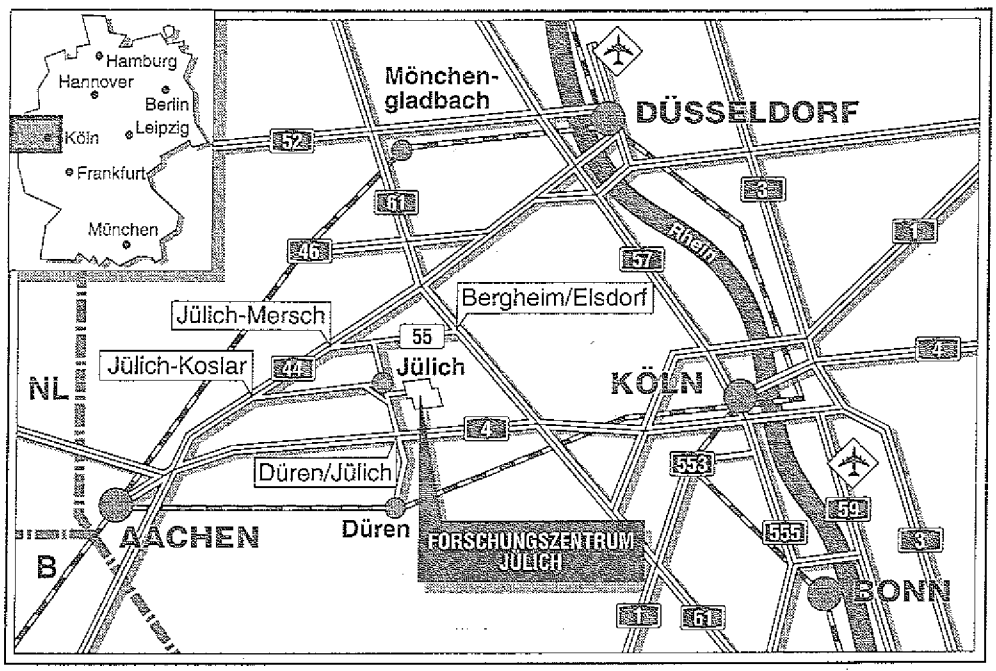
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentralinstitut für Angewandte Mathematik

Analyse der Konfliktauflösung von synchronen Vektorzugriffen auf den gemeinsamen Speicher bei CRAY-Mehrprozessorsystemen

Achim Pitsch



Berichte des Forschungszentrums Jülich ; 2619
ISSN 0366-0885
Zentralinstitut für Angewandte Mathematik Jül-2619

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
Postfach 1913 · D-5170 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Analyse der Konfliktauflösung von synchronen Vektorzugriffen auf den gemeinsamen Speicher bei CRAY-Mehrprozessorsystemen

Achim Pitsch

Inhalt

Einleitung

Supercomputer haben sich in den letzten Jahren als unverzichtbar für Naturwissenschaft und Technik erwiesen. Viele Probleme aus diesen Disziplinen werden heute nicht mehr mit theoretisch-analytischen Methoden, sondern mit Hilfe von umfangreichen Simulationen auf Supercomputern behandelt. Beispielhaft seien hier Vielteilchenprobleme in der Physik, meteorologische Untersuchungen und Strömungsverhalten von Fahr- und Flugzeugen genannt. Trotz der Leistung heutiger Systeme, die meist als Vektorrechner mit wenigen Prozessoren ausgelegt sind, existieren Problemstellungen, deren Anforderungen durch verfügbare Architekturen nicht erfüllt werden können. In diesem Bereich scheint eine grundsätzliche Lösung lediglich mit Hilfe von *massiv parallelen Systemen* möglich zu sein. Die Entwicklung im Zusammenhang mit entsprechenden Konzepten ist jedoch noch in der Anfangsphase. So verhindert ein Mangel an Programmiererfahrung und vor allem ein Mangel an verfügbaren Werkzeugen zur Programmentwicklung und Leistungsverbesserung bislang noch den breiten Einsatz existierender massiv paralleler Systeme als allgemeine Hochleistungsrechner für Wissenschaft und Industrie [Hoß91].

Viele Algorithmen, insbesondere auch rechenintensive Simulationen, arbeiten auf umfangreichen Datenmengen. Damit fällt dem Datentransport bei entsprechenden Applikationen eine große Bedeutung zu. Oft wird die Leistung eines Systems durch die Transferkapazität begrenzt. Einfluß auf die Transferleistung eines Systems hat vor allem die zugrundeliegende Architektur. Je nach Anordnung des Speichers in den betrachteten Systemen unterscheidet man zwei Konzepte: Beim *Shared-Memory-Konzept* greifen alle Prozessoren auf einen großen, gemeinsamen Hauptspeicher zu, der auch als Medium für den Austausch von Daten genutzt wird. Typische Vertreter dieser Klasse von Rechnern wie die SIEMENS VP-S mit zwei Prozessoren, die NEC SX-3 und die CRAY X-MP mit vier Prozessoren oder die CRAY Y-MP mit acht Prozessoren werden auch als *schwach parallele Systeme* bezeichnet. Auch die kürzlich vorgestellte CRAY Y-MP C90 mit 16 Prozessoren kann noch dieser Klasse zugeordnet werden [BEK90, SW91]. Beim gleichzeitigen Zugriff mehrerer Prozessoren auf einen gemeinsamen Hauptspeicher treten *Speicherzugriffskonflikte* auf, die zu einer Verzögerung des Programmablaufs auf den betroffenen Prozessoren führen. Mit zunehmender Prozessoranzahl steigt die Anzahl dieser Speicherzugriffskonflikte. Daher erweist sich für große Prozessoranzahlen der gemeinsame Hauptspeicher als kritische Ressource. Massiv parallele Systeme der neuen Generation wie die Connection Machine CM-5 von Thinking Machines und Intels PARAGON sind für 1024 bzw. 2048 leistungsfähige Prozessoren ausgelegt. Bei dieser Größenordnung

der Prozessoranzahl ist ein gemeinsamer Speicher nicht mehr effizient realisierbar, daher verfügen die einzelnen Prozessoren über jeweils lokale Speicher. Da solche Systeme mit verteiltem Speicher (*Distributed Memory*) den Datenaustausch und die Kommunikation mit anderen Prozessoren über Kommunikationsnetzwerke realisieren, werden sie auch als *Message-Passing-Systeme* bezeichnet. Die Bewältigung der Interprozessorkommunikation und der Transport von Daten stellen auf heutigen Rechnern dieses Typs noch Probleme dar, die sich stark in Leistungsverlusten bemerkbar machen. Besonders deutlich fällt daher bei diesen Systemen die Differenz von theoretischer und tatsächlich erreichbarer Leistung aus [HB91, Hel91].

Shared-Memory-Rechner zeichnen sich durch ihre gute Programmierbarkeit und die mittlerweile verfügbaren Programmierhilfen aus, sind jedoch für große Prozessoranzahlen nicht effizient zu verwirklichen und daher in ihrer Leistung begrenzt. Demgegenüber lassen sich massiv parallele Systeme mit skalierbarer Prozessoranzahl problemlos konstruieren, sind jedoch wiederum nur schwer zu programmieren. Ein Lösungsansatz ist die Idee der *Virtual-Shared-Memory-Systeme*. Sie sind als massiv parallele Systeme konstruiert, ihr verteilter Speicher wird jedoch logisch und programmiertechnisch wie ein gemeinsamer Speicher behandelt [Ber92].

Die hohen Prozessorleistungen heutiger Systeme erfordern hohe Transferleistungen zwischen Speicher und Prozessoren. Als problematisch erweist sich dabei das *Latenz*-Problem der Speicherbausteine. Durch die Größe der Hauptspeicher müssen die verwendeten Speicherbausteine nicht nur möglichst kompakt, sondern auch preiswert gehalten werden, so daß meist Speicherbausteine mit größeren Zugriffszeiten als der Prozessortakt eingesetzt werden. Ein solcher Baustein ist durch einen Zugriff für die Dauer von mehreren Prozessortakten belegt (*Speicherzykluszeit*). Die Hauptspeicher der Shared-Memory-Systeme sind daher im allgemeinen in mehrere, physikalisch unabhängige *Speicherbänke* aufgeteilt (verschränkt), so daß nicht der gesamte Speicher durch einen Zugriff blockiert wird und auch mehrere Prozessoren gleichzeitig auf verschiedene Bänke zugreifen können. Eine darüber hinausgehende Möglichkeit zur Beschleunigung von Prozessorzugriffen ist der zusätzliche Einsatz eines *Cache-Speichers*.

Dem Cache-Konzept liegt eine hierarchische Speicherstruktur zugrunde. Zwischen Hauptspeicher und Prozessor wird ein schneller Halbleiterspeicher geringer Größe installiert, dessen Zugriffszeit erheblich unter der des Hauptspeichers liegt. Im Verlauf der Bearbeitung eines Programms finden Speicherzugriffe im Mittel überwiegend lokal statt. Daher werden zusammenhängende Blöcke des gerade bearbeiteten Programms in den Cache kopiert, um dem Prozessor auf diese Daten erheblich schnelleren Zugriff als auf Daten im Hauptspeicher zu ermöglichen. Im Zusammenhang mit diesem Konzept treten viele Fragestellungen auf, von denen hier lediglich Ersetzungsstrategie und Redundanzproblematik in Mehrprozessorsystemen exemplarisch genannt sein sollen. Ausführliche Betrachtungen zu diesen Themen können [Smi82] entnommen werden. Untersuchungen zum Einfluß des Cache-Mechanismus auf die Leistung von Großrechnern finden sich u.a. in [HH91].

Die Bänke eines verschränkten Hauptspeichers sind über ein *Speicherverbindungsnetzwerk* mit den Prozessoren verbunden, die ihrerseits meist über mehrere Zugänge zum Speicher (*Ports*) verfügen. Im Zusammenhang mit dem Zugriff mehrerer unabhängiger Zugriffspfade auf einen verschränkten Speicher ist der Begriff der *Bandbreite* von großer Bedeutung. Die Bandbreite eines Speichersystems ist ein Maß für die durchschnittliche Anzahl der in einem Takt gleichzeitig zugreifbaren Speicherworte und berechnet sich aus dem Quotienten von Bankanzahl und *Bankzykluszeit*. Sie sollte an die Anzahl der parallel zugreifenden Ports in einem System angeglichen sein, damit alle Ports gleichzeitig arbeiten können. Bei steigender Prozessoranzahl und damit steigender Bankanzahl erhöht sich zwangsläufig die Komplexität des Verbindungsnetzwerkes. Eine sogenannte *Crossbar*-Verbindung, die jede Bank mit jedem Port direkt verbindet, ist wegen des hohen Aufwands technisch nicht möglich oder nicht effizient zu realisieren. Daher werden *mehrstufige Speicherverbindungsnetzwerke* eingesetzt, die zu einer drastischen Reduzierung der Verbindungszahl führen.

Bedingt durch die Struktur eines solchen Systems sind Konfliktsituationen beim Zugriff der Prozessoren auf den gemeinsamen Hauptspeicher nicht ausgeschlossen. Diese Speicherzugriffskonflikte finden entweder bereits im Verbindungsnetzwerk statt oder entstehen durch die Referenz auf belegte Speicherbänke. Mechanismen zur Vermeidung oder Auflösung solcher Konflikte müssen in die Systeme implementiert werden. Insbesondere bei massiven Speicherzugriffen mehrerer Prozessoren können diese Netzwerke einen Engpaß zum Speicher darstellen, der zu einer Beeinträchtigung der Transferleistung und damit zu einer Reduzierung der Systemleistung führt. Daher fällt dieser Problematik auch große Bedeutung beim Entwurf neuer Rechnersysteme zu.

Mit der grundsätzlichen Leistungsfähigkeit von verschränkten Speichern und der Problematik von Interferenzen auf diesen befaßt sich eine Vielzahl von Untersuchungen. Vor der Verfügbarkeit von entsprechenden Architekturen waren sie zwangsläufig theoretischer Natur [Bha75, BS76, Hoo77] und beschäftigten sich häufig mit der Problematik der Bandbreite solcher Systeme [Rau79b, Rau79a, CKL77, Rav72]. Nach dem Debüt der CRAY-Multiprozessorsysteme befaßten sich mehrere Artikel mit den Besonderheiten dieser Systeme, insbesondere werden die CRAY X-MP und die CRAY 2 häufig betrachtet. Dabei sind die durch Speicherzugriffskonflikte verursachten Leistungsverluste häufig das zentrales Thema. Oed und Lange [Oed85, OL85, OL86] widmeten sich allgemein der Analyse von Konfliktsituationen beim gleichzeitigen Zugriff mit mehreren Ports auf verschränkte Speicher. Mit Hilfe mathematischer Modelle stellten sie allgemeine Lösungsvorschläge für den konfliktfreien Zugriff von zwei Ports vor. Aussagen über komplexere Zugriffskonfigurationen für die CRAY X-MP/2 trafen sie anhand von Simulationen. Cheung und Smith [CS85, CS86] führten ähnliche Untersuchungen speziell für die CRAY X-MP/2 mittels Simulation durch. Sie beschrieben architekturabhängige, regelmäßig auftretende Konfliktsituationen und stellten Strategien zu deren Vermeidung vor. Mendez und Tang [MT89] entwarfen ein Wahrscheinlichkeitsmodell zur Betrachtung von Vektorzugriffen mit mehreren Ports auf einen verschränkten Speicher und verglichen

ihre Ergebnisse mit Messungen auf einer CRAY X-MP/2 und einer CRAY Y-MP/8. Calahan veröffentlichte zwischen 1985 und 1990 eine Reihe von Untersuchungen zu Speicherzugriffskonflikten auf den Systemen CRAY X-MP und CRAY 2. Mit Elliott [CE85] entwickelte er einen umfangreichen, Code-gestützten Simulator des Speichersystems der CRAY X-MP Rechner, anhand dessen Ursachen für Verzögerungen durch Zugriffskonflikte erkannt und analysiert werden konnten. In [Cal88a] untersuchte er den Einfluß der Zuordnung von Bänken zu Sektionen auf Startup-Situationen von Vektorzugriffen. In Zusammenarbeit mit Bucher betrachtete er Zugriffskonflikte mit Hilfe von Warteschlangenmodellen [BC90] . Arbeiten zur Modellierung von Zugriffskonfliktsituationen auf CRAY X-MP und CRAY 2 führte er zusammen mit Bailey durch [CB88]. Reinhardt [Rei88b] betrachtete die Leistung von Algorithmen mit unterschiedlicher Speichernutzung unter Microtasking auf CRAY X-MP und CRAY Y-MP. Detert und Hofemann [Det89, Hof90, DH91] verglichen die Speicherarchitekturen von CRAY X-MP auf CRAY Y-MP durch Messungen von vektorisierenden Kernen und Algorithmen.

Nachdem in [Hof90] bereits ausführliche Analysen über das Verhalten der Systeme CRAY X-MP und CRAY Y-MP bei variabler Speicherbelastung durch einen Prozessor vorgestellt wurden, liefern die hier vorliegenden Untersuchungen einen ersten Ansatz zur Betrachtung der Auswirkungen von Speicherzugriffskonflikten bei der Nutzung von Multitasking. Mit Hilfe eines Modellproblems wird die Erzeugung von gleichzeitigen Vektorzugriffen mehrerer Prozessoren auf gemeinsame Daten bei einstellbarer Speicherbelastung ermöglicht. Theoretisch-analytische Untersuchungen des dabei auftretenden Zugriffsverhaltens werden mit Meßergebnissen zum Laufzeitverhalten der Systeme CRAY X-MP und CRAY Y-MP verglichen. Darüber hinaus gehen in die Auswertung des Laufzeitverhaltens für die CRAY X-MP Ergebnisse einer speziell für das Modellproblem entwickelten Simulation ein.

In Kapitel 1 wird ein Überblick über die Architektur der betrachteten Systeme gegeben. Dabei wird vor allem die CRAY X-MP ausführlich hinsichtlich des Zeitverhaltens für Vektoroperationen betrachtet. Für die CRAY Y-MP werden wegen der Ähnlichkeit der beiden Systeme nur die signifikanten Unterschiede herausgestellt.

Das Kapitel 2 behandelt die Implementierung von Multitasking auf den untersuchten Rechnern. Im Mittelpunkt der Betrachtungen stehen dabei das Autotasking-Konzept und die bei der Ausführung eines Multitasking-Programms zur Verfügung stehenden Scheduling-Strategien für einzelne Prozesse.

Kapitel 3 enthält Überlegungen zur Vorgehensweise bei der Untersuchung von Speicherzugriffskonflikten. Für die vorliegende Untersuchung wird eine Methodik vorgestellt, die anhand eines Modellproblems Laufzeitmessungen, analytische Modellierung und Simulation vorsieht. Der Entwurf des Modellproblems und seine Implementierung in FORTRAN77 wird betrachtet. Eine Analyse des Assembler-Codes ermöglicht die Modellierung des Laufzeitverhaltens auf den beiden betrachteten Systemen. Ein analytisches Modell zur Prognostizierung der Laufzeit wird vorgestellt.

Kapitel 4 beschreibt das im Rahmen dieser Arbeit entwickelte Simulationsprogramm. Es modelliert zeitgesteuert das Zugriffsverhalten bei Ausführung des Modellproblems auf der CRAY X-MP.

Kapitel 5 stellt die Ergebnisse der Untersuchungen vor. Nach einer Beschreibung der Meßumgebung werden zunächst die Ergebnisse der Laufzeitmessung auf der CRAY X-MP analysiert. Daran anschließend findet eine detaillierte Betrachtung einzelner Messungen mit Hilfe der Simulation statt. Die Diskussion der auf der CRAY Y-MP gemessenen Laufzeiten schließt das Kapitel ab.

Kapitel 1

Architektur der CRAY-Systeme

Vier Jahre nach der Gründung des Unternehmens *Cray Research, Inc.* wurde 1976 der erste Hochleistungsvektorrechner, die CRAY 1, in Betrieb genommen. Zielsetzung bei der Entwicklung des Systems war die Bereitstellung größtmöglicher Rechenleistung. Der Rechner verfügte über einen Prozessor, der mit einer Zykluszeit von 12.5 Nanosekunden (*nsec*) betrieben wurde. Der 1 Megawort (MW) umfassende Hauptspeicher mit einer Zykluszeit von 4 Takten war in 16 Bänke unterteilt (*interleaved*). Der Rechner verfügte über eine Reihe innovativer Architekturmerkmale, die bis heute kennzeichnend für alle Generationen von CRAY-Supercomputern sind:

- eine Wortlänge von 64 *Bit*
- ein verschränkter Hauptspeicher
- ein eingeschränkter Befehlssatz, der strikt zwischen Speicherzugriffsoperationen und Operationen innerhalb der CPU unterscheidet (Register-zu-Register-Verarbeitung)
- die Trennung von Interpretations- und Ausführungsphase der Maschineninstruktionen
- die Trennung von Adreß-, Skalar- und Vektorverarbeitung
- voneinander unabhängige Funktionseinheiten, die gleichzeitig betrieben werden können
- die Auslegung der Funktionseinheiten als Pipelines

1982 wurde als erstes CRAY-Multiprozessorsystem die Serie X-MP mit 2 Prozessoren und einer Zykluszeit von 9.5 *nsec* vorgestellt. Sie verfügte über einen Hauptspeicher von maximal 2 MW, der in 32 Bänke eingeteilt war. Seit 1984 ist eine auf 4 Prozessoren und 16 MW Speicher erweiterte Version verfügbar, die mit einer Zykluszeit von 8.5 *nsec* betrieben wird. Der 64fach verschränkte Hauptspeicher hat eine Zykluszeit von 34 *nsec*. Während auf der CRAY 1 lediglich ein Zugriffspfad zum Speicher zur Verfügung stand, wurden auf der CRAY X-MP drei unabhängige

Zugriffspfade pro Prozessor implementiert. Der Zugriff der Prozessoren auf den gemeinsamen Hauptspeicher erfolgt über ein Verbindungsnetzwerk. Weitere Verbesserungen bestanden in einer automatischen Verkettung von Vektorbefehlen (*Chaining*) und der Einführung von Hardware-Unterstützung für *Gather/Scatter*-Operationen, die indizierten Vektorzugriff erlauben.

Die 1987 eingeführte CRAY Y-MP stellt mit maximal 8 Prozessoren und einer Taktzeit von 6 *nsec* eine Weiterentwicklung der sogenannten *MP-Linie* dar. Der Hauptspeicher von maximal 64 *MW* hat eine Zykluszeit von 30 *nsec* und ist in 256 Bänke eingeteilt.

Ende 1991 wurde die CRAY Y-MP C90 mit 16 Prozessoren und einer Hauptspeichergröße von 256 *MW* vorgestellt. Die Zykluszeiten von Prozessoren und Speicher wurden auf 4 *nsec* bzw. 15 *nsec* verringert. Der Hauptspeicher ist in 1024 Bänke unterteilt. Die wichtigste Neuerung dürfte die eingeführte Doppelwortverarbeitung darstellen. Jede Aktion des Systems bearbeitet in einem Systemtakt parallel zwei aufeinanderfolgende Worte [SW91]. Weitere Informationen über die Architektur des Systems liegen bisher nicht vor.

Mit der CRAY 2 wurde alternativ zur MP-Linie 1985 eine von der Architektur eher an der CRAY 1 orientierte Weiterentwicklung eingeführt. Das System zeichnet sich durch eine Zykluszeit von 4.1 *nsec* und eine Hauptspeichergröße von maximal 256 *MW* aus. Die 4 Prozessoren dieser Maschine basieren auf der Architektur des CRAY 1 Prozessors und besitzen wie dieser lediglich einen Datenpfad als Verbindung zum Hauptspeicher. Zusätzlich wurde je Prozessor ein lokaler Speicher von 16 Kiloworten und Hardware für den effizienten Datentransfer zwischen CPUs und Speicher implementiert. Die geordnete Reihenfolge des Datenstroms bei Speicherzugriffen muß nicht eingehalten werden. Einerseits entstehen daher geringere Wartezeiten durch Zugriffskonflikte, andererseits muß deshalb auf den Einsatz von Chaining verzichtet werden. Da auf dieser von der MP-Linie abweichenden Architektur andere Problemstellungen auftreten, soll an dieser Stelle nicht näher auf die CRAY 2 eingegangen werden. Detaillierte Informationen können [Cal88b, CB88] entnommen werden.

Da in der vorliegenden Arbeit das zentrale Interesse der CRAY X-MP gilt, wird in den folgenden Abschnitten ausführlich auf Einzelheiten dieser Architektur eingegangen. Die hier vorgestellten Informationen bilden die Grundlage für die Untersuchung der Ausführung von Programmsequenzen auf Assemblerebene, wie sie im Abschnitt 3.3.2 vorgenommen werden. Im Anschluß an diese Betrachtungen sollen kurz die Architekturunterschiede der CRAY Y-MP im Vergleich zur CRAY X-MP angesprochen werden. Ein ausführlicher Vergleich der beiden Systeme findet sich in [Hof90].

1.1 Die CRAY X-MP/416

Alle folgenden Angaben beziehen sich auf die am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich (KFA) installierte CRAY X-MP/416 [HT91]. Hauptbestandteile dieses Systems bilden die 4 Prozessoren und der gemeinsame Hauptspeicher von 16 MW, das *Ein-/Ausgabe-System* (I/O Subsystem, IOS) und ein *Schnellspeicher* (Solid-state Storage Device, SSD) mit 32 MW. Die Prozessoren sind untereinander durch eine Interprozessorkommunikation und mit dem Hauptspeicher durch ein mehrstufiges Verbindungsnetzwerk verbunden. Die Zykluszeit des Rechners beträgt 8.5 $nsec$.

Kenntnisse über Peripherieeinheiten wie SSD, IOS und externe Speicher sind für die Untersuchungen nicht erforderlich. Daher wird an dieser Stelle auf eine tiefergehende Betrachtung dieser Einheiten verzichtet. Informationen dazu finden sich in [HJ88, HB84]. Weitergehende Beschreibungen der Systemarchitektur können [Cra86, Cra87, RR89] entnommen werden.

1.1.1 Aufbau eines Prozessors

Eine CPU der CRAY X-MP läßt sich in mehrere Komponenten mit unterschiedlichen Aufgabenstellungen unterteilen. Die *Kontrollsektion* ist für die Befehlsdecodierung und Verwaltung der Ressourcen innerhalb des Prozessors zuständig. *Skalar-, Adreß-* und *Vektorsektion* bearbeiten arithmetische Operationen. Die 14 voneinander unabhängigen Funktionseinheiten für arithmetische Operationen werden in Skalar-, Adreß-, Vektor- und Gleitkommaeinheiten unterteilt, wobei die ersten drei Bereiche entsprechenden Sektionen zugeordnet sind und der letzte von Skalar- und Vektorsektion gemeinsam genutzt wird. Die folgenden Abschnitte beschreiben die einzelnen Sektionen, wobei besonderes Interesse dem Zeitverhalten bei der Ausführung von Vektoroperationen gilt.

Die Kontrollsektion

Die Instruktionsabarbeitung auf von-Neumann-Rechnern wird allgemein in eine *Aufsetzphase* (Befehlsinterpretationsphase, Issue Phase) und eine *Ausführungsphase* (Befehlsausführungsphase, Execution Phase) unterteilt. Dabei sind Laden und Decodieren des Befehls und eventuell vorhandener Operanden sowie Hochzählen des Programmzählers Abschnitte der Aufsetzphase. Ein Befehl gilt als *aufgesetzt* (issued), wenn er in die Ausführungsphase eintritt. In den CRAY-Systemen werden diese beiden Phasen getrennt voneinander in unterschiedlichen Sektionen bearbeitet. Die Kontrollsektion übernimmt die Aufgaben der Aufsetzphase und muß zusätzlich zur Sicherstellung der Ausführung des Befehls in einer der übrigen Sektionen alle benötigten Ressourcen auf Verfügbarkeit prüfen und reservieren. Durch diese Entkopplung von Aufsetz- und Ausführungsphase ist eine parallele Abarbeitung in dem Sinne möglich, daß Operationen aufgesetzt werden können, während andere Operationen noch ausgeführt werden. Einen Überblick über die Kontrollsektion gibt Abbildung 1.1.

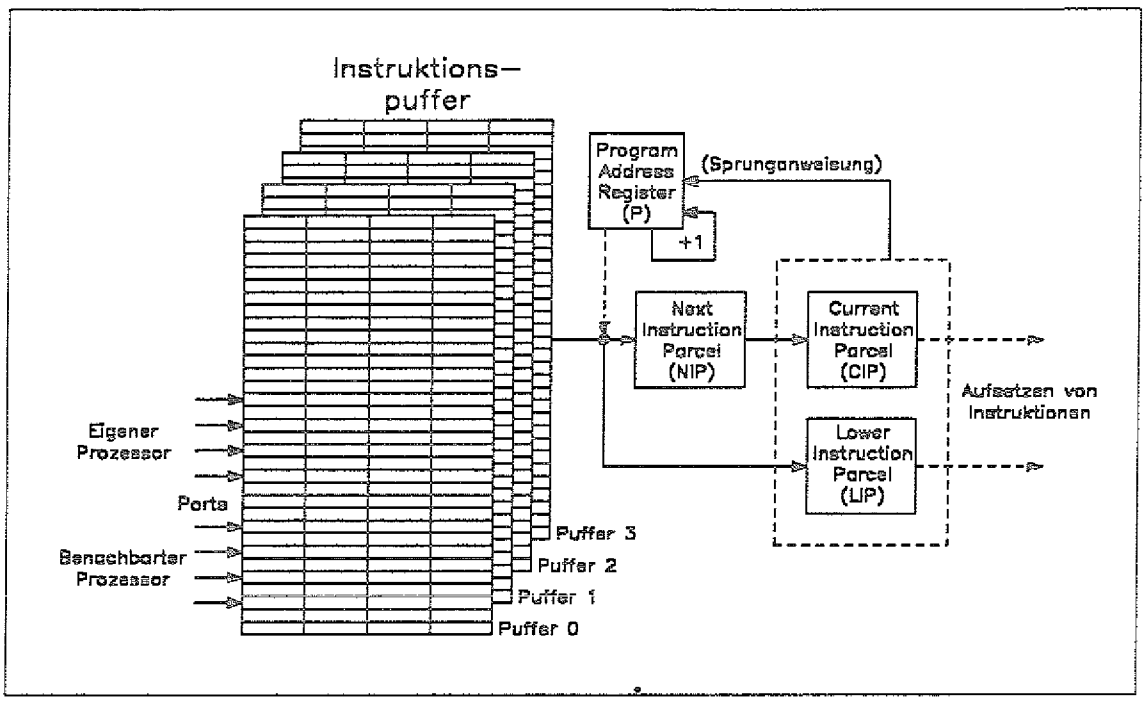


Abbildung 1.1: Kontrollsektion eines CRAY X-MP Prozessors (nach [RR89])

Jeder Prozessor verfügt über 4 eigene *Instruktionspuffer* (Instruction Buffers), die jeweils 32 Worte enthalten. Ein Wort wird in vier 16-Bit Teilworte (*Parcel*) unterteilt. Eine Instruktion der CRAY X-MP besteht aus bis zu 2 Parceln. Die Instruktionspuffer können als Instruktions-Cache mit einer FIFO-Ersetzungsstrategie betrachtet werden. Wird eine Instruktion benötigt, die in keinem Puffer verfügbar ist (*Out of Buffer Condition*), so wird der entsprechende Speicherbereich aus dem Hauptspeicher in einen Puffer geladen (*Code Fetch*).

Während der Aufsetzphase wird eine Ein-Parcel-Instruktion von einem Puffer in das *NIP-Register* (Next Instruction Parcel Register) geladen. Im nächsten Zyklus wird sie ins *CIP-Register* (Current Instruction Parcel Register) geschoben, wo sie decodiert wird. Gleichzeitig wird das nächste Parcel ins NIP-Register geladen. Im CIP-Register wird ein Parcel so lange gehalten, bis der Befehl in die Ausführungsphase eintreten kann. Während dieser nur einen Zyklus dauernden Verweilzeit im CIP-Register werden die zur Ausführung der Operation notwendigen Ressourcen wie Ports, Register oder Funktionseinheiten reserviert. Da ein Großeil der Instruktionen nur ein Parcel lang ist, kann so unter optimalen Bedingungen mit jedem Takt eine Operation aufgesetzt werden. Zwei-Parcel-Befehle benötigen zwei Takte zum Aufsetzen der Instruktion. Während das erste Parcel vom NIP-Register ins CIP-Register übertragen wird, kann gleichzeitig das zweite Parcel direkt vom Puffer ins *LIP-Register* (Lower Instruction Parcel Register) geladen werden, wo es sofort interpretiert wird. Das NIP-Register wird mit einer Null-Operation (No Operation, NOP) initialisiert. Im nächsten Zyklus wird keine Operation aufgesetzt, da das NIP-Register erst wieder nachgeladen werden muß. Verzögerungen der Befehlsinitiierung treten nur durch notwendiges Nachladen eines Instruktionspuffers oder

blockierte Ressourcen auf.

Eine Sonderfunktion fällt dem Sprungbefehl zu. Bedingte Sprünge werden nur in Abhängigkeit vom Skalarregister *S0* oder Adreßregister *A0* erlaubt. Während der Auswertung der Sprungbedingung können keine weiteren Instruktionen aufgesetzt werden. Findet kein Sprung statt, so werden zwei Takte für die Ausführung dieser Instruktion benötigt. Andernfalls ist die Dauer einer Sprunganweisung abhängig von der Verfügbarkeit der adressierten Instruktion in einem Instruktionspuffer. Nach Ausführung der Anweisung wird wie üblich mit der weiteren Instruktionsabarbeitung fortgefahren.

Die Skalarsektion

Die Skalarsektion, schematisch dargestellt in Abbildung 1.2, besteht aus 8 *Skalarregistern* (S-Register), 64 *Blockregistern* (T-Register) und 4 Funktionseinheiten (Functional Units) zur Ausführung von ganzzahligen Operationen. Für die Gleitkommaarithmetik werden die 3 Gleitkomma-Funktionseinheiten gemeinsam mit der Vektorsektion genutzt. Alle Funktionseinheiten und Register sind mit einer Wortbreite von 64 Bit ausgelegt. Die S-Register dienen primär zur Ein-/Ausgabe für Berechnungen, während die T-Register als Hilfsregister zur temporären Speicherung von Daten über keinen direkten Anschluß an die Funktionseinheiten verfügen. Die

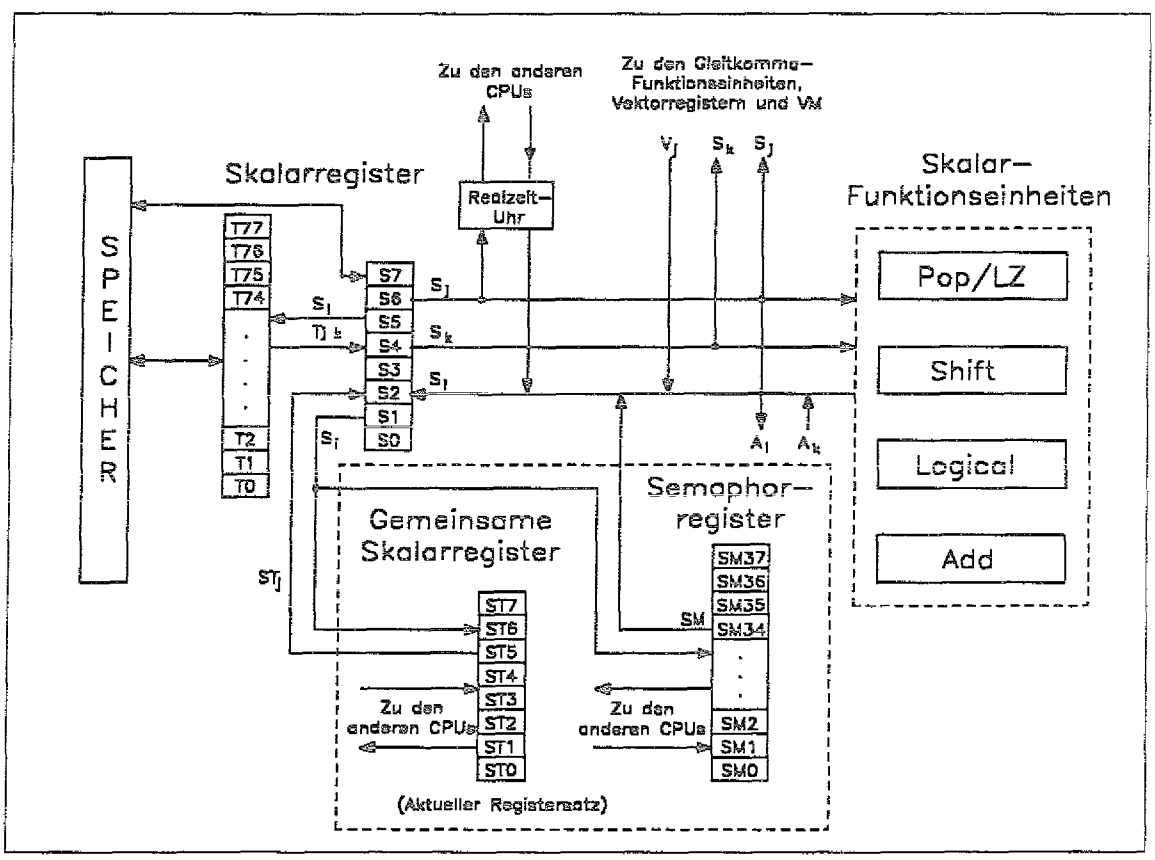


Abbildung 1.2: Skalarsektion eines CRAY X-MP Prozessors (nach [RR89])

Skalarregister haben unter anderem Zugriff auf die *Echtzeituhr* (Real Time Clock, RTC) des Systems und auf die gemeinsamen Skalar- und Semaphorregister der Interprozessorkommunikation. Ein direkter Datentransfer ist sowohl mit den Adreßregistern als auch mit einzelnen Vektorelementen der Vektorregister möglich. Darüber hinaus können Werte der S-Register als Operanden für einige Vektoroperationen genutzt werden.

Jede Funktionseinheit eines Prozessors ist als Pipeline ausgelegt, daher können ihr in jedem Zyklus neue Operanden zugeführt werden. Eine Operation gilt als komplett abgearbeitet, wenn das Ergebnis in ein Register geschrieben wurde. Die Dauer einer Operation läßt sich somit an der Stufenanzahl der Pipeline ablesen. Eine Verzögerung einer Skalaroperation kann also dann auftreten, wenn ein benötigter Operand noch nicht komplett berechnet ist, nicht jedoch aufgrund Blockade einer Pipeline durch eine andere Skalaroperation. Eine solche Situation wird von der Kontrollsektion erkannt, und die Operation wird erst bei Verfügbarkeit des fehlenden Operanden aufgesetzt. Da die Gleitkommaeinheiten auch von der Vektorsektion genutzt werden, ist eine Verzögerung einer skalaren Gleitkommaoperation wegen der Reservierung der entsprechenden Pipeline durch eine laufende Vektoroperation ebenfalls möglich.

Die Adreßsektion

Abbildung 1.3 zeigt den schematischen Aufbau der Adreßsektion. Analog zur Skalarsektion besitzt sie 8 *Adreßregister* (A-Register) und 64 Blockregister (B-Register).

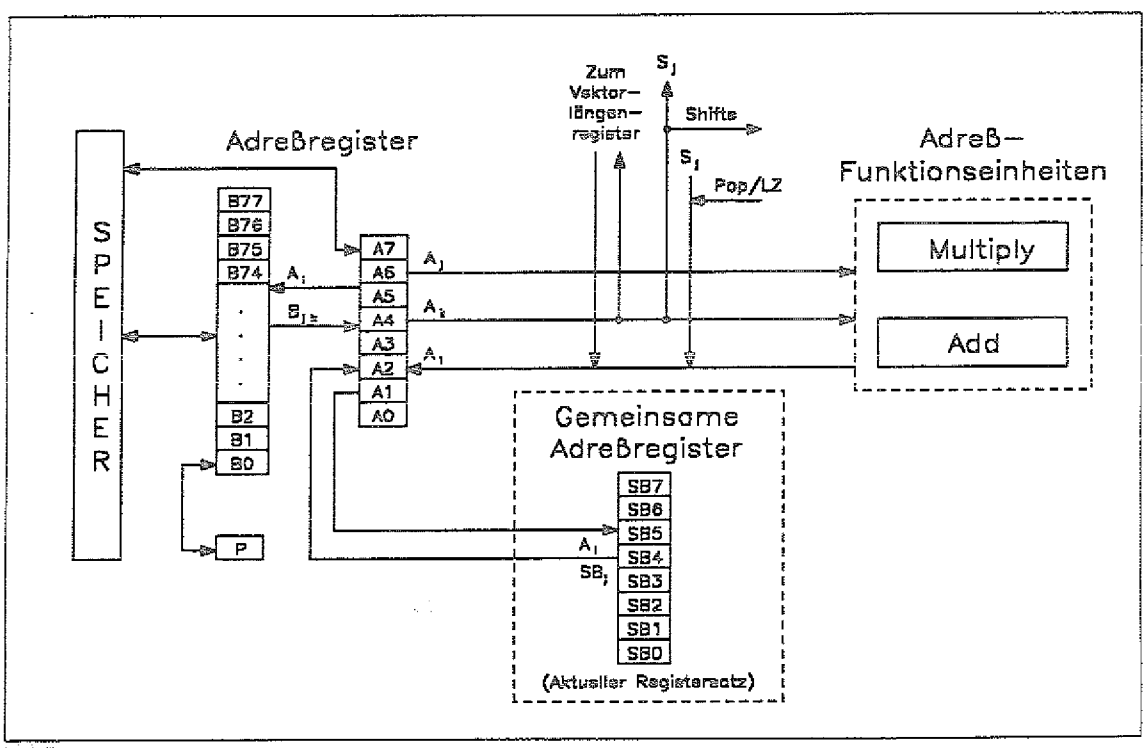


Abbildung 1.3: Adreßsektion eines CRAY X-MP Prozessors (nach [RR89])

Sie verfügt über zwei Funktionseinheiten zur Adreßberechnung und nutzt einige Pipelines der Skalarsektion. Im Gegensatz zu allen übrigen Ressourcen der CPU verfügen die Register und Pipelines der Adreßsektion lediglich über eine Wortbreite von 24 Bit. A-Register werden als Ein-/Ausgabe für Adreßberechnungen und als Indexregister genutzt. Die Adressen werden in Wortadressen (22 Bit) und Parceladressen (24 Bit) unterschieden. Die Art der Adressierung wird durch den entsprechenden Befehl festgelegt. Die Adreßregister sind mit den gemeinsamen Adreßregistern der Interprozessorkommunikation verbunden und können mit diesen Daten austauschen.

Die Vektorsektion

Als *Vektor* im Sinne der Datenverarbeitung auf Computersystemen wird eine Folge von Werten gleichen Typs bezeichnet. Soll die gleiche Operation auf allen Elementen eines Vektors durchgeführt werden, so wird zur Spezifikation genau ein *Vektorbefehl* benötigt. Die Bearbeitung solcher Vektorbefehle setzt eine spezielle Hardware voraus. Sie nutzt die Möglichkeit der Segmentierung einer Operation zur Beschleunigung der Bearbeitung vieler Elemente durch eine *Pipeline* (siehe Abbildung 1.4). Da alle Operationen der CRAY-Systeme dieses Prinzip nutzen, soll vor der genaueren Betrachtung der Vektorsektion am Beispiel der Addition kurz das Grundprinzip des *Pipelining* vorgestellt werden.

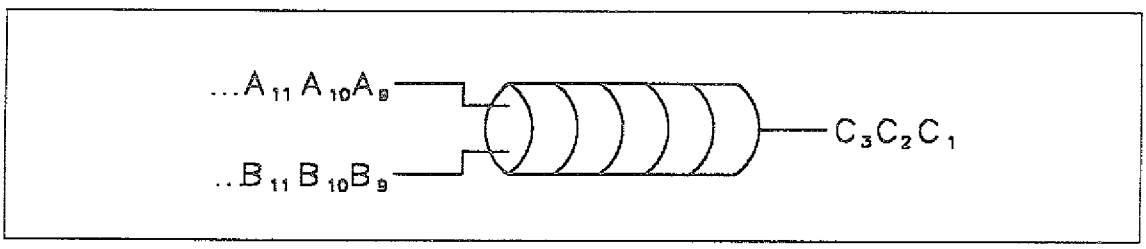


Abbildung 1.4: Pipelining einer Vektoroperation

Abbildung 1.5 (Seite 14) zeigt die Aufteilung einer Additionsoperation zweier Gleitkommawerte in 5 Stufen. In der ersten Stufe werden die Zahlen in Mantissenwerte und Exponentenwerte getrennt, eine eventuell vorhandene Differenz der Exponenten wird ermittelt. Die zweite Stufe regelt die Angleichung der Mantisse der kleineren Zahl abhängig von dieser Differenz. Die beiden Mantissenwerte werden in Stufe drei addiert. Die Stufen vier und fünf dienen der Normalisierung des Ergebnisses. Den Stufen sind jeweils Speicher zwischengeschaltet, die das Ergebnis einer Stufe aufnehmen und am Anfang des folgenden Schritts an die nächste Stufe weiterleiten. Die Aufgabenverteilung wird so bemessen, daß die Bearbeitungszeiten der einzelnen Stufen ähnlich sind. Implementiert man diese Konstruktion auf einem Digitalrechner, so gibt die längste Bearbeitungszeit dieser 5 Stufen eine Schranke für die Dauer eines Systemtakts an, die nicht unterschritten werden kann. Unter diesen Voraussetzungen werden für die Berechnungen einer Addition im vorliegenden Fall fünf Takte benötigt. Bei der sukzessiven Bearbeitung von gleichen Operationen können nun in jedem Takt neue Daten in die erste Stufe der Pipeline eingegeben werden. Während der ersten 5 Takte durchläuft das erste Operandenpaar nacheinander

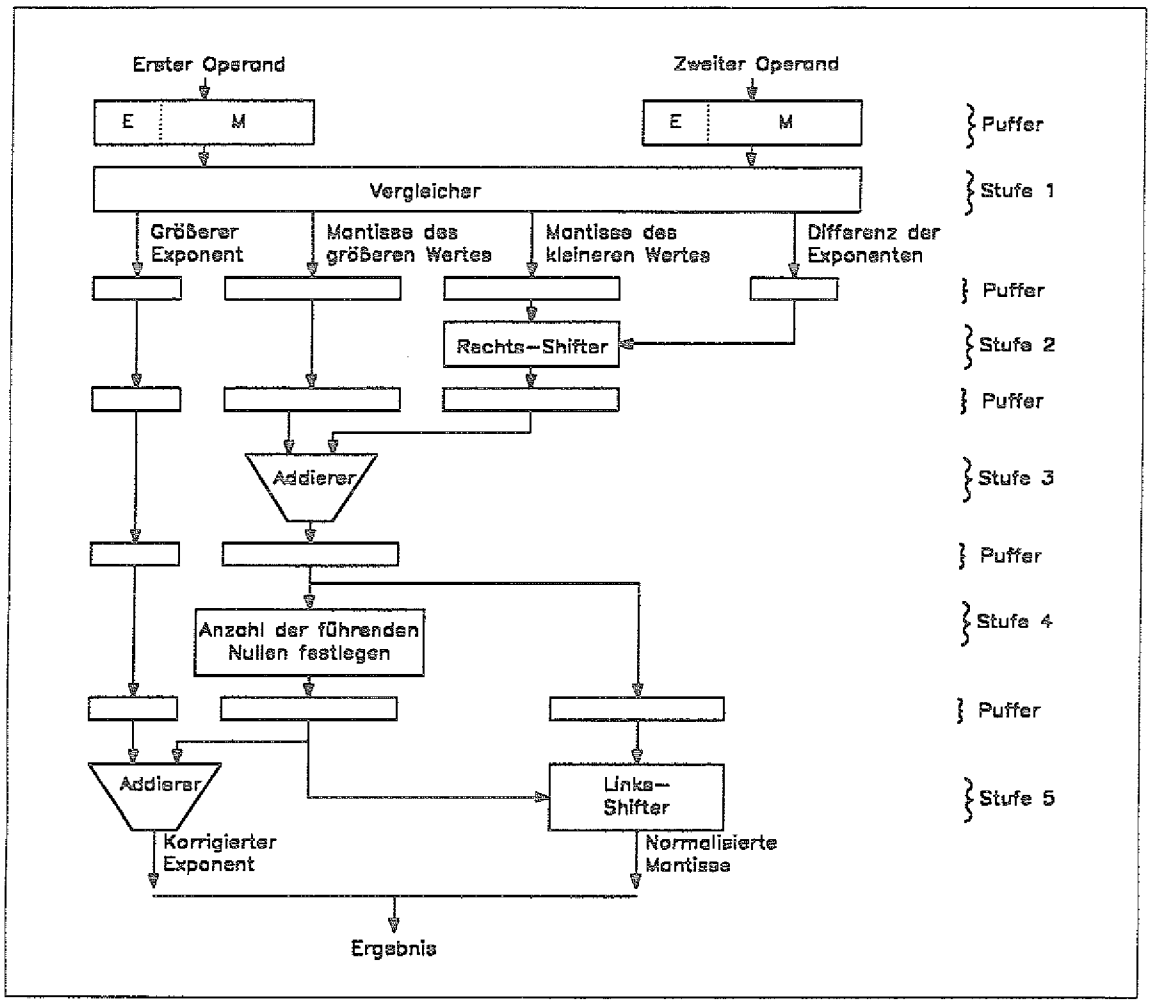


Abbildung 1.5: Pipelining am Beispiel der Gleitkommaaddition (nach [Kog81])

alle Stufen. Dabei wird in jedem Zyklus ein weiteres Operandenpaar in die Pipeline eingespeist. Diese zur Füllung der Pipeline notwendige Zeit wird als *Startup-Zeit* bezeichnet. Danach wird in jedem Takt ein weiteres Ergebnis produziert. Die Zeit zur Berechnung einer Operation beträgt nach wie vor 5 Takte, da jedoch nunmehr in jedem Takt ein Ergebnis vorliegt, erhält man eine asymptotische Steigerung der Bearbeitungsgeschwindigkeit (*Speedup*) um den Faktor fünf.

Abbildung 1.6 zeigt den schematischen Aufbau der Vektorsektion. Alle Funktionseinheiten sind als Pipelines ausgelegt. Zusätzlich zu den 5 eigenen Einheiten zur Bearbeitung von ganzzahligen Operationen nutzt die Vektorsektion die 3 Gleitkommaeinheiten. Die Einheiten *Second Vector Logical* und *Floating Point Multiply* benutzen gemeinsame Datenpfade zur Ein-/Ausgabe; *Pop/Parity* und *Reciprocal Approximation* verwenden denselben Eingabepfad. Von diesen Ausnahmen abgesehen können alle Funktionseinheiten gleichzeitig genutzt werden. Die 8 Vektorregister können je 64 Elemente aufnehmen und haben die Möglichkeit, über drei Ports parallel auf den Speicher zuzugreifen. Zwei Ports können Daten lesen, während auf dem dritten Schreibzugriffe durchgeführt werden.

Die Vektoroperationen werden vom *Vektormaskenregister VM* (Vector Mask Register) und vom *Vektorlängenregister VL* (Vector Length Register) beeinflußt. Das VM-Register erlaubt das Ausblenden einzelner Werte der Vektorregister bei einer Berechnung, und das VL-Register bestimmt die Länge des zu bearbeitenden Vektors.

Eine Vektoroperation wird in drei Phasen zerlegt :

1. Während der *Initialisierungsphase* (Setup Phase) wird die Pipeline initialisiert und die Datenpfade zwischen ihr und den Registern etabliert. Während dieser 3 Takte andauernden Phase können der Pipeline keine neuen Werte zugeführt werden, bereits in der Pipeline vorhandene Werte werden jedoch weiterbearbeitet.

2. In der *Ausführungsphase* (Execution Phase) wird die Berechnung durchgeführt. In jedem Takt wird ein Operandenpaar von den *Operandenregistern* in die erste Stufe der Pipeline übertragen. Nach einer Startup-Zeit, die der Stufenanzahl der Pipeline entspricht, wird in jedem Takt ein Ergebnis in das *Ergebnisregister* geschrieben. Nachdem das letzte Operandenpaar in die Funktionseinheit eingegeben wurde, können die Operandenregister wieder genutzt werden. Einen Zyklus später steht auch die Pipeline für eine weitere Operation zur Verfügung, obwohl noch Elemente berechnet werden.

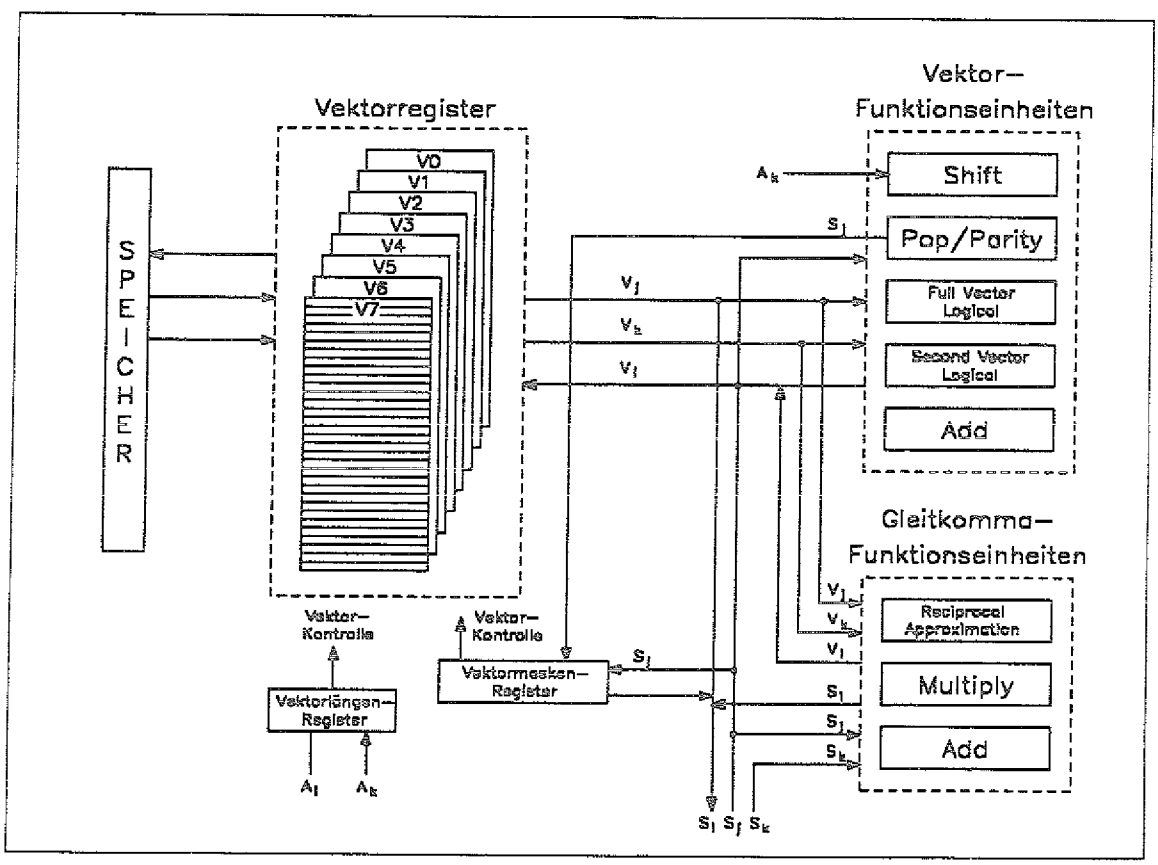


Abbildung 1.6: Vektorsektion eines CRAY X-MP Prozessors (nach [RR89])

3. Die *Rücksetzphase* (Shutdown Phase) betrifft nur das Ergebnisregister und beginnt mit dem Eintreffen des letzten Ergebnisses. Sie dauert 3 Takte, erst danach darf das Ergebnisregister wieder durch einen neuen Befehl reserviert werden.

Bezeichnet man die Vektorlänge mit vl und die Stufenzahl einer Pipeline mit s, so lassen sich aus diesen Angaben die Reservierungszeiten der einzelnen Ressourcen berechnen. Ein Operandenregister ist erst nach dem Auslesen seines letzten Elements für die nächste Operation nutzbar. Es ist also durch eine Vektoroperation für $3 + vl$ Takte belegt. Die Pipeline ist erst einen Takt nach Empfang des letzten Elements belegt und wird daher für $3 + vl + 1 = vl + 4$ Takte durch eine Operation reserviert. Obwohl nach dieser Reservierungszeit bei längeren Pipelines noch Elemente in Bearbeitung sind, kann sie von der nächsten Operation genutzt werden. Es wird deutlich, daß pro Vektoroperation, unabhängig von der Startup-Zeit s, 4 Takte für die Berechnung verlorengehen. Die Pipeline kann in dieser Zeit nicht nachgefüllt werden. Das letzte Ergebnis einer Vektoroperation verläßt die Pipeline $3 + vl + s$ Takte nach dem Aufsetzen der Operation und wird in das Ergebnisregister geschrieben. Dieses ist dann noch wegen der Rücksetzphase für weitere 3 Takte belegt, insgesamt also für $s + vl + 6$ Takte. Für diesen Zeitraum ist eine weitere Nutzung als Ergebnisregister verboten, eine einmalige Verwendung als Operandenregister jedoch erlaubt.

Während der laufenden Operation können bereits im Ergebnisregister vorliegende Daten als Eingabe für eine zweite Vektoroperation dienen. Dieser als *Chaining* (Verkettung) bezeichnete Mechanismus wird automatisch durchgeführt und bleibt für den Benutzer transparent. Durch die gleichzeitige Nutzung von zwei Pipelines können so in jedem Zyklus zwei Ergebnisse produziert werden. Insbesondere die Bearbeitung komplexer Berechnungen wird so beschleunigt. Ein Wert eines Ergebnisregisters kann mit einem Takt Verzögerung nach seinem Eintreffen genutzt werden.

Ein Register kann unabhängig von einer möglichen Nutzung als Ergebnisregister immer genau einmal als Operandenregister verwendet werden. Ein Register darf jedoch nur dann als Ergebnisregister genutzt werden, wenn es zum Zeitpunkt der Befehlsinitiierung noch nicht genutzt wird.

Die bisherigen Betrachtungen setzen in Registern vorliegende Operanden voraus. Da jedoch vor einem Zugriff diese Elemente erst aus dem Speicher geladen werden, kann es trotz initialisierter Komponenten zu Wartezeiten kommen, wenn Elemente noch nicht verfügbar sind.

1.1.2 Die Hardware der Interprozessorkommunikation

Fünf identische *Registersätze* (Cluster) mit gemeinsamen Registern sind direkt mit allen CPUs verbunden. Jedes Cluster enthält

- 8 gemeinsame 24-Bit Adreßregister (Shared Address Registers),

- 8 gemeinsame 64-Bit Skalarregister (Shared Scalar Registers),
- 32 1-Bit Semaphorregister (Semaphore Registers).

Die Vergabe der Cluster wird vom Betriebssystem geregelt. Wie den Abbildungen 1.2 und 1.3 zu entnehmen ist, sind die gemeinsamen Adreßregister mit den A-Registern und die gemeinsamen Skalarregister mit den S-Registern der Prozessoren direkt verbunden und können mit diesen einzelne Werte austauschen. Die Semaphorregister sind zusätzlich an die skalaren Funktionseinheiten angeschlossen. *Test-and-Set*, *Clear* und *Set* sind unteilbare Operationen, die direkt auf den Semaphorregistern implementiert sind. Unteilbare (atomare) Operationen können nicht durch andere Prozesse unterbrochen werden. Sie werden zur Verwaltung gemeinsamer Ressourcen, Synchronisation und Prozeßkommunikation eingesetzt und bilden die Grundlage der Implementation von Multitasking auf der CRAY X-MP.

1.1.3 Der Speicherzugang

Jede CPU verfügt über 4 Ports, die als Schnittstelle zum gemeinsamen Hauptspeicher des Systems dienen. Der Datentransfer zwischen dem verschränkten Speicher und den Ports wird über ein mehrstufiges Verbindungsnetzwerk vorgenommen. Die folgenden Abschnitte befassen sich mit den einzelnen Komponenten.

Die Ports

Die 4 Ports A bis D einer CPU sind voneinander unabhängig und können gleichzeitig betrieben werden. Durch einen Speichertransferbefehl wird ein Port initialisiert. Es bearbeitet den Zugriff eigenständig und wird nach abgeschlossener Operation wieder freigegeben. Ein Prozessor kann nur durch seine eigenen Ports auf Daten aus dem gemeinsamen Speicher zugreifen. Einzige Ausnahme ist der Code-Fetch, der zu den eigenen noch die 4 Ports der benachbarten CPU nutzt, wobei die Prozessoren 0 und 1 bzw. 2 und 3 miteinander assoziiert sind. Der Code-Fetch hat höchste Priorität und unterbricht die laufenden Aktionen aller genutzten Ports. Um Konfliktfrei auf den Speicher zugreifen zu können, werden die 32 zu referierenden Bänke reserviert, so daß auch die übrigen Prozessoren auf die Hälfte des Speichers keinen Zugriff mehr haben. Nach seiner zwischen 7 und 10 Takten dauernden Bearbeitung werden die vorher auf den einzelnen Ports begonnenen Aktionen weitergeführt [Cra86].

Das Laden oder Speichern von Operanden über ein Port wird allgemein als *Speicherzugriff* oder einfach *Zugriff* bezeichnet. Hier soll der Begriff *Vektorzugriff* synonym für das Laden und Speichern von Vektoren über ein Port verwendet werden. Analog werden die Begriffe *Blockzugriff* für die Blockregister und *Skalarzugriff* für die Skalar- und Adreßregister definiert. Die Ports A und B sind ausschließlich für das Laden der Block- und Vektorregister reserviert. Dabei können B-Register nur über Port A und die T-Register nur über Port B geladen werden, während Vektorzugriffe beide Ports nutzen können. Alle Schreibzugriffe auf den Speicher sowie alle Zugriffe auf skalare Daten werden über das Port C abgewickelt. Vektor- und Blockregister

können mit einer Transferrate von einem Wort pro Takt geladen oder gespeichert werden, während skalare Daten nur jeden zweiten Takt übertragen werden können. Vektorzugriffe sind außer durch den Code-Fetch nicht unterbrechbar. Port D (I/O-Port) verbindet den Hauptspeicher mit der Ein-/Ausgabesektion und steht nicht für den Transport von Daten zwischen Prozessor und Speicher zur Verfügung. Einzige Ausnahme ist der Code-Fetch, bei dem das I/O-Port ebenfalls am Ladevorgang der Instruktionspuffer beteiligt ist. Ansonsten kann es unabhängig von den Operationen der übrigen Ports arbeiten, wird jedoch über die CPU programmiert.

Skalar- und Vektorzugriffe dürfen nicht verschränkt vorgenommen werden, da sonst die Reihenfolge der durch den Programmcode vorgegebenen Operationen gestört werden könnte. Ein Skalarzugriff setzt die Verfügbarkeit aller Ports A, B und C voraus und sperrt diese dann vier Takte lang für Block- oder Vektorzugriffe. Wird ein Skalarzugriff während dieser Zeit durch Konflikte verzögert, so können während der Verzögerung noch maximal zwei weitere Skalarzugriffe aufgesetzt werden. Ein vierter Zugriff kann nur dann aufgesetzt werden, wenn der durch den ersten Zugriff entstandene Konflikt beendet ist. Aufeinanderfolgende Skalarzugriffe erleiden keine Portkonflikte, da keine anderen Zugriffe der gleichen CPU aktiv sind.

Der Speicher

Der Hauptspeicher der CRAY X-MP hat eine Kapazität von 16 MW bei einer Wortbreite von 64 *Bit*. Die 64 Bänke sind zyklisch auf 4 Sektionen verteilt, auf deren Bedeutung im Zusammenhang mit dem Verbindungsnetzwerk eingegangen werden soll. Auf verschiedene Bänke kann gleichzeitig von unterschiedlichen Ports zugegriffen werden. Eine Bank ist durch einen Zugriff 4 Takte lang belegt. Während dieser *Bankzykluszeit* (bank cycle time, bank busy time) ist kein weiterer Zugriff auf diese Bank möglich. Alle anderen Bänke bleiben von diesem Zugriff unbeeinflußt.

Ein Zugriff auf den Hauptspeicher erfolgt immer wortweise. Die Daten aufeinanderfolgender Adressen befinden sich in aufeinanderfolgenden Bänken. Die letzten 6 Bit einer 24-Bit Adresse legen die Bankzugehörigkeit fest (siehe Abbildung 1.7). Jeweils vier aufeinanderfolgende Adressen und damit Bänke sind einer Sektion zugeordnet. Bit zwei und drei einer Adresse bestimmen die Sektionszuordnung. Wählt man für den Adreßbereich die Basisadresse 0 und bezeichnet man die Bänke mit $B_0 \ldots B_{63}$, die Sektionen mit $S_0 \ldots S_3$, so erfolgt bei Referenz einer Adresse A_i ein

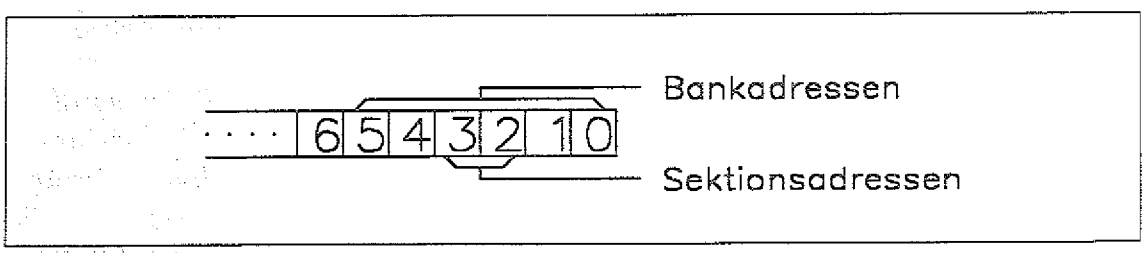


Abbildung 1.7: Adressierung der CRAY X-MP

Zugriff auf die Bank B_j in der Sektion S_k mit

$$j = i \, mod \, 64$$

$$k = (i \, mod \, 16) \, div \, 4.$$

Das Verbindungsnetzwerk

Da pro Takt eine Übertragung von mehr als einem Wert zwischen den Prozessoren und dem Hauptspeicher vorgesehen ist, muß die Verbindung als ein Netzwerk mit unabhängigen Datenpfaden ausgelegt werden. Im Idealfall ist die Verbindung durch ein *vollständiges Verbindungsnetzwerk* (Crossbar Network) realisiert, das jedes Port mit jeder Speicherbank verbindet. Bei p Ports und m Bänken würde diese Strategie jedoch eine Komplexität von $p * m$ Datenpfaden erfordern, die ihrerseits wiederum jeweils in der Wortbreite des betrachteten Systems ausgelegt sein müßten. Der durch solch ein vollständiges Netzwerk verursachte Aufwand ist in vielen Fällen technisch und wirtschaftlich nicht mehr vertretbar. In solchen Fällen stellen sogenannte *mehrstufige Verbindungsnetzwerke* (Multistage Interconnection Networks) eine preiswerte Alternative dar. Sie ermöglichen durch das Einfügen von Schalterstufen zwischen Speicher und Ports und eine Permutation der Verbindungen zwischen den einzelnen Stufen eine erhebliche Reduzierung des Verbindungsaufwands. Die Grundlagen solcher mehrstufigen Netzwerke werden in [Sie79, MGN79] behandelt.

In der CRAY X-MP ist ein zweistufiges Speicherverbindungsnetzwerk realisiert (Abbildung 1.8). Während ein vollständiges Netzwerk zwischen p Ports und m Speicherbänken einen Aufwand von $p * m$ Verbindungen erfordern würde, sind auf diese Weise bei s Sektionen lediglich $(p * s) + (s * m)/s = p * s + m$ Verbindungen notwendig. Der so erzielte Vorteil wird jedoch dadurch relativiert, daß nicht mehr jede Kombination von Verbindungen gleichzeitig geschaltet werden kann, so daß in Einzelfällen Kollisionen von Speicherzugriffswünschen bereits im Netzwerk auftreten können.

Das zweistufige Speicherverbindungsnetzwerk besteht aus den Stufen *Speicherzugangskontrolle* (Memory Path Selection) und *Sektionsverteilung*. Die Speicherzugangskontrolle besteht aus vier 4×4 *Kreuzschienenschaltern* (Crossbar Switches), benötigt also $4 * (4 * 4) = 64$ Verbindungen. Der Speicher ist in 4 Sektionen zu je 16 Bänken zusammengefaßt, die jeweils von einer CPU nicht parallel angesprochen werden können. Die Sektionsverteilung besteht aus vier 4×16 Kreuzschienenschaltern und regelt die Verteilung der Zugriffswünsche in einer Sektion auf die angesprochenen Bänke. Sie benötigt $4 * (4 * 16) = 256$ Verbindungen. Zusammengenommen werden also für das Verbindungsnetzwerk 320 Verbindungen gegenüber 1024 Verbindungen für das vollständige Netzwerk benötigt.

Da die Datenintegrität gewährleistet sein muß, wird in jedem Takt eine Kontrolle aller Zugriffe auf mögliche Konflikte durchgeführt. Ein auftretender Konflikt wird durch eine Berücksichtigung von Prioritäten der Ports aufgelöst. Ein Port, dem der Zugriff gestattet wird, erleidet keine Verzögerung, während andere beteiligte Ports

für die Dauer des laufenden Taktes angehalten werden. Die I/O-Ports haben immer die niedrigste Priorität. Die Konflikte werden in *Inter-CPU-Konflikte* (zwischen den Prozessoren) und *Intra-CPU-Konflikte* (durch einen Prozessor selbst verursacht) unterschieden. Bei letzteren wird als Entscheidungskriterium zur Konfliktauflösung der *Stride* (Schrittweite) eines Vektorzugriffs benutzt, also der Abstand zweier Adressen, auf die sukzessive zugegriffen wird.

- Ein *Bankkonflikt* (Bank Busy Conflict) tritt durch den Zugriff auf eine noch aktive Bank auf (sowohl Inter-CPU- als auch Intra-CPU-Konflikt). Alle Ports einer beteiligten CPU werden für die Dauer von ein bis drei Takten angehalten, bis die entsprechende Bank wieder frei ist.
- Ein *Simultankonflikt* (Simultaneous Bank Access) erfolgt durch gleichzeitigen Zugriff unterschiedlicher CPUs auf dieselbe, freie Bank (Inter-CPU-Konflikt). Die CPUs verfügen über eine zyklische Priorität, die alle vier Takte wechselt. Die CPU mit der höchsten Priorität darf zugreifen, während die übrigen Ports einen Zyklus angehalten werden. Auf einen Simultankonflikt erfolgt immer ein Bankkonflikt.
- Ein *Sektionskonflikt* (Section Conflict) entsteht durch den gleichzeitigen Zugriffswunsch mehrerer Ports einer CPU auf die gleiche Sektion (Intra-CPU-Konflikt).

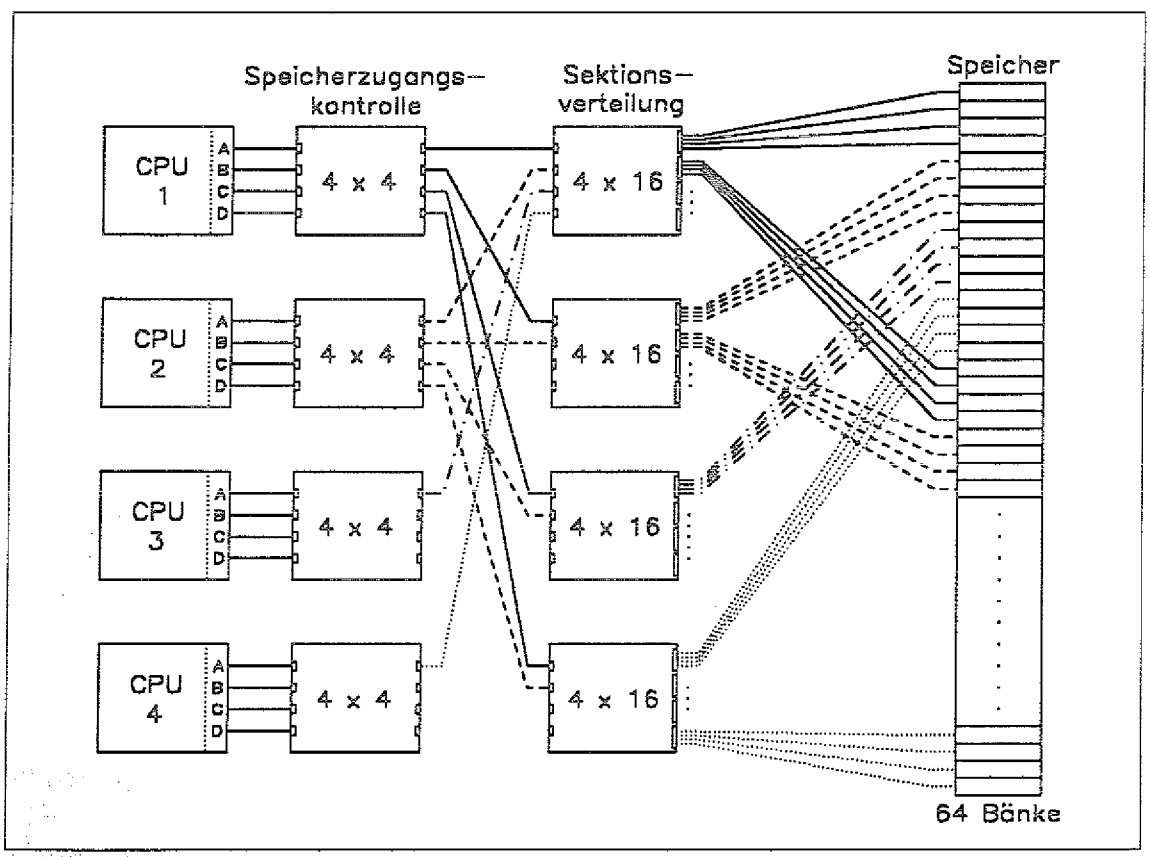


Abbildung 1.8: Speicherverbindungsnetzwerk der CRAY X-MP

- Ein Port, das mit ungeradem Stride auf den Speicher zugreift, erhält immer den Vorzug vor einem mit geradem Stride zugreifenden Port.
- Unterscheiden sich die Ports nicht durch ihre Zugriffsart, so wird dem Port mit früherer Aufsetzzeit (siehe Abschnitt 1.1.1) der Zugriff erlaubt.

Ein Port mit geringerer Priorität wiederholt den Zugriffsversuch im nächsten Zyklus.

Auswirkungen des Netzwerks auf Speicherzugriffe

Eine Vektorspeicheroperation belegt ein Port für $vl + 6$ Takte, wird aber schon nach $vl + 3$ Takten als beendet angesehen. Eine Vektorladeoperation dagegen belegt ein Port für $vl + 5$ Takte, dauert jedoch $vl + 17$ Takte. Die Dauer einer Operation bemißt sich an der Reservierungszeit der durch sie verwendeten Register. Da Lese- und Schreibzugriffe unterschiedliche Ports nutzen, erscheinen die unterschiedlichen Reservierungszeiten der Ports plausibel. Die voneinander abweichenden Angaben bezüglich der Reservierungszeiten und Ausführungszeiten sind nicht unmittelbar einsichtig und bedürfen einer Erklärung. Dazu ist eine genauere Betrachtung von Lade- und Speichervorgängen auf Netzwerken erforderlich.

Um Werte zwischen Speicher und Prozessoren zu übertragen, werden Informationen über Herkunfts- und Zieladresse sowie über die Art des Zugriffs (Laden/Speichern) benötigt. Diese vorerst nur in einem Port verfügbaren Informationen sind auch am Speicher zur Ermittlung der angesprochenen Adresse erforderlich, müssen also in einem ersten Schritt über das Netzwerk dorthin transportiert werden. Da die von einem Port übermittelten Informationen und die vom Speicher gesendeten Daten auf einem einfachen Netzwerk kollidieren würden, liegt die Vermutung nahe, daß ein symmetrisch aufgebautes Netzwerk für den Datentransport verwendet wird. Ein Teil ist in diesem Fall für den Transport vom Prozessor zum Speicher ausgelegt, der andere Teil realisiert die umgekehrte Richtung. Die Laufzeit der Signale und die für die Entscheidung an den Schalterstufen (im Fall eines mehrstufigen Netzwerks) benötigte Zeit verzögern den Datentransfer und damit die Ankunft der durch einen Lesezugriff angeforderten Daten im Prozessor. Unter dieser Annahme kann in Anlehnung an die Überlegungen zum Netzwerk davon ausgegangen werden, daß ein Port durch zwei physikalisch und logisch getrennte Komponenten realisiert ist, die für die unterschiedlichen Übertragungsrichtungen verantwortlich sind. Eine Bestätigung dieser Überlegungen ist in [ST91] zu finden.

Ein Vektorschreibzugriff kann als beendet angesehen werden, wenn die Reservierung des Operandenregisters aufgehoben wird, also wenn alle Werte vom Port übertragen worden sind. Diese Situation tritt nach $vl + 3$ Takten ein. Nach diesem Schreibzugriff ist das Port noch für weitere 3 Takte belegt. Eine Ladeoperation ist erst mit dem Eintreffen des letzten Wertes im Zielregister beendet. Bei einer Setup-Zeit von 3 Takten und einer Laufzeit der Daten von 14 Takten, die sich aus 4 Takten für den Zugriff auf die Adresse (*Chip Access Time*) und 10 Takten Laufzeit über das Netzwerk zusammensetzt [Cra86], wird der letzte Wert erst nach $vl + 17$ Takten in

das Zielregister geschrieben. Das Port zum Speicher hat jedoch nach $vl + 3$ Takten die letzte Information ausgegeben und kann dann 2 Takte später für den nächsten Zugriff freigegeben werden, während das zugeordnete Port vom Speicher noch Daten zu den Registern transportiert. Die Zugriffszeit für Adreß- und Datenregister wird in [Cra86] mit 14 Takten beziffert. Abweichend davon werden in [RR89] jedoch 13 Takte als experimentell ermittelt angegeben.

1.2 Die CRAY Y-MP8/832

Neben der bisher beschriebenen CRAY X-MP betreibt die KFA für das HLRZ (Höchstleistungsrechenzentrum) eine CRAY Y-MP, auf die sich die folgenden Angaben dieses Abschnitts beziehen. Da die CRAY Y-MP als Weiterentwicklung der CRAY X-MP betrachtet werden kann, sollen hier nur kurz die gegenüber ihrem Vorläufer geänderten Architekturmerkmale beschrieben werden. Die 8 Prozessoren mit einer Zykluszeit von $6\,nsec$ greifen auf einen $32\,MW$ großen, gemeinsamen Hauptspeicher zu, dessen 256 Bänke eine Zykluszeit von 5 Prozessortakten haben. Die Interprozessorkommunikation wurde auf 9 Registersätze erweitert. Durch die physikalische Anordnung der Cluster haben sich die Zugriffszeiten der gemeinsamen Register erhöht. Um den Hauptspeicher weiterhin mit einer realen Speicherverwaltung benutzen zu können, wurden alle Adreßregister auf $32\,Bit$ vergrößert. Der

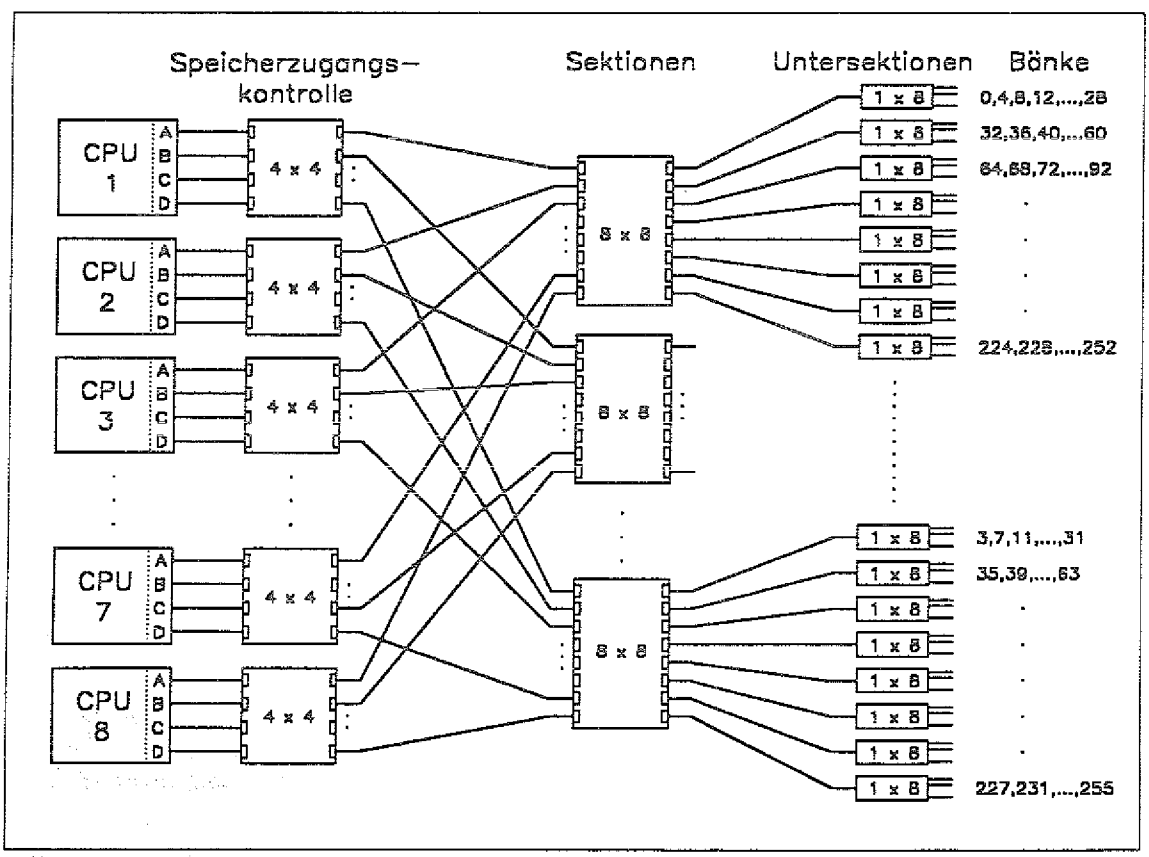


Abbildung 1.9: Speicherverbindungsnetzwerk der CRAY Y-MP (nach [ST91])

Adreßraum der CRAY Y-MP beträgt somit 4 Gigaworte. Wegen der erweiterten Adreßlänge mußten einige Befehle von zwei auf drei Parcel erweitert werden.

Im Gegensatz zur CRAY X-MP wird ein Port nicht mehr durch Bank- oder Simultankonflikte anderer Ports derselben CPU blockiert, wenn es nicht selbst einen Zugriffskonflikt erleidet. Auch das Laden des Befehlspuffers stört nicht mehr die Abarbeitung von Zugriffsvorgängen, da die Befehlspuffer nunmehr nur noch über das Port D geladen werden.

Die Vervierfachung der Bankanzahl und die Verdopplung der Prozessoranzahl erforderte eine Modifikation des Speicherverbindungsnetzwerks. Mit den *Untersektionen* (Subsections) wurde eine zusätzliche Stufe in das Speicherverbindungsnetzwerk eingefügt. Abbildung 1.9 zeigt schematisch das so entstandene dreistufige Verbindungsnetzwerk. Jeder der 4 Sektionen sind jetzt 8 Untersektionen zugeordnet, die wiederum je 8 Bänke enthalten. Wie bei der CRAY X-MP ist der gleichzeitige Zugriff auf Bänke einer Sektion durch mehrere Ports eines Prozessors nicht möglich. Alle Prozessoren können auf unterschiedliche Bänke einer Untersektion gleichzeitig zugreifen. Ein Pfad einer CPU zu einer Untersektion ist durch einen Zugriff jedoch für fünf Takte belegt, so daß dieselbe CPU in diesem Zeitraum nicht mehr auf dieselbe Sektion zugreifen kann. Ein Zugriff während dieser Zeit verursacht einen *Untersektionskonflikt* (Subsection Conflict) und betrifft nur die Speicherzugriffe des referierenden Ports. Aus Sicht einer CPU stehen daher in jedem Takt lediglich 32 unabhängige Wege zum Speicher zur Verfügung.

Die Zuordnung der Sektionen, Untersektionen und Bänke zu Adressen erfolgt so, daß bei aufeinanderfolgenden Zugriffen jede der drei Instanzen gewechselt wird. Bei einem Zugriff auf eine Adresse bestimmen die letzten 8 Bit die Bank, die letzten 5 Bit die Untersektion und die letzten beiden Bit die Sektion. Beginnt man die Numerierung bei Adresse 0, so berechnen sich die Nummern der angesprochenen Bank B_j, Sektion S_k und Untersektion U_l für eine Adresse A_i wie folgt:

$$j = i \, mod \, 256$$

$$k = i \, mod \, 4$$

$$l = i \, mod \, 32$$

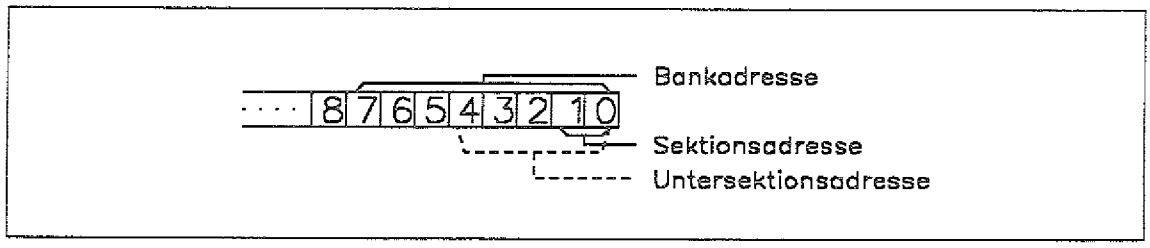


Abbildung 1.10: Adressierung der CRAY Y-MP

Beim Code-Fetch hat Port D Priorität über alle Ports einer CPU. Ansonsten werden Intra-CPU-Konflikte analog zur CRAY X-MP aufgelöst. Die Auflösung von Inter-CPU-Konflikten richtet sich nach einem festen Prioritätenschema, in das für jede Referenz CPU-Nummer, Sektion, Untersektion und Bank eingehen.

Kapitel 2

Parallelverarbeitung auf CRAY-Systemen

Der Einsatz mehrerer Prozessoren in einem System wird durch das zugrundeliegende Betriebssystem gesteuert. Meist werden sie in einer Mehrbenutzerumgebung im Multiprogramming-Betrieb eingesetzt. Dabei werden die einzelnen Programme der Benutzer auf unterschiedliche Prozessoren verteilt und laufen gleichzeitig ab. Auf diese Weise wird die Durchsatzleistung des Systems erhöht, und die gemeinsamen Ressourcen der Prozessoren werden effizienter genutzt. Der einzelne Benutzer hat im allgemeinen keinen Einfluß auf das Ablaufverhalten seines Jobs in einer solchen Umgebung. Durch die Nutzung von *Multitasking* kann ein Anwender parallele Strukturen in seinem Programm formulieren, diese gleichzeitig auf verschiedenen Prozessoren ausführen lassen und so die Laufzeit seines Programms reduzieren. Aus Sicht eines Programms ist dies auf unterschiedlichen Ebenen möglich:

- **Unterprogrammebene**: Einzelne Unterprogramme werden von verschiedenen Prozessoren bearbeitet. Die von jeder CPU zu leistende Arbeit wird durch die Zuteilung einer Codesequenz festgelegt und kann während der Abarbeitung nicht mehr verändert werden. Eventuell notwendige Kommunikation findet über implementierte Interprozessor-Kommunikationshardware statt.

- **Anweisungssequenzebene**: Im Programmcode aufeinanderfolgende Anweisungssequenzen werden auf unterschiedliche Prozessoren verteilt und gleichzeitig ausgeführt. Um unnötigen Kommunikationsaufwand zu vermeiden, sollten diese Sequenzen weitgehend voneinander unabhängig sein. Als Spezialfall kann die parallele Ausführung unterschiedlicher Schleifeniterationen angesehen werden. In diesem Fall bearbeiten auf allen Prozessoren Kopien des Programmcodes unterschiedliche Daten.

- **Anweisungsebene**: Einzelne, im Programmkontext aufeinanderfolgende Anweisungen werden parallel ausgeführt. Dies bedingt, daß die betroffenen Operationen voneinander unabhängig sind.

Voraussetzung für die Anwendung von Multitasking ist die Verfügbarkeit entsprechender Programmiersprachen bzw. Compiler und die Unterstützung durch das

jeweilige Betriebssystem. Darüber hinaus existiert mit der Hardware des Rechners eine für den Benutzer transparente Ebene, auf der gerade bei heutigen Supercomputern Funktionen zur parallelen Bearbeitung von einzelnen Instruktionen installiert sind. Eine übliche Methode ist der Einsatz von Pipelines, die zu erheblichen Speedups bei Berechnungen auf Vektoren führt.

Auf die Implementierung seitens der Hardware wurde für die CRAY-Systeme bereits im letzten Kapitel eingegangen. Neben dem Pipelining wird zusätzlich noch die überlappende und damit parallele Bearbeitung von unabhängigen Instruktionen unterstützt. Die folgenden Abschnitte vermitteln eine Übersicht über die auf CRAY-Systemen verfügbaren Multitasking-Konzepte und deren Umsetzung durch das Betriebssystem. Von zentralem Interesse ist dabei vor allem das Autotasking, das automatisch parallel ablaufende FORTRAN-Programme erzeugt.

2.1 CRAY-Multitasking

Die im letzten Abschnitt vorgestellten Konzepte werden auf CRAY-Computern unterschiedlich unterstützt. Das Betriebssystem UNICOS, als Multitasking-Multiuser-Betriebssystem auf UNIX basierend, verteilt auszuführende Programme auf die Prozessoren des Systems. Mit Macrotasking, Microtasking und Autotasking stehen dem Benutzer drei Multitasking-Konzepte zur Verfügung. Durch Nutzung von Pipelines bei Vektorberechnungen ist auch ein paralleles Bearbeiten von Daten zusätzlich zur parallelen Instruktionsverarbeitung möglich. Die Nutzung der Vektorverarbeitung erfolgt automatisch über den Compiler und kann nur in engem Rahmen vom Benutzer gesteuert werden; auf die parallele Instruktionsverarbeitung auf Prozessorebene kann er keinen Einfluß nehmen.

Da für die Betrachtungen im Rahmen dieser Arbeit lediglich das Autotasking-Konzept relevant ist, wird hier nur eine kurze Zusammenfassung der Möglichkeiten von Macrotasking und Microtasking gegeben. Eine tiefergehende Beschreibung dieser Konzepte liefern [Reg89, Nag90b]. Weitere Informationen können [Nag90a, Cra91a] entnommen werden.

2.1.1 Macrotasking und Microtasking

Mit der Einführung der CRAY X-MP/4 im Jahr 1984 stellte CRAY mit Macrotasking ein grobgranulares Multitasking-Konzept vor, das Parallelität auf Unterprogrammebene nutzt. Der Benutzer kann dieses Konzept durch spezielle Unterprogrammaufrufe der Multiprocessing-Bibliothek zur Laufzeit des Programms nutzen (siehe Abbildung 2.1). Die in diesen Aufrufen spezifizierten Unterprogramme des Benutzers werden dann während der Laufzeit als eigenständige Prozesse auf unterschiedliche Prozessoren verteilt. Die Multiprocessing-Bibliothek stellt Unterprogramme zur Taskgenerierung, Kommunikation und Synchronisation von Tasks zur Verfügung. Neben der Verantwortlichkeit für die korrekte Semantik des Programms

muß der Programmierer zusätzlich noch die *Balancierung*, d.h. die gleichmäßige Verteilung der Arbeitslast auf die parallelen Unterprogramme, beachten. Bei schlecht balancierter Arbeitslast können Effizienzverluste durch auf Synchronisation wartende Prozesse entstehen. Die mit Macrotasking entwickelten Programme sind wegen des expliziten Parallelismus nur mit erhöhtem Aufwand auf andere Systeme portierbar.

Seit 1986 ermöglicht das Microtasking-Konzept feingranulare Parallelisierung auf Anweisungssequenzebene. Durch Einfügen von Compiler-Direktiven der Form *CMICS <Option>* können parallele Regionen bestimmt werden. Der Präprozessor FMP fügt entsprechend dieser Direktiven zusätzlichen Code zur Initialisierung und Synchronisierung der parallelen Strukturen ein (siehe Abbildung 2.1). Für jede parallele Region erzeugt der Präprozessor zusätzlich ein Unterprogramm, das jeweils einem Prozeß zugeordnet wird. Der für Multitasking zusätzlich benötigte Aufwand wird durch Microtasking erheblich reduziert, so daß eine bessere Nutzung der Ressourcen erfolgen kann. Dies belegen auch Untersuchungen von parallelisierbaren Algorithmen der Graphentheorie und der linearen Algebra [HKN89]. Der Programmierer ist weitgehend von der Verantwortung für Synchronisation, Kommunikation und Balancierung befreit. Die mit Microtasking parallelisierten Programme sind portierbar, da die Direktiven von FORTRAN-Compilern als Kommentare interpretiert werden.

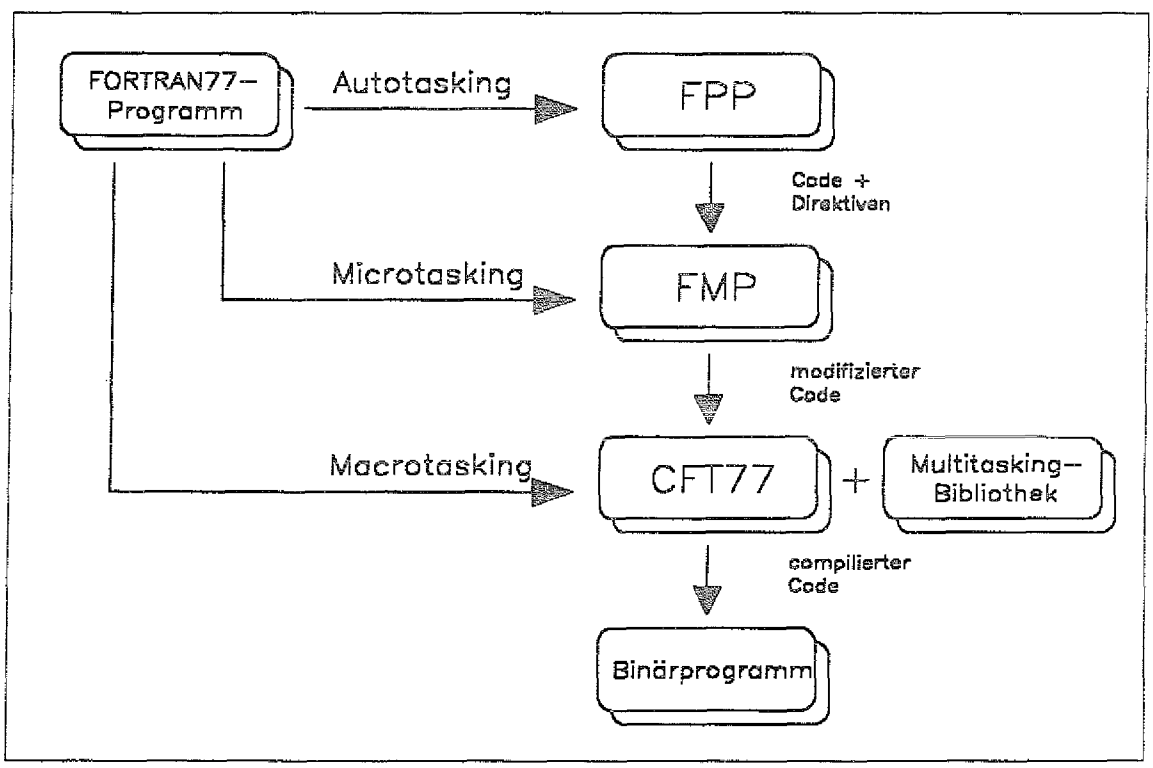


Abbildung 2.1: Umsetzung der unterschiedlichen Multitasking-Konzepte durch verschiedene Ebenen des Compiling-Systems CF77

2.1.2 Autotasking

Autotasking ist seit 1989 verfügbar und stellt für den Programmierer ein transparentes Interface zum feingranularen Multitasking dar. Ein Standard FORTRAN77-Programm wird automatisch auf DO-Loop-Ebene parallelisiert und vektorisiert. Zusätzlich zur automatischen Analyse hat der Benutzer noch die Möglichkeit, wie in Microtasking durch Direktiven parallele Strukturen zu spezifizieren. Autotasking unterstützt über die von Microtasking angebotenen Eigenschaften hinaus noch weitere Möglichkeiten. Parallele Strukturen werden mit weniger Aufwand implementiert als unter Microtasking. Eine parallele Region beginnt nicht bereits mit Eintritt in das umgebende Unterprogramm, sondern erst mit Eintritt in diese Region. Autotasking ist implementiert im CF77-Compiling-System (Cray Fortran77) und gliedert sich in die drei Phasen Abhängigkeitsanalyse, Übersetzung und Codegenerierung. Die beiden ersten Phasen sind durch die Präprozessoren FPP und FMP, die letzte durch den Compiler CFT77 realisiert. Abbildung 2.1 zeigt, wie die einzelnen Multitasking-Konzepte durch die unterschiedlichen Stufen des Compiling-Systems CF77 umgesetzt werden.

Abhängigkeitsanalyse (Dependence Analysis)

Das vom Benutzer erstellte FORTRAN-Programm wird durch den *FORTRAN Pre-Processor* (FPP) auf vektorisierbare und parallelisierbare Schleifen untersucht. Bei der Analyse von Schleifenstrukturen bevorzugt FPP Vektorisierung vor Parallelisierung, da Vektorisierung auf Prozessorebene implementiert ist und im Gegensatz zur Parallelisierung auf Programmebene kein zusätzlicher Aufwand entsteht. Aus diesem Grund werden einfache Schleifen grundsätzlich vektorisiert. Liegen geschachtelte Schleifen vor, so wird die Schleife der untersten Schachtelungstiefe (*Inner Loop*) vektorisiert, während die äußerste Schleife (*Outer Loop*) parallelisiert wird. In Abhängigkeit vom Ergebnis der Analyse fügt FPP entsprechende Direktiven in den FORTRAN-Code ein. Eine Übersicht und Funktionsbeschreibung der Autotasking-Direktiven gibt [Cra91a].

Eine Schleife kann grundsätzlich in der Art parallelisiert werden, daß bei sequentieller Bearbeitung aufeinanderfolgende Schleifendurchläufe (Iterationen) parallel bearbeitet werden. Dazu muß auf jedem Prozessor eine Kopie (*Redundant Code*) des Schleifenkörpers als eigener Prozeß ausgeführt werden. Die dynamische Zuordnung der einzelnen Iterationen zu jeweils freien Prozessen muß dabei global erfolgen. Eine parallele Bearbeitung ist nur dann sinnvoll, wenn die im Schleifenkörper bearbeiteten Daten für unterschiedliche Iterationen keine Abhängigkeiten aufweisen, da sonst der Aufwand für Kommunikation und Synchronisation zu groß wäre.

Übersetzungsphase (Translation)

Der um Direktiven erweiterte FORTRAN-Code wird vom *FORTRAN-Midst-Processor* (FMP) weiterverarbeitet. FMP modifiziert anhand der eingefügten Direktiven den Code einer als parallelisierbar gekennzeichneten Schleife und fügt entspre-

chende Autotasking-Kontrollstrukturen ein. Die dazu notwendigen Autotasking-Routinen sind in der Multiprocessing-Bibliothek zusammengefaßt. Abbildung 2.2 verdeutlicht schematisch die Umsetzung einer automatisch als parallelisierbar erkannten Schleife. Erkennt FMP eine entsprechende Direktive, so wird der Code wie folgt bearbeitet: Die Anweisungen der Schleife werden kopiert, und die beiden Versionen werden von Kontrollstrukturen umgeben. Die erste Version wird für den Gebrauch als parallele Region modifiziert (*Multitasked Loop*), während die zweite unverändert bleibt (*Unitasked Loop*). Die Kontrollstrukturen bestimmen zur Laufzeit, welche der beiden Versionen dann ausgeführt wird. Bedingungen für die parallele Ausführung sind seitens des Betriebssystems die Verfügbarkeit von weiteren Prozessoren und seitens des Programms eine genügend große Problemgröße, die eine parallele Bearbeitung rechtfertigt. Der so modifizierte Code wird als *Master* bezeichnet. Mit Hilfe einer weiteren Kopie der Multitasked Loop wird ein Unterprogramm erstellt. Dieses enthält neben der Initialisierung lediglich die parallel ausführbare Version der Schleife. Dieses als *Slave* bezeichnete Unterprogramm ist für die parallele Ausführung zu der entsprechenden Version im Master vorgesehen. Auf diese Weise existiert für jede parallele Region eine Version im Master und jeweils ein Slave. Zur Laufzeit des Programms wird während der Initialisierungsphase auf jedem Prozessor eine Version jedes Slaves gestartet.

Codegenerierungsphase (Code Generation)

Der autovektorisierende *CRAY FORTRAN77 Compiler* (CFT77) übersetzt schließlich den vom FMP modifizierten Code in Assemblerinstruktionen. Der Code wird optimiert, und die Vektorisierung innerer Schleifen wird aufgrund der von FPP eingefügten Direktiven vorgenommen.

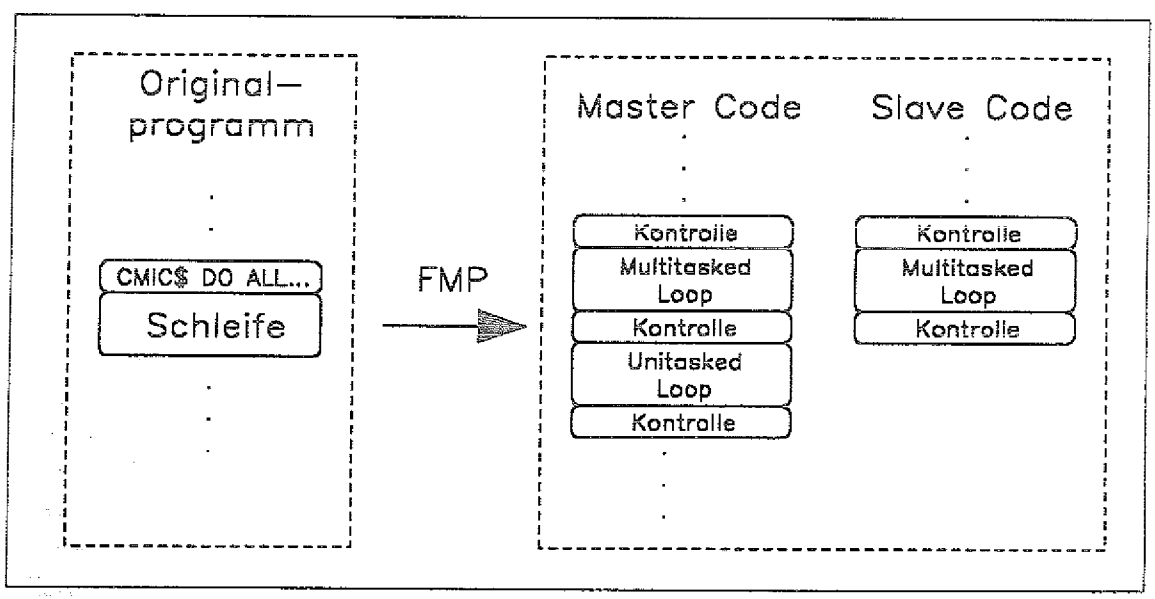


Abbildung 2.2: Modifikationen des Programmcodes durch FMP

2.2 Programmausführung durch Multitasking

Als *Multitasking-Programm* wird ein Programm bezeichnet, zu dessen Ausführung mehrere Prozessoren parallel eingesetzt werden. Es enthält *parallele Regionen* (parallele Blöcke), die von mehreren Prozessoren gleichzeitig bearbeitet werden, und *serielle Codesektionen*, die nur von einem Prozessor ausgeführt werden. Eine parallele Region kann entweder aus verschiedenen Codesequenzen bestehen, die auf die Prozessoren verteilt werden (*Partitioned Code*), oder sie enthält eine Codesequenz, die von allen Prozessoren gemeinsam bearbeitet wird (*Redundant Code*).

Zu Beginn eines Multitasking-Programms wird der Bibliotheksscheduler (*Library Scheduler*), der Teil der Multiprocessing-Bibliothek ist, gestartet. Während des Programmablaufs dient er als Schnittstelle zwischen dem Programm und dem Job-Scheduler des Betriebssystems. Ebenfalls zu Beginn des Programms werden alle Prozesse initialisiert. Man unterscheidet zwischen *Master-Prozeß* und *Slave-Prozessen*. Die Bezeichnung der Prozesse erfolgt jeweils nach den auf ihnen ausgeführten Codesequenzen. Der Master-Prozeß ist demnach für die Ausführung des Master-Codes und damit für die Bearbeitung der seriellen Codesektionen zuständig, während die Slave-Prozesse lediglich gemeinsam mit dem Master-Prozeß an der Bearbeitung paralleler Regionen teilnehmen. Die Slave-Prozesse bleiben so lange inaktiv, bis der Master-Prozeß eine parallele Region betreten und die entsprechenden Slaves angefordert hat. Nach erfolgter Bearbeitung einer parallelen Region werden alle Slave-Prozesse wiederum inaktiv und der Master-Prozeß setzt die Bearbeitung des seriellen Codes fort. Mit der Terminierung des Master-Prozesses werden auch alle Slave-Prozesse und der Bibliotheksscheduler beendet.

Die Leistung eines Multitasking-Programms wird durch das Scheduling der beteiligten Prozesse erheblich beeinflußt. Unter der zur Zeit der Messungen aktuellen Betriebssystemversion UNICOS 6.1 sind zwei unterschiedliche Scheduling-Konzepte verfügbar, die wahlweise verwendbar sind. Sie sind auf unterschiedliche Nutzung der Systeme ausgelegt.

Das erste Konzept ist aus der bisherigen Betriebssystemversion 5.1 übernommen und wird durch das Setzen der Systemvariablen MP_DEDICATED=1 aktiviert. Diese Version nutzt Multitasking mit Hilfe von Semaphorregistern für alle Slave-Prozesse eines Programms. Die Ausführung eines solchen Programms verläuft wie folgt: Zu Beginn der Bearbeitung werden alle Slave-Prozesse des Programms initialisiert und auf Semaphorregistern geparkt. Tritt der Master-Prozeß bei der Bearbeitung des Programms in eine parallele Region ein, so gibt er das entsprechende Semaphorregister frei, und alle diesem Register zugeordneten Slave-Prozesse können sofort an der Bearbeitung des parallelen Blocks partizipieren. Jeder Slave-Prozeß bearbeitet nun einen Teil der Arbeit und wird anschließend wiederum auf dem entsprechenden Semaphorregister geparkt. Nachdem die parallele Region von allen Slave-Prozessen verlassen wurde, kann der Master-Prozeß die nächste serielle Codesektion bearbeiten.

Zwei gewichtige Nachteile entstehen durch diese Vorgehensweise: Durch das Abfragen der Semaphorregister sind die Prozessoren, auf denen die Slave-Prozesse ausgeführt werden, auch während der Bearbeitung einer seriellen Codesektion durch den Master-Prozeß mit *Busy-Waiting* belegt, obwohl ihre Leistung nicht vom Programm benötigt wird. Damit stehen sie in einer Mehrbenutzerumgebung nicht für andere Aufgaben zur Verfügung, der Durchsatz des Systems verschlechtert sich. Wird ein Slave-Prozeß durch das Betriebssystem unterbrochen, so muß der Master-Prozeß so lange mit dem Beenden der parallelen Region warten, bis der entsprechende Slave-Prozeß zurückkehrt und seinen Teil der Arbeit beendet. Auf diese Weise können zusätzliche Wartezeiten des Master-Prozesses entstehen und die Ausführungszeit des Programms wird verlängert.

Das zweite Konzept, genannt *Cooperative Parallel Interface* [BLR91], wurde für CRAY-Systeme neu entwickelt und ist an das Konzept des MACH-Betriebssystems [Bla90] angelehnt. Die Slave-Prozesse werden nach der Initialisierung nicht mehr auf Semaphorregistern geparkt, sondern nach einer Sicherung ihres Prozeßkontextes suspendiert. Tritt der Master-Prozeß in eine parallele Region ein, so wird eine Variable in der Kontextstruktur aller dieser Region zugeordneten Slave-Prozesse gesetzt. Das Betriebssystem überprüft jede 60stel Sekunde mit dem Interrupt *Minor Clock Cycle* alle Kontexte der Slave-Prozesse auf diese Variablen und startet die Prozesse, deren Variable gesetzt ist. Wird ein Prozessor durch das Betriebssystem angefordert, so erhält der laufende Prozeß die Möglichkeit, seinen Kontext vor Prozessorentzug wieder in den Benutzerbereich zurückzuspeichern. Ein beliebiger anderer Prozeß kann nun diese Daten aufnehmen und mit dem gleichen Kontext weiterarbeiten.

Diese Konstruktion erlaubt einerseits die Vergabe der Prozessoren an andere Benutzer, solange diese nicht benötigt werden, und verhindert andererseits unnötige Wartezeiten durch suspendierte Slave-Prozesse. Es ist erkennbar, daß damit die beiden oben genannten Probleme beseitigt werden konnten. Die ersichtlichen Vorteile müssen allerdings mit leichten Leistungseinbußen bezahlt werden. Durch die Überprüfung der Variablen in den Kontextstrukturen durch das Betriebssystem entsteht einerseits zusätzlicher Aufwand für die Ausführung des Interrupts Minor Clock Cycle und andererseits eine Verzögerung des Eintretens der Slave-Prozesse in die Bearbeitung der parallelen Region. Vor dem Entzug eines Prozessors muß die Möglichkeit gegeben werden, den Prozeßkontext zu sichern, so daß keine direkte Unterbrechung mehr erfolgen kann. Eine ausführliche Beschreibung, die den Rahmen dieser Arbeit bei weitem überschreiten würde, kann [BLR91] entnommen werden.

Das erste Konzept ist auf eine Nutzung des Systems im dedizierten Betrieb zugeschnitten. Durch die Verwendung von Semaphorregistern für die Kommunikation werden schnelle Prozeßwechsel ermöglicht, so daß einem dediziert laufenden Benutzerprogramm schneller Zugriff auf alle Prozessoren garantiert ist. Das Cooperative Parallel Interface ermöglicht in einer Mehrbenutzerumgebung die Nutzung von durch Multitasking zeitweise nicht benötigten Prozessoren für andere Anwendungen. Der dabei entstehende Mehraufwand durch Betriebssystemunterbrechungen und Prozeßwechsel steht jedoch in keinem Verhältnis zu der durch diese Methode gewonnenen

CPU-Zeit. Der Durchsatz in einer Multiuser-Umgebung kann mit Verwendung des zweiten Konzepts deutlich gesteigert werden.

Kapitel 3

Meßmethodik

Bei der Aufgabe, die Leistung eines Systems zu bestimmen, fällt der Auswahl von Meßmethodik und Meßgrößen besondere Bedeutung zu. Grundsätzlich stehen die drei Methoden

- analytische Modellierung
- Simulation
- Messung

zur Verfügung, wobei eine gleichzeitige Realisierung von wenigstens zwei Verfahren zur gegenseitigen Validierung der Ergebnisse günstig erscheint. Analytische Modelle gewähren im allgemeinen guten Einblick in die Effekte verschiedener Parameter und ihrer Abhängigkeiten. Ihre Genauigkeit ist dagegen gering, da viele Einschränkungen getroffen und einige Annahmen gemacht werden müssen, um die Komplexität für eine analytische Betrachtung zu reduzieren. Eine Simulation kann dagegen mehr Details einschließen und benötigt weniger Annahmen. Die erzielbare Genauigkeit ist vor allem von den Kenntnissen über das betrachtete System abhängig [Jai91]. Beide Methoden können auch für die Betrachtung von nicht verfügbaren Systemen gut eingesetzt werden. Eine Messung erfordert dagegen ein konkretes Testsystem. Ihre Genauigkeit ist im wesentlichen abhängig von der für die Ermittlung der Meßgrößen zur Verfügung stehenden Meßtechnik. Zusätzlichen Einfluß auf die Meßergebnisse nehmen für den Analytiker nicht sichtbare Hardware- und Betriebssystemeigenschaften sowie Wechselwirkungen mit anderen Benutzern in einer Multiuser-Umgebung.

Für die vorliegende Untersuchung von Speicherzugriffskonflikten werden neben der analytischen Betrachtung auch Messungen durchgeführt. Für die CRAY X-MP wird darüber hinaus eine Simulation vorgestellt, mit der eine im Vergleich zum Modell bessere Berücksichtigung der Komplexität des Systems und der vielfältigen Einflüsse ermöglicht wird. Die CRAY X-MP zeigt sich als gut geeignet für den Entwurf einer Simulation, da die benötigten Informationen weitgehend vorhanden sind. Besondere Bedeutung bei den Messungen an den Systemen kommt der Wahl der Meßverfahren zu.

Der folgende Abschnitt beschäftigt sich mit allgemein verfügbaren und hier im speziellen anwendbaren Meßmethoden. Anschließend wird ein geeignetes Meßverfahren entwickelt, das zusammen mit einem Modellproblem in die Entwicklung eines Programms zur Erzeugung von erwünschten Zugriffsmustern durch parallele Speicherzugriffe mehrerer Prozessoren eingeht. Dieses FORTRAN-Programm soll im folgenden auch als *Meßprogramm* bezeichnet werden. Es wird mit Hilfe der in Kapitel 1 und Kapitel 2 vorgestellten Grundlagen analytisch auf sein erwartetes Laufzeitverhalten untersucht. Mit Hilfe des durch diese Untersuchung gewonnenen Modells läßt sich eine Erwartung für das Zugriffsverhalten des Meßprogramms angeben. Ausführliche Überlegungen zur Realisation der Simulation erfolgen separat in Kapitel 4.

3.1 Meßwerkzeuge

Zur Messung der Systemleistung existieren je nach Art der benötigten Informationen unterschiedliche Konzepte. *Hardware-Monitore* können *Ereignisse* (Events) auf Prozessorebene aufzeichnen, *Software-Monitore* ermöglichen eine Registrierung von Zustandsänderungen auf Programmebene. *Hybrid-Monitore* vereinen die Fähigkeiten von Hardware- und Software-Monitoren und ermöglichen eine Zuordnung der Ereignisse verschiedener Ebenen. Darüber hinaus lassen sich spezielle Informationen auf Programmebene durch gezielte *Instrumentierung* von Anwenderprogrammen gewinnen. Im folgenden soll kurz auf die Konzepte von Hardware- und Software-Monitoren eingegangen werden. Weitere Informationen können [Rei90, SKB89] entnommen werden.

Ein Hardware-Monitor erkennt bestimmte Ereignisse in einem Rechnersystem, indem er die Signale der Rechner-Hardware überwacht. Er verfügt dazu über eine eigene Elektronik, so daß er ohne Beeinflussung des Systems Ereignisse erkennen und aufzeichnen kann. Diese Aufzeichnung erfolgt üblicherweise durch Zählregister. Die Anzahl der gleichzeitig protokollierbaren Events ist durch die Anzahl der verfügbaren Zähler begrenzt. Oft ist es möglich, mehrere Events alternativ von einem Zähler aufzeichnen zu lassen.

Man unterscheidet interne und externe Hardware-Monitore. Interne Hardware-Monitore sind vom Hersteller in das System integriert und werden über dieses gesteuert. Dem stehen externe Hardware-Monitore gegenüber, die meist von Fremdherstellern entwickelt sind und erst noch an die zu betrachtenden Rechner angeschlossen und auf diese abgestimmt werden müssen. Die Hardware eines solchen Monitors muß dementsprechend so ausgelegt sein, daß tatsächlich alle für die Untersuchung relevanten Vorgänge beobachtet werden können. Solche externen Hardware-Monitore werden heute vor allem zur Überwachung von Netzwerken eingesetzt. Im Zuge der Geschwindigkeitssteigerung moderner Hochleistungsrechner und der damit verbundenen Integration der Hardware ist eine Verwirklichung externer Monitore für solche Systeme nicht weiter realisierbar. Für heutige Supercomputer sind daher externe Hardware-Monitore nicht verfügbar.

Eine Alternative zu nicht immer vorhandenen Hardware-Monitoren stellen Software-Monitore dar. Ein Software-Monitor ist ein Meßprogramm, das den Zustand eines Meßobjekts überwacht und auf demselben Computer abläuft. Als Meßobjekte sind neben einzelnen Programmen beispielsweise auch ganze Betriebssysteme denkbar. Die Betrachtung soll sich jedoch im folgenden auf die Beobachtung von Anwenderprogrammen beschränken. Da der Monitor gleichzeitig zum zu untersuchenden Programm abläuft, benötigt er auch die Ressourcen des Systems und führt zu zusätzlichem Aufwand (*Overhead*), der eventuell signifikante Auswirkungen auf das Anwenderprogramm ausübt. Prinzipiell existieren zwei Verfahren, um Informationen über das Meßobjekt zu erlangen: die *Sampling-Methode* und das *Event-Tracing*.

Beim Sampling läuft der Monitor auf dem System im Hintergrund zum zu untersuchenden Programm und erfragt periodisch den Status der Anwendung. Anfallende Informationen werden für eine spätere Auswertung gespeichert. Als problematisch zeigt sich hier eine Wechselwirkung zwischen der Genauigkeit der Messung und der Verfälschung der Meßwerte durch zu hohen Meßaufwand. Während bei geringer Sampling-Frequenz ein Zustandswechsel übersehen werden kann, wenn dieser kürzer als eine Periode andauert, kann sich ein zu häufiges Abtasten in vielfältiger Weise auf die Ausführung des Programms auswirken (Laufzeitverzögerung, Speicherplatzbedarf). In beiden Fällen ist eine Rekonstruktion des Pogrammablaufs unter normalen Bedingungen nur noch schwer möglich.

Das Event-Tracing bietet eine Möglichkeit, ein vollständiges Ablaufprotokoll (*Trace*) mit hoher Zeitauflösung zu erhalten. Der Monitor wird dabei nicht in periodischen Zeitintervallen, sondern durch Ereignisse im Verlauf der Bearbeitung des Programms aktiviert. Ereignisse sind beispielsweise der Aufruf eines Unterprogramms oder die Anforderung eines weiteren Prozessors durch das Programm. Die Verfolgung solcher Events geschieht im allgemeinen automatisch in Abhängigkeit von der Einstellung des Monitors. Daneben können aber auch vom Benutzer spezifizierte Ereignisse (User-Events) definiert und in den Quelltext eingetragen werden (Instrumentierung). Diese Einträge bestehen aus Unterprogrammaufrufen, die zur Laufzeit bestimmte Aktionen des Monitors anstoßen. Alle Events werden protokolliert, mit einer Zeitmarke (*Time-Stamp*) versehen und für eine weitere Auswertung gespeichert. Auch bei dieser Methode läßt sich Overhead nicht vermeiden, der proportional zu der Anzahl der vom Programm erzeugten Events ansteigt. Der Programmierer hat jedoch gegenüber dem Sampling einen größeren Einfluß auf die Art und Menge der protokollierten Ereignisse.

Eine Instrumentierung kann für spezielle Probleme auch unabhängig von einem Monitor vorgenommen werden. Oft existieren Werkzeuge, die für bestimmte Aufgaben eine solche häufig sehr aufwendige Instrumentierung der Software automatisch durchführen können. Der Benutzer ist dann jedoch verantwortlich für die Verwaltung der im Programmverlauf gewonnenen Daten. Die auf diese Weise entstehenden Protokolldateien können auch für kurze Programme bereits erhebliche Ausmaße annehmen.

Ist die Erfassung von Meßwerten über vorhandene Monitore nicht möglich, so müssen Meßmethoden entwickelt werden, die in Verbindung mit analytischer Modellierung und Simulation durch Computerprogramme zur Ermittlung der gewünschten Meßgrößen beitragen. Eine Instrumentierung zu untersuchender Programme kann in solchen Fällen zu Informationen über das Zeitverhalten von Modellproblemen führen. Bei guter Übereinstimmung des gemessenen Zeitverhaltens mit Ergebnissen aus Modell und Simulation liegt die Vermutung über die weitgehende Übereinstimmung der Modellvorstellungen mit der Realität nahe. In diesem Fall können durch diese Verfahren weitere Informationen gewonnen werden.

3.2 Meßwerkzeuge der CRAY-Systeme

Für die vorliegende Fragestellung werden Informationen über Zugriffe einzelner Prozessoren auf bestimmte Speicherbänke benötigt. Dabei auftretende Zugriffskonflikte sollen erkannt und den betroffenen Prozessoren zugeordnet werden. Dazu sind folgende Informationen notwendig:

- der Zeitpunkt des Auftretens eines Zugriffskonflikts;
- die Art des Zugriffskonflikts;
- die von einem Zugriffskonflikt betroffenen Prozessoren.

Die folgenden Abschnitte geben eine Übersicht über Meßwerkzeuge auf den Systemen CRAY X-MP und CRAY Y-MP und diskutieren die Verwendbarkeit der verfügbaren Meßwerkzeuge für die vorliegende Zielsetzung.

3.2.1 Der Hardware Performance Monitor

Mit dem *Hardware Performance Monitor* (HPM) verfügen sowohl die CRAY X-MP wie auch die CRAY Y-MP über einen internen Hardware-Monitor, der über acht 46-Bit Zählregister pro Prozessor zur Erkennung und Aufzeichnung unterschiedlicher Hardware-Ereignisse verfügt. Einzelne Zähler können entsprechend ihrer Aufgabe bis zu drei Ereignisse pro Takt gleichzeitig erkennen. Jeder Zähler ist in der Lage, alternativ vier verschiedene Events zu registrieren. Gleichzeitig erkennbare Events sind zu Gruppen zusammengefaßt. Die so entstehenden Gruppen werden mit *Program Totals*, *Hold-Issue-Conditions*, *Memory Activity* und *Instruction Summary* bezeichnet. Die Auswahl der Gruppe und das Auslesen der Zähler in Skalarregister werden über Assemblerinstruktionen gesteuert. Die Gruppe Program Totals enthält Informationen zur erzielten Rechenleistung, Hold-Issue-Conditions gibt Auskunft über bei der Ausführung entstandene Wartezeiten durch belegte Ressourcen. In der Gruppe Memory Activity wird die Anzahl der Zugriffe und Konflikte bei Vektor-, Block- und Skalarzugriffen getrennt voneinander angegeben. Instruction Summary faßt schließlich Informationen über die Häufigkeit einzelner Befehlsklassen (z.B. Sprungbefehle, Vektorbefehle) zusammen. Der HPM akkumuliert

alle Meßwerte über die gesamte Laufzeit eines Programms und alle an der Bearbeitung beteiligten Prozessoren zusammen. Er ist über das UNICOS-Systemkommando *hpm* zugänglich.

3.2.2 Der Multitasking History Trace Buffer

Der in die Klasse der Software-Monitore einzuordnende *Multitasking History Trace Buffer* (MHTB) zeichnet für jedes ablaufende Multitasking-Programm automatisch ohne Instrumentierung des Programmcodes Aktionen im Zusammenhang mit Kommunikation und Synchronisation von Prozessen fortlaufend in einem Ringpuffer auf. Der Ringpuffer kann auf Sekundärspeicher ausgelagert werden, so daß der Anwender ein vollständiges Ablaufprotokoll mit den von ihm spezifizierten Ereignissen erhält. Insgesamt stehen 38 Standard-Events zur Verfügung, von denen beliebig viele gleichzeitig verfolgt werden können. Zusätzlich erhält der Benutzer über einige Unterprogrammaufrufe die Möglichkeit zur Initialisierung und Spezifizierung von maximal 64 User-Events. Neben einer Zeitmarke und einer Prozeßnummer enthält jede Aufzeichnung eines Events dessen Nummer, eine damit fest verbundene Zeichenkette und einen Eintrag, in den der Benutzer den Wert einer Variablen während des Programmablaufs ablegen kann.

3.2.3 Werkzeuge zur Laufzeitmessung

Bei der Untersuchung des Laufzeitverhaltens von Multitasking-Programmen und speziell der Laufzeit von parallel ausgeführten Codesequenzen ist eine Zuordnung von Laufzeit und ausführender CPU sinnvoll. Die durch Zugriffskonflikte entstehenden Verzögerungen bewegen sich in der Größenordnung von nur wenigen Takten pro Zugriff. Daher werden möglichst genaue Zeitmessungen benötigt, die nur geringe Auswirkungen auf den Verlauf eines Programms haben dürfen. Bei der folgenden Betrachtung verfügbarer Meßmöglichkeiten sind daher Meßgenauigkeit und zusätzlich verursachter Meßaufwand sowie dessen Rückwirkung auf das Multitasking-Programm besonders hervorgehoben.

Alle CRAY-Routinen zur Zeitmessung basieren auf dem Auslesen oder Setzen der Echtzeituhr und berechnen weitere Informationen aus vorgegebenen Systemeinstellungen (z.B. Datum, Zeit). Die Echtzeituhr ist ein Zählregister großer Wortbreite, die mit jedem Systemtakt inkrementiert wird. Zur Bestimmung der Laufzeiten von Multitasking-Programmen stehen die Routinen TSECND, ICPUSED und IRTC zur Verfügung. TSECND akkumuliert die von allen Prozessoren eines Multitasking-Programms verbrauchte Zeit und erweist sich daher für eine Messung als prinzipiell nicht geeignet. ICPUSED bestimmt die seit dem Start des aufrufenden Benutzerprozesses vergangene, vom Benutzer verbrauchte CPU-Zeit (Benutzerzeit) des Prozesses in Takten. Während der Entwicklungsphase hat sich jedoch gezeigt, daß neben einem Overhead von im Mittel etwa 250 Takten pro Aufruf von ICPUSED diese Routine nicht immer zuverlässige Werte liefert. Eine weitere Möglichkeit zur Zeitmessung ist durch die FORTRAN-Routine IRTC (Integer Real Time Clock) gegeben, die den aktuellen Stand der Echtzeituhr als Integer-Wert ermittelt. Die Rückwirkung

der Routine IRTC auf die Laufzeit des aufrufenden Programms kann vernachlässigt werden, da ein Aufruf von IRTC direkt vom Compiler in die Assembleranweisung RT (Realtime Clock) übersetzt wird. RT liest die Echtzeituhr des Systems in ein Skalarregister ein und benötigt dafür lediglich einen Takt. Auf diese Weise ist eine genaue Messung der zwischen zwei Aufrufen vergangenen Realzeit möglich. Zu beachten ist jedoch, daß diese Realzeit neben der Benutzerzeit auch noch durch Betriebssystemunterbrechungen verursachte Systemzeiten enthalten kann. Die Voraussetzung zur Anwendung dieser Methode ist also die Verfügbarkeit einer Meßumgebung, in der Systemunterbrechungen nicht unkontrolliert auftreten können.

Die Zuordnung eines Prozesses zur jeweils ausführenden CPU erweist sich auf den CRAY-Systemen als nicht realisierbar, da das Betriebssystem keine entsprechende Meßroutine zur Verfügung stellt. In Abschnitt 5.3 wird gezeigt, daß für die hier vorgelegten Betrachtungen eine Aussage über die eindeutige Zuordnung eines Prozesses zu einer CPU für seine gesamte Dauer getroffen werden kann. Diese nicht allgemeingültige Aussage soll daher für alle folgenden, das Meßprogramm betreffenden Betrachtungen als Voraussetzung dienen. Unter UNICOS 6.1 ist weiterhin eine Trennung der Begriffe Prozeß und *Thread* vorgesehen. Ein Thread, mit dem auch die bearbeiteten Daten verbunden sind, kann nach der Unterbrechung des ihn bearbeitenden Prozesses von einem anderen Prozeß aufgenommen werden. Mit der Einstellung MP_DEDICATED=1 kann dieses Vorgehen verhindert werden, so daß die Zuordnung zwischen einzelnen Prozessen und bearbeiteten Daten eindeutig ist. Bei Bearbeitung eines Multitasking-Programms mit mehreren Prozessoren läßt sich ein auf einem Prozessor entstehender Meßwert damit eindeutig einem Prozeß, mit der obengenannten Voraussetzung also auch einem Prozessor, zuordnen. Die C-Routine GETPID (GET Process ID) ermittelt die dem Aufruf zugeordnete Prozeßnummer, benötigt jedoch für einen Aufruf in Abhängigkeit von der durch Multitasking verwendeten Prozessoranzahl im Mittel etwa 2300 Takte zusätzlichen Aufwand.

3.2.4 Auswahl der Meßmethode

Für die vorliegenden Untersuchungen sind Angaben über Speicherzugriffskonflikte in ihrem Kontext mit betroffenem Prozessor und Zeitpunkt des Auftretens wünschenswert. Solche Konflikte sind Hardware-Ereignisse und – außer durch den HPM – nicht direkt meßbar. Dieser kann jedoch lediglich Zugriffskonflikte zählen und keinerlei Angaben über den gewünschten Kontext liefern. Der HPM scheidet in diesem Fall als Meßmethode aus. Speicherzugriffskonflikte führen aber in jedem Fall zu einer Verzögerung für einen oder mehrere Takte. Diese wiederum können sich auf Laufzeiten von Programmen auswirken, so daß auf diesem Wege eine indirekte, quantitative Bestimmung von durch Zugriffskonflikte erzeugten Verzögerungen möglich ist. Implementiert man nun ein Modellproblem, das in vorbestimmter Weise Zugriffe auf den Speicher realisiert, sowohl als Meßprogramm wie auch als Computersimulation, so können die mit dem Meßprogramm ermittelten Laufzeiten zwecks Validierung mit den von der Simulation errechneten Laufzeiten verglichen

werden. Mit Hilfe der Simulation können also detaillierte Aussagen über Zugriffskonflikte gewonnen werden. Zur Durchführung der Laufzeitmessungen werden geeignete Meßverfahren benötigt, die einerseits eine hohe Meßgenauigkeit bieten und andererseits möglichst geringe Rückwirkungen auf das Meßprogramm haben. Der MHTB scheint zunächst ein geeignetes Meßinstrument zu sein, da er in einfacher Weise die Zuordnung zwischen Meßzeit und Prozeßnummer realisiert. Durch das Auslösen eines Benutzer-Events entstehen jedoch Overhead-Zeiten zwischen 230 und 270 Takten, zusätzlich werden durch das Sichern des Pufferinhalts auf Sekundärspeicher nicht reproduzierbare Effekte erzeugt. Wegen dieser zu großen Rückwirkungen scheidet der MHTB als Meßmethode aus. Als Meßwerkzeug zur Bestimmung der Laufzeit eignet sich die Routine IRTC, die lediglich einen Takt zusätzlichen Aufwand erzeugt und damit exakte Ergebnisse liefern kann. Als problematisch stellt sich die Zuordnung eines Meßwerts zu einem Prozeß heraus. Eine Verwendung der Routine GETPID ist wegen des zu hohen Meßaufwands nicht möglich. Hier konnte jedoch im Verlauf der Entwicklung des Meßprogramms ein Verfahren entwickelt werden, das mit nur geringem zusätzlichen Aufwand eine Zuordnung von mit IRTC gemessenen Laufzeiten zu ausführenden Prozessen gestattet.

Im Verlauf der Programmentwicklung des Meßprogramms hat sich herausgestellt, daß durch Betriebssystemeinflüsse einzelne Messungen nachhaltig gestört werden können. Bei der Auswertung können solche Effekte, sofern sie unerkannt bleiben, zu Fehlinterpretationen führen. Um die Reproduzierbarkeit der Meßergebnisse zu sichern, werden daher zu unterschiedlichen Zeitpunkten Referenzmessungen durchgeführt. Zusätzlich werden nach dem folgenden Schema jeweils vergleichende Messungen mit Hilfe der Routine ICPUSED vorgenommen. Direkt aufeinanderfolgend finden zwei Messungen statt. In der ersten wird lediglich mit der Routine IRTC gemessen, während in der zweiten Messung zusätzlich eine Zeitbestimmung mit ICPUSED vorgenommen wird. Die Vektorschleife des Meßprogramms wird zusätzlich zu Aufrufen von IRTC noch mit Aufrufen von ICPUSED eingeschachtelt. Durch diese zusätzliche Zeitnahme entsteht ein Overhead, der die zweite Messung verfälscht. Weichen jedoch die Ergebnisse von IRTC und ICPUSED der zweiten Messung nur durch einen weitgehend konstanten Wert (Overhead von ICPUSED) voneinander ab, so kann davon ausgegangen werden, daß diese Messung nicht durch Betriebssystemeinflüsse gestört wurde. Unterscheiden sich darüber hinaus die Ergebnisse von IRTC der ersten Messung nicht signifikant von denen der zweiten, so läßt sich aus der Kombination der Aussagen mit großer Wahrscheinlichkeit ein Einfluß durch das Betriebssystem für die eigentliche Messung ausschließen.

3.3 Das Modellproblem

3.3.1 Ein Modellalgorithmus für den parallelen Zugriff

Das Modellproblem soll in vorherbestimmter Weise mit mehreren Prozessoren parallele Speicherzugriffe durchführen. Dabei ist eine Einstellung von verschiedenen Zugriffsmustern wünschenswert. Folglich muß das Modellproblem die Eigenschaf-

ten der Vektorisierung zur Realisierung von einstellbaren Speicherzugriffen und der Parallelisierung für den gleichzeitigen Zugriff mehrerer Prozessoren miteinander verbinden.

Betrachtet man eine Matrix

$$(a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix},$$

so läßt sich eine Addition aller Elemente einer Zeile i, $i \in \{1 \ldots n\}$, auf einen skalaren Wert

$$c_i = \sum_{j=1}^{m} a_{ij}$$

als Vektorreduktion des Zeilenvektors $\vec{a_i}$ auffassen. Soll nun jede Zeile der Matrix (a_{ij}) auf diese Weise bearbeitet werden, so kann diese Berechnung durch folgenden FORTRAN-Programmkern implementiert werden:

```
      :
    DO 10 I = 1, N
       C(I) = 0
       DO 20 J = 1, M
          C(I) = C(I) + A(I,J)
20     CONTINUE
10  CONTINUE
      :
```

Mit der Möglichkeit der Autovektorisierung und Autoparallelisierung, dem Autotasking (siehe Abschnitt 2.1.2), bietet das CF77-Compiling-System eine elegante Möglichkeit, auf einfache Weise die geforderten Eigenschaften des Modellproblems umzusetzen. Bearbeitet man diesen Programmkern mit dem CF77-Compiling-System, so wird die innere J-Loop vektorisiert und die äußere I-Loop parallelisiert. Unterschiedliche Prozessoren ermitteln also gleichzeitig Ergebnisse unterschiedlicher Zeilenvektorreduktionen bzw. unterschiedlicher Durchläufe (Iterationen) bezüglich der I-Loop. Damit greifen im optimalen Fall alle Prozessoren gleichzeitig auf die Matrix (a_{ij}) zu. Da lediglich Lesezugriffe durchgeführt werden, sind keine Maßnahmen zur Verhinderung des gleichzeitigen Zugriffs erforderlich.

Eine Matrix wird in FORTRAN in der Form abgelegt, daß aufeinanderfolgende Spaltenelemente aufeinanderfolgende Adressen und damit auf einem CRAY-System aufeinanderfolgende Bänke belegen (*Column Major*). Die Distanz zweier Zeilenvektorelemente im Speicher und damit die Schrittweite (Stride) bei einem Zugriff auf einen Zeilenvektor entspricht also genau der Zeilenanzahl der vom Benutzer definierten Matrix. Mit der Wahl der Schrittweite können die *Zugriffsmenge* und die

Zugriffsfrequenz eines Vektorzugriffs eingestellt werden. Elemente einer Zugriffsmenge sind alle Bänke, die im sogenannten *Zugriffsstrom* eines Vektors liegen. Die Zugriffsfrequenz legt die Häufigkeit fest, mit der auf eine Bank zugegriffen wird. Setzt man beispielsweise für die CRAY X-MP die Bank 0 als Startbank voraus, so bedeutet ein Zugriff mit Stride 16, daß nacheinander die Bänke 0, 16, 32 und 48 zyklisch adressiert werden, da das System über 64 Bänke verfügt. Auf jede betrachtete Bank wird nach einer Dauer von jeweils 4 Takten erneut zugegriffen. Durch die Deklaration der Zeilenanzahl der Matrix läßt sich so auf einem Teil des Speichers eine bestimmte Belastung durch Zugriffe einer CPU einstellen.

Da eine Matrix spaltenweise abgespeichert ist, liegen die Startadressen aufeinanderfolgender Zeilenvektoren in unterschiedlichen Bänken. Daher würde die parallele Bearbeitung unterschiedlicher Iterationen durch den parallelen Zugriff mehrerer Prozessoren nicht zu Speicherkollisionen führen. Aus diesem Grund wird das Modellproblem derart modifiziert, daß alle Prozessoren den ersten Zeilenvektor parallel bearbeiten. Der Grad der Parallelität kann dabei über die Anzahl der dem Programm zur Verfügung stehenden Prozessoren bestimmt werden. Bei der CRAY X-MP sind dies maximal 4 und bei der CRAY Y-MP maximal 8 verfügbare Prozessoren.

Implementierung des Modellproblems in FORTRAN

Gemessen werden sollen die Laufzeiten für eine Zeilenvektorreduktion bei unterschiedlicher Speicherauslastung, um die Overhead-Zeiten für Speicherzugriffskonflikte zu ermitteln. Durch Einschachtelung der vektorisierbaren Schleife mit Aufrufen von IRTC läßt sich deren Laufzeit durch Differenzbildung der beiden Meßergebnisse bestimmen. Da alle Messungen in einer dedizierten Umgebung erfolgen, sind keine äußeren Beeinflussungen der Messung zu erwarten. Durch die hohe Auflösung der Meßroutine können Ergebnisse auf den Systemtakt genau ermittelt werden. Um die jeweils gemessenen Laufzeiten eindeutig bestimmten Prozessen zuordnen zu können, muß die Implementierung des Modellproblems leicht modifiziert werden. Anstelle einer Parallelisierung über die Anzahl der Schleifeniterationen der I-Loop wird eine Parallelisierung über die Anzahl der gewünschten Prozessoren vorgenommen. Dabei werden weiterhin verschiedene Schleifendurchläufe auf unterschiedlichen Prozessoren ausgeführt.

In Abbildung 3.1 (Seite 42) ist das Meßprogramm dargestellt. Die Matrix A ist mit einer Zeilenanzahl STRIDE und einer Spaltenanzahl VL deklariert. Der Vektor C erhält zu jeder Iteration das Ergebnis der entsprechenden Zeilenvektorreduktion zugewiesen. Das zweidimensionale Feld TIME dient zur Zwischenspeicherung der Meßwerte. Das Meßprogramm besteht aus drei ineinander geschachtelten Schleifen. Die äußere L-Loop (7,25) wird gemäß der Direktive CMIC$ DO ALL parallelisiert, so daß auf jedem Prozessor eine Kopie des Schleifenrumpfs als eigener Prozeß zur Ausführung kommt. Die Umsetzung der Parallelisierung geschieht in der Weise, daß jedem Prozeß eine Nummer gemäß der Schleifenanweisung zugeordnet wird. Abbildung 3.2 (Seite 43) zeigt schematisch den Verlauf der parallelen Bearbeitung.

Die als SHARED deklarierten Variablen werden dabei gemeinsam von allen parallelen Prozessen benutzt, während die PRIVATE-Variablen in jedem Prozeß lokal vorhanden sind. Durch die Zuweisung der globalen Schleifennummer L auf die in jedem Prozeß lokale Variable LOCL wird jedem Prozeß eine eindeutige Kennung mitgegeben. Die Zeilen 9 - 14 im Zusammenhang mit der GOTO-Anweisung in Zeile 23 simulieren eine *while*-Schleife, die eine Ausführung der Zeilen 15 - 22 so lange garantiert, bis alle durch die Variable ITR vorgegebenen Iterationen berechnet sind. In der mit CMIC$ GUARD und CMIC$ END GUARD eingeschlossenen kritischen Region wird der globale Iterationszähler I inkrementiert und auf eine in dem ausführenden Prozeß lokale Variable LOCI geschrieben. Durch das Inkre-

```
 1                REAL A(STRIDE, VL)
 2                REAL C(ITR)
 3                INTEGER TIME(ITR, 3)
                     :
 4                I = 0

 5      CMIC$     DO ALL SHARED(A, C, TIME, ITR, VL, I, L, CPUS)
 6      CMIC1$    PRIVATE(TIME1, TIME2, LOCI, LOCL, J)

 7                DO 30 L = 1, CPUS
 8                   LOCL = L
 9      10           CONTINUE

10      CMIC$     GUARD
11                   I = I + 1
12                   LOCI = I
13      CMIC$     END GUARD

14                   IF (LOCI .LE. ITR) THEN

15                      TIME1 = IRTC()
16                      DO 20 J = 1, VL
17                         C(LOCI) = C(LOCI) + A(1,J)
18      20              CONTINUE
19                      TIME2 = IRTC()

20                      TIME(LOCI, 1) = TIME1
21                      TIME(LOCI, 2) = TIME2
22                      TIME(LOCI, 3) = LOCL

23                   GOTO 10
24                   ENDIF
25      30        CONTINUE
                     :
```

Abbildung 3.1: Das FORTRAN-Meßprogramm

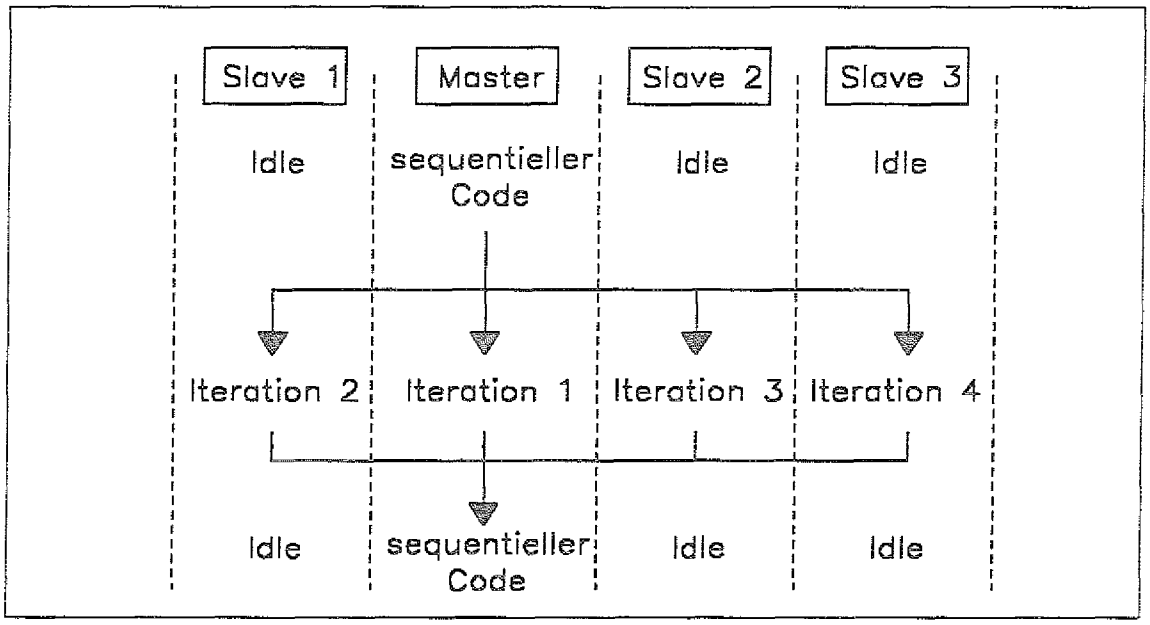


Abbildung 3.2: Parallele Ausführung mehrerer Iterationen unter Autotasking

mentieren des Iterationszählers erhält jeder Prozeß die Nummer der als nächstes zu bearbeitenden Iteration. Auf diese Weise wird die chronologische Reihenfolge der einzelnen Iterationen eingehalten. Ist die durch die globale Variable ITR bestimmte Maximalzahl der Iterationen überschritten, so wird die Berechnung nicht mehr ausgeführt und der Prozeß nach dem *ENDIF* beendet. Sonst wird der Rumpf der IF-Anweisung ausgeführt, der die vektorisierbare J-Loop sowie Anweisungen zur Laufzeitbestimmung und Sicherung der ermittelten Werte enthält. Durch die lokalen Variablen LOCL und LOCI kann jedem Meßwert so eindeutig ein Prozeß und eine Iterationsnummer zugeordnet werden. Um Ein-/Ausgabe während eines Testlaufs zu vermeiden, werden die in einer Messung ermittelten Werte zwischengespeichert und abschließend zwecks späterer Auswertung in Protokolldateien geschrieben. Die Meßdaten eines Prozesses werden dabei an die der bearbeiteten Iterationsnummer entsprechenden Stelle im globalen Array TIME abgelegt. Über die Einstellung von Konstanten können die Arbeitslast (Länge des zu addierenden Vektors VL, Anzahl der Iterationen ITR), der Stride (STRIDE) und die Anzahl der Prozessoren (CPUS) beliebig eingestellt werden. Für eine Einstellung von ITR bei unterschiedlich vielen Prozessoren ändert sich die Arbeitslast je Iteration pro Prozessor nicht. Jede CPU führt zwar mit steigender Prozessoranzahl weniger Iterationen durch, dies hat jedoch lediglich eine Verminderung der Anzahl der parallelen Arbeitsphasen zur Folge.

Die Umsetzung in Master- und Slave-Code durch den FMP erfolgt wie erwartet. Die Multitasked Loop (vergleiche Kapitel 2) umfaßt in beiden Fällen die Zeilen 8 – 24, die von Multitasking-Kontrollstrukturen umgeben werden. Bei Start des Programms werden alle Slave-Prozesse initialisiert. Bei Eintritt des Master-Prozesses in diese parallele Region werden die Slave-Prozesse angefordert, die bis dahin auf Semaphoren geparkt waren und dementsprechend nach wenigen Takten Verzögerung mit der Bearbeitung ihres Codes beginnen können. Jeder Prozeß greift

nach der Bestimmung der Prozeßkennung (Zeile 8) auf die kritische Region (Zeilen 10 – 13) zu, in der eine eindeutige Iterationsnummer vergeben wird. Die kritische Region kann zu einem Zeitpunkt nur von einem Prozeß betreten werden, daher tritt hier auf jeden Fall eine Verzögerung der einzelnen Prozesse ein. Hat ein Prozeß die kritische Region verlassen, so tritt er nach der Überprüfung der Iterationsnummer (Zeile 14) und nach der Zeitnahme in die parallele Bearbeitung des Zeilenvektors ein, in der die Speicherzugriffe stattfinden (Zeilen 16 – 18). Nach Beendigung dieser Bearbeitung erfolgt wiederum eine Zeitnahme und nach einem Sprung ein weiterer Durchlauf der kritischen Region. Liegt im späteren Verlauf eine Iterationsnummer eines Prozesses über der durch den Inhalt der Variablen ITR bestimmten Iterationsanzahl, so wird ein Slave-Prozeß wiederum auf einem Semaphorregister geparkt, während der Master-Prozeß an dieser Stelle wartet, bis alle Slave-Prozesse wieder geparkt sind.

3.3.2 Analyse des Assemblercodes

Das Meßprogramm wird vom Compiler CFT77 in Assemblercode umgewandelt. Von zentraler Bedeutung ist die durch Aufrufe der Zeitmeßroutine IRTC eingeschlossene J-Loop. Daher soll im folgenden die Umsetzung dieser Schleife in Assemblercode betrachtet werden. Die Bearbeitung des Zeilenvektors findet in Segmenten zu je 128 Elementen statt. Der Code kann in vier Bereiche eingeteilt werden:

1. *Initialisierung*: Initialisierung aller für die Berechnung benötigten Werte;
2. *Restbearbeitung*: Verarbeitung von bis zu 128 Elementen;
3. *Hauptschleife*: Verarbeitung der verbleibenden Elemente des Zeilenvektors in Segmenten der Länge 128;
4. *Reduktion*: Reduktion des 64-elementigen Resultatvektors auf einen Wert.

Im ersten Teil werden alle zur Berechnung benötigten Werte initialisiert. Vor der eigentlichen Bearbeitung des Zeilenvektors werden bis zu 128 Elemente in maximal zwei Verarbeitungsschritten geladen und auf ein gemeinsames Vektorregister addiert, so daß die verbleibende Vektorlänge ganzzahlig durch 128 teilbar ist. Dabei wird zum Laden der Vektoren dasselbe Register benutzt. In der anschließenden Hauptschleife werden in jedem Durchgang 128 Elemente des Zeilenvektors bearbeitet. Zwei Vektorregister werden mit fortlaufenden, 64-elementigen Segmenten des Zeilenvektors geladen und in jedem Schleifendurchlauf auf ein Vektorregister addiert. Diese von CRAY als *Unrolling* bezeichnete Strategie ermöglicht die parallele Nutzung beider Lade-Ports einer CPU. Da während der Bearbeitung der Hauptschleife die Addition mangels einer zweiten Additionspipeline nicht parallel durchgeführt werden kann, beschränkt sich in diesem Fall der Vorteil einzig auf das Chaining und auf die Möglichkeit, Elemente eines Vektors vor Beginn der nächsten Addition laden zu können. Abbildung 3.3 zeigt schematisch die Vorgehensweise. Nach vollendeter Bearbeitung des Zeilenvektors wird das Resultatregister in einem iterativen Verfahren auf ein Element reduziert. Eine genauere Betrachtung insbesondere der abschließenden Reduktion kann [Hof90] entnommen werden.

Von großer Bedeutung für eine Messung sind die Speicherzugriffe des Meßprogramms und die dabei auftretenden Verzögerungen durch Zugriffskonflikte. Bedingt durch die FORTRAN-Instrumentierung wird im vorliegenden Fall die Laufzeit für alle beschriebenen Codesequenzen gemessen. Es ist wünschenswert, den Einfluß der hier nicht interessierenden Bereiche so gering wie möglich zu halten. Der Zeitaufwand für die Initialisierung ist konstant, da dort keine Speicherzugriffe stattfinden und keine Schleifenstrukturen durchlaufen werden. Unter der Voraussetzung, daß die Vektorlänge ganzzahlig durch 128 teilbar gewählt wird, werden während der Restbearbeitung zwei Zugriffe auf Vektoren der Länge 64 durchgeführt. Da diese Situation der eines Schleifendurchlaufs durch die Hauptschleife entspricht, soll dieser Teil der Bearbeitung im folgenden als ebenfalls der Hauptschleife zugehörig betrachtet werden. Die Reduktion enthält insgesamt 5 Vektorzugriffe, die alle mit Stride 1 vorgenommen werden. Da sich die Bearbeitung der letzten Ladeoperation der Hauptschleife und die Reduktion überlagern, können hier noch zusätzliche Effekte auftreten, die die Genauigkeit der Messung beeinflussen. Vernachlässigt man diesen Einfluß und nimmt den Aufwand für die Bearbeitung der Reduktion als konstant an, so entsteht lediglich ein durch Initialisierung und Reduktion verursachter, für jede Messung gleichbleibender Fehler, der im folgenden mit dem Begriff *IR-Aufwand* oder *IR-Zeit* bezeichnet wird. In Abschnitt 3.3.3 wird ein Verfahren zur Berechnung dieses Aufwands angegeben.

Untersuchung der Hauptschleife

Der Assemblercode der Hauptschleife ist in Abbildung 3.4 (Seite 46) dargestellt und soll nun anhand analytischer Methoden auf sein Ausführungsverhalten untersucht werden. Die Angaben in den mit I und E bezeichneten Spalten beziehen sich auf die Anzahl der zur Ausführung der Befehlsinitiierung (*Issue*) bzw. der Befehlsausführung (*Execute*) notwendigen Takte unter optimalen Bedingungen, also

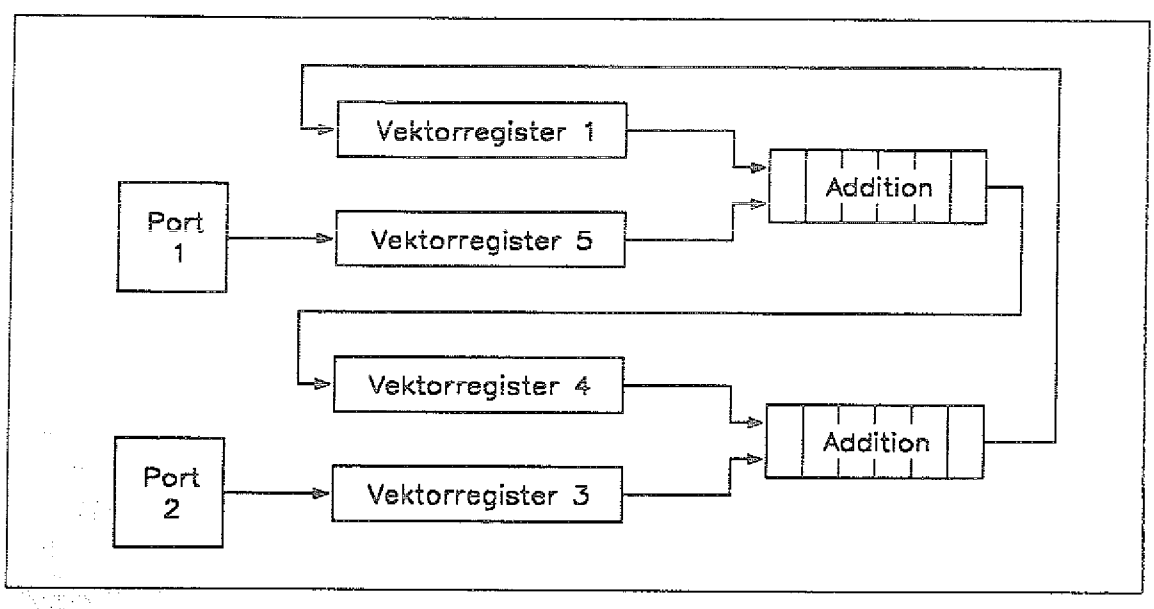


Abbildung 3.3: Schematischer Verlauf der Addition

Nr.	Befehl		I	E	Kommentar
1	A7	1	1	1	
2	A0	A7 + A2	1	2	Basisadresse erster Vektor
3	V5	,A0 ,A1	1	vl+17	Ersten Vektor laden
4	V4	V1 + FV5	1	vl+11	Ersten Vektor addieren
5	A6	64	2	1	Vektorlänge für zweiten Zugriff merken
6	A5	VL	1	1	Vektorlänge des ersten Zugriffs merken
7	A4	A1 * A5	1	4	Distanz erster Vektor
8	A4	A2 + A4	1	2	
9	A0	A7 + A4	1	2	Basisadresse zweiter Vektor
10	VL	A6	1	1	Vektorlänge zweiter Zugriff
11	V3	,A0 ,A1	1	vl+17	Zweiten Vektor laden
12	A7	A1 * A6	1	4	Distanz zweiter Vektor
13	S7	+A5	1	2	
14	S6	S1 + S7	1	3	
15	S7	A6	1	2	
16	S1	S6 + S7	1	3	Schleifenzähler hochzählen
17	A2	A4 + A7	1	2	Basisadresse für nächste Schleife
18	V1	V4 + FV3	1	vl+11	Zweiten Vektor addieren
19	A0	S1	1	1	
20	JAN	< 1 >	5	0	Sprung nach 1, solange A0 negativ

Abbildung 3.4: Die zentrale Assemblerschleife

bei Verfügbarkeit aller Ressourcen und ohne Zugriffskonflikte [Cra86]. Die in der Spalte Execute angegebene Größe *vl* bezeichnet die Vektorlänge des jeweils zu bearbeitenden Vektors.

Bei Eintritt in die Schleife sind die Register V1, A1, A2, S1 und VL vorbelegt. Für die hier vorliegenden Betrachtungen sind alle Elemente des Ergebnisregisters V1 vor Eintritt in die Schleife gleich Null. Der Inhalt von Register A1 bestimmt den Stride, mit dem auf die Elemente der Vektoren zugegriffen wird. In Register A2 befindet sich die um eins reduzierte Basisadresse des nächsten zu ladenden Vektorelements. Register S1 enthält die Anzahl der noch zu ladenden Elemente in negativer Darstellung und wird als Zähler verwendet. Das VL-Register ist mit dem Wert 64 vorbesetzt.

Die folgende Beschreibung der Schleifenstruktur referiert in Klammern entweder die Nummer der Instruktion nach Abbildung 3.4 oder die betroffenen Register. In jedem Schleifendurchlauf werden zwei Vektoren der Länge 64 geladen (3;11) und auf ein gemeinsames Vektorregister aufaddiert (4;18). Vor jedem Zugriff sind Basisadresse (A0), Stride (A1) und Länge des Vektors (VL) bekannt. Nach jedem Zugriff wird anhand dieser Informationen die Basisadresse des nächsten Zugriffs be-

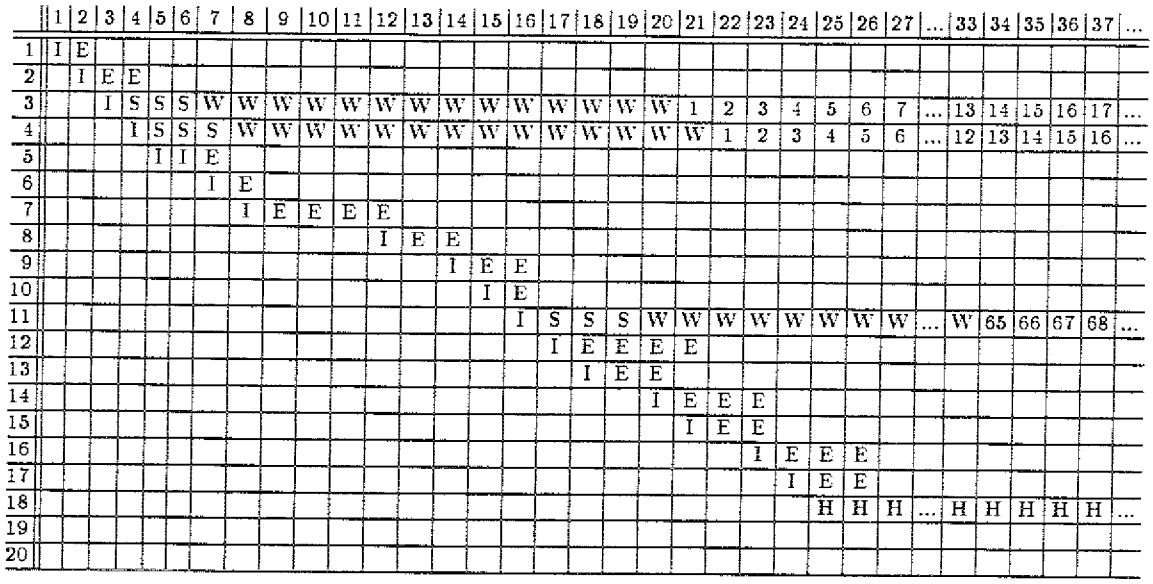


Abbildung 3.5a: Ausführungsverhalten der Operationen in der Assemblerschleife für die Takte 1 – 37

rechnet (7 – 9;12,17,1,2). Da die Information über die Anzahl der noch zu ladenden Elemente als negative Zahl vorliegt (S1), brauchen die beiden bearbeiteten Vektorlängen (A5,A6) lediglich auf diesen Wert addiert zu werden (13 – 16), um dann auf die Sprungbedingung geprüft zu werden (19,20).

Wird der angegebene Code ausgeführt, so sind alle Bedingungen, die zu Verzögerungen führen können, zu beachten (siehe Abschnitt 1.1.1). Für die nun folgenden Betrachtungen seien alle äußeren Einflüsse vernachlässigt, insbesondere soll die Ausführung nicht durch Zugriffskonflikte behindert sein. Die Abbildungen 3.5a und 3.5b (Seite 48) zeigen den Verlauf der Ausführung der Hauptschleife. In den Zeilen sind die Instruktionen entsprechend mit ihrer Numerierung nach Abbildung 3.4 eingetragen, jede Spalte entspricht einem Takt. Die Eintragungen beziehen sich auf Aufsetz- und Ausführungsphasen (I bzw. E) der einzelnen Instruktionen, Setup-Phasen (S) der Vektorbefehle, Wartezeiten auf Daten aus dem Hauptspeicher (W), Verzögerungen der Vektorinstruktionen durch blockierte Ressourcen (H), abschließende Blockierung der Ports durch Vektorbefehle (B), oder sie benennen den Index eines Vektorelements beim Eintritt in eine Vektoreinheit. Dabei sind die Elemente des Zeilenvektors durchgehend numeriert. Man beachte, daß die angegebenen Setup-Zeiten sich auf die Initialisierung der Einheiten zur Ausführung eines Befehls beziehen und nicht mit Startup-Zeiten zu verwechseln sind, die zur Auffüllung der Pipelines benötigt werden (siehe Abschnitt 1.1). Diese Startup-Zeiten sind für die Vektorladeoperationen 3 und 11 mit W bezeichnet. Die Beschreibung der Abbildungen im folgenden Abschnitt soll das Verständnis des Ablaufs der inneren Schleife erleichtern.

	...	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	...	114	115	116	117	...
1																									
2																									
3	...	63	64	B	B																				
4	...	62	63	64																					
5																									
6																									
7																									
8																									
9																									
10																									
11	...	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	B	B							
12																									
13																									
14																									
15																									
16																									
17																									
18	...	H	H	H	I	S	S	S	65	66	67	68	69	70	71	72	73	74	75	...	89	90	91	92	...
19						I	E																		
20							I	I	I	I	I														
1												I	E												
2													I	E	E										
3														I	S	S	S	W	W	...	W	W	129	130	...
4															H	H	H	H	H	...	H	H	H	H	...

Abbildung 3.5b: Ausführungsverhalten der Operationen in der Assemblerschleife für die Takte 83 – 117

Die Operationen 1 – 7 sind bezüglich der benötigten Ressourcen voneinander unabhängig und können direkt aufeinanderfolgend aufgesetzt werden. Der Vektorladebefehl 3 benötigt Operanden aus dem Speicher, die nach 3 Takten Setup-Zeit für die Instruktion erst nach weiteren 14 Takten eintreffen. Die Operationen 3 und 4 werden verkettet ausgeführt. Nach dem Setup der Additionsoperation wartet die Pipeline, bis der erste Operand im Register V5 verfügbar wird. Einen Takt später kann dann das erste Operandenpaar der Pipeline zugeführt werden, danach wird in jedem weiteren Takt ein neues Operandenpaar an die Pipeline weitergegeben. Operation 5 benötigt zwei Takte Aufsetzzeit, daher wird Instruktion 6 erst im siebten Takt initiiert. Befehl 8 muß 4 Takte auf das Eintreffen des Ergebnisses von Befehl 7 in Register A4 warten, entsprechendes gilt für Befehl 9 mit 2 Takten Wartezeit. Die Operationen 10 – 13 benutzen wiederum unabhängige Ressourcen und können so jeweils aufeinanderfolgend aufgesetzt werden. In Takt 16 ist zum Zeitpunkt der Initiierung des zweiten Ladebefehls 11 der Ladevorgang aus Operation 3 auf einem Port noch in Ausführung. Da das zweite Ladeport noch frei ist und ein freies Register angesprochen wird, kann auch dieser Befehl sofort ausgeführt werden. Operation 14 muß auf das Ergebnis von Operation 13 in Register S7 warten. Befehl 15 kann direkt nach Befehl 14 aufsetzen, da der Wert eines Operandenregisters bei einer Skalaroperation im ersten Takt in die Pipeline kopiert wird und das entsprechende Register direkt im nächsten Takt wieder benutzt werden kann. Instruktion 16 muß auf die Ergebnisse der Instruktionen 14 und 15 in den Registern S6 bzw. S7 warten. Die Operation 17 wird wiederum ohne Verzögerung aufgesetzt. Die Additionsoperation 18 benutzt die Additionspipeline, die jedoch zu diesem Zeitpunkt noch durch die

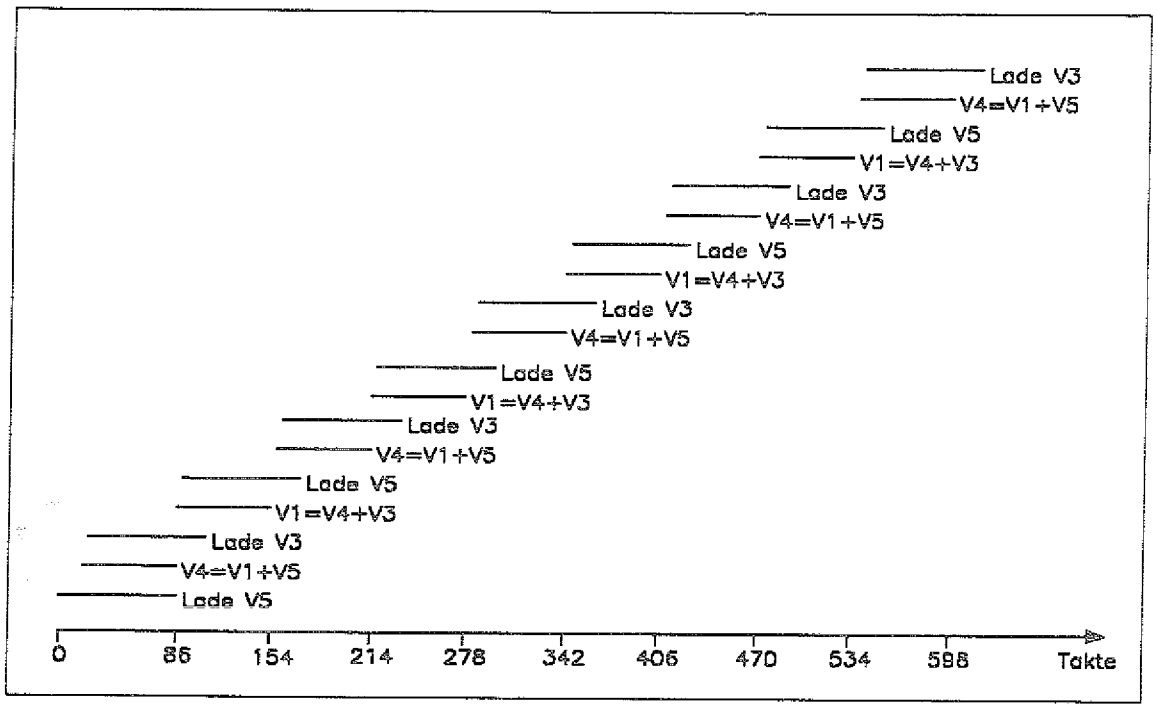


Abbildung 3.6: Verlauf der ersten Lade- und Additionsvorgänge bei konfliktfreiem Speicherzugriff

erste Additionsoperation 4 belegt ist. Das Aufsetzen von Befehl 18 wird dementsprechend bis zur Verfügbarkeit der Pipeline im Takt 87 verzögert. Zu diesem Zeitpunkt ist schon ein großer Teil der Ladeoperation 11, die mit der Additionsoperation 18 verkettet wird, durchgeführt worden, so daß die Pipeline direkt im Anschluß an die Setup-Phase die Addition aufnehmen kann. Die Instruktionen 18 – 20 werden wiederum aufeinanderfolgend aufgesetzt. Der Sprungbefehl 20 verhindert für die nächsten 5 Takte jegliche Befehlsinitiierung. Im zweiten Schleifendurchgang können die Befehle 1 – 3 wiederum sofort aufeinanderfolgend aufgesetzt werden. Das Register V5 wurde bereits vor Beendigung der Additionsoperation 4 aus dem ersten Durchgang freigegeben. Ebenso steht wiederum ein freies Port zur Verfügung, da der Ladevorgang von Befehl 3 aus dem ersten Durchgang in Takt 86 beendet wurde. Analog zur Situation der Addition 18 im ersten Durchgang kann jedoch Befehl 4 erst dann aufgesetzt werden, wenn die Additionspipeline wieder freigegeben wird.

Ab dem Zeitpunkt des Aufsetzens der Operation 18 im ersten Durchlauf hat sich eine stabile Phase eingependelt. Während eine Additionsoperation abgearbeitet wird und der Vektors für die nächste Operation geladen wird, wird die Befehlsinitiierung vor der nächsten Additionsoperation angehalten. Abbildung 3.6 zeigt schematisch den weiteren Verlauf der Ladevorgänge unter den bisherigen Voraussetzungen.

3.3.3 Das analytische Modell

Mit Hilfe der vorangegangenen Analyse kann ein Modell für die Ermittlung der Laufzeit der Assemblerschleife angegeben werden. Dabei ist zu beachten, daß sich die Analyse auf die Betrachtung eines Prozessors beschränkt und daher bei der Erweiterung auf mehrere Prozessoren nicht regelmäßig auftretende Interprozessorkonflikte und deren Auswirkungen nicht berücksichtigen kann. Zusätzlich muß die Annahme getroffen werden, daß die Interprozessorprioritäten für den Zugriff auf den Hauptspeicher gerecht auf die Prozessoren verteilt sind, d.h. daß im Mittel kein Prozessor gegenüber einem anderen bevorzugt wird. Diese Annahme erscheint wegen des zyklischen Prioritätenverfahrens zumindest für die CRAY X-MP gerechtfertigt. Unter diesen Voraussetzungen kann das Modell im wesentlichen sowohl für CRAY X-MP als auch für CRAY Y-MP verwendet werden, da durch die identischen Compiler der gleiche Code erzeugt wird und die von der Ausführung betroffene Prozessor-Hardware keine signifikanten Unterschiede aufweist. Da jedoch das Zugriffsverhalten wesentlich durch die unterschiedlichen Speicherverbindungsnetzwerke bestimmt wird, sind für die beiden Systeme teilweise leicht modifizierte Betrachtungen erforderlich.

Das Modell geht von einer festen Anzahl k von Schleifendurchläufen aus, die sich für eine ganzzahlig durch 128 teilbare Vektorlänge vl zu

$$k = vl \; div \; 128$$

berechnet. Unter bestimmten Voraussetzungen verändert sich die Dauer für einen Schleifendurchlauf. Für das Modell werden alle grundsätzlichen Möglichkeiten, die sich durch einen Zugriff von mehreren Prozessoren auf den Hauptspeicher ergeben können, angegeben. Dabei wird zuerst das Verhalten eines Prozessors untersucht und anschließend eine Erweiterung auf die Bearbeitung mit mehreren Prozessoren vorgenommen. In Abhängigkeit von der jeweiligen Zugriffssituation wird eine entsprechende Formel zur Berechnung der Dauer der Bearbeitung der Assemblerschleife angegeben. Anschließend wird noch ein Verfahren zur Bestimmung des durch die Bearbeitung von Initialisierung und Reduktion entstehenden Aufwands vorgestellt.

Die analytischen Betrachtungen für die Ausführung der Assemblerschleife bei Stride 1 ohne das Auftreten von Verzögerungen führt zu folgenden Ergebnissen:

- Das Laufzeitverhalten wird von dem Zeitaufwand für die Ausführung der Vektorbefehle dominiert. Die Skalarbefehle werden parallel dazu bearbeitet und haben keine Auswirkungen auf die Laufzeit.

- Die Vektorladeoperationen können verschränkt vorgenommen werden, so daß während einer Addition bereits Elemente für die nächste Berechnung in ein Register gepuffert werden. So sind außer beim ersten Schleifendurchlauf immer alle für eine Operation benötigten Daten in einem Register verfügbar. Die Ladeoperationen verursachen keine Verzögerungen.

- Nach einer Startup-Phase während des ersten Schleifendurchlaufs erfolgt die Bearbeitung der übrigen Durchgänge identisch. Alle Berechnungen werden direkt aufeinanderfolgend durchgeführt. Die Dauer eines Schleifendurchlaufs ist bestimmt durch die Dauer der Additionsoperation. Die Berechnung eines 64-elementigen Vektors belegt die Pipeline für insgesamt 68 Takte: drei Setup-Takte, einen Takt für jedes zu berechnende Element und einen Takt nach Eintritt des letzten Elements in die Pipeline. Da ein Schleifendurchlauf zwei Vektoradditionen mit je 64 Elementen enthält, werden jeweils 136 Takte für die Bearbeitung von 128 Elementen benötigt.

Vernachlässigt man die Startup-Phase des ersten Schleifendurchlaufs, so kann unter den vorgegebenen Umständen eine Abschätzung für die Dauer der Bearbeitung der Hauptschleife in Takten mit

$$= k * (64 + 4) * 2 = k * 136 \tag{3.1}$$

angegeben werden.

Ab Stride 16 bei der CRAY X-MP und Stride 4 bei der CRAY Y-MP liegen alle referierten Speicherbänke in derselben Sektion. Da nur ein Zugriffspfad pro CPU zu jeder Sektion existiert, kann unter diesen Voraussetzungen zu einem Zeitpunkt nur noch ein Port pro CPU auf den Speicher zugreifen. Die Ports können also nicht mehr verschränkt arbeiten, sondern wechseln sich mit dem Laden der Vektoren ab. Setzt man nun voraus, daß unter diesen Umständen keine weiteren Zugriffskonflikte erfolgen, so ändert sich an der Bearbeitung der Vektorelemente bis zum Ende des Zugriffs auf den ersten Vektor nichts. Erst danach können die Elemente für den zweiten Zugriff geladen werden. Das folgende Schema zeigt für diesen Fall den Übergang zwischen den beiden Vektoradditionen für fortschreitenden Takt. Die

V5	...	63	64	-	-	-	-	-	-	-	-	...
V3	...	-	-	65	66	67	68	69	70	71	72	...
Pipe	...	62	63	64	D	S	S	S	65	66	67	...

Vektorelemente sind fortlaufend numeriert. Die Zeilen zu V5 und V3 zeigen den Beginn der Verfügbarkeit der indizierten Elemente im entsprechenden Register, die Zeile *Pipe* den Index der in die Pipeline eintretenden Elemente. Mit D (Shutdown) und S (Setup) sind die laut Abschnitt 1.1.1 definierten Overhead-Zeiten der Pipeline bezeichnet, in denen diese keine Elemente entgegennehmen kann. Der Darstellung ist zu entnehmen, daß mit jedem der vier Takte der Overhead-Zeit ein weiteres Element in das nächste Register geladen wird (Elemente 66 – 69). Danach kann die Pipeline wiederum in jedem Takt ein Element aufnehmen. Dieser Vorgang wiederholt sich jeweils bei Ende einer Vektoraddition, so daß für die jeweils folgende Operation bereits Elemente im entsprechenden Register zur Verfügung stehen. Die Bearbeitung der Schleife ist also wie im ersten Fall von der Pipeline abhängig und kann daher auf diesen zurückgeführt werden.

Das Ausführungsverhalten ändert sich bei Auftreten von regelmäßigen Bankkonflikten in Zusammenhang mit dem Zugriff über ein einzelnes Port. Das Schema gibt am Beispiel eines regelmäßig um einen Takt verzögerten Zugriffsstroms einen Einblick in die Ausführung der Vektoraddition. Die erste Vektoraddition wird ent-

V5	...	63	-	64	-	-	-	-	-	-	-	-	-	-	-	-	-	...
V3	...	-	-	-	-	65	-	66	-	67	-	68	-	69	-	70	-	...
Pipe	...	-	63	-	64	D	S	S	S	65	66	67	68	-	69	-	70	...

sprechend der nur jeden zweiten Takt eintreffenden Elemente um einen Takt pro Operand verzögert. Die Verzögerung durch den Overhead der Pipeline wird durch diese regelmäßigen Verzögerungen kompensiert, da die Pipeline nach ihrer Setup-Phase so lange ein Element pro Takt aufnehmen kann, bis sie wieder durch den lediglich in jedem zweiten Takt erfolgreichen Zugriff auf neue Elemente warten muß (Elemente 65 – 68). Die Dauer eines Schleifendurchlaufs wird jetzt also nicht weiter durch die Additionsoperation, sondern durch die Dauer der pro Zugriff auftretenden Verzögerung oder auch durch die Dauer eines Zugriffs bestimmt. Im vorliegenden Beispiel werden für jeden Zugriff zwei Takte benötigt. Die Berechnung muß daher abweichend von den bisherigen Fällen mit

$$T_2 = k * 128 * \tau_d, \tag{3.2}$$

angegeben werden, wobei τ_d für die Dauer eines Zugriffs in Takten steht.

Durch den Einsatz mehrerer Prozessoren ist bei entsprechender Speicherauslastung eine Erhöhung der Zugriffskonflikte zu erwarten. Für die Situationen, in denen sich einzelne Prozessoren nicht durch Zugriffe auf die gleiche Sektion selbst behindern, ist eine allgemeine Vorhersage des Laufzeitverhaltens schwierig. Der Pufferfunktion der Register stehen die zunehmenden Konflikte gegenüber, deren Einfluß von der jeweiligen Zugriffssituation abhängig ist. Solange die durchschnittliche Dauer eines Zugriffs nicht größer als ein Takt wird, erscheint jedoch für eine Abschätzung der Laufzeit die Berechnung nach Formel 3.1 weiterhin geeignet.

Für das Laden über ein Port gelten für mehrere Prozessoren prinzipiell die gleichen Überlegungen wie bei einem Prozessor. Da alle Elemente sukzessive geladen werden müssen, wirken sich regelmäßige Konflikte direkt auf die Laufzeit der Schleife aus. Für solche Fälle muß die durchschnittliche Dauer eines Zugriffs berechnet. Sie geht in der Formel 3.2 als τ_d ein.

Nachdem nun für die unterschiedlichen Zugriffssituationen Formeln für den Zugriff eines Prozessors angegeben wurden, muß in einem weiteren Schritt geklärt werden, in welcher Weise diese Situationen von den Parametern Stride und Prozessoranzahl bei paralleler Verarbeitung abhängig sind. Fixiert man die Prozessorzahl, so kann durch den Stride die Zugriffsmenge für das Meßprogramm eingeschränkt und damit gleichzeitig die Zugriffsfrequenz auf die verfügbaren Bänke erhöht werden. Wird

eine Bank häufiger angesprochen, als es die Bankzykluszeit ohne Konflikte erlaubt, so erfolgt ein Bankkonflikt, der ein Stocken des Zugriffsstroms verursacht. Für einen festen Stride bedeutet die Hinzunahme weiterer Prozessoren für die gleichzeitige Bearbeitung aus der Sicht des Speichers ebenfalls eine Erhöhung der Zugriffsfrequenz, da alle Prozessoren auf derselben Zugriffsmenge arbeiten. Die Bandbreite eines Speichers gibt an, auf wieviele Speicherworte im Mittel pro Taktzyklus zugegriffen werden kann. Zu beachten ist, daß sie eine theoretische Größe ist und nicht das den Speicher und die Prozessoren verbindende Netzwerk berücksichtigt. Für einen Speicher mit m unabhängigen Bänken und einer Speicherzykluszeit von n_c (gemessen in Systemtakten) berechnet sich die Bandbreite B bei fortlaufend zyklischem Zugriff auf die Bänke zu

$$B = \frac{Bankanzahl}{Bankzykluszeit} = \frac{m}{n_c}.$$

Erfolgt ein Zugriff mit einem Stride s, wobei s eine Zweierpotenz ist, so berechnet sich die Anzahl der im Zugriff liegenden Bänke mit

$$m_s = \frac{m}{s}.$$

Diese Überlegungen führen zu der Definition der *relativen Speicherbandbreite* B_s, die im folgenden die von einem bestimmten Stride abhängige Transferrate eines Speichers angibt und sich zu

$$B_s = \frac{m_s}{n_c} = \frac{m}{s \times n_c}$$

berechnet. Greift lediglich ein Prozessor auf den Speicher zu, so bestimmt B_s die maximale Anzahl von Speicherworten, die durchschnittlich zwischen dem Prozessor und dem Speicher pro Takt übertragen werden können. Durch Hinzunahme weiterer Prozessoren ändert sich zwar nicht die relative Bandbreite des Speichers, jedoch konkurrieren nun alle Prozessoren gemeinsam um den Speicherzugriff. Aus Sicht einer CPU kann also mit

$$B_{p/s} = \frac{B_s}{p} = \frac{m}{s \times n_c \times p}$$

ein Wert für die pro Prozessor transferierbaren Speicherworte angegeben werden, wobei p die Anzahl der Prozessoren angibt. Sinkt dieser Wert unter eins, so ist nicht in jedem Takt ein Zugriff möglich. In diesem Fall beschreibt der Kehrwert $1/B_{p/s}$ die mittlere Dauer τ_d für einen Zugriff in Takten.

Für die CRAY X-MP kann so eine Berechnung unabhängig von der Parameterkonstellation vorgenommen werden. Für die CRAY Y-MP gelten einige Einschränkungen. Bei der Bearbeitung mit Strides von 1 – 4 wird auch mit 8 Prozessoren nie eine relative Bandbreite unter 1 erreicht. Für diese Fälle ist also die Formel 3.1 anzuwenden. Für alle höheren Strides dient daher die Formel 3.2 als Grundlage. Ab Stride 8 macht sich der Einfluß der Untersektionen auf den Zugriff einzelner Prozessoren bemerkbar. Aus Sicht eines Prozessors ist lediglich ein gleichzeitiger Zugriff

auf 32 Untersektionen möglich (siehe Abschnitt 1.2). Für die Zugriffe eines Prozessors mit Strides bis 32 berechnet sich die relative Bandbreite in diesem Sonderfall zu

$$B_y = \frac{32}{s * n_c}.$$

Da bei dieser Berechnung der Einfluß durch andere Prozessoren nicht mit eingerechnet ist, müssen die ausschlaggebenden Einflüsse gegeneinander abgewägt werden. Es zeigt sich, daß die durch Untersektionskonflikte eingeschränkten relativen Bandbreiten immer kleiner oder gleich den prozessorspezifischen relativen Bandbreiten sind, daher beträgt die in die Formel einzusetzende Verzögerung

$$\tau_d = \frac{s * n_c}{32}.$$

Für Strides größer 32 sind die Verzögerungen jeder CPU durch die Untersektionskonflikte konstant. Da in jedem Takt ein Zugriff eines Prozessors auf dieselbe Untersektion stattfindet und damit um vier Takte verzögert wird, beträgt die mittlere Dauer eines Zugriffs dann mindestens 5 Takte. Weitere Verzögerungen sind erst dann zu erwarten, wenn die Verzögerung durch Interprozessorkonflikte über die durch Untersektionskonflikte verursachte Verzögerung hinausgeht. Die mittlere Verzögerung läßt sich also bestimmen zu

$$\tau_d = \begin{cases} \frac{s * n_c * p}{256} & \text{für} \quad B_{p/s} < 1/5 \\ 5 & \text{für} \quad B_{p/s} \geq 1/5 \end{cases}.$$

Für jede beliebige Parameterkonfiguration kann nun mit Hilfe des Modells eine Prognose für die Laufzeit der Hauptschleife angegeben werden. Dabei ist zuerst die Festlegung der Formel und danach eventuell die Berechnung der mittleren Dauer eines Zugriffs erforderlich.

Eingangs wurden die Einschränkungen bei der Betrachtung der Zugriffssituationen mit Hilfe des Modells beschrieben. Die Entwicklung des Modells für einen Prozessor und der Schluß auf das Ausführungsverhalten mit mehreren Prozessoren verhindert die Berücksichtigung von sporadischen Interprozessorkonflikten und deren Auswirkungen beispielsweise auf die Blockierung von Ports. Auch das Verhalten der dynamischen Prioritätenregelung kann nicht näher modelliert werden, daher muß von einer Gleichberechtigung der Prozessoren in allen Fällen ausgegangen werden. Solche Effekte bilden mögliche Ursachen für eventuell auftretende Abweichungen der Meßwerte von den mit dem Modell ermittelten Werten.

Es ist zu beachten, daß die durch das Modell berechnete Größe lediglich ein Maß für die Dauer der Bearbeitung der Hauptschleife des Assemblercodes darstellt. In der gemessenen Laufzeit einer Iteration ist jedoch darüberhinaus der im wesentlichen konstante Aufwand für die Bearbeitung von Initialisierung und Reduktion enthalten (siehe Seite 44). Im folgenden soll daher ein Verfahren angegeben werden, mit dem aufgrund verschiedener Meßresultate der Umfang dieses IR-Aufwands berechnet werden kann. Voraussetzung zur Anwendung des Verfahrens ist die Annahme, daß

der entstehende Zeitaufwand für Messungen mit einer bestimmten Parameterkonfiguration unabhängig von der Vektorlänge konstant ist. Diese Annahme erscheint unter Betrachtung der auf Seite 44 genannten Gründe zulässig.

Wählt man für zwei Messungen verschiedene, jedoch durch 128 ganzzahlig teilbare Vektorlängen vl_1 und vl_2, so unterscheidet sich die Ausführung der beiden Meßprogramme lediglich in der Anzahl der Durchläufe durch die Hauptschleife. Geht man weiter davon aus, daß eine Messung repräsentativ für die gewählte Parameterkombination ist und nicht von außen beeinflußt wird, so bestimmt die Differenz der beiden Laufzeiten exakt die für die Berechnung von $|vl_2 - vl_1|$ Elementen notwendige Zeit. Bei einer Wahl der Vektorlängen von $vl_1 = 2^k$ und $vl_2 = 2^{k+1}$ erhält man so die Zeit, die ohne die IR-Zeiten für die Berechnung von 2^k Elementen benötigt wird. Durch Subtraktion dieses Berechnungsaufwands vom Meßwert für vl_1 kann der Meßaufwand, der nicht durch die Berechnung des Vektors entsteht und durch die FORTRAN-Instrumentierung mitgemessen wird, auf einfache Weise bestimmt werden.

Faßt man die Ergebnisse des Modells zur Berechnung der Laufzeit der Assemblerschleife und des Verfahrens zur Bestimmung der IR-Zeiten zusammen, so erhält man eine Aussage über die erwartete Dauer für die Ausführung einer Iteration. Auf diese Weise sind Ergebnisse des Modells und der Laufzeitmessung direkt vergleichbar.

Kapitel 4

Entwicklung eines Simulators

Simulationen bieten die Möglichkeit, detaillierte Informationen über komplexe Vorgänge dort zu erhalten, wo mit analytischen Methoden und Messungen keine weiteren Einzelheiten verfügbar gemacht werden können. Wie in Abschnitt 3.2.4 gezeigt, ist eine detaillierte Untersuchung von Konfliktsituationen durch Speicherzugriffe mit Hilfe von Messungen auf den Systemen CRAY X-MP und CRAY Y-MP nicht realisierbar. Durch das in Abschnitt 3.3.3 vorgestellte analytische Modell können zwar auch für den Fall mehrerer, parallel zugreifender Prozessoren aus dem Zugriffsmuster Aussagen über das Laufzeitverhalten für ein vorgegebenes Problem abgeleitet werden, eine tiefergehende Betrachtung von einzelnen Konfliktsituationen kann jedoch nicht durchgeführt werden. Daher erweist sich für die vorliegende Fragestellung der Entwurf eines Computerprogramms zur Simulation der Vorgänge als erforderlich. Wegen der Komplexität der betrachteten Systeme und des Zeitaufwands für Entwurf und Verifikation einer solchen Simulation muß sie sich im Rahmen dieser Arbeit auf einen Rechner beschränken. Da im Gegensatz zur CRAY Y-MP über die CRAY X-MP alle dafür notwendigen Informationen veröffentlicht sind und vorliegen, bietet sich dieses System als Objekt für die Simulation an.

Für die Implementierung der Simulation wurde die Programmiersprache PASCAL gewählt. Die Entwicklung des Programms fand auf einem Personal Computer unter Benutzung der interaktiven Entwicklungsumgebung TURBO PASCAL 5.0 (Borland) statt.

Das Ziel der Simulation ist die Erkennung und Erklärung von komplexen Konfliktsituationen, die bei der Ausführung des im vorangegangenen Kapitel vorgestellten Modellproblems auf der CRAY X-MP entstehen. Mit Hilfe der Simulationsergebnisse sollen Ursachen für die Entstehung solcher Situationen aufgezeigt werden können. Als Untersuchungsobjekt wird der Ablauf des Modellproblems auf dem Rechner CRAY X-MP gewählt. Die Verifikation der Simulationsergebnisse erfolgt durch den Vergleich mit Messungen der Ausführungszeiten des Modellproblems und den Abschätzungen aus der analytischen Modellierung.

Da sich das Ziel auf die Betrachtung von Konfliktsituationen durch Speicherzugriffe beschränkt, ist im Rahmen der Simulation des Modellproblems eine Ein-

schränkung auf die Modellierung der Programmteile sinnvoll, in denen Speicherzugriffe stattfinden. Nach den analytischen Betrachtungen des Abschnitts 3.3.2 reicht also eine Modellierung der Vektoroperationen der inneren Assemblerschleife (siehe Abbildung 3.4) unter Berücksichtigung des Zeitverhaltens der Skalaroperationen (siehe Abbildung 3.5) aus. Dabei sind die nach Abschnitt 3.3.1 auftretenden Differenzzeiten des Eintritts verschiedener Prozesse in die parallele Bearbeitung zu berücksichtigen. Während das Zeitverhalten der Skalaroperationen konstant ist (siehe Abbildung 3.5), sind die Multitasking-Verzögerungszeiten von der Programmausführung abhängig und müssen meßtechnisch ermittelt werden.

Der Abbildung 3.5 ist zu entnehmen, daß lediglich die Vektorladeoperationen 3 und 11 sowie die Vektoradditionsoperationen 4 und 18 zu simulieren sind. Da über zwei unterschiedliche Ports in unterschiedliche Register geladen wird, können Ladeoperationen gleichzeitig stattfinden. Die Additionsoperationen benutzen dieselbe Pipeline und können nur jeweils aufeinanderfolgend, dabei aber parallel zu den Ladeoperationen, durchgeführt werden. Für die Implementierung des Simulationsprogramms müssen diese parallelen Abläufe sequentialisiert werden. Daher betrachtet der zeitgesteuerte Simulator für jeden Systemtakt den Zustand der von laufenden Operationen betroffenen Ressourcen, ermittelt anhand dieser Zustände durchzuführende Aktionen und verändert entsprechend Zustände betroffener Ressourcen. Sind parallel laufende Operationen voneinander unabhängig, d.h. benutzen sie unterschiedliche Ressourcen, so ist die Reihenfolge der Bearbeitung unerheblich. Voneinander abhängige Operationen werden in der Art simuliert, daß die durchzuführenden Aktionen nach absteigenden Zugriffsprioritäten durchgeführt werden. Damit ist beispielsweise bei Zugriffskonflikten die Reihenfolge eindeutig festgelegt, in der die einzelnen Prozessoren auf den Speicher zugreifen können.

Die für die Ausführung des Modellproblems betrachteten Ressourcen sind die vier Prozessoren, das Speicherverbindungsnetzwerk und der Speicher. Sie werden durch Datenstrukturen modelliert. Zu jedem der vier Prozessoren werden zwei Ports, die Additionspipeline und vier Register betrachtet. Für den Speicher und das Verbindungsnetzwerk werden die Bänke und die Sektionspfade modelliert. Die Betrachtung einzelner Adressen erweist sich als nicht notwendig, da diese nur bezüglich ihrer relativen Lage im Speicher von Interesse sind.

Die Zustände der Ressourcen bestimmen, welche Aktionen im jeweiligen Fall durchführbar sind. Eine Ladeoperation wird von einem Port durchgeführt und ist abhängig von dem Zustand der referierten Bank, des benötigten Sektionspfades und des mit der Ladeoperation verbundenen Registers. Eine Additionsfunktion wird von der Additionspipeline durchgeführt und hängt von den mit der Operation verbundenen Registern ab. Aus diesen Überlegungen heraus können Ports und Pipeline als aktive (arbeitende) Ressourcen, alle anderen als passive Ressourcen bezeichnet werden. Die gegenseitige Abhängigkeit der verschiedenen Operationen wird durch die Verfügbarkeit der Register (bei Beginn einer Operation) bzw. durch die Verfügbarkeit einzelner Vektorelemente in den Registern (Chaining der Additionsfunktion) festgelegt.

Jeder aktiven Ressource ist ein Satz von Zuständen zugeordnet. Ein Zustand (Status) einer Einheit beschreibt die Aktionsmöglichkeiten dieser Einheit im laufenden Simulationstakt. Beispielsweise stehen für ein Port unter anderem die Zustände *Execute* und *PortFinished* zur Verfügung, die entsprechend die Ausführung der Ladeoperation bzw. das Ende der Bearbeitung anzeigen. Zu Beginn jedes Simulationstaktes wird für alle Einheiten überprüft, ob aufgrund der im vorangegangenen Takt durchgeführten Aktionen eine Statusänderung erforderlich ist. Anschließend an die eventuell erfolgten Statusänderungen erfolgt die Bearbeitung der Aktionen der Ressourcen für den aktuellen Simulationstakt. Dabei können in Abhängigkeit vom jeweils vorliegenden Status nur bestimmte Aktionen vorgenommen werden. Diese Aktionen werden in der Datenstruktur der Ressource protokolliert. Zähler zeichnen bei Ausführung einer Vektoroperation dabei jeweils die Nummer des nächsten zu bearbeitenden Vektorelements des 64-elementigen Vektors sowie des langen Zeilenvektors auf. Die Vektorregister sind als Arrays entsprechend der Länge der Register ausgelegt. Für ein referiertes Element wird jeweils der Takt eingetragen, in dem es angesprochen wurde. Die Speicherbänke werden durch ein Array entsprechend der Bankanzahl repräsentiert. Der Inhalt einer Speicherzelle bezeichnet die Anzahl der Takte, für die diese Bank noch belegt ist.

Der Speicher und das Verbindungsnetzwerk werden modelliert, indem in jedem Simulationstakt zu jedem Zugriff die referierte Bank und Sektion ermittelt wird und dann aufgrund von Intra- und Interprozessorprioritäten alle Zugriffe auf ihre Durchführbarkeit geprüft werden. Für alle Ports wird das Ergebnis dieser Prüfung protokolliert. In Abhängigkeit der Ergebnisse werden dann Speicherzugriffe durchgeführt. Dabei wird angenommen, daß die Konfliktauflösung direkt an den Ports stattfindet. Diese vereinfachte Annahme wird mangels vorliegender Informationen über die Struktur des Speicherverbindungsnetzwerks getroffen und stellt eine mögliche Ursache für eventuell auftretende Abweichungen zwischen Simulations- und Meßergebnissen dar.

Parameter

Für die Simulation einer konkreten Messung können analog zur Wahl der Parameter für die Laufzeitmessung folgende Parameter eingestellt werden:

- die Anzahl der für die Abarbeitung verfügbaren Prozessoren p, $1 \leq p \leq 4$;
- der Stride $s = 2^k$, $1 \leq k \leq 64$;
- die Anzahl der abzuarbeitenden Iterationen;
- die Vektorlänge (als Vielfaches von 128).

Da die Interprozessorprioritäten für den gleichzeitigen Speicherzugriff zyklisch verändert werden, kann man sich die Prozessoren bezüglich dieser Reihenfolge kreisförmig angeordnet vorstellen (siehe Abbildung 4.1). In diesem Fall läßt sich bei parallelem Einsatz von zwei Prozessoren zwischen der Ausführung mit benachbarten

bzw. nicht benachbarten (diagonalen) Prozessoren unterscheiden. Bei der Auswertung der Messungen zeigt sich ein unterschiedliches Verhalten der Ausführung je nach Konfiguration der Prozessoren (vergleiche Abschnitt 5.3). Daher bietet die Simulation als weiteren Parameter bei Einsatz von zwei Prozessoren die Festlegung auf benachbarte bzw. diagonale Prozessoren an.

In einzelnen Resultaten der Messungen hat sich deutlich der Einfluß des Interrupts Minor Clock Cycle gezeigt (Abschnitt 5.3). Daher bietet das Programm die Option, das Auftreten des Minor Clock Cycle zu simulieren. In diesem Fall können die Dauer einer Unterbrechung und die Periodenlänge des Interrupts in Takten angegeben werden. Das erste Einsetzen des Interrupts wird durch einen Zufallszahlengenerator gesteuert. Der Interrupt wird dann jeweils demselben Prozessor zugeordnet.

In den Entwurf der Simulation geht das von der Verarbeitungsgeschwindigkeit der Ressourcen bestimmte Zeitverhalten der Operationen mit ein (Abschnitt 1.1.1). Architektur- und systemabhängige Parameter, die Einfluß auf das Laufzeitverhalten nehmen, werden als Konstanten implementiert. Architekturabhängige Größen sind Bankanzahl, Prozessoranzahl des Systems, Sektionsanzahl, Bankzykluszeit sowie alle Setup-, Startup- und Shutdown-Zeiten der Ressourcen. Dazu zählen auch die Zeiten, die für die Bearbeitung der Skalaroperationen zwischen den Vektoroperationen benötigt werden. Sie haben Einfluß auf den Zeitpunkt der Befehlsinitiierung der Vektoroperationen.

Wie zuvor erwähnt, entstehen vor und zwischen der Bearbeitung der Iterationen zusätzliche Verzögerungszeiten (siehe Abschnitt 3.3.1 und 3.3.3), die durch Messungen bestimmt werden können. Diese gehen ebenfalls in die Simulation ein. Im einzelnen sind zu nennen:

1. Multitasking-Initialisierungszeiten: die Verzögerung beim Eintreten der Prozessoren in die parallele Bearbeitung. Für den ersten Prozessor bewegen sich diese Zeiten zwischen 81 und 111 Takten, für den zweiten Prozessor zwischen 123 und 152 Takten und für den dritten Prozessor zwischen 139 und 183 Takten. Diese Fluktuationen lassen sich nicht in ein Schema einordnen. Daher

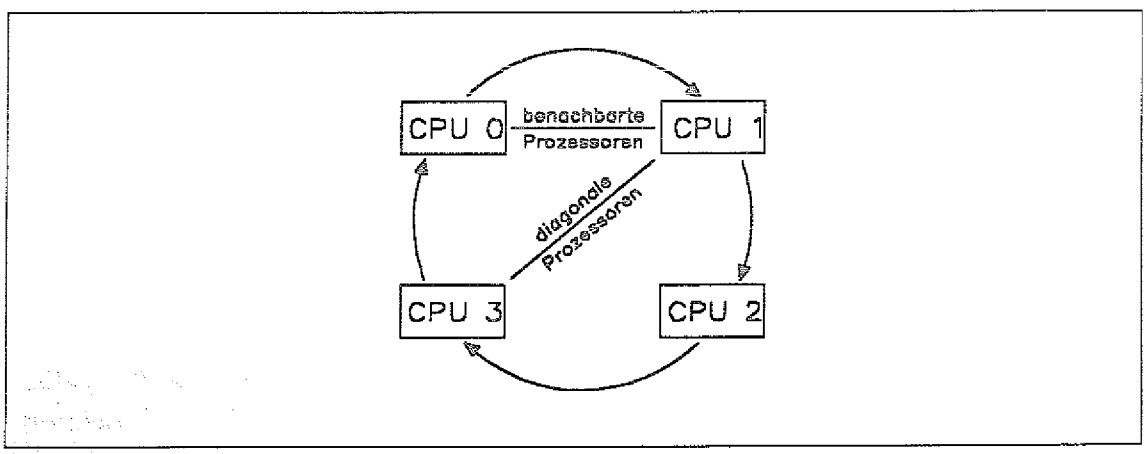


Abbildung 4.1: Zyklisches Prioritätenverfahren bei der CRAY X-MP

werden sie in der Simulation in den vorgegebenen Grenzen durch Zufallszahlen variiert.

2. Multitasking-Overhead: die Zeit zwischen dem Ende einer Iteration und dem Beginn der nächsten Iteration auf einer CPU. Die ermittelten Werte zeigen sich bei allen Messungen für einen Prozessor mit 44 Takten konstant, für mehrere Prozessoren schwanken die Werte im Bereich von 70 – 103 Takten je Messung, wobei im Mittel 71 – 75 Takte erreicht werden. Hier lassen sich zwar im Gegensatz zum vorangegangenen Punkt gewisse Schemata erkennen, jedoch sind diese zu komplex, um in der Simulation berücksichtigt zu werden. Daher werden auch hier die jeweiligen Werte im Rahmen der festgestellten Mittelwerte durch Zufallszahlen simuliert.

3. Zusätzliche Zeiten durch die FORTRAN-Instrumentierung:

   - die Zeit für die Schleifeninitialisierung (siehe Abschnitt 3.3.2).
   - der Zeitaufwand für die Vektorreduktionen: Zum Abschluß jeder Iteration werden die 64 Komponenten eines Vektorregisters in einem Skalarregister akkumuliert.

Über die realen Möglichkeiten des Systems hinaus sind optional noch Variationen der Simulation des Speicherverbindungsnetzwerks bzw. der Konfliktauflösung möglich. Ein möglicher Zugriff eines Ports auf einen freien Speicherplatz wird dann blockiert, wenn ein anderes Port derselben CPU einen Speicherzugriffskonflikt erleidet. Bei der Simulation können für den Zugriff eines Ports bei gleichzeitigem Konflikt des anderen Ports folgende Strategien optional eingestellt werden:

- eine Blockierung erfolgt durch Bank- oder Simultankonflikt (Voreinstellung);
- eine Blockierung erfolgt nur durch Bankkonflikt;
- eine Blockierung erfolgt nur durch Simultankonflikt;
- es erfolgt keine Blockierung des Zugriffs;
- beide Ports können blockiert werden (Voreinstellung);
- nur das niedriger priorisierte Port kann blockiert werden;
- nur das höher priorisierte Port kann blockiert werden.

Trotz der Berücksichtigung einer Vielzahl von Parametern können folgende zwei Einflüsse in die Simulation nicht mit einbezogen werden: Der Vorgang des Ladens von Programmcode in einen Instruktionspuffer (Code-Fetch) blockiert nicht nur für die betroffene CPU den Zugriff auf den Speicher, sondern ebenso für einen benachbarten Prozessor (vergleiche Abschnitt 1.1.3). Dabei werden 32 aufeinanderfolgende Bänke reserviert, so daß ein Zugriff auf diesen Speicherbereich auch durch die beiden anderen Prozessoren nicht mehr möglich ist. Während der Bearbeitung

der Assemblerschleife auf einer CPU findet kein Code-Fetch statt, da der Assemblercode vollständig in die Instruktionspuffer paßt. Während der Initialisierung und der Reduktion werden jedoch längere Codesequenzen bearbeitet, so daß hier das Auftreten von Ladeoperationen der Instruktionspuffer wahrscheinlich ist. Da die einzelnen Bearbeitungszeiten der Assemblerschleife auf den Prozessoren leicht versetzt beginnen und je nach Prozessor unterschiedlich lange andauern, überlappen sie sich mit Initialisierung und Reduktion auf anderen Prozessoren. Der durch diese Überlappung entstehende Einfluß kann wegen seiner Abhängigkeit von der Instruktionsbearbeitung nicht in dem taktgestützten Simulator berücksichtigt werden. Für eine detaillierte Berücksichtigung von Initialisierung und Reduktion wäre eine zusätzliche Simulation erforderlich. Dem geringen Einfluß dieses Effektes auf das Gesamtergebnis einer Messung steht jedoch ein überproportional großer Aufwand zu seiner Berücksichtigung in der Simulation gegenüber. Aus diesem Grund geht der IR-Aufwand analog zu den Overheadzeiten lediglich mit einer in der Auswertung ermittelten Abschätzung in die Simulation ein. Bei der Ermittlung der IR-Zeiten zeigt sich, daß diese sich für einzelne Parameterkombinationen unterscheiden und nicht immer eindeutig bestimmen lassen (vergleiche Abschnitt 5.2).

Ergebnisse

Ein Simulationslauf liefert folgende Ausgaben:

1. In einer *Trace-Datei* wird der zeitliche Verlauf der Zugriffe festgehalten. Ein Eintrag gibt für einen bestimmten Takt Auskunft über die Aktion jedes Ports und die Prioritätenverteilung der Prozessoren. Darüber hinaus wird noch vermerkt, welche Additionspipeline aktiv ist.

2. In einer *Protokolldatei* wird analog zu einer Messung zu jeder Iteration deren Nummer, Startzeitpunkt, Terminierungszeitpunkt, Dauer und ausführender Prozessor aufgezeichnet. Zusätzlich werden für jede Iteration noch die Simultan-, Bank-, Sektionskonflikte und Portblockierungen gezählt.

3. Die über eine Messung ermittelten Werte werden zusammengefaßt. Zu Laufzeit und allen Konfliktarten werden jeweils die Extrem- und Mittelwerte angegeben. Diese Angaben werden beiden Ausgabedateien geschrieben.

Der Simulator bietet zwei verschiedene Betriebsarten, die beide sowohl interaktiv als auch im Batch-Betrieb verwendet werden können:

1. *Detail-Mode*: Alle Resultate werden für eine festgelegte Parameterkombination Prozessoranzahl/Stride ermittelt. Dabei kann vor Beginn der Simulation die Anzahl der Simulationstakte festgelegt werden, für die der zeitliche Verlauf der Speicherzugriffe bei Ausführung des Modellproblems in die Trace-Datei geschrieben wird.

2. *Summary-Mode*: Die Simulation wird für eine Reihe von Messungen durchgeführt, wobei keine Ausgabe der Trace-Datei erfolgt und alle übrigen Ergebnisse auf die Standardausgabe umgelenkt werden. Dazu kann vor Beginn der Simulation je ein Intervall für Prozessoranzahl und Stride angegeben werden.

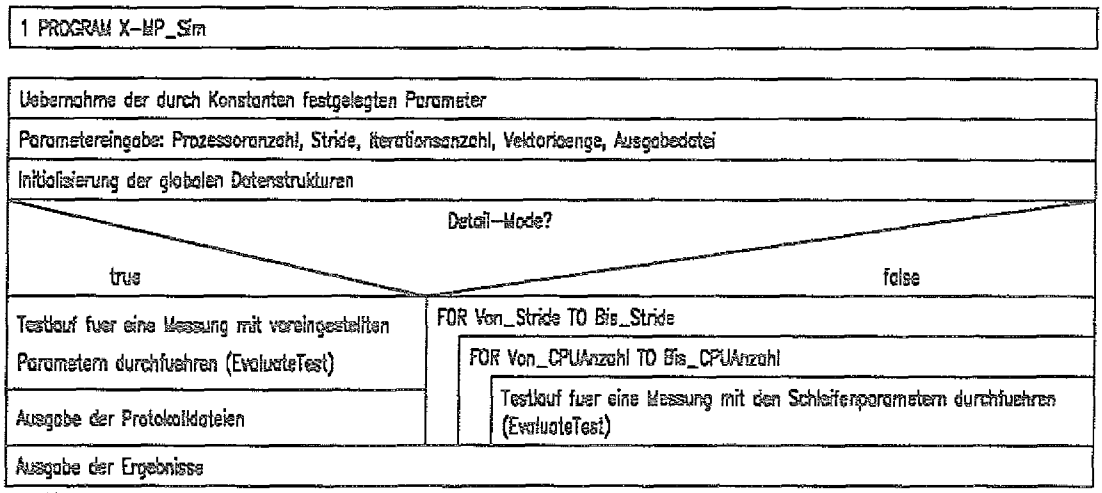


Abbildung 4.2: Das Hauptprogramm der Simulation

Algorithmus

Im folgenden soll kurz mit Hilfe von Nassi-Shneiderman-Diagrammen die Struktur des Simulationsprogramms erläutert werden. Abbildung 4.2 zeigt die Struktur des Hauptprogramms mit den Möglichkeiten des Detail- und des Summary-Mode, der für einen vorgegebenen Wertebereich von Prozessoranzahl und Stride jeweils Simulationen für einzelne Parameterkombinationen durchführt. Eine Simulation für eine

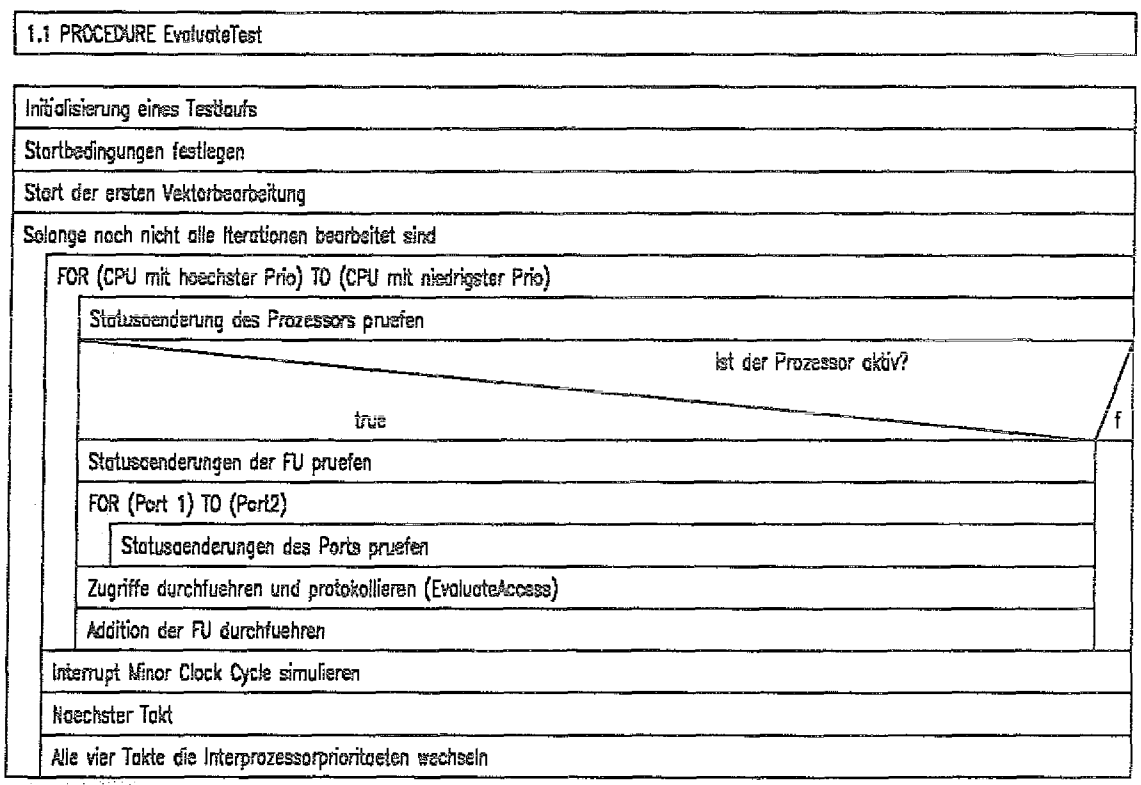


Abbildung 4.3: Simulationslauf für eine Parameterkombination

Parameterkonfiguration erstreckt sich über die Bearbeitung einer in der Initialisierung festgelegten Anzahl von Iterationen für jeden Prozessor. Im Anschluß an die Vorbesetzung der Datenstrukturen werden die Einstellungen für die erste Vektoroperation vorgenommen (siehe Abbildung 4.3, Seite 63). Hier wird das abweichende Verhalten des ersten Schleifendurchlaufs nach Abschnitt 3.3.2 anhand der Verzögerungszeiten für die einzelnen Vektoroperationen berücksichtigt. Nach dem Start der Simulation wird in jedem Simulationstakt für jeden an der parallelen Bearbeitung partizipierenden Prozessor das Zugriffsverhalten geprüft und eine eventuell stattfindende Addition registriert. Um die Interprozessorprioritäten beim gleichzeitigen Zugriff auf eine Speicherbank zu berücksichtigen, erfolgt die Bearbeitung der Aktionen einzelner Prozessoren aufeinanderfolgend in der absteigenden Reihenfolge der Interprozessorprioritäten. Auf diese Weise erhält ein priorisierter Prozessor ohne weitere Prüfung den Zugriff auf eine freie Speicherbank, die dann für im Verlauf des Programms später berücksichtigte Prozessoren als belegt gilt.

Nach der Kontrolle der Ressourcen in bezug auf Zustandsänderungen werden die Zugriffe der Ports auf den Speicher entsprechend der Abbildung 4.4 berechnet. Dabei wird zuerst der Sektionskonflikt und anschließend der Zugriff auf eine Speicherbank (Abbildung 4.5) behandelt. Wird bei der Prüfung eine Belegung der referierten Bank für 4 Takte (Bankzykluszeit) festgestellt, so hat im soeben bearbeiteten Simulationstakt bereits ein Zugriff stattgefunden. Dieser kann nur durch eine höher priorisierte CPU vorgenommen worden sein, daher liegt ein Simultankonflikt vor. Der vorläufig mögliche Erfolg des Zugriffs wird vermerkt, der Zugriff jedoch noch nicht durchgeführt. Anschließend wird kontrolliert, ob für eines der Ports ein Simultan- oder

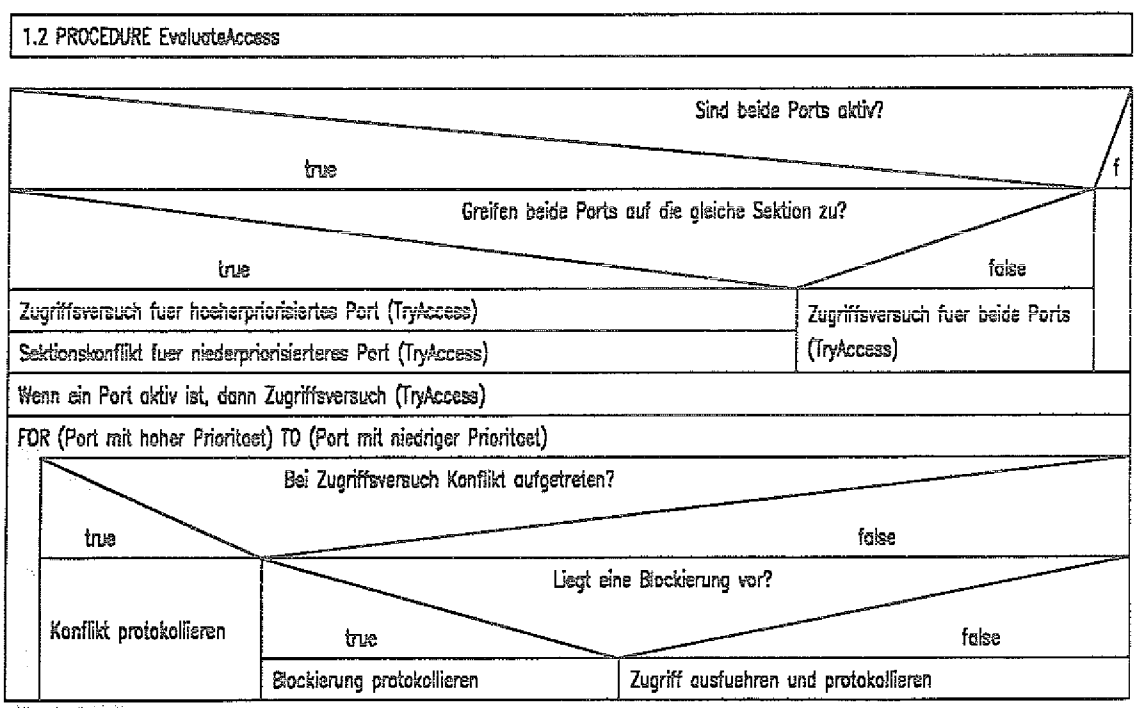


Abbildung 4.4: Durchführung und Protokollierung eines Zugriffs

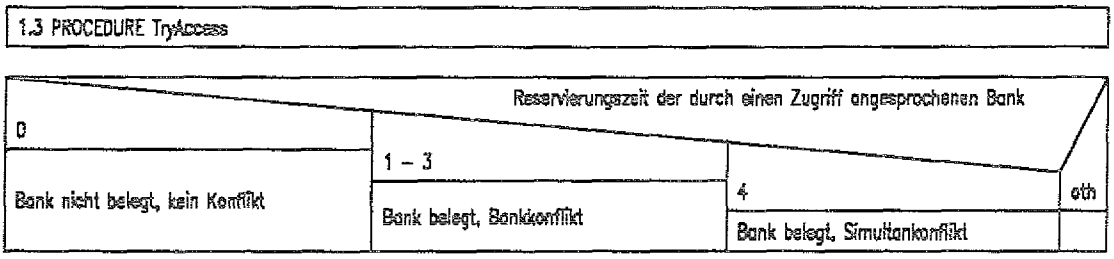


Abbildung 4.5: Zugriffsversuch durchführen

Bankkonflikt aufgetreten ist. In diesem Fall wird eine Portblockierung für das andere Port protokolliert.

Das Auftreten des Interrupts Minor Clock Cycle wird in der Weise simuliert, daß beim Start des Programms durch einen Zufallsgenerator ein Zeitpunkt für das erste Auftreten festgelegt wird. In jedem Simulationstakt findet nach der erfolgten Bearbeitung der Prozessoraktivitäten ein Vergleich statt, ob der Zeitpunkt für die Simulation des Interrupts erreicht ist. In diesem Fall wird für eine festgelegte Zeit der Prozessor 0 in einen Wartezustand versetzt, in der er keine Aktionen durchführt und die Zustände der ihm zugeordneten Ressourcen nicht verändert werden. Nach seinem ersten Auftreten erfolgt die Simulation des Interrupts periodisch.

In jedem vierten Simulationstakt erfolgt eine Änderung der Interprozessorprioritäten. Die Prioritäten werden anfangs entsprechend der Prozessornumerierung 0 – 3 vergeben, wobei 0 die höchste Priorität darstellt. Sie wechseln zyklisch, dabei wird für jeden Prozessor die neue Priorität P mit

$$P = (alte Priorität + 1)\, mod\, 4$$

berechnet.

Kapitel 5

Ergebnisse

In diesem Kapitel sollen die experimentellen Ergebnisse für die Systeme CRAY X-MP und CRAY Y-MP vorgestellt und im Zusammenhang mit der jeweiligen Architektur diskutiert werden. Im folgenden Abschnitt wird eine auf beiden Rechnern identische Meßumgebung vorgestellt. Im Anschluß an eine darauf folgende Zusammenfassung der für die Auswertung auf beiden Rechnersystemen gemeinsamen Voraussetzungen werden die für die CRAY X-MP ermittelten Ergebnisse vorgestellt. Dabei werden Modellberechnungen und Simulationsergebnisse den Laufzeitmessungen gegenübergestellt. In einem weiteren Abschnitt werden die Ergebnisse der Simulation genauer betrachtet. Abschließend werden die auf der CRAY Y-MP ermittelten Laufzeiten vorgestellt.

5.1 Die Meßumgebung

Alle Messungen auf den Rechnern CRAY X-MP und CRAY Y-MP wurden unter dem Betriebssystem UNICOS Release 6.1.4 vorgenommen. UNICOS ist ein Multiuser-Multitasking-Betriebssystem, das hauptsächlich auf dem *AT&T UNIX System V* basiert und einige Erweiterungen der *Fourth Berkeley Software Distribution (BSD)* enthält [Rei88a]. Der Zugang zu den CRAY-Systemen am Forschungszentrum Jülich wird über *remote job entry* ermöglicht. Ein Job eines Benutzers wird durch eine Datei beschrieben, die NQS (Network Queueing System) [Cra91b] Befehle zur Beschreibung der Jobumgebung und UNICOS Befehle zur Jobsteuerung enthält. Die Datei wird mit dem KFAnet/Internet [For91] an das Zielsystem befördert und dort interpretiert. Des weiteren gibt es eine Möglichkeit zur Submittierung von Jobs vom vorgeschalteten IBM-Rechner ES/9000 über eine Kanalkopplung. Der beschriebene Job wird bearbeitet, und der Benutzer erhält anschließend ein Laufzeitprotokoll zurück. Aus Sicht des Benutzers handelt es sich also um eine Batch-Verarbeitung. Darüber hinaus ist eine interaktive Nutzung der CRAY X-MP ebenfalls möglich.

Es hat sich herausgestellt, daß Laufzeitmessungen in einer solchen Multiuser-Umgebung mit unbekannter Arbeitslast keine repräsentativen Ergebnisse liefern können, da eine Reihe von Einwirkungen das Laufzeitverhalten eines Programms beeinflussen. In dieser Umgebung konkurriert das Meßprogramm mit allen übrigen

auf der Maschine laufenden Programmen um die Ressourcen des Systems. Daher treten nichtdeterministisch Situationen auf, in denen nicht alle angeforderten Prozessoren zur Verfügung gestellt werden können oder Prozessoren während der laufenden Bearbeitung des Meßprogramms entzogen werden. Weitere Einflüsse ergeben sich durch parallel zum Meßprogramm ablaufende Jobs anderer Benutzer, deren Speicherzugriffe nicht reproduzierbare Konfliktsituationen verursachen. Für die hier vorgenommenen Untersuchungen werden jedoch möglichst exakte Meßergebnisse benötigt, da die zu messenden Laufzeitabweichungen im Bereich von nur wenigen Takten liegen. Alle Messungen werden deshalb im *dedizierten Betrieb* vorgenommen, in dem der Rechner ausschließlich für die Bearbeitung des Meßprogramms reserviert ist. Andere Benutzer erhalten für die Zeit der Messung keinen Zugang zum betroffenen System. Dazu wurde eigens eine Betriebssystemumgebung installiert, in der ein CRAY-Rechner interaktiv im Einzelbenutzerbetrieb (Single-User-Mode) betrieben wird. Da diese Umgebung für alle Messungen verwendet wurde, soll sie im folgenden auch als Meßumgebung bezeichnet werden. Während einer Sitzung stehen also für das Meßprogramm alle Ressourcen des Systems zur freien Verfügung, und eine Beeinflussung durch andere Benutzer ist damit ausgeschlossen.

Das Betriebssystem UNICOS besteht aus einem UNIX-Kern und vielen asynchron ablaufenden *Dämonprozessen* (Daemons). Ein Dämonprozeß ist ein Systemprozeß, der durch einen Interrupt oder durch einen äußeren Einfluß nur für einen beschränkten Zeitraum aktiv wird. Diese Prozesse sind zum korrekten Betrieb des Systems notwendig. Da das Auftreten eines Dämonprozesses aus Sicht eines Benutzers nicht vorhersagbar ist, können solche Prozesse nicht reproduzierbare Störungen im Verlauf einer Messung verursachen. Alle für den Einzelbetrieb nicht notwendigen Systemdienste wurden daher in der verwendeten Meßumgebung ausgeschaltet, jedoch sind wegen der Komplexität des UNIX-Betriebssystems auch weiterhin Störungen nicht grundsätzlich auszuschließen. Um eine realistische Basis für allgemeine Aussagen zu erhalten, wurden die Ergebnisse der Laufzeitmessungen – wie in Abschnitt 3.2.4 beschrieben – mittels Referenzmessungen und Messungen mit der Routine ICPUSED überprüft. Diese Vorgehensweise gewährleistet, daß die hier präsentierten Meßergebnisse als repräsentativ für das Ausführungsverhalten des Meßprogramms unter dieser Meßumgebung angesehen werden können. Die Kontrollmessungen mit ICPUSED zeigten nur einzelne Anhaltspunkte für Einflüsse des Betriebssystems im Zusammenhang mit dem Interrupt Minor Clock Cycle. Unter UNICOS 6.1 stehen zwei Konzepte für das Prozeßscheduling bei Multitasking zur Verfügung (vergleiche Abschnitt 2.2). Für eine Messung in dedizierter Umgebung erscheint die Verwendung des Cooperative Parallel Interface, das auf eine Multiuser-Umgebung zugeschnitten ist, nicht sinnvoll. Daher werden alle Messungen unter der Voreinstellung MP_DEDICATED=1 vorgenommen.

5.2 Gemeinsame Voraussetzungen

Das Meßprogramm wird mit den variablen Parametern Prozessoranzahl und Stride auf beiden betrachteten Systemen ausgeführt. Alle übrigen Parameter wer-

den für die Laufzeitbestimmung konstant und für beide Systeme übereinstimmend gewählt. Damit alle Effekte über einen genügend langen Zeitraum beobachtet werden können und der Einfluß eventueller Störungen im Gesamtergebnis so gering wie möglich bleibt, wird mit $vl = 2^{16} = 65536$ die Spaltenanzahl der Matrix und damit die Zeilenvektorlänge sehr groß gewählt. Die sich gegenseitig beeinflussenden Prozesse erzeugen Auswirkungen, die sich über die Dauer der Berechnung eines Zeilenvektors hinaus bemerkbar machen. Für die Darstellung und Auswertung hat sich eine Beobachtungsdauer von 60 Iterationen als ausreichend und zudem übersichtlich erwiesen. Um den repräsentativen Aussagewert für diese festen Parameter zu kontrollieren, wurden Messungen mit unterschiedlichen Vektorlängen zwischen 2^8 und 2^{15} sowie mit bis zu 1000 Iterationen durchgeführt. Dabei wurde weitgehend Übereinstimmung mit den Meßergebnissen für die hier fest gewählten Parameter erzielt. Auf gelegentlich auftretende Ausnahmefälle bei der CRAY X-MP soll im Verlauf der Auswertung eingegangen werden. Die Anzahl der bearbeiteten Iterationen teilt sich auf die für die Berechnung zur Verfügung stehenden Prozessoren auf. Mit steigender Prozessoranzahl sinkt die im Durchschnitt von jedem Prozessor zu bearbeitende Iterationsanzahl. Zu beachten ist, daß die insgesamt bearbeitete sowie die in jeder Iteration von einem Prozessor zu bearbeitende Arbeitslast konstant bleiben.

Über die Schrittweite, mit der ein Vektorzugriff auf den m-fach verschränkten Speicher zugreift, kann die Zugriffsmenge und damit die Bandbreite des Speichers auf beiden betrachteten Systemen eingeschränkt werden. Eine Untersuchung der Zugriffsmengen für alle Strides von 1 bis 256 ergibt, daß in allen Fällen die Anzahl der referierten Bänke eine Zweierpotenz ist. Zugriffe auf diese Bankanzahlen lassen sich exemplarisch durch Verwendung der Strides 1 bis 256 in Zweierpotenz-Schritten erreichen. In diesen Fällen berechnet sich die Anzahl der von einem mit einem Stride d_i zugreifenden Zugriffsstrom V_i genutzten Bänke b_i zu $b_i = \frac{m}{d_i}$. Daher werden sich alle Messungen auf diese Teilmenge der Strides beschränken. Für den sequentiellen Fall wurden ähnliche Situationen bereits ausführlich untersucht [Oed85, Hof90]. Von Interesse sind im vorliegenden Fall vor allem die durch den Zugriff mehrerer Prozessoren entstehenden Konflikte und ihre Auswirkungen. Auf dem jeweiligen System wird daher das Meßprogramm mit variierender Prozessoranzahl für jeden betrachteten Stride ausgeführt. Tabelle 5.1 faßt die Parameter für die in der jeweiligen Auswertung betrachteten Laufzeitmessungen auf beiden Rechnern zusammen.

Parameter	CRAY X-MP/416	CRAY Y-MP8/832
Prozessoranzahl	1 – 4	1 – 8
Strides in Zweierpotenzen	1 – 64	1 – 256
Vektorlänge	$2^{16} = 65536$	
Anzahl Iterationen	60	

Tabelle 5.1: Parameter der Messung auf CRAY X-MP und CRAY Y-MP

Eine *Meßreihe* umfaßt alle Ergebnisse der Laufzeitmessung für jede Kombination aus Prozessoranzahl und Stride. Alle Messungen einer Meßreihe werden direkt aufeinanderfolgend durchgeführt. Die Dauer zur Durchführung einer Meßreihe bewegt sich in der Größenordnung von etwa 171 Millionen Takten (1.45 Sekunden) auf der CRAY X-MP und etwa 520 Millionen Takten (3.12 Sekunden) auf der CRAY Y-MP. Diese Zeiten entsprechen nicht den CPU-Zeiten, da große Teile der Bearbeitung parallel stattfinden und so wesentlich mehr CPU-Zeit verbraucht wird. Zu jeder Messung werden für alle Iterationen jeweils die Nummer, der Startzeitpunkt, der Terminierungszeitpunkt, die Dauer und die Prozessorkennung in einer Protokolldatei gespeichert. Das für eine Meßreihe anfallende Datenvolumen beträgt etwa 108 KByte bei der CRAY X-MP und 278 KByte bei der CRAY Y-MP. Der relativ geringe Meßaufwand ermöglicht eine häufige Wiederholung einzelner Meßreihen. Auf diese Weise kann die Reproduzierbarkeit der Ergebnisse überprüft werden. Die Auswertung nimmt Bezug auf eine beispielhafte Meßreihe. In den Fällen, in denen sich das vorgestellte Ergebnis nicht als repräsentativ erweist, wird auf die Ergebnisse anderer Meßreihen zurückgegriffen.

Da durch den Stride die Zugriffsmenge der genutzten Bänke festgelegt wird, erscheint es sinnvoll, die Auswertung der Meßwerte nach dem Stride zu gruppieren. Die Meßergebnisse werden in aufsteigender Reihenfolge zu jedem Stride betrachtet. Jede Messung umfaßt die Laufzeiten von 60 Iterationen, daher werden für eine übersichtliche Auswertung je Messung Minimal-, Maximal- und Mittelwerte angegeben. Eine intensivere Untersuchung einzelner Messungen wird durch eine graphische Darstellung ermöglicht. Die Werte des analytischen Modells werden unter Berücksichtigung der verfügbaren relativen Bandbreite und der Anzahl der parallel zugreifenden Prozessoren ermittelt. Sie geben also bezüglich einer bestimmten Zugriffssituation ein Maß für die unter optimalen Bedingungen minimal möglichen Ausführungszeiten an und können deshalb als Prognose für die von einer Messung erwartete Laufzeit herangezogen werden. Die Höhe der Abweichung einer Messung zeigt an, wie stark diese durch Zugriffskonflikte behindert wird.

Bei den Einzelprozessormessungen können keine Interprozessorkonflikte auftreten, eventuell auftretende Abweichungen sind daher ausschließlich auf Systemeinflüsse zurückzuführen und müssen von der Betrachtung ausgeschlossen werden. Da die Minimalwerte unter diesen Umständen einen konfliktfreien Verlauf des Meßprogramms repräsentieren, sind nur sie für den Vergleich mit den prognostizierten Ausführungszeiten relevant. Bei den Mehrprozessormessungen können nicht weiter die Laufzeiten einzelner Iterationen betrachtet werden, da sich die ausführenden Prozessoren gegenseitig beeinflussen. Daher werden in diesen Fällen für den Vergleich mit den prognostizierten Werten die Mittelwerte dieser Messungen herangezogen. Diese Daten werden jeweils in Tabellen bezüglich eines festen Strides angegeben. Eine einzelne Messung soll im folgenden durch die Angabe ihrer beiden Parameter Prozessoranzahl und Stride identifiziert werden, so steht z.B. 4/32 für die Messung mit 4 Prozessoren und Stride 32.

5.3 Messung und Simulation auf der CRAY X-MP

Im Verlauf der folgenden Abschnitte erfolgt die Auswertung der Laufzeitmessungen auf der CRAY X-MP im Vergleich mit den Ergebnissen aus analytischem Modell und Simulation. Im ersten Schritt erfolgt eine Gegenüberstellung der Ergebnisse der drei Untersuchungsmethoden nach aufsteigendem Stride. Dabei sollen einzelne Meßergebnisse diskutiert und anhand analytischer Methoden untersucht werden. Im zweiten Schritt folgen genauere Betrachtungen der Simulationsergebnisse und – soweit möglich – die Verifizierung der analytischen Betrachtungen.

Für Simulation und Laufzeitmessung sind alle Parameter übereinstimmend gewählt worden. Die Angaben der Laufzeitmessung und Simulation umfassen für jede Messung Minimal-, Maximal- und Mittelwerte aus 60 Iterationen. Zu Simulation und Modell sind jeweils die Abweichungen zur entsprechenden Laufzeitmessung in Prozent angegeben.

Während in der Simulation die IR-Zeiten für Initialisierung und Vektorreduktion berücksichtigt werden können, müssen diese für das Modell aus verschiedenen Messungen errechnet werden (siehe Abschnitt 3.3.3). Dabei wird wie folgt verfahren: Für Vektorlängen $2^i, i \in \{8, \ldots, 16\}$ werden jeweils mehrere komplette Meßreihen durchgeführt. Mit den Ergebnissen wird für jede Kombination der Parameter Prozessoranzahl und Stride der jeweilige IR-Aufwand berechnet. Dabei werden jeweils die kürzesten Laufzeiten einer Messung herangezogen, da alle anderen Ergebnisse durch Konflikte beeinflußt sind. Einschränkungen bei der Genauigkeit der Aussagen sind für Kombinationen mit mehreren Prozessoren zu erwarten. Entgegen der Voraussetzung zur Anwendung des Verfahrens sind die Meßwerte in diesen Fällen konfliktbehaftet. Tabelle 5.2 gibt einen Überblick über die errechneten Zeiten. Als Häufungswerte werden Meßpunkte bezeichnet, die überdurchschnittlich oft bestimmt wurden. Wie erwartet lassen sich für die meisten Mehrprozessormessungen keine realistischen Werte angeben. Dagegen erscheinen die Angaben für einen Prozessor zuverlässig, da die ermittelten Werte für alle Vektorlängen übereinstimmen.

	1 Prozessor	2 – 4 Prozessoren	
		Bereich	Häufungswerte
Stride 1, 2	313	306 – 309	308
Stride 4	313	269 – 419	—
Stride 8	346	—	—
Stride 16	313	—	—
Stride 32	296	—	—
Stride 64	262	—	—

Tabelle 5.2: Für den IR-Aufwand auf der CRAY X-MP ermittelte Zeiten (in Takten)

Auffallend ist die deutlich sichtbare Abhängigkeit der Zeiten vom gewählten Stride. Dabei ist tendenziell mit zunehmendem Stride eine Verringerung der IR-Zeiten zu beobachten. Der Grund für die unterschiedlichen Ergebnisse dürfte in der überlappenden Bearbeitung von Reduktion und letzter Vektoraddition der Assemblerschleife zu finden sein. Eine genauere Analyse dieses Zeitverhaltens kann im Rahmen dieser Arbeit nicht durchgeführt werden.

Da die IR-Zeiten der Mehrprozessormessungen nicht genau ermittelt werden können, werden in diesen Fällen die dem Stride entsprechenden Zeiten für einen Prozessor verwendet. Für den Vergleich mit den Meßergebnissen wird zum jeweils ermittelten Modellwert der IR-Aufwand für den entsprechenden Stride addiert. Durch diese Vorgehensweise entstehen zwangsläufig leichte Fehler der Modellwerte, deren Umfang jedoch im Vergleich zur für die Messungen verwendeten Vektorlänge von 2^{16} vernachlässigt werden kann.

Bevor die einzelnen Ergebnisse gegenübergestellt und diskutiert werden, sollen noch einige allgemein auftretende Effekte behandelt werden. Bei einer Anzahl von Messungen zeigen sich sporadisch einzelne Abweichungen, die sich durch deutlich vergrößerte Laufzeiten bei einzelnen Iterationen bemerkbar machen. Sie erweisen sich als nicht repräsentativ und lassen sich daher auch nicht auf das Meßprogramm zurückführen. In anderen Fällen treten diese Störungen periodisch auf. Abbildung 5.1 zeigt diesen Effekt exemplarisch am Beispiel einer Messung mit einem Prozessor und Stride 64. In der Graphik sind die Ausführungszeiten der einzelnen Iterationen dargestellt und zur Verdeutlichung miteinander verbunden. Bestimmt man in diesem Fall die mittlere Periodenlänge zwischen den verzögerten Iterationen, so erhält man etwa 1.971.700 Takte. Multipliziert mit der Zykluszeit von

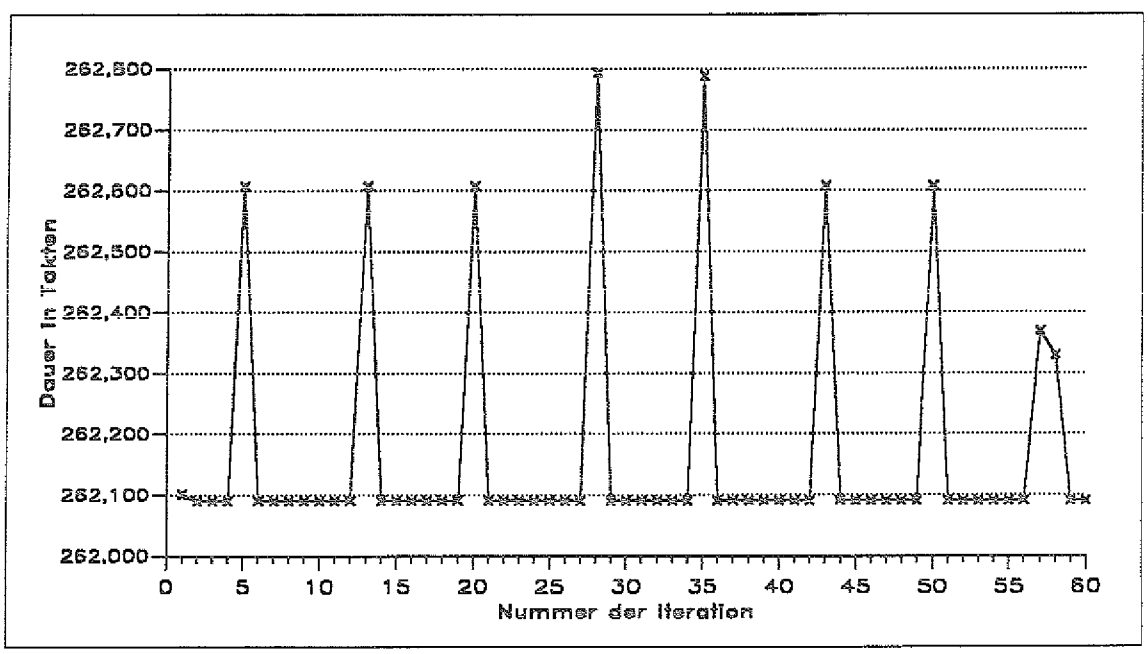


Abbildung 5.1: Auswirkung des Minor Clock Cycle bei einem Prozessor und Stride 64

8.5 *nsec* der CRAY X-MP ergibt sich so eine Periode von etwa einer 60stel Sekunde. Wegen der Bearbeitung des Meßprogramms im Multitasking-Betrieb ist ein Auftreten des Interrupts Minor Clock Cycle mit einer Periode von einer 60stel Sekunde zu erwarten. Diese Übereinstimmung erlaubt eine Rückführung der periodischen Störungen auf diesen Interrupt. Messungen mit einzeln auftretenden Abweichungen liegen in der Meßdauer alle unterhalb von einer 30stel Sekunde, also unterhalb der doppelten Periodenlänge des Minor Clock Cycle. Erst bei Überschreitung der doppelten Periodenlänge ist eine Beobachtung von mindestens zwei Störungen sicher. Diese Messungen wurden mit größeren Iterationsanzahlen ausgeführt, wobei sich ebenfalls Störungen mit der beobachteten Periode einstellten. Der Einfluß dieser Verzögerungen ist ebenso unterschiedlich wie die durch sein Auftreten verursachten Auswirkungen. Für Einzelprozessormessungen liegt die Verzögerung einzelner Iterationen zwischen 70 und 950 Takten, offensichtlich unabhängig vom verwendeten Stride. Im Verlauf einer Messung mit mehreren Prozessoren ist immer dieselbe CPU von der Verzögerung betroffen. Für Strides unter 16 liegen dabei die Abweichungen der betroffenen Iterationen um etwa 24.000 Takte über den jeweiligen Minimalwerten, nehmen aber keinen Einfluß auf die weitere Bearbeitung. Bei Strides größer 16 liegen die Abweichungen meist bei 14.000 Takten, wirken sich dann aber auch auf die Bearbeitung anderer Iterationen aus. Bei einigen Messungen ist das Auftreten des Interrupts nicht zu erkennen. Da das Ausführungsverhalten derart unterschiedlich erscheint, ist eine generelle Vernachlässigung dieses Einflusses nicht möglich. Die gemittelten Meßergebnisse würden bei kurzen Ausführungszeiten durch die Unterbrechungen erheblich verfälscht, daher werden in solchen Einzelfällen die Laufzeiten betroffener Iterationen auf den Mittelwert der nicht vom Interrupt betroffenen Iterationen gesetzt. Diese Veränderungen der Meßwerte werden nur dann vorgenommen, wenn die Verzögerungen eindeutig erkennbar und ohne Auswirkung auf die übrigen Meßwerte sind.

Für alle Einprozessormessungen liegt die Dauer der ersten Iteration um 11 Takte über der Dauer der übrigen Iterationen, die außer bei Störung durch den Minor Clock Cycle immer identisch sind. Eine Erklärung dieses Verhaltens allein aufgrund der Messungen ist nicht möglich.

Stride 1

Die Ausführung des Meßprogramms mit Stride 1 stellt für das Zugriffsverhalten den optimalen Fall dar, da alle 64 Bänke des Systems angesprochen werden können. Insbesondere sind für die Ausführung mit einem Prozessor keine Zugriffskonflikte zu erwarten. Auch für den Zugriff von mehreren Prozessoren sollten die auftretenden Verzögerungen nur geringen Einfluß auf die Bearbeitung nehmen. Tabelle 5.3 (Seite 74) stellt die Ergebnisse von analytischem Modell und Simulation der Laufzeitmessung gegenüber. Zu Laufzeitmessung und Simulation sind je Prozessor immer Minimum, Maximum und der aus allen Iterationen eines Durchgangs berechnete Mittelwert angegeben. Der für das Modell angegebene Wert beschreibt für mehrere Prozessoren immer den aufgrund der verfügbaren Bandbreite zu erwartenden Mittelwert der Iterationen. Da die Abweichungen vom Minimalwert bei einem Pro-

Prozessoren		1	2	3	4
Messung	Minimum	69944	69940	69940	69940
	Mittelwert	69944	69944	69943	69945
	Maximum	69955	69999	69981	69970
Analytisches Modell		69945	69945	69945	69945
Abweichung von Messung		0%	0%	0%	0%
Simulation	Minimum	69943	69969	69969	69969
	Mittelwert	69943	69971	69971	69971
	Maximum	69943	69977	69976	69973
Abweichung von Messung		0%	0.03%	0.04%	0.03%

Tabelle 5.3: Ausführungszeiten (in Takten) für Stride 1

zessor nicht durch Interprozessorkonflikte verursacht sein können, bezieht sich die prozentuale Abweichung in den entsprechenden Fällen auf die Minimal- und sonst auf die Mittelwerte.

Der Tabelle sind für alle Fälle nur geringfügige Abweichungen zwischen den Ergebnissen aller drei Methoden zu entnehmen. Auch für massive Speicherzugriffe durch mehrere Prozessoren treten keine signifikanten Konfliktsituationen auf. Die Übereinstimmung von Modell und Laufzeit für alle Messungen bestätigt die grundsätzlichen Überlegungen für das analytische Modell. Auffallend und mit vorliegenden Informationen nicht erklärbar ist die für einen Prozessor etwas geringer ausfallende Streuung der Meßwerte. Die mittleren Abweichungen der Messungen vom jeweiligen Minimalwert sind gering.

Abbildung 5.2 zeigt die graphische Darstellung der Messung 4/1, aus der Details über die Dauer einzelner Iterationen und ihrer gegenseitigen Beeinflussung zu entnehmen sind. Die Iterationen sind mit ihrer Nummer fortlaufend auf der Abszisse aufgetragen. Zu jeder Iterationsnummer werden in der Graphik zwei Einträge vorgenommen. Jeder dieser Meßpunkte besteht dabei aus einem Marker, der die Ausführung der Iteration mit einem bestimmten Prozessor symbolisiert. Die Zuordnung zwischen Prozessoren und Markern ist der Legende über der Graphik zu entnehmen. Die durch eine Linie verbundenen Meßpunkte liefern im Zusammenhang mit der linken Ordinatenachse Angaben über die jeweiligen Laufzeiten einer Iteration in Takten. Die Verbindung der einzelnen Meßpunkte dient dabei lediglich der Übersichtlichkeit. Auf diese Weise lassen sich eventuell auftretende Muster leicht erkennen. Die nicht verbundenen Meßpunkte zeigen im Zusammenhang mit der rechten Ordinatenachse die Startzeitpunkte der jeweiligen Iterationen und erscheinen in der Graphik auf der Winkelhalbierenden. So erhält man einen Überblick über die Reihenfolge, mit der die einzelnen Prozessoren Iterationen bearbeiten. Liegen die Marker benachbarter Startzeitpunkte horizontal nebeneinander, so werden die entsprechenden Iterationen parallel ausgeführt. Ein daraus ersichtliches und für alle

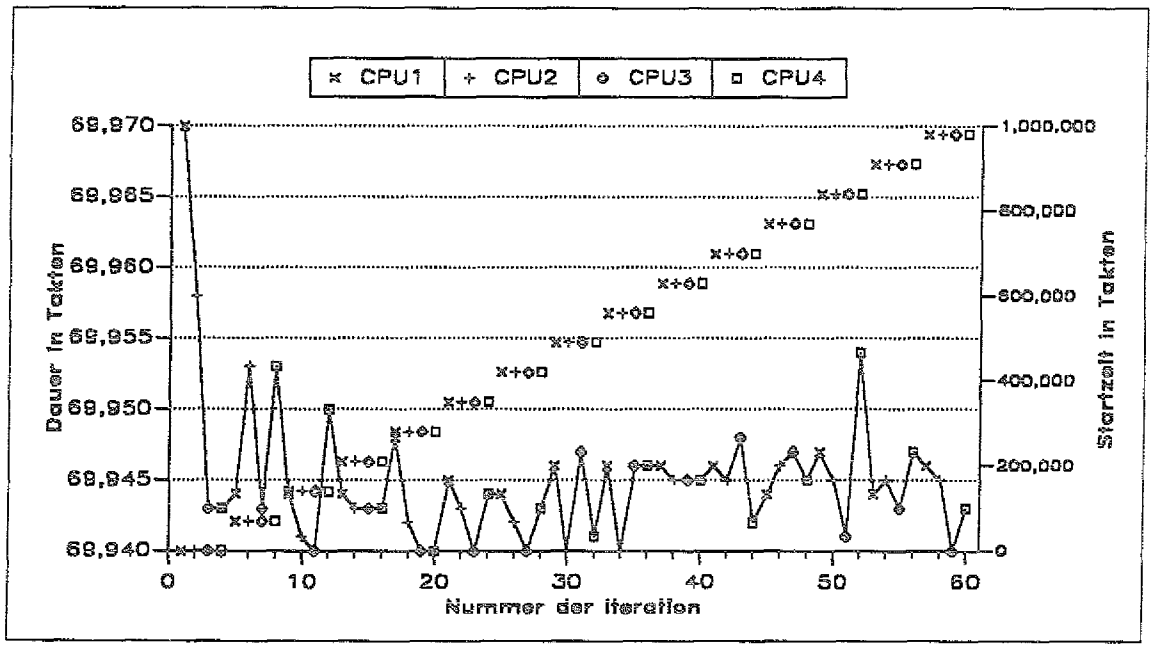


Abbildung 5.2: Meßergebnis bei 4 Prozessoren und Stride 1

Messungen allgemeingültiges Ergebnis ist die Feststellung, daß die einzelnen Iterationen chronologisch entsprechend ihrer natürlichen Reihenfolge ausgeführt werden. In keinem Fall wird die Bearbeitung einer Iteration mit einer höheren Nummer vor einer mit niedrigerer Nummer vorgenommen.

Anhand Abbildung 5.2 ist zu erkennen, daß der Großteil der Ausführungszeiten bei 4 Prozessoren unregelmäßig im Bereich zwischen 69.940 (dem Minimalwert) und 69.953 Takten schwankt. Die Darstellung der Startzeiten bestätigt die parallele Ausführung von jeweils vier aufeinanderfolgenden Iterationen. Es ist zu vermuten, daß die Abweichungen durch Interprozessorkonflikte während des ersten Vektorzugriffs verursacht werden. Während dieses Zugriffs werden die über das Port geladenen Elemente direkt zur Weiterverarbeitung an die Pipeline geleitet (siehe Abbildung 3.5). Tritt in dieser Phase eine Verzögerung eines Zugriffs auf, so erfolgt ebenfalls eine Verzögerung der Berechnung. Die gesamte Ausführungszeit wird also um die Anzahl der während der ersten Vektorladeoperation auftretenden Verzögerungen verlängert.

Für 2 und 3 Prozessoren stellen sich die Abweichungen als weitgehend regelmäßige Muster dar. Bei der Messung 3/1 ist die Ausführungszeit eines Prozessors mit 69.944 Takten konstant, während Werte der beiden anderen Prozessoren darunter liegen und leicht schwanken. Die Ausführungszeiten der einzelnen Iterationen bei einem Prozessor sind bis auf die Dauer der ersten Iteration identisch. Ein ähnliches Verhalten macht sich auch bei den anderen Messungen zum Stride 1 bemerkbar. Die jeweils erste Iteration auf einem Prozessor erfährt dabei eine Verzögerung der Größenordnung zwischen 11 und 55 Takten.

Stride 2

Für Zugriffe mit Stride 2 steht nur noch die Hälfte der Bänke zur Verfügung. Daher ist mit einer Häufung von Interprozessorkonflikten zu rechnen, die sich jedoch nicht in größerem Maße auf das Zugriffsverhalten auswirken dürften. In Tabelle 5.4 ist gegenüber Stride 1 ein leichter Anstieg der Laufzeit für die Bearbeitung mit einem Prozessor zu vermerken, während die Minimalwerte der übrigen Messungen im Vergleich identisch sind. An den Mittelwerten läßt sich der erwartete Anstieg

Prozessoren		1	2	3	4
Messung	Minimum	69951	69940	69940	69940
	Mittelwert	69951	69951	69958	69966
	Maximum	69962	69990	70006	70007
Analytisches Modell		69945	69945	69945	69945
Abweichung von Messung		0%	0%	0.01%	0.03%
Simulation	Minimum	69943	69969	69969	69969
	Mittelwert	69943	69970	69971	69971
	Maximum	69943	69975	69979	69975
Abweichung von Messung		0.01%	0.02%	0.01%	0%

Tabelle 5.4: Ausführungszeiten (in Takten) für Stride 2

erkennen, der erwartungsgemäß gering ausfällt. Abbildung 5.3 zeigt, daß bei der Messung 2/2 über einen längeren Zeitraum ein periodisches Verhalten festzustellen ist. Während die Ausführungszeiten eines Prozessors konstant bleiben, benötigt der

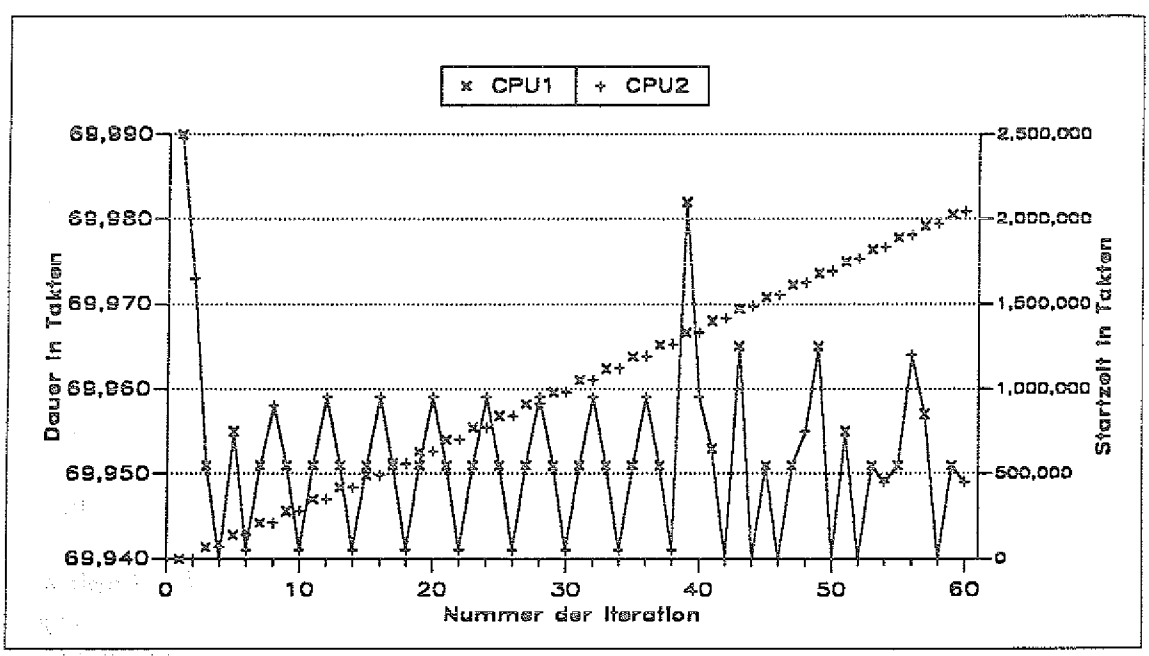


Abbildung 5.3: Meßergebnis bei 2 Prozessoren und Stride 2

andere abwechselnd 10 Takte weniger bzw. 8 Takte mehr. Die Schwankungen bei 3 und 4 Prozessoren erweisen sich mit steigender Prozessoranzahl als intensiver und lassen keine Muster mehr erkennen. Insgesamt bewegen sich die Störungen einzelner Iterationen im Bereich von weniger als 70 Takten. Gerechnet auf die Vektorlänge von 65536 Elementen ist dieser Einfluß verschwindend gering.

Stride 4

Für Stride 4 berechnet sich die relative Bandbreite bei 16 verfügbaren Bänken zu $B_4 = \frac{64}{4*4} = 4$. Für maximal 4 zugreifende Prozessoren ist also prinzipiell noch eine Transfer von durchschnittlich einem Wort pro Takt und Prozessor gewährleistet, so daß es aus dieser Sicht nicht zu erheblichen Verzögerungen kommen sollte. Da jedoch jeder Prozessor mit 2 Ports gleichzeitig zugreift, kann es durch die zu hohe Speicherbelastung bei 3 und 4 Prozessoren bereits zu Interprozessorkonflikten kommen. Tabelle 5.5 ist zu entnehmen, daß für einen Prozessor die Ergeb-

Prozessoren		1	2	3	4
Messung	Minimum	69951	69943	69955	87069
	Mittelwert	69951	69980	70056	102733
	Maximum	69962	70130	70333	108199
Analytisches Modell		69945	69945	69945	69945
Abweichung von Messung		0%	0.05%	0.15%	46.87%
Simulation	Minimum	69959	69985	69976	91571
	Mittelwert	69959	69987	70089	110670
	Maximum	69959	69989	70604	136568
Abweichung von Messung		0.01%	0.01%	0.04%	7.17%

Tabelle 5.5: Ausführungszeiten (in Takten) für Stride 4

nisse mit der entsprechenden Messung bei Stride 2 übereinstimmen. Im Mittel wird jede Iteration bei 2 Prozessoren um 37 Takte und bei 3 Prozessoren sogar um 101 Takte verzögert. In keinem Fall wird der Minimalwert der vorangegangenen Strides erreicht. Die Ausführung mit 2 Prozessoren zeigt anfangs wiederum für einige Iterationen ein regelmäßiges Verhalten, danach werden die Störungen stärker und fallen unregelmäßig aus. Die Abweichungen einzelner Iterationen bewegen sich dabei in den meisten Fällen zwischen 10 und 91 Takten verglichen mit dem Minimalwert. Für 3 Prozessoren liegen die meisten Abweichungen im Bereich zwischen 16 und 172 Takten und fallen damit wesentlich stärker aus als in allen bisherigen Messungen. Das Ergebnis der Messung mit 4 Prozessoren fällt bei einer mittleren Abweichung von etwa 47 % gegenüber dem Modell deutlich höher aus, als durch den eingangs geäußerten Hinweis auf die Interprozessorkonflikte vermutet. Während die Simulationsergebnisse bisher gute Übereinstimmung mit den jeweiligen Meßergebnissen zeigen, wird in diesem Fall eine noch stärker vom Modellwert abweichende Laufzeit ermittelt. Diese stimmt jedoch tendenziell mit dem Meßergebnis

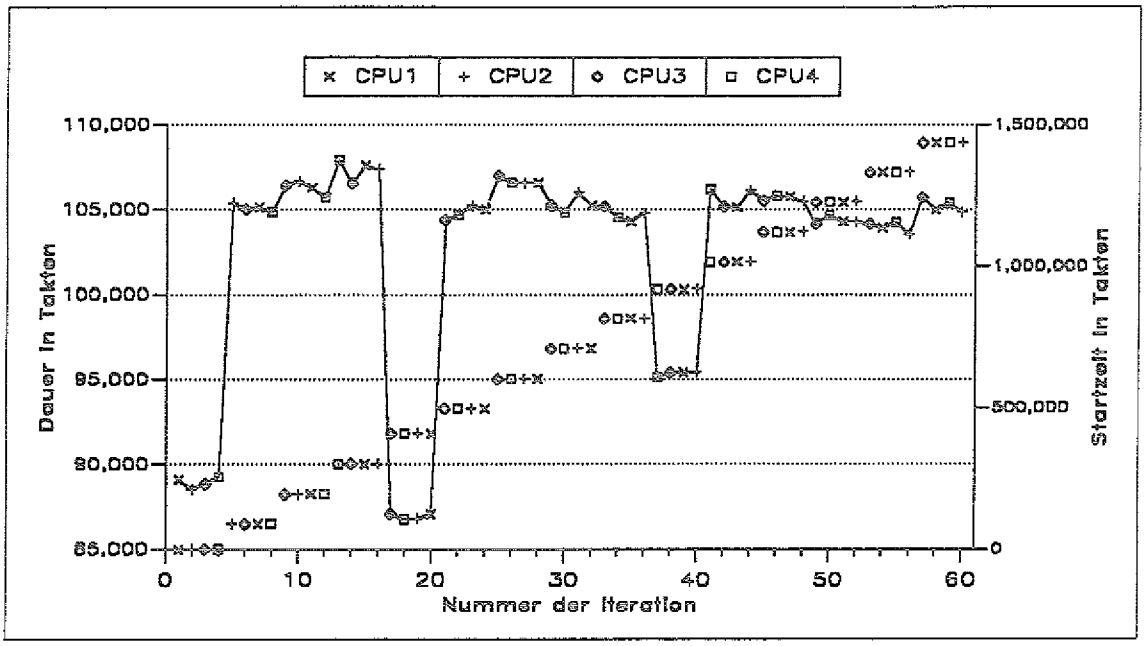


Abbildung 5.4: Meßergebnis bei 4 Prozessoren und Stride 4

überein. Betrachtet man die Ergebnisse unterschiedlicher Meßreihen für diese Messung, so bewegen sich die errechneten Abweichungen zwischen 47.05% und 52.04%. Das vorliegende Ergebnis kann also durchaus als repräsentativ angesehen werden. Auch für geringere Vektorlängen (bis 1024 Elemente) lassen sich diese Abweichungen feststellen. Messungen mit der Routine ICPUSED führen sogar zu Abweichungen bis 65%, so daß hier keine Betriebssystem-Einflüsse für diese Verzögerung verantwortlich gemacht werden können. Die in Abbildung 5.4 dargestellte Graphik der

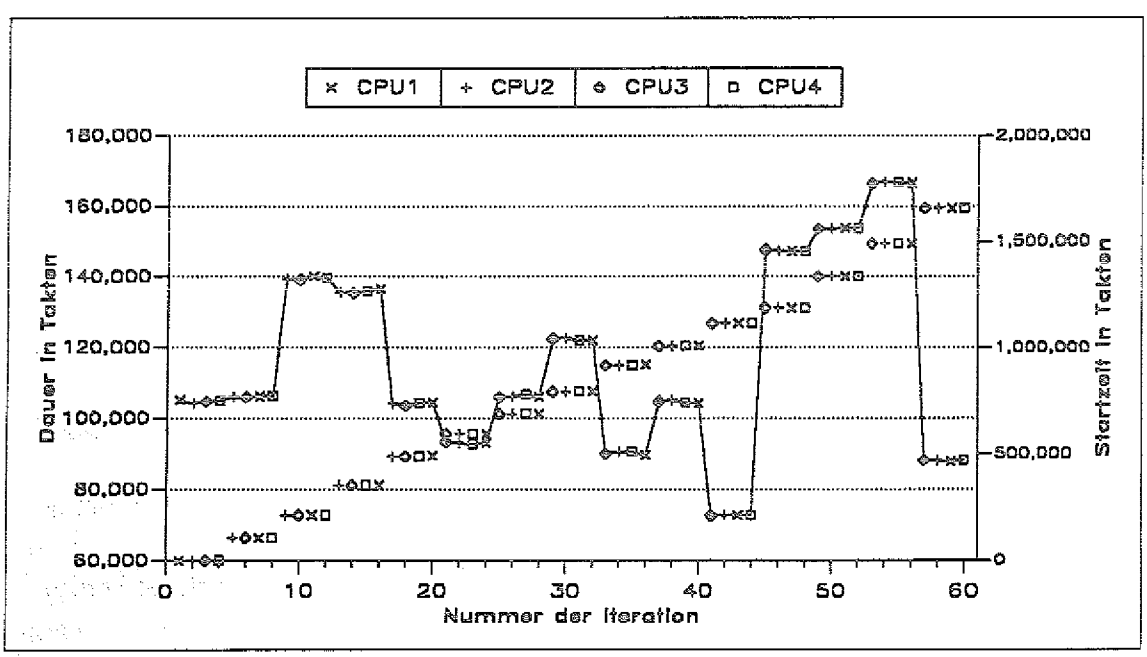


Abbildung 5.5: Meßergebnis mit ICPUSED bei 4 Prozessoren und Stride 4

betrachteten Messung zeigt ein sehr inhomogenes Verhalten. Die meisten Meßzeiten liegen unregelmäßig verteilt im Bereich zwischen 103.850 und 107.940 Takten. Daneben fallen drei Phasen auf, in denen jeweils vier parallel bearbeitete Iterationen deutlich kürzere Zeiten benötigen. Diese liegen in etwa bei 89.000, 87.000 und 95.500 Takten. Die Betrachtung von Ergebnissen aus anderen Meßreihen zeigt, daß das Ausführungsverhalten bei dieser Parameterkombination nicht einheitlich ist. Einige Messungen weisen nur Meßwerte zwischen 100.000 und 110.000 Takten auf und lassen keine ähnlichen Effekte erkennen. Bei anderen Messungen fällt der beschriebene Effekt weitaus deutlicher aus, wobei die einzelnen Zeiten extrem zwischen etwa 70.000 und 150.000 Takten schwanken. Abbildung 5.5 zeigt ein mit der Routine ICPUSED ermitteltes Ergebnis, das ebenfalls solchen Schwankungen unterworfen ist. Einflüsse, die diese unterschiedlichen Meßzeiten bewirken, können daher nicht durch das Betriebssystem verursacht sein und müssen im Zugriffsverhalten des Meßprogramms gesucht werden.

Stride 8

Die relative Bandbreite beim Zugriff mit Stride 8 beträgt nur noch $B_8 = \frac{64}{4*8} = 2$. Der Zugriff wird auf acht Bänke in 2 Sektionen eingeschränkt. Da jeder Prozessor mit 2 Ports gleichzeitig auf den Speicher zugreift, wird die verfügbare Kapazität bereits bei einem Prozessor voll ausgenutzt. Bei 2 Prozessoren kann im Durchschnitt jeweils noch ein Element pro Takt transferiert werden, doch sind hier nach den Erfahrungen der Messung 4/4 auch schon erhebliche Verzögerungen zu erwarten. Für

Prozessoren		1	2	3	4
Messung	Minimum	69951	69994	98024	176046
	Mittelwert	69951	80806	124171	185810
	Maximum	69962	89422	148271	199374
Analytisches Modell		69978	69978	98650	131418
Abweichung von Messung		0.03%	15.47%	25.87%	41.38%
Simulation	Minimum	69992	70045	115620	141872
	Mittelwert	69992	93984	138886	149211
	Maximum	69992	134759	162164	173781
Abweichung von Messung		0.05%	14.02%	10.59%	24.52%

Tabelle 5.6: Ausführungszeiten (in Takten) für Stride 8

größere Prozessoranzahlen ist jedoch wegen der mangelnden Bandbreite bereits mit Verzögerungen zu rechnen, wie die Angaben des Modells in Tabelle 5.6 deutlich zeigen. Für einen Prozessor treten wie erwartet noch keine Zugriffskonflikte auf. Für alle anderen Fälle zeigen sich jedoch größere Abweichungen von den Meßergebnissen, als durch das Modell angegeben. Dabei nimmt deren Ausmaß mit steigender Prozessoranzahl deutlich zu.

Die Messung mit 2 Prozessoren entspricht aus Sicht der einzelnen Prozessoren der Zugriffssituation der Messung 4/4. Die Abweichung der Laufzeit gegenüber dem Modell fällt jedoch im Mittel geringer aus, da lediglich 2 Prozessoren auf gemeinsame Speicherbänke zugreifen. Ihre Höhe bewegt sich je nach Meßreihe zwischen 8.66% und 19.84%. Tabelle 5.7 zeigt für unterschiedliche Vektorlängen die mit Hilfe entsprechender Meßreihen ermittelten Abweichungen zum Modellwert. Ihr

Vektorlänge	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
Meßreihe 1	19.48	13.75	10.18	20.20	20.07	18.42	12.57	14.26	5.19
Meßreihe 2	15.47	18.88	13.23	18.69	19.56	20.29	18.19	3.38	17.30

Tabelle 5.7: Prozentuale Abweichungen bei 2 Prozessoren und Stride 8 für verschiedene Vektorlängen

ist zu entnehmen, daß die Schwankungen offensichtlich nicht von der Vektorlänge abhängig sind. Betriebssystemeinflüsse als Ursache für die Störungen können analog zum zuletzt betrachteten Fall ausgeschlossen werden. Diese Aussage gilt auch für alle noch folgenden Untersuchungen dieses Strides.

Die entsprechende Meßgraphik 5.6 zeigt ein ähnliches Verhalten wie bei Messung 4/4. Die Meßwerte nehmen zumindest in der ersten Hälfte der Messung Werte von entweder ungefähr 70.000 oder 85.500 Takten an. Auffallend ist wie bei Messung 4/4 die häufige Übereinstimmung der Laufzeiten von parallel ausgeführten Iterationen. Abbildung 5.7 zeigt für die gleichen Parameter ein grundsätzlich unterschiedliches Ausführungsverhalten aus einer anderen Meßreihe. In fast allen Fällen

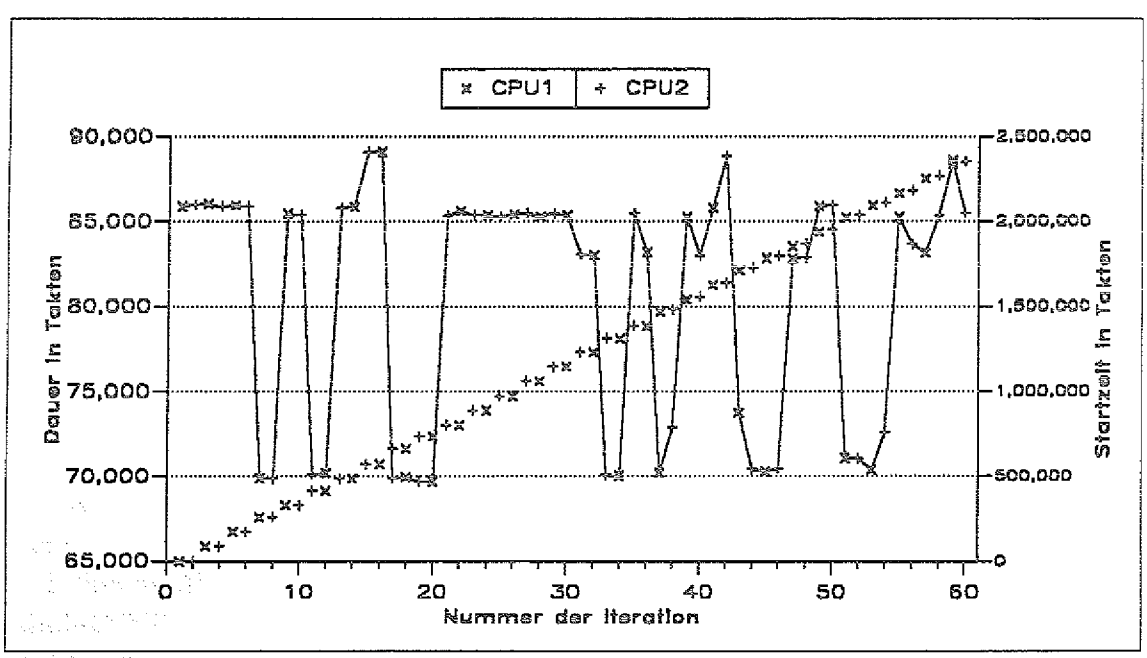


Abbildung 5.6: Meßergebnis bei 2 Prozessoren und Stride 8

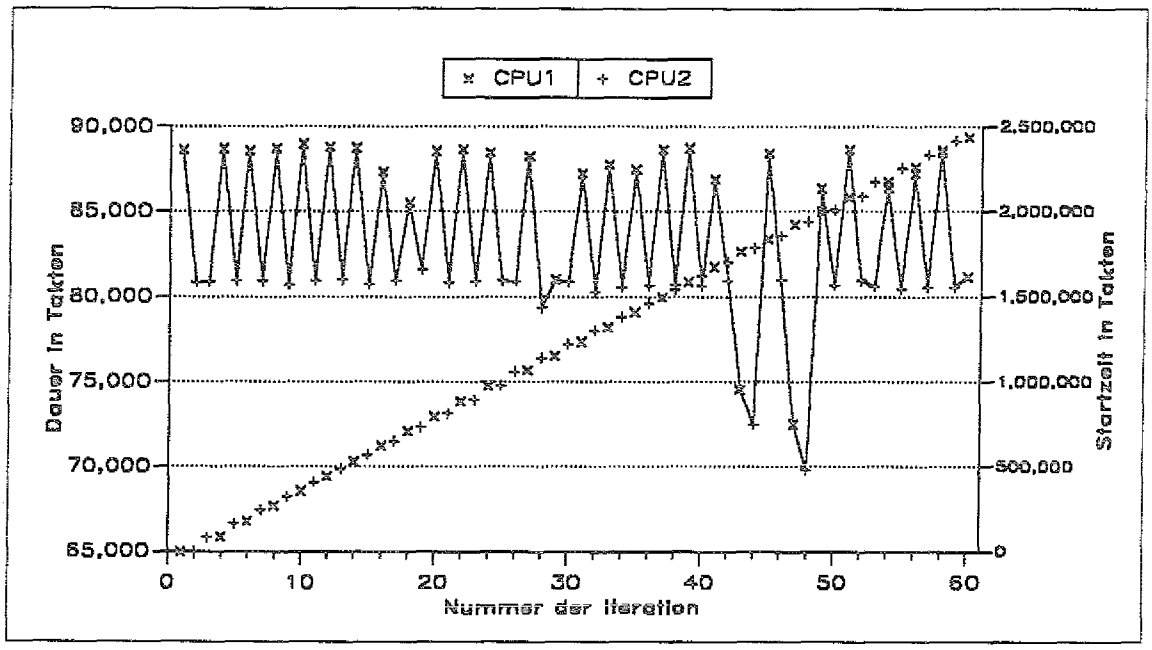


Abbildung 5.7: Meßergebnis bei 2 Prozessoren und Stride 8

bewegen sich die Ausführungszeiten je nach ausführendem Prozessor in engen, disjunkten Intervallen. In allen Fällen wird offensichtlich derselbe Prozessor bevorzugt behandelt. Wiederum fallen drei Abweichungen vom normalen Verhalten auf, von denen die beiden parallel bearbeiteten Iterationen etwa in gleicher Weise betroffen sind. Die Abweichung dieser Messung vom Modell liegt mit 19.48% ein wenig höher als die der anderen Messung.

Bei 3 Prozessoren ist die aus Tabelle 5.6 zu entnehmende Abweichung von etwa 26% als repräsentativ für alle Meßreihen anzusehen. Die Ergebnisse aus verschiedenen Meßreihen, auch die mit ICPUSED ermittelten, bewegen sich zwischen 25.17% und 25.99%. Für fallende Vektorlängen sinken die Abweichungen gleichmäßig bis zu etwa 17% bei Vektorlänge 2048, bei 1024 Elementen liegen sie um 9%, und darunter weichen sie nicht mehr von den prognostizierten Werten ab. Diese Beobachtungen führen zu der Annahme, daß sich bei längeren Zugriffen Situationen einstellen, in denen es zu einer regelmäßigen Blockierung durch sich wechselseitig beeinflussende Zugriffskonflikte kommt. Die Auswertung der Graphik in Abbildung 5.8 (Seite 82) läßt bezüglich der Ausführungszeiten ein regelmäßiges Muster erkennen. Die Zeiten verschiedener Iterationen bewegen sich bis auf wenige Ausnahmen in Abhängigkeit vom ausführenden Prozessor in drei disjunkten Intervallen. Diese liegen bei etwa 109.000, 124.000 und 145.000 Takten. Durch die unterschiedlichen Ausführungszeiten verschieben sich im Verlauf der Messung die Startzeitpunkte gegeneinander. Insgesamt sind nur drei Situationen mit in etwa gleichzeitig beginnenden Iterationen zu beobachten. Dieses Muster ist bei allen entsprechenden Messungen unterschiedlicher Meßreihen zu erkennen und muß seine Ursache ebenfalls im Zugriffsverhalten des Meßprogramms haben.

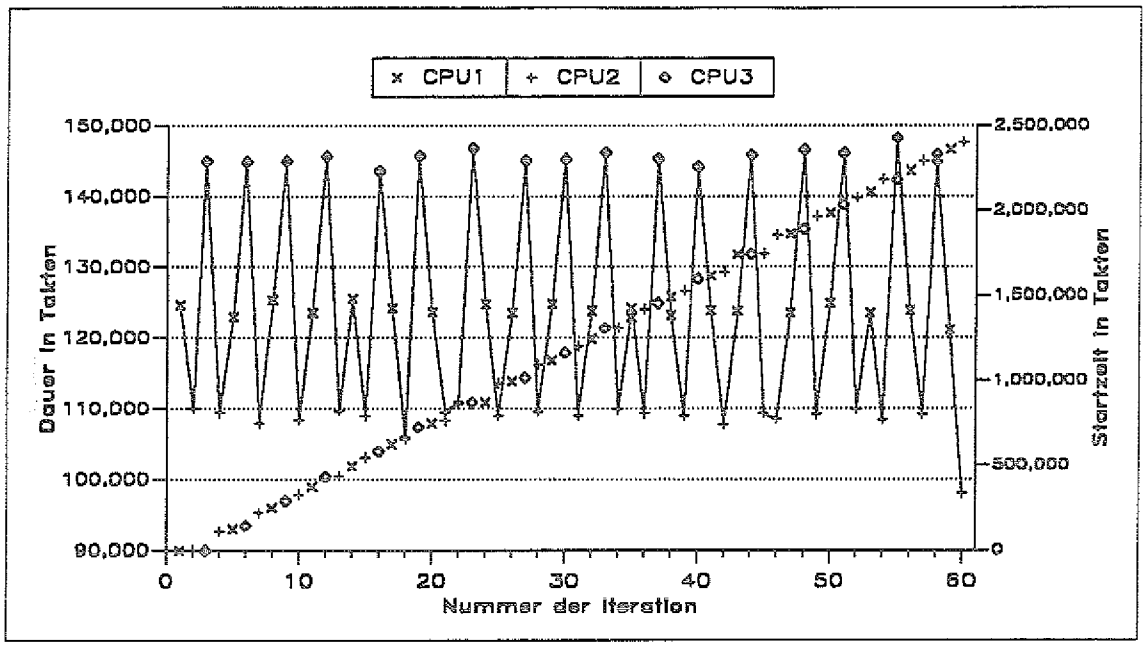


Abbildung 5.8: Meßergebnis bei 3 Prozessoren und Stride 8

Die höchsten Abweichungen zum Modell bei Stride 8 zeigt die Messung mit 4 Prozessoren. Dabei liegt sie jedoch prozentual gesehen noch unter der Abweichung der entsprechenden Messung bei Stride 4. Wie bereits für die Messung mit 3 Prozessoren schwanken die entsprechenden Mittelwerte anderer Meßreihen nur leicht. Dort liegen die Abweichungen im Bereich von 41.03% bis 41.58%. Für fallende Vektorlängen sinken sie nur bis etwa 38% bei Vektorlänge 9192. Danach fallen sie stärker ab. Im Gegensatz zu den bei 2/8 und 3/8 erkennbaren Mustern der Ausführungszeiten läßt

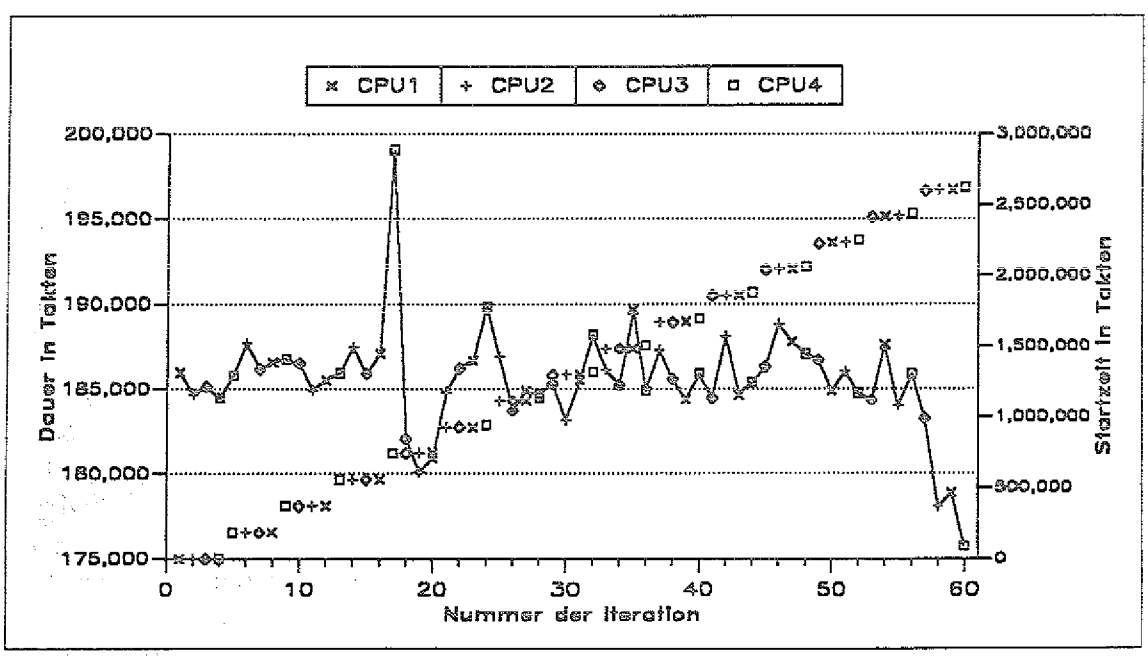


Abbildung 5.9: Meßergebnis bei 4 Prozessoren und Stride 8

die Messung 4/8 in Abbildung 5.9 einen unregelmäßigen Verlauf erkennen. Die meisten Meßwerte bewegen sich zwischen etwa 184.000 und 187.000 Takten, wobei jeweils vier Iterationen etwa gleiche Startzeiten aufweisen. Die überdurchschnittliche Verzögerung von Iteration 17 ist mit hoher Wahrscheinlichkeit auf eine Unterbrechung durch den Minor Clock Cycle zurückzuführen. Offensichtlich profitieren die drei parallel laufenden Iterationen von dieser Störung, da ihre Laufzeiten deutlich unter dem Durchschnitt liegen. Die dargestellte Messung zeigt sich – wie auch die zuletzt betrachtete – wiederum als repräsentativ für diese Parameterzusammenstellung.

Wie bereits bei Messung 4/4 fallen auch bei den Mehrprozessormessungen zu Stride 8 die deutlichen Abweichungen von der Prognose des analytischen Modells auf. Auch die Ergebnisse der Simulation zeigen große Abweichungen verglichen mit zuvor betrachteten Messungen niedrigerer Strides. Dabei liegen die simulierten Laufzeiten für die Parameterkombinationen 4/4, 2/8 und 3/8 bis etwa 14% über den gemessenen Werten, die von 4/8 jedoch mit etwa 25% deutlich darunter.

Stride 16

Bei Stride 16 erfolgt der Zugriff auf nur noch 4 Bänke, die sich alle in derselben Sektion befinden. Da ein Prozessor nur über einen Datenpfad zu jeder Sektion verfügt, kann der Zugriff der beiden Ports nicht mehr wie bisher gleichzeitig erfolgen. Nur das Port mit dem zuerst initiierten Zugriff kann diesen Pfad nutzen, während der Zugriff auf dem anderen Port zwar vorbereitet werden, jedoch erst nach Ende des ersten Vektorzugriffs beginnen kann. Die relative Bandbreite beträgt $B_{16} = \frac{64}{4*16} = 1$. Für die Mehrprozessormessungen sind also ausnahmslos Verzögerungen durch Bankkonflikte zu erwarten. Tabelle 5.8 bestätigt diese Erwartungen

Prozessoren		1	2	3	4
Messung	Minimum	69944	103925	131229	236240
	Mittelwert	69944	130735	195606	261743
	Maximum	69955	149561	262162	276345
Analytisches Modell		69945	131385	196921	262457
Abweichung von Messung		0%	0.49%	0.66%	0.27%
Simulation	Minimum	69587	74828	112932	261400
	Mittelwert	69913	130267	195296	262026
	Maximum	69943	259665	262084	262276
Abweichung von Messung		0.51%	0.35%	0.15%	0.10%

Tabelle 5.8: Ausführungszeiten (in Takten) für Stride 16

und vermittelt gleichzeitig die sehr gute Übereinstimmung von Modell, Simulation und Meßwerten. Für einen Prozessor treten keine Verzögerungen auf, da der Zugriff auf ein Element je Takt gesichert ist. Auffallend ist hier wie bei Stride 1 die um

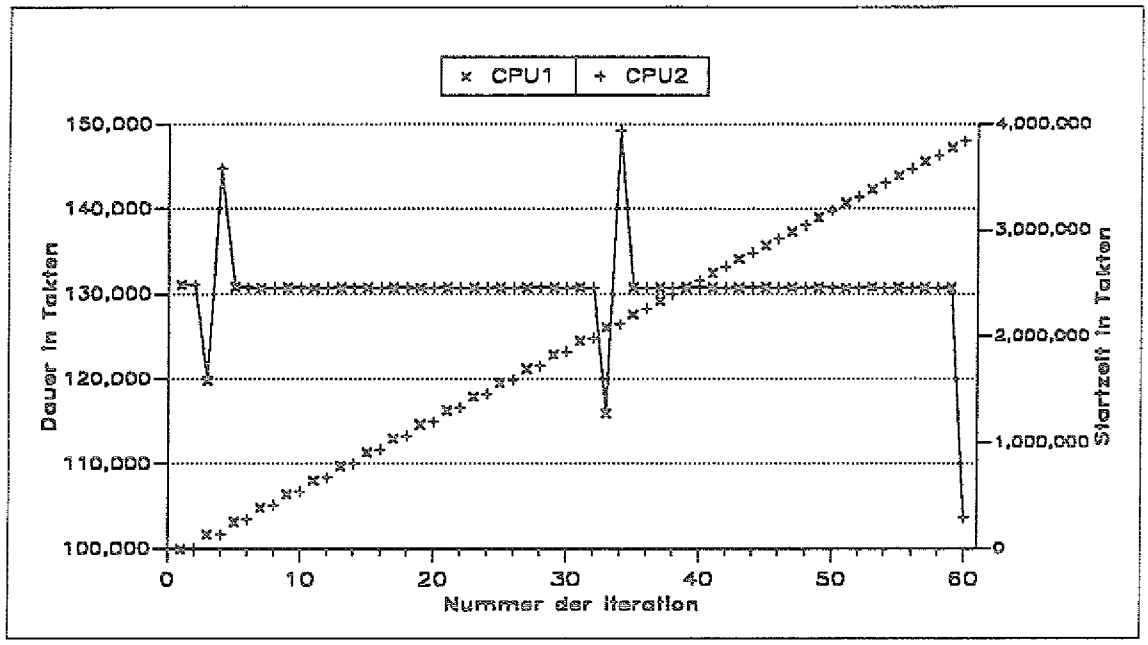


Abbildung 5.10: Meßergebnis mit 2 diagonalen Prozessoren bei Stride 16

7 Takte kürzere Meßzeit gegenüber den Einzelprozessormessungen für die Strides 2 bis 8. Die langsamere Ausführung bei diesen Messungen bleibt ungeklärt, da alle Zugriffe über 2 Ports erfolgen und die Pufferfunktion der Register Verzögerungen nach dem ersten Vektorzugriff verhindert. Dieser wiederum kann nicht verzögert werden, da er auf jeden Fall Priorität hat. Die Mittelwerte der Mehrprozessormessungen steigen wie erwartet linear mit der Prozessoranzahl an. In der Tabelle fallen auch die extremen Abweichungen von Minimal- und Maximalwert bei 3 Prozessoren auf.

Abbildung 5.10 zeigt für 2 Prozessoren, daß bis auf vier Ausnahmen alle Meßwerte in der erwarteten Höhe liegen. Jeweils zwei Abweichungen lassen sich unter dem Begriff *Doppelspitze* zusammenfassen, da von zwei parallel ausgeführten Iterationen im Vergleich zu den übrigen Laufzeiten eine verzögert und eine beschleunigt abgearbeitet wird. In beiden Situationen fällt die Verzögerung vom Betrag her mit etwa 3000 bzw. 3650 Takten höher aus als die Beschleunigung. Die Startzeiten der Iterationen verschieben sich durch die gegensätzlichen Auswirkungen von Verzögerung und Beschleunigung, so daß im weiteren Verlauf der Messung die Bearbeitung der Iterationen nunmehr verschränkt erfolgt. Bei einer Messung mit höherer Iterationsanzahl bestätigt sich das periodische Verhalten. In einigen Fällen ist die Abfolge der beiden Effekte vertauscht, in jedem Fall sind aber benachbarte Iterationen betroffen. Das Auftreten der Verzögerung ist dabei immer an denselben Prozessor gebunden. Die Verzögerungen lassen sich anhand der Feststellung ihres zeitlichen Abstandes auf den Einfluß des Minor Clock Cycle zurückführen, der die Bearbeitung einer Iteration auf einem Prozessor unterbricht. Wie bereits bei der Messung 4/8 profitiert der andere Prozessor von dieser Unterbrechung, da er währenddessen ungestört auf den Speicher zugreifen kann.

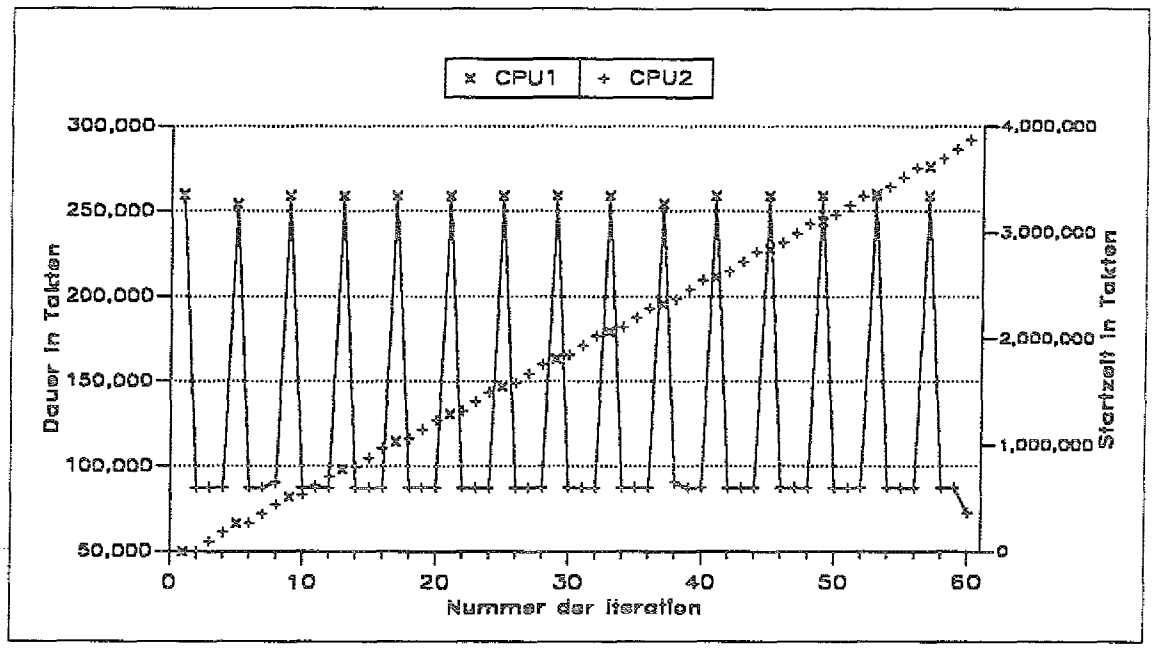


Abbildung 5.11: Meßergebnis mit 2 benachbarten Prozessoren bei Stride 16

Abbildung 5.11 präsentiert das Resultat einer Messung mit gleichen Parametern und unter gleichen Bedingungen wie im letzten Fall. Deutlich zu erkennen ist der Unterschied bei der Ausführung der Iterationen, der auch an die unterschiedlichen Messungen für 2 Prozessoren bei Stride 8 erinnert. Der Graphik ist zu entnehmen, daß je nach Prozessor die Dauer einer Iteration stark unterschiedlich ausfällt. Iterationen auf Prozessor 2 werden etwa dreimal so schnell ausgeführt wie jene auf Prozessor 1. So können während der Bearbeitung einer Iteration durch Prozessor 1 drei Iterationen auf Prozessor 2 ausgeführt werden. Der in diesem Fall berechnete Mittelwert ergibt sich zu 130.003 Takten, ist also nahezu identisch mit der ersten Messung. Die Graphik weist 2 Unregelmäßigkeiten auf: Die Iterationen 8 und 38 werden verzögert, dafür können die Iterationen 5 und 37 relativ zu den übrigen Laufzeiten auf Prozessor 1 schneller ausgeführt werden. Dieser Effekt läßt sich aufgrund des zeitlichen Abstands, in dem die Unregelmäßigkeiten auftreten, wiederum mit hoher Wahrscheinlichkeit auf den Interrupt Minor Clock Cycle zurückführen.

Der Unterschied der beiden Messungen läßt sich durch die Verteilung der Prozessoren in Verbindung mit den Interprozessorprioritäten erklären. Das Prioritätenverfahren legt fest, welcher Prozessor bei einem gleichzeitigen Zugriff auf eine Speicherbank den angeforderten Zugriff durchführen kann. Es ordnet dazu den 4 Prozessoren des Systems eine zyklische Reihenfolge zu (siehe Abschnitt 1.1). Bei der Bearbeitung eines Multitasking-Programms mit 2 Prozessoren können bezüglich dieser Reihenfolge zwei grundsätzlich verschiedene Situationen auftreten:

- zwei *benachbarte* Prozessoren führen die Bearbeitung durch;
- zwei *nicht* benachbarte (diagonale) Prozessoren führen die Bearbeitung durch.

Im folgenden soll das auf diese Weise entstehende Zugriffsverhalten unter der jeweiligen Situation analysiert werden. Tabelle 5.9 zeigt den Zugriff von benachbarten Prozessoren. Dargestellt ist die durch das Prioritätenschema entstehende Zugriffssituation ohne Beachtung der Einschwingphase. Jeweils nach 4 Takten wechseln die Prioritäten der Prozessoren, dargestellt in den mit P bezeichneten Spalten. Dabei sind niedrigeren Zahlen jeweils höhere Prioritäten zugeordnet. Da in jedem Takt

Takt	P	0	1	2	3	P	4	5	6	7	P	8	9	10	11	P	12	13	14	15	P	16	17	
CPU 1	1	1	2	3	4	4	S	B	B	B	3	5	6	7	8	2	9	10	11	12	1	13	14	
CPU 2	2	S	B	B	B	1	1	2	3	4	4	S	B	B	B	3	S	B	B	B	2	S	B	...
CPU 3	3	-	-	-	-	2	-	-	-	-	1	-	-	-	-	4	-	-	-	-	3	-		
CPU 4	4	-	-	-	-	3	-	-	-	-	2	-	-	-	-	1	-	-	-	-	4	-		

Tabelle 5.9: Zugriff mit zwei benachbarten Prozessoren bei Stride 16

ein Zugriffsversuch jedes Prozessors auf die Sektion stattfindet, erleidet der Prozessor mit der jeweils niedrigeren Priorität einen Sektionskonflikt (S) und danach drei Bankkonflikte (B). Der Tabelle ist zu entnehmen, daß zyklisch gesehen ein Prozessor für drei aufeinanderfolgende Prioritäteneinstellungen im direkten Vergleich zum zweiten Prozessor von seiner höheren Priorität profitiert. Daher kann er auch im Mittel dreimal so oft zugreifen wie der andere Prozessor und ist dementsprechend um den Faktor 3 schneller, da für Strides ab 16 die mittlere Dauer eines Zugriffs proportional zur Ausführungszeit der Schleife ist. Demgegenüber ist aus Tabelle 5.10 für die diagonale Verteilung der Prozessoren zu entnehmen, daß jeder Prozessor jeweils 2 aufeinanderfolgende Prioritäteneinstellungen über den anderen dominiert. Im Mittel können also beide Prozessoren gleich oft auf den Speicher zugreifen, daher

Takt	P	0	1	2	3	P	4	5	6	7	P	8	9	10	11	P	12	13	14	15	P	16	
CPU 1	1	1	2	3	4	4	S	B	B	B	3	S	B	B	B	2	9	10	11	12	1	13	
CPU 2	2	-	-	-	-	1	-	-	-	-	4	-	-	-	-	3	-	-	-	-	2	-	
CPU 3	3	S	B	B	B	2	1	2	3	4	1	5	6	7	8	4	S	B	B	B	3	S	...
CPU 4	4	-	-	-	-	3	-	-	-	-	2	-	-	-	-	1	-	-	-	-	4	-	

Tabelle 5.10: Zugriff mit zwei diagonalen Prozessoren bei Stride 16

stimmen unter diesen Umständen die Laufzeiten beider Prozessoren überein. Beide Situationen können mit Hilfe der Simulation sehr genau nachgebildet werden.

Abbildung 5.12 zeigt eine weitere Messung mit 2 Prozessoren bei Stride 16. Zu erkennen sind für die ersten 22 Iterationen identische Ausführungszeiten, die auf eine Ausführung des Meßprogramms mit diagonalen Prozessoren hindeuten. Nach einem kurzen Übergang erkennt man das Muster für die Ausführung mit benachbarten Prozessoren. Offensichtlich hat hier während der Ausführung des Meßprogramms ein Prozessorwechsel stattgefunden. In den Iterationen 50 und 53 sind Abweichungen

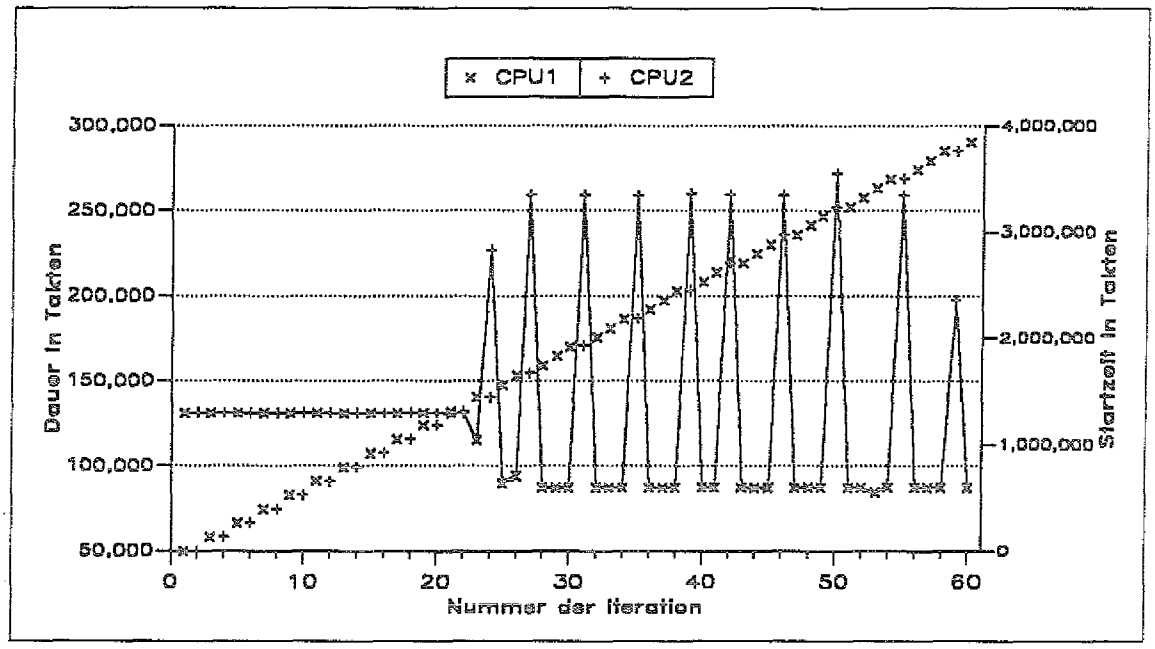


Abbildung 5.12: Prozessorwechsel während der Bearbeitung mit 2 Prozessoren

zu erkennen, die wahrscheinlich durch den Minor Clock Cycle verursacht werden. Der Interrupt wird dabei auf Prozessor 2 ausgeführt. Berechnet man ausgehend vom Startzeitpunkt der Iteration 53 das vorherige Auftreten des Minor Clock Cycle, so muß dieser in den Iterationen 23 und 24 aufgetreten sein. Während dieser Iterationen findet auch der Wechsel des Ausführungsverhaltens statt. Daher ist anzunehmen, daß hier der Minor Clock Cycle ursächlich für den Prozessorwechsel verantwortlich zu machen ist. Dieses Beispiel zeigt, daß zwar ein Wechsel des Prozessors während der Bearbeitung einer Iteration durchaus möglich ist, daß dieser jedoch von Störungen begleitet wird und sich sichtbar auf das Ausführungsverhalten auswirkt. Daher können entsprechende Situationen identifiziert werden. Im Verlauf der Messungen wurden alle Ergebnisse auf ähnliche Störungen untersucht. Zusammenfassend kann die Aussage getroffen werden, daß solche Prozessorwechsel eher eine Ausnahme sind und höchst selten auftreten. Daher ist eine grundsätzliche Beeinflussung der Meßergebnisse durch derartige Einflüsse auszuschließen.

Abbildung 5.13 (Seite 88) zeigt die Graphik zur Messung mit 3 Prozessoren. Grundsätzlich ist zu erkennen, daß eine CPU etwa doppelt so schnell ihre Iterationen bearbeitet wie die beiden anderen. Auf die Ausführung einer Iteration der Prozessoren 1 und 3 kommen also zwei Iterationen für Prozessor 2. Das so entstehende Muster zeigt sich bis auf drei Ausnahmen sehr regelmäßig. Die Ausführungszeiten der Iterationen 2 und 3 sind nahezu identisch. Iteration 25 wird genauso wie Iteration 53 verzögert. Der Abstand der beiden letztgenannten Verzögerungen läßt wiederum auf den Minor Clock Cycle als Ursache für die Störung schließen. Wie bereits in zuvor betrachteten Situationen profitieren die beiden anderen Prozessoren von dieser Verzögerung (Iterationen 22 und 24 bzw. 52 und 54). Dabei liegt die Beschleunigung der Bearbeitung bei CPU 3 deutlich höher. Mit Hilfe der Betrach-

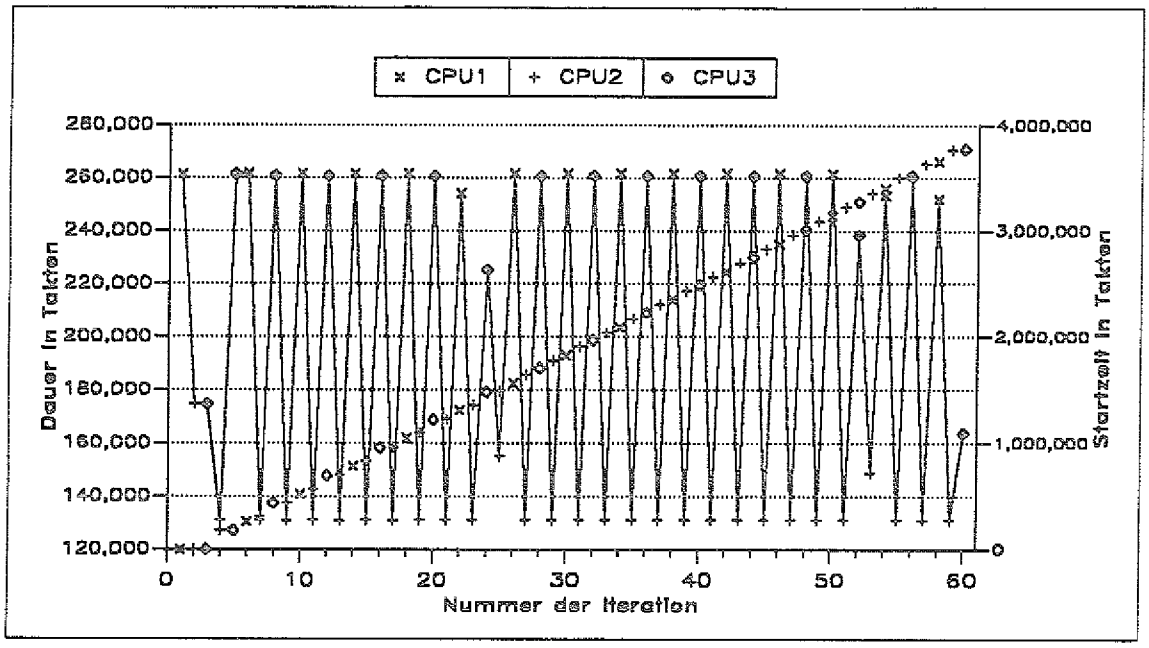


Abbildung 5.13: Meßergebnis bei 3 Prozessoren und Stride 16

tungen im letzten Abschnitt kann daher geschlossen werden, daß die Prozessoren 1 und 3 benachbart liegen und Prozessor 3 im Prioritätenschema vor Prozessor 1 eingeordnet ist. Lägen die Prozessoren nicht benachbart, so müßten sie etwa im gleichen Maß von der Verzögerung der CPU 2 profitieren. Tabelle 5.11 beschreibt das Zugriffsverhalten. Durch das Fehlen eines Prozessors bleibt die relative Priorität zwischen den übrigen Prozessoren in einem Fall für zwei Prioritätenverteilungen von jeweils 4 Takten konstant. Ein Prozessor kann demnach auf doppelt soviele Elemente wie die anderen Prozessoren zugreifen und ist daher um den Faktor zwei mit der Ausführung schneller.

Takt	P	0	1	2	3	P	4	5	6	7	P	8	9	10	11	P	12	13	14	15	P	16	
CPU 1	1	1	2	3	4	4	S	B	B	B	3	S	B	B	B	2	5	6	7	8	1	5	
CPU 2	2	S	B	B	B	1	1	2	3	4	4	S	B	B	B	3	S	B	B	B	2	S	...
CPU 3	3	S	B	B	B	2	S	B	B	B	1	1	2	3	4	4	S	B	B	B	3	S	
CPU 4	4	-	-	-	-	3	-	-	-	-	2	-	-	-	-	1	-	-	-	-	4	-	

Tabelle 5.11: Zugriff mit drei Prozessoren bei Stride 16

Der Graphik für 4 Prozessoren in Abbildung 5.14 ist ein ähnliches Verhalten wie der Messung mit 2 diagonalen Prozessoren (siehe Abbildung 5.10) zu entnehmen. Da alle 4 Prozessoren auf den Speicher zugreifen, werden sie im Mittel durch das Prioritätenverfahren gleichberechtigt behandelt. Wiederum sind Doppelspitzen, verursacht durch den Minor Clock Cycle, zu erkennen. Das Verhalten der parallel zu der jeweils verzögerten Iteration ausgeführten Iterationen unterliegt den Bedingun-

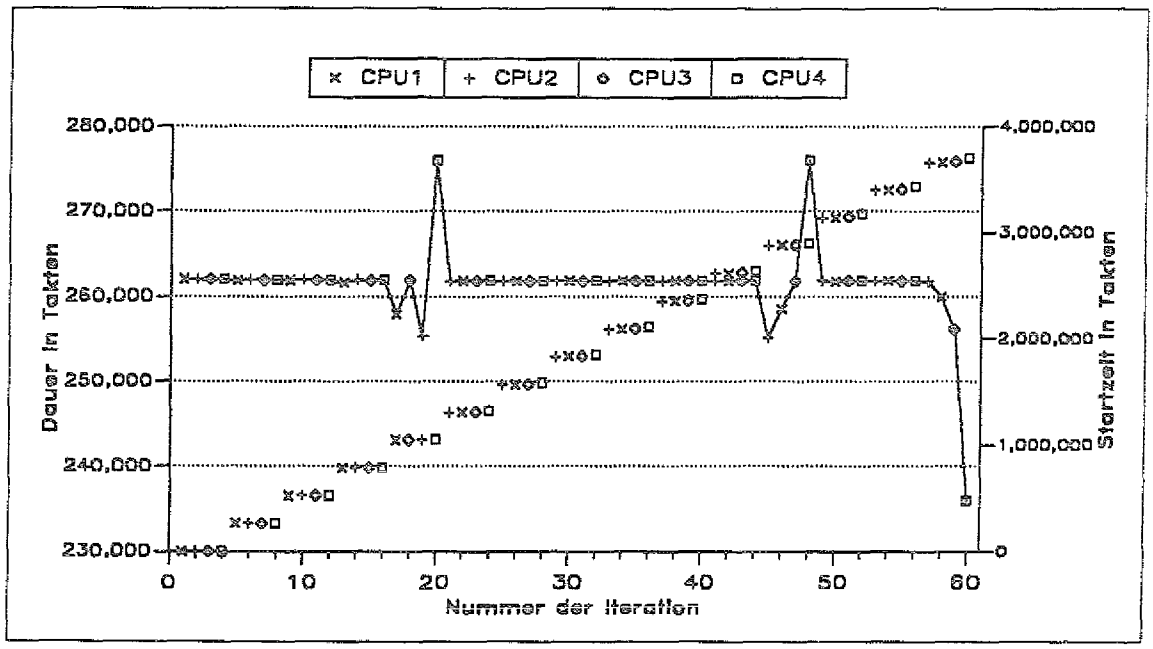


Abbildung 5.14: Meßergebnis bei 4 Prozessoren und Stride 16

gen beim Zugriff von 3 Prozessoren. Dementsprechend profitiert vor allem eine CPU von der Unterbrechung.

Strides 32 und 64

Da bereits bei Stride 16 der Zugriff auf eine Sektion eingeschränkt wird und dort für mehrere Prozessoren kein gleichzeitiger Zugriff auf den Speicher möglich ist, sind auch für die Strides 32 und 64 keine grundsätzlich neuen Effekte zu erwarten. In beiden Fällen wird jeweils die Bandbreite zum vorangegangenen Stride halbiert. Bei Stride 32 werden nur noch 2 Bänke des Systems referiert, bei einer Bankzykluszeit von 4 Takten kann im Schnitt also nur jeden zweiten Takt ein Element aus dem

Prozessoren		1	2	3	4
Messung	Minimum	131340	160515	262265	471718
	Mittelwert	131402	261513	391788	523097
	Maximum	132281	522485	524266	538521
Analytisches Modell		131341	262413	393485	524557
Abweichung von Messung		0%	0.34%	0.43%	0.27%
Simulation	Minimum	130983	174643	259102	523606
	Mittelwert	131324	261526	392678	524005
	Maximum	131339	521773	524182	524374
Abweichung von Messung		0.27%	0%	0.22%	0.17%

Tabelle 5.12: Ausführungszeiten (in Takten) für Stride 32

Speicher geladen werden. Daher wird bereits der Zugriff einer CPU behindert. Da die Ausführungszeit ab Stride 16 proportional zu der mittleren Dauer eines Zugriffs ist, benötigt der Prozessor in diesem Fall vergleichsweise etwa doppelt so lange wie bei Stride 16. Die mittleren Ausführungszeiten für die Mehrprozessormessungen multiplizieren sich entsprechend mit der Anzahl der eingesetzten Prozessoren. Diese Aussagen lassen sich entsprechend auf Stride 64 übertragen, bei dem nur noch eine Bank des Systems referiert wird. Die Tabellen 5.12 und 5.13 bestätigen die Prognosen und zeigen die weitgehende Übereinstimmung der Ergebnisse aller drei Meßmethoden. Die graphische Auswertung der einzelnen Messungen lassen die für Stride 16 vorgestellten Ausführungsmuster mit größeren Laufzeiten der einzelnen Iterationen erkennen. Auch der Zeitbedarf für eine Messung vergrößert sich, so daß sich der periodische Einfluß des Minor Clock Cycle erhöht.

Prozessoren		1	2	3	4
Messung	Minimum	262405	380436	427364	950160
	Mittelwert	262480	522031	781622	1045444
	Maximum	263108	573725	1049020	1066955
Analytisches Modell		262406	524550	786694	1048838
Abweichung von Messung		0%	0.48%	0.64%	0.32%
Simulation	Minimum	262047	346903	523063	1047886
	Mittelwert	262343	523442	785911	1048288
	Maximum	262403	1046070	1048463	1048654
Abweichung von Messung		0.13%	0.26%	0.54%	0.27%

Tabelle 5.13: Ausführungszeiten (in Takten) für Stride 64

Zusammenfassung

Im Überblick zeigt die Auswertung gute Übereinstimmung der Ergebnisse aller drei Untersuchungsmethoden. Betrachtet man die in den Tabellen zusammengefaßten Werte, so erkennt man mit Ausnahme der Parameterkonfigurationen 4/4, 2/8, 3/8 und 4/8 kaum Abweichungen zwischen Messungen und Modell bzw. Simulation. Im Verlauf der Auswertung konnten mit Hilfe analytischer Methoden bereits einige Erklärungsversuche für verschiedene Effekte bereitgestellt werden. Andere Effekte, insbesondere in den zuvor benannten kritischen Fällen, haben sich als zu komplex und nicht unmittelbar einsichtig erwiesen. Im folgenden sollen nun mit Hilfe der Simulation einerseits die analytischen Überlegungen verifiziert und andererseits Interpretationen für vorerst ungeklärte Effekte gefunden werden.

5.4 Diskussion der Simulationsergebnisse

Im Verlauf der Auswertung konnte die Simulation bereits an der guten Übereinstimmung der Ergebnisse bezüglich der Meßergebnisse bestätigt werden. Die dabei

auftretenden Abweichungen zwischen den beiden Untersuchungsmethoden lagen bis auf vier Ausnahmen unter 1%. Die Simulation liefert über die Laufzeit für eine bestimmte Kombination von Parametern hinaus zusätzliche Informationen über den Verlauf der einzelnen Zugriffe und stellt Statistiken über die Häufigkeit einzelner Konfliktypen in Abhängigkeit von den betroffenen Prozessoren zur Verfügung. Mit Hilfe dieser Informationen können die in der Auswertung vorgenommenen analytischen Überlegungen für eine Vielzahl von Effekten verifiziert werden. Die Analyse der von den Messungen abweichenden Situationen liefern darüber hinaus Hinweise auf ein Verhalten der Hardware, das nach Kenntnis der veröffentlichten Unterlagen nicht zu erwarten war.

Da das Simulationsprogramm zeitgesteuert ist und pro Simulationstakt eine Vielzahl von Vergleichsoperationen durchgeführt werden muß, zeigt sich die Simulation für Verwendung derselben Parameter wie in der Messung als sehr zeitintensiv. Ihr Zeitbedarf steigt linear mit der gemessenen Laufzeit der behandelten Parameterkombination. Dies wirkt sich insbesondere für die Simulationen aus, bei denen der Parameter Stride 16 oder größer gewählt wird. Die Zeiten für die Ausführung der Simulation einer einzelnen Messung auf der CRAY X-MP schwanken von etwa 200 CPU-Sekunden im besten Fall (4 Prozessoren, Stride 1) bis 2600 CPU-Sekunden im ungünstigsten Fall (4 Prozessoren, Stride 64). In die Simulation gehen einige Overheadzeiten als Zufallsgrößen ein (vergleiche Kapitel 4). Um ihren Einfluß zu bestimmen, müssen je Parameterkombination mehrere Simulationsläufe durchgeführt werden. Zur Begrenzung der Rechenzeit wurden dabei Iterationsanzahl bzw. Vektorlänge verkürzt. Dabei zeigte sich der Simulator unempfindlich gegenüber einer Verkürzung der Vektorlänge und weitgehend auch in bezug auf eine Veränderung der Iterationsanzahl. Hier sind jedoch in Einzelfällen bei den Strides 4 und 8 Ausnahmen zu vermerken, auf die im Laufe der weiteren Bearbeitung noch eingegangen wird.

Beispielhaft sollen im folgenden Simulation und Messung zu 3 Prozessoren und Stride 16 gegenübergestellt werden. Abbildung 5.15 (Seite 92) zeigt die graphische Darstellung der jeweils ermittelten Laufzeiten. Messung wie auch Simulation zeigen – wie in der Auswertung auf Seite 88 bereits beschrieben – die bevorzugte Behandlung eines Prozessors bei der Bearbeitung. Der Trace-Datei der Simulation ist ein Zugriffsverhalten analog zu Tabelle 5.11 (Kapitel 5) zu entnehmen. Prozessor 1 wird aufgrund des zyklischen Prioritätenverfahrens bevorzugt behandelt und kann doppelt so oft zugreifen wie die beiden anderen Prozessoren. Daher benötigen die auf ihm ausgeführten Iterationen nur etwa die Hälfte der Laufzeit der übrigen Iterationen. Die Verzögerungen der Iterationen 25 und 53 in der Messung werden auf den Einfluß des Interrupts Minor Clock Cycle zurückgeführt. Die in der Simulation erkennbaren Störungen der Iterationen 22 und 51 beruhen auf der Berücksichtigung des Minor Clock Cycle und zeigen ein mit der Messung vergleichbares Verhalten. Neben den analytischen Betrachtungen ist dies ein weiterer Beleg für den kausalen Zusammenhang zwischen dem Interrupt und den periodischen Störungen.

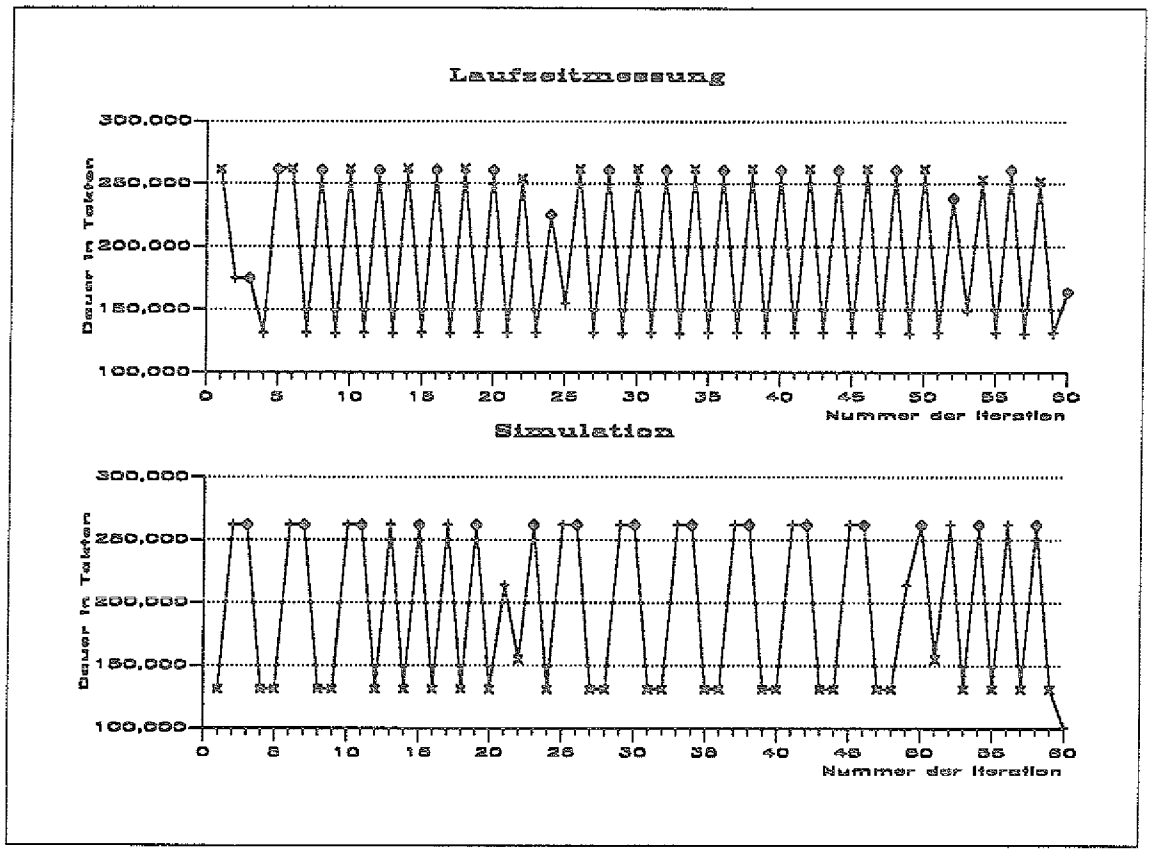


Abbildung 5.15: Gegenüberstellung von Messung und Simulation bei 3 Prozessoren und Stride 16

In Tabelle 5.14 sind die für die einzelnen Prozessoren ermittelten Laufzeiten und Konfliktanzahlen gegenübergestellt. Bei Stride 16 finden alle Zugriffe nur noch auf eine Sektion statt. Die Ports eines Prozessors können unter diesen Umständen nicht mehr parallel arbeiten, so daß keine Portblockierungen auftreten können. Die Tabelle zeigt, daß sowohl Laufzeiten wie auch Konfliktanzahlen der Prozessoren 2 und 3 kaum voneinander abweichen. Gegenüber Prozessor 1 zeigen die Simultan- und Bankkonflikte etwa eine Verdreifachung, während sich die Sektionskonflikte etwas

	Prozessor 1	Prozessor 2		Prozessor 3
		ohne MCC	mit MCC	
Laufzeit	131088 ± 5	262094 ± 12	214096 ± 16	262136 ± 8
Simultankonflikte	16382 ± 1	49134 ± 4	37135 ± 4	49145 ± 2
Bankkonflikte	49151 ± 3	147406 ± 10	111407 ± 12	147437 ± 6
Sektionskonflikte	100780 ± 4	231660 ± 16	183660 ± 1	231716 ± 8

Tabelle 5.14: Mit der Simulation ermittelte Laufzeiten (in Takten) und Konfliktanzahlen bei 3 Prozessoren und Stride 16 mit Berücksichtigung des Minor Clock Cycle (MCC)

mehr als verdoppeln. An diesem Beispiel wird deutlich, daß eine Steigerung der Konfliktanzahl sich nicht unbedingt in gleichem Maße auf die Laufzeit auswirken muß. Eine durch den Interrupt Minor Clock Cycle auf diesem Prozessor verursachte verlängerte Laufzeit hat keinen Einfluß auf die Konfliktanzahlen und ist daher in der Tabelle nicht berücksichtigt. Das Prioritätenverfahren bewirkt, daß ausschließlich Prozessor 2 von dem Aussetzen des Prozessors 1 profitiert. Die in diesen Fällen deutlich reduzierten Konflikte machen sich in einer verkürzten Ausführungszeit bemerkbar.

Die auf diese Weise verglichenen Messungen mit Strides der Größe 16 und höher zeigen alle ähnlich gute Übereinstimmung sowohl bezogen auf das Ausführungsverhalten wie auch auf die Einwirkung des Minor Clock Cycle. Insbesondere können auch die Doppelspitzen, die bei Messungen mit 2 diagonalen Prozessoren bzw. 4 Prozessoren auftreten (siehe Seiten 84, 89), anhand der Simulation eindeutig auf den Interrupt zurückgeführt werden. Für die Simulation mit 2 Prozessoren sind jeweils deutlich die unterschiedlichen Zugriffsmuster je nach Anordnung der Prozessoren zu erkennen. In allen Fällen können die Überlegungen der Auswertung anhand der entsprechenden Trace-Dateien verifiziert werden.

Neben diesen Effekten, die sich vor allem in regelmäßigen Zugriffs- und Ausführungsmustern bemerkbar machen, können auch einige, nur wenige Takte betreffende Störungen mit Hilfe der Simulation erklärt werden. Bei der Auswertung der Messungen mit niedrigem Stride sind in den Meßergebnissen sporadische Abweichungen von wenigen Takten festzustellen, die auf einzelne Sektionskonflikte während der ersten Vektorladeoperationen zurückgeführt werden (siehe Seite 75). Anhand der Trace-Dateien kann für einige Fälle festgestellt werden, daß tatsächlich einzelne Bank- und Sektionskonflikte während der ersten Vektorladeoperation stattfinden und für diese Verzögerungen verantwortlich sind. Die nicht einheitlich reproduzierbaren Konfliktsituationen hängen jedoch offenbar auch von den zufällig gewählten Parametern der Simulation ab.

Im Bereich der Strides 1 – 4 werden, abgesehen von der ersten Iteration für die Einzelprozessormessungen, keine Abweichungen festgestellt. Dieses Verhalten wird erwartungsgemäß durch die Simulation bestätigt. In jeder Messung dieses Bereichs mit mehreren Prozessoren treten Abweichungen einzelner Iterationen unterschiedlichen Umfangs bis zu maximal 0.5% des jeweiligen Minimalwertes auf. Ähnliche Unregelmäßigkeiten können, wenn auch meist nur in geringerem Umfang, auch in der Simulation beobachtet werden. Um die feineren Effekte auch mittels der Simulation zuverlässig erfassen zu können, müßten weitere Einzelheiten des Systems berücksichtigt werden, beispielsweise eventuell stattfindende Ladeoperationen der Instruktionspuffer, die bei der CRAY X-MP zwei Prozessoren beeinflussen (siehe Abschnitt 1.1). In diesem Fall wäre auch eine genauere Untersuchung des Einflusses der durch Zufallsgrößen bestimmten Overheadzeiten erforderlich. Eine systematische Untersuchung dieser Einflüsse ist wegen des damit verbundenen Aufwands in dieser Arbeit nicht durchführbar.

Die Ergebnisse der Simulation weichen lediglich in 4 von 28 Fällen stärker von denen der Laufzeitmessung ab. Dabei handelt es sich um die Parameterkombinationen 4/4, 2/8 und 3/8, in denen die Simulation im Mittel bis zu 14% längere Ausführungszeiten aufzeigt, sowie die Kombination 4/8, bei der das Simulationsergebnis um etwa 25% unter der mittleren Laufzeit bleibt. Tabelle 5.15 stellt für diese kritischen Fälle die jeweiligen Mittelwerte und ihre Abweichungen zum Modell gegenüber. Dabei ist zu erkennen, daß die Laufzeiten auch erheblich von den Vorstellungen des Modells abweichen. An dieser Stelle sei noch einmal darauf hingewiesen, daß bis auf die betrachteten Ausnahmen alle Methoden sehr gute Übereinstimmungen zeigen. Die Beobachtung deutet darauf hin, daß bei der Modellierung gewisse Aspekte der Architektur, die sich nur bei extremen Zugriffssituationen bemerkbar machen, nicht berücksichtigt werden. Das analytische Modell leidet an dieser Stelle sicherlich unter der Vernachlässigung der Portblockierungen und unregelmäßigen Simultankonflikte (vergleiche Kapitel 3.3.3). In der Simulation werden jedoch alle über die Komponenten des Systems verfügbaren Informationen berücksichtigt. An diesem Punkt liegt die Vermutung nahe, daß über die publizierten Informationen hinaus noch unbekannte Eigenschaften Einfluß auf das Zugriffsverhalten nehmen.

Parameterkombination		4/4	2/8	3/8	4/8
Messung	Mittelwert	102733	80806	124171	185810
	Abweichung vom Modell	46.87%	15.47%	25.87%	41.38%
Simulation	Mittelwert	110670	93984	138886	149211
	Abweichung vom Modell	58.22%	34.31%	40.78%	13.56%
Abweichung Simulation – Messung		7.17%	14.02%	10.59%	-24.52%

Tabelle 5.15: Gegenüberstellung der kritischen Fälle

Stellt man die Ergebnisse von Laufzeitmessung und Simulation für die kritischen Parameterkombinationen graphisch gegenüber, so ist zumindest für 2 Prozessoren ein sich ähnelndes Ausführungsverhalten zu erkennen. Daher ist anzunehmen, daß aus der entsprechenden Simulation gewisse Rückschlüsse auf das tatsächliche Zugriffsverhalten gezogen werden können. Daher sollen im folgenden an diesem Beispiel die Ursachen für die starken Verzögerungen exemplarisch untersucht werden.

Wie bereits in der Auswertung beschrieben und in Abbildung 5.16 zu erkennen, benötigen je zwei parallel ablaufende Iterationen jeweils ähnliche Ausführungszeiten, die aber im Verlauf der Messung stark schwanken. Die Simulation zeigt für die Minimalwerte eine gute Übereinstimmung der Meßwerte, weicht aber in den Maximalwerten erheblich von der Messung ab. Diese Abweichung ist ausschlaggebend für die hohe Abweichung der Mittelwerte von Simulation und Messung.

Aus der Graphik der Simulation läßt sich erkennen, daß die Ausführungszeiten im wesentlichen auf drei disjunkte Intervalle verteilt sind. Tabelle 5.16 gibt einen Überblick über die jeweils ermittelten Laufzeiten und Konfliktanzahlen. Der Einfluß der

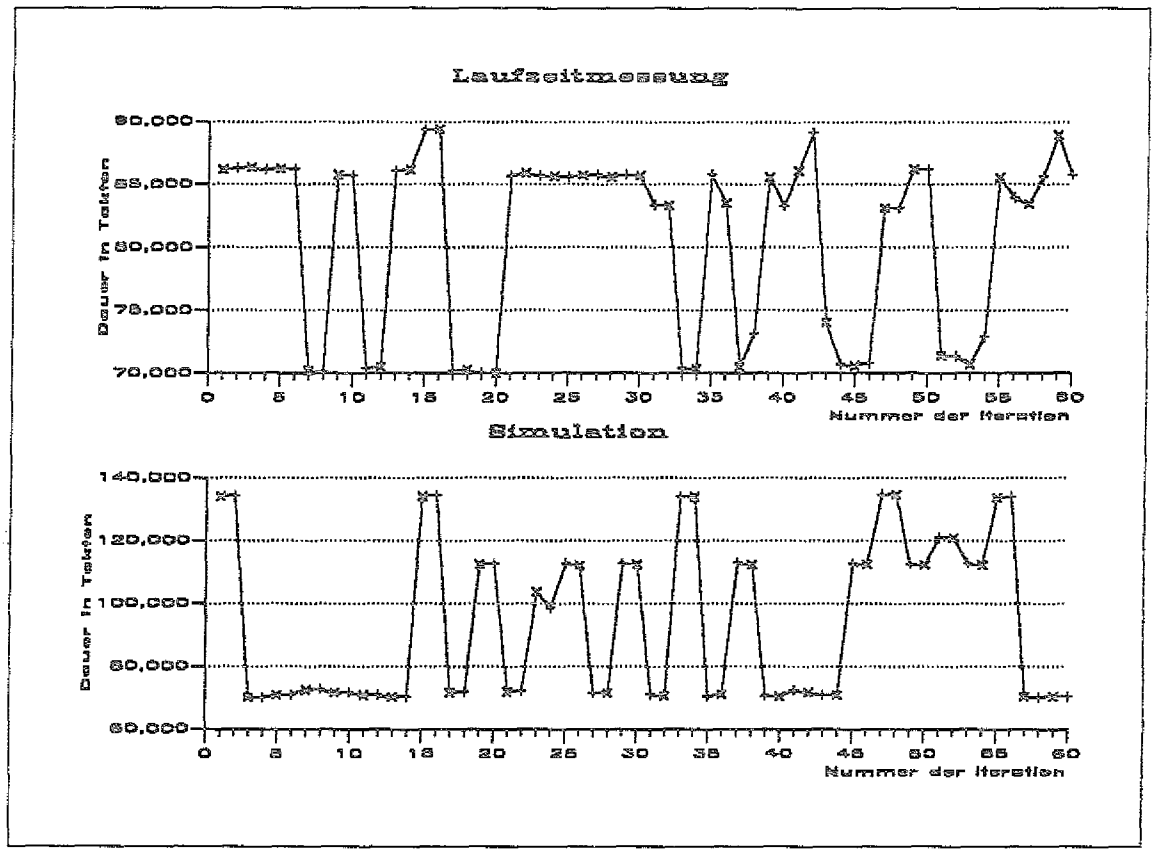


Abbildung 5.16: Gegenüberstellung von Messung und Simulation bei 2 Prozessoren und Stride 8

Simultan- und Sektionskonflikte erscheint gering gegenüber dem der Bankkonflikte und Portblockierungen. Eine Portblockierung ist die Verhinderung eines durchführbaren Zugriffs auf einem Port durch den Zugriffskonflikt auf einem anderen Port derselben CPU. Während für die Bankkonflikte ein zu den Laufzeiten proportionaler Verlauf zu erkennen ist, läßt sich für die anderen Konfliktarten kein direkter Zusammenhang zur Laufzeit angeben. Die unterschiedlichen Laufzeiten in den einzelnen Intervallen erschweren die Erkennung von Zusammenhängen. Das Verhalten einzelner Iterationen soll daher in den folgenden Abschnitten näher mit Hilfe der

	Intervall 1	Intervall 2	Intervall 3
Laufzeit	70045 - 72814	112159 - 113003	133704 - 134759
Simultankonflikte	2 - 792	263 - 1115	5 - 119
Bankkonflikte	592 - 9165	66680 - 122402	110488 - 137272
Sektionskonflikte	0 - 1004	512 - 2489	3 - 46
Portblockierungen	43 - 6372	2961 - 60536	34778 - 59840

Tabelle 5.16: Mit der Simulation ermittelte Laufzeiten (in Takten) und Konfliktanzahlen bei 2 Prozessoren und Stride 8

Trace-Datei der Simulation untersucht werden.

Bei einem Zugriff mit Stride 8 liegen 8 Bänke im Zugriffsstrom. Die Bandbreite von $B_8 = \frac{64}{4*8} = 2$ erlaubt im Mittel einen Zugriff auf maximal 2 Elemente pro Takt. Bei 2 Prozessoren konkurrieren 4 Ports um diese Zugriffe mit dem Ergebnis, daß durch eine gegenseitige Behinderung auf weniger als durchschnittlich 2 Elemente pro Takt zugegriffen werden kann. Betrachtet man nun für eine Iteration mit erhöhter Ausführungszeit das in der Trace-Datei der Simulation aufgezeichnete Zugriffsverhalten, so läßt sich im Verlauf der Bearbeitung die in Tabelle 5.17 dargestellte Situation erkennen. Innerhalb von vier Takten können alle Ports nur

Takt		...	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	...
CPU 0	Port 1	...	692	B	B	B	693	B	B	B	694	B	...
	Port 2	...	723	B	B	B	724	B	B	B	725	B	...
CPU 1	Port 1	...	645	X	X	X	646	X	X	X	647	X	...
	Port 2	...	610	B	B	B	611	B	B	B	612	B	...

Tabelle 5.17: Zugriffsmuster während einer stark verzögerten Iteration

jeweils einmal zugreifen, wobei alle Zugriffe in demselben Takt stattfinden. Im Mittel wird also lediglich ein Element pro Takt geladen. Für die folgenden drei Takte erleiden drei Ports Bankkonflikte (B), während das vierte Port wegen der Portblockierung (X) nicht zugreifen kann. Auf diese Weise wird in jedem Takt ein möglicher Zugriff verhindert. Dieses Verhalten wiederholt sich periodisch und wird erst dann durchbrochen, wenn ein Port das Laden eines Teilvektors beendet hat. Daraufhin stellt sich eher zufällig ein anderes Zugriffsmuster ein, das jedoch nicht zwangsweise zu einer höheren Transferrate führt. Setzt das zuvor freigewordene Port mit seinem nächsten Vektorzugriff ein, so entsteht nach einer kurzen, unübersichtlich verlaufenden Übergangssituation wiederum das zuvor beschriebene Zugriffsmuster.

Betrachtet man nun die Bearbeitung während der Berechnung von zwei parallel laufenden Iterationen mit kürzerer Laufzeit, so ist anfangs ein ähnliches Verhalten wie zuvor beschrieben zu beobachten. Unvermittelt erfolgt jedoch ein Wechsel des Zugriffsmusters. Zwei auf unterschiedliche Prozessoren verteilte Ports beenden ihre Vektorzugriffe nahezu gleichzeitig. Daraufhin können die beiden verbleibenden Ports ungehindert auf den Speicher zugreifen und jeweils pro Takt einen Wert übertragen. Tabelle 5.18 zeigt die Ausgabe der Simulation für diesen Fall. Etwa 24 Takte nach dem Wechsel des Zugriffsverhaltens beginnen die beiden anderen Ports mit ihrem nächsten Vektorzugriff. Das zuvor beschriebene Zugriffsverhalten setzt daraufhin wieder ein und dominiert für etwa 64 Takte, bis die beiden anderen Ports ihre Vektorbearbeitung beendet haben. Der Wechsel der beiden Muster erfolgt periodisch auf diese Weise bis zum Ende der Bearbeitung der Iteration. Durch die zwischenzeitlich eintretende Situation kann die zur Verfügung stehende Bandbreite voll ausgenutzt werden. Daher entsteht im Gesamtergebnis für die betrachteten Iterationen ein besseres Ergebnis als in der zuerst diskutierten Situation.

Takt		...	28329	28330	28331	28332	28333	28334	28335	...
CPU 0	Port 1	...	2075	2076	2077	2078	2079	2080	2081	...
	Port 2	...	–	–	–	–	–	–	–	...
CPU 1	Port 1	...	–	–	–	–	–	–	–	...
	Port 2	...	1767	1768	1769	1770	1771	1772	1773	...

Tabelle 5.18: Zugriffsmuster während einer schwach verzögerten Iteration

Die Bearbeitung zweier Iterationen mit kurzer Laufzeit wird nahezu von Beginn an durch das zuletzt beschriebene Verhalten dominiert, in dem sich starke Verzögerungen mit der Ausnutzung der Bandbreite abwechseln. Im Verlauf der Bearbeitung tritt dann die Situation ein, daß die beiden ausschließlich zugreifenden Ports ihre Vektorzugriffe nahezu gleichzeitig mit dem Einsetzen der Zugriffe der anderen Ports beenden. Die neu einsetzenden Ports können dann ebenfalls ungehindert zugreifen. Die Vektorzugriffe der nunmehr inaktiven Ports können vorerst nicht aufgesetzt werden, da die entsprechenden Register noch durch die Additionsoperation reserviert sind. Sie setzen erst dann wieder an, wenn die Zugriffe der beiden aktiven Ports beendet sind. Auf diese Weise greifen die Ports nicht mehr gleichzeitig, sondern verschränkt auf den Speicher zu. In den kurzen Überlagerungsphasen treten nur noch wenige Zugriffskonflikte auf. Dieses Verhalten wird erst dann unterbrochen, wenn ein Prozessor die Speicherzugriffe für eine Iteration beendet und nach einer durch die Bearbeitung von Reduktion und Initialisierung für die nächste Iteration bestimmten Zwischenzeit wiederum mit beiden Ports gleichzeitig auf den Speicher zugreift.

Offensichtlich stellen sich im vorliegenden Fall je nach Zeitverhalten der vorherigen Vektorzugriffe verschiedene stabile Zugriffsmuster ein, die dann den Verlauf der Bearbeitung bestimmen. Diese Zugriffsmuster sind gegenüber äußeren Einflüssen sehr labil. Die Messungen zeigen, daß gerade für 2 Prozessoren und Stride 8 erhebliche Unterschiede bei der Bearbeitungszeit verschiedener Messungen auftreten.

Um die Ursache der beschriebenen Laufzeitverzögerungen gegenüber der Modellvorstellung einzugrenzen, wurden verschiedene Parameter der Simulation variiert. Für Änderungen der Vektorlänge und der Iterationsanzahl wurde kein großer Einfluß auf das Ausführungsverhalten festgestellt. Wird jedoch die Blockierung der Ports durch einen Zugriffskonflikt auf einem anderen Port derselben CPU (kurz: die Portblockierung) vernachlässigt, so weichen die unter diesen Voraussetzungen simulierten Laufzeiten kaum noch von den Modellvorstellungen ab. Ohne die Berücksichtigung dieser Portblockierungen werden von der Simulation also nahezu die Optimalwerte erreicht. Die durch die Simulation ermittelten Verzögerungen sind also eindeutig auf Einflüsse der Portblockierung zurückzuführen. Daher liegt die Vermutung nahe, daß die gemessenen Verzögerungen der vier abweichenden Laufzeitmessungen ebenfalls durch diese Portblockierungen verursacht werden. Für die Messungen stellen sich jedoch die Auswirkungen anders dar, als sie aufgrund der verfügbaren Hardware-

Informationen zu erwarten gewesen wären. In diesem Zusammenhang sei auf die Untersuchungen von Hofemann [Hof90] und Detert [DH91] hingewiesen, die bei Untersuchungen von Vektorzugriffen eines Prozessors mit verschiedenen Vektorlängen bei Stride 8 ebenfalls Effekte beobachteten, die mit der veröffentlichten Hardware-Beschreibung nicht zu erklären sind.

5.5 Meßergebnisse auf der CRAY Y-MP

In den folgenden Abschnitten werden die für die CRAY Y-MP erzielten Ergebnisse vorgestellt und diskutiert. Da für die Messungen die gleiche Meßumgebung zur Verfügung steht wie auf der CRAY X-MP, erlauben die Ergebnisse einen direkten Vergleich der Auswirkungen der unterschiedlichen Architekturen. Neben einem vergrößerten Wertebereich für die Parameter Prozessoranzahl und Stride sind bei der Auswertung folgende, gegenüber der CRAY X-MP veränderte Architekturmerkmale zu beachten:

- Die Bankzykluszeit der CRAY Y-MP beträgt 5 Takte.
- Die Änderung der Zuordnung von Speicheradressen und Sektionen schränkt den Zugriff schon ab Stride 4 auf eine Sektion ein. Ein Prozessor kann also ab Stride 4 nur noch mit einem Port zugreifen.
- Ab Stride 32 greifen alle Prozessoren auf eine Untersektion zu. Aus Sicht eines Prozessors ist ein Pfad zu einer Untersektion durch einen Zugriff für 5 Takte belegt.

Die schnellere Zykluszeit der CRAY Y-MP wirkt sich nicht auf die Betrachtungen aus, da als Einheit für alle Ergebnisse die Anzahl der benötigten Takte verwendet wird.

Zur Bestimmung der IR-Zeiten werden analog zur CRAY X-MP mehrere Meßreihen mit unterschiedlichen Vektorlängen $2^i, i \in \{8, \ldots, 16\}$ vorgenommen. Tabelle 5.19 zeigt den nach Abschnitt 3.3.3 aus den Ergebnissen berechneten IR-Aufwand. Im Gegensatz zur CRAY X-MP treten dabei starke Schwankungen bei Mehrprozessormessungen nur noch in Ausnahmefällen auf (Strides 4 – 16 bei 6 – 8 Prozessoren). Die *Häufungswerte* bezeichnen überdurchschnittlich oft festgestellte Zeiten. Analog zur CRAY X-MP sind die errechneten Zeiten für einen Prozessor konstant und fallen – mit einer Ausnahme bei Stride 8 – mit wachsendem Stride. Analog zum bisherigen Vorgehen werden die Zeiten der Einzelprozessormessungen auch auf die Mehrprozessormessungen übertragen und gehen in Abhängigkeit vom Stride zusammen mit den Ergebnissen des Modells in den Vergleich der einzelnen Laufzeiten mit ein. Es sei nochmals auf den durch diese Festlegung entstehenden leichten Fehler bei der Berechnung des Modellwertes hingewiesen.

	1 Prozessor	2 – 8 Prozessoren	
		Bereich	Häufungswerte
Stride 1 – 4	341	332 – 335	333, 334
Stride 8	360	357 – 370	—
Stride 16	305	299 – 321	—
Stride 32 – 256	296	283 – 308	298

Tabelle 5.19: Für den IR-Aufwand auf der CRAY Y-MP ermittelte Zeiten (in Takten)

Über die Auswirkungen des Interrupts Minor Clock Cycle können für die CRAY Y-MP ähnliche Aussagen wie für die CRAY X-MP getroffen werden. Der Interrupt tritt auch bei diesem System jede 60stel Sekunde auf und läßt sich in vielen Messungen über die Periodendauer von umgerechnet etwa 2.8 Millionen Takten identifizieren. Analog zum Vorgehen bei den Messungen der CRAY X-MP werden auch hier die Auswirkungen des Interrupts für die Auswertung nach Möglichkeit eliminiert. Ein weiterer Effekt tritt bei der Ausführung auf einer CPU bei jedem Stride auf. Die erste Iteration benötigt jeweils 18 Takte mehr als alle übrigen, ungestörten Iterationen. Bei allen Messungen ab Stride 8 zeigt sich periodisch der Einfluß des Minor Clock Cycle. Ansonsten treten keinerlei Störungen auf, und die Ausführungszeiten der einzelnen Iterationen stimmen bei gleichem Stride überein.

Überblick

Betrachtet man die Meßergebnisse im Überblick, so läßt sich eine Gliederung in vier Gruppen erkennen. Dabei können jeweils über die Strides 1 – 2, 4 – 8, 16 – 32 und 64 – 256 weitgehend gemeinsame Aussagen getroffen werden.

Im ersten Bereich der Strides 1 und 2 bewegen sich die Meßwerte unabhängig von der Prozessoranzahl in der gleichen Größenordnung und lassen keine nennenswerten Abweichungen erkennen. Betrachtet man die Ergebnisse des zweiten Bereichs (Strides 4 und 8), so läßt sich ein Trend feststellen, daß mit wachsender Prozessoranzahl die Streuung der Ausführungszeiten einzelner Iterationen je Messung zunimmt. Für Stride 4 liegen die jeweils minimalen Laufzeiten wieder etwa in der Größenordnung der im ersten Bereich ermittelten Werte, bei doppeltem Stride liegen sie etwa 20% darüber. Beachtenswert ist das stark abweichende Verhalten des Meßwertes für Stride 8 und 8 Prozessoren. Die jeweiligen Durchschnittswerte verändern sich für kleinere Prozessoranzahlen (bis zu 6 Prozessoren bei Stride 4 und bis zu 4 Prozessoren bei Stride 8) nicht wesentlich, steigen dann aber deutlich.

Für den dritten Bereich (Strides 16 und 32) sind die Meßwerte wieder weitgehend invariant bezüglich der Prozessoranzahl. Die Ausführungszeiten liegen je nach Stride etwa doppelt so hoch wie bei den entsprechenden Messungen mit halbem Stride. Für den letzten Bereich der Strides 64 – 256 fällt vor allem die geringe Streuung der Minimalwerte auf, die sich nur über wenige Takte erstreckt. Für die Maximalwerte

ist die folgende Beobachtung typisch: Es werden – jeweils abhängig vom Stride – nur zwei Meßwerte notiert (unter Vernachlässigung geringer Abweichungen). Im Verlauf der Variation der Prozessoranzahl ist stets ein sprunghafter Anstieg zu registrieren: für Stride 64 bei 5 Prozessoren um den Faktor 15, für Stride 128 bei 3 Prozessoren um den Faktor 30 sowie für Stride 256 bei 2 Prozessoren um den Faktor 60. Für die Durchschnittswerte sind je Stride ebenso zwei Phasen zu unterscheiden, deren erste analog zu den Maximalwerten verläuft. In der zweiten Phase ist für die Mittelwerte ein linearer Anstieg bei steigender Prozessoranzahl zu verzeichnen.

Stride 1

Die Verwendung der geringsten Schrittweite für Vektorzugriffe bewirkt die Ausnutzung der gesamten Bandbreite des Systems. Dementsprechend sind nur geringe Einflüsse durch Zugriffskonflikte zu erwarten, von denen keine spürbare Verlängerung der Laufzeit einer Messung zu erwarten ist. Dies wird durch die Darstellung der Meßergebnisse in Tabelle 5.20 bestätigt. Das Modell liefert für alle Prozessoranzahlen einen identischen Wert, die jeweiligen Abweichungen von den Meßwerten sind zu vernachlässigen. Betrachtet man die Messungen im Detail, so ist mit zunehmender Prozessoranzahl eine leicht steigende Tendenz in bezug auf Umfang und Häufigkeit der einzelnen Störungen zu erkennen. Bei bis zu 4 genutzten Prozessoren lassen sich Muster ähnlich der Darstellung in Abbildung 5.17 erkennen. Mit der Zunahme der Störungen bei Nutzung von mehr Prozessoren sind diese Muster kaum noch zu erkennen. Die Verzögerungen einzelner Iterationen in den Messungen betragen im Extremfall mit maximal 279 Takten bei 7 Prozessoren nur etwa 0.4% vom entsprechenden Minimalwert.

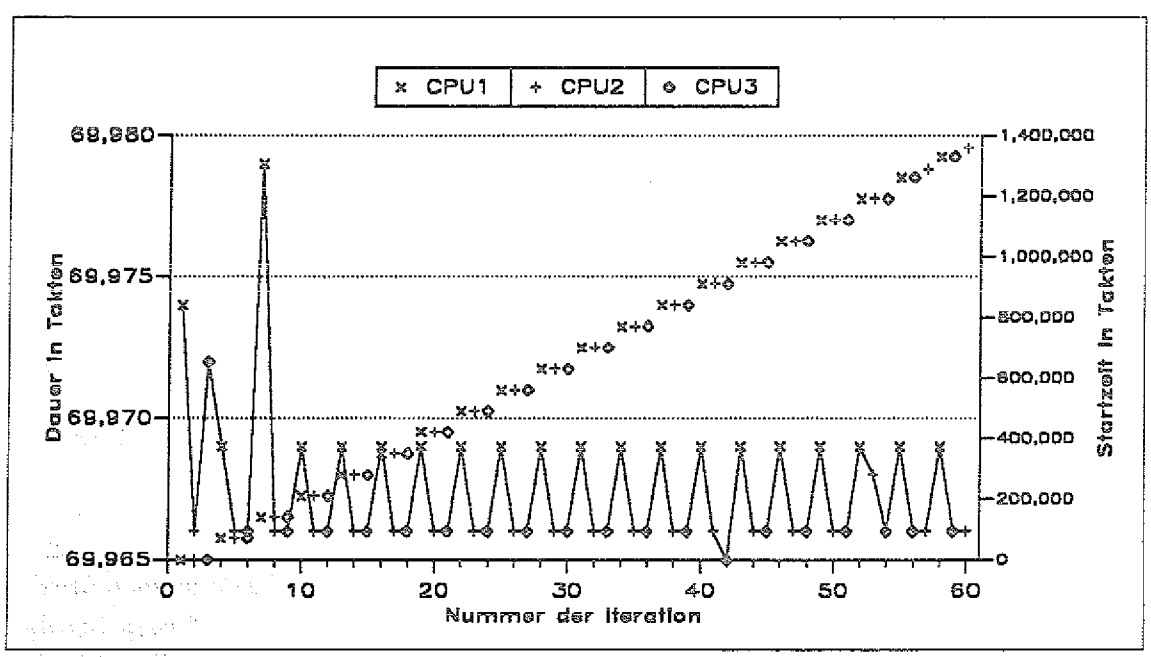


Abbildung 5.17: Meßergebnis bei 3 Prozessoren und Stride 1

Prozessoren	1	2	3	4	5	6	7	8
Minimum	69971	69965	69965	69966	69965	69966	69966	69966
Mittelwert	69971	69967	69967	69969	69970	69969	69979	69969
Maximum	69989	69974	69979	70025	70026	70016	70245	70014
Modell	69973	69973	69973	69973	69973	69973	69973	69973
Abweichung	0%	0%	0%	0%	0%	0%	0%	0%

Tabelle 5.20: Ausführungszeiten (in Takten) für Stride 1

Stride 2

Durch Zugriffe mit Stride 2 wird die Hälfte der vorhandenen Bänke genutzt. Diese verteilen sich auf zwei Sektionen, auf die von jedem Prozessor abwechselnd zugegriffen wird. Tabelle 5.21 ist wie im vorangegangenen Fall zu entnehmen, daß nur geringe Abweichungen der Meßwerte vom Modell auftreten, die sich auch lediglich für 5, 7 und 8 Prozessoren prozentual bemerkbar machen. Die Minimalwerte liegen bei allen Mehrprozessormessungen um 3 – 4 Takte höher als bei Stride 1. Die Maximalwerte erreichen bei 7 Prozessoren mit 385 Takten eine Abweichung von 0.55% vom Minimalwert, sind also in ihrem Umfang für die Bearbeitung ebenfalls zu vernachlässigen.

Prozessoren	1	2	3	4	5	6	7	8
Minimum	69971	69969	69969	69969	69968	69969	69970	69969
Mittelwert	69971	69970	69970	69972	69996	69974	70008	70049
Maximum	69989	69990	69994	70000	70250	70002	70355	70270
Modell	69973	69973	69973	69973	69973	69973	69973	69973
Abweichung	0%	0%	0%	0%	0.03%	0%	0.05%	0.10%

Tabelle 5.21: Ausführungszeiten (in Takten) für Stride 2

Mit Hilfe der graphischen Auswertung ist zu erkennen, daß im Vergleich zu den Messungen mit Stride 1 mehr Konflikte auftreten. Besonders auffallend ist das Ausführungsverhalten bei 5 Prozessoren (siehe Abbildung 5.18, Seite 102): Nach anfangs kleineren Konfliktsituationen im Bereich von 50 Takten treten für eine Dauer von 30 Iterationen (also etwa 6 Iterationen je Prozessor) nahezu keine Beeinflussungen auf, während danach Verzögerungen von bis zu 300 Takten einsetzen, die sich über 15 Iterationen erstrecken. Ein ähnliches Verhalten läßt sich auch bei 7 Prozessoren beobachten.

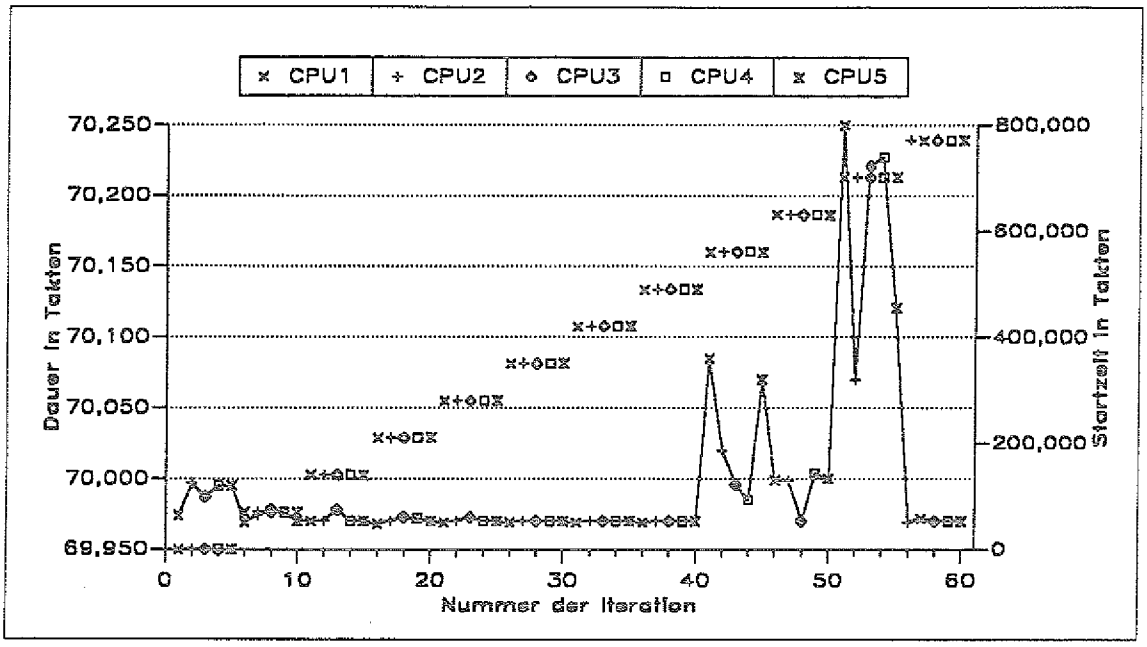


Abbildung 5.18: Meßergebnis bei 5 Prozessoren und Stride 2

Stride 4

Ab Stride 4 liegen alle referierten Bänke in einer Sektion. Wegen der Beschränkung auf einen Datenpfad zwischen einer CPU und einer Sektion analog zur CRAY X-MP kann damit von einem Prozessor lediglich ein Wert pro Takt aus dem Speicher geladen werden (siehe Abschnitt 1.1). Zugriffskonflikte beim Laden eines Vektorregisters wirken sich damit wegen der fehlenden Pufferfunktion eines parallelen Ladestroms direkt auf die Ausführungszeiten aus. Gegenüber den bisherigen Situationen ist also beim Einsatz mehrerer Prozessoren mit deutlich höher ausfallenden Verzögerungen zu rechnen.

Betrachtet man die Minimalwerte in Tabelle 5.22 unter Ausschluß der Ergebnisse für 7 und 8 Prozessoren, so liegen diese wiederum in einer mit dem Stride 1 vergleichbaren Größenordnung, damit also niedriger als bei Stride 2. Offensichtlich treten dort in der Anfangsphase der Bearbeitung einer Iteration häufiger Konfliktsituationen auf. Diese Differenzen bezüglich des Konfliktverhaltens sind dadurch zu

Prozessoren	1	2	3	4	5	6	7	8
Minimum	69971	69966	69966	69966	69966	69967	70217	70678
Mittelwert	69971	69970	69976	69980	70094	70415	73604	78543
Maximum	69989	70002	70062	70185	71223	72648	84847	93876
Modell	69973	69973	69973	69973	69973	69973	69973	69973
Abweichung	0%	0%	0%	0.01%	0.17%	0.63%	5.18%	12.24%

Tabelle 5.22: Ausführungszeiten (in Takten) für Stride 4

Prozessoren	6	7	8
Minimale Abweichung	0.11%	0.65%	2.78%
Maximale Abweichung	4.74%	5.22%	13.41%

Tabelle 5.23: Bereich der Abweichungen vom Modell für Stride 4 über mehrere Meßreihen

erklären, daß bei Stride 2 mit 2 Ports auf lediglich 2 Sektionen zugegriffen wird, während bei Stride 1 vier Sektionen zur Verfügung stehen. Die beiden Ports einer CPU behindern sich im Fall Stride 2 gegenseitig bei den Zugriffen auf die Sektionen.

Besonders deutlich ist der kontinuierliche Anstieg der Mittelwerte ab 6 Prozessoren. In diesen Fällen sind gegenüber den bisherigen Betrachtungen deutliche Verzögerungen zu erkennen, die nicht vom analytischen Modell prognostiziert werden. Betrachtet man die Meßergebnisse anderer Meßreihen, so sind für 6 – 8 Prozessoren jeweils signifikante Abweichungen vom Modell zu vermerken, die je nach Meßreihe stark unterschiedlich ausfallen. In Tabelle 5.23 sind die Bereiche der Abweichungen vom Modell dargestellt, die in mehreren Meßreihen ermittelt wurden. Es ist zu erkennen, daß mit zunehmender Prozessoranzahl auch stärkere Schwankungen auftreten. Die graphische Darstellung zeigt bereits ab 6 Prozessoren, daß durch Speicherzugriffskonflikte beeinflußte Iterationen das Ausführungsverhalten dominieren. Dabei treten je nach Verzögerung unterschiedliche Muster auf. Bei größeren Abweichungen lassen sich Muster ähnlich Abbildung 5.19 erkennen, während das Verhalten der weniger gestörten Messungen eher Abbildung 5.20 entspricht. Die

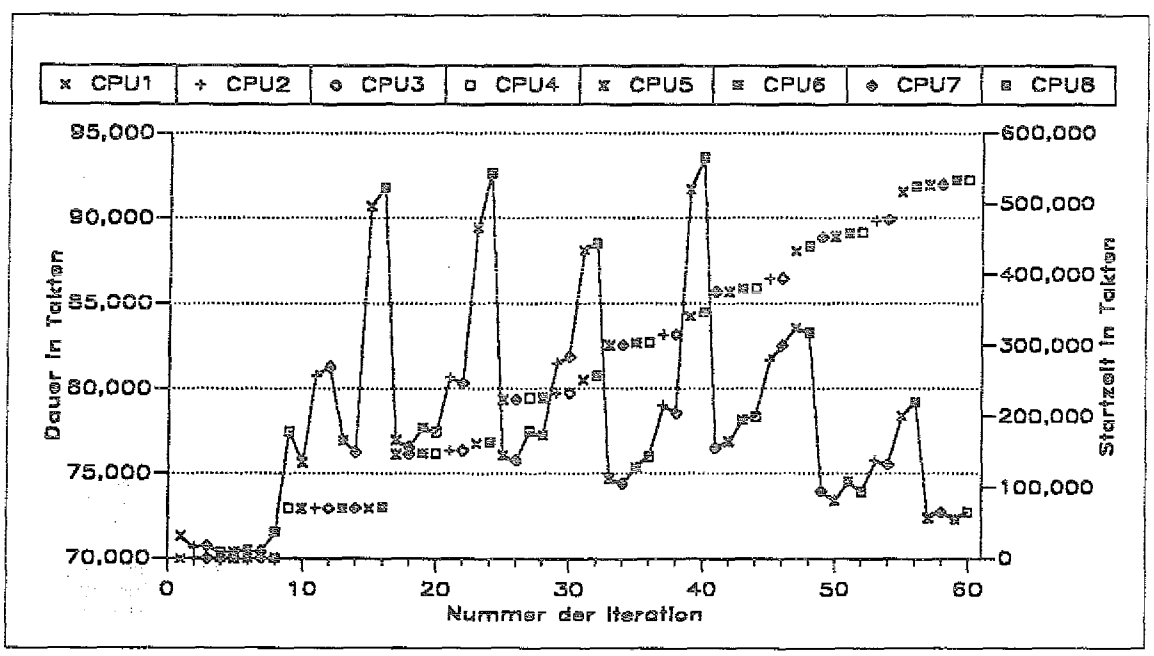


Abbildung 5.19: Hohe Abweichung bei 8 Prozessoren und Stride 4

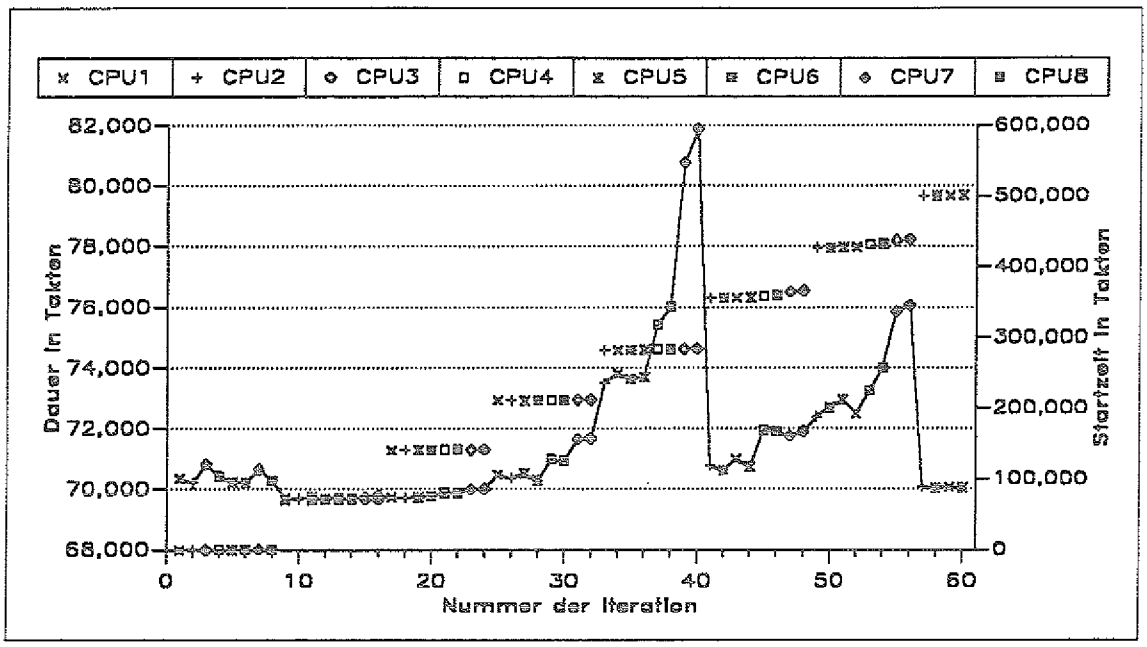


Abbildung 5.20: Geringe Abweichung bei 8 Prozessoren und Stride 4

relativen Bandbreiten liegen für 8 Prozessoren bei 1.6, für 7 Prozessoren bei 1.83 und bei 6 Prozessoren sogar über 2 pro CPU, so daß prinzipiell eine genügend große Bandbreite für einen konfliktfreien Zugriff zur Verfügung steht. Vergleichbare Situationen führen auf der CRAY X-MP nicht zu Verzögerungen dieser Größenordnung. Eine Ursache dafür könnte in der Wechselwirkung verschiedener Konfliktarten zu suchen sein, wobei auch die Untersektionsstruktur der CRAY Y-MP einen erheblichen Einfluß haben dürfte.

Stride 8

Bei Stride 8 sind die 32 referierten Bänke auf 4 Untersektionen verteilt, die zyklisch angesprochen werden, d.h. auf jede Untersektion wird jeden vierten Takt zugegriffen. Da eine Untersektion jedoch durch einen Zugriff für 5 Takte belegt ist, entsteht in jedem fünften Zugriff ein Untersektionskonflikt. Tabelle 5.24 zeigt das Zugriffsverhalten am Beispiel von 3 Prozessoren. Betrachtet wird eine Startsituation eines Vektorzugriffs, wobei vorausgesetzt wird, daß alle Ladeströme gleichzeitig einsetzen und die Prioritäten der zugreifenden Prozessoren entsprechend aufsteigender Prozessornummer vergeben sind. In der Tabelle beschreiben Kleinbuchstaben

Takt	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
CPU 1	a1	b2	c3	d4	U	a5	b6	c7	d8	U	a9	b10	c11	d12	...
CPU 2	S	B	B	B	B	a1	b2	c3	d4	U	a5	b6	c7	d8	...
CPU 3	S	B	B	B	B	S	B	B	B	B	a1	b2	c3	d4	...

Tabelle 5.24: Zugriffsverhalten von 3 Prozessoren bei Stride 8

die referierten Untersektionen ($a \ldots d$), Zahlen kennzeichnen den Vektorindex des geladenen Elements. Großbuchstaben stehen für Konfliktsituationen, wobei U einen Untersektionskonflikt, S einen Simultankonflikt und B einen Bankkonflikt bezeichnen. Durch die Blockierung jedes fünften Zugriffs ist im Mittel eine Verzögerung der Laufzeit von etwa 20% gegenüber den beim vorangegangenen Stride betrachteten Werten zu erwarten.

Prozessoren	1	2	3	4	5	6	7	8
Minimum	82272	82272	82281	82287	82279	82771	84499	112932
Mittelwert	82277	82298	82295	82478	88894	99195	122083	140541
Maximum	82566	82519	82407	82981	118296	134661	139029	177501
Modell	82280	82280	82280	82280	82280	82280	82280	82280
Abweichung	0%	0.02%	0.01%	0.24%	8.03%	20.55%	48.37%	70.80%

Tabelle 5.25: Ausführungszeiten (in Takten) für Stride 8

In Tabelle 5.25 ist die Bestätigung dieser Überlegungen für bis zu 4 Prozessoren sichtbar. Alle weiteren Messungen zeigen – deutlich stärker noch als bei Stride 4 – zusätzliche Verzögerungen. Betrachtet man die Resultate mehrerer Meßreihen zu Stride 8, so ist eine sehr starke Streuung der Meßergebnisse für entsprechende Messungen zu vermerken. Tabelle 5.26 verdeutlicht, daß die Streuung der Meßwerte für 6 und 7 Prozessoren im Bereich von etwa 25% liegt und für 7 und 8 Prozessoren die Messungen regelmäßig sehr stark verzögert werden. Entsprechende Fluktuationen

Prozessoren	4	5	6	7	8
Minimale Abweichung	0.08%	2.99%	3.21%	36.52%	69.50%
Maximale Abweichung	6.95%	11.82%	27.05%	59.35%	82.62%

Tabelle 5.26: Bereich der Abweichungen vom Modell für Stride 8 über mehrere Meßreihen

werden auch bei Referenzmessungen mit der Meßroutine ICPUSED festgestellt. Betrachtet man die relativen Bandbreiten bezüglich der jeweiligen Prozessoranzahl, so liegen die entsprechenden Werte für bis zu 6 Prozessoren über 1, für 7 Prozessoren bei 0.91 und bei 8 Prozessoren analog zur Begrenzung durch die Untersektionen bei 0.8. Als Ursache für die überproportionalen Verzögerungen kann daher wie im vorangegangenen Fall eine Wechselwirkung der verschiedenen Konfliktarten angenommen werden.

Stride 16

Die bei Stride 16 referierten 16 Bänke liegen in zwei Untersektionen, auf die abwechselnd zugegriffen wird. Nach zwei erfolgreichen Zugriffen treten dann für jeden Prozessor 3 Takte lang Untersektionskonflikte auf. Da lediglich 2 von 5 Zugriffen

Prozessoren	1	2	3	4	5	6	7	8
Minimum	164143	164141	164142	164142	164145	164145	164143	164156
Mittelwert	164161	164163	164162	164183	164190	164202	164217	165572
Maximum	164468	164490	164366	164483	164427	164361	164436	172454
Modell	164145	164145	164145	164145	164145	164145	164145	164145
Abweichung	0%	0.01%	0.01%	0.02%	0.02%	0.03%	0.04%	0.86%

Tabelle 5.27: Ausführungszeiten (in Takten) für Stride 16

erfolgreich sind, verzögert sich die Ausführung von einer Iteration um den Faktor 2.5 gegenüber dem optimalen Fall. Diese Verzögerung ist den Meßwerten in Tabelle 5.27 deutlich zu entnehmen. Im Gegensatz zur Abweichung bei Stride 8 zeigt sich hier wiederum eine sehr gute Übereinstimmung zwischen analytischem Modell und Laufzeitmessung für nahezu alle betrachteten Fälle. Lediglich für 8 Prozessoren tritt eine höhere Verzögerung auf, die jedoch weniger als 1% Abweichung zum Modell beträgt. Ansonsten sind die Unterschiede bezüglich der Minimal- und Mittelwerte nur gering.

Abbildung 5.21 ist für die Messung mit 8 Prozessoren ein ähnliches Verhalten zu entnehmen, wie bereits in den Messungen 4/4 und 2/8 bei der CRAY X-MP (siehe Seite 78). Die Ausführungszeiten parallel ausgeführter Iterationen sind nahezu identisch, für unterschiedliche parallele Phasen treten jedoch deutlich differierende Laufzeiten auf. Ähnliche Effekte treten auf der CRAY Y-MP nur bei Messungen mit Stride 16 auf. Dieses Verhalten deutet analog auf das Auftreten verschiedener, sich gegenseitig beeinflussender Zugriffskonflikte hin.

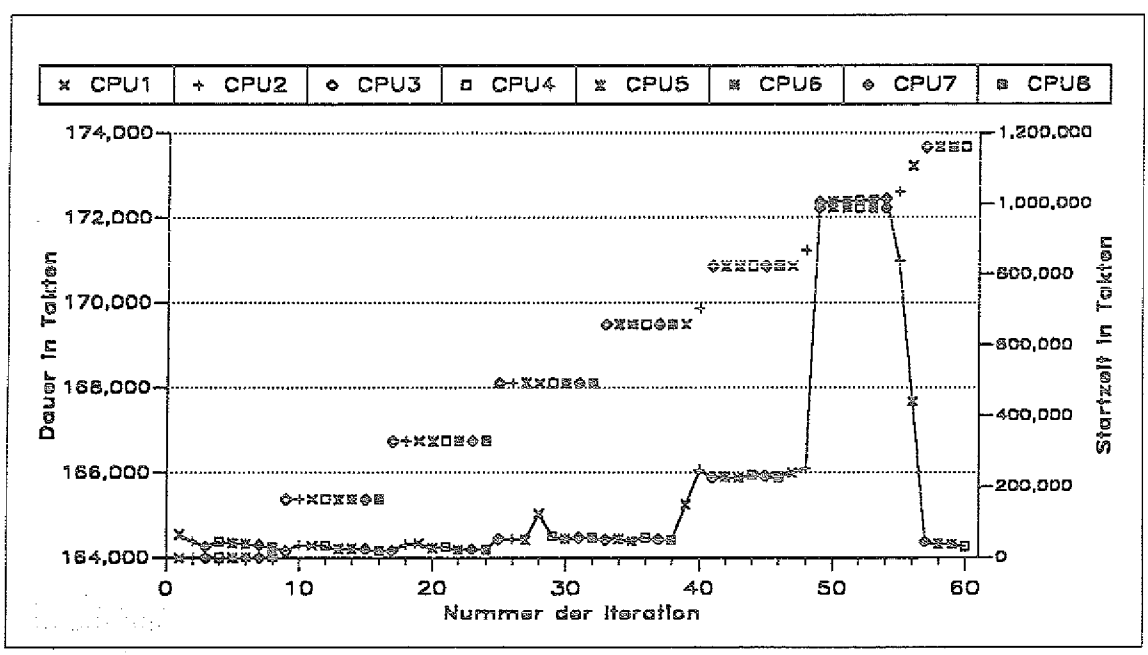


Abbildung 5.21: Meßergebnis bei 8 Prozessoren und Stride 16

Stride 32

Ab Stride 32 liegen die 8 referierten Bänke in derselben Untersektion. Alle Zugriffssituationen mit Stride 32 und höher zeichnen sich daher dadurch aus, daß jeder Prozessor nach einem Zugriff jeweils 4 Untersektionskonflikte erleidet. Ein Zugriff ist also je Prozessor höchstens jeden fünften Takt möglich. Daher wird das Zeitverhalten wie bereits in den vorherigen Fällen durch die Untersektionskonflikte bestimmt. Tabelle 5.28 zeigt wiederum eine gute Übereinstimmung von Modell und Laufzeitmessung. In keinem Fall sind deutliche Abweichungen zu erkennen.

Prozessoren	1	2	3	4	5	6	7	8
Minimum	327979	327975	327975	327975	327979	327980	327978	327980
Mittelwert	328007	327984	327995	327996	328019	328020	328035	328072
Maximum	328861	328061	328115	328122	328196	328201	328298	328484
Modell	327976	327976	327976	327976	327976	327976	327976	327976
Abweichung	0%	0%	0%	0%	0.01%	0.01%	0.01%	0.02%

Tabelle 5.28: Ausführungszeiten (in Takten) für Stride 32

Die Schwankungen der Meßwerte untereinander sind im Vergleich zur Höhe der Meßwerte sehr gering.

Stride 64

Bei Zugriff mit Stride 64 werden nur noch 4 Bänke einer Untersektion angesprochen. Bei mehr als 4 Prozessoren sind also Verzögerungen bis zum Faktor 2 bei maximal 8 Prozessoren zu erwarten. Tabelle 5.29 bestätigt diese Vermutungen und zeigt darüber hinaus eine leichte, mit zunehmender Prozessoranzahl steigende Abweichung der von den zusätzlichen Konflikten betroffenen Messungen gegenüber dem Modell. In der Tabelle fällt vor allem der Anstieg der Maximalwerte um einen Faktor 15 beim Übergang von 4 auf 5 Prozessoren auf. Die Graphik für die Messung mit 5 Prozessoren zeigt für die erste Iteration eine verlängerte Laufzeit mit der Dauer des Maximalwerts, während die Laufzeiten der übrigen Iterationen denen der Messung mit 4 Prozessoren gleichen. Offensichtlich wird der Prozessor, der die erste Iteration bearbeitet, bis zum Ende der Bearbeitung verzögert, während die anderen Prozessoren ihre Arbeit normal durchführen. Eine vergleichbare Situation kann auch für alle folgenden Messungen zu diesem Stride beobachtet werden. Ledig-

Prozessoren	1	2	3	4	5	6	7	8
Minimum	327979	327978	327975	327978	327978	327978	327978	327973
Mittelwert	327987	327996	328000	328014	405348	481124	557680	634214
Maximum	328051	328119	328155	328176	4912393	4930541	4938527	4940691
Modell	327976	327976	327976	327976	409896	491816	573736	655656
Abweichung	0%	0%	0%	0.01%	1.10%	2.17%	2.79%	3.27%

Tabelle 5.29: Ausführungszeiten (in Takten) für Stride 64

lich 4 Prozessoren sind an der laufenden Bearbeitung beteiligt, während die übrigen auf die beschriebene Weise verzögert werden. Abbildung 5.22 zeigt beispielhaft die Meßgraphik für die Messung mit 8 Prozessoren, aus der auch zu entnehmen ist, daß der Zugriff eines Prozessors offensichtlich nicht vom Zeitpunkt seines Eintretens in die parallele Bearbeitung abhängig ist. Hier werden die Prozessoren mit den Iterationen 1, 2, 4 und 5 verzögert, während die vier verbleibenden Prozessoren ungestört arbeiten können.

Diese nicht unmittelbar erklärbare Situation erfordert eine allgemeine Betrachtung des Ausführungsverhaltens bei Stride 64. Alle vorhandenen Prozessoren bekommen zu Beginn der parallelen Bearbeitung mit kurzen Abständen jeweils eine Iteration zugeordnet. Da bei 8 Prozessoren lediglich 4 Bänke zur Verfügung stehen, erfolgen zwangsläufig gleichzeitige Zugriffswünsche mehrerer Prozessoren auf gemeinsame Bänke. Bei einem gleichzeitigen Zugriff auf eine Bank erhält nur die CPU mit der höchsten Priorität den Zugriff. Im Gegensatz zur CRAY X-MP verwendet die CRAY Y-MP zur Auflösung von Interprozessorkonflikten ein statisches Prioritätenschema. Dabei wird die Priorität eines Zugriffes aufgrund der referierten Sektion und Untersektion wie auch in Abhängigkeit von der Nummer der referierenden CPU ermittelt (siehe Abschnitt 1.2). Für den Zugriff auf eine bestimmte Bank kann also für die 8 Prozessoren eine feste Prioritätenreihenfolge festgelegt werden. Offensichtlich sind für die vier betrachteten Bänke die Prioritäten so vergeben, daß immer dieselben 4 Prozessoren verzögert werden, während die anderen Prozessoren nur durch die Untersektionskonflikte behindert werden und ansonsten immer zugreifen können. Diese Betrachtung klärt auch das unterschiedliche Verhalten einer später in die Bearbeitung eintretenden CPU. Entweder ist sie höher priorisiert und kann auf Kosten einer arbeitenden CPU die Arbeit fortsetzen (siehe Abbildung 5.22,

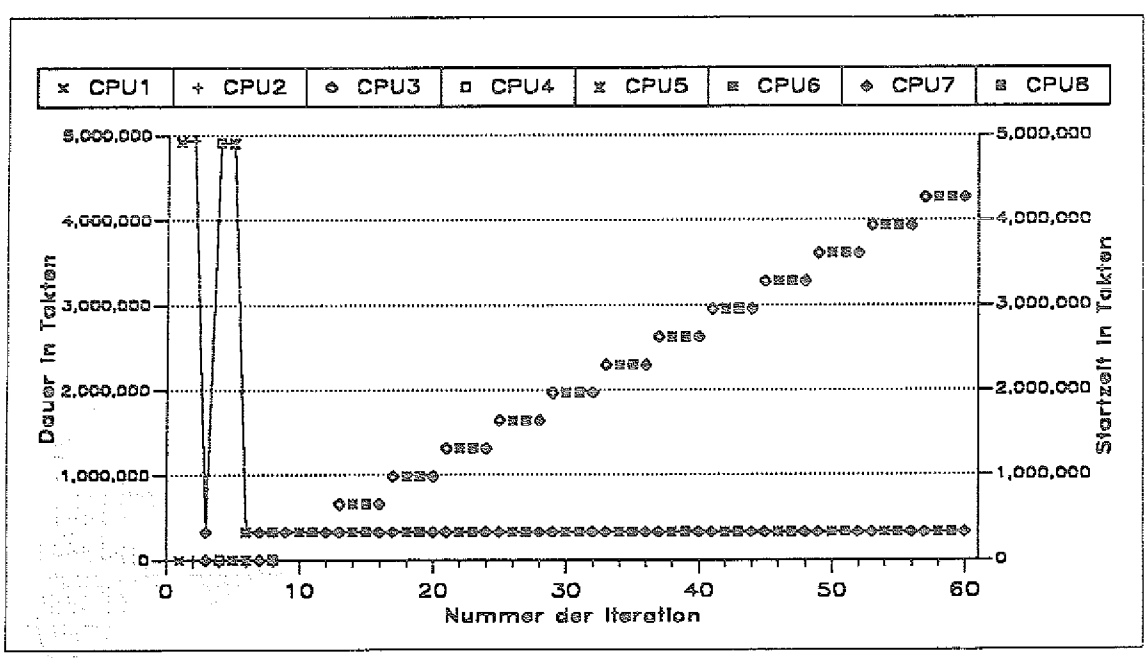


Abbildung 5.22: Meßergebnis bei 8 Prozessoren und Stride 64

Prozessor 3), oder sie hat eine niedrigere Priorität als die aktiven Prozessoren und kann daher keine Zugriffe durchführen.

Vergleicht man die Werte des Modells mit den Mittelwerten der Laufzeitmessung, so fällt auf, daß ab 5 Prozessoren die Meßwerte unter den Modellwerten liegen. Diese unerwartete Abweichung kann durch die Einschränkungen des Modells erklärt werden. Zwischen den Ladevorgängen von zwei Iterationen auf einem Prozessor entstehen für diesen Prozessor Zeiten, in denen er nicht auf den Speicher zugreift. Diese setzen sich aus den IR-Zeiten sowie aus der durch die für die Steuerung von Multitasking notwendigen Zeit zusammen. In dieser Phase können die Vektorzugriffe der Prozessoren fortfahren, die wegen einer niedrigeren Priorität ansonsten keine Zugriffe durchführen können.

Stride 128 und Stride 256

Prozessoren	1	2	3	4	5	6	7	8
Minimum	327979	327978	327978	327978	327978	327978	327978	327973
Mittelwert	327994	328004	487850	645493	797539	955022	1098394	1245865
Maximum	328109	328141	9824230	9881090	9831846	10102874	9867978	9854343
Modell	327976	327976	491816	655656	819496	983336	1147176	1311016
Abweichung	0%	0%	0.80%	1.55%	2.67%	2.87%	4.25%	4.96%

Tabelle 5.30: Ausführungszeiten (in Takten) für Stride 128

Die Interpretation der Ergebnisse von Stride 128 und 256 in den Tabellen 5.30 und 5.31 können im wesentlichen auf die Beobachtungen von Stride 64 zurückgeführt werden. Bei Stride 128 ist der Zugriff auf 2 Bänke eingeschränkt, so daß mit 2 Prozessoren bereits die gesamte verfügbare Bandbreite des Systems ausgelastet ist. Bei Stride 256 steht nur noch eine Bank zur Verfügung. Entsprechend treten bei der Nutzung von mehr Prozessoren in beiden Fällen Wartezeiten auf. Bei Stride 128 können lediglich 2 Prozessoren zugreifen, bei Stride 256 nur ein Prozessor. Alle übrigen CPUs werden analog zu den entsprechenden Situationen bei Stride 64 verzögert. Sie erhalten zwar zu Beginn der Bearbeitung eine Iteration zugewiesen, können jedoch nicht auf den Speicher zugreifen, bis die priorisierten Prozessoren ihre Arbeit beendet haben. Erst am Ende der Bearbeitung können diese Iterationen in der Reihenfolge der Prozessorprioritäten fertiggestellt werden.

Prozessoren	1	2	3	4	5	6	7	8
Minimum	327979	327978	327978	327978	327978	327978	327978	327973
Mittelwert	327979	653696	972029	1278965	1585340	1885021	2180381	2470451
Maximum	327997	19663988	19663212	19661963	19662263	19661923	19662078	19663508
Modell	327976	655656	983336	1311016	1638696	1966376	2294056	2621736
Abweichung	0%	0.29%	1.14%	2.44%	3.25%	4.13%	4.95%	5.77%

Tabelle 5.31: Ausführungszeiten (in Takten) für Stride 256

Zusammenfassung

Für niedrige Strides zeigt sich bei der CRAY Y-MP auch für mehrere zugreifende Prozessoren ein ähnlich gutes Verhalten wie bei der CRAY X-MP. Die Verzögerungen durch Zugriffskonflikte liegen hier in fast allen Fällen unter 0.2%. Für Stride 4 wurden jedoch für 7 und 8 Prozessoren Leistungseinbußen bis zu etwa 13.5% festgestellt. Ab Stride 8 macht sich die Auswirkung der Untersektionsstruktur in Verzögerungen bemerkbar. Darüber hinaus treten bei Stride 8 und mehr als 4 Prozessoren erhebliche Verzögerungen auf, die nicht unmittelbar durch die Architektur zu erklären sind. Sie liegen mit bis zu etwa 83% auch deutlich höher als vergleichbare Werte auf der CRAY X-MP. Aufgrund der ähnlichen Erscheinungsbilder kann auch hier wie bei der CRAY X-MP geschlossen werden, daß sich verschiedene Konfliktarten gegenseitig beeinflussen und zu Blockierungseffekten im Speicherverbindungsnetzwerk führen. Während für die Strides 16 und 32 die Zugriffsleistung nur durch die Untersektionskonflikte begrenzt wird, wirkt sich für höhere Strides erwartungsgemäß die reduzierte Bandbreite aus. Für diese Fälle ist ein von der CRAY X-MP deutlich abweichendes Zugriffsverhalten aufgrund der unterschiedlichen Prioritätensysteme für den gleichzeitigen Speicherzugriff festzustellen.

Zusammenfassung und Ausblick

Die parallele Ausführung von vektorisierenden Programmen kann bei CRAY-Mehrprozessorsystemen zu einer hohen Belastung des gemeinsamen Hauptspeichers führen. Die Auswirkungen von dabei entstehenden Speicherzugriffskonflikten wurden exemplarisch für die CRAY X-MP und CRAY Y-MP untersucht. Mit Hilfe eines synthetischen Modellproblems war es möglich, die Größe des referierten Speichers, die Zugriffshäufigkeit sowie die Anzahl der genutzten Prozessoren so zu variieren, daß das Konfliktverhalten auch für Extremfälle betrachtet werden konnte. Da die verfügbaren Meßmethoden sich zur Messung von Speicherzugriffskonflikten als nicht hinreichend geeignet zeigten, wurde ein eigens auf die Problemstellung zugeschnittenes Meßverfahren entwickelt. Basierend auf dem Assemblercode des Modellproblems wurde ein analytisches Modell für jedes der beiden Rechnersysteme entworfen, anhand dessen die jeweiligen Ausführungszeiten prognostiziert werden konnten. Darüber hinaus wurde die Programmausführung und das Zugriffsverhalten auf der CRAY X-MP beispielhaft mit Hilfe eines im Rahmen der Arbeit entwickelten Simulationsprogramms analysiert.

Für den überwiegenden Teil der untersuchten Fälle zeigten die Ergebnisse aus Messung, analytischem Modell und Simulation sehr gute Übereinstimmung. Die Abweichungen betrugen weniger als 1%. Mit Hilfe der Simulation konnten die auftretenden Effekte auf der CRAY X-MP weitgehend erklärt werden. Lediglich bei vier Zugriffssituationen, die jeweils an die Kapazitätsgrenze des Speicherverbindungsnetzwerks heranreichten, wiesen die Ergebnisse der verschiedenen Meßmethoden beträchtliche Differenzen auf. Zwischen Laufzeitmessung und Simulation ergaben sich dabei Abweichungen bis zu 25%, die anhand des analytischen Modells prognostizierten Ausführungszeiten wichen in diesen Fällen von den gemessenen Werten sogar bis zu 70% ab. Hier hat sich gezeigt, daß die öffentlich zugänglichen Informationen über die CRAY-Systeme nicht zur Erklärung aller auftretenden Effekte ausreichen. Die Simulation konnte aber auch in diesen Fällen dazu beitragen, die Ursachen der gemessenen Effekte zu lokalisieren.

Für niedrige Strides haben die Untersuchungen gezeigt, daß selbst bei massiven Vektorzugriffen mehrerer Prozessoren auf beiden Systemen keine signifikanten Störungen durch Zugriffskonflikte zu erwarten sind. Lediglich für Stride 8 wurde ein Verhalten beobachtet, das deutlich höhere Leistungsverluste beim Speicherzugriff erzeugt, als dies durch die zur Verfügung stehenden Ressourcen zu vermuten ist. Als Ursache konnte hier für beide Systeme eine gegenseitige Beeinflussung der ver-

schiedenen Speicherzugriffskonfliktarten herausgefunden werden. Für höhere Strides wirkt sich erwartungsgemäß die reduzierte Bandbreite auf die Speicherzugriffe aus.

Aus Sicht des Anwenders dürften spürbare Verzögerungen durch Zugriffskonflikte, die durch synchrone Speicherzugriffe mehrerer Prozessoren verursacht werden, nur in Ausnahmefällen zu erwarten sein. Die hier mit Hilfe eines synthetischen Problems erzeugten Effekte sind in ähnlicher Form nicht bei praxisorientierten Applikationen zu erwarten. Zudem zeichnet sich das automatisch parallelisierende Compiling-System CF77 durch eine sehr gute Optimierung aus, die massive Speicherzugriffe mit ungünstigem Stride weitgehend vermeidet.

Eine weitere Präzisierung der Simulation erscheint durch Berücksichtigung zusätzlicher Informationen möglich. Durch Verwendung von konkreten Meßwerten anstatt bisher empirisch in die Simulation eingehender Größen wäre eine genauere Analyse einzelner Meßergebnisse möglich. Eine Untersuchung der Einflüsse anderer Zugriffssituationen, beispielsweise ein gleichzeitiger Zugriff mehrerer Prozessoren mit unterschiedlichem Stride, wäre ebenfalls interessant.

Die in dieser Arbeit vorgestellte Methode erscheint auch zur Durchführung ähnlicher Untersuchungen auf anderen Architekturen sinnvoll. Bei einer Übertragung auf Parallelrechner mit verteiltem Speicher ließe sich beispielsweise auf diese Weise statt der im Rahmen dieser Arbeit betrachteten Auswirkung von Speicherzugriffskonflikten der Einfluß der Kommunikation auf die Ausführung von Algorithmen untersuchen.

Literaturverzeichnis

[BC90] I. Y. Bucher, and D. A. Calahan. Access conflicts in multiprocessor memories queueing models and simulation studies. In *International Conference on Supercomputing*, 428–438, 1990.

[BEK90] T. Bönniger, R. Esser und D. Krekel. CRAY Y-MP, NEC SX-3 und SIEMENS VP-S. Vergleichende Darstellung von Höchstleistungscomputern. *Wirtschaftsinformatik*, **32**, 273–286, 1990.

[Ber92] R. Berrendorf. Memory access in shared virtual memory. Interner Bericht KFA–ZAM–IB–9202, Forschungszentrum Jülich, 1992.

[Bha75] D. P. Bhandarkar. Analysis of memory interference in multiprocessors. *IEEE Transaction on Computers*, **24**, 897–908, 1975.

[Bla90] D. L. Black. Scheduling support for concurrency and parallelism in the Mach Operating System. *IEEE Computer*, **23**, 35–41, 1990.

[BLR91] F. Barriuso, S. LaCroix, and S. Reinhardt. Exploiting fine-grain parallelism in a multiprogramming environment on CRAY Y-MP computer systems. To be published, 1991.

[BS76] F. Baskett, and A. J. Smith. Interference in multiprocessor computer systems with interleaved memories. *Communications of the ACM*, **19**, 327–334, 1976.

[Cal88a] D. A. Calahan. An analysis of vector startup access delays. *IEEE Transaction on Computers*, **37**, 1134–1137, 1988.

[Cal88b] D. A. Calahan. Characterization of memory conflict loading on the CRAY–2. In *Proceedings 1988 International Conference on Parallel Processing*, 299–303, 1988.

[CB88] D. A. Calahan, and D. H. Bailey. Measurement and analysis of memory conflicts on vector multiprocessors. In J. L. Martin, ed., *Performance Evaluation of Supercomputers*, 83–106. Elsevier Science Publishers B. V., 1988.

[CE85] D. A. Calahan, and K. Elliott. Memory conflict simulation of a many-processor CRAY architecture. Part I: A CRAY X-MP study. Technical Report, Los Alamos Nat. Lab. and Air Force Off. of Sci. Res., 1985.

[CKL77] D.Y. Chang, D. J. Kuck, and D.H. Lawrie. On the effective bandwidth of parallel memories. *IEEE Transaction on Computers*, **26**, 480–489, 1977.

[Cra86] Cray Research, Inc. *CRAY X-MP Computer Systems: Four-Processor Mainframe Reference Manual*, HR–0097, 1986.

[Cra87] Cray Research, Inc. *CRAY X-MP Computer Systems Functional Description Manual*, HR–3005, 1987.

[Cra91a] Cray Research, Inc. *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0, 1991.

[Cra91b] Cray Research, Inc. *UNICOS System Administration for Source Releases, Volume 2*, SG–2113 6.0, 1991.

[CS85] H. Cheung, and J. E. Smith. An analysis of the CRAY X-MP memory system. In *Proceedings 1985 International Conference on Parallel Processing*, 499–505, 1985.

[CS86] H. Cheung, and J. E. Smith. A simulation study of the CRAY X-MP memory system. *IEEE Transaction on Computers*, **35**, 613–622, 1986.

[Det89] U. Detert. Memory performance of CRAY X-MP and CRAY Y-MP. In *Proceedings 24th Semi-Annual Cray User Group Meeting, Trondheim*, 1989.

[DH91] U. Detert, and G. Hofemann. CRAY X-MP and Y-MP memory performance. *Parallel Computing*, **17**, 579–590, 1991.

[For91] Forschungszentrum Jülich. *Datenkommunikation in der KFA - Technische Informationen*, KFA–ZAM–BHB–0100, 1991.

[HB84] K. Hwang, and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.

[HB91] J. Helin, and R. Berrendorf. Analyzing the performance of MIMD message passing hypercubes: A study with the Intel iPSC/860. In *Proceedings 1991 International Conference on Supercomputing*, 1991.

[Hel91] J. Helin. Performance analysis of the CM–2, a massively parallel SIMD computer. Interner Bericht KFA–ZAM–IB–9114, Forschungszentrum Jülich, 1991.

[HH91] J. F. Hake, and W. Homberg. The impact of memory organization on the performance of matrix calculations. *Parallel Computing*, **17**, 311–327, 1991.

[HJ88] R. W. Hockney, and C. R. Jesshope. *Parallel Computers 2*. Adam Hilger, 1988.

[HKN89] F. Hoßfeld, R. Knecht, and W. E. Nagel. Multitasking: Experiences with applications on a CRAY X-MP. *Parallel Computing*, **12**, 259–283, 1989.

[Hof90] G. Hofemann. Vergleich der Speicherarchitekturen der Mehrprozessor-Vektorrechner CRAY X-MP und CRAY Y-MP anhand vektorisierender Algorithmen. Interner Bericht Jül–Spez–555, Forschungszentrum Jülich, 1990.

[Hoo77] C.H. Hoogendoorn. A general model for memory interference in multiprocessors. *IEEE Transaction on Computers*, **26**, 998–1005, 1977.

[Hoß91] F. Hoßfeld. "Grand Challenges" - wie weit tragen die Antworten des Supercomputing? Interner Bericht KFA–ZAM–IB–9117, Forschungszentrum Jülich, 1991.

[HT91] S. Hoefler-Thierfeldt. Zentrale Großrechnersysteme und Datenkommunikationsnetze der KFA: Hardware und Software. Interner Bericht KFA–ZAM–TKI–0131, Forschungszentrum Jülich, 1991.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, Inc., 1991.

[Kog81] P.M. Kogge. *The Architecture of Pipelinded Computers.* McGraw-Hill, 1981.

[MGN79] G. M. Masson, G. C. Gingher, and S. Nakamura. A sampler of circuit switching networks. *IEEE Computer*, **12**, No. 6, 32–48, 1979.

[MT89] H. M. Mendez, and P. Tang. Memory conflicts and machine performance. In *Proceedings of Supercomputing 89*, 826–831, 1989.

[Nag90a] W. E. Nagel. Parallelism on CRAY multiprocessor systems: Concepts of multitasking. In *Proceedings of the Ninth International Conference on Computing Methods in Applied Sciences and Engineering*, 393–412, 1990.

[Nag90b] W. E. Nagel. Prinzipien der Parallelverarbeitung auf Rechnern mit gemeinsamem Speicher. In A. Reuter (Hrsg.), *Proceedings der 20. GI-Jahrestagung 1990, Informatik-Fachbericht 257*, Springer Verlag, 1990.

[Oed85] W. Oed. Untersuchungen von Zugriffen auf den Arbeitsspeicher in Vektorrechnersystemen. Interner Bericht Jül–1994, Forschungszentrum Jülich, 1985.

[OL85] W. Oed, and O. Lange. On the effective bandwidth of interleaved memories in vector processor systems. *IEEE Transaction on Computers*, **34**, 949–957, 1985.

[OL86] W. Oed, and O. Lange. Modelling, measurement, and simulation of memory interference in the CRAY X-MP. *Parallel Computing*, **3**, 343–358, 1986.

[Rau79a] B. R. Rau. Interleaved memory bandwidth in a model of a multiprocessor computer system. *IEEE Transaction on Computers*, **28**, 678–681, 1979.

[Rau79b] B. R. Rau. Program behavior and the performance of interleaved memories. *IEEE Transaction on Computers*, **28**, 191–199, 1979.

[Rav72] C. V. Ravi. On the bandwidth and interference in interleaved memories. *IEEE Transaction on Computers*, **21**, 899–901, 1972.

[Reg89] H. Reger. Ein Vergleich der Multitasking-Implementierungen auf CRAY X-MP und IBM 3090. Interner Bericht Jül–Spez–542, Forschungszentrum Jülich, 1989.

[Rei88a] S. Reinhardt. A blueprint for the UNICOS operating system. *Cray Channels*, **10**, No. 3, 20–24, 1988.

[Rei88b] S. Reinhardt. Two parallel processing aspects of the CRAY Y-MP computer system. In *Proceedings of the 1988 International Conference on Parallel Processing*, 311–314, 1988.

[Rei90] M. H. Reilly. *A Performance Monitor for Parallel Programs.* Academic Press, Inc., 1990.

[RR89] K. A. Robbins, and S. Robbins. *The CRAY X-MP/Model 24, Lecture Notes in Computer Science, Vol. 374.* Springer–Verlag, 1989.

[Sie79] H. J. Siegel. Interconnection networks for SIMD machines. *IEEE Computer*, **12**, No. 6, 57–65, 1979.

[SKB89] M. Simmons, R. Koskela, and I. Bucher, eds. *Instrumentation for Future Parallel Computing Systems.* Addison-Wesley Publishing Company, 1989.

[Smi82] A. J. Smith. Cache memories. *ACM Computing Surveys*, **14**, 473–530, 1982.

[ST91] J. E. Smith, and W. R. Taylor. Accurate modelling of interconnection networks in vector supercomputers. In *Proceedings of the 1991 International Conference on Supercomputing*, 264–273, 1991.

[SW91] M. L. Simmons, and H. J. Wasserman. A preliminary look at a new Cray supercomputer: Performance evaluation of the "C90". *Vector Register*, **3**, No. 3, 3–7, 1991.

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

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich anfertigen zu können. Die hier erfahrene Unterstützung war wichtige Voraussetzung für das Erstellen dieser Arbeit. Mein besonderer Dank gilt Herrn W. Homberg für die fruchtbaren Diskussionen und Herrn W. Gürich für die Betreuung. Den Herren J.-Fr. Hake und W.E. Nagel möchte ich für ihr Interesse und ihre Anregungen danken.

FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Jül-2619
Mai 1992
ISSN 0366-0885