die mit der Operation in der zweiten Pipeline begonnen wird. So verbindet man mehrere Pipelines faktisch zu einer größeren Pipeline, mit dem entsprechend höheren Geschwindigkeitsgewinn und der entsprechend längeren Start-Up-Zeit. Diese **Verkettung** (Chaining) ist aber nur möglich,

falls keine der Einzelpipelines zur Laufzeit bereits benutzt wird (vgl. Reservierung der Ressourcen in Abschnitt 2.3.4).

Ein zusätzliches Register mit 64 binären Elementen, das Vektormaskenregister, dient der Steuerung von Vektoroperationen. Die CRAY Y-MP unterscheidet sich bezüglich der Vektorregister und Vektorfunktionseinheiten pro CPU nicht von der CRAY X-MP.

Einen interessanten Aspekt der Vektorverarbeitung stellen die benötigten Datenwege dar. Eine Vektoroperation, die in jedem Takt zwei Operandenelemente als Eingabe benötigt, braucht auch zwei getrennte Wege zu den Vektorregistern bzw. zum Hauptspeicher, um die Daten heranzuholen. Zusätzlich muß noch ein Weg für die Ergebnisse zur Verfügung stehen. Den Transport der Daten über Wege kann man als **Datenstrom** bezeichnen. Das Benutzen getrennter Wege für die Datenströme wird im folgenden Datenstromprinzip genannt. Die Anzahl der gleichzeitig möglichen Datenströme kann durch Parallelverarbeitung innerhalb der CPU beim Datentransport zwischen Funktionseinheiten und Registern einen Engpaß darstellen. Meist tritt dieser Engpaß zwischen Speicher und CPU auf.

2.3.4 Der Instruktionsteil

Der Instruktionsteil ist der Teil jeder CRAY-CPU, der Befehlscodes aus dem Hauptspeicher lädt, entschlüsselt und dann die gewünschten Aktionen anstößt. Die Instruktionen eines Rechners stellen die Schnittstelle zwischen Hardware und Programmierung dar. Die Instruktionen müssen deshalb so aufgebaut sein, daß die einzelnen Hardware-Komponenten des Systems möglichst einfach anzusprechen sind.

Eine wichtige Eigenschaft der CRAY-Systeme ist, daß die verschiedenen Prozessorkomponenten mehrfach bzw. spezialisiert vorhanden sind und unabhängig voneinander betrieben werden können. Prozessorkomponenten sind hier die mathematischen und logischen Funktionseinheiten, die Register und die Speicherzugänge. Der parallele Betrieb der Prozessorkomponenten wird beim Befehlssatz der CRAY-Systeme durch die Unabhängigkeit der Befehlsausführung von der Befehlsentschlüsselung und durch die Eigenschaft vieler Befehle erreicht, längere Ausführungszeiten als Aufsetzzeiten zu haben. Die Anzahl der Takte, in der der Code eines Befehls dekodiert und zur Ausführung gebracht wird, ist dann geringer als die Anzahl der Takte der Ausführung in den Prozessorkomponenten. Die Anzahl der Takte, die zwischen dem Start der Aktionen des vorhergehenden Befehls und dem Aufsetzen des aktuellen Befehls vergeht, nennt man **Aufsetzzeit** (Issue Time). Die Aufsetzzeit für Vektorbefehle ist in der Regel kleiner als die Ausführungszeit. Die Aufsetzzeit einer Vektormultiplikation ist z.B. im günstigsten Fall nur einen Takt lang, die Ergebnisse der Vektormultiplikation stehen aber frühestens nach $12 + n$ Takten (n Vektorlänge) vollständig zur Verfügung. Es findet keine ausdrückliche Programmierung der Parallelität aller Prozessorkomponenten statt, sondern die Reihenfolge der Befehle bestimmt den Anteil der Parallelverarbeitung. Das Programm ist nicht für das richtige Zusammenspiel der Komponenten verantwortlich, jedoch kann durch eine geeignete Anordnung der Instruktionen eines Programms die Ausführungszeit optimiert werden.

Um einen Befehl auszuführen, müssen alle für diesen Befehl benötigten Ressourcen verfügbar sein. **Ressourcen** bedeuten in diesem Zusammenhang die Register, die Funk-

tionseinheiten und die Speicherzugänge (Abb. 4). Dazu hat die Instruktionseinheit die Möglichkeit, Reservierungen der Ressourcen abzufragen. Die Ausführung eines Befehls beginnt erst, wenn alle benötigten Ressourcen frei sind. Dies hat die Reservierung dieser Ressourcen zur Folge, die durch das Ende der jeweiligen Nutzung automatisch wieder aufgehoben wird.

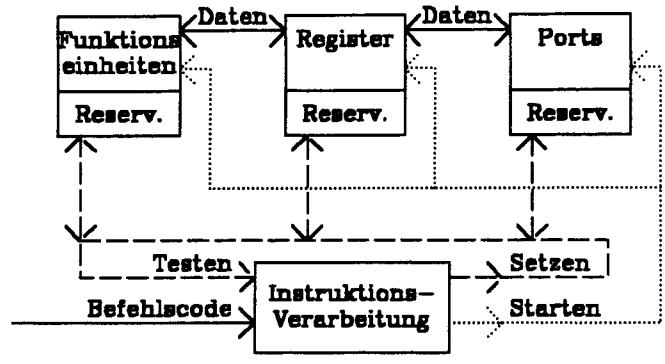


Abb. 4. Reservierung der Ressourcen durch die Instruktionseinheit

Befehle aus dem Hauptspeicher werden vor ihrer Dekodierung in einen von vier Instruktionspuffern geladen. Jeder dieser Puffer kann 32 Worte zu 64 Bit aufnehmen. Ein Ladevorgang füllt immer den gesamten Puffer. Die Befehle aus dem Befehlspuffer erreichen dann den Instruktionsdekodierer. Dieser entschlüsselt den Befehlscode, holt, wenn nötig, zugehörige Operanden und startet die eigentliche Operation, wenn alle dazu benötigten Ressourcen verfügbar sind. Eine Besonderheit der skalaren Lade- und Speicheroperationen ist es, daß bis zu vier dieser Operationen aufgesetzt werden können, ohne daß alle Ressourcen verfügbar sind. Der Programmzähler steuert das Laden des nächsten Befehls. Ist dieser Befehl nicht in einem der vier Befehlspuffer enthalten, wird der Inhalt eines Befehlspuffers neu geladen.

Der Operationscode wird in 16-Bit-Abschnitte, die Parcels, aufgeteilt. Die Aufsetzzeit eines Befehls wird im günstigen Fall von der Länge des Operationscodes (1,2,3 Parcel) bestimmt, im ungünstigen Fall wird sie aber von der Verfügbarkeit der Ressourcen festgelegt. Deshalb ist es eine wichtige Aufgabe eines Hochsprachen-Compilers, die Befehle so zu ordnen, daß die Ressourcen möglichst parallel belegt werden und beim Aufsetzen eines Befehls die benötigten Ressourcen gerade unbenutzt sind. Dies kann aber gerade bei Messungen von Ausführungszeiten zu ungewünschten Effekten führen.

2.3.5 Die Verbindung der Prozessoren

Die Prozessoren der CRAY-Systeme sind über einen gemeinsamen Hauptspeicher, aber auch über gemeinsame Register, die Shared Register, miteinander verbunden. Diese sind zu Registersätzen (Cluster) zusammengefaßt. Es ist jeweils ein Registersatz mehr als Prozessoren vorhanden, d.h. fünf Registersätze bei der CRAY X-MP mit vier Prozessoren und neun bei der CRAY Y-MP. Ein gemeinsamer Registersatz besteht aus acht Adreßregistern (SB), acht Skalarregistern (ST) und 32 binären Semaphorregistern (SM). Jeder CPU ist immer höchstens ein gemeinsamer Registersatz zugeordnet. Zur Syn-

chronisation mehrerer CPUs existiert ein unteilbarer Befehl, der den Wert null eines Semaphorregisters abwartet und dann dieses Register auf eins setzt (test-and-set).

2.3.6 Sonstige Prozessorkomponenten

Auf eine Erklärung der Komponenten

- Austausch-Pakete (Exchange Package),
- Ein-/Ausgabe-Architektur

wird in diesem Zusammenhang verzichtet. Stattdessen wird auf die einschlägige Literatur verwiesen ([HoJe,88], [HwBr,84], [IbTo,89], [Rege,89]).

2.3.7 Meßeinrichtungen

Alle Zentralprozessoren verfügen über Taktzählregister (Real Time Clock), die man abfragen kann und die damit eine taktbezogene Zeitmessung ermöglichen. Außerdem enthält jede CPU mit dem Hardware-Performance-Monitor eine Zählmöglichkeit für gewisse Systemereignisse wie die Benutzung von Gleitkommaoperationen oder Speicheroperationen. Damit kann man Eigenschaften von Programmen messen, ohne die Ausführung der Programme zu beeinflussen.

2.4 Die Speicherarchitektur der CRAY-Systeme

2.4.1 Das Prinzip des Speicheraufbaus

CRAY-Systeme haben einen Hauptspeicher, der von allen CPUs des jeweiligen Systems gemeinsam benutzt wird. Der Hauptspeicher ist in Bänke aufgeteilt. Diese Bänke stellen die kleinste unabhängige Hardware-Einheit des Speichers dar. Die Bänke sind in vier gleiche Gruppen, die Sektionen, zusammengefaßt. Bei der CRAY Y-MP sind diese Sektionen nochmals in acht **Untersektionen** (Subsections) aufgeteilt. Der Zugriff einer einzelnen CPU auf den Hauptspeicher wird durch vier eigene Speicherzugänge (Ports) hergestellt.

Beim Zugriff auf den Hauptspeicher legt die Adresse der gewünschten Speicherstelle fest, welche Sektion, Untersektion und Bank angesprochen wird. Welchen Port die CPU dabei benutzt, hängt vom Zweck des Zugriffs ab. Es ist zum Beispiel ein Unterschied, ob ein Befehlspuffer geladen wird (Code Fetch) oder der Inhalt eines Vektorregisters gespeichert wird. Bei der Nutzung des Speichers gibt es eine Reihe von Einschränkungen. Zu einem Zeitpunkt ist z.B. nur ein Zugriff auf eine Speicherbank möglich. Die Bank ist dabei für mehr als einen Takt durch diesen Zugriff reserviert. Es gibt Überschneidungen der Wege zum Speicher, die einzelnen Ports einer CPU den gleichzeitigen Zugriff zu einer oder mehreren Bänken des Hauptspeichers versperren. Diese Probleme faßt man in dem Begriff der **Zugriffskonflikte** zusammen. Zugriffsregeln ordnen die Zugriffswünsche zum Hauptspeicher. Einer der Zugriffswünsche erhält das Zugriffsrecht, die anderen Zugriffswünsche müssen warten. Deshalb führen Zugriffskonflikte zur Verlängerung der Ausführungszeit der ablaufenden Programme.

In den folgenden Abschnitten werden die Ports, das Zugriffsnetzwerk und die Bankstruktur im Detail erklärt und die Regeln der Konfliktbehandlung vorgestellt.

2.4.2 Die Speicherzugänge (Ports)

Alle Datentransfers vom und zum Hauptspeicher werden über die Ports abgewickelt (Abb. 5). Ports kann man als „Prozessoren" zur Kommunikation mit dem Hauptspeicher beschreiben. Auf der einen Seite der Ports stehen die Register der CPU, die Befehlspuffer und der I/O-Bereich, auf der anderen Seite das Speicherzugriffsnetzwerk zu den Bänken. Einer CPU sind jeweils vier Ports zugeordnet, die voneinander unabhängig arbeiten. Ein Port wird zuerst „programmiert" und organisiert dann eine Datenübertragung vom oder zum Speicher selbständig. Die Programmierung der Ports geschieht durch Lade- und Speicherbefehle im Programm, aber auch durch I/O-Transfer-Befehle und das Nachladen eines Befehlspuffers. Die Aufgabenaufteilung zwischen den verschiedenen Ports unterscheidet sich bei der CRAY X-MP und der CRAY Y-MP in wichtigen Einzelpunkten. Die vier Ports jeder CPU werden mit Port A,B,C und D bezeichnet. Dabei sind die Ports A und B nur zum Laden von Daten aus dem Speicher, der Port C nur zum Speichern und der Port D in beide Richtungen zu benutzen. (Ausnahme bei Port C ist das Laden eines Befehlspuffers auf der CRAY X-MP, wobei Port C auch zum Laden aus dem Speicher benutzt wird.)

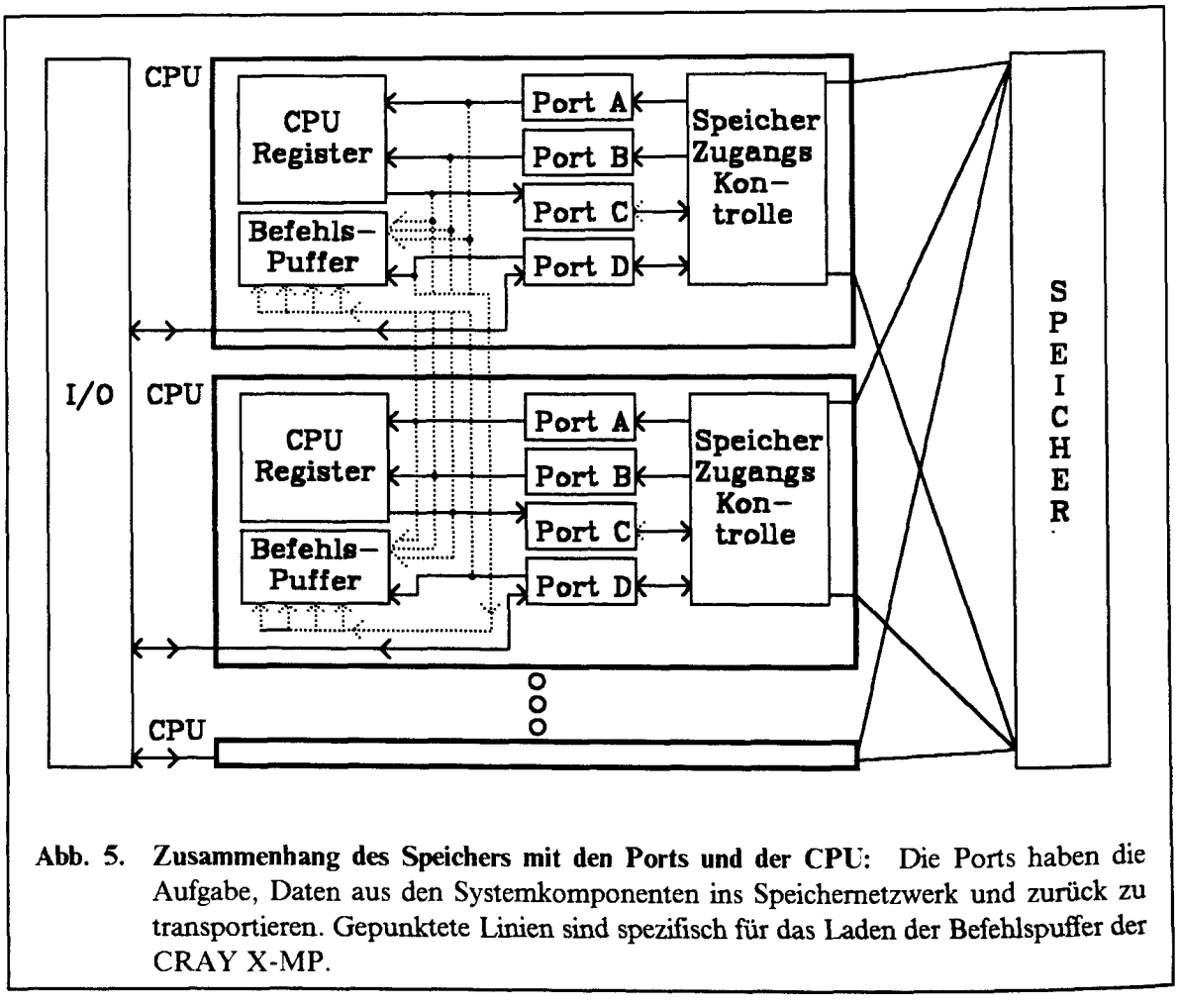


Abb. 5. Zusammenhang des Speichers mit den Ports und der CPU: Die Ports haben die Aufgabe, Daten aus den Systemkomponenten ins Speichernetzwerk und zurück zu transportieren. Gepunktete Linien sind spezifisch für das Laden der Befehlspuffer der CRAY X-MP.

Bei Lade- und Speicherbefehlen werden nur die Ports A und B zum Laden von Daten und der Port C zum Speichern von Daten benutzt. Die Zuordnung des Ladeports (A oder B) erfolgt bei Block- und Vektorbefehlen dynamisch beim Aufsetzen. Port D wird durch Ein-/Ausgabebefehle angesprochen. Diese Aufteilung in zwei Lade- und einen Speicherport beruht auf dem Datenstromprinzip. Zur Verarbeitung von Daten durch einen Prozessor müssen die zu verarbeitenden Daten dem Prozessor aus dem Arbeitsspeicher zugeführt werden und die Ergebnisse in den Arbeitsspeicher zurücktransportiert werden. Geht man dabei von einer zweistelligen Operation aus, so ergeben sich drei Datenströme (zwei Operanden, ein Resultat), die auf drei Ports abgewickelt werden können. Bei Berechnungen mit mehr als drei Datenströmen (z.B. bei Chaining) kann eine Benutzung der Vektorregister oft überflüssige Datentransporte vom und zum Speicher verhindern.

Das Laden der Befehlspuffer unterscheidet sich bei den beiden hier betrachteten CRAY-Systemen. Die CRAY X-MP benutzt dazu die acht Ports zweier benachbarter CPUs. Die gerade laufenden Operationen der Ports werden dafür unterbrochen. Jeder Port muß dabei vier Worte laden. Bei der CRAY Y-MP lädt Port D die 32 Worte in den Puffer, wobei die anderen Ports „ungestört" weiterarbeiten können. Wenn das erste Wort den Puffer erreicht hat, kann das Laden eines Befehlspuffers parallel zum Laden und Ausführen der Befehle durch den Instruktionsteil erfolgen.

2.4.3 Das Verbindungsnetzwerk

Das Verbindungsnetzwerk stellt die Verbindung zwischen den Ports der einzelnen CPUs und den Bänken des Hauptspeichers her. Es ist vom Schaltungsaufwand her nicht vertretbar, für jede Bank des Hauptspeichers eine Verbindung zu jedem Port jeder CPU zu haben und bei Konflikten erst bei der Bank zu entscheiden, wer den Zugriff erhält. Deshalb werden bei der CRAY X-MP die Bänke auf vier gleichgroße Sektionen verteilt. Es wird jeweils ein Weg von den vier Ports einer CPU zu jeder der vier Sektionen bereitgestellt. Die CRAY Y-MP hat zusätzlich zur Aufteilung in Sektionen eine Gliederung jeder Sektion in acht Untersektionen. In jede Untersektion gibt es von den Ports, die einer CPU zugeordnet sind, genau einen Weg.

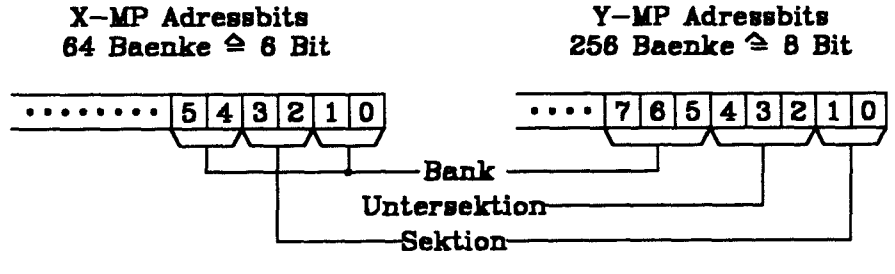


Abb. 6. Zuordnung zwischen physikalischer Adresse und Speicherorganisation: Die niederwertigen Bits der physikalischen Adresse legen die zugeordnete Bank, Sektion und Untersektion (Y-MP) fest.

Die Zuordnung zwischen Adressen und Sektionen, Untersektionen und Bänken erfolgt bei der CRAY Y-MP durch die letzten 8 Bits der physikalischen Adresse (Abb. 6). Die letzten beiden Bits legen die Sektion, die drei Bits davor die Untersektion und die drei übrigen Bits die Bänke innerhalb dieser Untersektion fest. Bei der CRAY X-MP mit 4 Prozessoren erfolgt die Zuordnung zwischen Bänken und Sektionen anders als bei der CRAY Y-MP. Die letzten 6-Bits der physikalischen Adresse bestimmen, welche Bank der Adresse zugeordnet ist. Die Sektionszuordnung ist aber durch die mittleren beiden der 6 Bits festgelegt, d.h. die Sektion wird immer erst nach vier aufeinanderfolgenden Datenworten gewechselt. Dies ist in den Arbeiten von Oed [Oed,85] und Cheung und Smith [ChSm,86] als Verbesserung der Architektur der CRAY X-MP vorgeschlagen, aber mit der um eine Hierarchiestufe erweiterten Architektur der CRAY Y-MP wieder fallengelassen worden.

2.4.4 Die Bänke

Die CRAY X-MP/416 hat eine Aufteilung des Speichers in 64 Bänke. Jede dieser Bänke ist bei einem Zugriff vier Takte lang belegt. Die CRAY Y-MP8/832 hat 256 Bänke; die Bankzykluszeit umfaßt aber fünf Takte. Taktbezogen ist der Bankzyklus der CRAY Y-MP also länger als bei der CRAY X-MP, zeitbezogen ist die Bankzykluszeit aber bei der CRAY X-MP mit 34 Nanosekunden höher als bei der CRAY Y-MP mit 30 Nanosekunden (Bankzyklus [in Takten] * Taktzeit [in nsec]). Das Datenstromprinzip in Verbindung mit Vektorverarbeitung und Pipelining ermöglicht eine hohe Verarbeitungsleistung einer CPU. Dazu ist es aber notwendig, daß die Datenströme möglichst wenig von Bankkonflikten unterbrochen werden. Die Zuordnung der Bänke zu den Adressen ist deshalb so gewählt, daß direkt aufeinanderfolgende Adressen aufeinanderfolgenden Bänken zugeordnet sind. Die Zykluszeit jeder Bank kann dann höher sein als die Zeit, in der die Anfragen des Datenstroms nacheinander erfolgen. Mit jeder Anfrage kann ein Datum aus dem Hauptspeicher übertragen werden, wenn die Zeit zwischen den Zugriffen auf die gleiche Bank der Zykluszeit dieser Bank entspricht oder länger ist.

Es ist aber nicht sichergestellt, daß Zugriffe auf den Hauptspeicher auf direkt aufeinanderfolgende Adressen geschehen. Daraus ergeben sich zwei Folgerungen:

1. Der Datenstrom kann durch Wartezeiten auf den Speicher verzögert werden.

2. Der Speicher wird universeller, wenn man ihn auch noch für größere Adreßabstände (zum Beispiel 2, 4, oder 8) auslegt.

Der Speicher verhält sich wegen dieses Aufbaus in wesentlichen Eigenschaften wie eine Pipeline. Es dauert mehr als einen Takt, bis das erste Datum eines Datenstroms verfügbar ist (Start-Up-Zeit der Pipeline). Danach kann in jedem Takt ein Datum „verarbeitet" werden.

2.4.5 Konfliktauflösung

Konflikte beim Speicherzugriff treten auf, wenn Wege zu Speicherbänken oder die Bänke selbst belegt sind. Die Konfliktauflösung ordnet die Zugriffe so, daß

- wichtige Speicherzugriffe Vorrang vor unwichtigen haben,

- die verschiedenen CPUs möglichst fair bedient werden,
- Konflikte sich möglichst nicht gegenseitig bedingen.

Grundsätzlich unterscheidet man bei CRAY-Systemen folgende Konflikte:

- Sektionskonflikte,
- Untersektionskonflikte,
- Bankkonflikte,
  - bei gleichzeitigem Bankzugriff,
  - bei belegten Bänken.

2.4.5.1 Sektionskonflikte

Sektionskonflikte treten auf, wenn mehrere Ports derselben CPU auf dieselbe Sektion zugreifen wollen. Sektionskonflikte sind also Konflikte, die nur von einer CPU abhängen. Sektionskonflikte erzeugen eine Verzögerung um einen Takt für den Port, der warten muß. Um die Sektionskonflikte aufzulösen, gibt es Prioritäten zwischen den Ports, die von dem Port selbst und von der Art des Zugriffs abhängen. Die CRAY X-MP und die CRAY Y-MP unterscheiden sich in der Art der Nutzung der Ports und deshalb auch in der Konfliktauflösung.

Bei der CRAY Y-MP hat der Port D beim Laden eines Befehlspuffers immer höchste Priorität. Bei der CRAY X-MP können hierbei keine Sektionskonflikte auftreten, weil das Laden eines Befehlspuffers über alle vier Ports zweier CPUs durchgeführt wird.

Bei Zugriffen durch Lade- und Speicherbefehle eines Programms ist entscheidend, ob der Zugriff mit gerader oder ungerader Schrittweite vorgenommen wird. Zugriffe mit ungerader Schrittweite haben die höhere Priorität. Die **Schrittweite** (Stride) ist dabei der Adreßabstand, in dem die einzelnen Elemente des Datenstroms im Speicher abgelegt sind. Bei indizierter Adressierung durch Vektoren wird dem Zugriff eine ungerade Schrittweite unterstellt. Reicht die Unterscheidung in gerade und ungerade Schrittweite nicht aus, werden die Zugriffe nach der Reihenfolge geordnet, in denen die zugehörigen Befehle aufgesetzt wurden.

2.4.5.2 Untersektionskonflikte

Untersektionskonflikte sind nur bei der CRAY Y-MP möglich, da die CRAY X-MP keine Untersektionen besitzt. Untersektionskonflikte betreffen nur jeweils den Speicherzugriff einer CPU. Eine Untersektion ist durch ihre Benutzung fünf Takte lang belegt. Es kann nicht vorkommen, daß der gleichzeitige Zugriff zweier Ports einer CPU auf eine Untersektion zu einem Untersektionskonflikt führt, weil dieser Konflikt bereits als Sektionskonflikt eingeordnet wird. Will ein Port einer CPU auf eine Untersektion zugreifen, die bereits belegt ist, muß er ein bis vier Takte warten. Warten mehrere Ports auf dieselbe Untersektion, so wird beim Freiwerden der Untersektion zuerst ein Sektionskonflikt erkannt und dieser aufgelöst.

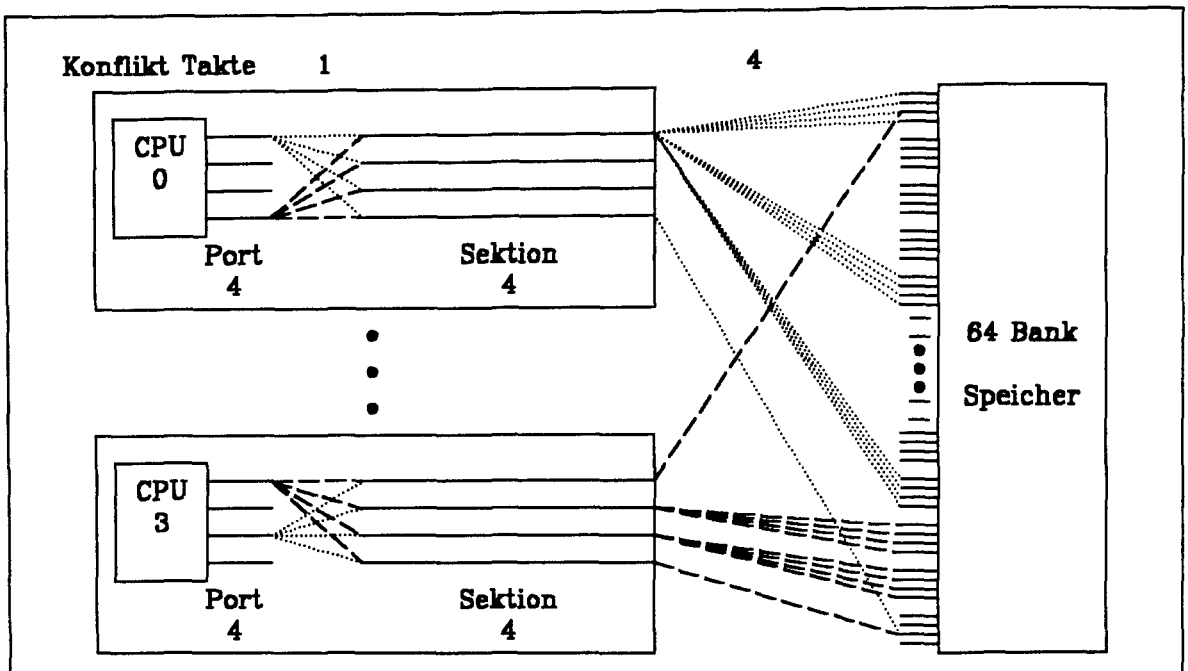


Abb. 7. Konfliktauflösung bei der CRAY X-MP/416

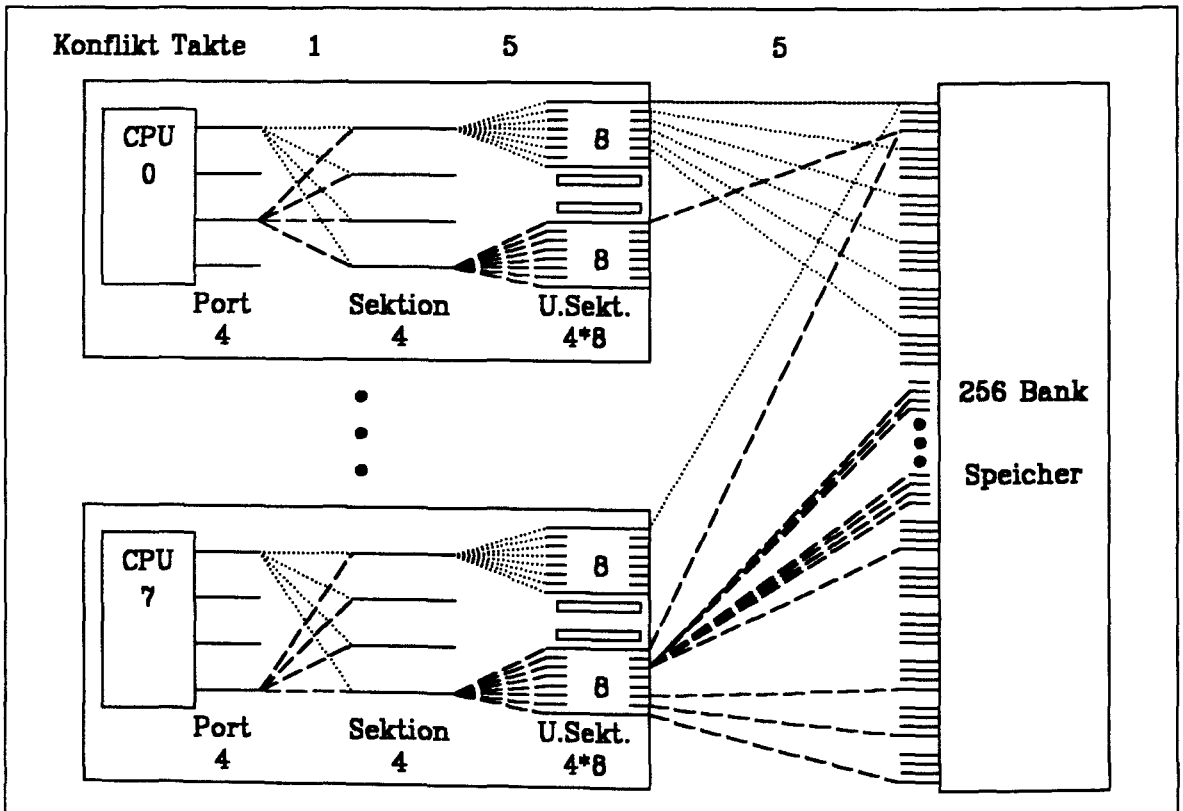


Abb. 8. Konfliktauflösung bei der CRAY Y-MP8/832

2.4.5.3 Bankkonflikte

Bankkonflikte werden entweder durch einen gleichzeitigen Zugriffswunsch auf eine Speicherbank oder durch den versuchten Zugriff auf eine belegte Bank hervorgerufen.

Gleichzeitiger Bankzugriff: Ein gleichzeitiger Bankzugriff kann nur zwischen verschiedenen CPUs auftreten. Gleichzeitige Zugriffe auf die gleiche Bank durch eine CPU werden durch die Sektionskonflikte verhindert. Bei der CRAY X-MP werden Bankkonflikte durch eine CPU-Priorität aufgelöst, die alle vier Takte wechselt. Bei der CRAY Y-MP erfolgt die Konfliktauflösung anhand einer festen Prioritätstabelle, die die Sektion, die Untersektion der gewünschten Bank und die CPU-Nummer berücksichtigt.

Belegte Bänke: Eine Bank ist durch einen Zugriff bei der CRAY X-MP vier Takte und bei der CRAY Y-MP fünf Takte lang belegt. Wartet ein Zugriff auf eine belegte Bank, so kann dies bei der CRAY X-MP drei Takte und bei der CRAY Y-MP vier Takte dauern. Bestehen danach mehrere Zugriffswünsche auf dieselbe Bank, so ist dies ein Konflikt durch gleichzeitigen Bankzugriff, der als solcher aufgelöst wird.

Wenn ein Bankkonflikt auftritt, werden bei der CRAY X-MP im Gegensatz zur CRAY Y-MP alle aktiven Speicherports der wartenden CPUs angehalten, bis der Konflikt aufgelöst ist. Durch einen Konflikt können also Ports gestört werden, die auf nicht belegte Sektionen und Bänke zugreifen wollen ([FaSc,89],[Dete,89]).

Abb. 7 und Abb. 8 beschreiben die beiden CRAY-Rechner aus Sicht der Konfliktauflösung und illustrieren das in den vorhergehenden Abschnitten Gesagte. Dazu ist eine Aufteilung in lokale und globale Konfliktauflösung vorgenommen worden. Lokal heißt dabei, daß der Konflikt nur eine CPU betrifft; globale Konflikte müssen zwischen den CPUs aufgelöst werden. Gestrichelte und gepunktete Linien in den Abbildungen beschreiben dabei alternative Datenpfade, die aber nicht immer gleichzeitig benutzt werden können. Die Unterscheidung in gestrichelte und gepunktete Pfade dient dabei nur der Übersichtlichkeit. Die Anordnung der einzelnen Bänke im Speicher entspricht einer fortlaufenden Speicheradressierung.

Mit der Beschreibung der Konfliktauflösung ist die Vorstellung der Speichereigenschaften und damit die eigentliche Beschreibung des Aufbaus der Rechner CRAY X-MP und CRAY Y-MP abgeschlossen. Da bei einer vergleichenden Betrachtung besonders die Unterschiede der beiden Rechner interessieren, sollen diese im folgenden Abschnitt noch einmal zusammengefaßt werden.

2.5 Zusammenfassung der Unterschiede im Systemaufbau

Die CRAY Y-MP8/832 und die CRAY X-MP/416 haben sehr ähnliche Prozessoren, von denen die CRAY X-MP vier und die CRAY Y-MP acht besitzt. Der für den Anwender wichtigste Unterschied der CPUs ist die verkürzte Taktzykluszeit bei der CRAY Y-MP. Ein weiterer Unterschied der CPUs ist die auf 32 Bit erweiterte Adreßrechnung, die auch zu einer Verlängerung einiger Befehlscodes auf drei Parcels gegenüber maximal zwei Parcels bei der CRAY X-MP führt.

Die deutlichsten Unterschiede finden sich bei der Speicherarchitektur. Der von 8.5 auf 6.0 Nanosekunden verbesserten Zykluszeit der CPU der CRAY Y-MP steht hier ein auf die Anzahl der Takte bezogen verschlechterter Bankzyklus (fünf Takte gegen vier bei der X-MP) gegenüber. Dafür ist die Anzahl der Bänke bei der CRAY Y-MP auf 256 gegenüber 64 bei der CRAY X-MP erhöht, was einer Verdoppelung der Bankanzahl je CPU (32 gegenüber 16) entspricht.

Das Speicherzugriffsnetzwerk wurde vom zweistufigen Netzwerk mit Sektionen und Bänken bei der CRAY X-MP zu einem dreistufigen Netzwerk mit Sektionen, Untersektionen und Bänken erweitert. Durch die Untersektionsstufe wird die Behandlung von Konflikten stärker auf die einzelne CPU bezogen. Mit den Untersektionen hängt eine zusätzliche Stufe der Konfliktauflösung zusammen. Die Veränderung des Speichernetzwerkes bringt außerdem eine andere Zuordnung der Hardware-Adressen zu den Sektionen und ein verändertes Prioritätenschema bei Bankkonflikten mit sich.

Die vier Ports der CRAY Y-MP sind gegenüber den vier Ports der CRAY X-MP dadurch verbessert worden, daß ein Konflikt, der einen Port einer CPU betrifft, nicht mehr die anderen Ports derselben CPU behindert. Weiter fällt die Änderung des Zugriffs beim Laden der Befehlspuffer auf. Beeinflußte dieses Laden bei der X-MP noch alle Ports zweier CPUs, so wird dies jetzt auf einen Port der betroffenen CPU verlagert. Dies bringt eine Verlangsamung des Ladevorgangs bei der CRAY Y-MP, aber auch eine deutliche Spezialisierung der Ports und eine weitere Stärkung der Autonomie der einzelnen CPUs hinsichtlich der Durchführung ihrer jeweiligen Aufgaben mit sich.

Die Auswirkungen, die die Entwicklung der CRAY Y-MP aus der CRAY X-MP mit sich gebracht hat, sollen durch die folgenden Kapitel genauer untersucht werden.

3.0 Die Messung der Systemeigenschaften

3.1 Ziele

Die Beschreibung der Hardware-Funktionen im vorhergehenden Kapitel ist die Grundlage für die Betrachtung von Programmen auf CRAY-Systemen. In diesem Kapitel wird das Zeitverhalten von elementaren Programmen, sogenannten Programmkernen, auf dieser Hardware betrachtet. Dabei wird von der Untersuchung maschinennaher Assembler-Programmkerne zur Untersuchung von benutzernäheren FORTRAN-Programmkernen übergegangen. Dieser Übergang zu einer höheren Programmiersprache bezieht die Eigenschaften des Compilers mit in die Zeitmessungen ein. Die in diesem Kapitel betrachteten einfachen FORTRAN-Programme bilden die Grundlage der Bewertung komplexerer Programme und der darin enthaltenen Algorithmen im nachfolgenden Kapitel 4.

Für die Beurteilung und den Vergleich der CRAY-Systeme ist es sinnvoll, möglichst die gleichen Methoden und die gleichen Leistungsmaßstäbe auf beiden Systemen zu verwenden. Die gemeinsame Methode ist hierbei, ausgehend von Messungen auf nur einer CPU, zu Messungen auf mehreren CPUs überzugehen und dort den Einfluß des Speichers als gemeinsamer Ressource zu betrachten. Die benutzten Leistungsmaßstäbe werden in den folgenden Abschnitten genauer vorgestellt.

3.2 Die Programmierumgebung

Für das Verständnis der durchgeführten Messungen ist die Grundkenntnis der Arbeitsumgebung auf CRAY-Systemen und speziell der FORTRAN-Compiler erforderlich. Als Programmiersprache für die Messungen von Kernen und Algorithmen bietet sich FORTRAN aus folgenden Gründen an:

- die weite Verbreitung von FORTRAN als Programmiersprache für technisch-wissenschaftliche Probleme,
- die gute Ausnutzung der Rechnerarchitektur der CRAY-Systeme durch die FORTRAN-Compiler, und damit
- der in der Praxis des Höchstleistungsrechnens auf CRAY-Systemen nahezu ausschließliche Einsatz von FORTRAN.

Für die beiden hier betrachteten CRAY-Rechner stehen zwei Betriebssysteme zur Verfügung: COS (CRAY Operating System, Version 1.17) und UNICOS (die UNIX-Version für CRAY-Systeme, Version 5.0). Es existiert auch ein C-Compiler für die CRAY-Systeme. Die weiter verbreitete Programmiersprache ist aber nach wie vor FORTRAN. Es existieren auf beiden Betriebssystemen zwei verschiedene Compiler: der ältere, künftig nicht mehr weiter entwickelte CFT-Compiler (Version 1.15 auf der CRAY X-MP, Version 1.16 auf der CRAY Y-MP) und der neuere CFT77-Compiler (Version 3.1 auf beiden Rechnern). Für die Messungen wird nur der CFT77-Compiler verwendet. Der in den maschinennahen Messungen eingesetzte Assembler der CRAY-Systeme ist CAL (CRAY Assembly Language, Version 2.3.2).

Bei der Untersuchung des Speicherverhaltens interessieren besonders Programme, die den Speicher intensiv nutzen. Deshalb werden solche Programme betrachtet, die die Eigenschaften der Vektorverarbeitung der CRAY-Systeme beanspruchen, da nur damit hohe Datentransferraten zwischen CPU und Hauptspeicher unter Programmkontrolle zu erreichen sind. Dazu ist es notwendig, Programmkonstrukte in FORTRAN zu kennen, die diese Vektor-Hardware ausnutzen. Das typische Programmkonstrukt, das vom CFT77-Compiler in Vektorbefehle umgesetzt wird, ist eine Programmschleife mit Zugriffen auf FORTRAN-Felder (Abb. 9). Diese Umsetzung wird **Vektorisierung** genannt. Da hier besonders die Eigenschaft interessiert, einen Datenstrom mit Vektorbefehlen zu verarbeiten, verwendet man die Vektorlänge als Maß für den Aufwand bei der Berechnung. Die Vektorlänge ist dabei identisch mit der Anzahl der Wiederholungen des Schleifenkerns der DO-Schleife. An dieser Stelle soll auf eine genauere Beschreibung der Umstände, unter denen FORTRAN-Programme auf CRAY-Systemen vektorisieren, verzichtet werden. Dies findet man aber ausführlich in [CRAY,88a] und [Dete,87] beschrieben.

```
      DO 10 I=1,N
         A(I) = B(I) + C(I) * D(I)
10 CONTINUE
```

Abb. 9. Beispiel für einen Programmkern: Das Programm vektorisiert, d. h. die Operationen der Zuweisung können mit Hilfe von Vektorbefehlen in Vektorregistern und Vektorfunktionseinheiten bearbeitet werden.

Vektorisierung bedeutet für den Compiler, die FORTRAN-DO-Schleife in Vektorbefehle umzusetzen. Dies ist relativ einfach, wenn die Vektorlänge die Länge der Vektorregister nicht überschreitet, da dann die Schleife in einen Vektorbefehl oder eine Sequenz von Vektorbefehlen umgesetzt werden kann. Bei längeren Schleifen muß die Befehlssequenz wiederholt werden, d.h. die FORTRAN-Schleife wird in eine Schleife mit Vektorbefehlen umgesetzt. Diese Aufteilung in Vektorbefehle heißt **Vektorzerlegung** (Strip-Mining). Dabei wird der Vektor in **Teilvektoren** (Chunks) der Länge 64 (mit Rest) unterteilt.

Über die durch die Hardware bedingte Aufteilung in Teilvektoren hinaus nimmt der Compiler aus Optimierungsgründen eine Aufteilung in übergeordnete **Vektorabschnitte** der Länge 128 vor. Diese Aufteilung ist dann möglich, wenn für die Vektoroperationen nicht mehr als vier Vektorregister benötigt werden. Dann kann die gleiche Berechnung für den zweiten der beiden Teilvektoren der Länge 64 mit vier anderen Vektorregistern durchgeführt werden. Dies macht ein Warten auf die Ressource Register überflüssig und beschleunigt damit die Berechnungen. Ein weiteres wichtiges Ziel der Optimierung durch den Compiler ist die Ausnutzung der im Abschnitt 2.3.4 beschriebenen Unabhängigkeit der Hardware-Ressourcen durch Variation der Befehlsreihenfolge (Code Scheduling). Dadurch wird eine möglichst günstige zeitliche Überlappung unabhängiger Operationen angestrebt.

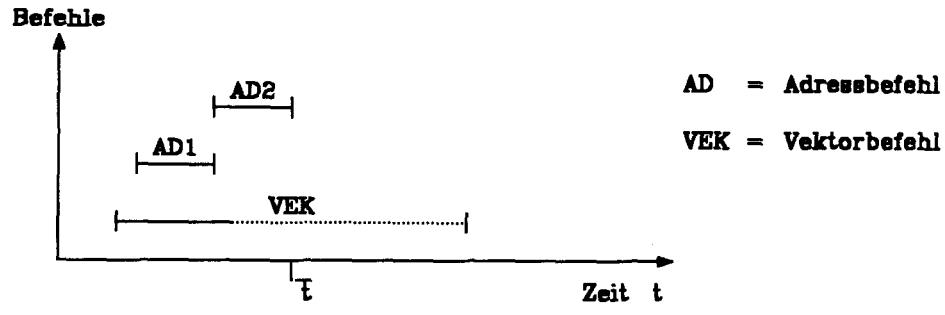


Abb. 10. Überlappung von Operationen

Typisch für FORTRAN-Vektorschleifen ist z.B. die Überlappung von Adreßarithmetik und Gleitkommaarithmetik. Das zeitliche Verhalten überlappter Operationen ist schematisch in Abb. 10 beschrieben. Der Grad der Überlappung hängt von der Vektorlänge ab. Diese wird durch den gepunkteten Anteil des Vektorbefehls angedeutet. Der Befehl AD1 wird immer vollständig vom Vektorbefehl überlappt. Der Grad der Überlappung von AD2 hängt von der Vektorlänge von VEK ab und ist erst zum Zeitpunkt $\bar{t}$ vollständig. Ist die Vektorverarbeitung vor $\bar{t}$ abgeschlossen, bestimmt AD2 die Ausführungszeit, wird sie danach abgeschlossen, spielt die Adreßrechnung keine Rolle mehr für die Ausführungszeit der Vektorschleife.

3.3 Leistungsmaßstäbe

Zur Beurteilung der Messungen der Ausführungszeiten von Programmkernen auf CRAY-Systemen benötigt man Leistungsmaßstäbe, die für die Beurteilung sowohl der Hardware der betrachteten Systeme als auch von Algorithmen tragfähig sind. Im folgenden werden verschiedene Ansätze vorgestellt und es wird ihre Aussagefähigkeit für diesen Zusammenhang beurteilt:

- Die Anzahl der Instruktionen pro Sekunde (MIPS = Million Instructions per Second): Da sich bei Vektorrechnern mit einer Instruktion mehrere Operationen anstoßen lassen (SIMD-Rechner [HwBr,84]), ist dieser Maßstab für die Vektorverarbeitung auf CRAY-Systemen nicht aussagekräftig.
- Die Anzahl der Gleitkommaoperationen pro Sekunde (MFLOPS = Million Floating-Point Operations per Second): Da die Speichereigenschaften betrachtet werden, ist dieser Maßstab ungeeignet, weil er die mit den Gleitkommaoperationen verbundenen Speicherzugriffe nicht umfaßt.
- Die verbrauchte CPU-Zeit eines Programms (in Sekunden oder Takten): Die CPU Zeit mißt eine Mischung des Zeitverhaltens von Speicher und CPU und bietet deshalb auch ein Maß für das Speicherverhalten eines Programms.
- Die Anzahl der Speicherzugriffe pro Sekunde (MOPS = Memory Operations per Second): Dieser Maßstab ist gut geeignet, die Leistung des Speichers zu beurteilen. Er beschreibt aber nur einen Aspekt des Speicherverhaltens, nämlich das durchschnittliche Zeitverhalten.

- Die Kenngrößen r_∞, $n_{1/2}$ nach Hockney: Diese Kenngrößen sollen in einer veränderten Form die Grundlage für die weitere Leistungsbewertung sein. Deshalb werden sie im nächsten Abschnitt ausführlich vorgestellt.

3.4 Leistungsbewertung nach Hockney

Die Leistungsbewertung nach Hockney ist in mehreren Büchern und Artikeln vorgestellt und auf verschiedene Rechnersysteme und Programme angewendet worden ([Hock,83], [Buch,83], [Hock,85], [HoJe,88], [HoCu,89]).

3.4.1 Definition

Die Definition der Leistungskenngrößen nach Hockney geht von einem linearen Zusammenhang zwischen der Ausführungszeit von Programmen und der betrachteten Vektorlänge aus. Dieser Zusammenhang läßt sich durch die folgende Gleichung beschreiben.

$$(1) \qquad t(n) = \tau \bullet n + t_0$$

Die einzelnen Variablen und Parameter in dieser Gleichung haben folgende Bedeutung:

t	Ausführungszeit,
n	Vektorlänge,
τ	Einzelergebniszeit (Time per Result),
t_0	Aufsetzzeit (Start-Up Time).

Ausgehend von dem linearen Zeitverhalten definiert Hockney zwei Kenngrößen r_∞ und $n_{1/2}$. Dabei soll r_∞ die Leistung sein, die das gemessene Programm bei wachsender Vektorlänge auf dem betrachteten Rechner maximal erreichen kann. r_∞ heißt **Grenzleistung** (asymptotic performance). $n_{1/2}$ soll die Vektorlänge sein, bei der die halbe Grenzleistung für das betrachtete Problem erreicht wird. $n_{1/2}$ wird als **Halbleistungslänge** (half performance length) bezeichnet.

Der allgemeine Leistungsbegriff stellt sich, unabhängig vom betrachteten Problem, als

$$Leistung = \frac{Arbeit}{Zeit}$$

oder hier spezieller

$$Leistung = \frac{Operationen}{Zeit}$$

dar. Geht man zunächst davon aus, daß die Vektorlänge mit der Anzahl der Operationen übereinstimmt, so kann man mit diesem Ansatz die Leistung r formal als

$$(2) \qquad r(n) = \frac{n}{t(n)} = \frac{n}{\tau \bullet n + t_0} = \frac{1}{\tau + \frac{t_0}{n}}$$

einführen und daraus als Grenzwert die Grenzleistung

$$(3) \qquad r_\infty := \lim_{n \to \infty} r(n) = \lim_{n \to \infty} \frac{1}{\tau + \frac{t_0}{n}} = \frac{1}{\tau}$$

bestimmen. Sucht man jetzt die Vektorlänge $n_{1/2}$, für die gilt

$$r(n_{1/2}) = \frac{1}{2} r_\infty ,$$

ergibt sich mit (2) und (3) aus

$$(4) \qquad \frac{n_{1/2}}{\tau \bullet n_{1/2} + t_0} = \frac{1}{2 \bullet \tau}$$

der Wert für $n_{1/2}$:

$$(5) \qquad n_{1/2} = \frac{t_0}{\tau} = t_0 \bullet r_\infty .$$

Bringt man nun die Bestimmungsgleichungen (3) für r_∞ und (5) für $n_{1/2}$ in die Geradengleichung (1) statt τ und t_0 ein, erhält man eine andere Form der Geradengleichung :

$$(6) \qquad t(n) = \frac{1}{r_\infty} (n + n_{1/2})$$

Formel (6) beschreibt also nach Hockney das Zeitverhalten von Rechnern und Programmen mit Hilfe der Kenngrößen r_∞ und $n_{1/2}$.

3.4.2 Interpretation anhand von Beispielen

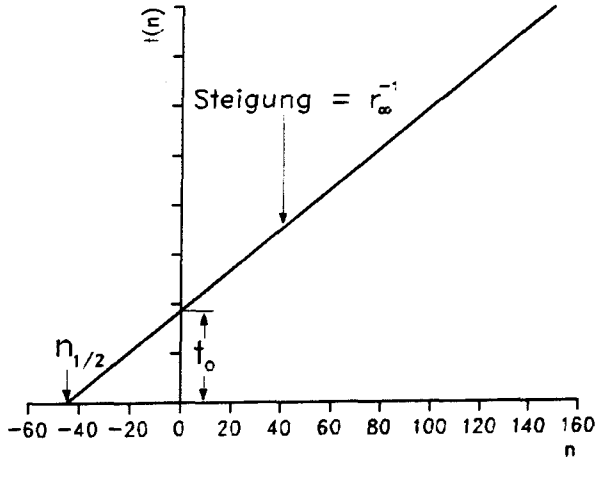


Abb. 11. Leistungskenngrößen nach Hockney

Abb. 11 verdeutlicht Formel (6). $n_{1/2}$ ergibt sich aus dem Schnittpunkt der Gerade mit der Abszisse. r_∞ ist der Kehrwert der Steigung der Gerade. Der Schnittpunkt der Gerade mit der Ordinate bestimmt die Aufsetzzeit t_0, wie sie in Gleichung (1) eingeführt worden ist.

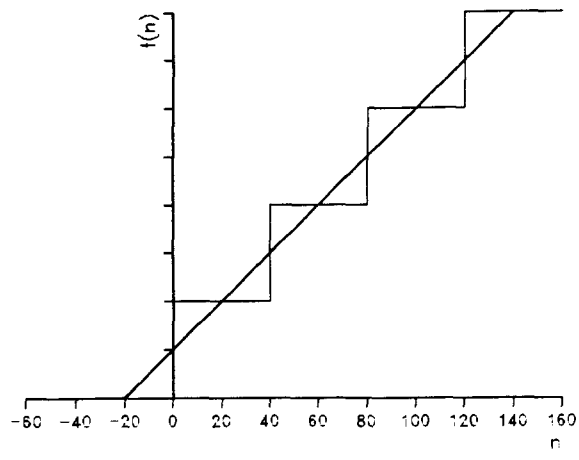


Abb. 12. Kenngrößen bei Parallelrechnern

Bei Parallelrechnern wird die durch die Hardware bedingte Meßkurve von der Verteilung paralleler Berechnungen auf die verschiedenen Prozessoren bestimmt. Die Berechnungszeit ist konstant, solange bei wachsender Problemgröße n für jede weitere Berechnung ein freier Prozessor hinzugenommen werden kann. Sind alle Prozessoren belegt, müssen einige Prozessoren nacheinander zwei Berechnungen durchführen, bis alle Prozessoren mit zwei Berechnungen beschäftigt werden; danach muß der erste Prozessor drei Berechnungen durchführen usw. Aus diesem Vorgehen ergibt sich ein für Parallelrechner typisches Stufenbild als Leistungsverlauf (Abb. 12). Der Übergang von einer Stufe zur nächsten erfolgt immer dann, wenn n ein Vielfaches der Anzahl der Prozessoren überschreitet. (Die Beispielkurve bezieht sich also auf einen Rechner mit 40 Prozessoren.)

Eine wichtige Voraussetzung bei der Einführung der Kenngrößen r_∞ und $n_{1/2}$ ist das lineare Zeitverhalten. Dies ist offensichtlich bei Parallelrechnern so nicht mehr gegeben. Ist n jedoch wesentlich größer als die Zahl der Prozessoren, kann man das globale Zeitverhalten dennoch als linear ansehen, wenn man es durch eine Regressionsgerade annähert. Die Ausgleichsgerade ergibt in Abb. 12 für $n_{1/2}$ gerade einen Wert von 20, die halbe Prozessoranzahl. Die Grenzleistung wird von der Höhe und der Breite der Stufen bestimmt.

Der Wert der Halbleistungslänge $n_{1/2}$ kann für die beiden unterschiedlichen Kurven Abb. 11 und Abb. 12 verschiedenen Ursachen zugeordnet werden:

- dem einmaligen Aufwand beim Anfang der Berechnung (Start-Up);
- der Parallelität in der zugrundeliegenden Hardware.

Für den Anwender ist nur die Auswirkung der Halbleistungslänge von Bedeutung. Deshalb betrachtet Hockney $n_{1/2}$ als Maß für Parallelität und spricht auch von scheinbarer Parallelität, die er nicht von der Parallelität des Rechners unterscheidet [HoJe,88].

Wie schon beschrieben existieren bei CRAY-Systemen beide Elemente (Aufsetzzeit und „innere" Parallelität). Abb. 13 zeigt das hieraus resultierende Zeitverhalten. Wiederum ist es sinnvoll, die Regressionsgerade zur Beschreibung des globalen Zeitverlaufs zu ver-

wenden. Die Diskussion der Bedeutung von $n_{1/2}$ und r_∞ erfolgt ausführlich anhand konkreter Meßergebnisse in diesem Kapitel.

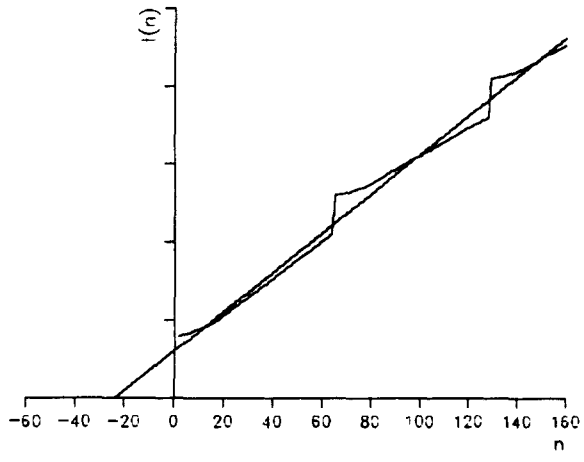


Abb. 13. Leistungskenngrößen bei CRAY-Rechnern

3.4.3 Erweiterungen der Leistungsparametrisierung

Die bisherige Betrachtung geht von einem pauschalen Leistungsbegriff aus und betrachtet z.B. nicht, welche Operationen gemessen werden. Hockney ist bei der Einführung der Kenngrößen von Gleitkommaoperationen ausgegangen. Genauso können aber auch Speicheroperationen bewertet werden.

In dem bisherigen Ansatz wird davon ausgegangen, daß pro Schleifendurchlauf genau eine Operation durchgeführt wird. Allgemein mögen jetzt pro Schleifendurchlauf k Operationen auftreten, also bei Vektorlänge n insgesamt $k \bullet n$ Operationen.

Für die Berechnung der Leistung $r(n)$ bedeutet dies:

$$(7) \qquad r(n) = \frac{k \bullet n}{t(n)}$$

Daraus ergibt sich durch Einsetzen wie oben

$$(8) \qquad r_\infty = \frac{k}{\tau} \qquad \text{(vgl. (3))}$$

Die asymptotische Leistung skaliert sich also mit dem Faktor k hoch.

Für $n_{1/2}$ ergibt sich der Einfluß des Faktors k aus den Gleichungen (7) und (8) analog zu (4):

$$(9) \qquad r(n_{1/2}) \doteq \frac{1}{2} r_\infty \Leftrightarrow \frac{k \bullet n_{1/2}}{\tau \bullet n_{1/2} + t_0} = \frac{k}{2 \bullet \tau}$$

d.h. in (9) kürzt sich k heraus. $n_{1/2}$ ist also unabhängig von der Betrachtung der Anzahl der Operationen innerhalb der Schleife.

Je nach zugrundegelegtem Leistungsbegriff sollen folgende Bezeichnungen gewählt werden:

r_∞ , $n_{1/2}$	Operationen (allgemein) pro Zeiteinheit,
r_∞^{mem} , $n_{1/2}^{mem}$	Speicheroperationen pro Zeiteinheit,
r_∞^{flop} , $n_{1/2}^{flop}$	Gleitkommaoperationen pro Zeiteinheit,
r_∞^{loop} , $n_{1/2}^{loop}$	Schleifendurchläufe pro Zeiteinheit.

Grundlage für die Beschreibung der Kenngrößen r_∞ und $n_{1/2}$ ist Formel (1)

$$(1) \qquad t(n) = \tau \bullet n + t_0$$

Hierin ist frei bestimmbar, was als Zeiteinheit benutzt wird. Üblicherweise werden Nanosekunden als Bezugseinheit gewählt. Für manche der im folgenden angestellten Untersuchungen (insbesondere für die Rechnervergleiche) kann es sinnvoll sein, Maschinentakte als Zeiteinheit zu wählen, d.h. die Leistung auf die Zykluszeit des betrachteten Rechners zu beziehen. Analog den Überlegungen zur Abhängigkeit der Leistung von der Anzahl der Operationen ergibt sich eine Skalierung der Grenzleistung r_∞ durch einen Faktor. Die Halbleistungslänge $n_{1/2}$ bleibt von der Wahl der Zeiteinheit unbeeinflußt.

3.4.4 Verallgemeinerung der Leistungsbewertung

Im Zusammenhang mit der Bewertung der Eigenschaften des Hauptspeichers und deren Auswirkungen auf die Ausführungszeit soll hier noch eine erweiterte Form der Leistungsbewertung vorgestellt werden, die auf den beiden Kenngrößen $n_{1/2}$ und r_∞ aufbaut ([HoJe,88], [HoCu,89]). Diese Bewertung nimmt die Einflüsse eines Kommunikationsengpasses zwischen den Vektorfunktionseinheiten zur Berechnung von Arithmetik und dem Hauptspeicher in die Berechnung der Kenngrößen $n_{1/2}$ und r_∞ auf. Dazu wird das Verhältnis von arithmetischen Operationen zu Speicheroperationen eines betrachteten Programms als Parameter f eingeführt und als **Berechnungsintensität** (Computational Intensity) bezeichnet. Genauso wie man bei einer vorgegebenen Vektorlänge n das $n_{1/2}$ als Vergleichswert für die zu erwartende Leistung $r(n)$ benutzt, wird für eine gegebene Berechnungsintensität f ein $f_{1/2}$ eingeführt, mit dem man die zu erwartenden Werte für $n_{1/2}$ und r_∞ bestimmen kann. Analog zu $n_{1/2}$ heißt $f_{1/2}$ die **Halbleistungsintensität** (Half Performance Intensity) und ist der Wert der Berechnungsintensität f, bei der die halbe Grenzleistung für eine Kombination aus Speicheroperationen und arithmetischen Operationen erreicht wird. Hockney geht bei der Definition von $f_{1/2}$ davon aus, daß bei Vektoroperationen mit n Operanden das Zeitverhalten jeder Speicheroperation und jeder arithmetischen Operation linear ist , also

$$t_1 = \frac{1}{r_\infty^{mem}} (n + n_{1/2}^{mem})$$

$$t_2 = \frac{1}{r_\infty^{flop}} (n + n_{1/2}^{flop})$$

Rechnet man den Anteil der Speicheroperationen in die Ausführungszeit einer arithmetischen Operation mit ein, erhält man eine durchschnittliche Ausführungszeit $\bar{t}$, für die gilt:

$$\bar{t} = \frac{1}{\bar{r}_\infty}(n + \bar{n}_{1/2})$$

Zur Bestimmung der Formel für $f_{1/2}$ muß noch der Zusammenhang zwischen Speicherzugriffen und Arithmetik eingebracht werden. Dazu gibt Hockney zwei Fälle an, für die die Ergebnisse im folgenden zusammengestellt sind:

Speicherzugriffe und Arithmetik seriell: Es tritt keine Überlappung zwischen Speicherzugriffen und Arithmetik auf. Daraus ergibt sich die durchschnittliche Ausführungszeit als

$$\bar{t} = \frac{t_1}{f} + t_2 \quad \text{und damit} \quad f_{1/2} = \frac{r_\infty^{flop}}{r_\infty^{mem}} .$$

Jetzt ist es möglich, $\bar{r}_\infty$ und $\bar{n}_{1/2}$ als Funktion der Berechnungsintensität f bzw. $f/f_{1/2}$ anzugeben

$$\bar{r}_\infty(f) = \frac{r_\infty^{flop}}{1 + f_{1/2}/f}$$

$$\bar{n}_{1/2}(f) = \frac{(n_{1/2}^{mem} + f/f_{1/2} \bullet n_{1/2}^{flop})}{(1 + f/f_{1/2})}$$

Speicherzugriffe und Arithmetik überlappt: Die Übertragung der Daten vom und zum Speicher findet gleichzeitig zur Berechnung statt. Dabei ist für die Ausführungszeit nur der dominierende Anteil von Bedeutung, also entweder die Zeit für die Speicherzugriffe oder die Zeit für die arithmetischen Operationen. Dies drückt sich in der Verwendung der sogenannten Kniefunktion (knee) aus:

$$knee(x) = \begin{cases} x & bei\ x \leq 1 \\ 1 & bei\ x \geq 1 \end{cases}$$

Damit ergibt sich r_∞ als

$$r_\infty(f) = r_\infty^{flop} \bullet knee(\frac{1}{2} \bullet f/f_{1/2}) \quad ,$$

wobei

$$f_{1/2} = \frac{1}{2} \bullet \frac{r_\infty^{flop}}{r_\infty^{mem}} \quad .$$

Als Beispiel für den seriellen als auch für den Überlappungsfall dient in [HoCu,89] ein Rechner-FPS 5000, bei dem der Engpaß im Datentransfer zwischen lokalem und Hauptspeicher auftritt. Die Speicherzugriffe werden dabei über eigene Befehle gesteuert. Die Übertragung der dort vorgenommenen Messungen auf CRAY-Systeme scheint mit einigen Problemen behaftet zu sein; Ursachen sind zu sehen in:

- der Unterscheidung in zwei Ladeports und einen Speicherport,

- dem Auftreten von sowohl sequentiellem Verhalten (zuwenig Ports vorhanden) als auch überlapptem Verhalten (Engpaß durch Zugriffskonflikte),
- einer fehlenden Berücksichtigung der Verteilung der Speicheroperationen auf das betrachtete Programm,
- dem Einfluß des Compilers, durch den nicht mehr für jeden Algorithmus klar ist, welche Operationen zu Speicherzugriffen führen und welche durch Register abgewickelt werden können.

Auf eine Untersuchung von $f_{1/2}$ auf CRAY-Systemen wird deshalb hier verzichtet.

3.5 Einzelprozessormessungen

Die Untersuchungen der Eigenschaften eines einzelnen Prozessors bilden die Grundlage für die Mehrprozessormessungen. Zuerst wird dabei untersucht, wie sich die Prozessorarchitektur auf die Ausführungszeit von Kernen auswirkt, die nur auf die Ressourcen des Prozessors zugreifen, also Vektorarithmetik ohne Speicherzugriffe durchführen. Danach wird der Speicher aus Sicht eines Prozessors möglichst isoliert betrachtet. Darauf baut eine Betrachtung des Zusammenspiels von Speicher und Arithmetik auf.

Die vorgestellten Ergebnisse wurden für Vektorlängen von ein bis 1024 Elementen gemessen, wobei für jede Vektorlänge jeweils der Minimalwert über 100 Wiederholungsmessungen genommen wurde. Die Messungen wurden hier auf einen Prozessor beschränkt; dazu wurde ihm als einzigem Zugriff zum Hauptspeicher gewährt (dedizierte Messung). Die Messungen wurden unter dem COS-Betriebssystem durchgeführt.

3.5.1 Unterschiede im Prozessorverhalten

Der Aufbau der Prozessoren der CRAY-Systeme unterscheidet sich funktional nur in wenigen Komponenten. Diese betreffen nicht die arithmetischen Vektoroperationen. Das muß aber nicht bedeuten, daß gleiche Befehle auf beiden Rechnern die gleiche Anzahl von Takten benötigen.

Eine Programmierung isolierter Arithmetikoperationen aus FORTRAN heraus ist nicht gezielt möglich. Deshalb wurde hier der Assembler-Code, den der FORTRAN-Compiler erzeugt, modifiziert. Das gleiche Vorgehen wurde bei einigen Kernen zur Speichermessung gewählt. Für eine Vektoradditionsoperation wurde z. B. als Kern der Code einer FORTRAN-Zuweisung mit Addition genommen, dabei wurden alle Assembler-Befehle entfernt, die für die Adreßrechnung und den Datentransport vom und zum Speicher sorgen.

Tab. 2 faßt die Meßergebnisse zusammen, die im folgenden genauer interpretiert werden sollen. Die Angaben für r_∞ sind dabei, wie auch in den folgenden Tabellen, auf tausend Takte der jeweiligen CPU bezogen. Der maximal erreichbare Wert ist also 1000.

	r_∞^{loop}		$n_{1/2}$	
	X-MP	Y-MP	X-MP	Y-MP
Addition	940.36	940.56	35.56	37.31
Multiplikation	940.10	940.32	36.80	38.51
Division	290.11	286.26	25.15	25.76

Tab. 2. r_∞ und $n_{1/2}$ bei Prozessoroperationen: Es sind Vektoroperationen ohne Speicherzugriff gemessen worden. Die Grenzleistung ist taktbezogen und wird als r_∞^{loop} • 1000 angegeben.

3.5.1.1 Vektoraddition und -multiplikation

Auf den ersten Blick ist die Vektoraddition ein typisches Beispiel für die Aufteilung der Vektoren in Teilvektoren der Länge 64 auf CRAY-Rechnern und für das daraus resultierende Zeitverhalten (Abb. 14). Eine genauere Betrachtung führt zu einem differenzierteren Bild.

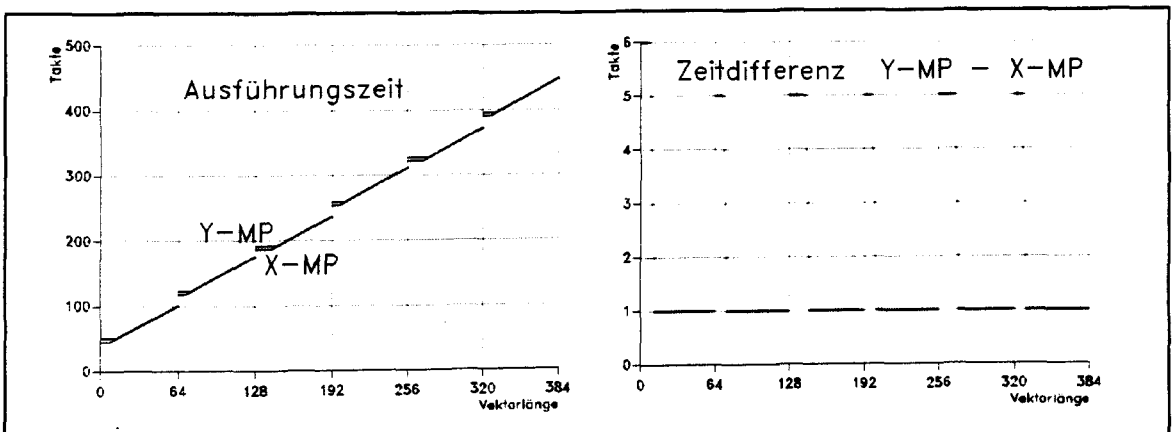


Abb. 14. Vektoraddition ohne Speicherzugriffe

Eine auffällige Eigenschaft der Meßkurve ist die Abflachung jeweils am Anfang eines Teilvektors aus 64 Elementen. Diese Abflachung wird im folgenden als **Plateau** bezeichnet. Das Besondere am Anfang eines Teilvektors ist, daß dort der erste Teilvektor der Vektorzerlegung jeweils relativ kurz ist. Dies macht es unmöglich, alle nach der Vektoroperation aufgesetzten anderen Operationen parallel zu der Vektoroperation auszuführen. Erreicht man durch Vergrößerung der Vektorlänge gerade diese parallele Ausführung, ist das Ende des Plateaus erreicht. Abb. 15 nimmt noch einmal das Beispiel in Abb. 10 auf und zeigt die beiden Teilkurven, die das Plateau erzeugen.

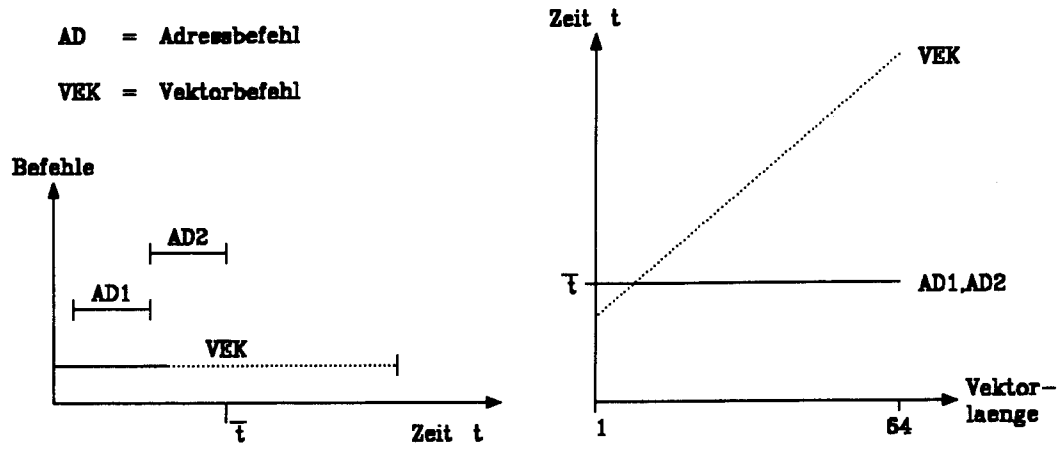


Abb. 15. Überlappung von Operationen in Verbindung mit der Variation der Vektorlänge: Der resultierende Zeitverlauf bei Variation der Vektorlänge ist das Maximum der Teilkurven für den Vektorbefehl und die beiden Adreßbefehle.

Beim Vergleich der beiden Rechner zeigt sich, daß die Plateaus der Y-MP etwas länger als die der X-MP sind, was in der Taktdifferenz gut zu erkennen ist. Dieser Unterschied ist darauf zurückzuführen, daß einige Befehle der CRAY Y-MP mehr Takte verbrauchen als die gleichen Befehle der CRAY X-MP. Ein Beispiel dafür ist die Zuweisung eines skalaren Wertes auf ein Adreßregister. In diesem Zusammenhang ist es sehr schwierig, die Änderung der Ausführungszeit in Takten an einzelnen Befehlen festzumachen, da die Verzögerung oft erst im Zusammenhang mit der Belegung verschiedener Prozessorressourcen auftritt. Die Verlängerung der Plateaus macht jedoch deutlich, daß es Unterschiede im Aufbau der Prozessoren gibt, die auch zu Unterschieden in der Anzahl der Ausführungstakte führen können.

Die Addition ist auch ein Beispiel für die Aufteilung des erzeugten Codes in Vektorabschnitte der Länge 128 (vgl. Abschnitt 3.2). Dies kommt zum Tragen, sobald die Vektorlänge den Wert 128 überschreitet. Im Aufbau der Meßkurven ist dies zu erkennen an

- dem stärkeren Anheben der Meßkurven jeweils bei den zweiten 64 Vektorelementen eines Abschnitts von 128 Elementen und dem relativen Zurückfallen bei den ersten 64 Elementen des Abschnitts (Ausführungszeit),
- den längeren Plateaus bei der Vektorlänge 129, 257, 385 gegenüber den Plateaus bei der Vektorlänge 64, 192, 320 (Taktdifferenz).

Der unterschiedliche Hub der Meßkurven liegt an der Aufteilung der Vektorlänge in Vektorabschnitte, die in zwei verschiedenen Code-Abschnitten (I, II) berechnet werden. Zum Beispiel wird ein Vektor der Länge 200 in drei Teilvektoren der Länge 8, 64, 128 aufgeteilt, wobei die ersten beiden Teile von Code I (I_a, I_b) , der dritte von Code II (II_a) berechnet wird. Bei der Variation der Vektorlänge erfolgt die Aufteilung so, daß in Code II immer Vektorabschnitte der Länge 128 berechnet werden, bei I_b einmalig ein Teilvektor der Länge 64 und bei I_a maximal 64 Vektorelemente berechnet werden. Der Teilvektor I_a tritt immer auf, der Code II kann mehrfach durchlaufen werden. Die Zer-

legung bei Vektorlänge 328 ist z.B.: I_a, I_b, II_a, II_b. Die Bearbeitung der ersten beiden Teilvektoren I_a und I_b benutzt mit demselben Code zweimal dieselben Vektorregister, was zu Wartezeiten auf Ressourcen führt. Außerdem behandelt der Code I mehr Sonderfälle und ist deshalb etwas länger. Fällt der Teilvektor I_b mit 64 Vektorelementen weg, wird ein kürzerer Code durchlaufen, und die Ressourcen werden besser genutzt. Die Ausführungszeit wird deshalb relativ kürzer.

Für die Länge der Plateaus ist der Übergang vom Abschnitt I_a zur nächsten Vektoroperation entscheidend. Daraus ergeben sich drei Fälle:

1. I_a ist der einzige Vektorabschnitt: Es folgt der Code der Zeitmessung. Dieser Code enthält eine Vektoroperation, damit die zu messende Vektoroperation ausgeführt und nicht nur aufgesetzt wurde, bevor die Zeitmessung erfolgt (vgl. Abschnitt 2.3.4).
2. I_a und I_b treten auf: Die Wiederholung des Codes I bestimmt die Länge des Plateaus.
3. Teilvektor II_a tritt auf, aber nicht Teilvektor I_b: Der Übergang von Code I auf Code II bestimmt die Länge des Plateaus.

Die bestimmende Differenz zwischen den Ausführungszeiten auf Y-MP und X-MP beträgt konstant einen Takt, d.h. sie ändert sich nicht mit der Vektorlänge. Die Differenz läßt sich durch eine längere Ausführungszeit vor der ersten Vektoroperation begründen.

Mit Hilfe der Meßkurve kann man jetzt auch die Werte für $n_{1/2}$ und r_∞ interpretieren (Tab. 2). Der Wert für r_∞^{loop} ist bei beiden Rechnern relativ nahe am **theoretischen Grenzwert** (r_∞^θ). Dieser ist dann erreicht, wenn in jedem Takt ein Schleifendurchlauf durchgeführt wird, d.h. er hat den Wert 1000 (Schleifendurchläufe pro 1000 Takte). Zeitbezogen berechnet sich r_∞^θ als Kehrwert der Prozessorzykluszeit (Taktdauer), da maximal pro Takt ein Ergebnis aus einer Pipeline erreicht werden kann (X-MP: $r_\infty^\theta = 117$ MFLOPS , Y-MP: $r_\infty^\theta = 166$ MFLOPS). Setzt man die Steigungsabschnitte bis zum übernächsten Teilvektor der Länge 64 linear fort, erkennt man die geringe Verschiebung der Steigungsabschnitte in Zeitrichtung. Diese Verschiebung und die etwas ungünstige Implementierung des Code-Abschnitts I_b sind die Gründe für die Abstriche von den maximalen Werten.

Der Wert für $n_{1/2}$ kann durch folgende Einflußfaktoren erklärt werden:

- Zeitverbrauch durch Initialisierung der Schleifensteuerung,
- Zeitverbrauch durch das Durchlaufen der Additionspipeline,
- Verschiebung der Ausgleichsgerade durch die Plateaus,
- Verschiebung der Ausgleichsgerade durch die belegten Ressourcen beim Auftreten von Teilvektor I_b.

Im Sinne der Hockneyschen Interpretation des Parameters $n_{1/2}$ tritt hier also sowohl Start-up-Overhead als auch vorhandener Parallelismus in Erscheinung. Der Unterschied zwischen X-MP und Y-MP ist dabei durch die zusätzlich benötigten Takte bei der Schleifeninitialisierung und bei den Plateaus zu erklären.

Die isolierte Vektormultiplikation ohne Speicherzugriffe unterscheidet sich von der Addition durch die um eine Stufe längere Pipeline in der Funktionseinheit. Dies wirkt sich im Vergleich zur Addition beim r_∞ wenig aus, führt aber beim $n_{1/2}$ zu einer Verlängerung um etwas mehr als eine Gleitkommaoperation.

3.5.1.2 Vektordivision

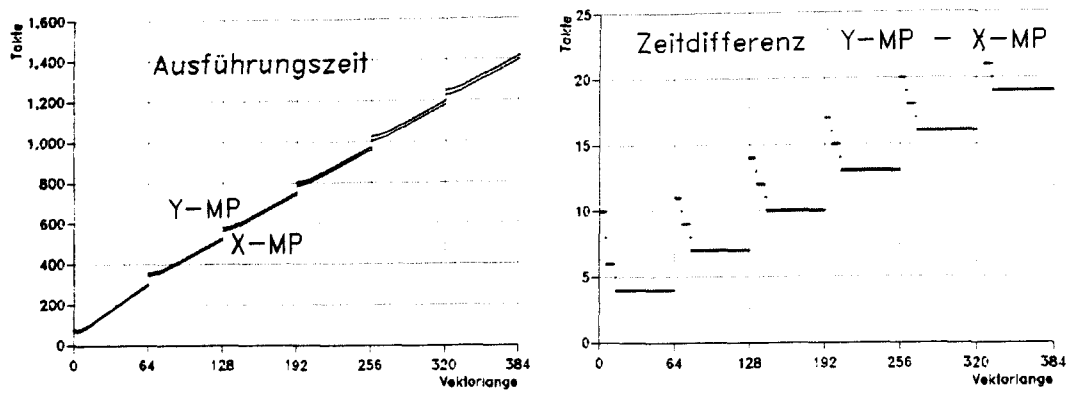


Abb. 16. Vektordivision ohne Speicherzugriffe

Die isolierte Vektordivision ohne Speicherzugriffe unterscheidet sich grundlegend von der Addition und Multiplikation, da hier nicht nur die Operation einer Funktionseinheit gefordert ist, sondern ein Zusammenspiel aus Reziprokwertbildung und Multiplikation stattfindet (Abb. 16). Die Berechnung wendet das Newton-Verfahren zur Nullstellenberechnung an. Dies wird anschaulich in [HoJe,88] beschrieben. Es werden dazu vier Vektoroperationen benötigt. Da hierbei mehr als vier der acht Vektorregister verwendet werden, ist eine Zusammenfassung in Vektorabschnitte der Länge 128 nicht sinnvoll und wird auch vom CFT77-Compiler nicht durchgeführt. Die vier Vektoroperationen nutzen kein Chaining. Sie sind aber voneinander abhängig und können deshalb auch nicht parallel ablaufen. Bei Vektorlängen bis 64 Elemente werden daher pro Element vier Takte benötigt. Bei Vektorlängen von mehr als 64 Elementen gibt es mehrere Teilvektoren, in die der Vektor zur Bearbeitung zerlegt wird. Die letzte Vektoroperation eines Teilvektors kann jeweils mit der ersten Vektoroperation des nächsten Teilvektors parallel durchgeführt werden. Die Aufsetzzeiten der Befehle zwischen den beiden Vektoroperationen werden dabei mitgemessen, weil sie nicht durch die Vektoroperationen überlappt werden. Deshalb ist die absolute Verschiebung der Zeiten der verschiedenen Teilvektoren zueinander bei der Division deutlich höher als bei der Vektoraddition oder -multiplikation. Außerdem wirkt sich die größere Taktanzahl einiger Befehle bei der Y-MP hier in einem Gesamtunterschied aus, der von Teilvektor zu Teilvektor um drei Takte wächst. Für die taktbezogene Grenzleistung bedeutet dies einen geringen Vorteil für die X-MP, der aber absolut kaum Bedeutung hat. Im Vergleich zur theoretischen Grenzleistung erhält man eine Verlangsamung um einen Faktor, der zwischen drei und vier liegt (X-MP 3.24, Y-MP 3.29).

Die Plateaus sind als Rundung in den Anfängen der Teilvektoren zu 64 in der Meßkurve vorhanden. Die Rundung ist jeweils das Ergebnis der vier Vektoroperationen, deren Ausführungszeit erst nach und nach die Ausführungszeit aller Operationen, die zwischen ihnen liegen, übertrifft. Auch hier sind die Plateaus bei der Y-MP etwas länger als bei der X-MP.

Der Wert für $n_{1/2}$ ist deutlich geringer als bei der Vektoraddition. Die Analyse der Meßwerte zeigt, daß sich zwar der Wert von t_0 erhöht hat, jedoch bewirkt der deutlich verringerte Wert für r_∞ ein insgesamt kleineres $n_{1/2}$ (vgl. Formel (5)).

3.5.2 Unterschiede des Speicherverhaltens

Es ist möglich, aus FORTRAN heraus Vektoroperationen zu erzeugen, die ausschließlich Speicherzugriffe durchführen. Dies erreicht man zum Beispiel durch eine Zuweisung zweier Vektoren, die in Lade- und Speicherbefehle des Prozessors umgesetzt wird. Es kann aber trotzdem notwendig sein, den Assembler-Code, den der FORTRAN-Compiler erzeugt, zu modifizieren. Dies geschieht, um eine Verfälschung der Messung zu vermeiden. Diese Verfälschungen sind in der Unabhängigkeit der Ressourcen (hier Vektorverarbeitung und Taktzählregister) begründet und führen dazu, daß die Ausführungszeiten von Befehlen vor dem Meßaufruf mitgemessen werden oder Ausführungszeiten der letzten Befehle des Kerns nicht vollständig gemessen werden (vgl. Abschnitt 2.3.4 und Abb. 10). Im Gegensatz zur Messung von Prozessorinstruktionen gibt es bei Speicherbetrachtungen die Notwendigkeit, die Messungen zu variieren, und zwar durch Veränderung der Zugriffsschrittweite und der Adressen der benutzten FORTRAN-Felder.

Tab. 3 gibt eine Übersicht der in den folgenden Abschnitten behandelten Programmkerne für Speicherzugriffe. Bei indizierter Zuweisung ($A(ISC(IS)) = B(IGT(IG))$, wobei die INTEGER-Felder ISC und IGT von 1 aufsteigend vorbesetzt sind) ergibt sich der niedrige Wert für r_∞^{loop} aus der Tatsache, daß sowohl für das Laden als auch das Speichern zwei Datenströme zwischen Speicher und CPU fließen, die Indexelemente und die eigentlichen Feldelemente. Eine detaillierte Betrachtung der indizierten Adressierung erfolgt hier nicht.

	r_∞^{loop}		$n_{1/2}$	
	X-MP	**Y-MP**	**X-MP**	**Y-MP**
Load1[3]	790.67	771.78	22.12	24.86
Load2[3]	790.67	767.58	27.88	30.12
Store[1]	906.27	914.82	37.64	48.30
Store[3]	901.68	914.46	31.56	36.23
Zuweisung[2]	872.27	897.30	50.40	65.46
Zuweisung[3]	704.65	742.98	35.37	41.15
Indiziert[4]	381.48	372.96	37.46	37.65

Tab. 3. **r_∞ und $n_{1/2}$ bei Speicherzugriffen:** Bedeutung:
1): CAL aus FORTRAN-Zuweisung
2): FORTRAN-Kern
3): CAL aus FORTRAN-Division
4): CAL aus FORTRAN-indizierter Zuweisung
Die Grenzleistung ist taktbezogen und wird als $r_\infty^{loop} \cdot$ 1000 angegeben.

3.5.2.1 Vektorladeoperationen

Der Assembler-Code für die Vektorladeoperationen ist aus dem Code für die Vektordivision abgeleitet worden. Der Assembler-Code aus der FORTRAN-Zuweisung ist hier bewußt nicht als Grundlage verwendet worden. Es entsteht hierbei nämlich durch die Benutzung verschiedener Vektorregister beim Laden genau eines Vektors die Eigenschaft, daß die beiden Lade-Ports der CPU (Port A und B) parallel für aufeinanderfolgende Ladebefehle benutzt werden.
Dieser Effekt ist in Verbindung mit der Zuweisung zweier Vektoren nicht mehr möglich, da hier Lade- und Speicherbefehle paarweise auftreten. Es existiert aber nur ein Speicherport (Port C), so daß dieser die Ressource darstellt, auf die die Ladeoperationen warten müssen. Bei der aus der Division abgeleiteten Ladeoperation (Load1), stellen die Register die „knappe" Ressource dar, da der gleiche Code für Vektorabschnitte der Länge 64 benutzt wird. Die Grenzleistung der Vektorladeoperation liegt deutlich unterhalb der Grenzleistung der Addition ohne Speicherzugriff. Dafür liegen die Werte für Halbleistungslänge niedriger als die Werte des entsprechenden Rechners bei allen betrachteten arithmetischen Operationen. Dies läßt darauf schließen, daß man sich ein kleines $n_{1/2}$ durch ein niedriges r_∞ erkauft. Im Vergleich der beiden Systeme untereinander zeigt sich ein geringer Vorteil bei der Grenzleistung für die CRAY X-MP, was auch für die Halbleistungslänge gilt. Hier macht sich die bei der CRAY Y-MP etwas längere Ladepipeline bestehend aus Port und Speicherzugriffsnetzwerk bemerkbar. Der Übergang auf zwei Ladeoperationen (*Load2*) pro Schleifendurchlauf verändert die Situation gegenüber dem einfachen Load wenig, da hier die zwei vorhandenen Lade-Ports zum Tragen kommen. Der für beide Rechner gemeinsame Unterschied in der Halbleistungslänge ist auf die Adreßoperationen und das Aufsetzen der zweiten Ladeoperation zurückzuführen.

3.5.2.2 Vektorspeicheroperation

Die Vektorspeicheroperation ist in zwei Versionen getestet worden. Die eine ist aus der FORTRAN-Zuweisung (Aufteilung in Vektorabschnitte der Länge 128) und die andere aus der FORTRAN-Division (Teilvektoren der Länge 64) entwickelt worden. Die Grenzleistung unterscheidet sich praktisch nicht zwischen den beiden Kernen. Dagegen schlägt sich der etwas aufwendigere Code in dem erhöhten Wert für $n_{1/2}$ nieder. Im Vergleich zwischen X-MP und Y-MP zeigt sich unabhängig vom benutzten Kern ein geringer Vorteil in der Grenzleistung für die Y-MP, dem ein Nachteil bei der Halbleistungslänge gegenübersteht. Die Werte der beiden Kenngrößen erreichen bei der Speicheroperation fast die entsprechenden Werte der Addition, wobei das $n_{1/2}$ in der Version, die von der Division abgeleitet ist, sogar die Addition unterbietet. Der wichtigste Unterschied ist aber die bessere Grenzleistung der Speicheroperation gegenüber der Ladeoperation (Y-MP 18%, X-MP 12%). Dem steht eine deutliche Erhöhung im $n_{1/2}$ entgegen (Y-MP 43%, X-MP 46%).

3.5.2.3 Vektorzuweisung

Die Vektorzuweisung kann als Zusammenfassung der beiden gerade analysierten Speicherzugriffe Laden und Speichern angesehen werden. Da sich die beiden

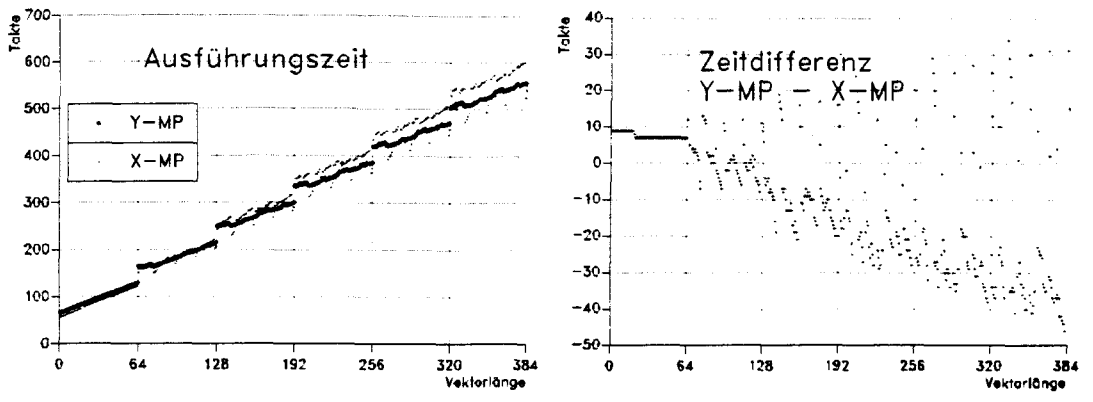


Abb. 17. Vektorzuweisung: Der Programmkern ist aus dem Assembler-Code der Division entwickelt worden.

Vektoroperationen verketten, erwartet man eine Grenzleistung, die knapp unterhalb der der dominierenden niedrigen Leistung der Vektorladeoperation liegt. Betrachtet man die Ergebnisse für die aus der Division abgeleitete Zuweisung, so sieht man diese Erwartung erfüllt. Der vom Compiler erzeugte Code für die Zuweisung zeigt, daß durch geschickte Ausnutzung aller Ressourcen (Register, beide Load Ports) eine Grenzleistung nahe der reinen Speicherleistung erreicht werden kann, was aber durch eine deutlich erhöhte Halbleistungslänge bezahlt werden muß. Der Vergleich der Grenzleistungen zwischen den beiden Rechnern zeigt unabhängig vom betrachteten Zuweisungskern, daß der Vorteil beim Speichern bei der Y-MP die Gesamtleistung stärker zu beeinflussen scheint als der Vorteil des Ladens bei der X-MP. Dieser Vorteil kann aber auch eine Folge von vermehrten Speicherkonflikten bei der X-MP gegenüber der Y-MP sein (Abb. 17). Die Differenzkurve zeigt eine Dominanz von Meßwerten der X-MP, die höher als die der Y-MP liegen. Die Meßkurve wird bei beiden Rechnern von starken Schwankungen geprägt, die sich deutlich vom monoton ansteigenden, vorhersagbaren Verlauf der Meßkurven bei den isolierten Prozessoroperationen unterscheidet.

Beim Vergleich der beiden Varianten für die Vektorzuweisung werden also die Vorteile der Optimierung der Ressourcen durch den Compiler deutlich. Außerdem wird sowohl an den Ergebnissen für r_∞ als auch für $n_{1/2}$ klar, daß es sehr schwierig ist, aus den verwendeten Grundoperationen auf das Verhalten der Verknüpfung dieser Operation zu schließen.

3.5.3 Das Zusammenspiel von Speicher und Prozessor

In diesem Abschnitt wird von der Messung an Assembler-Kernen zur Messung an FORTRAN-Kernen übergegangen. Diese Kerne liefern eine Grundmenge von Leistungsbewertungen einfacher und häufig benutzter mathematischer Operationen. Tab. 4 gibt eine Übersicht der durchgeführten Messungen. Die Unterschiede der einzelnen Messungen werden im folgenden erklärt.

	r_∞^{loop}		$n_{1/2}$	
	X-MP	Y-MP	X-MP	Y-MP
Addition	823.31	864.78	63.71	82.92
Multiplikation	824.08	850.32	63.96	80.48
Division	261.72	253.32	30.23	31.68
Skalarmultiplikation	870.74	894.60	58.75	76.11
Triade	543.49	534.66	50.32	50.70
SAXPY	744.69	752.16	37.35	42.63
Skalarprodukt	940.78	947.04	300.08	337.38
Logarithmus	42.59	38.70	8.30	7.80

Tab. 4. **r_∞ und $n_{1/2}$ bei Speicherzugriffen in Verbindung mit Arithmetik:** Die Programmkerne sind in FORTRAN codiert. Die Grenzleistung ist taktbezogen und wird als $r_\infty^{loop} \cdot 1000$ angegeben.

3.5.3.1 Arithmetische Grundoperationen

Die Addition $(A(I) = B(I) + C(I))$ und die Multiplikation $(A(I) = B(I) \cdot C(I))$ sind Operationen, die mit zwei Ladebefehlen und einem Speicherbefehl pro Element die drei vorhandenen Ports der Hardware voll ausnutzen und damit die maximale auf einem Prozessor bei Schrittweite eins erzeugbare Last widerspiegeln. In Abb. 18 sind die Ergebnisse für die Addition dargestellt.

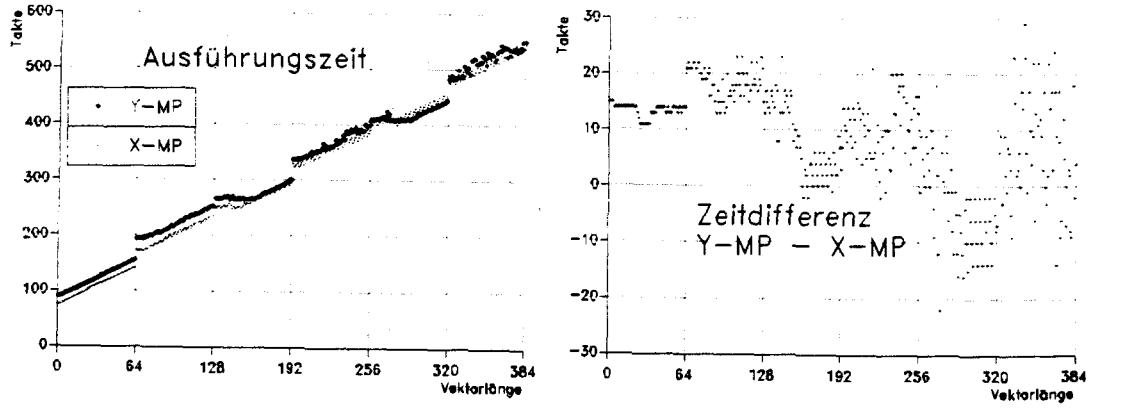


Abb. 18. **Zuweisung mit Addition**

Die Grundstruktur der Meßkurven, wie sie die Addition ohne Speicherzugriffe geliefert hat (Abb. 14), ist wegen der Streuung der Meßwerte hier kaum noch zu erkennen. Dies spiegelt sich in der Differenzdarstellung wider. Bei der Grenzleistung der beiden Systeme hat die Y-MP leichte Vorteile gegenüber der X-MP, was auf eine Dominanz des Speicherverhaltens hindeutet. Die Werte für die Halbleistungslänge sind auf dem Grundniveau aufgebaut, das sich schon als typisch für die Messung von Compiler und Rechner zusammen bei der Zuweisung aus FORTRAN gezeigt hat. Die Werte für $n_{1/2}$ sind bei der Y-MP höher als bei der X-MP. Betrachtet man die Addition mit Speicherzugriffen als eine Mischung aus FORTRAN-Zuweisung und isolierter Addition, so ergibt sich

sowohl bei der Grenzleistung als auch bei der Halbleistungslänge ein deutlich schlechterer Wert als bei beiden zugrundeliegenden Kernen. Die Grenzleistung ist aber immer noch höher als bei der aus der Division entwickelten isolierten doppelten Ladeoperation (Teilvektoren der Länge 64 kontra Vektorabschnitte der Länge 128).

Die Unterschiede zwischen Addition und Multiplikation spielen bei den Kenngrößen praktisch keine Rolle, d.h. sie liegen im Rahmen der Meßfehler, wie man im Vergleich der Grenzleistungen zwischen Addition und Multiplikation bei der X-MP sieht.

Die Division ($A(I) = B(I)/C(I)$) zeigt bei beiden Kenngrößen keine besonderen Unterschiede zwischen den beiden Systemen. Es fällt auf, daß im Gegensatz zur Addition und Multiplikation die X-MP die etwas bessere Grenzleistung aufweist. Erwartet man als Ursache Eigenschaften des Speicherzugriffs, so kann man dies an dem Verhältnis von Speicher- und arithmetischen Operationen festmachen (3/4). Die Division ist aus vier arithmetischen Operationen aufgebaut (siehe Abschnitt 3.5.1.2), die in mindestens drei nicht überlappenden Ausführungsphasen liegen. Auf jede dieser drei Phasen entfällt also im Durchschnitt nur eine Speicheroperation. Daraus ergibt sich eine deutlich geringere Belastung des Speichers als bei der Addition, bei der mit jeder Arithmetikoperation drei Speicheroperationen auftreten, die im günstigsten Falle parallel ablaufen können.

Das Produkt aus einer skalaren Größe und einem Vektor ($A(I) = S \cdot C(I)$) ist vom Aufbau aus Speicher- und Arithmetik-Operationen betrachtet zwischen Vektorzuweisung und Vektorprodukt angesiedelt, da es eine Lade- und eine Speicheroperation pro Teilvektor benötigt. Aus dieser Eigenschaft kann man die Werte von r_∞ und $n_{1/2}$ im Vergleich der drei Kerne erklären. r_∞liegt auf dem gleichen Niveau wie bei der Zuweisung, $n_{1/2}$ liegt zwischen den beiden Vergleichswerten.

3.5.3.2 Einige Vektorausdrücke

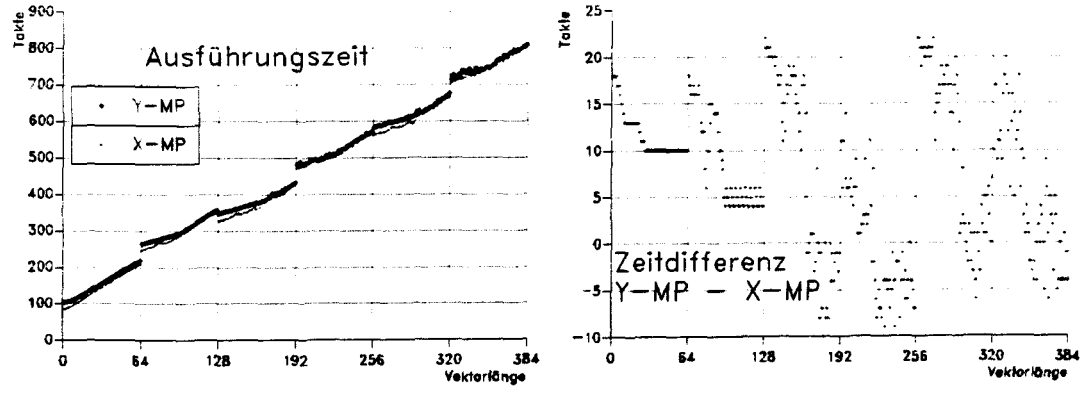


Abb. 19. Triade

Es werden Kerne mit Ausdrücken betrachtet, die vektorisieren. Zu den betrachteten Grundoperationen hinzu wird durch die Kerne Triade ($A(I) = B(I) \cdot C(I) + D(I)$) und SAXPY ($B(I) = B(I) + S \cdot C(I)$) die Verkettung der Funktionseinheiten ausgenutzt (Abb. 19). In beiden Fällen werden die Ergebnisse der Multiplikation direkt bei der

Addition weiterverwendet. Der Unterschied der beiden Kerne beruht darauf, daß die Triade drei Ladeoperationen durchführt, SAXPY aber nur zwei. Da nur zwei Ladeports vorhanden sind, müssen die drei Ladeoperationen der Triade getrennt ausgeführt werden. Deshalb ist r_∞ bei der Triade deutlich niedriger als bei SAXPY, überschreitet aber die halbe Grenzleistung. Dies ist das Ergebnis einer Überlappung der Ausführung der Vektoroperationen zwischen zwei Teilvektoren der Länge 64 (Abb. 20).

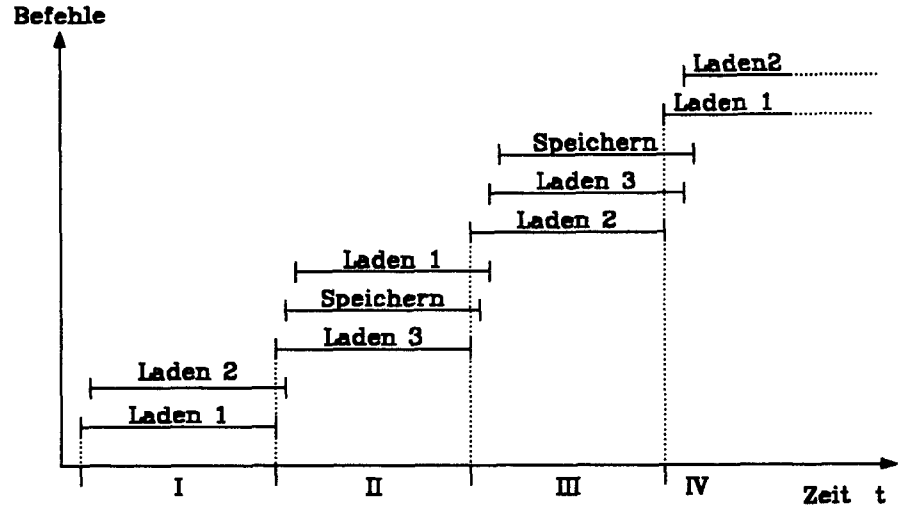


Abb. 20. Ausführungsüberlappung bei der Triade: Die Ausführungszeit zweier Teilvektoren teilt sich auf drei Zeitabschnitte (I, II, III) auf. (Ab Phase IV wiederholt sich das Überlappmuster aus I, II und III.) Es wird dabei nur der Einfluß der Speicheroperationen betrachtet (Laden 1, Laden 2, Laden 3, Speichern).

Theoretisch ist dadurch eine Grenzleistung möglich, die 2/3 der theoretischen Grenzleistung r_∞^θ (hier $r_\infty^\theta = 1000$) entspricht. Hier wird ein Wert deutlich über 500 bei möglichen 666 erreicht. Eine Voraussetzung für die Überlappung ist aber, daß die anderen Ressourcen nicht belegt sind. Dabei kommt wieder die Aufteilung der Bearbeitung in Vektorabschnitte der Länge 128 durch den CFT77-FORTRAN-Compiler zum Tragen, d.h. die Überlappung findet erst bei Vektorlängen statt, die größer als 128 sind. Da sowohl bei der Triade als auch bei SAXPY zwei arithmetische Operationen im Kern durchgeführt werden, ist hier r_∞^{flop} doppelt so hoch wie r_∞^{loop}. Aus Sicht der Speicherzugriffe bietet sich bei SAXPY ein Vergleich mit der Addition oder der Multiplikation von Vektoren an (Tab. 4). Hierbei zeigt sich, daß das r_∞ zwischen 10 und 20 Prozent niedriger ist, aber auch das $n_{1/2}$ bei SAXPY einen kleineren Wert besitzt. Im Vergleich der beiden Rechner untereinander sind die Unterschiede sowohl bei der Triade als auch bei SAXPY relativ gering.

Die Kerne für Skalarprodukt und Logarithmus behandeln Operationen, die für die Berechnung auf CRAY-Systemen besonders aufwendig sind. Die Problematik bei der Berechnung des Skalarprodukts (generell Akkumulieren von Produkten: $S = S + B(I) \cdot C(I)$) auf CRAY-Rechnern besteht darin, daß es keine Hardware-Einrichtung der CPUs gibt, um die Elemente eines Vektorregisters in einem

Skalarregister aufzusummieren. Deshalb wird der Code des Skalarprodukts genauer untersucht.

Das Grundprinzip der Vektorreduktion besteht darin, daß Vektoren der Länge n in k Teilvektoren der Länge ℓ aufgeteilt werden. Anders als bei der sequentiellen Vektorreduktion werden nun nicht benachbarte Vektorelemente in einer skalaren Variablen aufsummiert, sondern die gebildeten Teilvektoren werden durch Vektoraddition komponentenweise aufaddiert. Damit reduzieren sich die k Teilvektoren zu einem Teilvektor, der ℓ Teilsummen enthält. Auf CRAY Rechnern entspricht ℓ der Länge 64 der Vektorregister. Ist n nicht durch 64 teilbar, so wird der Restvektor mit Nullelementen auf die Länge 64 aufgefüllt. Zusätzlich führt der Compiler noch die Aufteilung in Vektorabschnitte (vgl. Abschnitt 3.2) durch.

Das Ergebnis ist in allen Fällen ein Vektorregisterinhalt von 64 Elementen, die aufsummiert werden müssen. Bei dieser Summation werden, wenn möglich, Vektoroperationen auf Teilvektoren angewendet. Es verbleibt ein Rest von skalaren Additionen. Im Detail ergeben sich folgende Schritte bei der Zusammenfassung der letzten 64 Summanden:

1. Für die Aufteilung der Elemente eines Vektorregisters auf zwei oder mehrere Vektorregister muß der Hauptspeicher benutzt werden, weil Zuweisungen zwischen Vektorregistern immer beim ersten Element beider Vektorregister anfangen müssen. Die 64 Vektorelemente werden so zunächst abgespeichert und drei Vektoradditionen mit Vektorlänge 16 durchgeführt. Es werden zuerst die zweiten 16 Vektorelemente wieder geladen und zu den ersten 16 Elementen, die sich noch im Register befinden addiert. Danach werden die dritten und vierten 16 Elemente aus dem Speicher geladen und addiert. Die beiden Summen zu 16 Elementen werden dann zu einer Summe zu 16 Elementen vektoriell addiert.

2. Die verbliebenen 16 Elemente der Teilsummen werden auf 8 reduziert. Dazu wird eine Verschiebungsoperation auf Vektorregistern ausgenutzt. Diese verwendet als Quelloperand zwei benachbarte Elemente eines Vektorregisters. Diese benachbarten Elemente werden als 128 Bit-Wort aufgefaßt, um 64 Bit nach links verschoben und die oberen 64-Bit in ein Element des Zielregisters abgelegt. Diese Operation wirkt, auf das gesamte Zielregister bezogen, wie eine Verschiebung um ein Element zum Anfang hin. Das Ergebnisregister wird zu dem 16-elementigen Ausgangsvektor addiert. Die Registerelemente 0, 2, 4, 6, 8, 10, 12, 14 enthalten jetzt die Summen der vorher benachbarten Elemente. Damit ist die Reduktion auf 8 Summanden erreicht. Es fallen erneut die Grenzen der CRAY-Hardware auf. Eine intuitive Vorgehensweise wäre hier, die hinteren acht Quellelemente (Elemente 8 bis 15) an den Anfang (Elemente 0 bis 7) eines anderen Vektorregisters zu kopieren und nur acht Elementpaare vektoriell zu addieren.

3. Die verbliebenen acht Summanden werden jetzt skalar addiert. Dies nutzt die Möglichkeit aus, einzelne Elemente eines Vektorregisters in ein Skalarregister zu übertragen.

Der Aufbau des vom Compiler erzeugten Codes zeigt deutliche Schwächen. Um den gleichen Code für die ersten 64 und die weiteren bis maximal 128 Elemente benutzen zu

können, wird ein Vektorregister auf allen 64 Elementen mit Null vorbesetzt. Zusätzlich ersetzt eine unnötige Addition diese Nullen durch die Ergebnisse der Multiplikation. Besonders störend wirkt sich bei kurzen Vektorlängen das unflexible Verfahren der Zusammenfassung der letzten 64 Summanden aus. Es wird nicht nach der Vektorlänge unterschieden, zum Beispiel in weniger als 32 oder weniger als 16 Elemente. Hierdurch würden aufwendige Reduktionsschritte entfallen.

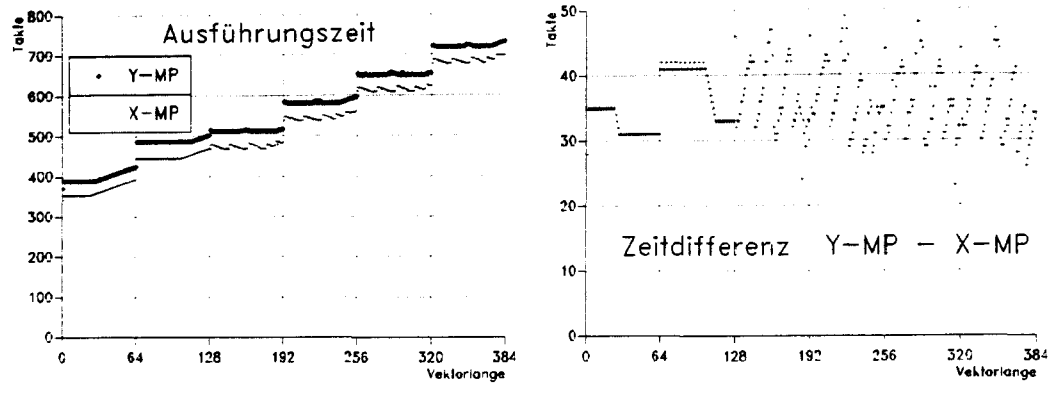


Abb. 21. Skalarprodukt

Der Aufbau des Codes für das Skalarprodukt spiegelt sich im Aufbau der Meßkurve wider (Abb. 21). Besonders auffällig ist dabei die lange Zeit, die schon für die ersten Elemente benötigt wird. Diese läßt sich durch das aufwendige Vorgehen beim Zusammenfassen der letzten 64 Elemente begründen. Die Länge der Plateaus erstreckt sich hier fast über den ganzen Teilvektor der Länge 64. Das ist auf die Überlappung der Initialiserung eines Vektoregisters auf volle Länge mit dem Wert Null und der Berechnung des Produkts für den ersten Teilvektor mit kürzerer Länge zurückzuführen. Dadurch entsteht eine Meßkurve, wie man sie ähnlich bei Parallelrechnern erwartet (vgl. Abb. 12), wobei aber die hohe Aufsetzzeit t_0 vom Ideal abweicht. Ein hohes Maß an Überlappung wird auch beim Übergang zur Berechnung von Teilvektoren der Länge 128 erreicht, was man an dem praktisch nicht vorhandenen Sprung bei Vektorlänge 128 erkennt.

Der Wert der Grenzleistung r_∞^{loop} (Tab. 4) ist bei beiden Rechnern sehr nahe am erreichbaren Maximum und hat sogar den Spitzenwert für alle bisher betrachteten Kerne, was sogar zu einem doppelt so hohen Wert für r_∞^{flop} führt, da pro Schleifendurchlauf zwei Gleitkommaoperationen durchgeführt werden. Dies wird natürlich durch einen stark überhöhten Wert für $n_{1/2}$ erkauft. Der Wert für $n_{1/2}$ liegt bei der Y-MP noch etwas höher als bei der X-MP, was man durch den Abstand der beiden Meßkurven (siehe Differenzkurve) erklären kann.

Der Logarithmus ($A(I) = \log(B(I))$) ist eine vektorisierende Funktion. Der genaue Aufbau soll hier nicht untersucht werden; es ist aber naheliegend, daß relativ wenige Speicheroperationen einer Vielzahl von arithmetischen Operationen gegenüberstehen. Auffallend am Logarithmus ist ein relativ deutlicher Unterschied der Grenzleistung r_∞^{loop}, die bei der CRAY X-MP etwa 10% höher liegt als bei der CRAY Y-MP. Dieser Unterschied führt dazu, daß der Wert für $n_{1/2}$ bei der Y-MP kleiner als bei der X-MP ist, da

durch die größere Steigung der Ausgleichsgerade (vgl. Abb. 13) für die Ausführungszeit als Funktion von n bei gleicher Aufsetzzeit t_0 der Wert für $n_{1/2}$ fällt.

3.5.4 Variation der Schrittweite

Bei der Beschreibung der Systeme wurde im Abschnitt 2.4.4 der verschränkte Aufbau der Bänke des Hauptspeichers der CRAY-Systeme erklärt. Zugriffskonflikte auf Bankebene entstehen, wenn ein neuer Zugriff auf eine Bank erfolgen soll, bevor ein voriger Zugriff abgearbeitet ist (Bankzykluszeit). Offensichtlich sind hier Schrittweiten kritisch, die einen gemeinsamen Teiler mit der Bankanzahl haben. Sie führen nämlich dazu, daß von allen verfügbaren Bänken nur ein Teil tatsächlich benutzt wird [Oed,85].

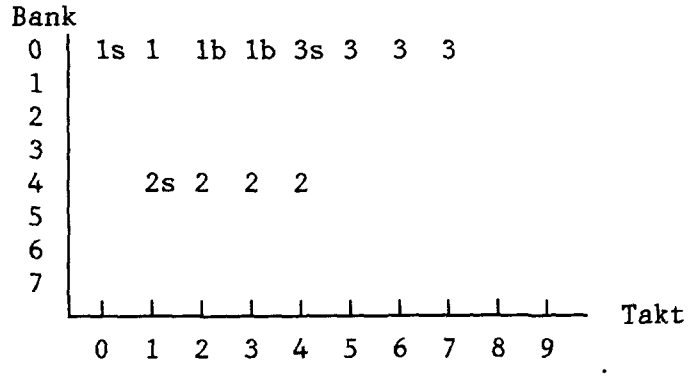


Abb. 22. Bankbelegung bei drei Zugriffen

Das Beispiel (Abb. 22) zeigt drei aufeinanderfolgende Zugriffe eines Datenstroms bei einem Speicher mit acht Bänken und einer Bankzykluszeit von vier Takten. Die gewählte Schrittweite ist dabei vier, so daß Zugriff 3 genau wie Zugriff 1 auf Bank 0 erfolgt. Der Zusatz „b" bei den letzten beiden Takten von Zugriff 1 deutet den Bankkonflikt mit Zugriff 3 bei Bank 0 an. Der Zusatz „s" für den ersten Takt jedes Zugriffs soll die Belegung der Sektion für einen Takt verdeutlichen, die Zugriffe anderer Ports der gleichen CPU verzögern kann. Es ist offensichtlich, daß die Betrachtung der gleichzeitigen Datenströme weiterer Ports und mehrerer Prozessoren das Gesamtbild deutlich verkomplizieren. Ausführlichere Betrachtungen solcher Diagramme und Sachverhalte, die meist aus Simulationen gewonnen werden, finden sich in [Oed,85], [OeLa,85], [OeLa,86], [CaEl,85], [Cala,88] und [ChSm,86].

Die CRAY Y-MP besitzt neben den Sektionen eine zusätzliche Unterteilung in Untersektionen (vgl. Abb. 8). Betrachtet man die Zugriffe einer CPU, so übernehmen die Untersektionen bei der CRAY Y-MP die gleiche Funktion, wie die Bänke bei der CRAY X-MP. Die Untersektionen sind fünf Takte (eine Bankzykluszeit der CRAY Y-MP) belegt und nehmen alle durch eine CPU erzeugbaren Bankkonflikte vorweg. Eine CPU allein kann also keine Bankkonflikte erzeugen. Deshalb kann man bei isolierten Einprozessormessungen den Speicher der Y-MP als in 32 Bänke aufgeteilt ansehen [FaSc,89]. Dies ändert sich erst, wenn man Zugriffe mehrerer CPUs betrachtet.

	r_∞^{flop}		$n_{1/2}$	
Schrittweite	**X-MP**	**Y-MP**	**X-MP**	**Y-MP**
1	100.67	142.60	122.64	145.19
2	74.83	100.43	74.84	84.81
4	69.21	54.36	71.62	41.68
8	41.41	42.76	31.75	32.17
16	39.22	22.08	38.43	17.82
32	19.60	11.06	18.47	8.47
64	9.80	11.04	8.83	6.88
128	9.80	11.05	8.83	7.79
256	9.80	11.10	8.83	8.00

Tab. 5. r_∞ in MFLOPS und $n_{1/2}$ bei der Addition: Die Schrittweite der Speicherzugriffe variiert zwischen 1 und 256. (Startbänke: (0,0,0))

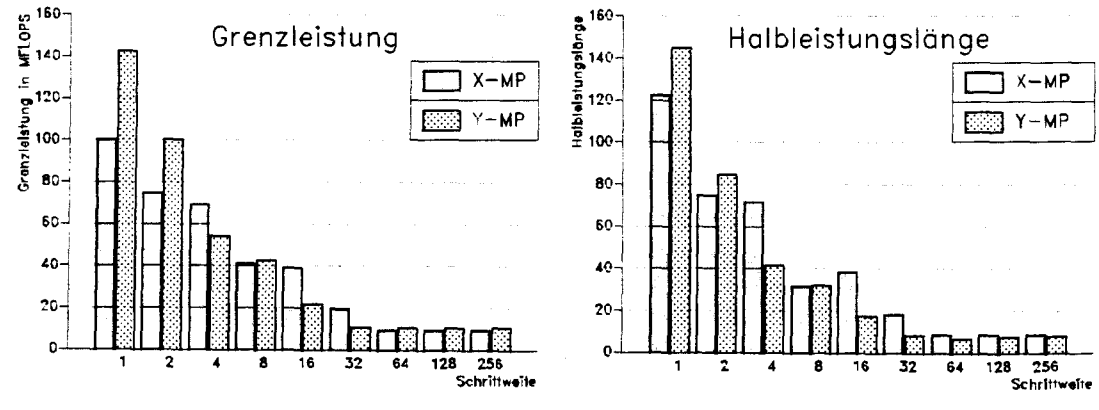


Abb. 23. r_∞ in MFLOPS und $n_{1/2}$ bei der Addition: Die Schrittweite der Speicherzugriffe variiert zwischen 1 und 256. (Startbänke: (0,0,0))

Die Methode der Variation der Schrittweiten wird mit dem Programmkern Addition ($A(I) = B(I) + C(I)$) untersucht. Dabei ist der Aufbau der DO-Schleife etwas aufwendiger als bei den Messungen mit Schrittweite 1 im Abschnitt 3.5.3. Deshalb sind die Messungen nicht direkt vergleichbar. Die Erhöhung in den Werten für $n_{1/2}$ ist gravierend (X-MP 92%, Y-MP 75% gegenüber Tab. 4 auf Seite 38), was die Eigenschaft der Halbleistungslänge unterstreicht, nicht nur von der Hardware sondern auch im großem Maße von dem vom Compiler erzeugten Code abhängig zu sein und schon auf kleine Änderungen der Schleifensteuerung und des Kerns (variable Schrittweite, variable Startelemente) stark zu reagieren.

Als Schrittweite werden Zweierpotenzen benutzt, da diese Teiler der jeweiligen Bankanzahl sind. Die Addition erzeugt drei Datenströme für die Felder A, B, C. Die Bank des Speichers, auf der das erste Element eines Feldes liegt, nennt man die **Startbank** dieses Feldes. Hier sind die drei Felder auf die gleiche Startbank festgelegt (0, 0, 0). Die Tab. 5 und Abb. 23 stellen die Werte für die beiden Kenngrößen r_∞und $n_{1/2}$ abhängig von der Schrittweite und dem benutzten Rechner vor. Da in jedem Schleifendurchlauf

eine arithmetische Operation durchgeführt wird, sind hier r_∞^{loop} und r_∞^{flop} identisch. Die Angaben erfolgen im Gegensatz zu den vorherigen Abschnitten in MFLOPS, da bei der Betrachtung vermehrter Speichereinflüsse die Prozessorzykluszeit nicht mehr die angemessene Bezugsgröße darstellt.

Der grundsätzliche Verlauf der Meßkurve für die Grenzleistung ist fallend mit der Schrittweite. Bei der CRAY X-MP ist das Minimum bei Schrittweite 64 (Anzahl der Bänke) erreicht, bei der Y-MP ist dies bereits bei 32 (Anzahl der Untersektionen) der Fall. Geht man davon aus, daß bei dieser minimalen Grenzleistung die Bankkonflikte die Grenzleistung beherrschen, so kann man die Werte für r_∞ bei Schrittweite 1 als Werte für Zugriffe quasi ohne Bankkonflikte ansehen. Die erwarteten Faktoren zwischen Minimal- und Maximalmalleistung betragen deshalb bei der X-MP zwölf und bei Y-MP 15 (drei Datenströme mal Bankzykluszeit). Diese erwarteten Faktoren werden bei der CRAY X-MP zu 76%, bei der CRAY Y-MP sogar zu 96% erreicht, was ein Indiz für die Vorteile der Speicherstruktur der CRAY Y-MP darstellt. Der Verlauf des Balkendiagramms bei der Halbleistungslänge ist für beide Rechner im wesentlichen analog zum Verlauf im jeweiligen Diagramm der Grenzleistung. Die Halbleistungslänge der CRAY Y-MP gleicht sich mit wachsender Schrittweite dem $n_{1/2}$ der CRAY X-MP an und unterschreitet sie sogar.

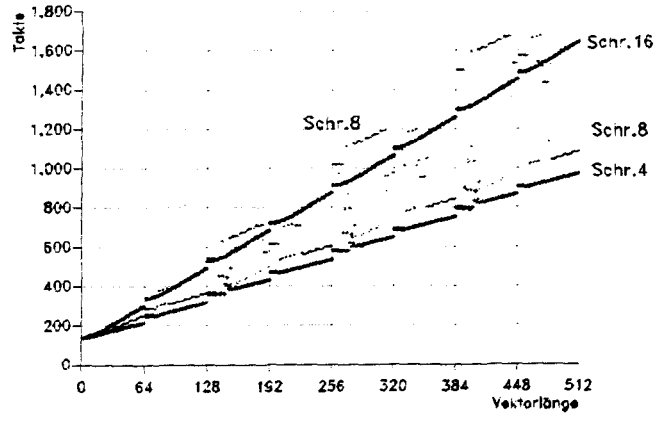


Abb. 24. Ausführungszeiten bei der Addition bei Schrittweite 4, 8 und 16 auf der CRAY X-MP: Man erkennt die Auswirkungen der Speicherkonflikte bei Schrittweite 8

Bei der CRAY X-MP fällt die niedrige Grenzleistung für die Schrittweite acht auf, die sich bei der Halbleistungslänge sogar in einem niedrigeren Wert als bei Schrittweite 16 auswirkt. Diese niedrige Grenzleistung läßt sich mit Hilfe der Meßkurve über die Vektorlänge n etwas besser interpretieren. Hier erkennt man den unruhigen Verlauf der Meßwerte im Vergleich zu den Meßkurven der „benachbarten" Schrittweiten 4 und 16, der nicht allein anhand der Anzahl der verfügbaren Bänke (untere Teilkurve für Schrittweite 8) erklärt werden kann. Hier spielt die Konfliktauflösung, bedingt durch die Sektions- und Portstruktur, in die Meßwerte hinein (obere Teilkurve und die vereinzelten Werte für Schrittweite 8).

3.5.5 Variation der Startbänke

In den Messungen des vorherigen Abschnittes sind die Startbänke der Felder so angeordnet, daß bei den größeren Schrittweiten (X-MP größer als 64, Y-MP größer als 32) die drei Datenströme der Addition an derselben Bank warten müssen. Dadurch sind die erzielten Grenzleistungen bestimmt. Dieses Warten kann man dadurch verkürzen, daß man die Startbänke so zueinander anordnet, daß die Zugriffe der drei Datenströme auf drei disjunkte Teilmengen der Bänke erfolgen. Hier sind die Startbänke der Felder A, B, C zu 0, 1, 2 gewählt (vgl. [Dete,89]).

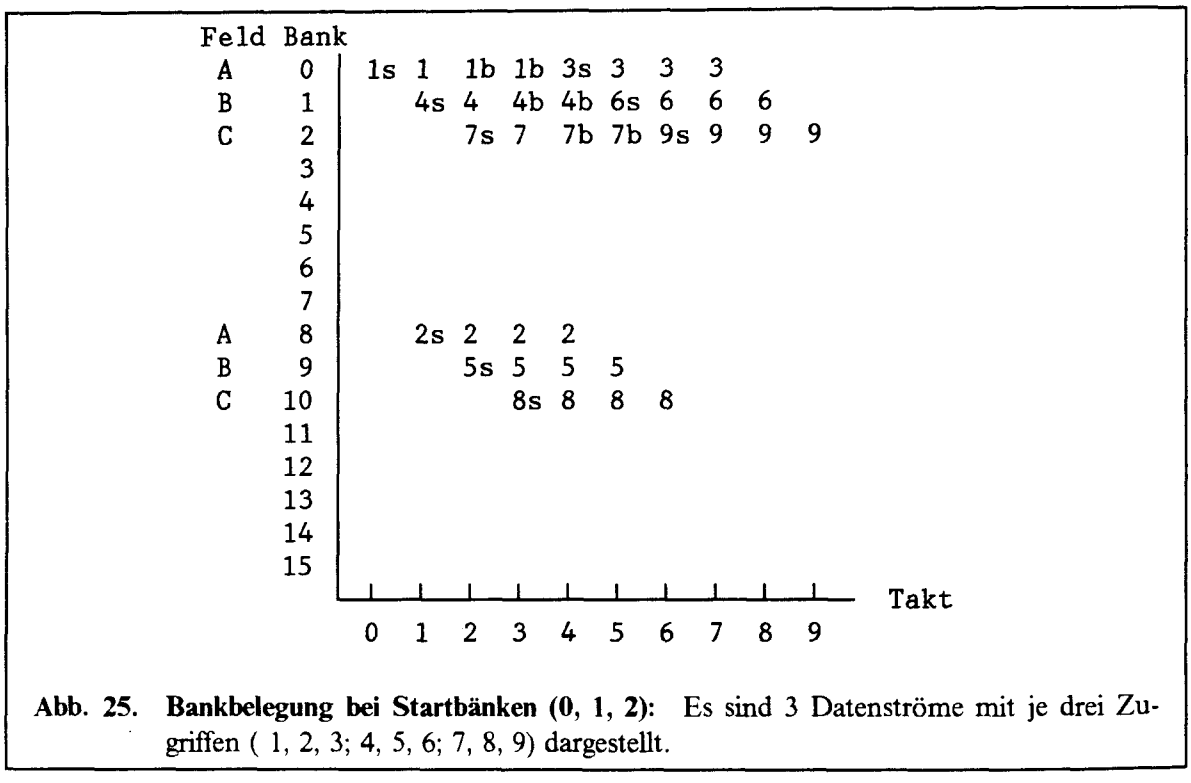


Abb. 25. Bankbelegung bei Startbänken (0, 1, 2): Es sind 3 Datenströme mit je drei Zugriffen (1, 2, 3; 4, 5, 6; 7, 8, 9) dargestellt.

Das Beispiel (Abb. 25) stellt im Gegensatz zum Beispiel in Abb. 22 ein Speichersystem mit 16 Bänken dar. Die Schrittweite acht führt auch hier zu den durch jeden Datenstrom selbst erzeugten Bankkonflikten, die Datenströme untereinander sind aber nur durch die Sektionsaufteilung miteinander verbunden, die hier mindestens vier Sektionen besitzen muß (drei Datenströme) und von Bank zu Bank wechselt (Zugriffe 2s, 4s haben sonst einen Konflikt).

Vermindert man die Schrittweite unter den Wert von vier, ist die Aufteilung der Datenströme auf verschiedene Bänke nicht mehr vorhanden. Bei Schrittweite zwei verwenden Feld A und C die gleichen Bänke, bei Schrittweite eins sind die Datenströme vollständig gemischt. Die Datenströme haben aber auch mehr Bänke als bei Schrittweiten ab vier zur Verfügung, auf die sich die Zugriffe verteilen können.

	r_∞^{flop}		$n_{1/2}$	
Schrittweite	**X-MP**	**Y-MP**	**X-MP**	**Y-MP**
1	95.68	137.80	118.01	130.88
2	105.71	134.68	126.35	125.75
4	104.70	146.35	122.62	140.71
8	78.16	119.84	93.03	116.44
16	39.22	64.14	36.90	58.97
32	19.91	32.58	17.97	28.33
64	9.91	32.90	8.41	29.18
128	9.91	32.86	8.41	29.02
256	9.91	32.76	8.41	30.25

Tab. 6. r_∞ in MFLOPS und $n_{1/2}$ bei der Addition: Die Schrittweite der Speicherzugriffe variiert zwischen 1 und 256. Außerdem sind die Startbänke der Felder (A, B, C) auf 0, 1, 2 festgelegt.

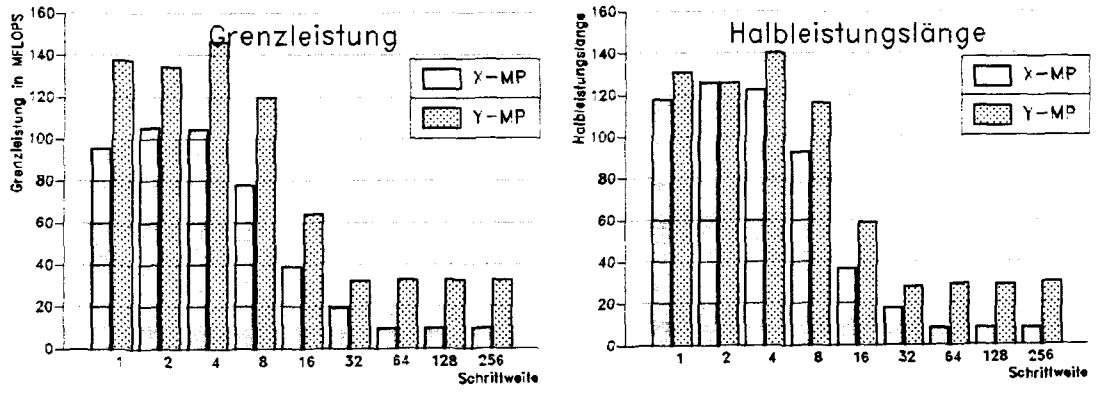


Abb. 26. r_∞ in MFLOPS und $n_{1/2}$ bei der Addition: Die Schrittweite der Speicherzugriffe variiert zwischen 1 und 256. Außerdem sind die Startbänke der Felder (A, B, C) auf 0, 1, 2 festgelegt.

Die Ergebnisse der Messungen sind in Tab. 6 und in Abb. 26 dargestellt. Der grobe Verlauf der Grenzleistung bei der CRAY Y-MP ab Schrittweite vier ist abfallend durch die Verminderung der benutzten Bänke, liegt aber auf deutlich höherem Niveau als bei der auf eine Bank fixierten Anordnung der Startbänke (Tab. 5 und Abb. 23). Bei der minimalen Grenzleistung erhält man hier einen um den Faktor drei verbesserten Wert gegenüber dem Wert bei gleichen Startbänken. Es treten zwar ständig Bankkonflikte auf, diese können aber für die drei Datenströme unabhängig behandelt werden und sind nicht nacheinander auf einer Bank fixiert. Auffällig ist auch, daß die Grenzleistung für Schrittweite vier bei getrennten Startbänken die Grenzleistung bei Schrittweite eins und gemeinsamen Startbänken überschreitet. Für Schrittweite zwei ist bei verschiedenen Startbänken die Grenzleistung deutlich höher als bei gleichen Startbänken, aber niedriger als für Schrittweite eins.

Das Bild der Grenzleistung bei der CRAY X-MP weicht deutlich von dem der Y-MP ab, was vor allem der Vergleich zu den Messungen mit gleicher Startbank deutlich macht. Schon bei Schrittweite 16 ist praktisch keine Verbesserung der Grenzleistung durch die Verschiebung der Startbänke zu erkennen, was auch für die größeren Schrittweiten gilt. Hier kommt die Abhängigkeit der Ports bei der CRAY X-MP zum Tragen, wenn man die Auflösung von Bankkonflikten betrachtet (vgl. Abschnitt 2.4.4). Jeder Bankkonflikt eines Ports unterbricht dabei die Arbeit aller anderen Ports der CPU. Dieser Effekt heißt **Portverzögerung**. Da die Verminderung der benutzbaren Bänke bis hin zu einer Bank pro Datenstrom zu ständigen Bankkonflikten führt, werden die Datenströme an den Ports statt, wie es bei gleichen Startbänken geschieht, an den Bänken serialisiert. Dies dauert im schlechten Fall jeweils drei Takte pro Zugriff, da Konflikte bei gleichzeitigem Bankzugriff nur zwischen CPUs auftreten. Zusätzlich liegen bei Schrittweite 64 alle drei benutzten Bänke in derselben Sektion, was vor jeder Portverzögerung noch zu einem Sektionskonflikt führt. Bei der CRAY Y-MP liegen die Startbänke dagegen gerade in verschiedenen Sektionen.

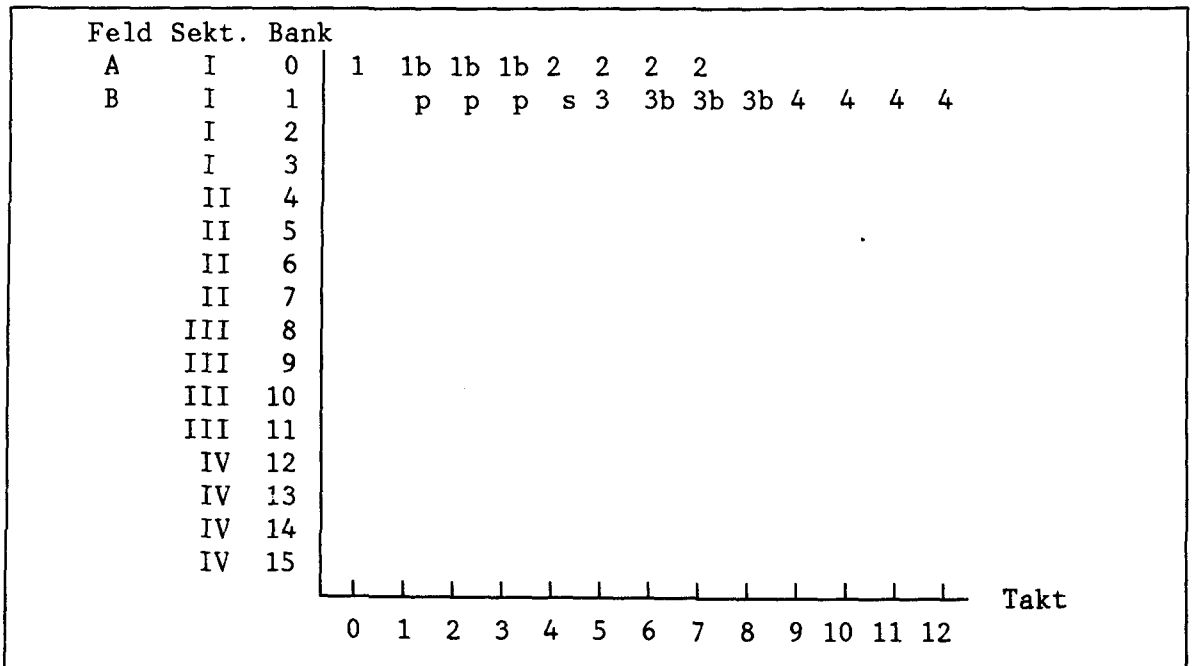


Abb. 27. Bankbelegung zweier Datenströme auf der CRAY X-MP: Es sind zwei Datenströme mit je zwei Zugriffen (1,2; 3,4) dargestellt. Dabei treten neben Bank- und Sektionskonflikten auch Portverzögerungen auf.

Das Beispiel (Abb. 27) zeigt schematisch für ein CRAY X-MP-System (aus hier nur 16 Bänken) mit einer Schrittweite, die gleich der Bankanzahl ist, wie sowohl ein Sektionskonflikt (s) als auch eine Portverzögerung (p) bei schon jeweils zwei Zugriffen auftreten. Die Portverzögerung scheint schon bei Schrittweite vier einzusetzen, da im Gegensatz zur CRAY Y-MP die maximale Grenzleistung schon bei Schrittweite zwei erreicht wird (Tab. 6). Der Wert für die Halbleistungslänge ist bei beiden Rechnern vor allem durch das Verhalten der Grenzleistung geprägt.

3.5.6 Zusammengefaßte Ergebnisse der Einzelprozessormessungen

Für die Messungen mit Schrittweite eins kann zusammenfassend festgestellt werden, daß sich die CRAY Y-MP und die CRAY X-MP beim Speicherzugriff bei keinem der gemessenen Kerne stark unterscheiden. Die Betrachtung der beiden Kenngrößen zeigt Abweichungen für die Grenzleistung, die taktbezogen zehn Prozent nicht überschreiten, meist sogar deutlich darunterliegen. Der Vorteil in der Grenzleistung ist dabei nicht auf einen der beiden Rechner fixiert, sondern ist spezifisch für den betrachteten Kern im Zusammenhang mit dem Compiler. Die Prozessorzykluszeit der CRAY Y-MP ist gegenüber der CRAY X-MP um den Faktor 1.42 kürzer, die Bankzykluszeit jedoch nur um den Faktor 1.13. Dieser Nachteil wird jedoch durch die verbesserte Struktur des Speichers und des Verbindungsnetzwerks bei der CRAY Y-MP in den meisten Fällen fast vollständig ausgeglichen oder sogar überkompensiert. Hierbei ist der Logarithmus ausgenommen, über dessen Aufbau (Funktionsaufruf) an dieser Stelle auch keine genaueren Untersuchungen gemacht worden sind.

Der deutlichen Verbesserung der Grenzleistung bei der Y-MP steht aber eine Verschlechterung der Halbleistungslänge $n_{1/2}$ von 1% bis 30% gegenüber, die zum Teil für die längeren Start-up-Zeiten des Speichers, zum anderen Teil für längere Ausführungszeiten einzelner Befehle bezahlt werden müssen.

Die Variation der Schrittweite ergibt noch deutlichere Ergebnisse. Betrachtet man die Grenzleistung r_∞, so fallen bei ungeschickter Wahl der Startbänke die Nachteile der Untersektionsstufe und der längeren Bankzykluszeit für die Y-MP voll ins Gewicht, so daß die Grenzleistung in MFLOPS bei Schrittweite 16 für die CRAY X-MP sogar deutlich besser ist als für die CRAY Y-MP (78%). Auf der Y-MP kann man jedoch durch geschickte Anordnung der Startbänke der Felder z.B. für die Schrittweite 32 einen Gewinn von bis zu 200% erzielen, was für die X-MP wegen der Portverzögerungen nicht gelingt. Die Halbleistungslänge $n_{1/2}$ ist bei der Variation der Schrittweite weniger aussagekräftig, da sie stark mit der Grenzleistung korreliert ist.

3.6 Mehrprozessormessungen

Die bisherigen Messungen betrachten das Verhalten des Hauptspeichers der CRAY-Rechner aus der Sicht eines Prozessors. Sie zeigen also nur die Last dieses Prozessors auf dem Hauptspeicher auf. Vergleiche des Hauptspeicherverhaltens bei Mehrprozessormessungen sind bereits in [FaSc,89] und [Dete,89] beschrieben worden. Erhöht man die Anzahl der benutzten Prozessoren, zeigt sich ein Abfallen der Leistung in MFLOPS für große Vektorlängen und damit das r_∞. Dieses Abfallen ist bei der CRAY Y-MP deutlich geringer als bei der CRAY X-MP. Gründe dafür sind

1. die erhöhte Anzahl der Bänke pro CPU bei der CRAY Y-MP (doppelt soviele wie bei der CRAY X-MP),
2. die dreistufige Hierarchie im Speicherzugriff und in der Konfliktauflösung,
3. die veränderte Strategie der Portbehandlung bei Bankkonflikten.

Neben dem Verhalten der reinen Grenzleistung r_∞ soll hier auch das Verhalten der Halbleistungslänge $n_{1/2}$ in Abhängigkeit von der Prozessoranzahl untersucht werden.

3.6.1 Das Meßverfahren

Die Mehrprozessormessungen sind im Gegensatz zu den Einprozessormessungen unter dem Betriebssystem UNICOS durchgeführt worden, da die zu gewinnenden Aussagen für die Leistungsfähigkeit des Multiprocessing und des Multitasking, d.h. der Parallelverarbeitung, auf den CRAY-Systemen unter dem zukunftsorientierten Betriebssystem UNICOS essentiell sind.

Der Übergang von einem auf mehrere Prozessoren wird dabei durch Aufrufen des gleichen Programms in verschiedenen Betriebssystem-Prozessen erreicht, die auf unterschiedliche Prozessoren verteilt sind. Durch das wiederholte Ausführen des Kerns in einer Programmschleife wird dabei für ein sinnvolles Verhältnis zwischen den Ausführungszeiten im Betriebssystem und im gemessenen Programm gesorgt.

Zur Begrenzung des Zeitaufwandes der Messungen ist nicht für jede Vektorlänge zwischen 1 und 1024 die Ausführungszeit des Kerns gemessen worden. Statt dessen ist der Bereich von 1 bis 1024 in 16 Abschnitte zu 64 Vektorlängen aufgeteilt worden. Aus jedem Abschnitt sind jeweils fünf Werte als Vektorlänge einer Messung verwendet worden ($n_{ki} = 64 \cdot k + i$ mit $0 \leq k \leq 15$ und $i \in \{2, 17, 33, 49, 64\}$). Insgesamt sind also 80 verschiedene Vektorlängen als Grundlage der Berechnung von $n_{1/2}$ und r_∞ untersucht worden. Dieses Vorgehen berücksichtigt die Länge 64 der Vektorregister bei den CRAY-Rechnern. Die Messungen sind aber selbst bei den Meßergebnissen mit nur einem Prozessor nicht direkt mit den Einzelprozessormessungen aus Abschnitt 3.5 vergleichbar.

3.6.2 Die Meßergebnisse

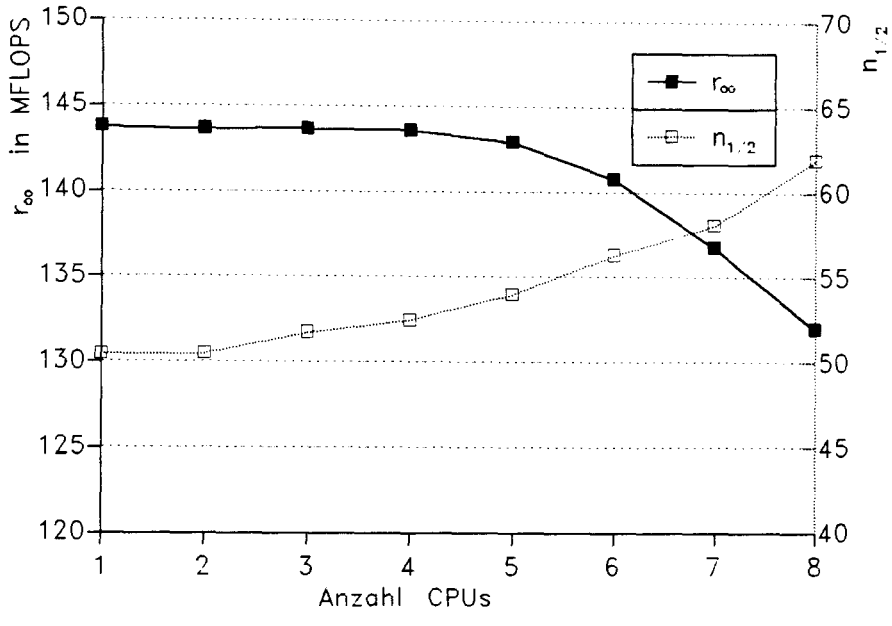


Abb. 28. **r_∞ und $n_{1/2}$ bei Variation der CPU-Anzahl auf der CRAY Y-MP:** Der untersuchte Kern ist die Addition. Bei steigender CPU-Anzahl fällt die Grenzleistung und steigt die Halbleistungslänge.

Abb. 28 zeigt die Meßergebnisse für r_∞ und $n_{1/2}$.

r_∞: Für die Grenzleistung bestätigen sich die Ergebnisse, die schon in [Dete,89] vorgestellt werden, d.h. mit steigender Anzahl benutzter CPUs fällt die Grenzleistung ab und vermindert sich bei acht CPUs um 8.9% gegenüber der Grenzleistung bei Benutzung einer CPU. Die Ursache dafür sind die Bankkonflikte, die erst bei der Benutzung mehrerer CPUs auftreten. Diese Bankkonflikte werden wahrscheinlicher, je mehr CPUs gleichzeitig auf den Speicher zugreifen.

$n_{1/2}$: In [FaSc,89] und [Dete,89] wird keine Betrachtung der Halbleistungslänge vorgenommen. Diese steigt mit der Anzahl der CPUs um maximal 22,8% bei acht CPUs gegenüber dem Wert bei Benutzung einer CPU. Dies verwundert deshalb, weil bei der Verknappung der benutzbaren Bänke durch Variation der Schrittweite (vgl. Abschnitt 3.5.4) die Halbleistungslänge mit der sinkenden Grenzleistung fällt, was direkt durch den Einfluß von r_∞ auf Beziehung (5) in Abschnitt 3.4.1 zu erklären ist. Hier muß also ein anderer Effekt die fallende Tendenz von $n_{1/2}$ ins Gegenteil umwandeln, d.h. eine Erhöhung der gemessenen Zeiten um einen für die jeweilige Anzahl der CPUs spezifischen Wert erzeugen, der von der Vektorlänge unabhängig ist.

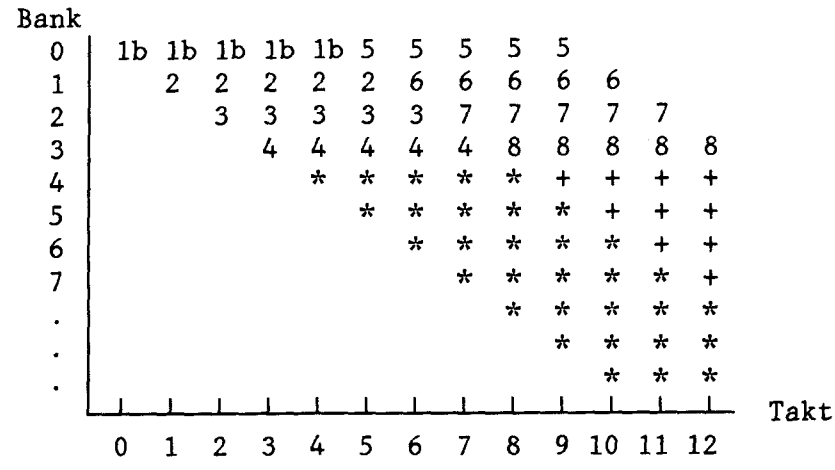


Abb. 29. Bankbelegung zweier Datenströme auf der CRAY Y-MP: Es sind zwei Datenströme mit mehreren Zugriffen (1,2,3,4,*; 5,6,7,8,+) dargestellt.

Eine Erklärung dieses Effekts ist eine Synchronisation der Datenströme der verschiedenen Prozessoren an den Startbänken, wie sie bei [Oed,85],[ChSm,86] und [CaEl,85] anhand von Simulationen aufgezeigt wird. Abb. 29 zeigt zwei Datenströme, die sich an der Startbank 0 synchronisieren. Zugriff 5 muß dabei eine Bankzykluszeit (hier 5 Takte) warten. Dann laufen die Zugriffe 6,7,8 des zweiten Datenstroms jeweils unbehindert nach den Zugriffen 2,3,4 des anderen Datenstroms ab. Es bilden sich zwei unbehinderte diagonale „Zugriffsstreifen" durch die beiden Datenströme, wie sie durch die weiteren Zugriffe mit * und + angedeutet werden. Treffen allgemein auf einer Bank die Zugriffswünsche mehrerer Datenströme mit Schrittweite eins aufeinander, so werden diese Datenströme jeweils um eine Bankzykluszeit verzögert aktiv, bis alle Konflikte aufgelöst worden sind. Die Verzögerung der Zugriffe der verschiedenen Datenströme führt dazu, daß danach alle Datenströme auf freie Bänke zugreifen, da insgesamt ausreichend Bänke vorhanden sind. Es ergibt sich also eine einmalige Verzögerung (Start-Up-Zeit) der Datenzugriffe, die nach der Synchronisation theoretisch nicht mehr auftritt.

Der absolute Wert für $n_{1/2}$ liegt deutlich niedriger als bei den Messungen mit variabler Schrittweite in Abschnitt 3.5.4 , obwohl der gleiche FORTRAN-Code für die Kernschleife verwendet wird. Hier zeigt sich der Einfluß der Optimierung durch den Compiler bei der Verwendung mehrfach geschachtelter Schleifen. Es kann dabei nämlich ein invarianter Initialisierungsteil (Arithmetik zur Schrittweitenberechnung) aus der innersten Schleife vor die äußerste Schleife gezogen werden, wodurch der Einfluß dieses Teils auf die Bestimmung von $n_{1/2}$ deutlich vermindert wird.

Der absolute Wert für $n_{1/2}$ ist also stark vom benutzten Meßprogramm abhängig. Unabhängig davon ist die steigende Tendenz von $n_{1/2}$ bei wachsender Anzahl benutzter CPUs als Folge der verstärkten Speicherzugriffe bei der CRAY Y-MP deutlich erkennbar.

4.0 Untersuchung von Algorithmen

Laufzeitmessungen von Programmkernen in Kapitel 3 sind ein geeignetes Hilfsmittel, um gezielt einzelne Aspekte der Speicherarchitektur von CRAY-Rechnern zu analysieren. Bei der Untersuchung von Algorithmen überlagern sich die Auswirkungen mehrerer Systemeigenschaften in der Ausführungszeit. Dies ist gerade das Verhalten der Systeme, das die Sicht des normalen Anwenders bestimmt. Der Wunsch, vor allem Eigenschaften der Vektorverarbeitung und regelmäßige Zugriffe auf den Hauptspeicher zu untersuchen, legt eine Verwendung von Matrixalgorithmen nahe. Hier sollen deshalb die Matrixmultiplikation und der Warshall-Floyd-Algorithmus (zur Berechnung aller kürzesten Wege in einem Graphen [Deo,74]) untersucht werden. Alle in diesem Kapitel vorgestellten Messungen wurden unter dem Betriebssystem UNICOS durchgeführt.

4.1 Matrixmultiplikation

4.1.1 Algorithmus und Programmierung

Die Matrixmultiplikation ist einer der grundlegenden Algorithmen der linearen Algebra. Das Produkt $C = (c_{ij}) \in \mathbb{R}^{m \times p}$ zweier Rechtecksmatrizen $A = (a_{ik}) \in \mathbb{R}^{m \times n}$ und $B = (b_{kj}) \in \mathbb{R}^{n \times p}$ wird dabei durch den folgenden Zusammenhang beschrieben:

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}.$$

Bei der Umsetzung des Algorithmus in eine Programmiersprache entstehen sechs Varianten dadurch, daß die Anordnung der Schleifen für die Indizes i, j und k verändert wird ([DoGu,84],[Orte,88]). Die Bezeichnung der Varianten erfolgt durch das jeweilige Indextripel. In der *kji* Variante wird z.B. in der äußersten Schleife der Index k und in der innersten Schleife der Index i verändert. Die sechs Varianten werden im folgenden in einem PASCAL-ähnlichen Pseudo-Code vorgestellt.

```
for i=1 to m
  for j=1 to p
    cij = 0
    for k=1 to n
      cij = cij + aik*bkj
    end
  end
end
```

Abb. 30. Form ijk

```
for j=1 to p
  for i=1 to m
    cij = 0
    for k=1 to n
      cij = cij + aik*bkj
    end
  end
end
```

Form jik.

Benutzt man die Level 1 BLAS Notation [LaHa,79], so kann man als Kern der *ijk*- und *jik*-Versionen die SDOT-Operation (Skalarprodukt) identifizieren (Abb. 30). Ein Element der Produktmatrix wird berechnet, indem man einen Zeilenvektor der Matrix des ersten Faktors mit einem Spaltenvektor des zweiten Faktors skalar multipliziert.

```
for i=1 to m
  for j=1 to p
    c_ij = 0
  end
end
for k=1 to n
  for i=1 to m
    for j=1 to p
      c_ij = c_ij + a_ik*b_kj
    end
  end
end
```

Abb. 31. Form kij

```
for i=1 to m
  for j=1 to p
    c_ij = 0
  end
  for k=1 to n
    for j=1 to p
      c_ij = c_ij + a_ik*b_kj
    end
  end
end
```

Form ikj

Die in Abb. 31 dargestellten *kij*- und *ikj*-Versionen enthalten im Kern SAXPY-Operationen. Die Zeilenvektoren der Produktmatrix werden als eine Linearkombination von Zeilenvektoren des zweiten Faktors berechnet.

```
for j=1 to p
  for i=1 to m
    c_ij = 0
  end
end
for k=1 to n
  for j=1 to p
    for i=1 to m
      c_ij = c_ij + a_ik*b_kj
    end
  end
end
```

Abb. 32. Form kji

```
for j=1 to p
  for i=1 to m
    c_ij = 0
  end
  for k=1 to n
    for i=1 to m
      c_ij = c_ij + a_ik*b_kj
    end
  end
end
```

Form jki.

Die innerste Schleife der *kji*- und *jki*-Versionen stellt ebenfalls SAXPY-Operationen dar. Hier wird aber auf Spaltenvektoren der Produktmatrix eine Linearkombination von Spaltenvektoren des ersten Faktors akkumuliert (Abb. 32).

Diese sechs Versionen wurden für quadratische Matrizen in FORTRAN codiert und die zugehörigen Laufzeiten auf der CRAY X-MP und der CRAY Y-MP gemessen. Abb. 33 zeigt als Beispiel die Implementierung der *kji*-Version. Hierbei ist hervorzuheben, daß die FORTRAN-Matrizen stets exakt dimensioniert worden sind. Matrizen werden in FORTRAN spaltenweise im Speicher abgelegt, d.h. Komponenten eines Spaltenvektors belegen im Speicher benachbarte Plätze. Dies bedeutet, daß bei zeilen-

weisem Zugriff auf die Matrizen der Speicher mit einer Schrittweite von n durchlaufen wird, wobei n die Ordnung der Matrix darstellt.

```
      PARAMETER (N=772)
      REAL    A(N,N), B(N,N), C(N,N)
      DO 10 J=1,N
         DO 10 I=1,N
            C(I,J) = 0.0
10       CONTINUE
      DO 20 K=1,N
         DO 20 J=1,N
            DO 20 I=1,N
               C(I,J) = C(I,J) + A(I,K)*B(K,J)
20    CONTINUE
```

Abb. 33. FORTRAN-Code der kji-Version.

Für die Laufzeitmessungen sind die voll besetzten Matrizen A und B mit den Werten aus dem Beispiel 3.12 in [GrKa,69] initialisiert worden.

4.1.2 Einzelprozessormessungen

Die Bezeichnung Ausführungszeit wird im folgenden als Synonym für die CPU-Zeit verwendet. In Abb. 34 sind die CPU-Zeiten der verschiedenen Varianten der Matrixmultiplikation auf einem Prozessor von CRAY X-MP und CRAY Y-MP gegenübergestellt. Die betrachtete Problemgröße ist $n = 772$.

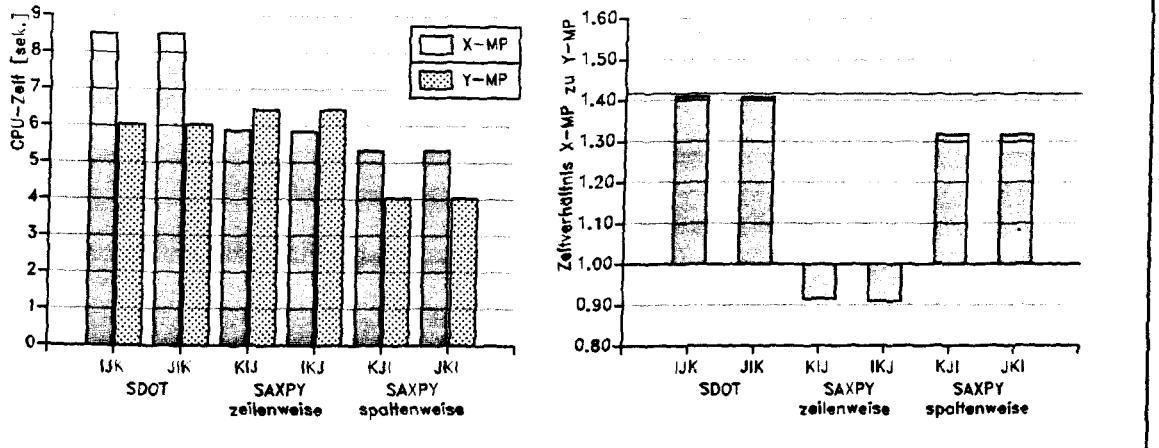


Abb. 34. CPU-Zeit und Zeitverhältnis der Matrixmultiplikation bei Einprozessormessung für n = 772: Die Referenzlinie im Zeitverhältnis ist beim Verhältnis 1.42 der Prozessorzykluszeiten eingetragen.

Es zeigt sich, daß die Anordnung der beiden äußeren Schleifen keinen Einfluß auf die Ausführungszeit hat. Deshalb werden im folgenden nur noch die Varianten IJK, KIJ

und KJI diskutiert. Die kürzesten Ausführungszeiten werden bei beiden Rechnern für die SAXPY-Variante mit spaltenweisem Zugriff (KJI) erreicht. Bei der CRAY X-MP ergeben sich die schlechtesten Ausführungszeiten für die SDOT-Version (IJK). Dies ist auf den schon bei der Einzelprozessormessung an Kernen festgestellten hohen Aufwand für die Vektorreduktion von 64 Elementen auf ein Element zurückzuführen (vgl. Abschnitt 3.5.3.2).

Auf der CRAY Y-MP schneidet die SAXPY-Version mit Zugriff auf Zeilenvektoren (KIJ) am schlechtesten ab. Der im Vergleich zu der KJI-Version hohe CPU-Zeitverbrauch wird von der Wahl der Matrixordnung $n = 772$ beeinflußt. Da in diesem Fall die Schrittweite der Vektorzugriffe des Programms 772 ist, ergibt sich in bezug auf die Speicherbänke eine Schrittweite von vier als 772 modulo 256 bei der CRAY Y-MP bzw. 772 modulo 64 bei der CRAY X-MP. Da vier ein Teiler der Bankanzahl ist, steht dem Programm nur ein Viertel der Bänke zur Verfügung, was zu einer erhöhten Anzahl von Speicherkonflikten führt. Bei der Y-MP wirkt sich dieser Effekt durch die Untersektionsstruktur und die dadurch aus der Sicht einer CPU zur Verfügung stehenden 32 „Bänke" gegenüber den 64 Bänken der X-MP stärker aus. Ferner bewirkt eine Schrittweite von vier auf der Y-MP Sektionskonflikte, wenn über mehrere Ports gleichzeitig auf den Speicher zugegriffen wird. Bei der X-MP wird bei dieser Schrittweite jeweils eine andere Sektion referiert (vgl. Abb. 6). Aus diesen Gründen wird die Ausführungszeit der KIJ-Version auf der Y-MP höher als bei der X-MP.

Das Verhältnis der Ausführungszeiten erreicht für den SDOT-Fall ungefähr den Wert des Verhältnisses der Taktzykluszeiten zwischen CRAY X-MP und CRAY Y-MP und liegt in allen anderen Fällen darunter.

4.1.3 Optimierung der Speichernutzung

Zum Verbessern des Zugriffsverhaltens der verschiedenen Versionen der Matrixmultiplikation sind hier zwei Ansätze untersucht worden:

1. Beeinflussen der Lage der Felder im Speicher,
2. Verminderung der Zugriffe auf den Speicher.

In Abschnitt 3.5.5 hat sich am Beispiel der Vektoraddition gezeigt, daß man bei einer Schrittweite von vier durch eine um jeweils eine Bank verschobene Festlegung der Startbänke von Feldern ähnliche oder sogar bessere Ausführungszeiten erreichen kann als bei Zugriffen mit Schrittweite eins. Diese Aussage gilt sowohl für die CRAY X-MP als auch für die CRAY Y-MP. Deshalb ist hier die SAXPY-Version mit zeilenweisem Zugriff auf diesen Effekt hin untersucht worden. Die gewünschte Anordnung der Startbänke zueinander erreicht man dabei durch Zusammenfassen der jeweiligen Felder in einen COMMON-Block und das Verwenden von zusätzlichen Feldern, die die Verschiebung bewirken (Abb. 35). Der Wert 241 ergibt sich dabei als $256 + 1 - (772^2 \text{ modulo } 256)$ und bewirkt die gewünschte Bankverschiebung um eins sowohl bei 64 als auch bei 256 Bänken.

```
      PARAMETER (N=772, ISHIFT=241)
      REAL    A(N,N), B(N,N), C(N,N)
      REAL    SHIFT1(ISHIFT), SHIFT2(ISHIFT)
      COMMON /FELDER/ A, SHIFT1, B, SHIFT2, C
```

Abb. 35. FORTRAN-Code zur Verschiebung der Startbänke

Eine Verbesserung der Ausführungszeiten stellt sich hierbei jedoch weder bei Einprozessor- noch bei Mehrprozessormessungen ein. Bei der Addition in Abschnitt 3.5.5 erfolgen drei parallele Vektorzugriffe auf drei durch die Bankverschiebung disjunkte Teilmengen der Bänke. Der Code der KIJ-Version der Matrixmultiplikation führt dagegen parallel zwei Vektorzugriffe auf das gleiche Feld C und nur einen Vektorzugriff auf eine disjunkte Teilmenge der Bänke bei Feld B durch. Bei dieser Konstellation ist es offensichtlich nicht mehr möglich, die Nachteile der Einschränkung der effektiven Bankanzahl durch die Verschiebung der Startbänke zu vermindern.

Eine Möglichkeit der Verminderung der Zugriffe auf den Hauptspeicher stellt das sogenannte **Unrolling** dar [DoEi,84]. Dabei wird die Zahl der Wiederholungen einer Schleife durch Ausprogrammieren mehrerer Durchläufe der Anweisungen in der innersten Schleife reduziert. Lassen sich dabei mehrere Zuweisungen zu einer zusammenfassen, spart man häufig Speicherzugriffe durch die Benutzung der Vektorregister als Zwischenspeicher.

```
      DO 10 J = 1,N
        DO 10 I = 1,N
        C(I,J) = 0.0
10    CONTINUE
      DO 20  K = 1,N,4
        DO 20  J = 1,N
          DO 20  I=1,N
            C(I,J) = C(I,J) + A(I,K)  *B(K,J)   + A(I,K+1)*B(K+1,J)
     *                      + A(I,K+2)*B(K+2,J) + A(I,K+3)*B(K+3,J)
20    CONTINUE
```

Abb. 36. Unrolling der KJI-Variante der Matrixmultiplikation

Abb. 36 zeigt als Beispiel ein Unrolling der KJI-Variante. Dabei wird für den Wert von N vorausgesetzt, daß er ein Vielfaches von vier ist, da hier gerade vier Wiederholungen der innersten Schleife zusammengefaßt werden. Das Unrolling betrifft in allen sechs Varianten der Matrixmultiplikation den Index K und führt in der jeweiligen innersten Schleife zu der gleichen Zuweisung. Für die Matrixmultiplikation mit Unrolling zeigt sich ebenfalls, daß die Anordnung der beiden äußeren Schleifen sich nicht auf die Ausführungszeiten auswirkt (Abb. 37).

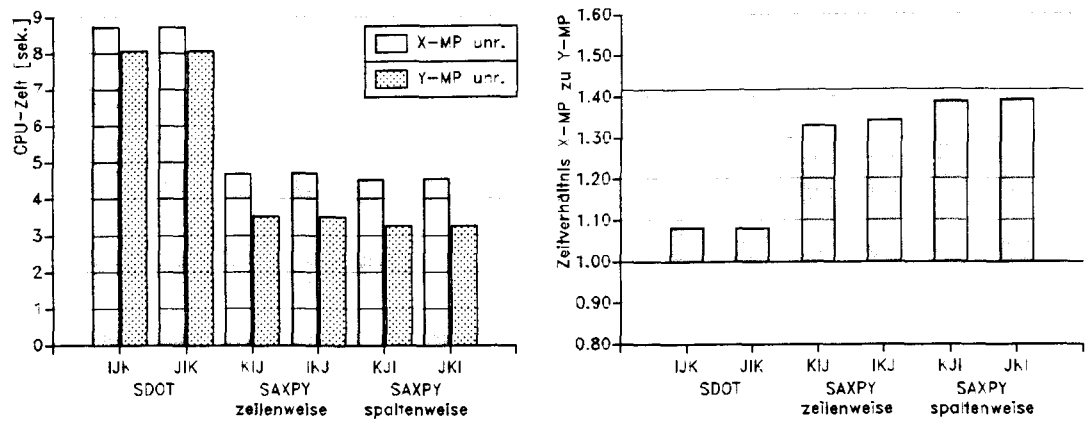


Abb. 37. CPU-Zeit und Zeitverhältnis der Matrixmultiplikation mit Unrolling bei Einprozessormessung für n = 772: Die Referenzlinie im Zeitverhältnis ist beim Verhältnis 1.42 der Prozessorzykluszeiten eingetragen.

Bei SAXPY-Versionen führt der zeilenweise Zugriff auf die Matrizen hier nicht zu einer wesentlich schlechteren Laufzeit. Dies bedeutet auf der CRAY Y-MP für die KIJ-Variante nahezu eine Halbierung der Laufzeit durch das Unrolling. Die Verbesserung von ca. drei Sekunden läßt sich durch eine Verminderung der Untersektionskonflikte erklären. Ohne Unrolling erfolgen gleichzeitig drei Vektorzugriffe mit Schrittweite vier (Drei-Port-Code). Es stehen dadurch nur acht Untersektionen zur Verfügung. In Verbindung mit der Bankzykluszeit von fünf Takten ergibt sich im Mittel eine Zugriffszeit von 15/8 Takten, was eine Verzögerung pro Element von 7/8 Takten bzw. n^3*0.875 Takte bzw. 2.4 Sekunden hervorruft.

Für die SAXPY-Versionen ergeben sich durch Unrolling deutliche Verbesserungen der Ausführungszeiten gegenüber der schnellsten Version ohne Unrolling; sie liegen zwischen 12% und 15% bei der CRAY X-MP und zwischen 13% und 19% bei der CRAY Y-MP.

Die Verminderung der Speichereinflüsse hat beim Verhältnis der Ausführungszeiten zwischen den beiden Rechnern zur Folge, daß praktisch das Verhältnis 1.42 der Prozessorzykluszeiten erreicht wird. Dies gilt nicht für die SDOT-Varianten, bei denen die Ausführungszeiten der CRAY Y-MP fast die der CRAY X-MP erreichen und für beide Rechner höher sind als bei der jeweiligen Version ohne Unrolling.

Zusammenwirken mehrerer Eigenschaften am Beispiel der SDOT-Versionen: Für den IJK-Fall bedeutet Unrolling nach dem Index K, daß die Anzahl der Wiederholungen der innersten Schleife vermindert wird. Damit führt das Unrolling hier zu einer Verminderung der Vektorlänge auf ein Viertel. Man erhält außerdem die Überlagerung zweier Schrittweiten beim Bankzugriff:

1. die Schrittweite von vier beim zeilenweisen Zugriff auf die Matrix, die implizit durch die Matrixordnung 772 bewirkt wird,
2. die explizite Schrittweite von vier, die bei der innersten Schleife angegeben ist.

Dies ergibt Schrittweiten von vier bei spaltenweisem und 16 bei zeilenweisem Zugriff. Bei der CRAY Y-MP bewirkt eine Schrittweite von 16 beim Zugriff auf die Speicherbänke, daß bei jedem zweiten Zugriff mindestens drei Takte Verzögerung beim Laden über diesen Port auftreten, d.h. im Mittel 1.5 Takte Verzögerung. Da n^3 dieser Zugriffe durchgeführt werden, ist so eine Erhöhung der Ausführungszeit um 1.5 * n^3 Takte, also ca. 4 Sekunden, zu erklären.

FORTRAN-Kern

```
            C(I,J) = C(I,J) + A(I,K)  *B(K,J)   + A(I,K+1)*B(K+1,J)
     *                      + A(I,K+2)*B(K+2,J) + A(I,K+3)*B(K+3,J)
```

Assembler-artige Notation der zugehörigen Ladebefehle

Die Wahl der Schrittweite von 16 dient nur dem Verständnis aus Sicht der Bankzugriffe. Genau genommen müßte hier ein Wert von 772*4 stehen.

```
                                          ┌──── Vektorlänge
                                          │  ┌─ Schrittweite
                                          │  │
(A1)     LOADVEK  A(1,K  ),64,16      --> Port A
(B1)     LOADVEK  B(K,1  ),64,4       --> Port B
                  ...
(A2)     LOADVEK  A(1,K+1),64,16      --> Port B
(B2)     LOADVEK  B(K+1,1),64,4       --> Port A
                  ...
(A3)     LOADVEK  A(1,K+2),64,16      --> Port A
(B3)     LOADVEK  B(K+2,1),64,4       --> Port B
                  ...
(A4)     LOADVEK  A(1,K+3),64,16      --> Port B
(B4)     LOADVEK  B(K+3,1),64,4       --> Port A
```

Zeitverhalten

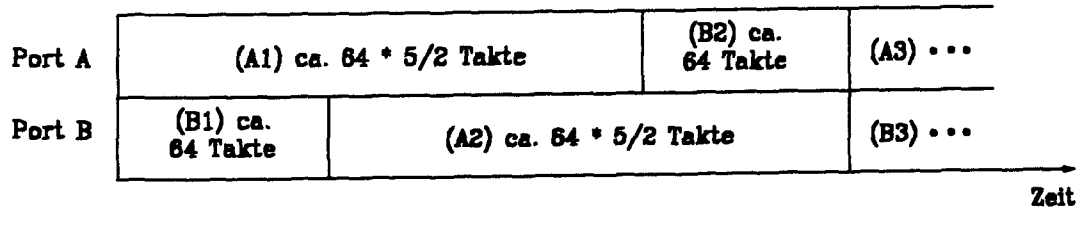


Abb. 38. FORTRAN Code und Pseudo-Assembler beim Unrolling der JIK Version der Matrixmultiplikation

Um die gemessene weit niedrigere Verzögerungzeit zu erklären, muß man die Eigenschaften der beiden Ladeports berücksichtigen. Die Zuordnung zwischen Vektorbefehlen und Ports erfolgt nämlich dynamisch zur Laufzeit des Programms (vgl. Abschnitt 2.4.2). Aus dem Unrolling resultieren vier Zugriffspaare aus je einem zeilenweisen Zugriff auf Feld A und einem spaltenweisen Zugriff auf Feld B. Da der Zugriff auf einen Zeilenvektor deutlich länger als der Zugriff auf einen Spaltenvektor dauert, wird der

zeilenweise Zugriff nicht auf dem gleichen Port durchgeführt, sondern die Zuordnung zwischen Feldzugriff und Port alterniert. Dadurch laufen zwei Referenzen auf Zeilenvektoren parallel. Eine gegenseitige Behinderung tritt aber nicht auf, weil die vier Ladebefehle für die Matrix A jeweils auf eine andere Bankmenge zugreifen, was durch die Indizes K, K + 1, K + 2, K + 3 bewirkt wird. Aus Abb. 38 ergibt sich damit pro Datenelementpaar aus Feld A und B eine durchschnittliche Ladezeit von 7/4 Takten, d.h. eine Verzögerung gegenüber dem konfliktfreien Zugriff von 3/4 Takten. Bezieht man den Berechnungsaufwand von n^3 und die Taktzykluszeit der CRAY Y-MP ein, so ergibt sich eine Zeitverzögerung von ca. 2 Sekunden, was der beobachteten Verzögerung durch das Unrolling sehr nahe kommt. Die Zeitverzögerung bei der CRAY X-MP fällt dagegen sehr viel geringer aus. Dies ist auf den kürzeren Bankzyklus und auf die doppelte Anzahl von Bänken zurückzuführen, wie sie aus Sicht einer CPU hier zur Verfügung stehen.

4.1.4 Mehrprozessormessungen

Nachdem die Interpretation der Einprozessormessungen einen Eindruck davon vermittelt hat, welche Einflüsse Änderungen am Programm auf die zu erwartende Leistung haben, sollen die Mehrprozessormessungen die Einflüsse auf das Gesamtsystem untersuchen. Die Mehrprozessormessungen an Matrixalgorithmen sind analog zu den Mehrprozessormessungen an Kernen als Multiprocessing-Messungen durchgeführt worden. Es wird also das gleiche Programm auf mehreren CPUs gestartet und dadurch eine Beurteilung für das Lastverhalten des Speichers in Abhängigkeit von

- dem gewählten Rechner,
- der Anzahl der aktiven CPUs,
- der gewählten Programmvariante

gewonnen. Durch dieses Vorgehen ist nicht gewährleistet, daß Programmteile mit intensivem Speicherzugriff immer gegeneinander laufen. Vielmehr verschieben sie sich oft derart, daß sie parallel zu Programmteilen laufen, die den Speicher weniger intensiv nutzen. Dadurch treten die Speicherlasteffekte weniger deutlich auf als bei expliziter Synchronisation von Programmteilen mit intensivem Speicherzugriff. Die Untersuchungen beschränken sich hier auf drei Varianten ohne Unrolling und die KJI-Variante mit Unrolling, da sie bei den Einprozessormessungen die kürzeste Laufzeit aufweist.

Es wird der Mittelwert der Ausführungszeiten des jeweils betrachteten Programms über die Anzahl der aktiven CPUs gebildet. Als Maß für die Leistungfähigkeit wird in Abb. 39 die Anzahl der Gleitkommaoperationen pro Zeiteinheit verwendet, die hier aus der Ausführungszeit mit Hilfe der Anzahl von $2 \bullet n^3$ Gleitkommaoperationen der Matrixmultiplikation berechnet wird.

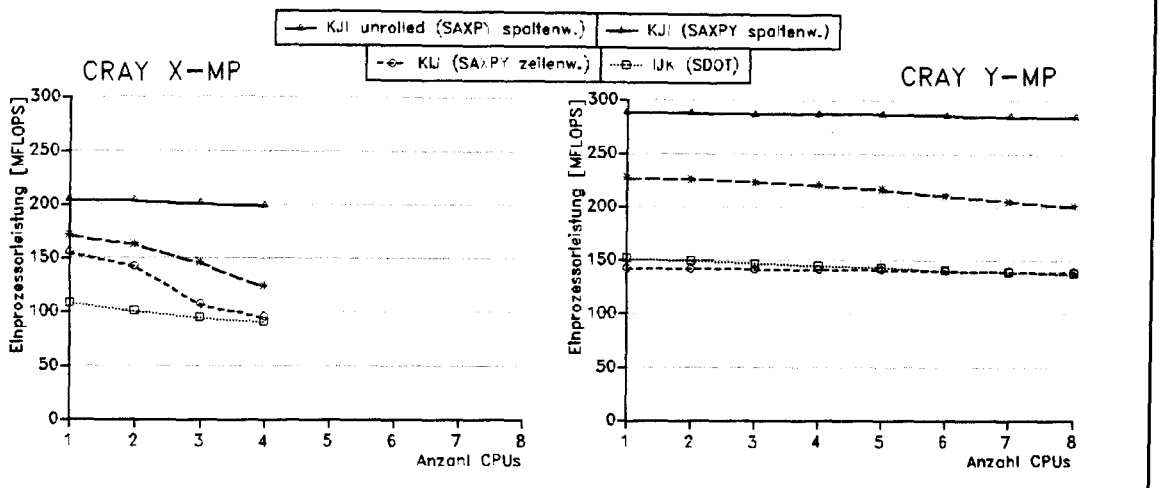


Abb. 39. Einprozessorleistung (in MFLOPS) für die Matrixmultiplikation bei Variation der Prozessoranzahl für n = 772: Es werden die Meßergebnisse von CRAY X-MP und CRAY Y-MP gegenübergestellt.

Die CRAY X-MP zeigt für die vier Varianten im Groben ein Verhalten, wie man es aufgrund der benutzten Schrittweiten und der Intensität des Speicherzugriffs erwarten kann. Die geringste Verminderung der MFLOP-Rate durch das Hinzunehmen weiterer CPUs erfolgt bei der Version mit Unrolling, da die Anzahl und die Schrittweite der gleichzeitigen Zugriffe relativ geringe Anforderungen an das Speichersystem stellen. Hierbei wird auf vier CPUs jeweils mit 199 MFLOPS noch 85% der theoretischen Grenzleistung von 235 MFLOPS erreicht. Die Verminderung der MFLOP-Rate ist bei der SDOT-Variante niedriger als bei den beiden anderen betrachteten Versionen ohne Unrolling. Hier kommt der Aufbau des erzeugten Assemblercodes in

- Vektorberechnungen mit parallelen Vektorladeoperationen (Zwei-Port-Code),
- wenig speicherintensiver Vektorreduktion

zum Tragen (vgl. Abschnitt 3.5.3.2). Auf den verschiedenen Prozessoren laufen also speicherintensive zu wenig speicherintensiven Programmteilen parallel, was die Gesamtbelastung des Speichers relativ niedrig beläßt. Die in allen Aspekten speicherintensivste KIJ-Variante hebt sich bei der Erhöhung der CPU-Anzahl von eins auf vier durch den absolut höchsten Leistungsverlust (60 MFLOPS) von den anderen Varianten ab. Vergleicht man die Leistungsunterschiede, die man durch die Benutzung aller Prozessoren erreicht, so zeigt sich z.B. für die SAXPY-Variante mit Schrittweite vier eine Änderung in der MFLOP-Rate, die fast den Vorteil dieser Variante gegenüber der SDOT-Variante aufhebt, wie er bei der Einprozessormessung in Abb. 39 zu beobachten ist.

Das Mehrprozessorverhalten der CRAY Y-MP ist im ganzen betrachtet so gutmütig, wie man es nach den Erfahrungen aus [FaSc,89] und [Dete,89] erwarten kann. Die Unterschiede in der MFLOP-Rate zwischen Ein- und Achtprozessormessung fallen deutlich weniger ins Gewicht als der Leistungsgewinn durch das Unrolling in der hier betrachteten Form. Die Feinanalyse der Trends erfordert dabei die Berücksichtigung von mehr Randbedingungen als bei der CRAY X-MP. Einfache Regeln wie z.B. „bei Schrittweite vier steigt die Last, also auch die Ausführungszeit über die Anzahl der CPUs

stark" stimmen hier nicht. Ursache dafür ist vor allem die zusätzliche Untersektionsstufe. Treten Verzögerungen bereits an den zu den einzelnen CPUs lokalen Untersektionen auf, wird die Anzahl der gleichzeitigen Speicherzugriffe zwischen den CPUs niedriger und damit das Gesamtsystem entlastet. Ein Indiz dafür ist, daß die Schrittweite vier der KIJ-Version einen geringeren Leistungsverlust bei Variation der CPU-Anzahl bewirkt als eine Schrittweite von eins bei der IJK-Variante ohne Unrolling. Unterschreitet man dagegen eine gewisse Schwelle der Gesamtbelastung z.B. durch Unrolling, so reagiert auch das Gesamtsystem nicht mehr, d.h. die Speicherzugriffskonflikte spielen weder auf Untersektionsebene noch auf Bankebene eine gravierende Rolle. Dadurch wird auf acht CPUs jeweils mit 285 MFLOPS pro CPU noch 86% der theoretischen CPU-Grenzleistung von 333 MFLOPS erreicht. Der ungleichmäßige Aufbau des Codes der SDOT-Variante spiegelt sich hier in einem Leistungverhalten wieder, das zwischen dem der anderen Varianten liegt.

Im Vergleich der MFLOP-Raten zwischen CRAY X-MP und CRAY Y-MP fällt für die KIJ-Variante auf, daß während die Leistung bei der X-MP im Einprozessorbetrieb höher als bei der Y-MP ist, die Erhöhung der CPU-Anzahl beim jeweiligen Maximalwert einen deutlichen Vorteil für die Y-MP bewirkt.

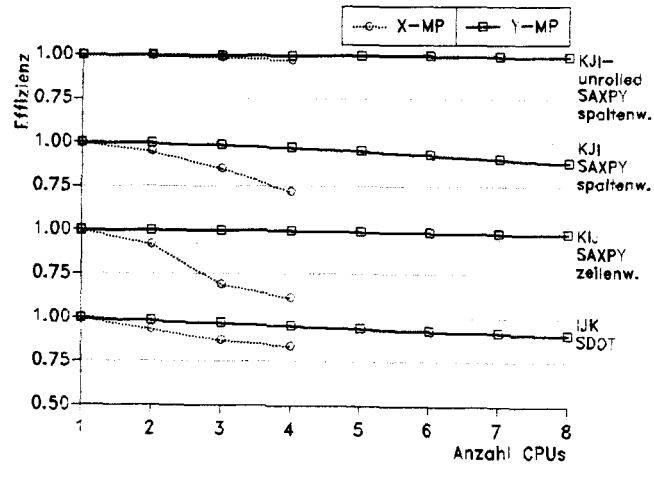


Abb. 40. Effizienz von vier Varianten der Matrixmultiplikation für n = 772

Die **Effizienz** ist hier als Verhältnis der Ausführungszeiten auf einer CPU zu der Ausführungszeit auf mehreren CPUs definiert. Die Effizienz ergibt sich auch als Kehrwert des Quotienten der aus den Ausführungszeiten abgeleiteten MFLOP-Raten. Die Effizienzkurven beschreiben also das Mehrprozessorverhalten in Relation zum Einprozessorverhalten (Abb. 40) und sind bereits in [Dete,89] zum Vergleich einer CRAY X-MP mit einer CRAY Y-MP benutzt worden. Es wird dabei deutlich, daß für alle vier betrachteten Varianten die Effizienzkurve bei der Y-MP erheblich flacher über die Anzahl der CPUs verläuft als bei der X-MP und bei acht CPUs nicht auf den Wert abfällt, der bei der X-MP bereits bei vier CPUs auftritt. Der relative Leistungsverlust erreicht dabei ein Maximum von 11,4% für die spaltenweise SAXPY-Version für die Y-MP gegenüber 38,4% bei der zeilenweisen SAXPY-Version auf der X-MP. (Es sei darauf

aufmerksam gemacht, daß in Abb. 40 zur besseren Darstellung die Ordinate für die einzelnen Varianten jeweils im Nullpunkt versetzt ist.)

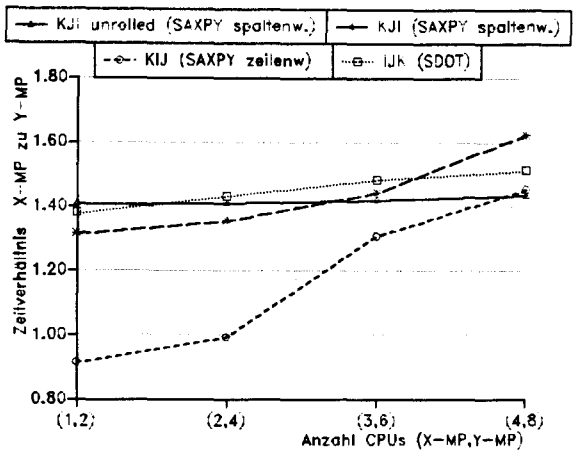


Abb. 41. Zeitverhältnis zwischen CRAY X-MP und CRAY Y-MP für Varianten der Matrixmultiplikation für n = 772: Es wird das Verhältnis der Ausführungszeiten jeweils für die Anzahl der CPUs des jeweiligen Systems betrachtet, die den gleichen Anteil an der Gesamtanzahl der CPUs darstellen.

Zum Vergleich der beiden CRAY-Systeme bei gleichen Programmvarianten, wird in Abb. 41 das Zeitverhältnis zwischen den Ausführungszeiten auf der CRAY X-MP und der CRAY Y-MP dargestellt, was dem Reziprokwert des Verhältnisses der MFLOP-Raten entspricht. Da hier das Lastverhalten des Systems bei Hinzunahme weiterer CPUs interessiert, werden CPU-Anzahlen gegenübergestellt, die 25%, 50%, 75% und 100% der gesamten CPU-Anzahl des betrachteten Systems entsprechen. Es zeigt sich, daß in allen vier Varianten bei voller CPU-Anzahl das Verhältnis der Prozessorzykluszeiten von 1.42 erreicht oder sogar deutlich überschritten wird (1.62 bei der KJI-Version). Aus der Sicht des Gesamtsystems ist die CRAY Y-MP also hier der CRAY X-MP nicht nur technologisch sondern auch in der Architektur, speziell in der Speicherarchitektur, überlegen. Der deutlich unterschiedliche Verlauf der vier Kurven zeigt aber auch auf, daß ein ähnliches Gesamtverhalten auf durchaus verschiedene Einflüsse zurückzuführen ist.

4.2 Der Algorithmus von Warshall-Floyd

4.2.1 Algorithmus und Programmierung

Der Algorithmus von Warshall-Floyd zählt zu der Klasse der Graphenalgorithmen und berechnet die kürzesten Wege zwischen allen Knoten in einem gerichteten Graphen mit Kantenbewertungen ([Deo,74],[Gurk,86]). Die Kantenbewertungen kann man dabei als Distanz der Knoten des Graphen interpretieren. Der Graph wird in Form der bewerteten $(n \times n)$-Adjazenzmatrix $W = (\omega_{ij})$ vorgegeben. Die Idee des Algorithmus ist dabei, die Distanz des direkten Weges zwischen zwei Knoten des Graphen durch eine kürzere Distanz zu ersetzen, die man durch Benutzung von Wegen über Zwischenknoten erhält. Dies wird systematisch durch Untersuchung jedes Knotens als Zwischenknoten zwischen allen relevanten Knotenpaaren erreicht. Dabei wird die jeweils bessere Distanz als neuer Wert in die Matrix W eingetragen. Nach dem Einbeziehen des neuen Zwischenknotens l erhält man so eine neue Matrix $W^{(l)}$, die sich formal beschreiben läßt als

$$\omega_{ij}^{(0)} = \omega_{ij} \quad ,$$

für $l = 1,2,...,n$

$$\omega_{ij}^{(l)} = \min\{\omega_{il}^{(l-1)} + \omega_{lj}^{(l-1)}, \omega_{ij}^{(l-1)}\}$$

Abb. 42 zeigt eine FORTRAN-Version des Warshall-Floyd-Algorithmus.

```
      PARAMETER (N=772)
      INTEGER W(N,N),INF
      DO 10 K = 1,N
         DO 20 J = 1,N
            B = W(K,J)
            IF ((B.LT.INF).AND.(K.NE.J)) THEN
               DO 30 I = 1,N
                  W(I,J) = MIN(W(I,J),W(I,K) + B)
30             CONTINUE
            ENDIF
20       CONTINUE
10    CONTINUE
```

Abb. 42. Warshall-Floyd-Algorithmus in FORTRAN codiert: Es erfolgt ein spaltenweiser Zugriff auf die Matrix durch die innerste DO-Schleife. INF ist dabei ein ausgewählter Wert, der den Wert ∞ repräsentiert.

Genau wie bei den SAXPY-Versionen bei der Matrixmultiplikation unterscheidet man bei der Implementierung des Warshall-Floyd-Algorithmus zwischen zeilenweisem (Index J in der innersten DO-Schleife) und spaltenweisem (Index I in der innersten DO-Schleife) Zugriff auf die Matrixelemente. Die Zeitkomplexität des Algorithmus ist $O(n^3)$. Im Unterschied zur Matrixmultiplikation hängt der Zeitaufwand der Berechnung beim Warshall-Floyd-Algorithmus von den Werten in der Matrix ab (IF-Anweisung). Die

hier für Messungen verwendeten Matrizen werden mit Hilfe des FORTRAN-Zufallszahlengenerators RANF erzeugt [CRAY,88c]. Es handelt sich dabei ausschließlich um ganzzahlige Matrixelemente. Die Berechnungen erfolgen im Gegensatz zur Matrixmultiplikation am Platze, d.h. die Ergebnisse ersetzen sofort die „alten" Werte der Matrix. Untersuchungen des Warshall-Floyd-Algorithmus auf CRAY-Rechnern sind bereits in [HoKn,89] und [Gurk,86] für die CRAY X-MP vorgestellt worden.

4.2.2 Einzelprozessormessungen

In Abb. 43 sind Ausführungszeiten der verschiedenen Versionen des Warshall-Floyd-Algorithmus dargestellt für die Problemgrößen $n = 772$ und $n = 800$. Für diese beiden Fälle ergeben sich als Anzahlen der Durchläufe der innersten Schleife Werte von $0.536 * n^3$ bzw. $0.551 * n^3$.

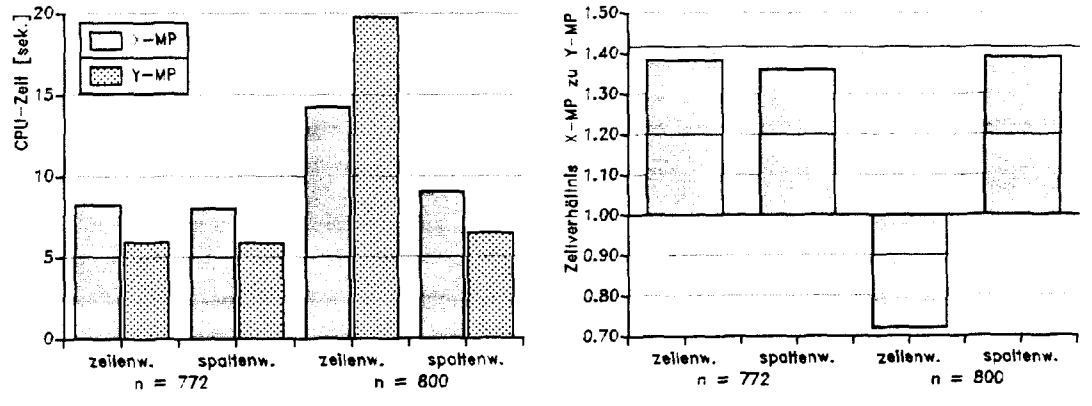


Abb. 43. Ausführungszeiten und Zeitverhältnis beim Warshall-Floyd-Algorithmus auf einem Prozessor

Im Fall $n = 772$ zeigt sich sowohl auf der CRAY X-MP als auch auf der CRAY Y-MP kein signifikanter Unterschied in den Ausführungszeiten der beiden Versionen. Beim Vergleich der beiden Rechner ergibt sich in etwa das durch die unterschiedlichen Prozessorzykluszeiten zu erwartende Verhältnis von 1.42 (zeilenweiser Zugriff: 1.38; spaltenweiser Zugriff 1.36).

Erhebliche Unterschiede dazu sind für den Fall $n = 800$ festzustellen. Während das Verhältnis der Ausführungszeiten auf der CRAY X-MP zwischen zeilenweiser und spaltenweiser Version 1.6 ist, ergibt sich bei der CRAY Y-MP 3.0, so daß sich für den zeilenweisen Zugriff für die Y-MP auch eine erheblich höhere Ausführungszeit ergibt als bei der X-MP. Dies ist durch die für die Speicherbänke effektive Schrittweite von 32 bei der Problemgröße 800 zu erklären. Die Schrittweite 32 tritt bei zeilenweisem Zugriff auf die Matrix W auf und trifft dabei auf der Y-MP erneut auf die 32 Untersektionen, auf der X-MP auf die 64 Speicherbänke (vgl. Abschnitt 3.5.4).

4.2.3 Mehrprozessormessungen

In Abb. 44 ist das Zeitverhalten für die beiden Problemgrößen $n = 772$ und $n = 800$ gegenübergestellt. Hierbei zeigt sich, daß das Mehrprozessorverhalten in beiden Fällen vom Einprozessorverhalten dominiert wird. Ein Grund dafür ist, daß die Speicherzugriffe nur einen Teil der Berechnungszeit beanspruchen, während versetzt dazu Vektorbefehle ohne Speicherzugriffe wie ganzzahlige Addition und Erzeugung einer Bitmaske für die Berechnung des Minimums erfolgen, die nicht durch Chaining überlappen. Deshalb laufen nicht immer Speicherzugriffe mehrerer CPUs gleichzeitig ab.

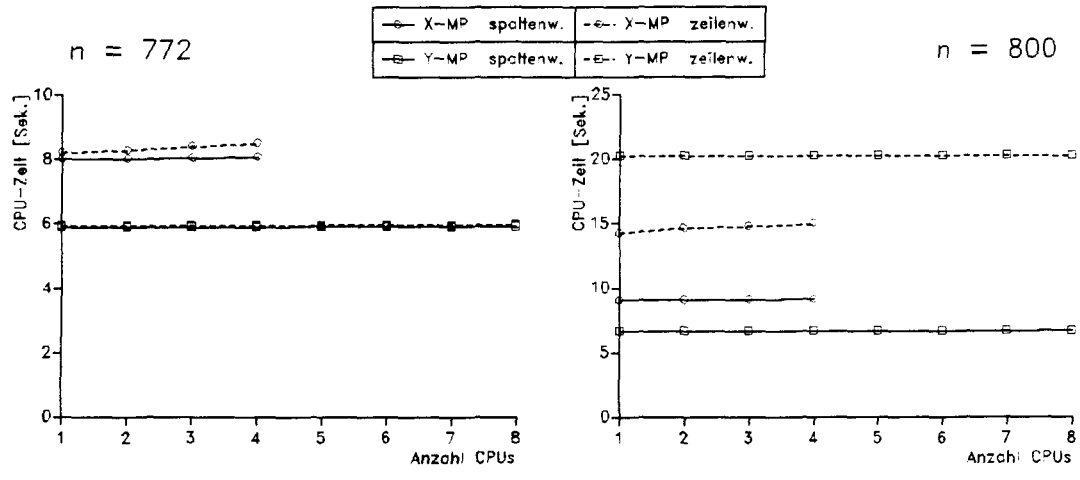


Abb. 44. Ausführungszeiten beim Warshall-Floyd-Algorithmus auf mehreren CPUs

Bei der Problemgröße 772 ist nur bei der CRAY X-MP ein Einfluß der Anzahl der benutzten CPUs auf das Zeitverhalten zu erkennen, wobei besonders der zeilenweise Zugriff zu einer leichten Erhöhung der Ausführungszeit führt, was erneut auf die Wahl der Schrittweite zurückzuführen ist. Bei der CRAY Y-MP ist weder für den zeilenweisen noch für den spaltenweisen Zugriff auf den Speicher ein relevanter gegenseitiger Einfluß der CPUs festzustellen.

Für $n = 800$ treten bereits bei Benutzung nur eines Prozessors auf beiden Rechnern erheblichen Verzögerungen durch Speicherkonflikte auf, wenn zeilenweise auf die Matrix W zugegriffen wird. Die nur geringe Zeitverzögerung bei der Hinzunahme weiterer Prozessoren kann nicht allein durch Überlappung mit wenig speicherintensiven Berechnungen auf den anderen CPUs erklärt werden, da allein bei einem Prozessor auf der CRAY X-MP eine Verzögerung durch zeilenweisen Zugriff von über 30% auftritt, so daß bei Benutzung von vier Prozessoren eine Überlappung nur unvollständig möglich ist. Durch den Multiprocessing-Betrieb ist nicht gewährleistet, daß der zeilenweise Zugriff verschiedener CPUs synchron auf die gleiche Zeile erfolgt. Bezogen auf die Speicherbänke gibt es hier 32 verschiedene Zeilen, die disjunkten Bankmengen entsprechen und somit unbehindert nebeneinander verwendet werden können. Dies bedeutet selbst bei der Benutzung von acht CPUs noch eine deutliche Auswahl an konfliktfreien Zugriffsalternativen. Beim spaltenweisen Zugriff sind die gleichen Argumente wie bei $n = 772$ anzuführen.

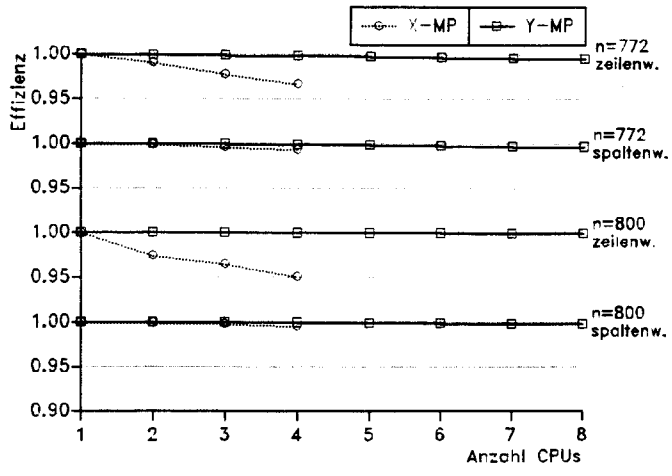


Abb. 45. Effizienz beim Warshall-Floyd-Algorithmus

Die Effizienzdarstellung in Abb. 45 unterstreicht die geringe Lastabhängigkeit bei Hinzunahme mehrerer CPUs für beide Rechner. Gegenüber der Effizienzbetrachtung bei der Matrixmultiplikation (vgl. Abb. 40) ist hier ein deutlich gespreizter im jeweiligen Nullpunkt ebenfalls versetzter Maßstab gewählt worden. Merkliche Effizienzverluste treten nur bei der CRAY X-MP im zeilenweisen Zugriff auf, bleiben aber unterhalb von 5%.

Beim Warshall-Floyd-Algorithmus spielt der Speicherzugriff also eine wesentliche Rolle, wenn man die zeilenweise Version betrachtet. Diese Effekte treten bereits bei der Benutzung einer CPU auf und lassen sich durch den Multiprocessing-Betrieb mit dem gleichen Programm praktisch nicht verstärken. Deshalb wird bei der Wahl einer ungünstigen Schrittweite die Gesamtleistung auf der CRAY Y-MP sogar deutlich schlechter (Zeitverhältnis ca. 0.7) als auf der CRAY X-MP. Dies relativiert den einheitlich positiven Eindruck der CRAY Y-MP als Gesamtsystem, wie er bei der Matrixmultiplikation gewonnen werden konnte.

4.3 Fazit der Messungen an Algorithmen

Bei der Betrachtung des Speicherzugriffs der hier untersuchten Algorithmen haben sich besonders die Parameter

- Schrittweite,
- lokale Speicherintensität,
- Verhältnis von speicherintensiven und wenig speicherintensiven Programmteilen

als bestimmend für das Zeitverhalten herausgestellt.

Die Schrittweite der Speicherzugriffe hat bei der CRAY X-MP und der CRAY Y-MP einen gravierenden Einfluß auf das Laufzeitverhalten. Hier zeigt sich gerade bei Mehrprozessormessungen, daß die CRAY Y-MP die schon in Kapitel 3 festgestellten Vorteile

für Schrittweiten von eins und vier auch bei der Anwendung auf Algorithmen erzielen kann. Der Leistungsgewinn pro Prozessor ist dabei teilweise höher, als die Verbesserung der Prozessorzykluszeit der CRAY Y-MP gegenüber der CRAY X-MP erwarten läßt. Es tritt aber auch bei Algorithmen der schon bei Messungen an Kernen erkannte Effekt zutage, daß für gewisse Kombinationen aus Anzahl der gleichzeitigen Speicherzugriffe und Schrittweite die CRAY Y-MP mit einem Prozessor wie auch als Gesamtsystem absolut schlechtere Ausführungszeiten erreicht als die CRAY X-MP. Es lohnt sich also auf jeden Fall, bei der Wahl der Datenstrukturen und dem Aufbau der inneren Schleifen auf die resultierenden Schrittweiten zu achten.

Unabhängig von der Wahl der Schrittweite ist es erstrebenswert, Speicherzugriffe durch die Verwendung der Vektorregister als Zwischenspeicher zu vermeiden. Die inzwischen übliche Technik des Unrolling hat auch bei der CRAY Y-MP ihre volle Berechtigung und führt zu erheblichen Leistungsgewinnen und zu einer hohen Unabhängigkeit des lokalen Zeitverhaltens auf dem benutzen Prozessor vom Geschehen im Gesamtsystem.

Sowohl für das einzelne Programm als auch für das Gesamtsystem ist es von erheblicher Bedeutung, wie sich der Code bei der Ausführung in prozessorintensive und speicherintensive Bestandteile aufteilt. Dies ist aber für den Anwender nicht immer leicht einsehbar, da er in der Regel nicht über die Umsetzung seines Programms in Maschinenbefehle informiert ist und sich diese auch mit neuen Versionen des Compilers ändern kann. Als Hilfsmittel für das Verständnis der Vorgänge ist hier häufig die Analyse des vom Compiler erzeugten Assembler-Codes und dabei speziell die Anordnung der Vektorbefehle benutzt worden. Diese Analyse gibt Aufschluß über Verzögerungen durch nicht verfügbare Rechnerressourcen und macht somit Verzögerungen auf Taktebene pro Vektorelement sichtbar. Beispiele sind hier die Vektorreduktion beim Skalarprodukt und die Umsetzung der Minimumfunktion in Vektorbefehle.

5.0 Zusammenfassung und Ausblick

Die CRAY Y-MP8/832 stellt eine Weiterentwicklung der CRAY X-MP/416 dar, wobei die folgenden Änderungen besonders hervorzuheben sind:

- die Erhöhung der Prozessoranzahl,
- die Verminderung der Prozessorzykluszeit,
- die Veränderung der Struktur des Hauptspeichers.

Die verminderte Prozessorzykluszeit der CRAY Y-MP läßt eine dazu proportionale Leistungsverbesserung erwarten. Diese Verbesserung ist in den in dieser Arbeit vorgestellten Messungen auf allen Prozessoren der CRAY Y-MP in vielen Anwendungsfällen erreicht und teilweise sogar überschritten worden; die Arbeit bestätigt damit die Ergebnisse von [Dete,89] und [FaSc,89].

Die Speichersysteme der CRAY X-MP und der CRAY Y-MP sind in der Lage, sogar bei voller Ausnutzung aller Ports durch alle Prozessoren Berechnungen durchzuführen, die die Grenzen der Leistungsfähigkeit der Funktionseinheiten und der übrigen Systemkomponenten nahezu erreichen lassen. Dies gilt besonders für das Speichersystem der CRAY Y-MP. Hier ist es gelungen, für viele Anwendungsfälle durch ein angemessen gewähltes Zugriffsnetzwerk, ein verändertes Prioritätenschema und eine Verbesserung der Ports sogar technologische Nachteile der Speicherbänke in Relation zur Prozessorzykluszeit auszugleichen. Trotzdem kann zum Erreichen der maximalen Leistungsfähigkeit auf eine Berücksichtigung

- der Hardware,
- der Software-Entwicklungsumgebung und
- der Algorithmenstruktur

nicht verzichtet werden, um eine annähernd optimale Leistung für ein spezifisches Problem zu erreichen.

Im Detail läßt sich aus der Sicht des Anwenders eine Vielzahl von Aspekten finden, die die Leistungsfähigkeit des Systems beeinflussen und die bei der Optimierung der Leistungsfähigkeit ausgewählter Anwendungen beachtet werden müssen. Dabei ergibt sich ein differenzierteres Bild der Systeme, das in Einzelfällen durchaus auch absolute Vorteile für die CRAY X-MP aufweisen kann.

Die Bankstruktur des Hauptspeichers macht beide Systeme anfällig gegenüber Vektorzugriffen mit Schrittweiten, die Potenzen von zwei sind. Dabei ist bei der CRAY Y-MP nicht die Bankanzahl 256 die allein ausschlaggebende Bezugsgröße sondern vielmehr die Anzahl 32 der Untersektionen. Dies hat für Anwendungsprogramme auf einzelnen Prozessoren verschiedene Folgen:

- Lastspitzen im Speicherzugriff einzelner Programme können nicht durch geringe Last im Gesamtsystem ausgeglichen werden.
- Wenig speicherintensive Programme werden kaum von speicherintensiven Programmen gestört.

- Die parallele Ausführung speicherintensiver Programme führt häufig nur zu geringen Leistungseinbußen der einzelnen Programme.

Eine ungünstige Wahl der Schrittweite bei Speicherzugriffen allein erzeugt nur dann eine erhebliche Belastung des Systems, wenn die verschiedenen Zugriffe auch auf gleiche Bankmengen erfolgen. Bei der CRAY Y-MP bedeutet dies für Mehrportzugriffe der Bezug auf die 32 Untersektionen und für Mehrprozessorzugriffe den Bezug auf die 256 Bänke. Hier kann eine Verschiebung der Startadresse von Feldern, aber auch eine Verschiebung der Zugriffe auf andere Teilbereiche des gleichen Feldes zur Entschärfung der Situation auf einem Prozessor oder auch im Gesamtsystem beitragen. Deshalb ist es nicht selbstverständlich, daß die Verteilung von Feldzugriffen auf mehrere Prozessoren, wie sie z.B. bei Multitasking-Anwendungen auftreten, bei Potenzen von zwei als Schrittweiten verstärkte Zugriffskonflikte erzeugen. Vielmehr muß auch hier die Einzelkonstellation untersucht werden. Die Vorteile disjunkter Mengen von benutzten Bänken kann die CRAY X-MP aber nur dann wirklich nutzen, wenn sie nicht mit Portverzögerungen einhergehen.

Der Preis, den der Anwender der CRAY Y-MP immer für die Verbesserung der Technologie zahlen muß, ist eine relativ längere Start-Up-Zeit bei Vektorverarbeitung, wie sie sich in den Hockney-Parametern r_∞ und $n_{1/2}$ gezeigt hat. Diese Eigenschaft ergibt sich aus der Verlängerung der Ausführungszeiten einiger Befehle, die nur auf den einzelnen Prozessor wirken, und der Verlängerung der Speicherpipeline, wie sie z.B. durch den auf fünf Takte verlängerten Bankzyklus erzeugt wird. Interessant ist außerdem die steigende Tendenz der Halbleistungslänge $n_{1/2}$, die bei wachsender Anzahl benutzter CPUs zu beobachten ist.

Für die Anwendung auf Algorithmen ergeben sich aus den Eigenschaften der Speichersysteme, wie sie in den in dieser Arbeit durchgeführten Messungen erkennbar sind, bei der CRAY Y-MP die gleichen Ansatzpunkte, die bereits von der CRAY X-MP bekannt sind. Schrittweiteneffekte sind besonders als Folge von Zugriffen auf Felder mit ungünstigen Dimensionen und Schleifen mit expliziten Schrittweiten zu erwarten. Als Hilfsmittel zur Verminderung der Speicherzugriffe ist auch bei der CRAY Y-MP das Unrolling von Schleifen sehr effektiv. Eine weitere Optimierung scheint nur in Ausnahmefällen sinnvoll zu sein, da sie meist von der reinen Betrachtung auf Hochsprachenebene hin zur Untersuchung des Assembler-Codes und auch relativ schwacher Effekte führt.

In der vorliegenden Arbeit sind die Speicherarchitekturen der CRAY X-MP und der CRAY Y-MP mit verschiedenen Methoden und unter wechselnden Aspekten untersucht worden. Trotzdem bleiben einige interessante Ansätze unberücksichtigt. Im folgenden werden einige dieser Alternativen aufgezeigt:

Besonders interessant in Hinblick auf die Universalität der Speicherkonzepte sind die indizierten Vektorzugriffe (Scatter, Gather) auf den Hauptspeicher. Diese Zugriffe erzeugen meist keine regelmäßige Zugriffsstruktur und unterlaufen damit z.B. die einmalige Synchronisation mehrerer Vektorzugriffe auf den Speicher.

Da bei den hier durchgeführten Messungen der Ablauf gleichzeitig ablaufender Programme im Mehrprozessorbetrieb nur beim Start synchronisiert wird, sind Messungen

von Interesse, die eine Synchronisation der Programme während der Ausführungszeit ermöglichen. Diese Möglichkeit bietet das Multitasking [Rege,89]. Hiermit ist eine gezielte Steuerung und Untersuchung gleichzeitiger Speicherzugriffe mehrerer Prozessoren möglich.

Als konsequente Fortsetzung der Einzelprozessormessungen mit einer Verschiebung der Startbänke zueinander bieten sich Messungen an, in denen die Startbänke der Felder zwischen den einzelnen Programmen auf den verschiedenen Prozessoren verschoben angeordnet werden.

Als Folgerungen aus den in dieser Arbeit gemachten Untersuchungen lassen sich Verbesserungen der Hardware und Software der CRAY-Rechner vorgeschlagen. Ein Ansatzpunkt sind die vom Compiler erzeugten Informationen. Die Optimierungstrategien des FORTRAN-Compilers haben sich bei der Untersuchung mehrerer Programme als sehr ausgefeilt herausgestellt, die Informationen darüber beschränken sich für den Anwender aber auf die Meldung, daß ein Programmstück vektorisiert wird. Hier ist eine weitere Information über

- die erwartete Anzahl von nacheinander ablaufenden Vektoroperationen (kein Chaining, keine parallele Abarbeitung mehr möglich),
- die dabei jeweils auftretenden Vektorspeicherzugriffe (ein, zwei oder drei Ports benutzt)

wünschenswert. Die bereits vorhandene Optimierung der Benutzung von Registern, Funktionseinheiten und Ports in Programmen durch den Compiler baut gerade auf derartigen Informationen auf. Auf Basis dieser Informationen ließe sich möglicherweise ein Maß für die pro Schleifendurchlauf benötigten Takte und die auftretende Speicherintensität schon zur Übersetzungszeit gewinnen. Die Aussagekraft dieses statischen Maßes wäre aber sicherlich begrenzt, wie das Verhalten während der Laufzeit z.B. bei der Triade (vgl. Abschnitt 3.5.3.2) zeigt.

Das Skalarprodukt bietet mehrere Ansatzpunkte für Verbesserungen:

- eine Einführung einer speziellen Funktionseinheit zur Vektorreduktion,
- ein flexiblerer Aufbau der Vektorregister, der eine Zuweisung von Teilvektoren ermöglicht, die nicht beim ersten Register-Element anfangen,
- eine Verbesserung des vom Compiler erzeugten Codes bei der Vektorreduktion für Vektorlängen, die kürzer als eine Vektorregisterlänge sind.

Literaturverzeichnis

[AuBr,84] M.C. August, G.M. Brost, C.C. Hsiung, A.J. Schiffleger
CRAY X-MP: The Birth of a Supercomputer
Computer (USA) 22 (1) (1989) 45-52.

[Bail,87] D.H. Bailey
Vector Computer Memory Bank Contention
IEEE Trans. Comp. 36 (3) (1987) 293-298.

[Buch,83] I.Y. Bucher
The Computational Speed of Supercomputers
ACM Proceedings of SIGMETRICS, 1983.

[CaEl,85] D.A. Calahan, K. Elliot
Memory Conflict Simulation of a Many-Processor CRAY Architecture PART I: CRAY X-MP Study
Report, Los Alamos National Laboratory and Air Force Office of Scientific Research, 1985.

[Cala,88] D.A. Calahan
An Analysis of Vector Startup Access Delays
IEEE Trans. Comp. 37 (9) (1988) 1134-1137.

[ChSm,86] T. Cheung, J.E. Smith
A Simulation Study of the CRAY X-MP Memory System
IEEE Trans. Comp. 35 (7) (1986) 613-622.

[CRAY,85] CRAY Research Inc.
CRAY X-MP Four-Processor Mainframe Reference Manual
CRAY Research Inc., HR-0097, 1985.

[CRAY,88a] CRAY Research Inc.
CFT-Optimization Guide
CRAY Research Inc., SG-0115, 1988.

[CRAY,88b] CRAY Research Inc.
CRAY Y-MP System Programmer Reference Manual
CRAY Research Inc., CSM-0400-000, 1988.

[CRAY,88c] CRAY Research Inc.
CFT77 Reference Manual
CRAY Research Inc., SR-0018, 1988.

[Deo,74] N. Deo
Graph Theorie with Applications to Engeneering and Computer Science
Prentice Hall, Englewood Cliff N.J., 1974

[Dete,87] U. Detert
Programmiertechniken für die Vektorisierung
Praxis der Informatik und Kommunikation 3 (1987) 189-195.

[Dete,89] U. Detert
Memory Performance of CRAY X-MP and CRAY Y-MP
Proceedings 24th Semi-Annual Cray User Group Meeting, Trondheim, 1989.

[DoEi,84] J.J. Dongarra, S.C. Eisenstat
Squeezing the Most out of an Algorithm in CRAY FORTRAN
ACM Trans. Math. Softw. 10 (3) (1984) 219-230.

[DoGu,84] J.J. Dongarra, F.G. Gustavson, A. Karp
Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine
SIAM Review 26 (1) (1984) 91-112.

[FaSc,89] G.J. Faanes, J.L. Schwarzmeier
Comparing the Performance of CRAY Y-MP and CRAY X-MP Computer Systems
CRAY Channels, Winter 1989.

[Gilo,81] W.K. Giloi
Rechnerarchitektur
Springer Verlag, Berlin/Heidelberg/New York, 1981.

[GrKa,69] R.T. Gregory, D.L. Karney
A Collection of Matrices for Testing Computational Algorithms
Wiley-Interscience, New York, 1969.

[Gurk,86] R. Gurke
Graphenalgorithmen für MIMD-Rechner
Kernforschungsanlage Jülich, Jül-Spez-355, 1986.

[Hock,83] R.W. Hockney
Characterizing Computers and Optimizing the FACR(l) Poisson-Solver on Parallel Unicomputers
IEEE Trans. Comp. 32 (10) (1983) 933-941.

[Hock,85] R.W. Hockney
(r_∞, $n_{1/2}$, $s_{1/2}$) Measurements on the 2-CPU CRAY X-MP
Parallel Computing 2 (1) (1985) 1-14.

[HoCu,89] R.W. Hockney, I.J. Curington
$f_{1/2}$: A Parameter to Characterize Memory and Communication Bottlenecks
Parallel Computing 10 (3) (1989) 277-286.

[HoJe,88] R.W. Hockney, C.R. Jesshope
Parallel Computers 2
Adam Hilger, Bristol and Philadelphia, 1988.

[HoKn,89] F. Hoßfeld, R. Knecht, W.E. Nagel
Multitasking: Experience with Applications on a CRAY X-MP
Parallel Computing 12 (3) (1989) 259-283.

[Hoßf,88] F. Hoßfeld
Vector-supercomputers
Parallel Computing 7 (3) (1988) 373-385.

[HwBr,84] K. Hwang, F.A. Briggs
Computer Architecture and Parallel Processing
McGraw-Hill, New York, 1984.

[IbTo,89] R.N. Ibbett, N.P. Topham
Architecture of High Performance Computers Vol. I
Macmillan Education Ltd., Houndsmills, 1989.

[LaHa,79] C.L. Lawsen, R.J. Hanson, D.R. Kincaid, F.T. Krogh
Basic Linear Algebra Subprograms for Fortran Usage
ACM Trans. Math. Softw. 5 (3) (1979) 308-323.

[Oed,85] W. Oed
Untersuchungen von Zugriffen auf den Arbeitsspeicher in Vektorrechnersystemen
Kernforschungsanlage Jülich, Jül-1994, 1985.

[OeLa,85] W. Oed, O. Lange
On the Effective Bandwidth of Interleaved Memories in Vector Processing Systems
IEEE Trans. Comp. 34 (10) (1985) 949-957.

[OeLa,86] W. Oed, O. Lange
Modelling, Measurement, and Simulation of Memory Interference in the CRAY X-MP
Parallel Computing 3 (4) (1986) 343-358.

[Orte,88] J.M. Ortega
Introduktion to Parallel and Vector Solution of Linear Systems
Plenum Press, New York/London, 1988.

[Rege,89] H. Reger
Vergleich der Multitasking Implementierungen auf CRAY X-MP und IBM 3090
Kernforschungsanlage Jülich, Jül-Spez-542, 1989.

[SaSc,89] Y. Saad, M.H. Schultz
Data Communication in Parallel Architectures
Parallel Computing 11 (2) (1989) 131-150.

[Schn,86] H.J. Schneider (Hrsg.)
Lexikon der Informatik und Datenverarbeitung
Oldenbourg Verlag, München/Wien, 1986.

[Ston,80] H.S. Stone (Hrsg.)
Introduction to Computer Architecture
SRA, Chicago, 1980.

[ZAM,89] Zentralinstitut für Angewandte Mathematik der KFA Jülich
Zentrale Datenverarbeitung und Mathematik, Systeme und Forschungsarbeiten, Statusbericht 1989
Kernforschungsanlage Jülich, Zentralinstitut für Angewandte Mathematik, 1989.

Diese Arbeit wurde als Diplomarbeit am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen angefertigt.

Herrn Prof. Dr. F.Hoßfeld danke ich für die Möglichkeit, die Arbeit im Zentralinstitut für Angewandte Mathematik des Forschungszentrums (KFA) Jülich anzufertigen; die hier erfahrene Unterstützung war wichtige Grundlage für das Erstellen dieser Arbeit. Mein besonderer Dank gilt Herrn U. Detert für die intensive Betreuung und Unterstützung.