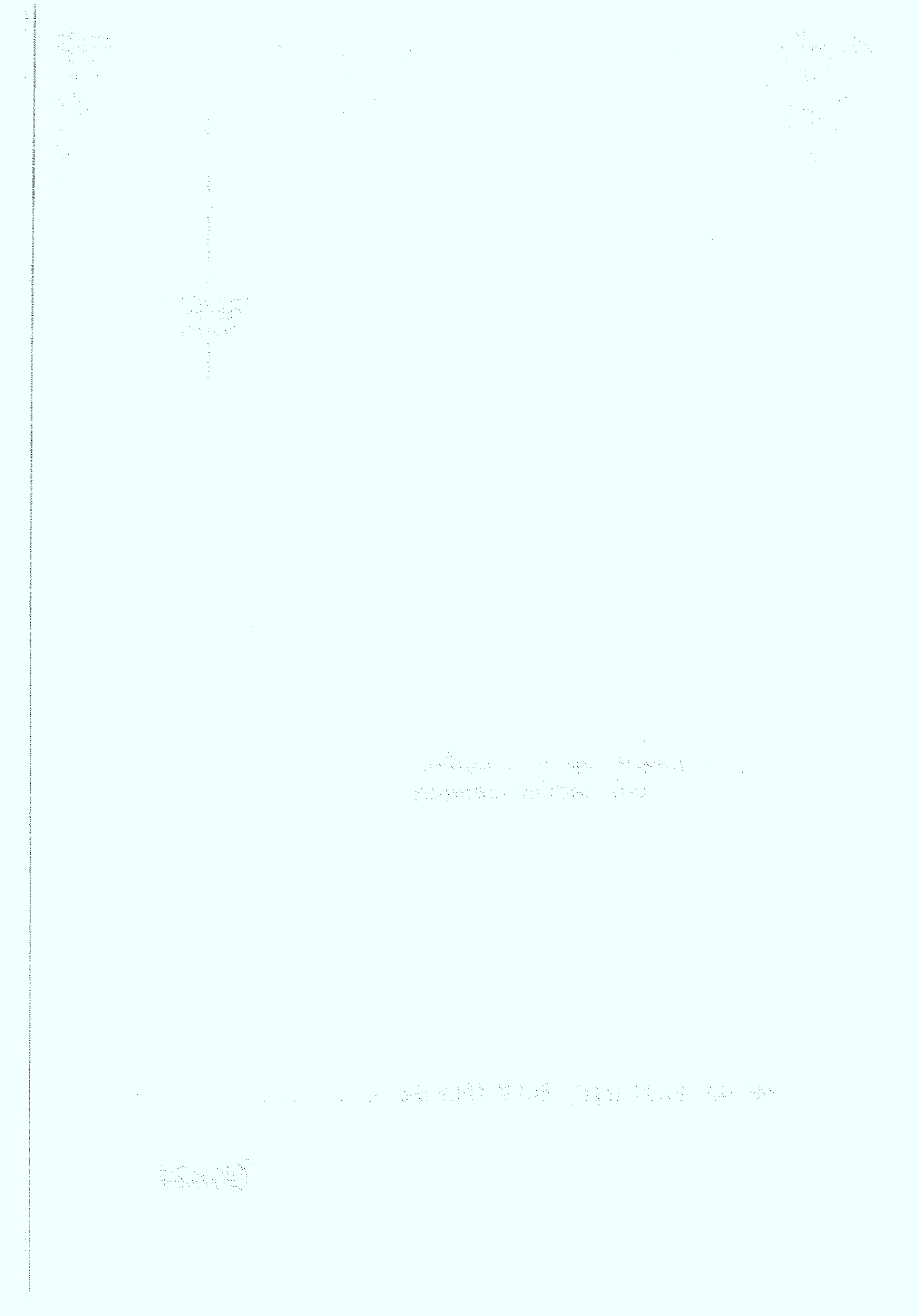
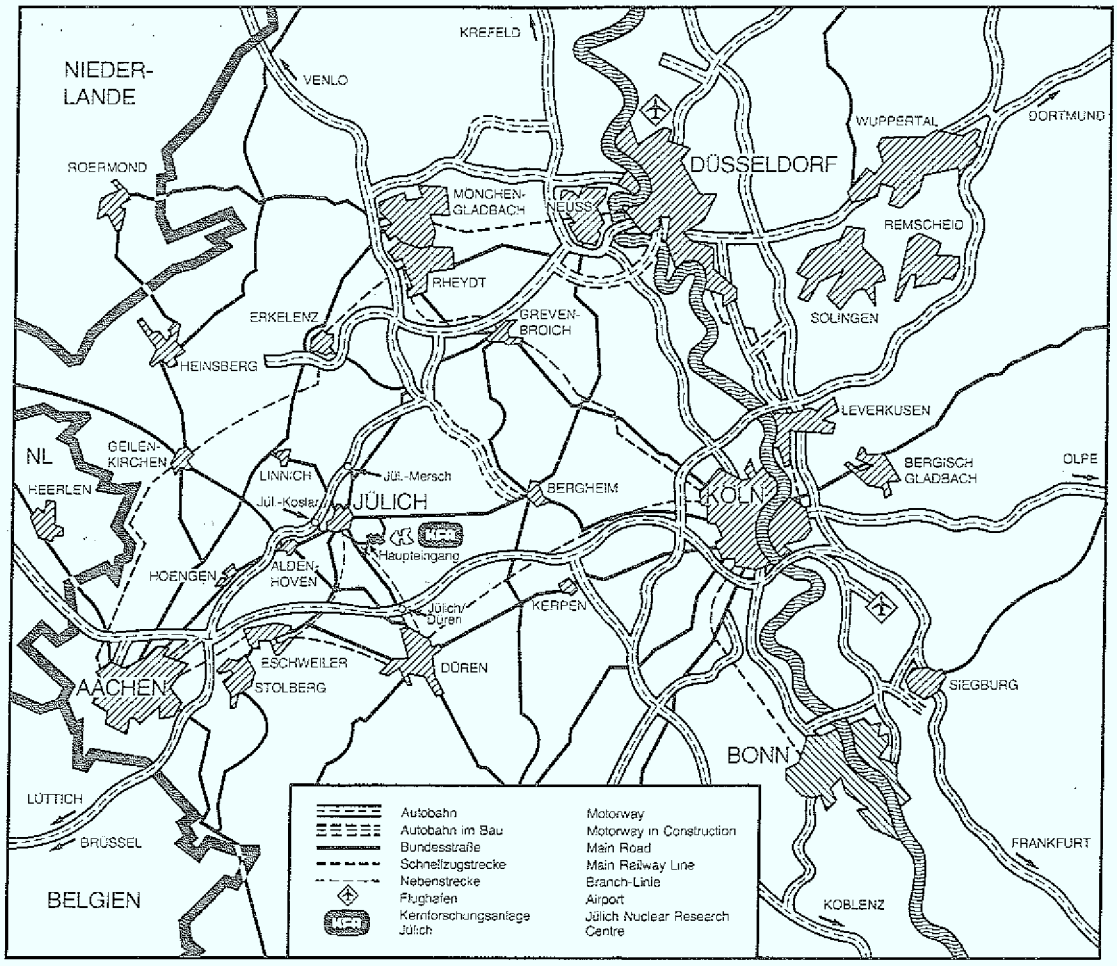
KERNFORSCHUNGSANLAGE JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik

Untersuchungen von Zugriffen auf den Arbeitsspeicher in Vektorrechnersystemen

von
W. Oed

Jül-1994
Juni 1985
ISSN 0366-0885





Als Manuskript gedruckt

Berichte der Kernforschungsanlage Jülich – Nr. 1994
Zentralinstitut für Angewandte Mathematik Jül-1994

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH
Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)
Telefon: 02461/610 · Telex: 833556-0 kf d

Untersuchungen von Zugriffen auf den Arbeitsspeicher in Vektorrechnersystemen

von
W. Oed

D 82 (Diss. T.H. Aachen)

Diese Arbeit ist aus vielfältigen Untersuchungen im Umfeld von Vektorrechnersystemen hervorgegangen. Die Anregung und Ermutigung dazu verdanke ich dem Direktor des Zentralinstituts für Angewandte Mathematik der Kernforschungsanlage Jülich GmbH, Herrn Dr. F. Hoßfeld. Er ermöglichte mir die Durchführung der Arbeit während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an diesem Institut.

Herrn Prof. O.Lange gilt mein besonderer Dank für die verständnisvolle und aufopfernde Betreuung. Zahlreiche Diskussionen und gemeinsame Aktivitäten auf dem Gebiet der Paralleldatenverarbeitung haben mir wichtige Impulse für diese Arbeit gegeben.

Herrn Prof. D.Haupt danke ich für das wohlwollende Interesse, das er dieser Arbeit entgegengebracht hat.

Das gute Arbeitsklima im Zentralinstitut für Angewandte Mathematik und die stete Hilfsbereitschaft meiner Kollegen haben wesentlich zum Gelingen dieser Arbeit beigetragen. Besonderer Dank gebühren U.Detert, W.Gürich, W.Meyer und W.Nagel sowie Herrn Prof. C.D.Feustel von der Virginia Polytechnic Institute and State University, Blacksburg Va., U.S.A., der zur Zeit als Gastwissenschaftler am Zentralinstitut für Angewandte Mathematik tätig ist.

Nicht zuletzt möchte ich mich noch bei meiner Frau Ingrid für ihre moralische Unterstützung sowie die Erstellung der Plot-Programme zur Darstellung der Meß- und Simulationsergebnisse bedanken.

Kurzfassung

Vektorrechner stellen die derzeit leistungsfähigsten Rechnersysteme dar und werden hauptsächlich für technisch-wissenschaftliche Anwendungen eingesetzt. Die Leistungsfähigkeit der als Pipelines ausgelegten Funktionseinheiten in Vektorrechnern kann allerdings nur dann effizient genutzt werden, falls es gelingt, den notwendigen Datentransfer zwischen dem Arbeitsspeicher und der CPU, bzw. den CPUs bei Mehrprozessor-Systemen, optimal zu bewerkstelligen.

In dieser Arbeit werden Untersuchungen von Zugriffen auf den Arbeitsspeicher von Vektorrechnersystemen (sowohl Ein-Prozessorsysteme als auch Mehrprozessorsysteme mit gemeinsamem Arbeitsspeicher) mit Hilfe analytischer Modelle, Messungen auf einem Doppelprozessorsystem CRAY X-MP sowie Simulationen durchgeführt. Der Schwerpunkt der Untersuchungen liegt dabei auf den in Vektorrechnern auftretenden, meist längeren Folgen von Zugriffen auf Speicherzellen, deren Adressen sich jeweils um einen konstanten Abstand voneinander unterscheiden.

Abstract

Presently, the highest performance computer systems are the vector processors which are mainly employed for technical and scientific applications. However, the performance potential of the pipelined functional units of a vector processor can only efficiently be used if the required data transfer between main memory and the CPU, respectively the CPUs in case of multiprocessor systems, can optimally be routed.

This research investigates the access to main memory of a vector processor system (single processor system as well as multiprocessor systems with shared memory) with the aid of analytic models, measurements on a double processor system CRAY X-MP and simulations. The emphasis of the investigations is put on long sequences of accesses to memory cells whose addresses differ from each other by a constant distance as they typically appear in vector processors.

INHALTSVERZEICHNIS

1 EINLEITUNG

In dem vielfältigen Anwendungsspektrum heutiger Rechner zeichnet sich das Gebiet der technisch-wissenschaftlichen Datenverarbeitung vor allem durch einen enormen Bedarf an Rechenleistung aus [HOß 84]. Seit der Pionierzeit der elektronischen Rechenanlagen in den vierziger und fünfziger Jahren ist die Leistungsfähigkeit solcher Maschinen hauptsächlich durch technologische Fortschritte um mehrere Größenordnungen gestiegen. Wenn auch die technologische Entwicklung noch lange nicht abgeschlossen ist, so sind, von den problematischen Fertigungsmöglichkeiten hoch integrierter Technologien einmal ganz abgesehen, die physikalischen Grenzen nicht mehr allzu fern [FuB 77].

Weitere Steigerungen der Rechenleistung werden daher durch neuartige Rechnerarchitekturen angestrebt, die eine Abkehr bzw. Erweiterung von der konventionellen von-Neumann-Architektur darstellen. Der magische Begriff heißt hierbei *Parallelität*, d.h. Vervielfältigung von Betriebsmitteln, vor allem von Prozessoren und Speichereinheiten, mit deren Hilfe möglichst gleichzeitig, also *parallel*, ein Programm bearbeitet werden soll. Dabei werden die unterschiedlichsten Konzepte verfolgt, von Systemen, die, aus sehr vielen preiswerten Einzelprozessoren bestehend, eine hohe Leistungsfähigkeit besitzen (z.B. C.mmp [WUL 74], ICL-DAP [HuJ 81], NYU-Ultracomputer [Gua 82]), bis hin zu den *Supercomputern*, deren erster der berühmte ILLIAC IV [Bua 68] darstellte, mit denen die Leistungsgrenze, vor allem der für das *Number Crunching* benötigten Gleitkommaoperationen, immer weiter vorgeschoben werden soll.

Im Verbund mit schnellster Technologie sind seit etwa Mitte der siebziger Jahre Maschinen verfügbar, deren Rechenleistung um annähernd zwei Größenordnungen über der der bislang schnellsten konventionellen Rechner liegt, was vor allem auf die Auslegung von Gleitkommafunktionseinheiten als *Pipelines* [KOG 81] zurückzuführen ist. Aufgrund ihrer Architektur verarbeiten diese Maschinen insbesondere solche Gleitkommaoperationen effizient, die in Vektor- und Matrixoperationen vorkommen, und werden daher meist als *Vektorrechner* bezeichnet (z.B. CRAY-1 [RUS 78], CYBER 205 [FuM 82], Fujitsu VP-100/VP-200 [MuU 84]).

Einen weiteren Schritt hin zu schnelleren Maschinen stellen *Mehrprozessorsysteme* dar (z.B. CRAY X-MP Serie [CRA 84b]), deren einzelne Prozessoren (*CPUs*) wiederum extrem schnelle Vektorrechner sind, die einerseits

jeweils für sich ein eigenständiges Programm bearbeiten können (Erhöhung des Durchsatzes), andererseits die *parallele* Ausführung eines in mehrere *Tasks* zerlegten Programms ermöglichen (Verkürzung der Verweilzeit).

Mit der Verfügbarkeit solcher Supercomputer wird allerdings auch vielfach deutlich, daß ohne geeignete, an die Architektur angepaßte Algorithmen das Leistungspotential dieser Rechner oft nur zu einem Bruchteil genutzt werden kann. Es werden daher vielfältige Anstrengungen unternommen, die Möglichkeiten solcher Maschinen besser zu nutzen, sei es durch optimierende Compiler [KUC 82], Spracherweiterungen [JES 82], Programmumstellungen oder Entwicklung anderer Algorithmen [HOß 83], wobei sich vor allem letzteres als besonders schwierig erweist.

Die Leistungsfähigkeit der Prozessoren bzw. Funktionseinheiten in Parallel- und Vektorrechnern kann allerdings nur dann effizient genutzt werden, falls es auch gelingt, den notwendigen Datentransfer zwischen den Registern der CPU und dem gemeinsamen Arbeitsspeicher optimal zu bewerkstelligen. Damit mehrere Prozessoren auch tatsächlich parallel arbeiten können, muß ein paralleler Zugriff auf jeweils benötigte Daten möglich sein, was durch die Auslegung des Speichers in mehrere sogenannte *Bänke* erreicht wird, die über ein geeignetes *Verbindungsnetzwerk* [BuH 83] mit den einzelnen Prozessoren verbunden werden können (s. Abb.1.1).

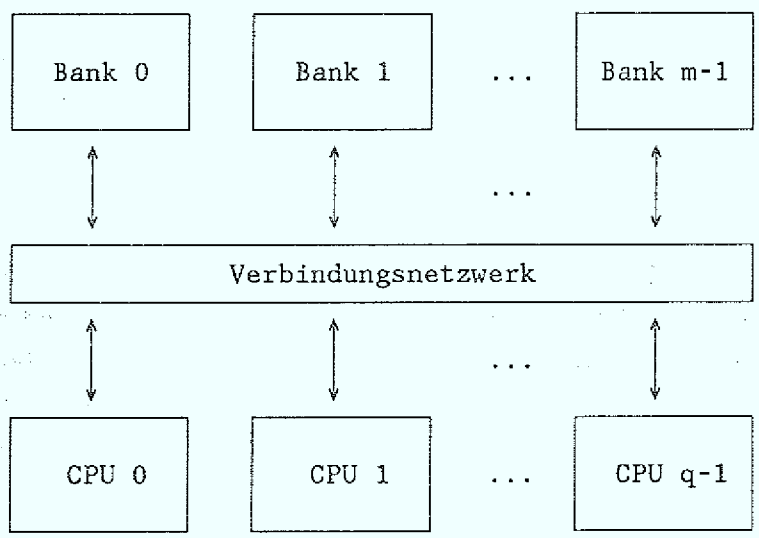


Abb.1.1 Zugriff von *q* CPUs über ein Verbindungsnetzwerk auf *m* Bänke

Dennoch kann nicht grundsätzlich davon ausgegangen werden, daß der jeweils gewünschte parallele Zugriff auf Daten im Arbeitsspeicher auch stets möglich ist, falls sich beispielsweise mehrere der benötigten Daten

innerhalb der gleichen Bank befinden oder Konflikte auf den Pfaden des Verbindungsnetzwerks entstehen. Solche *Zugriffskonflikte* müssen entweder von vornherein ausgeschlossen werden, indem nur solche Zugriffe zugelassen werden, die konfliktfrei durchgeführt werden können, oder *dynamisch* dadurch aufgelöst werden, daß ein erwünschter, jedoch nicht durchführbarer Zugriff verzögert und erst zu einem späteren Zeitpunkt zugelassen wird. In beiden Fällen ziehen Verzögerungen im Speicher entsprechende Verzögerungen bei der Ausführung der Operationen in den Rechenwerken nach sich.

In der vorliegenden Arbeit werden Zugriffe auf den Speicher eines Vektorrechners untersucht, soweit diese Zugriffe von der CPU bzw. bei Mehrprozessorsystemen von den verschiedenen CPUs aus erfolgen. Es ist dabei von Interesse, wie die verschiedenen Einflußgrößen zusammenwirken, wodurch Konflikte entstehen können, welche Konstellationen besonders ungünstig sind und wie diese vermieden werden können.

Der Schwerpunkt der Untersuchungen liegt dabei auf den in Vektorrechnern auftretenden, meist längeren Folgen von Zugriffen auf Speicherzellen, deren Adressen sich jeweils um einen konstanten Abstand voneinander unterscheiden. Eine solche Folge wird auch als *Zugriffsstrom* bezeichnet, da im optimalen Fall des konfliktfreien Zugriffs auf die Speicherzellen mit jedem Takt eine Date pro Folge aus dem Speicher heraus (*Lesen*) bzw. in den Speicher hinein (*Schreiben*) transportiert werden kann.

Der E/A-Verkehr, der von separaten E/A-Subsystemen oder E/A-Kanälen aus ebenfalls auf den Speicher zugreift, wird in die Untersuchungen nicht mit eingeschlossen, da er in der Regel nach anderen, durch den Programmierer kaum beeinflußbaren Prinzipien abläuft.

Die Untersuchungen selbst werden mit Hilfe von analytischen Modellen, Simulationen und Messungen durchgeführt.

Im einzelnen hat die Arbeit folgenden Aufbau:

Das zweite Kapitel vermittelt einen Überblick über die Architektur und die Programmierung von Vektorrechnern.

Im dritten Kapitel werden die für die Untersuchungen dieser Arbeit benötigten Begriffe definiert und erläutert. Weiterhin werden einige Arbeiten vorgestellt, die sich bislang mit Speicherzugriffen auf Arbeitsspeicher in Parallel- und Vektorrechnern beschäftigt haben.

Im vierten Kapitel werden Speicherzugriffe in Vektorrechnern mit Hilfe analytischer Modelle behandelt. Die Modellierung geht zunächst im einfachsten Fall von einem Zugriffsstrom aus. Danach wird das Verhalten zweier miteinander konkurrierender Zugriffsströme analysiert. Schließlich wird ein Modell für den allgemeinen Fall mehrerer unabhängiger Zugriffsströme entwickelt, mit dessen Hilfe sich die effektive Speicherbandbreite berechnen läßt.

Im fünften Kapitel wird das Speicherverhalten eines Vektorrechners des Typs CRAY X-MP mit Hilfe von Messungen und Simulationen analysiert und mit den Ergebnissen der analytischen Modelle aus Kapitel 4 verglichen. Die Messungen sind hierbei sämtlich auf dem im *Zentralinstitut für Angewandte Mathematik der Kernforschungsanlage Jülich GmbH* installierten Doppelprozessorsystem CRAY X-MP (2MWorte Speicher, 16 Bänke) durchgeführt worden. Aufgrund der Resultate werden Hinweise für den Programmierer gegeben, wie Datenbereiche angelegt werden sollen, um die Anzahl der Speicherzugriffskonflikte möglichst gering zu halten. Schließlich werden mit Hilfe des Simulators die Auswirkungen der Zusammenschaltung mehrerer, auf einen gemeinsamen Arbeitsspeicher zugreifender CPUs, eines anderen Aufbaus des Speichers, unterschiedlicher Prioritätenregelungen sowie des Einsatzes von anderen Speicherelementen untersucht.

Im sechsten Kapitel schließlich werden die Ergebnisse dieser Untersuchungen zusammengefaßt und bewertet.

2 ARCHITEKTUR UND PROGRAMMIERUNG EINES VEKTORRECHNERS

Dieses Kapitel stellt die Grundprinzipien der Architektur und Programmierung eines Vektorrechners dar, soweit sie im Rahmen dieser Arbeit erforderlich sind; ausführliche Darstellungen sind beispielsweise in [KOG 81], [HuJ 81] oder [WIN 82] zu finden.

Während im klassischen von-Neumann-Rechner sämtliche Operationen in einer arithmetisch-logischen Einheit (ALU) ausgeführt werden, verfügt ein Vektorrechner über mehrere, sogenannte *Funktionseinheiten* (*functional units*), die jeweils eine spezielle Operation ausführen können. Ein wesentliches Merkmal eines Vektorrechners ist die Auslegung dieser Funktionseinheiten als *Pipelines* [KOG 81], was entscheidend zu der hohen Leistungsfähigkeit eines Vektorrechners beiträgt.

Prinzipiell besteht die Architektur der CPU eines Vektorrechners neben dem Arbeitsspeicher aus folgenden vier Teilen:

- dem *Instruktionsteil*, der den Instruktionspuffer, den Programmzähler sowie den Instruktionsdekodierer enthält;

- dem *Adreßteil*, der über Adreßregister und Funktionseinheiten zur Berechnung von Operandenadressen verfügt;

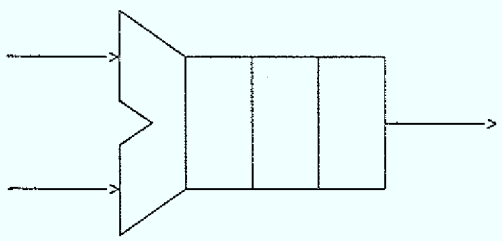
- dem *Skalarteil*, der über Register und Funktionseinheiten für sogenannte *Skalaroperationen* verfügt, wobei unter einer Skalaroperation eine Operation auf *einem* Operanden bzw. Operandenpaar verstanden wird;

- dem *Vektorteil*, der über Register und Funktionseinheiten für sogenannte *Vektoroperationen* verfügt, wobei unter einer Vektoroperation eine Operation auf n Operanden bzw. Operandenpaaren verstanden wird; n wird die *Vektorlänge* genannt. Eine Vektoroperation wird dabei durch eine einzige *Vektorinstruktion* aus dem Instruktionsteil heraus initiiert.

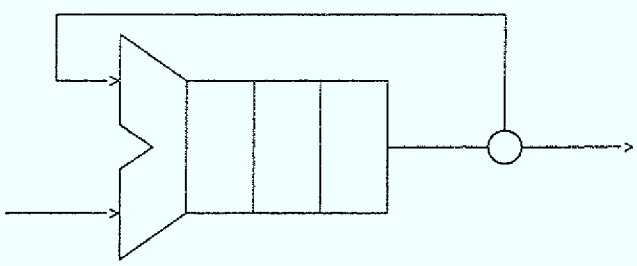
Vielfach ist die Trennung nicht ganz so strikt, da beispielsweise bestimmte Funktionseinheiten sowohl von Skalaroperationen als auch von Vektoroperationen verwendet werden. Logisch gesehen ist aufgrund entsprechender Instruktionen eine Einteilung dieser Art jedoch sinnvoll, zumal

sich erhebliche Unterschiede in der Ausführungszeit zwischen Skalar- und Vektoroperationen ergeben, selbst wenn die gleiche Anzahl an Operationen in der gleichen Funktionseinheit durchgeführt wird.

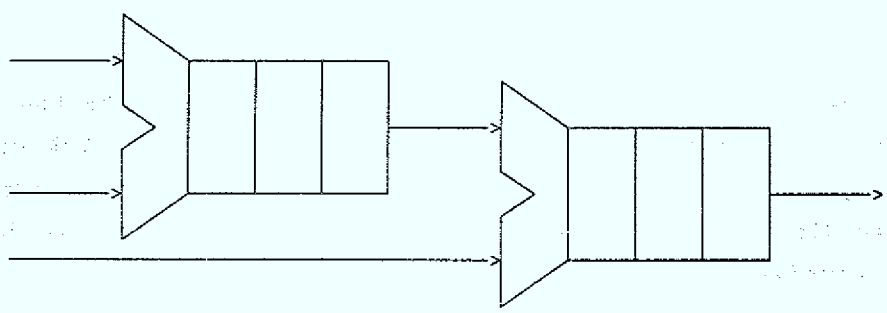
Neben dem Pipelining trägt die Möglichkeit der *parallelen* Arbeitsweise der einzelnen Funktionseinheiten zusätzlich zur Leistungsfähigkeit eines Vektorrechners bei. Beispielsweise können während der Ausführung einer Vektoroperation auf n Operandenpaaren bereits die gesamte Adreßmanipulation für die nächste Operation sowie Skalaroperationen gleichzeitig durchgeführt werden.



a) normaler Betrieb einer Pipeline



b) Rückkopplung des Ausgangs in einen Eingang der Pipeline



c) Verkettung zweier Pipelines

Abb.2.1 Betriebsarten von Pipelines

Besondere Formen des Betriebs von Pipelines stellen die *Rückkopplung (feedback)* des Ausgangs der Pipeline in einen Eingang sowie die *Verkettung (chaining)* mehrerer Pipelines dar. Beide Betriebsarten haben den Vorteil, Zwischenresultate ohne irgendwelche Umwege direkt zur nächsten Bearbeitung weiterzuleiten. Im Falle der Verkettung führt die parallele Arbeitsweise der Funktionseinheiten dazu, daß am Ausgang der letzten Pipeline der Kette pro Takt ein Resultat eines komplexen Ausdrucks erzielt wird. In Abb.2.1 sind die drei prinzipiellen Betriebsarten von Pipelines veranschaulicht.

Weitere Leistungsverbesserungen sind durch Mehrprozessorsysteme gegeben, wie beispielsweise den symmetrischen Mehrprozessorsystemen der CRAY X-MP Serie [CRA 84b], die, aus zwei oder vier *identischen* Vektorprozessoren bestehend, auf einem gemeinsamen Speicher unabhängig voneinander arbeiten. Einige gemeinsame Register (*shared registers*) dienen der Kommunikation zwischen den Prozessoren und ermöglichen somit die Bearbeitung eines Programms durch alle Prozessoren im sogenannten *Multitasking-Modus* [AuS 83, CRA 84a]. Die prinzipielle Architektur für ein Doppelprozessorsystem (z.B. CRAY X-MP) ist in Abb.2.2 wiedergegeben.

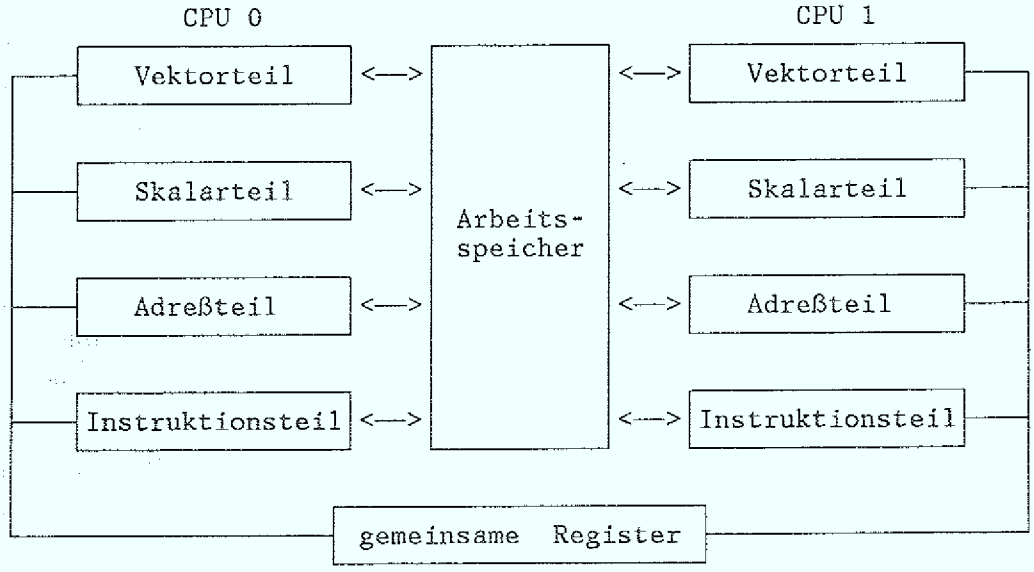


Abb.2.2 Prinzipielle Architektur eines Doppelprozessor-Vektorrechners

Das Prinzip des Pipelining verleiht den Vektorrechnern die Möglichkeit, Operationen auf Vektoren schneller als in konventionellen Rechnern auszuführen. Gleichzeitig müssen jedoch auch hohe Anforderungen an den Arbeitsspeicher der Maschine gestellt werden, damit die Operanden den

Funktionseinheiten entsprechend schnell zur Verfügung gestellt werden können.

Die Forderung, benötigte Daten stets im richtigen Zeitpunkt an der entsprechenden Stelle zur Verfügung zu haben, erweist sich aus Kosten- und Komplexitätsgründen in der Regel jedoch als unrealistisch. Deswegen wird ein akzeptabler Kompromiß zwischen Aufwand und Verfügbarkeit der Daten angestrebt, der in Vektorrechnern im wesentlichen durch die Verwendung folgender drei Speicherebenen realisiert wird:

- schnelle *Datenregister*, die innerhalb eines Taktes gelesen bzw. beschrieben werden können und den jeweiligen Einheiten zugeordnet sind;

- ein schnell zugreifbarer Instruktionsspeicher, - *Cache* oder auch *Instruktionspuffer* genannt -, der es ermöglicht, eine Instruktion innerhalb eines Taktes zu initiieren, und der dem Instruktionsteil zugeordnet ist;

- der *Arbeitsspeicher*, in dem die Programme mit ihren zugehörigen Datenbereichen abgelegt sind.

Es liegt auf der Hand, daß für ein Vektorrechnersystem, das über extrem schnelle Funktionseinheiten verfügt, ein möglichst reibungsloser Datenaustausch zwischen den Registern und dem Arbeitsspeicher gewährleistet sein muß, um nicht an dieser Stelle den möglichen Gewinn bei der Ausführungszeit bereits zu verspielen.

Die Programmierung eines Vektorrechners ist ebenso wie bei herkömmlichen Rechnern einerseits durch Maschinensprache andererseits durch höhere Programmiersprachen möglich. Unter den höheren Programmiersprachen ist FORTRAN aufgrund seiner starken Anwendung im technisch-wissenschaftlichen Bereich am weitesten verbreitet, und die zugehörigen Compiler sind am besten entwickelt, insbesondere was die automatische *Vektorisierung* anbetrifft.

Unter Vektorisierung wird dabei die Möglichkeit verstanden, im Programm vorhandene Vektorstrukturen in Vektorinstruktionen umzusetzen. FORTRAN kennt allerdings bislang keine Notation zur Beschreibung von Vektor- oder Matrixoperationen, wie sie in der linearen Algebra häufig vorkommen und bei der Lösung auf Rechenanlagen sehr viel Zeit in Anspruch nehmen. Daher wurden von manchen Herstellern (z.B. CDC [FuM 82]) FORTRAN-Spracherweiterungen eingeführt, andere Hersteller (z.B. CRAY [CRA 83]) führen eine

Vektorisierung auf der Ebene der *innersten Schleife* (*innermost DO-loop*) durch, sofern dies nicht aufgrund bestimmter Datenabhängigkeiten unmöglich ist [NAG 84].

Beispielsweise wird die komponentenweise Addition zweier Vektoren *a* und *b* der Länge *n* zu einem Vektor *c* in FORTRAN in einer Schleife der Form

```
      DO 1 I = 1,N
    1 C(I) = A(I) + B(I)
```

programmiert. Ein vektorisierender Compiler erzeugt daraus, abgesehen von notwendigen Adreßmanipulationen, typischerweise folgende vier Vektorinstruktionen, die jeweils für die Vektorlänge *n* initiiert werden:

```
lade A
lade B
C ← A + B
speichere C
```

Initiierung bedeutet hierbei ein Anstoßen jeweils einer Operation innerhalb eines Taktes. Danach bedarf es keiner weiterer Aktionen seitens des Instruktionsteils, um die vollständige Operation auf allen *n* Komponenten durchzuführen.

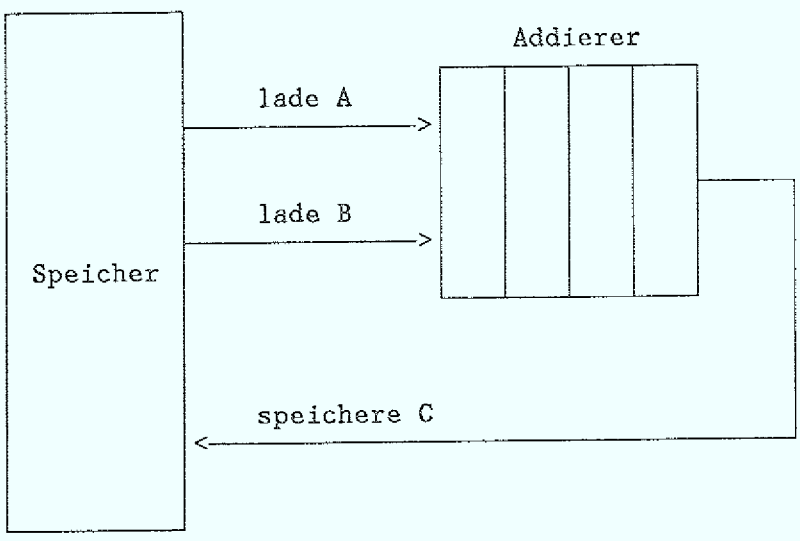


Abb.2.3 Datenströme bei einer Vektoraddition

Die höchste Leistungsfähigkeit erreicht ein Vektorrechner dann, wenn es gelingt, die benötigten Operanden aus dem Speicher ohne Unterbrechungen

zu laden, durch die jeweiligen Funktionseinheiten (in diesem Beispiel den Addierer) hindurchzuschicken und die Ergebnisse ohne Unterbrechung in den Speicher zu transportieren. In Abb.2.3 ist der Verlauf der *Datenströme* prinzipiell dargestellt; der Addierer ist als vierstufige Pipeline ausgelegt.

Die für eine Vektorinstruktion jeweils benötigten Rechenwerke und Register werden für die gesamte Ausführungsdauer dieser Instruktion reserviert. Andere Vektorinstruktionen, die irgendein reserviertes Betriebsmittel benötigen, können nicht initiiert werden.

Vielfach entstehen bei den Speicherzugriffen jedoch Konflikte, die aufgelöst werden müssen und zu einer Verlängerung der Ausführungszeit führen. Mit t_o werde die *optimale* Ausführungszeit für eine Operation bezeichnet, d.h. die Zeit, die minimal benötigt wird, falls es gelingt, die jeweiligen Operanden konfliktfrei aus dem Speicher bzw. dorthin zurück zu transportieren. Durch Störungen auftretende *Zeitverzögerungen* werden durch t_v berücksichtigt. Somit ergibt sich die *gesamte* Ausführungszeit t_g zu

$$t_g = t_o + t_v. \qquad (2.1)$$

Welche Faktoren Einfluß auf Speicherzugriffe speziell im Vektormodus haben und welche Voraussetzungen nötig sind, um nach Möglichkeit konfliktfreie Zugriffe aufrechtzuerhalten, ist Gegenstand der Untersuchungen dieser Arbeit.

3 ARBEITSSPEICHER IN VEKTORRECHNERN

In diesem Kapitel werden die speziell im Zusammenhang von Speicherzugriffen in Vektorrechnern benötigten Begriffe eingeführt und erläutert sowie einige bisherige Arbeiten über Speicher in Parallel- und Vektorrechnern vorgestellt.

3.1 GRUNDLEGENDE DEFINITIONEN

Def.3.1: *Speicherzugriffswunsch*

Ein Speicherzugriffswunsch (*memory access request*, im folgenden kurz mit *Zugriffswunsch* bezeichnet) bedeutet, daß eine bestimmte Speicherzelle des realen Adreßraums über ein Adreßregister mit dem Ziel angesprochen werden soll, den Inhalt der angesprochenen Speicherzelle entweder aus dem Speicher in ein Register der CPU zu kopieren (*Lesen* bzw. aus Sicht des Registers *Laden*) oder durch den Inhalt des Registers zu ersetzen (*Schreiben* bzw. aus Sicht des Registers *Speichern*).

Def.3.2: *zugelassener Speicherzugriff*

Ein zugelassener Speicherzugriff (*memory access*, im folgenden kurz mit *zugelassenem Zugriff* bezeichnet) bedeutet, daß ein vorliegender Zugriffswunsch durchgeführt wird.

Def.3.3: *Speicherbank*

Eine Speicherbank (*memory bank*, im folgenden kurz mit *Bank* bezeichnet) ist eine Einheit von Speicherzellen. Jeder Bank ist ein Adreßregister zugeordnet, über das auf eine der Speicherzellen innerhalb dieser Bank zugegriffen werden kann.

Anmerkung:

Von Bänken spricht man nur dann, falls der reale Adreßraum eines Speichers über mehrere, allgemein m Bänke verfügt, die von 0 bis $m-1$ durchnumeriert werden. Die Nummer einer Bank wird deren *Adresse* genannt.

Im folgenden interessieren ausschließlich die Adressen der Bänke, da kein weiterer Zugriff auf irgendeine Speicherzelle innerhalb der Bankzykluszeit (s. Def.3.6) zugelassen wird.

Def.3.4: *m-fach verschränkter Speicher*

Ein aus *m* Bänken bestehender Speicher heißt *m*-fach verschränkt (*m-way interleaved*), falls die Adressen des realen Adreßraums zyklisch auf die *m* Adressen der Bänke verteilt sind, d.h. $i = (j \bmod m)$, mit $j=0,1,2,...$ und *i* der Adresse der Bank [HEL 67].

Beispiel:

In Abb.3.1 ist die Aufteilung eines vierfach verschränkten Speichers mit 4K Adressen dargestellt.

Bank 0	Bank 1	Bank 2	Bank 3
0	1	2	3
4	5	6	7
.	.	.	.
.	.	.	.
.	.	.	.
4088	4089	4090	4091
4092	4093	4094	4095

Abb.3.1 4K Adressen in einem vierfach verschränkten Speicher

Def.3.5: *Sektion*

Eine Sektion (*section* [CRA 82]) ist eine Einheit von Bänken zu der es von jeder CPU aus genau einen Zugriffspfad gibt.

Anmerkungen:

Sektionen werden aus Komplexitäts- bzw. aus Kostengründen eingeführt, da von einer CPU aus statt *m* Zugriffspfade zu jeder Bank des Speichers lediglich *s* benötigt werden. Die Sektionen werden von *0* bis *s-1* durchnumeriert, mit $1 \le s \le m$. Die Nummer einer Sektion wird deren *Adresse* genannt.

Wenn nicht anderweitig angegeben, so wird davon ausgegangen, daß die *m* Bänke zyklisch auf die einzelnen Sektionen verteilt sind, d.h. $i = (j \bmod s)$, mit *j* der Adresse der Bank und *i* der Adresse der Sektion.

In einem Mehrprozessorsystem mit q CPUs existiert also von jeder CPU aus genau ein Zugriffspfad zu jeder Sektion, d.h. es gibt zu jeder Sektion insgesamt q Zugriffspfade.

In Abb.3.2 ist ein vierfach verschränkter Speicher eines Doppelprozessorsystems mit zwei Sektionen und zwei Zugriffspfaden pro CPU zu jeder Sektion dargestellt.

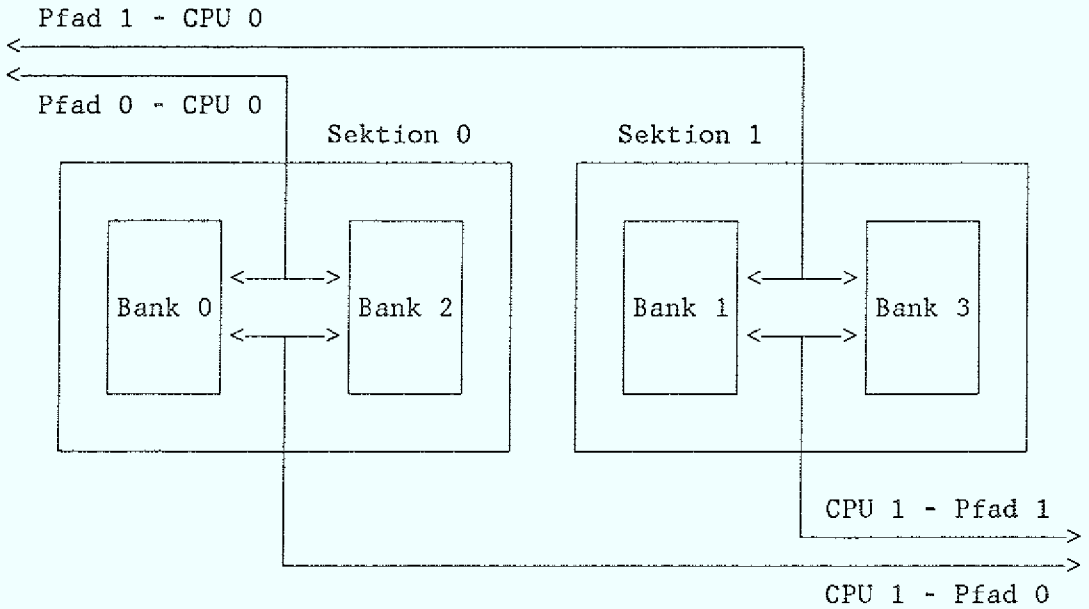


Abb.3.2 zwei Sektionen in einem vierfach verschränkten Speicher mit je zwei Zugriffspfaden pro CPU

Def.3.6: *Bankzykluszeit*

Die Bankzykluszeit $t_c=n_c\tau$ (*bank cycle time*) ist ein ganzzahliges Vielfaches n_c der Taktzeit τ *(clock period)* und bezeichnet die Zeit, die die Bank aufgrund eines zugelassenen Zugriffs auf eine innerhalb dieser Bank liegenden Speicherzelle aktiv ist. Während der Bankzykluszeit wird kein weiterer Zugriff auf eine Speicherzelle, die innerhalb dieser Bank liegt zugelassen, d.h. die Bank ist solange belegt.

Def.3.7: *Speicherzugriffszeit*

Die Speicherzugriffszeit $t_a=n_a\tau$ (*memory access time*) ist ein ganzzahliges Vielfaches n_a der Taktzeit τ und bezeichnet die Zeit, die vergeht, bis der Inhalt einer angesprochenen Speicherzelle im Zielregister zur Verfügung steht, nachdem der Zugriff auf die entsprechende Bank zugelassen worden ist.

Def.3.8: *Speicher-Port*

Ein Speicher-Port (im folgenden kurz mit *Port* bezeichnet) ist die *aktive* Schnittstelle zwischen dem Speicher und den Registern der CPU und veranlaßt den eigentlichen Datentransport.

Anmerkungen:

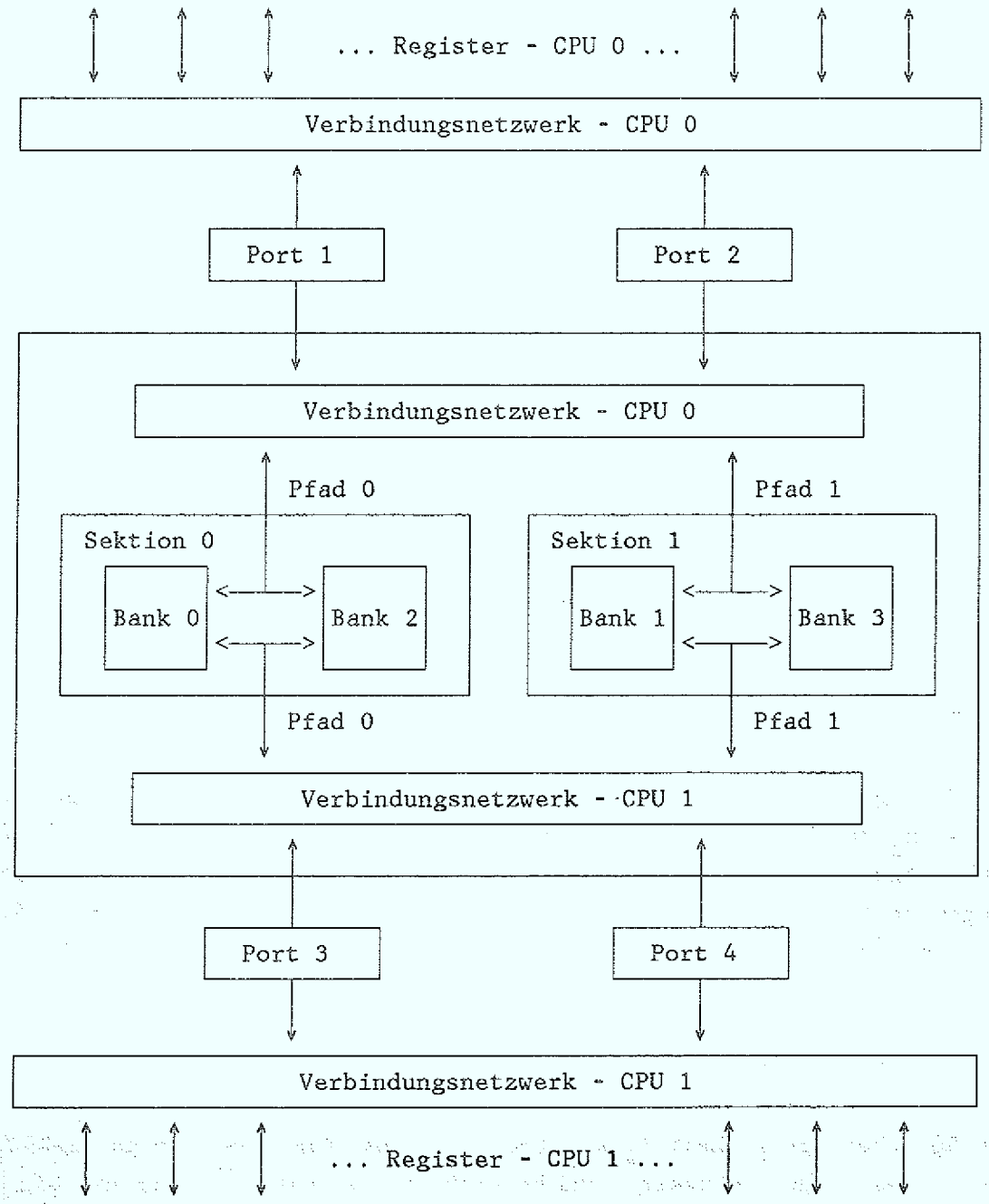


Abb.3.3 vollständiges Speichersystem mit vier Bänken, zwei Sektionen und vier Ports für zwei CPUs

Pro Takt kann maximal ein Datenwort (im folgenden kurz *Date* genannt) über einen Port transportiert werden. Ein Port hat weiterhin die Fähigkeit, einen Zugriffswunsch zu verzögern, falls die angesprochene Bank oder der benötigte Zugriffspfad blockiert ist. Mit p wird die Anzahl der Ports bezeichnet, über die *gleichzeitig* je ein Zugriff auf den Speicher möglich ist. Die Ports werden von *1* bis p durchnumeriert.

In Abb.3.3 ist ein vollständiges Speichersystem mit zwei CPUs und vier Ports (zwei Ports pro CPU) dargestellt. Der Speicher ist in vier Bänke und zwei Sektionen unterteilt. Bei der Initiierung einer Speicheroperation wird eine Verbindung zwischen einem Port und dem entsprechenden Register einer CPU über das dazwischenliegende Verbindungsnetzwerk hergestellt und bleibt bis zum Ende der Speicheroperation bestehen. Das entsprechende Verbindungsnetzwerk im Speicher stellt bei zugelassenem Zugriff eines Ports eine Verbindung zwischen dem betreffenden Port und der Sektion her, innerhalb der die anzusprechende Bank liegt. Diese Verbindung besteht für die Dauer eines Taktes.

Def.3.9: *Speicherbandbreite*

Die Speicherbandbreite b_w (*memory bandwidth*, im folgenden kurz *Bandbreite* genannt) gibt an, wieviele Daten pro Takt maximal zwischen Speicher und CPU-Registern ausgetauscht werden können.

Anmerkung:

Die Bandbreite eines Speichersystems beträgt $b_w=p$, falls $p<m/n_c$, andernfalls gilt $b_w=m/n_c$.

Def.3.10: *effektive Bandbreite*

Die effektive Bandbreite b_{eff} (*effective bandwidth* [CKL 77]) gibt an, wieviele Daten durchschnittlich pro Takt zwischen Speicher und Registern ausgetauscht worden sind; $b_{eff} \leq b_w$.

Anmerkung:

Wird innerhalb der Dauer von n_t Takten die Übertragung von $n_{\ddot{u}}$ Daten beobachtet, so ergibt sich die effektive Bandbreite zu

$$b_{eff} = n_{\ddot{u}}/n_t \leq b_w. \qquad (3.1)$$

Def.3.11: *Speicherauslastung*

Die Speicherauslastung u_m (*memory utilization*) gibt an, wie stark der Speicher durchschnittlich pro Takt belegt ist.

Anmerkungen:

Wird der Speicher für die Dauer von n_t Takten beobachtet, so ergibt sich die Speicherauslastung zu

$$u_m = (m \cdot n_t)^{-1} \sum_{i=0}^{m-1} \sum_{j=1}^{n_t} a_{ij}, \qquad (3.2)$$

mit $a_{ij}=0$, falls die i-te Bank im j-ten Takt inaktiv ist und $a_{ij}=1$, falls die i-te Bank im j-ten Takt aktiv ist.

Zwischen der effektiven Bandbreite und der Speicherauslastung gilt der Zusammenhang

$$u_m = b_{eff} n_c / m. \qquad (3.3)$$

Falls $b_w=m/n_c$, ist $u_m=b_{eff}/b_w$; falls $b_w=p<m/n_c$, ist eine 100%-ige Speicherauslastung nicht mehr möglich.

Def.3.12: *aufeinanderfolgender Zugriffswunsch*

Zwei Zugriffswünsche heißen aufeinanderfolgend (*successive memory access requests*), falls die beiden Zugriffswünsche in zwei unmittelbar aufeinanderfolgenden Takten erfolgen. Entsprechend erfolgen k aufeinanderfolgende Zugriffswünsche in k aufeinanderfolgenden Takten.

Def.3.13: *gleichzeitiger Zugriffswunsch*

Zwei Zugriffswünsche heißen gleichzeitig (*simultaneous memory access requests*), falls die beiden Zugriffswünsche innerhalb des gleichen Taktes erfolgen. Entsprechend erfolgen k gleichzeitige Zugriffswünsche innerhalb des gleichen Taktes.

Def.3.14: *Distanz*

Die Distanz d_i (*distance*) bezeichnet den Abstand zwischen den Bänken in einem m-fach verschränkten Speicher, in denen sich die vom i-ten Port aus in zwei aufeinanderfolgenden Zugriffswünschen angesprochenen Speicherzellen befinden, mit $0 \leq d_i \leq m-1$.

Def.3.15: *Zugriffswunsch im Skalarmodus*

Ein Zugriffswunsch im Skalarmodus ist der Zugriffswunsch auf genau eine Speicherzelle aufgrund einer Skalarinstruktion (*scalar load* oder *scalar store*).

Def.3.16: *Zugriffswunsch im Vektormodus - Zugriffsstrom*

Ein Zugriffswunsch im Vektormodus besteht aus n aufeinanderfolgenden Zugriffswünschen auf n mit konstantem Abstand auseinanderliegenden Speicherzellen aufgrund einer Vektorinstruktion (*vector load* oder *vector store*) mit der Vektorlänge n.

Anmerkungen:

Die Zugriffe im Vektormodus werden von einem Port i, beginnend auf der *Startbank* B_i mit der Adresse b_i jeweils mit der festen Distanz d_i durchgeführt. Die Adresse der Bank auf die der $(k+1)$-te Zugriff erfolgt, ergibt sich zu $j=(b_i+kd_i \bmod m)$, mit $k=0,1,2,\ldots,n-1$.

Da im Vektormodus die gewünschten Zugriffe möglichst konfliktfrei verlaufen und somit einen kontinuierlichen *Datenstrom* zur Folge haben sollen, wird synonym für *Zugriffswunsch im Vektormodus* auch der Begriff *Zugriffsstrom* verwendet. So werden für das in Abb.2.3 skizzierte Beispiel der Vektoraddition insgesamt drei Zugriffsströme auf den Speicher benötigt.

Wird der gerade anstehende Zugriffswunsch eines Zugriffsstroms aufgrund einer Konfliktsituation im Port verzögert, so werden die noch ausstehenden Zugriffswünsche dieses Zugriffsstrom entsprechend mitverzögert. Die Möglichkeit, einen dahinterliegenden Zugriffswunsch vorzuziehen, ist nicht gegeben.

Def.3.17: *Rückkehrzahl*

Die Rückkehrzahl r_i ist die Anzahl der Zugriffe, die der i-te Zugriffsstrom tätigt, bevor von ihm wiederum ein Zugriffswunsch auf die gleiche Bank erfolgt.

Def.3.18: *Zugriffsmenge*

Die Zugriffsmenge Z_i des i-ten Zugriffsstroms enthält als Elemente die Adressen der Bänke, in denen die Speicherzellen liegen, auf die zugegriffen werden soll.

Anmerkung:

Da bis zur Rückkehr auf die gleiche Bank auf r_i verschiedene Bänke zugegriffen wird (s. Theorem 4.1), ist mit der Rückkehrzahl auch die Anzahl der Elemente der Zugriffsmenge gegeben, die sich folgendermaßen angeben läßt:

$$Z_i = \{j \mid 0 \le j < m,\ \exists k (k=0,1,\ldots,r_i-1):\ j = (b_i + kd_i \bmod m)\}. \qquad (3.4)$$

Def.3.19: *Pfadmenge*

Die Pfadmenge P_i des i-ten Zugriffsstroms enthält als Elemente die Adressen der Sektionen (bzw. der Zugriffspfade zu den jeweiligen Sektionen), in denen die Bänke liegen, auf die zugegriffen werden soll.

$$P_i = \{j \mid 0 \le j < s,\ \exists k (k=0,1,\ldots,r_i-1):\ j = (b_i + kd_i \bmod s)\}. \qquad (3.5)$$

Def.3.20: *Restbelegungsvektor*

Der Restbelegungsvektor $V_t=(v_0,v_1,\ldots,v_{m-1})$ ist ein Vektor, dessen i-te Komponente die Anzahl der Takte angibt, die die i-te Bank im Zeitpunkt t noch aktiv ist, d.h. $0 \le v_i \le n_c$, $0 \le i < m$.

Def.3.21: *Bankkonflikt*

Mit Bankkonflikt (*bank conflict*) wird der Fall bezeichnet, daß ein Zugriffswunsch auf eine Speicherzelle erfolgt, die innerhalb einer aktiven Bank liegt.

Anmerkung:

Der Zugriffswunsch wird nicht zugelassen und im nächsten Takt unabhängig davon wiederholt, ob die Bank dann frei ist oder nicht. Falls nicht, entsteht erneut ein Bankkonflikt.

Def.3.22: *Simultan-Bankkonflikt*

Mit Simultan-Bankkonflikt (*simultaneous bank conflict* [CRA 83], im folgenden kurz *Simultankonflikt* genannt) wird der Fall bezeichnet, daß mehrere gleichzeitige Zugriffswünsche aus unterschiedlichen CPUs auf Speicherzellen erfolgen, die innerhalb der gleichen inaktiven Bank liegen.

Anmerkung:

Eine Prioritätenregelung entscheidet darüber, welcher der Zugriffswünsche zugelassen wird; die nicht zugelassenen werden im nächsten Takt wiederholt. Da dann die Bank aufgrund des mit der höchsten Priorität zugelassenen Zugriffs noch für n_c-1 Takte belegt ist, erfahren die nicht zugelassenen Speicherzugriffe im Anschluß an einen Simultankonflikt jeweils n_c-1 Bankkonflikte.

Def.3.23: *Pfadkonflikt*

Mit Pfadkonflikt (*section conflict* [CRA 82] oder *line conflict* [CuS 84]) wird der Fall bezeichnet, daß mehrere gleichzeitige Zugriffswünsche aus der gleichen CPU auf Speicherzellen erfolgen, die innerhalb unterschiedlicher inaktiver Bänke liegen, die sich jedoch innerhalb der gleichen Sektion befinden.

Anmerkung:

Eine Prioritätenregelung entscheidet darüber, welcher der Zugriffswünsche zugelassen wird; die nicht zugelassenen werden im nächsten Takt wiederholt. Ein Pfad wird stets nur für einen Takt belegt.

Def.3.24: *Verbundkonflikt*

Ein Verbundkonflikt ist das gemeinsame Auftreten zweier unterschiedlicher Konflikte, die sich aufgrund eines bestimmten Zugriffsverhaltens mehrerer Zugriffsströme wechselseitig bedingen.

Def.3.25: *Doppelkonflikt*

Ein Doppelkonflikt ist das Auftreten von Konflikten zweier sich gegenseitig verzögernder Zugriffsströme innerhalb ein und desselben Taktes.

Def.3.26: *Barriere-Situation*

Eine Barriere-Situation bezeichnet den Fall, daß zwei Zugriffsströme in einen Zyklus geraten, in dem der eine Zugriffsstrom konfliktfrei verläuft und der andere verzögert wird.

Def.3.27: *Synchronisation*

Mit Synchronisation wird der Fall bezeichnet, daß sich eine Konfliktsituation auflöst.

Def.3.28: *feste Priorität*

Feste Priorität (*fixed priority*) ist eine Prioritätenregelung, die zwischen mehreren aktiven Ports aufgrund bestimmter Eigenschaften der Ports festgelegt wird.

Anmerkungen:

Eigenschaften, die die Priorität eines Ports beeinflussen sind typischerweise die Distanz des Zugriffsstroms, der über den betreffenden Port läuft sowie der Zeitpunkt der Aktivierung.

Wird ein bislang inaktiver Port aktiviert, wird die Priorität zwischen allen Ports erneut festgelegt.

Def.3.29: *zyklische Priorität*

Zyklische Priorität (*cyclic priority*) ist eine Prioritätenregelung, die zwischen mehreren aktiven Ports unabhängig von deren Eigenschaften zyklisch alle n_p Takte verändert wird.

Anmerkung:

Diese Prioritätenregelung garantiert, daß bei p' aktiven Ports ($p' \leq p$) jeder aktive Port alle $p' \cdot n_p$ Takte die höchste Priorität erhält und sie für n_p Takte behält.

3.2 ÜBERSICHT ÜBER BISHERIGE ARBEITEN

Viele Untersuchungen haben sich bereits mit der Problematik von Zugriffskonflikten auf mehrfach verschränkte Arbeitsspeicher beschäftigt:

Hellerman [HEL 68] beschreibt ein einfaches Modell, das aus einer unendlich langen Folge von Zugriffswünschen auf ein m-fach verschränktes Speichersystem innerhalb *eines* Speicherzyklus Zugriffswünsche solange zuläßt wie kein weiterer Zugriffswunsch auf eine Bank vorliegt, auf die innerhalb des Zyklus bereits ein Zugriff zugelassen worden ist. Demnach werden beispielsweise aus der Folge von Zugriffswünschen

0 2 3 6 4|0 1 5 4|4 3 2 0|2 7|7|7 3 4 5 2 ...

in jedem Speicherzyklus die jeweils bis zu "|" markierten Zugriffe zugelassen. (Die Ziffern geben hierbei die Adresse der jeweils anzusprechenden Bank an.)

Unter der Annahme, daß die Adressen der Bänke, auf die zugegriffen werden soll, gleichverteilt sind, kommt Hellerman zu dem Schluß, daß die mittlere Anzahl der innerhalb eines Speicherzyklus zugelassenen Zugriffswünsche ungefähr $m^{0.56}$ für $1 \leq m \leq 45$ beträgt. Da in diesem Modell im günstigsten Fall m Zugriffe innerhalb eines Speicherzyklus zugelassen werden, entspricht m der Bandbreite und $m^{0.56}$ der effektiven Bandbreite des Systems.

Der entscheidende Nachteil dieses Modells liegt darin, daß die Anforderungen *serialisiert* sind, d.h. nicht den Anforderungen eines Parallelrechnersystems entsprechen.

Burnett und *Coffman* [BuC 70] nahmen detailliertere Analysen des Modells von Hellerman vor. Insbesondere unterscheiden sie zwischen Zugriffswünschen auf Daten, die zufällig nach einer Gleichverteilung erfolgen und Zugriffswünschen auf Instruktionen, die sequentiell erfolgen und somit eine größere Anzahl von Zugriffswünschen zugelassen werden kann.

Weiterhin beobachteten sie, daß in realen Systemen die Bandbreite b_w i.allg. erheblich kleiner als die Anzahl m der verfügbaren Bänke ist. Deshalb werden aus der Folge der Zugriffswünsche maximal $b_w < m$ Zugriffe zugelassen, womit die Wahrscheinlichkeit, auf Wiederholungen zu treffen, geringer wird.

Ravi [RAV 72] entwickelte ein Modell, das die Wahrscheinlichkeit von Zugriffskonflikten in Abhängigkeit von der Anzahl von p gleichzeitigen Zugriffswünschen auf ein m-fach verschränktes Speichersystem beschreibt.

Dieses Modell ist zur Beschreibung von Zugriffswünschen eines Parallelrechnersystems insofern geeignet, als es nicht mehr von einer Folge von Zugriffswünschen ausgeht, sondern davon, daß mit jedem Speicherzyklus genau p Zugriffswünsche vorliegen. Dabei zeigt sich, daß Hellermans Resultat zu pessimistisch ist.

Das Modell berücksichtigt allerdings nicht, daß die innerhalb eines Speicherzyklus nicht zugelassenen Zugriffswünsche im nächsten Zyklus wiederholt werden müssen, da die Anforderungen der entsprechenden Prozessoren noch nicht erfüllt worden sind.

Budnik und *Kuck* [BuK 71] initiierten mit ihren Untersuchungen die Theorie der *Skewing-Schemata*. Es handelt sich dabei um *Verschiebungen* bzw. allgemein um *Permutationen* der abzuspeichernden Daten.

Beispielsweise läßt sich in einem vierfach verschränkten Speichersystem nicht sowohl auf die Elemente einer Zeile als auch auf die einer Spalte einer 4×4 Matrix gleichzeitig konfliktfrei zugreifen. Befinden sich nämlich die Spaltenelemente auf verschiedenen Bänken, so liegen die Zeilenelemente einer Zeile jeweils auf der gleichen Bank.

Bank	0	1	2	3
	a_{11}	a_{12}	a_{13}	a_{14}
	a_{21}	a_{22}	a_{23}	a_{24}
	a_{31}	a_{32}	a_{33}	a_{34}
	a_{41}	a_{42}	a_{43}	a_{44}

a) Abspeicherung nicht verschoben: Zugriff auf Zeile und Diagonale

Bank	0	1	2	3
	a_{11}	a_{12}	a_{13}	a_{14}
	a_{24}	a_{21}	a_{22}	a_{23}
	a_{33}	a_{34}	a_{31}	a_{32}
	a_{42}	a_{43}	a_{44}	a_{41}

b) Abspeicherung verschoben: Zugriff auf Zeile und Spalte

Abb.3.4 konfliktfreie Zugriffsmöglichkeiten bei unterschiedlicher Abspeicherung in einem vierfach verschränkten Speicher

In Abb.3.4 ist für ein vierfach verschränktes Speichersystem einmal die Abspeicherung der nichtverschobenen Elemente a_{ij}, $1 \le i,j \le 4$ und einmal die Abspeicherung der gegegnüber der vorherigen Zeile jeweils um eine Position verschobenen Elemente einer 4×4 Matrix dargestellt. Dieses Skewing-Schema ermöglicht den konfliktfreien Zugriff sowohl auf die Spaltenele-

mente als auch auf die Zeilenelemente der Matrix, allerdings nicht mehr auf die Elemente der Diagonalen.

Lawrie [LAW 75] fand, daß ein paralleler Zugriff auf einen δ-geordneten Vektor mit n Elementen konfliktfrei möglich ist, falls

$$m \geq n \cdot GGT(\delta, m). \qquad (3.6)$$

Ein δ-geordneter Vektor mit n Elementen ist dabei ein Vektor, dessen i-tes Element in der Bank mit der Adresse $(\delta i+c) \bmod m$, $0 \leq i \leq n-1$ sowie Konstanten δ und c, abgespeichert ist. Über diesen Ansatz stellte er Bedingungen auf, unter denen ein konfliktfreier Zugriff auf Zeilen, Spalten, Diagonalen, Gegendiagonalen sowie auf Untermatrizen einer $n \times n$ Matrix möglich ist.

Weiterhin behandelte er den Gesichtspunkt, die Zuordnung der Daten zu den Prozessoren durch ein weniger aufwendiges Verbindungsnetzwerk als den Kreuzschienenverteiler [BuH 83] zu realisieren. Das von ihm vorgeschlagene *Omega*-Verbindungsnetzwerk ist ein mehrstufiges Verbindungsnetzwerk mit *ld k* Stufen, wenn k die Anzahl der Prozessoren (Zugriffspfade) ist.

Weiter vertieft wurde die Theorie der Skewing-Schemata vor allem durch *Shapiro* [SHA 78] sowie *van Leeuwen* und *Wijshoff* [LuW 83].

Bhandarkar [BHA 75] sowie *Baskett* und *Smith* [BuS 76] untersuchten mit Hilfe von Warteschlangenmodellen das Zugriffsverhalten von q Prozessoren auf ein Speichersystem mit m Bänken unter der Annahme, daß m der Bandbreite des Systems entspricht.

Rau [RAU 79a] untersuchte den Einfluß des Programmverhaltens beim Zugriff auf ein m-fach verschränktes Speichersystem durch einen Prozessor und kam dabei zu dem Ergebnis, daß die Annahme gleichverteilter Zugriffswünsche (z.B. Hellerman) i.allg. nicht geeignet ist. Die von ihm untersuchten Programme zeigten ein *least recently used* Zugriffsverhalten, d.h. es wird auf eine seit längerem nicht mehr benutzte Bank zugegriffen und somit wird eine höhere effektive Bandbreite als unter der Annahme der Gleichverteilung erreicht.

Ramamoorthy und *Wah* [RuW 81] untersuchten verschiedene Scheduling-Algorithmen, die aus einem Puffer mit Zugriffswünschen entscheiden, welcher der zulaßbaren Zugriffswünsche am besten zugelassen werden soll. Sie machen dabei die Annahme, daß ein Speicherzyklus in m "Unterzyklen" (gleich der Anzahl der Bänke) aufgeteilt ist und pro Unterzyklus lediglich ein Zu-

griffswunsch zugelassen werden kann. Als optimal stellte sich ein Algorithmus heraus, der unter den jeweils freien Bänken einen Zugriff auf die Bank zuläßt, auf die insgesamt die meisten Zugriffswünsche vorliegen.

Während die bisherigen Arbeiten im wesentlichen das Zugriffsverhalten auf *m*-fach verschränkte Arbeitsspeicher von Parallelrechnern behandelten, gibt es spezielle Untersuchungen über Speicherzugriffe auf Arbeitsspeicher von Vektorrechnern erst in allerjüngster Zeit:

Lee, Abu-Sufah und *Kuck* [LAK 84] untersuchten den Einfluß des Datentransfers zwischen Arbeitsspeicher und Registern in Vektorrechnern. Wenn Speicherzugriffe speziell im Vektormodus nicht initiiert werden können, weil beispielsweise kein Port dafür zur Verfügung steht, können keine weiteren Instruktionen initiiert werden.

Da für die Leistungsfähigkeit eines Vektorrechners das gleichzeitige Arbeiten möglichst vieler Funktionseinheiten wesentlich ist, kann ein geeignetes Instruktions-Scheduling [SOM 83] in vektorisierenden Compilern die Wartezeiten auf Daten dazu ausnutzen, andere Instruktionen, die nicht auf Daten aus dem Speicher warten vorzuziehen.

Cheung und *Smith* [CuS 84] untersuchten mit Hilfe eines Simulators einige Konstellationen beim Zugriff auf den Speicher des Doppelprozessorsystems CRAY X-MP, wobei sie sich im wesentlichen darauf beschränkten, daß alle Zugriffsströme mit der gleichen Distanz auf den Speicher zugreifen. Sie entdeckten dabei das Phänomen des "linked conflicts", ein Verbundkonflikt (s. Def.3.24) zwischen zwei Zugriffsströmen mit gleicher Distanz, die abwechselnd Bank- und Pfadkonflikte erhalten (s. Kap.4.3.2).

Daß Untersuchungen über das Zugriffsverhalten auf Speicher von Vektorrechnern erst jetzt an Interesse gewinnen, mag damit zusammenhängen, daß in den Vektorrechnern bislang der Zugang zum Speicher über Puffer (CDC CYBER 205) oder lediglich über einen Zugriffspfad (CRAY-1) kontrolliert wurde und somit mögliche Konflikte leicht vorhersehbar waren.

Diese Konzepte haben sich jedoch als erhebliche Engpässe erwiesen, so daß neuere Systeme (z.B. Fujitsu VP-100/VP-200, CRAY X-MP Serie) über mehrere Zugriffspfade verfügen, die miteinander um den Zugriff auf den Speicher konkurrieren. Dies trifft bei den CRAY X-MP Mehrprozessorsystemen nicht nur für die Zugriffspfade innerhalb einer CPU zu, sondern auch für die der anderen CPUs untereinander. Auftretende Konflikte werden hierbei *dynamisch* nach einer Prioritätenregelung gelöst.

4 MODELLIERUNG DES SPEICHERVERHALTENS EINES VEKTORRECHNERS

Zum Verständnis von Abläufen und Zusammenhängen in Rechnersystemen werden in der Leistungsanalyse (*performance analysis*) im wesentlichen folgende drei Methoden angewandt [KOB 78]:

- analytische Methoden
- Simulation
- Messung

In diesem Kapitel wird ein einfaches Modell des Speicherverhaltens eines Vektorrechners mit Hilfe von analytischen Methoden untersucht; mit Simulationen und Messungen beschäftigt sich darauf aufbauend das nächste Kapitel.

4.1 VORÜBERLEGUNGEN ZUR WAHL DER METHODIK

Da Vektorrechner ihre hohe Leistungsfähigkeit erst bei der Verarbeitung von Vektoren, Matrizen oder ähnlich regelmäßigen Datenstrukturen erreichen, werden Zugriffsfolgen auf solche Datenstrukturen im Vordergrund der Untersuchungen stehen. Das bedeutet aber auch, daß stochastische Prozesse sowie Warteschlangenmodelle zur Beschreibung des Zugriffsverhaltens auf Arbeitsspeicher in Vektorrechnern wenig geeignet sind.

Um analytische Untersuchungen überschaubar zu halten, müssen vereinfachte Annahmen getroffen werden. Die beiden wichtigsten Idealisierungen für die folgenden Untersuchungen sind:

- ein Speicherzugriff im Vektormodus sei unendlich lang;
- alle Zugriffsströme beginnen gleichzeitig.

Die erste Annahme wird aus der Tatsache heraus gemacht, daß die Anzahl der möglichen Zustände für einen m-fach verschränkten Speicher, auf den gleichzeitig über p Ports mit einer jeweils festen Distanz d_i,

$i=1,2,...,p$, zugegriffen wird, endlich ist. Die Anzahl aller möglichen Zustände setzt sich aus der möglichen Anzahl von Speicherzuständen und der möglichen Anzahl der Port-Zustände zusammen.

Der Speicherzustand in einem bestimmten Zeitpunkt t ist durch den Restbelegungsvektor V_t gekennzeichnet. Demnach sind $(n_c+1)^m$ verschiedene Speicherzustände möglich. Zugriffe können allerdings nur dann stattfinden, wenn mindestens eine Bank inaktiv ist, was in $(n_c+1)^m-n_c^m$ Fällen zutrifft.

Die Anzahl der möglichen Port-Zustände ist durch die Bank gegeben, auf die der jeweils nächste Zugriff erfolgen soll, bei m Bänken und p Ports demnach m^p, falls die Ports untereinander *feste Priorität* besitzen. Bei *zyklischer Priorität* ergeben sich noch einmal p Möglichkeiten. Die Gesamtzahl aller möglichen Zustände ist demnach durch das Produkt $(n_c+1)^m \cdot m^p \cdot p$ gegeben.

Demzufolge treten nach einer *Einlaufphase* die Zugriffsströme in einen Zyklus ein, da nach der Initiierung alle weiteren Zugriffe *determiniert* erfolgen. Sobald ein Zyklus vorgefunden wird, kann das weitere Zugriffsverhalten beschrieben werden. Der zyklische Zustand wird auch bei den Untersuchungen im Vordergrund stehen, da er das Verhalten für große Vektorlängen n wiederspiegelt, für die sich ständig wiederkehrende Konflikte besonders negativ bemerkbar machen.

Die zweite Annahme wird getroffen, um bei einer endlichen Zustandsmenge bleiben zu können. Diese Annahme bedeutet auch keine gravierende Vereinfachung des Modells, da die durch gegenseitige zeitliche Verschiebungen der Zugriffsströme entstehenden Konstellationen ebenso durch unterschiedliche Startbänke erreichbar sind.

Eine entscheidende Voraussetzung für die Brauchbarkeit eines Modells ist die Identifikation der wesentlichen Einflußgrößen.

Als wichtigstes Ergebnis soll das Modell die effektive Bandbreite b_{eff} liefern, da dies ein direktes Maß dafür ist, wie störungsfrei die Zugriffsströme verlaufen. Weitere sinnvolle, jedoch indirekt in der effektiven Bandbreite enthaltenen Angaben sind noch die Speicherauslastung sowie die Anzahl der vorgefundenen Bank-, Pfad- und Simultankonflikte.

Die entscheidenden Einflußgrößen für den Speicher sind:

- die Anzahl der Bänke m
- die Anzahl der Sektionen s
- die Bankzykluszeit t_c (bzw. n_c)

Der i-te Port wird ausreichend gekennzeichnet durch:

- die nächste zuzugreifende Bank
- die Distanz d_i
- die Priorität

"Die nächste zuzugreifende Bank" ist zu Beginn gleich der Startbank B_1 mit der Adresse b_1.

4.2 ZUGRIFF ÜBER EINEN PORT

Ist ein Zugriff lediglich über einen Port möglich (z.B. CRAY-1), so läßt sich das Verhalten dieses einen Zugriffsstroms relativ leicht analysieren, da weder Simultankonflikte noch Pfadkonflikte auftreten können. Verzögerungen entstehen also nur im Falle von Bankkonflikten.

Um festzustellen, welche Umstände überhaupt zu Bankkonflikten führen können, wird zunächst die *Rückkehrzahl* r_1 untersucht. Die Startbank B_1 wird dabei als Bezugspunkt gewählt. (Wegen $p=1$ werden in diesem Abschnitt $r_1=r$, $d_1=d$, $b_1=b$ und $B_1=B$ gesetzt.)

Theorem 4.1: Bei einem Speicherzugriff im Vektormodus mit der Distanz d ($0 \le d \le m-1$) auf einen m-fach verschränkten Speicher erfolgt auf die Startbank B nach

$$r = m/GGT(m,d) \qquad (4.1)$$

zugelassenen Zugriffen erneut ein Zugriffswunsch ($1 \le r \le m$).

Beweis:

Ausgehend von der Startbank B treten in der Sequenz $b+id$, $i=0,1,2,\ldots$ Zahlen auf, die bezüglich *mod* m identisch sind

$$(b + id) \equiv (b + jd) \bmod m, \quad i \neq j. \tag{4.2}$$

Daraus ergibt sich

$$(j - i)d \equiv 0 \bmod m. \tag{4.3}$$

Die minimale Differenz $j-i$, die die Gl.(4.3) erfüllt, entspricht gerade der gesuchten Rückkehrzahl r. Mit Gl.(4.3) ergibt sich

$$(j - i)d \equiv 0 \equiv km \bmod m, \tag{4.4}$$

mit $k=0,1,2,\ldots$. Seien $d'=d/GGT(m,d)$ und $m'=m/GGT(m,d)$, so sind d' und m' teilerfremd und es gibt ein k' für das sich Gl.(4.4) umformen läßt zu

$$(j - i) = k'm'/d'. \tag{4.5}$$

Der Wert von k' muß nun so bestimmt werden, daß $(j-i)$ minimal wird. Da m' und d' aber teilerfremd sind, ist dies der Fall für $k'=d'$ und somit

$$r = m' = m/GGT(m,d). \tag{4.6}$$

□

Korollar 4.1: Ist m eine Primzahl, so ist $r=1$ für $d=0$ und $r=m$ für $1 \leq d < m$.

Korollar 4.2: Die Menge M_r aller *möglichen* Rückkehrzahlen r ist gegeben durch

$$M_r = \{r \mid r \text{ ist Teiler von } m\}. \tag{4.7}$$

Für den in realen Systemen sehr wichtigen Fall, daß m eine Zweier-Potenz ist, d.h. $m=2^k$, ergeben sich die möglichen Rückkehrzahlen zu $r=2^j$, $j=0,1,\ldots,k$.

Theorem 4.2: Treten bei nur einem Zugriffsstrom Bankkonflikte auf, so erfolgen diese ausschließlich auf der Startbank.

Beweis:

Die ersten r Zugriffswünsche erfolgen auf r unterschiedliche Bänke und können somit jeweils zugelassen werden. Danach erfolgt wieder ein Zugriffswunsch auf die Startbank, wo nun erstmalig ein Bankkonflikt entstehen kann, falls die Startbank aufgrund des ersten zugelassenen Zugriffs noch aktiv ist. Dieser Fall ist für

$$r < n_c \tag{4.8}$$

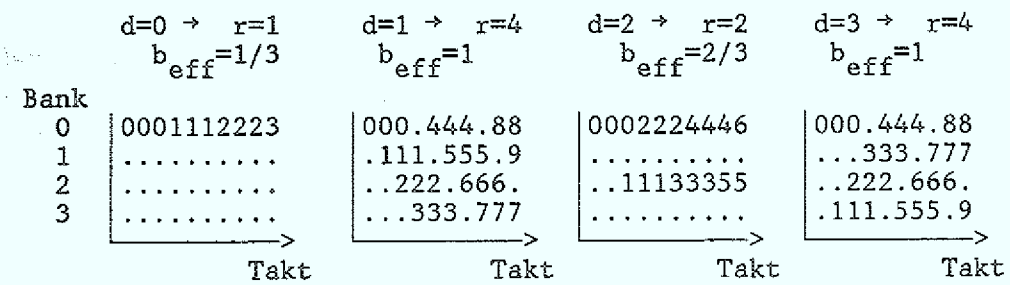
gegeben. Liegt solch ein Fall vor, so wird dieser Zugriffswunsch und damit alle nachfolgenden Zugriffswünsche aufgrund Def.3.16 im Port um

$$n_c - r \tag{4.9}$$

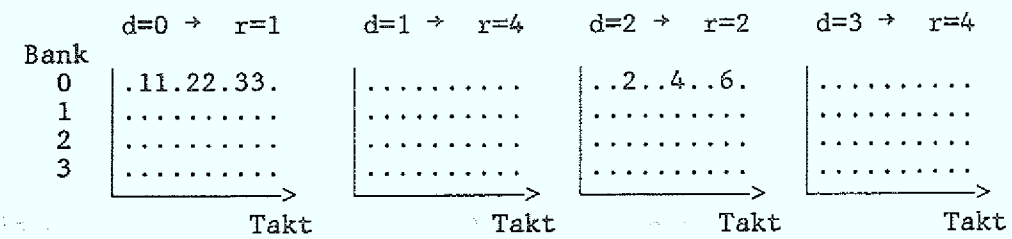
Takte verzögert.

□

Korollar 4.3: Ist die Bedingung (4.8) erfüllt, so treten innerhalb von r Zugriffen n_c-r Bankkonflikte auf.



a.) erfolgte Zugriffe



b.) erfolgte Bankkonflikte

Abb.4.1 Zugriffe und Bankkonflikte auf einen Speicher mit $m=4$, $n_c=3$, $b=0$ und $0 \le d \le 3$

In Abb.4.1 sind jeweils die Zugriffe der drei möglichen Rückkehrzahlen *1,2,4* für die Distanzen *0,1,2,3* auf ein vierfach verschränktes Speichersystem mit $n_c=3$ sowie die Bankkonflikte dargestellt. Die Adresse der Startbank ist mit *0* gegeben und die Ziffern geben die laufende Nummer des Zugriffs, beginnend bei *0* an.

Die effektive Bandbreite für den *Konfliktfall* beträgt

$$b_{eff} = r/n_c < 1, \tag{4.10}$$

denn innerhalb von jeweils $r+(n_c-r)=n_c$ Takten werden genau r Zugriffe zugelassen. Im konfliktfreien Fall, d.h. $r \geq n_c$, wird die maximale Bandbreite $b_w=1$ erreicht.

Daraus ergibt sich, daß ein verschränktes Speichersystem mindestens über $m=n_c$ Bänke verfügen sollte. Andernfalls bleibt $b_w<1$, da die maximale Speicherauslastung mit $u_m=1$ erreicht ist (vgl. Def.3.9).

4.3 ZUGRIFF ÜBER ZWEI PORTS

Das Zugriffsverhalten über zwei Ports ist dadurch gekennzeichnet, daß die Ports untereinander über keine Möglichkeit verfügen, Zugriffskonflikte im voraus zu erkennen. Das bedeutet, daß jeder aktive Port in jedem Takt versucht, seinen anstehenden Zugriffswunsch auszuführen.

Als Zugriffskonflikte kommen nun alle drei Konfliktsituationen

- Bankkonflikt
- Pfadkonflikt
- Simultankonflikt

in Frage, wobei ein gemeinsames Auftreten von Pfadkonflikten und Simultankonflikten bei zwei Ports nicht vorkommt.

Für den Fall eines Speichersystems, das über genauso viele Sektionen wie Bänke verfügt, d.h. $m=s$, wird bei gleichzeitigem Zugriffswunsch beider Ports auf die gleiche inaktive Bank festgelegt, daß es sich um einen Si-

multankonflikt handelt. Bei gleichzeitigem Zugriffswunsch beider Ports auf ein Speichersystem mit weniger Sektionen als Bänken, d.h. $s<m$, handelt es sich um einen Pfadkonflikt, falls die beiden Zugriffswünsche auf zwei inaktive Bänke innerhalb der gleichen Sektion erfolgen, selbst wenn es sich dabei um die gleiche Bank handelt.

Diese Festlegung ist dadurch gerechtfertigt, daß in einem Speichersystem mit zwei Ports und gleicher Anzahl von Sektionen und Anzahl von Bänken, jeder Port über einen eigenen Zugriffspfad Zugang zu jeder Bank hat. Die Zugriffsmenge Z_i ist also stets identisch mit der zugehörigen Pfadmenge P_i und die Pfade können somit nie einen Engpaß bilden. In einem Speichersystem mit geringerer Anzahl von Sektionen als Anzahl von Bänken ist u.U. ein Engpaß durch die zu geringe Anzahl von Zugriffspfaden gegeben.

Diese Unterscheidung kann auch für das in Abb.3.3 dargestellte Speichersystem in Abhängigkeit von der Herkunft der beiden Zugriffsströme getroffen werden. Trotz geringerer Anzahl von Sektionen als Anzahl von Bänken treten keine Pfadkonflikte auf, falls je ein Zugriffsstrom von je einer CPU aus auf den gemeinsamen Speicher zugreifen will. Andererseits treten keine Simultankonflikte auf, falls beide Zugriffsströme von einer CPU aus auf den Speicher zugreifen wollen.

Im weiteren werden ausschließlich Zugriffsströme betrachtet, deren jeweilige Rückkehrzahl groß genug ist, so daß keine selbstverschuldeten Zugriffskonflikte entstehen, d.h.

$$r_i \geq n_c. \tag{4.11}$$

Außerdem wird

$$m \geq 2n_c \tag{4.12}$$

vorausgesetzt. Demnach ist die Bandbreite des betrachteten Speichersystems durch die Anzahl der aktiven Ports begrenzt, d.h. $b_W=2$ (vgl. Def.3.9).

4.3.1 GLEICHE ANZAHL VON SEKTIONEN UND BÄNKEN

Erstrebenswert sind sicherlich diejenigen Zugriffsströme, die sich bei ihren jeweiligen Zugriffswünschen nicht gegenseitig stören, also *konfliktfrei* verlaufen und somit die effektive Bandbreite gleich der maximalen Bandbreite ist: $b_{eff}=b_w=2$. Diese Situation kann unter bestimmten Voraussetzungen erreicht werden, wie durch die Theoreme 4.3 bis 4.6 belegt wird.

Theorem 4.3: Zwei Zugriffsströme verlaufen konfliktfrei, falls ihre Zugriffsmengen disjunkt sind

$$Z_1 \cap Z_2 = \emptyset. \tag{4.13}$$

Beweis:

Disjunkte Zugriffsmengen bedeutet, daß auf die Bänke, auf die der eine Zugriffsstrom zugreift, der andere Zugriffsstrom niemals zugreift und somit eine gegenseitige Störung ausgeschlossen ist. Selbstverursachte Konflikte treten aufgrund der Voraussetzung (4.11) nicht auf.

□

In Abb.4.2 sind die konfliktfreien Zugriffsströme zweier disjunkter Zugriffsmengen für $d_1=2$ und $b_1=0$ sowie $d_2=4$ und $b_2=1$ auf ein Speichersystem mit $m=12$ und $n_c=3$ dargestellt. ("1" bzw. "2" bezeichnet die beiden Zugriffsströme.)

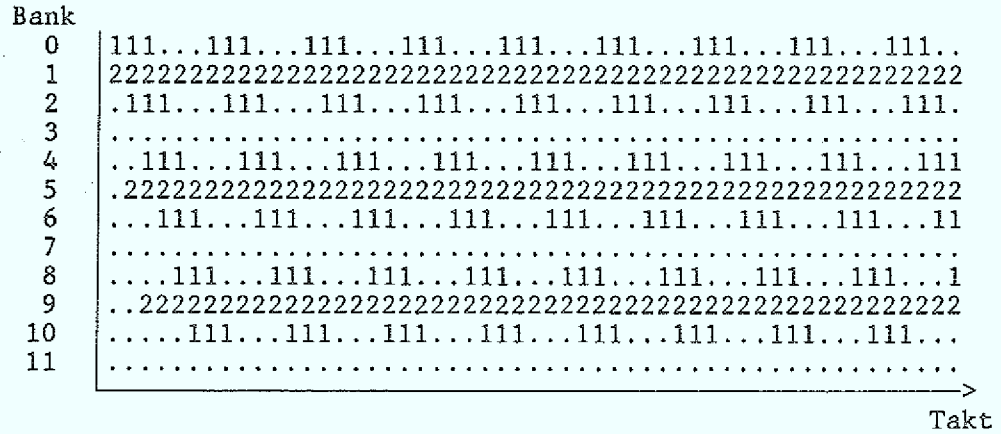


Abb.4.2 Zugriffe auf einen zwölffach verschränkten Speicher bei disjunkten Zugriffsmengen

Theorem 4.4: Es existieren Startbänke B_1 und B_2 mit den Adressen b_1 und b_2, so daß die zugehörigen Zugriffsmengen Z_1 und Z_2 zweier Zugriffsströme mit den Distanzen d_1 und d_2 disjunkt sind, g.d.w.

$$GGT(m,d_1,d_2) > 1. \tag{4.14}$$

Beweis:

Die Rückkehrzahl r_i gibt gleichzeitig die Anzahl der unterschiedlichen Bänke an, auf die überhaupt im Laufe des Zugriffsstroms zugegriffen wird. Disjunkte Zugriffsmengen sind nur dann erreichbar, falls keiner der beiden Zugriffsströme auf alle Bänke zugreift, d.h. es muß gelten

$$r_1, r_2 < m. \tag{4.15}$$

Unter der Annahme $f=GGT(m,d_1,d_2)>1$ ist Gl.(4.15) aufgrund Gl.(4.1) erfüllt, d.h.

$$\begin{aligned} f_1 &= GGT(m,d_1) \geq f \rightarrow r_1 = m/f_1 \\ f_2 &= GGT(m,d_2) \geq f \rightarrow r_2 = m/f_2. \end{aligned}$$

Weiterhin gilt

$$\begin{aligned} &\{i \mid \exists k(k=0,1,\ldots,m/f): i = (fk \bmod m)\} \cap \\ &\{j \mid \exists k(k=0,1,\ldots,m/f): j = (fk + 1 \bmod m)\} = \emptyset. \end{aligned} \tag{4.16}$$

Wegen $f \mid d_1$ und $f \mid d_2$, sind die beiden Zugriffsmengen Z_1 und Z_2 Teilmengen der beiden in Gl.(4.16) angegeben Mengen, falls die beiden Startbänke um 1 gegeneinander verschoben sind, d.h.

$$b_2 = b_1 + 1, \tag{4.17}$$

und somit ebenfalls disjunkt.

Nun bleibt noch zu zeigen, daß es für den Fall $GGT(m,d_1,d_2)=1$ nicht möglich ist disjunkte Zugriffsmengen zu erhalten.

Wird in Gl.(3.4) zur Berechnung der Elemente der Zugriffsmenge Z_1 anstelle der Distanz d_1 die Distanz $d_1'=GGT(m,d_1)$ eingesetzt, so erhält man damit die gleichen Elemente. Entsprechendes gilt für $d_2'=GGT(m,d_2)$. Da hier lediglich die beiden Zugriffsmengen Z_1 und Z_2, nicht jedoch die Reihenfolge

der Zugriffe von Bedeutung sind, können deswegen die beiden Distanzen d_1' und d_2' betrachtet werden.

Wegen der Voraussetzung $GGT(m,d_1,d_2)=1$ sind d_1' und d_2' teilerfremd und es gibt ein k_1 und ein k_2, so daß mit dem euklidischen Algorithmus (s. Anhang) gilt

$$1 \equiv k_1 d_1' - k_2 d_2' \bmod m. \tag{4.18}$$

Wird die Adresse der Startbank des ersten Zugriffsstroms zu $b_1=0$ angenommen, so muß die Adresse der Startbank des zweiten Zugriffsstroms zu $b_2 \neq 0$ gesetzt werden. Durch Multiplikation von Gl.(4.18) mit b_2 ergibt sich

$$b_2 k_1 d_1' \equiv b_2 + b_2 k_2 d_2' \bmod m. \tag{4.19}$$

Mit $i=b_2k_1$ und $j=b_2k_2$, ergibt sich Gl.(4.19) zu

$$i d_1' \equiv b_2 + j d_2' \bmod m, \tag{4.20}$$

womit gezeigt ist, daß sich zu jedem i ein j so finden läßt, daß die beiden Zugriffsmengen wenigstens ein gemeinsames Element besitzen und demnach nicht disjunkt sind.

□

Zwei Zugriffsströme mit nichtdisjunkten Zugriffsmengen sind solche, die mindestens auf eine gemeinsame Bank zugreifen. Es soll nun untersucht werden, unter welchen Voraussetzungen solche Zugriffsströme konfliktfrei verlaufen.

Beginnen beide Zugriffsströme gleichzeitig auf verschiedenen Startbänken B_1 und B_2, so lassen sich die Adressen der Bänke der jeweiligen Zugriffswünsche folgendermaßen gegenüberstellen:

$$\begin{array}{ll} b_1 \ , \ b_1 + d_1 \ , \ b_1 + 2d_1 \ , \ b_1 + 3d_1 \ , \ \ldots & \bmod m \\ b_2 \ , \ b_2 + d_2 \ , \ b_2 + 2d_2 \ , \ b_2 + 3d_2 \ , \ \ldots & \bmod m \end{array}$$

Da nach einem erfolgten Zugriff auf eine Bank diese für n_c Takte belegt ist, kann ein "Fenster" der Breite n_c Takt für Takt über die angegebene Sequenz geschoben werden. Die Zugriffsströme verlaufen dann konfliktfrei, wenn in jedem Zeitpunkt keine Bank innerhalb des Fensters mehr als einmal vorkommt, was sich durch

$$\forall k: \{b_1 + kd_1,\ b_1 + (k+1)d_1,\ldots,\ b_1 + (k+n_c-1)d_1\}_* \cap \{b_2 + kd_2,\ b_2 + (k+1)d_2,\ldots,\ b_2 + (k+n_c-1)d_2\}_* = \emptyset \qquad (4.21)$$

beschreiben läßt; $k \in \mathbb{Z}_m$. (Die Schreibweise $\{\}_*$ wird im weiteren verwandt, um anzudeuten, daß die Elemente der entsprechenden Menge jeweils *mod m* berechnet werden.)

Beispiel: $m=8,\ n_c=2,\ d_1=1,\ d_2=5,\ b_1=0,\ b_2=2$

$$\begin{aligned}
k = 0:&\ \{0,1\}_* \cap \{2,7\}_* = \{0,1\} \cap \{2,7\} = \emptyset\\
k = 1:&\ \{1,2\}_* \cap \{7,12\}_* = \{1,2\} \cap \{7,4\} = \emptyset\\
k = 2:&\ \{2,3\}_* \cap \{12,17\}_* = \{2,3\} \cap \{4,1\} = \emptyset\\
k = 3:&\ \{3,4\}_* \cap \{17,22\}_* = \{3,4\} \cap \{1,6\} = \emptyset\\
k = 4:&\ \{4,5\}_* \cap \{22,27\}_* = \{4,5\} \cap \{6,3\} = \emptyset\\
k = 5:&\ \{5,6\}_* \cap \{27,32\}_* = \{5,6\} \cap \{3,0\} = \emptyset\\
k = 6:&\ \{6,7\}_* \cap \{32,37\}_* = \{6,7\} \cap \{0,5\} = \emptyset\\
k = 7:&\ \{7,8\}_* \cap \{37,42\}_* = \{7,0\} \cap \{5,2\} = \emptyset
\end{aligned}$$

Ab $k=8$ wiederholt sich der Zyklus, so daß dieses Beispiel zwei konfliktfrei verlaufende Zugriffsströme darstellt.

Entscheidend für die Konfliktfreiheit sind jedoch weniger die absoluten Positionen der laufenden Zugriffswünsche, als ihre *relative* Position zueinander. Wird willkürlich $b_1=0$ gesetzt, so ergibt sich Gl.(4.21) zu

$$\forall k: \{kd_1,\ (k+1)d_1,\ldots,\ (k+n_c-1)d_1\}_* \cap \{b_2 + kd_2,\ b_2 + (k+1)d_2,\ldots,\ b_2 + (k+n_c-1)d_2\}_* = \emptyset. \qquad (4.22)$$

Durch Subtraktion von kd_1 *mod m* von den einzelnen Elementen in Gl.(4.22) läßt sich eine Fixierung der ersten Menge erreichen, d.h.

$$\forall k: \{0,\ d_1,\ldots,\ (n_c-1)d_1\}_* \cap \{b_2 + k(d_2-d_1),\ b_2 + d_2 + k(d_2-d_1),\ldots,\ b_2 + (n_c-1)d_2 + k(d_2-d_1)\}_* = \emptyset. \qquad (4.23)$$

Wird angenommen, daß zwei konfliktfrei verlaufende Zugriffsströme vorliegen, so stellt $b_2=(n_c d_1 \ mod\ m)$ die Adresse einer möglichen Startbank für den zweiten Zugriffsstrom dar, falls $b_1=0$ gesetzt wird, was sich aufgrund folgender Überlegung ergibt:

Damit die beiden Zugriffsmengen Z_1 und Z_2 nicht disjunkt sind, müssen sie über eine oder mehrere gemeinsame Adressen von Bänken verfügen. Im zeit-

lichen Ablauf der Zugriffe bleibt die Konfliktfreiheit gewahrt, falls die Zugriffsströme so verlaufen, daß auf mindestens einer Bank die zwei Zugriffe dicht aneinanderliegen, d.h. nach einem zugelassenen Zugriff durch den einen Zugriffsstrom wird nach genau n_c Takten ein Zugriff auf die gleiche Bank durch den anderen Zugriffsstrom zugelassen. Durch Anpassung der Adressen der Bänke kann diese gemeinsame Bank die Adresse $n_c d_1 \ mod \ m$ erhalten und als Startbank des zweiten Zugriffsstroms angesehen werden. Da der erste Zugriffsstrom im n_c-ten Takt auf diese Bank zugreift, erfolgte dessen Zugriff zum Startzeitpunkt auf die $n_c d_1 \ mod \ m$ zurückliegende, die 0-te Bank.

Somit läßt sich Gl.(4.23) als Bedingung für den konfliktfreien Verlauf zweier Zugriffsströme mit *nichtdisjunkten* Zugriffsmengen schreiben als

$$\begin{aligned}&\{i \mid 0\le i<m,\ \exists j(j=0,1,\ldots,n_c-1):\ i = (jd_1 \bmod m)\} \cap \\ &\{i \mid 0\le i<m,\ \exists j(j=0,1,\ldots,n_c-1),\ \exists k(k=0,1,2,\ldots,m-1): \\ &\qquad\qquad i = (n_c d_1 + jd_2 + k(d_2-d_1) \bmod m)\} = \emptyset. \end{aligned} \tag{4.24}$$

In anderer Darstellung ergibt Gl.(4.24)

$$\begin{aligned}&\forall k:\ \{0,\ d_1,\ldots,\ (n_c-1)d_1\}_* \cap \\ &\quad \{n_c d_1 + k(d_2-d_1),\ n_c d_1 + d_2 + k(d_2-d_1),\ldots \\ &\qquad\qquad \ldots,\ n_c d_1 + (n_c-1)d_2 + k(d_2-d_1)\}_* = \emptyset, \end{aligned} \tag{4.25}$$

was sich zu

$$\begin{aligned}&\forall k:\ \{0,\ d_1,\ldots,\ (n_c-1)d_1\}_* \cap \\ &\quad \{n_c d_1 + k(d_2-d_1),\ (n_c+1)d_1 + (d_2-d_1) + k(d_2-d_1),\ldots \\ &\qquad\qquad \ldots,\ (2n_c-1)d_1 + (n_c-1)(d_2-d_1) + k(d_2-d_1)\}_* \\ &= \{0,\ d_1,\ldots,\ (n_c-1)d_1\}_* \cap \\ &\quad \{n_c d_1 + k(d_2-d_1),\ (n_c+1)d_1 + (k+1)(d_2-d_1),\ldots \\ &\qquad\qquad \ldots,\ (2n_c-1)d_1 + (k+n_c-1)(d_2-d_1)\}_* = \emptyset \end{aligned} \tag{4.26}$$

umformen läßt. Da Gl.(4.26) für alle $k \in \mathbb{Z}_m$ erfüllt sein muß, kann Gl.(4.26) auch zu

$$\begin{aligned}&\forall k:\ \{0,\ d_1,\ldots,\ (n_c-1)d_1\}_* \cap \\ &\quad \{n_c d_1 + k(d_2-d_1),\ (n_c+1)d_1 + k(d_2-d_1),\ldots \\ &\qquad\qquad \ldots,\ (2n_c-1)d_1 + k(d_2-d_1)\}_* = \emptyset \end{aligned} \tag{4.27}$$

vereinfacht werden.

In Abb.4.3 sind die konfliktfreien Zugriffsströme zweier nichtdisjunkter Zugriffsmengen für $d_1=1$ und $d_2=7$ auf ein zwölffach verschränktes Speichersystem mit $n_c=3$ dargestellt. ("1" bzw. "2" bezeichnen die beiden Zugriffsströme.)

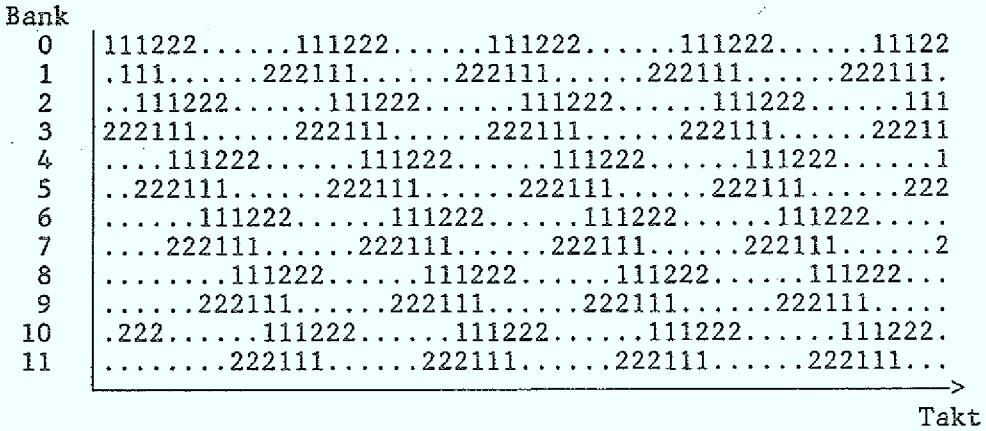


Abb.4.3 Zugriffe auf einen Speicher bei nichtdisjunkten Zugriffsmengen $m=12$, $n_c=3$, $d_1=1$, $b_1=0$, $d_2=7$, $b_2=3$

Aufgrund dieser Vorüberlegungen läßt sich nun zeigen:

Theorem 4.5: Sei $f=GGT(m,d_1,d_2)$. Es existieren für zwei Zugriffsströme mit den Distanzen d_1 und d_2 und nichtdisjunkten Zugriffsmengen Z_1 und Z_2 Startbänke B_1 und B_2, so daß die Zugriffsströme konfliktfrei verlaufen, g.d.w.

$$GGT(m/f,(d_2-d_1)/f) \geq 2n_c. \qquad (4.28)$$

Beweis:

Zunächst werden $m'=m/f$, $d_1'=d_1/f$ und $d_2'=d_2/f$ gesetzt, wodurch die relativen Positionen der Zugriffswünsche zueinander und somit auch die Rückkehrzahlen r_1 und r_2 nicht verändert werden, wohl aber die Differenz zwischen d_1' und d_2'. Mit f ist ein Faktor gegeben, der eine Spreizung der "effektiven" Bankanzahl bewirkt, auf das Zugriffsverhalten jedoch keinen Einfluß hat, denn die Adressen der Bänke, auf die überhaupt zugegriffen wird sind für beide Zugriffsströme Vielfache von f; d_1', d_2' und m' sind also teilerfremd. Weiterhin wird $d_1' | m'$ angenommen, da andere Werte für d_1' dazu isomorph sind (s. Anhang).

Werden die beiden Zugriffsfolgen, die jeweils auf Bank *0* beginnen folgendermaßen übereinandergeschrieben:

$$0 , d_1' , 2d_1' , 3d_1' , \ldots \qquad \text{mod } m'$$
$$0 , d_2' , 2d_2' , 3d_2' , \ldots \qquad \text{mod } m'$$

so läßt sich jeweils die Differenz $k(d_2'-d_1')$, $k=0,1,2,\ldots,m'-1$ bilden. Der bezüglich *mod* m' kleinste Wert der Differenz $k(d_2'-d_1')$ für beliebige $k \in \mathbb{Z}_m$, ist dabei $g=GGT(m',d_2'-d_1')$, wobei wegen $0 \equiv m'$ *mod* m' der Wert m' zu nehmen ist, falls eine Differenz den Wert 0 annimmt.

Damit lassen sich mit Hilfe des euklidischen Algorithmus (s. Anhang) Koeffizienten k_1 und k_2 so finden, daß gilt

$$g = k_1(d_2'-d_1') + k_2m'. \tag{4.29}$$

Für jedes andere Element

$$g' = k_3(d_2'-d_1') + k_4m', \tag{4.30}$$

gilt $g|g'$; g' ist also ein Vielfaches von g.

Aufgrund der obigen Vorüberlegungen existieren Startbänke B_1 und B_2, so daß die Zugriffsströme konfliktfrei verlaufen, g.d.w. die Zugriffe für die Startadressen $b_2=b_1+n_cd_1$ konfliktfrei verlaufen. Betrachtet man nun Gl.(4.27) und setzt darin zunächst $k=0$, so ergibt sich

$$\{0, d_1', \ldots, (n_c-1)d_1'\}_* \cap$$
$$\{n_cd_1', (n_c+1)d_1', \ldots, (2n_c-1)d_1'\}_* = \emptyset. \tag{4.31}$$

Aufgrund des "Fensters" in dem keine Adresse einer Bank mehrmals vorkommen darf, ergeben sich als Differenzen zwischen allen möglichen Paaren von Elementen

$$jd_1', \quad 1 \le j \le 2n_c-1.$$

Konfliktfreiheit ist gleichbedeutend damit, daß Gl.(4.31) erfüllt bleibt, wenn $k(d_2'-d_1')$, $k \in \mathbb{Z}_m$, beliebig, zur zweiten Menge hinzuaddiert wird. Dies ist jedoch gleichbedeutend damit, daß kein Element aus der einen Menge in der jeweils anderen enthalten sein darf bzw. alle möglichen paarweisen Differenzen zwischen Elementen der beiden Mengen müssen für $1 \le j \le 2n_c-1$ ungleich 0 sein, d.h.

$$\forall k: \quad jd_1' + k(d_2'-d_1') \not\equiv 0 \mod m'. \tag{4.32}$$

$g=GGT(m',d_2'-d_1')$ stellt das betragsmäßig kleinste Element dar, das bezüglich *mod* m' die gleichen Werte wie $k(d_2'-d_1')$ erzeugt, so daß sich Gl.(4.32) schreiben läßt als

$$\forall k: \quad jd_1' + kg \not\equiv 0 \mod m' \tag{4.33}$$

bzw. mit $k'=(m'-k)$

$$\forall k': \quad jd_1' \not\equiv k'g \mod m', \tag{4.34}$$

mit $1 \le j \le 2n_c-1$.

Da nach Voraussetzung d_1', d_2' und m' teilerfremd sind, sind nach Gl.(A.5) auch d_1' und $g=GGT(m',d_2'-d_1')$ teilerfremd. Damit gilt nach Gl.(A.3)

$$gd_1' \le m'. \tag{4.35}$$

Mit Gl.(4.35) sowie $GGT(d_1',g)=1$ ergibt sich, daß g der kleinste Wert für j ist, für den es ein k' gibt mit

$$jd_1' \equiv k'g \mod m'. \tag{4.36}$$

Gl.(4.34), was gleichbedeutend mit Konfliktfreiheit ist, ist also erfüllt, g.d.w.

$$g > (2n_c-1) \tag{4.37}$$

□

Korollar 4.4: Ist $d_1=d_2$, gilt $GGT(m,0)=m$. Zugriffsströme mit gleichen Distanzen laufen also stets konfliktfrei, falls für die Rückkehrzahl $r \ge 2n_c$ gegeben ist.

Korollar 4.5: Ist die Anzahl m der Bänke eine Primzahl, so gilt für $d_1 \neq d_2$ stets $GGT(m,d_2-d_1)=1$. In diesem Fall führen unterschiedliche Distanzen zweier Zugriffsströme stets zu Konflikten.

Theorem 4.6: Beginnen zwei Zugriffsströme, die Gl.(4.28) erfüllen, dergestalt, daß es aufgrund der jeweils gewählten Startbank auf einer beliebigen Bank zu einem Konflikt kommt, so findet auf dieser Bank eine Synchronisation (vgl. Def.3.27) statt.

Beweis:

Sobald ein Zugriffsstrom auf eine Bank trifft, die noch belegt ist wird der anstehende Zugriffswunsch verzögert, bis die Bank gerade frei geworden ist und somit auf dieser Bank die beiden Zugriffe dicht aneinanderliegen. Das entspricht jedoch der Situation nach Gl.(4.24) und es treten im weiteren Verlauf keine Konflikte mehr auf.

□

Korollar 4.6: Die Wahl $b_2=b_1+n_c d_1$ für die relative Startposition stellt bei Erfüllung von Gl.(4.28) die Konfliktfreiheit von Anfang an sicher. Wegen der Synchronisation wird stets ein konfliktfreier Zyklus unabhängig von der Wahl der Startbänke erreicht, falls Gl(4.28) erfüllt ist.

Alle anderen Zugriffsströme, deren Distanzen d_1 und d_2 nicht unter die durch die Theoreme 4.3 und 4.5 erfaßten Kombinationen fallen, führen zu Konflikten.

Ein spezieller Konfliktfall ist durch die Barriere-Situation (vgl. Def.3.26) gegeben. Dabei verläuft einer der beiden Zugriffsströme konfliktfrei und bildet für den anderen Zugriffsstrom eine "Barriere", wodurch dieser regelmäßig in seinen Zugriffswünschen verzögert wird.

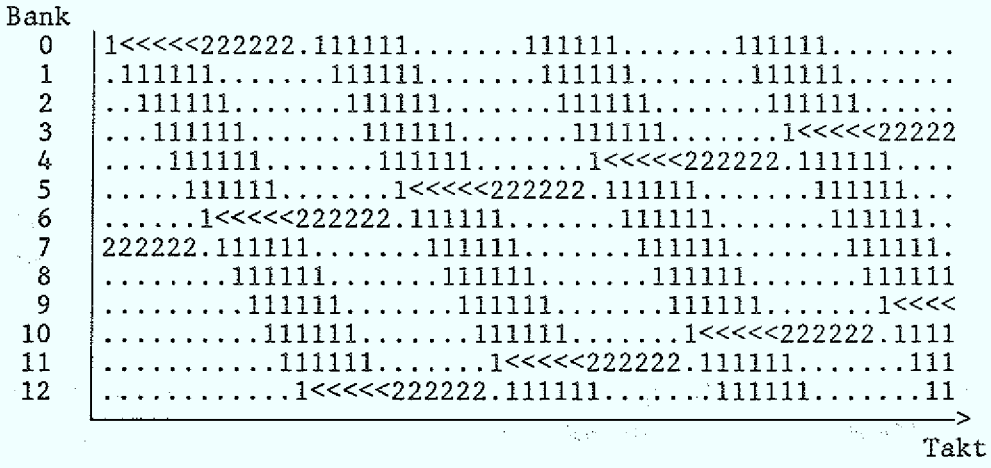


Abb.4.4 Barriere-Situation in einem Speicher mit $m=13$, $n_c=6$; $d_2=6$ wird ständig von $d_1=1$ verzögert; $b_1=0$, $b_2=7$.

In Abb.4.4 ist ein Beispiel für eine Barriere-Situation in einem dreizehnfach verschränkten Speicher mit $n_c=6$, $d_1=1$ und $d_2=6$ dargestellt. ("1" bzw. "2" bezeichnet die beiden Zugriffsströme und "<" jeweils eine Verzöge-

rung, die "2" durch "1" erfährt.) In diesem Beispiel erfolgt der jeweils nächste Zugriffswunsch des verzögerten Zugriffsstroms ("2") stets auf eine Bank, die noch durch den anderen Zugriffsstrom ("1") belegt ist und er wird in *jedem* seiner Zugriffswünsche verzögert.

Theorem 4.7: Sei $f=GGT(m,d_1,d_2)$; $r_1 \geq 2n_c$; $r_2 > n_c$; $Z_1 \cap Z_2 \neq \emptyset$; $d_1 | m$; $d_2 > d_1$. Es gibt Startbänke B_1 und B_2, so daß eine Barriere-Situation entsteht, falls

$$d_2/f = d_1/f + c + km/d_1, \quad 1 \leq c < n_c, \; k \in Z. \tag{4.38}$$

Beweis:

Sei $m'=m/f$, $d_1'=d_1/f$ und $d_2'=d_2/f$. Mit $d_1|m$ ist auch $d_1'|m'$ und daraus folgt, daß d_1' und d_2' teilerfremd sind. (Der Faktor f bewirkt wie im Fall von Theorem 4.5 eine Spreizung der "effektiven" Bankanzahl.)

Da die Zugriffsmengen nicht disjunkt sind, gibt es mindestens eine gemeinsame Bank, deren Adresse willkürlich zu *0* angenommen wird. Beginnen beide Zugriffsströme auf der Bank mit der Adresse *0* und wird der zweite Zugriffsstrom zunächst verzögert, so ergibt sich folgende Situation:

0	d_1'	...	$n_c d_1'$	$(n_c+1)d_1'$	...	$(n_c+d_1')d_1'$ mod m'
			0	d_2'	...	$d_1'd_2'$ mod m'

Die Adresse $d_1'd_2'$ *mod* m' stellt wegen der Teilerfremdheit zwischen d_1' und d_2' nach *0* die erste gemeinsame Adresse dar. Somit gilt für die zwischen *0* und $d_1'd_2'$ *mod* m' vorkommenden Adressen

$$\{d_2', 2d_2', \ldots, (d_1'-1)d_2'\} \bmod m' \notin Z_1. \tag{4.39}$$

Kommt die Adresse $d_1'd_2'$ *mod* m' in der Sequenz des ersten Zugriffsstroms in den n_c-*1* zurückliegenden Takten vor, d.h.

$$d_1'd_2' \bmod m' \in \{(n_c+d_1'-1)d_1', (n_c+d_1'-2)d_1', \ldots, (n_c+d_1'-(n_c-1))d_1'\} \bmod m', \tag{4.40}$$

so liegt eine Barriere-Situation vor. Dann wird nämlich der zweite Zugriffsstrom wiederum durch den ersten verzögert. Durch Umnumerierung der Bankadressen ergibt sich wieder die Ausgangssituation.

Wegen der Teilerfremdheit von d_1' und d_2' sowie $d_1'|m'$, läßt sich mit $m''=m'/d_1'$ Gl.(4.40) schreiben als

$$d_2' \in \{n_c+d_1'-1,\ n_c+d_1'-2,\ldots,n_c+d_1'-(n_c-1)\} \bmod m'', \qquad (4.41)$$

was sich weiter umformen läßt zu

$$d_2' \in \{d_1'+1,\ d_1'+2,\ldots,\ d_1'+n_c-1\} \bmod m''. \qquad (4.42)$$

Damit ergibt sich als Bedingung für die Barriere-Situation

$$d_2' = d_1' + c + km'',\quad 1 \le c < n_c,\ k \in Z. \qquad (4.43)$$

□

Korollar 4.7: Im Falle einer Barriere-Situation nach Gl.(4.38) verlaufen jeweils genau $d_1/f-1$ Zugriffe des zweiten Zugriffsstroms konfliktfrei, bevor wieder eine Verzögerung durch den ersten Zugriffsstrom erfolgt.

Korollar 4.8: Im Falle einer Barriere-Situation nach Gl.(4.38) wird jeder verzögerte Zugriffswunsch um $(d_2-d_1)/f$ Takte verzögert.

Da nach Korollar 4.7 $d_1/f-1$ Zugriffe konfliktfrei verlaufen, werden in d_1/f Takten $2d_1/f-1$ Zugriffe zugelassen. In den $(d_2-d_1)/f$ folgenden Takten werden $(d_2-d_1)/f$ Zugriffe des ersten, d.h. des unverzögerten, Zugriffsstroms und 1 Zugriff des zweiten, d.h. des verzögerten, Zugriffsstroms zugelassen. Somit werden innerhalb von $(d_2-d_1)/f+d_1/f=d_2/f$ Takten $2d_1/f-1+(d_2-d_1)/f+1=(d_2+d_1)/f$ Zugriffe zugelassen. Daraus ergibt sich die effektive Bandbreite für die Barriere-Situation zu

$$b_{eff} = 1 + d_1/d_2 \le 2. \qquad (4.44)$$

Gl.(4.44) zeigt auch, daß die effektive Bandbreite dann maximal wird, d.h. $b_{eff}=2$, wenn $d_1=d_2$ gegeben ist. Dieser Fall ist nach Korollar 4.4 konfliktfrei.

Korollar 4.9: Besteht zwischen den beiden Zugriffsstsrömen eine *feste* Prioritätenregelung, dann gilt Gl.(4.38) auch für $c=n_c$, falls der erste Zugriffsstrom über die höhere Priorität verfügt.

Bei entsprechenden Startbänken erfolgen in diesem Fall die jeweils zu Konflikten führenden Zugriffswünsche des zweiten Zugriffsstroms stets auf eine Bank, auf die auch *gleichzeitig* der erste Zugriffsstrom zugreifen möchte. Es entsteht dann ein Simultankonflikt, der aufgrund der höheren Priorität jedoch stets zugunsten des ersten Zugriffsstroms gelöst wird.

Die in Theorem 4.7 geforderten Bedingungen reichen jedoch nicht aus, um sicherzustellen, daß sich eine Barriere-Situation auch *unabhängig* von den relativen Startpositionen der beiden Zugriffsströme einstellt. Abb.4.5 zeigt, daß die Zugriffsströme des Beispiels aus Abb.4.4 in einen sich nicht auflösenden Doppelkonflikt (vgl. Def.3.25) geraten, falls $b_2=1$ gesetzt wird.

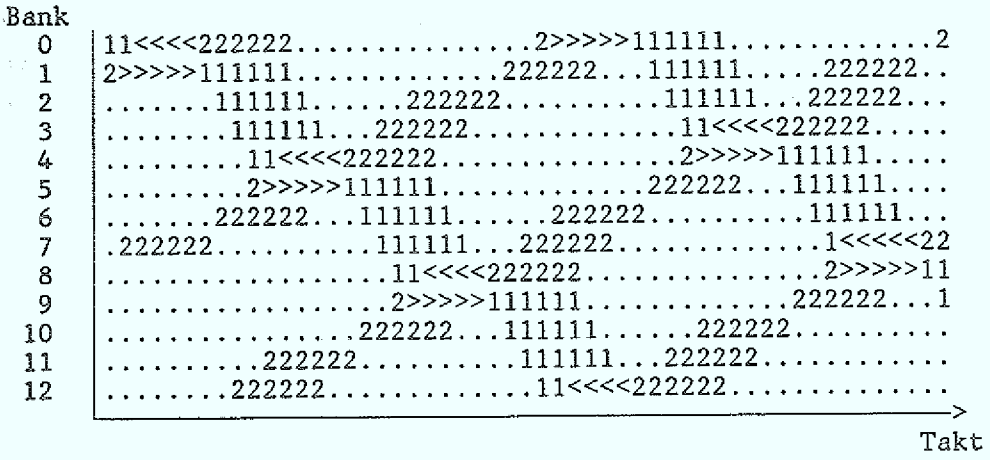


Abb.4.5 Doppelkonflikt in einem Speicher mit $m=13$, $n_c=6$; $d_1=1$, $d_2=6$, $b_1=0$, $b_2=1$; Barriere-Situation entsteht nicht

Da sich ein Doppelkonflikt nicht notwendigerweise auflöst, wie im Beispiel aus Abb.4.5, soll er für die weitere Analyse der Barriere-Situation ausgeschlossen werden.

Theorem 4.8: Sei $r_1 \geq 2n_c$; $r_2 > n_c$; $Z_1 \cap Z_2 \neq \emptyset$; $d_1 | m$; $d_2 > d_1$; dann ist ein Doppelkonflikt zwischen zwei Zugriffsströmen ausgeschlossen, falls

$$(n_c - 1) \cdot ((d_2 \bmod m/d_1) + d_1) < m. \qquad (4.45)$$

Beweis:

Sei $f=GGT(m,d_1,d_2)$, $m'=m/f$, $d_1'=d_1/f$ und $d_2'=d_2/f$. Mit $d_1 | m$ ist auch $d_1' | m'$ und daraus folgt, daß d_1' und d_2' teilerfremd sind.

Nach dem Start der beiden Zugriffsströme mit den Distanzen d_1' und d_2' ereigne sich der erste Konflikt auf der Bank mit der Adresse k. Es wird willkürlich angenommen, daß der Zugriffsstrom mit der Distanz d_1' dabei verzögert wird. Das "Auflaufen" des Zugriffsstroms auf die Bankadresse k kann innerhalb von n_c Takten erfolgen. Damit ergeben sich maximal n_c Positionen; von 0 bei gleichzeitigem Zugriff bis n_c-1, falls die Bank gerade

noch nicht freigegeben ist. Während der eine Zugriffsstrom blockiert wird, läuft der andere weiter.

Zur Vermeidung eines Doppelkonflikts muß nun sichergestellt werden, daß der Zugriffsstrom mit der Distanz d_2' in seinem weiteren Verlauf nicht auf eine Bank trifft, die noch von dem anderen Zugriffsstrom aufgrund seiner bereits getätigten Zugriffe belegt ist. Dies wird gewährleistet, falls der weiterlaufende Zugriffsstrom im nächsten Zugriff auf keine der Adressen $k-id_1'$, $i=1,2,\ldots,n_c-1$ zugreift, da diese Bänke noch von dem verzögerten Zugriffsstrom belegt sein können. Dieser Sachverhalt läßt sich in Abhängigkeit des Zeitpunkts des Auflaufens folgendermaßen darstellen (die Takte verlaufen von oben nach unten; von links nach rechts sind die möglichen Verzögerungen aufgetragen):

Verzögerung:	n_c	n_c-1	n_c-2	...	1
$k-(n_c-1)d_2'$	$k-(n_c-1)d_1'$				
$k-(n_c-2)d_2'$	$k-(n_c-2)d_1'$	$k-(n_c-1)d_1'$			
:	:	$k-(n_c-2)d_1'$	$k-(n_c-1)d_1'$		
$k-2d_2'$	$k-2d_1'$	:	$k-(n_c-2)d_1'$	.	
$k-d_2'$	$k-d_1'$	$k-2d_1'$	:	..	
k	k	$k-d_1'$	$k-2d_1'$	...	$k-(n_c-1)d_1'$
$k+d_2'$	..... k		$k-d_1'$	...	$k-(n_c-2)d_1'$
$k+2d_2'$	..... k			..	:
:				.	$k-2d_1'$
:					$k-d_1'$
$k+(n_c-1)d_2'$	..... k				

Der maximale Abstand wird erreicht, falls das "Auflaufen" eine Verzögerung um einen Takt bewirkt. Zur Vermeidung eines Doppelkonflikts ergibt sich somit für die Mindestanzahl der Bänke die obere Schranke

$$(n_c-1)\cdot(d_2' + d_1') < m'. \qquad (4.46)$$

Da in dieser Betrachtung lediglich die Bänke von Interesse sind, die in beiden Zugriffsmengen enthalten sind, muß aufgrund der obigen Voraussetzungen Gl.(4.46) nur für $d_2''=(d_2' \bmod m'/d_1')$ und nicht für d_2' erfüllt sein (vgl. Beweis Theorem 4.7), d.h.

$$(n_c-1)\cdot(d_2'' + d_1') < m'. \qquad (4.47)$$

Durch Multiplikation der Gl.(4.47) mit f ergibt sich Gl.(4.45).

□

In Abb.4.6 ist ein weiteres Beispiel einer Barriere-Situation dargestellt. Wegen $m=13$ und $n_c=4$ sowie $d_1=1$ und $d_2=3$ sind sowohl die Bedingungen des Theorems 4.7 als auch die des Theorems 4.8 erfüllt.

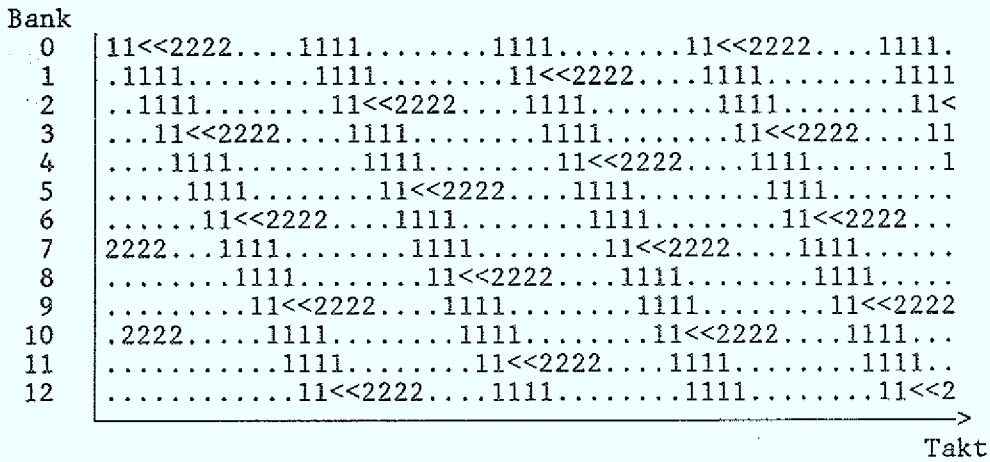


Abb.4.6 Barriere-Situation in einem Speicher mit $m=13$, $n_c=4$; $d_1=1$, $d_2=3$, $b_1=0$, $b_2=7$.

Allerdings zeigt Abb.4.7, daß sich die Barriere-Situation nicht eindeutig einstellt. Bei anderer Wahl der Startbänke "kippt" die Barriere-Situation dahingehend um, daß der Zugriffsstrom mit der größeren Distanz den mit der kleineren Distanz verzögert.

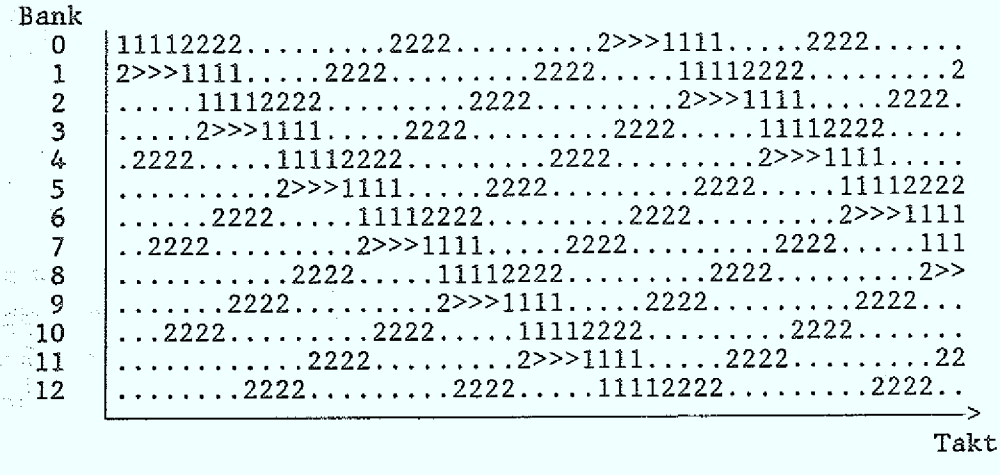


Abb.4.7 Umkippen der Barriere-Situation bei $m=13$, $n_c=4$; $d_1=1$, $d_2=3$, $b_1=0$, $b_2=1$.

In der Praxis sind die am meisten interessierenden Fälle diejenigen, für die sich *unabhängig* von der relativen Startposition die Barriere-Situation *eindeutig* einstellt. Da es nach Theorem 4.7 Startbänke gibt, so daß der

Zugriffsstrom mit der größeren Distanz von dem mit der kleineren Distanz verzögert wird, ist "eindeutig" gleichbedeutend damit, daß der Zugriffsstrom mit der größeren Distanz *stets* von dem mit der kleineren Distanz verzögert wird.

Theorem 4.9: Sei $r_1 \geq 2n_c$; $r_2 > n_c$; $Z_1 \cap Z_2 \neq \emptyset$; $d_1 | m$; $d_2 > d_1$. Ist Gl.(4.38) erfüllt, dann stellt sich unabhängig von den Startbänken die Barriere-Situation so ein, daß stets der Zugriffsstrom mit der Distanz d_2 verzögert wird, falls

$$(2n_c - 1)d_2 \leq m. \tag{4.48}$$

Beweis:

Sei $f=GGT(m,d_1,d_2)$, $m'=m/f$, $d_1'=d_1/f$ und $d_2'=d_2/f$. Mit $d_1 | m$ ist auch $d_1' | m'$ und daraus folgt, daß d_1' und d_2' teilerfremd sind.

Es wird der erste auftretende Konflikt betrachtet und dabei angenommen, daß der Zugriffsstrom mit der Distanz d_1' verzögert wird. Die Adresse der Bank, auf der dieser Konflikt stattfindet, sei willkürlich die Adresse *0*. In Abhängigkeit vom Zeitpunkt des "Auflaufens" des verzögerten Zugriffsstroms auf die Bank *0* lassen sich die Zugriffe folgendermaßen gegenüberstellen ("*" bezeichnet dabei eine Verzögerung jeweils um einen Takt):

0	d_2'	..	$(n_c-2)d_2'$	$(n_c-1)d_2'$	$n_c d_2'$	$(n_c+1)d_2'$	..	$(2n_c-1)d_2'$
$m-(n_c-1)d_1'$	$m-(n_c-2)d_1'$	..	$m-d_1'$	*	0	d_1'	..	$(n_c-1)d_1'$
$m-(n_c-2)d_1'$	$m-(n_c-3)d_1'$	..	*	*	0	d_1'	..	$(n_c-1)d_1'$
:	:	:	:	:	:	:	:	:
$m-d_1'$	*	..	*	*	0	d_1'	..	$(n_c-1)d_1'$

Da nach Voraussetzung $d_2' > d_1'$, ist $(2n_c-1)d_2' > (n_c-1)\cdot(d_2'+d_1')$, und somit ist ein Doppelkonflikt ausgeschlossen. Das bedeutet jedoch, daß während der Zugriffsstrom mit der Distanz d_1' noch blockiert ist, ein "Auflaufen" der Zugriffe im Bereich von d_2' bis $(n_c-1)d_2'$ auf eine der Adressen im Bereich von $m-(n_c-1)d_1'$ bis $m-d_1'$ nicht möglich ist.

Ein "Auflaufen" durch den Zugriffsstrom mit der Distanz d_2' auf Adressen im Bereich von $m-(n_c-1)d_1'$ bis $m-d_1'$ ist also frühestens möglich, sobald der Zugriffsstrom mit der Distanz d_1' auf die Bank mit der Adresse *0* zugreifen kann. Dafür kommen aber nur noch die Zugriffe im Bereich von $n_c d_2'$ bis $(2n_c-2)d_2'$ in Frage, da anschließend sämtliche Bänke aus dem Bereich von $m-(n_c-1)d_1'$ bis $m-d_1'$ wieder frei sind. Sollte ein "Auflaufen"

dieser Art stattfinden, so ist bereits die eindeutige Barriere-Situation eingetreten. Der Zugriffsstrom mit der Distanz d_1' kann wegen $d_1'|m'$ und $r_1 \geq 2n_c$ selbst nicht mehr auf diesen Bereich auflaufen.

Hat ein "Auflaufen" nicht stattgefunden, so muß lediglich sichergestellt werden, daß

$$(2n_c - 1)d_2' \leq m'. \tag{4.49}$$

Ist $(2n_c-1)d_2'=m'$, so ist $(2n_c-1)d_2' \equiv 0 \bmod m'$, d.h. der Zugriffsstrom mit der Distanz d_2' erfährt einen Konflikt, da die Bank mit der Adresse 0 noch belegt ist, und die eindeutige Barriere-Situation ist erreicht.

Ist $(2n_c-1)d_2'<m'$, so hat der Zugriffsstrom mit der Distanz d_1' seinerseits n_c Zugriffe getätigt, und ein "Überspringen" des Zugriffsstroms mit der Distanz d_1' durch den anderen Zugriffsstrom ist wegen der Voraussetzung $d_2'<n_c-d_1'$ (bzw. $(d_2' \bmod m'/d_1')<n_c-d_1'$ nach Beweis von Theorem 4.7) nicht mehr möglich.

Durch Multiplikation von Gl.(4.49) mit f erhält man Gl.(4.48).

□

Gl.(4.48) stellt eine obere Schranke für die Mindestanzahl von Bänken dar, für die das Erreichen einer eindeutigen Barriere-Situation garantiert ist. In Abhängigkeit der Distanzen d_1 und d_2 ist es u.U. möglich, auch bei weniger Bänken unabhängig von der relativen Startposition eine eindeutige Barriere-Situation zu erzielen.

Theorem 4.10: Sei $r_1 \geq 2n_c$; $r_2 > n_c$; $Z_1 \cap Z_2 \neq \emptyset$; $d_1|m$; $d_2 > d_1$. Sind Gl.(4.38) und Gl.(4.45) erfüllt, nicht jedoch Gl.(4.48), dann stellt sich unabhängig von den Startbänken die Barriere-Situation so ein, daß stets der Zugriffsstrom mit der Distanz d_2 verzögert wird, falls

$$kd_2 < (k - n_c)d_1 \bmod m,\ k = \lceil m/(d_1d_2) \rceil d_1 < 2n_c. \tag{4.50}$$

Beweis:

Sei $f=GGT(m,d_1,d_2)$, $m'=m/f$, $d_1'=d_1/f$ und $d_2'=d_2/f$. Mit $d_1|m$ ist auch $d_1'|m'$ und daraus folgt, daß d_1' und d_2' teilerfremd sind.

Zunächst gelten die gleichen Überlegungen wie im Beweis von Theorem 4.9. Findet ein "Auflaufen" des Zugriffsstroms mit der Distanz d_2' im Bereich der Adressen $m-(n_c-1)d_1'$ bis $m-d_1'$ statt, ist die eindeutige Barriere-Situation erreicht.

Da nach Voraussetzung $(2n_c-1)d_2'>m'$ bleibt jetzt noch zu untersuchen, wann ein Zugriff auf eine, beiden Zugriffsströmen gemeinsame Adresse durch den Zugriffsstrom mit der Distanz d_2' erfolgen soll, nachdem id_2', $i=n_c, n_c+1,\ldots, 2n_c-2$ größer als m' wird. Dies sei an der Stelle k' gegeben:

...	n_cd_2'	$(n_c+1)d_2'$	...	$k'd_2'$	...	$(2n_c-1)d_2'$	mod m'
*	0	d_1'	...	$(k'-n_c)d_1'$	...	$(n_c-1)d_1'$	mod m'

Befindet sich die Adresse $k'd_2'$ *mod* m' im Bereich von 0 bis $(k'-n_c-1)d_1'$, so läuft der Zugriffsstrom mit der Distanz d_2' auf eine durch den Zugriffsstrom mit der Distanz d_1' belegte Bank auf, und die eindeutige Barriere-Situation ist erreicht. Dies ist gleichbedeutend damit, daß die Adresse auf die der Zugriffsstrom mit der Distanz d_1' im gleichen Zeitpunkt zugreifen will, d.h. $(k'-n_c)d_1'$, größer als $k'd_1'$ (jeweils bezüglich *mod* m') ist, d.h.

$$k'd_2' < (k' - n_c)d_1' \bmod m'. \tag{4.51}$$

Da $d_1'|m'$ und lediglich die beiden Zugriffsströmen gemeinsamen Adressen von Interesse sind, können Konflikte ausschließlich auf Bänken stattfinden, deren Adressen Vielfache von $d_1'd_2'$ bezüglich *mod* m' sind. Daher ergibt sich die erste interessierende Adresse für

$$k' = \lceil m'/(d_1'd_2')\rceil d_1'. \tag{4.52}$$

Da $d_1'|m'$, gelten diese Überlegungen auch bei Multiplikation aller gestrichenen (') Größen mit f und mit Gl.(4.52) ergibt sich Gl.(4.50).

□

Korollar 4.10: Besteht zwischen den beiden Zugriffssts römen eine *feste* Prioritätenregelung und der erste Zugriffsstrom verfügt über die höhere Priorität, oder besteht zwischen den beiden Zugriffsströmen eine *zyklische* Priorität, dann stellt sich, bei Erfüllung der in Theorem 4.10 geforderten Voraussetzungen, die eindeutige Barriere-Situation auch dann ein, falls

$$kd_2 = (k - n_c)d_1 \bmod m,\ k = \lceil m/(d_1 d_2)\rceil d_1. \qquad (4.53)$$

Dieser Fall bedeutet einen Simultankonflikt zwischen den beiden Zugriffsströmen. Sobald er einmal zu Gunsten des ersten Zugriffsstroms gelöst worden ist, ist die eindeutige Barriere-Situation erreicht.

In Abb.4.8 ist ein Beispiel einer eindeutigen Barriere-Situation gezeigt, mit $m=12$, $n_c=3$, $d_1=2$, $b_1=10$, $d_2=3$ und $b_2=0$. Aufgrund dieser relativen Startpositionen wird der erste Zugriffsstrom ("1") zunächst einmal durch den zweiten Zugriffsstrom ("2") verzögert (">"). Anschließend wird jedoch die Barriere-Situation erreicht, d.h. der erste Zugriffsstrom verläuft ungestört, während der zweite in jedem zweiten Zugriff gemäß Korollar 4.7 verzögert wird ("<").

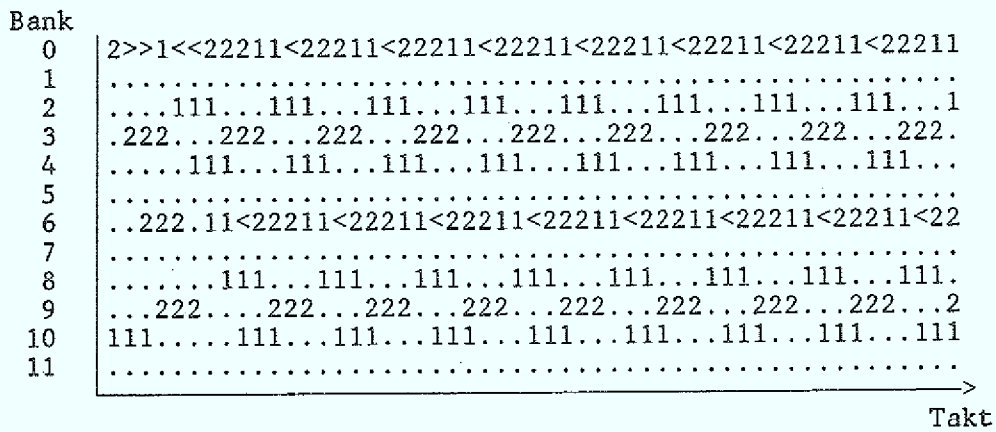
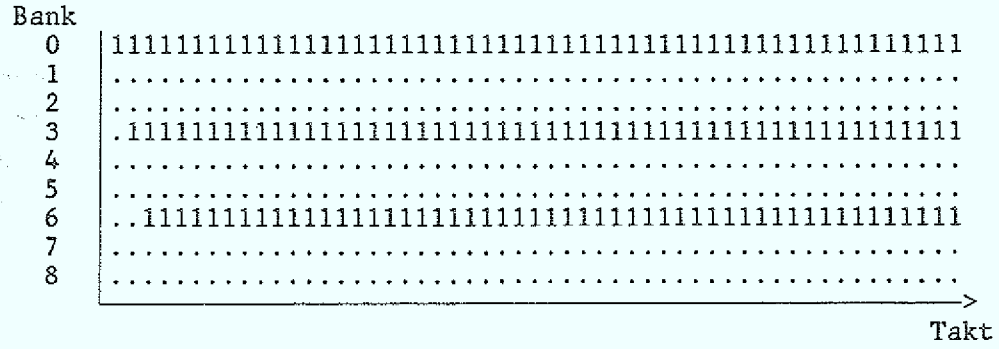


Abb.4.8 Barriere-Situation in einem Speicher mit $m=12$, $n_c=3$; $d_2=3$ wird von $d_1=2$ verzögert; $b_1=10$, $b_2=0$.

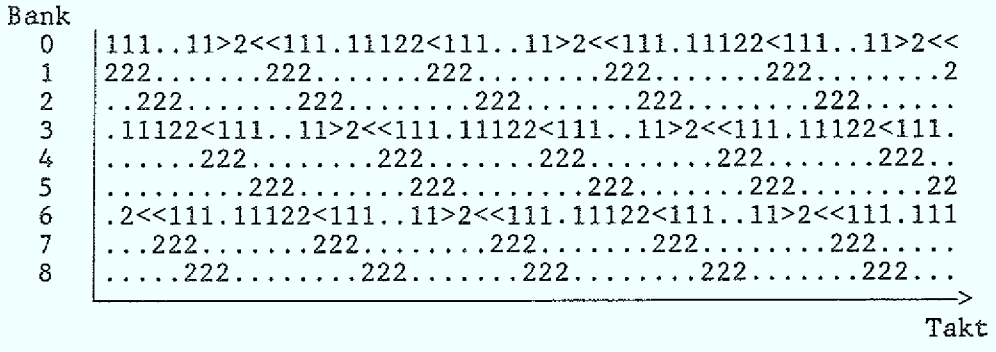
Sobald Zugriffsströme aufeinandertreffen, die nicht unter die bisher behandelten Fälle einzuordnen sind, hängt der weitere Verlauf sowie die effektive Bandbreite sowohl von der relativen Startposition als auch der geltenden Prioritätenregelung ab. Das folgende Beispiel $m=9$, $n_c=3$, $d_1=3$, $d_2=5$ soll das veranschaulichen.

Gilt eine feste Prioritätenregelung und hat der erste Zugriffsstrom die höhere Priorität gegenüber dem zweiten, so wird der zweite nie zugelassen, falls $b_1=b_2$. Die Rückkehrzahl ist mit $r_1=3$ so klein, daß der erste Zugriffsstrom immer wieder auf die Startbank zurückkehrt, wenn diese gerade frei geworden ist. Der dann zwischen den beiden Zugriffsströmen entstehende Simultankonflikt wird aufgrund der höheren Priorität des ersten Zugriffsstroms gegenüber dem zweiten stets zugunsten des ersten gelöst.

Beginnen beide Zugriffsströme jedoch versetzt gegeneinander, $b_2=b_1+1$, so fallen sie in einen Zyklus, in dem sich beide gegenseitig dergestalt verzögern, daß beide trotz der festen Priorität sowie der kleinen Rückkehrzahl des ersten Zugriffsstroms zugreifen können.



a) ein Zugriffsstrom kann niemals zugreifen: $b_1=b_2=0$; $b_{eff}=1$



b) beide Zugriffsströme können zugreifen: $b_1=0$, $b_2=1$; $b_{eff}=10/7$

Abb.4.9 unterschiedliches Zugriffsverhalten bei unterschiedlichen Startbänken konfliktbehafteter Zugriffsströme

In Abb.4.9 ist diese Situation dargestellt; Abb.4.9a zeigt den Fall, daß nur der erste Zugriffsstrom zugreifen kann; Abb.4.9b zeigt den Fall, daß beide zugreifen können und sich gegenseitig stören. ("1" bzw. "2" bezeichnet die beiden Zugriffsströme, "<" gibt eine Verzögerung des ersten durch den zweiten und ">" eine Verzögerung des zweiten durch den ersten an.)

Aufgrund der vielfältigen Möglichkeiten bei sich gegenseitig verzögernden Zugriffsströmen, lassen sich detailliertere Untersuchungen nicht mehr überschaubar darstellen. Zur Berechnung der effektiven Bandbreite solcher Fälle kann das in Abschn.4.4 entwickelte Modell verwendet werden.

4.3.2 WENIGER SEKTIONEN ALS BÄNKE

Aus Kostengründen ist es vor allem bei einer großen Anzahl von Bänken vorteilhaft, nicht zu jeder einzelnen Bank einen eigenen Zugriffspfad zu haben. Stattdessen werden mehrere Bänke zu einer Sektion zusammengefaßt, die über einen gemeinsamen Pfad erreicht werden, was zu Pfadkonflikten führen kann.

Im Prinzip ist die Zusammenfassung der Bänke beliebig. In der Praxis hat man jedoch in jeder Sektion die gleiche Anzahl von Bänken, d.h. $s|m$; dieser Fall soll daher ausschließlich Gegenstand der Untersuchungen sein. Weiterhin wird zunächst angenommen, daß die Bänke den Sektionen zyklisch zugeordnet sind (s. Def.3.5).

Im vorigen Abschnitt wurde der günstigste Fall betrachtet, daß die Anzahl s der Sektionen gleich der Anzahl m der Bänke ist und Pfadkonflikte somit von vornherein ausgeschlossen sind. Der ungünstigste Fall hingegen ist dadurch gegeben, daß alle Bänke zu einer Sektion zusammengefaßt werden, d.h. $s=1$.

In diesem Fall hätte es wenig Sinn, mehr als einen Port zur Verfügung zu haben, da in jedem Takt maximal ein Zugriffswunsch zugelassen werden kann. Ein Zugriffswunsch eines zweiten Ports könnte überhaupt nur dann zugelassen werden, wenn der andere wegen eines Bankkonflikts nicht zugreifen kann. Die maximale Bandbreite bleibt jedoch auf $b_w=1$ beschränkt. Eine Mindestforderung wird also sein, wenigstens so viele Sektionen bzw. Zugriffspfade wie Ports vorzusehen, d.h. $p \leq s$.

Auch hier interessiert im wesentlichen nur, ob der Zyklus, in den die beiden Zugriffsströme fallen, Pfadkonflikte beinhaltet oder nicht. Das Auflösen eines Pfadkonflikts erfolgt nach einer Prioritätenregelung und führt gemäß Def.3.23 zu einer Verzögerung um einen Takt, die Zeit, die ein Pfad belegt ist.

Im vorigen Abschnitt war gezeigt worden, daß Zugriffsströme, deren Zugriffsmengen disjunkt sind, konfliktfrei verlaufen. Das muß nicht mehr gegeben sein, falls mehrere Bänke zu jeweils einer Sektion zusammengefaßt werden. Werden beispielsweise in einem zwölffach verschränkten Speichersystem jeweils vier Bänke zu einer Sektion zusammengefaßt, d.h. $s=3$, so verlaufen die in Abb.4.2 dargestellten Zugriffsströme trotz disjunkter Zugriffsmengen nicht mehr konfliktfrei.

In Abb.4.10 ist dieser Fall einmal für eine feste Prioritätenregelung und einmal für eine zyklische Prioritätenregelung dargestellt. ("1" und "2" bezeichnen die beiden Zugriffsströme; die Zeile *Priorität* gibt an, welcher Zugriffsstrom gerade die höhere Priorität besitzt; "*" markiert eine Verzögerung aufgrund eines Pfadkonflikts.)

```
Priorität   11111111111111111111111111111111111111111111111111
Sektion Bank
   0  -  0  |111...111...111...111...111...111...111...111...
   1  -  1  |222..*222..*222..*222..*222..*222..*222..*222..*
   2  -  2  |.111...111...111...111...111...111...111...111..
   0  -  3  |.................................................
   1  -  4  |..111...111...111...111...111...111...111...111.
   2  -  5  |.*222..*222..*222..*222..*222..*222..*222..*222.
   0  -  6  |...111...111...111...111...111...111...111...111
   1  -  7  |.................................................
   2  -  8  |....111...111...111...111...111...111...111...11
   0  -  9  |...*222..*222..*222..*222..*222..*222..*222..*22
   1  - 10  |.....111...111...111...111...111...111...111...1
   2  - 11  |.................................................
             +-------------------------------------------------->
                                                          Takt
```

a) feste Priorität: $b_{eff}=3/2$

```
Priorität   11122211122211122211122211122211122211122211222
Sektion Bank
   0  -  0  |111.....111.....*111.....111.....*111.....111.
   1  -  1  |222.222*222222.222222.222*222222.222222.222*2222
   2  -  2  |.111.....*111.....111.....*111.....111.....*11
   0  -  3  |.................................................
   1  -  4  |..111.....*111.....111.....*111.....111.....*
   2  -  5  |.*222222.222222.222*222222.222222.222*222222.222
   0  -  6  |...*111.....111.....*111.....111.....*111.....
   1  -  7  |.................................................
   2  -  8  |.....*111.....111.....*111.....111.....*111...
   0  -  9  |...222222.222*222222.222222.222*222222.222222.22
   1  - 10  |.......111.....*111.....111.....*111.....111..
   2  - 11  |.................................................
             +-------------------------------------------------->
                                                          Takt
```

b) zyklische Priorität: $b_{eff}=3/2$

Abb.4.10 Pfadkonflikte beim Zugriff auf einen zwölffach verschränkten Speicher trotz disjunkter Zugriffsmengen

Die Voraussetzungen, unter denen bei disjunkten Zugriffsmengen zweier Zugriffsströme sowie einer kleineren Anzahl von Sektionen als Anzahl von Bänken weiterhin Konfliktfreiheit gewährleistet bleibt, werden durch folgendes Theorem beschrieben:

Theorem 4.11: Zwei Zugriffsströme mit disjunkten Zugriffsmengen Z_1 und Z_2 verlaufen dann und nur dann ohne Pfadkonflikte, falls

$$GGT(s,d_2-d_1) > 1. \tag{4.54}$$

Beweis:

Wird zunächst $f=GGT(s,d_1,d_2)=1$ angenommen, so sind trotz disjunkter Zugriffsmengen die beiden Pfadmengen nicht disjunkt. Damit liegt ein analoger Fall zu Theorem 4.5 vor, wobei in Gl.(4.28) m durch s ersetzt werden muß; f ist nach Vorraussetzung 1 und n_c ist ebenfalls 1, da die "Zykluszeit" für einen Pfad genau einen Takt beträgt. Ist Gl.(4.54) erfüllt, so entstehen also keine Pfadkonflikte.

Wird nun $GGT(s,d_1,d_2)>1$ angenommen, so entspricht dies dem Theorem 4.4, wobei in Gl.(4.14) m durch s ersetzt werden muß. Wegen $s|m$ sind in diesem Fall also nicht nur die beiden Zugriffsmengen disjunkt, sondern auch die beiden Pfadmengen. Dann ist aber Gl.(4.54) erfüllt.

□

Korollar 4.11: Falls $GGT(s,m,d_1,d_2)>1$ gegeben ist, sind bei relativer Verschiebung der Startbänke um 1 sowohl die Zugriffsmengen als auch die Pfadmengen disjunkt. Damit treten weder Bank- noch Pfadkonflikte auf, falls $r_1,r_2 \geq n_c$.

Korollar 4.12: Ist die relative Startposition der beiden Zugriffsströme ein Vielfaches von s und Gl.(4.54) erfüllt, so findet zu Beginn eine Synchronisation statt, d.h. der Zugriffsstrom mit der niedrigeren Priorität wird einmal um einen Takt verzögert.

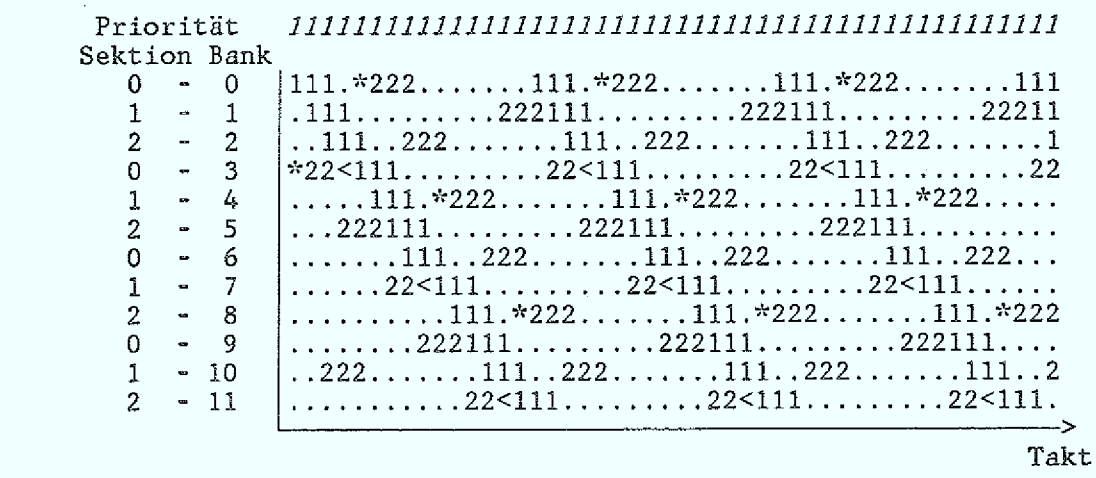
Bei Zugriffsströmen mit nichtdisjunkten Zugriffsmengen wird die Situation komplexer, da neben gemeinsamen Bänken zwangsläufig auch gemeinsame Sektionen bestehen.

Das Theorem 4.5 war unter der Voraussetzung bewiesen worden, daß die jeweiligen Startbänke die relative Position: $b_2=n_c d_1+b_1$ zueinander einnehmen. Daraus folgt jedoch, daß für den Fall

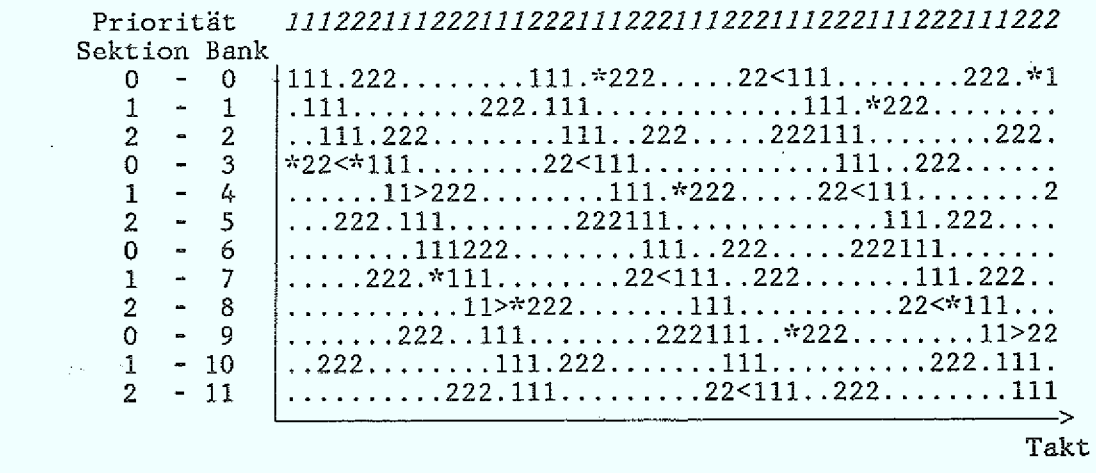
$$n_c d_1 = ks, \tag{4.55}$$

$k=1,2,3,\ldots$, zu Beginn ein Pfadkonflikt entsteht, da die beiden Startbänke innerhalb der gleichen Sektion liegen.

In Abb.4.11 ist das Beispiel aus Abb.4.3 dargestellt, mit $m=12$, $s=3$, $n_c=3$, $d_1=1$ und $d_2=7$. Trotz unterschiedlicher Prioritätenregelung treten nun Pfad- "*" und Bankkonflikte ("<": "1" ist durch "2" blockiert; ">": "2" ist durch "1" blockiert) gemeinsam auf. Da die beiden Konfliktarten sich wechselseitig bedingen, liegt in diesem Beispiel ein *Verbundkonflikt* vor.



a) feste Priorität: $b_{eff}=8/5$



b) zyklische Priorität: $b_{eff}=8/5$

Abb.4.11 Pfad- und Bankkonflikte (Verbundkonflikt) beim Zugriff auf einen zwölffach verschränkten Speicher

Andererseits läßt sich zeigen:

Theorem 4.12: Zwei Zugriffsströme mit nichtdisjunkten Zugriffsmengen verlaufen in einem m-fach verschränkten Speichersystem mit s Sektionen konfliktfrei, falls die in Theorem 4.5 geforderten Voraussetzungen und

$$n_c d_1 \neq ks, \tag{4.56}$$

$k=1,2,3,\ldots$, erfüllt sind.

Beweis:

Werden die Startbänke relativ um $n_c d_1$ gegeneinander verschoben, lassen sich die Zugriffsfolgen der beiden Zugriffsströme folgendermaßen übereinanderschreiben:

$$\begin{array}{llllll} n_c d_1 & , \; n_c d_1 + d_2 & , \; n_c d_1 + 2d_2 & , \; n_c d_1 + 3d_2 & , \; \ldots & \bmod m \\ 0 & , \; d_1 & , \; 2d_1 & , \; 3d_1 & , \; \ldots & \bmod m \end{array}$$

Die jeweilige Differenz ist $n_c d_1 + k(d_2 - d_1) \bmod m$, $k=0,1,2,\ldots$. Nach Gl.(4.29) läßt sich ein k_1 so finden, daß

$$g \equiv k_1(d_2 - d_1) \bmod m, \tag{4.57}$$

mit $g=GGT(m, d_2 - d_1)$. Allgemein gilt nach Gl.(4.30), daß die Differenzen Vielfache der minimalen Differenz g sind.

Mit der Wahl der Startbänke um relativ $n_c d_1$ zueinander ist jedoch ebenfalls eine minimale Differenz gewählt, d.h. die Bänke, auf die jeweils innerhalb eines Taktes ein Zugriff erfolgt, liegen um Vielfache von $n_c d_1$ auseinander. Ist nun $n_c d_1$ kein Vielfaches von s, so erfolgen die jeweils gleichzeitigen Zugriffe stets auf unterschiedliche Sektionen.

□

Korollar 4.13: Gibt es ein k mit $n_c d_1 = ks$, so ist $(n_c+1)d_1 \neq ks$. Damit läßt sich auch für den Fall, daß Gl.(4.56) nicht erfüllt ist, Konfliktfreiheit erreichen, falls

$$GGT(m/f, (d_2 - d_1)/f) \geq 2(n_c + 1), \tag{4.58}$$

mit $f=GGT(m, d_1, d_2)$ gegeben ist. Die Startbänke sind dann relativ um $(n_c+1)d_1$ zu verschieben.

Korollar 4.13 besagt, daß Konfliktfreiheit erreicht werden kann, falls "genügend Platz" für Zwischenräume besteht, um den zur Auflösung eines Pfadkonflikts benötigten Takt "unterzubringen". In Abb.4.12 ist ein zwölffach verschränktes Speichersystem mit zwei Sektionen und $n_c=2$ dargestellt. Die Distanzen $d_1=1$ und $d_2=7$ erfüllen zwar nicht Gl.(4.56), doch Gl.(4.58) und somit verlaufen die Zugriffsströme konfliktfrei.

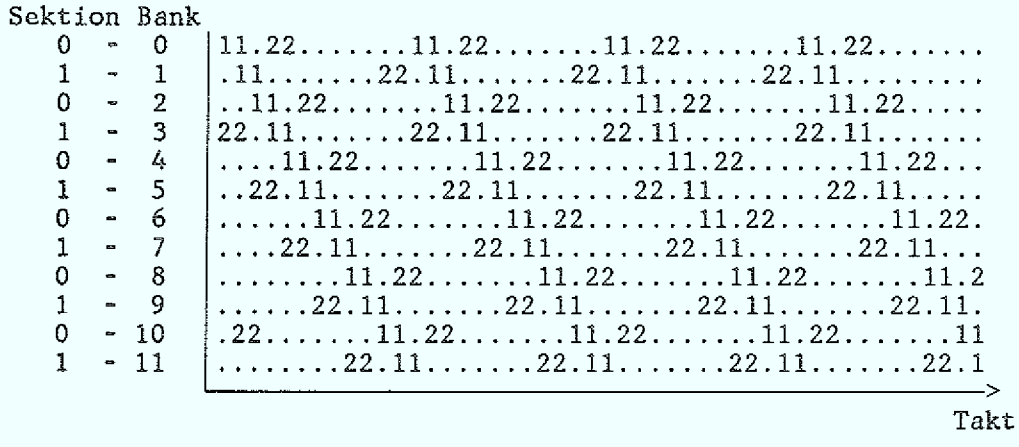


Abb.4.12 konfliktfreier Zugriff auf einen zwölffach verschränkten Speicher mit zwei Sektionen; $b_1=0$, $b_2=(n_c+1)d_1=3$

Während im Falle eines Speichersystems mit gleicher Anzahl von Sektionen wie Anzahl von Bänken, konfliktfreie Zyklen nichtdisjunkter Zugriffsmengen sich unabhängig von der relativen Startposition nach Theorem 4.6 von selbst einstellen, trifft dies für ein Speichersystem mit weniger Sektionen als Bänken nicht ohne weiteres zu.

In Abb.4.13 sind zwei Zugriffsströme mit $d_1=d_2=1$ in einem zwölffach verschränkten Speichersystem mit drei Sektionen und $n_c=3$ dargestellt. Abb.4.13a zeigt, wie die beiden Zugriffsströme aufgrund der festen Priorität sowie der Wahl der Startbänke in einen Verbundkonflikt fallen. Wird anstelle der festen Priorität eine zyklische Priorität gewählt, so wird dieser Verbundkonflikt durch "Auseinanderschieben" der beiden Zugriffsströme aufgelöst (Abb.4.13b). Damit wird eine durch Theorem 4.6 beschriebene Situation erreicht.

Ist Gl.(4.56) nicht erfüllt, kann ein Verbundkonflikt also nur vermieden werden, falls "genügend Platz" für Zwischenräume besteht. Das hat aber zur Folge, daß konfliktfreie Zyklen und eine maximale Speicherauslastung von $u_m=1$ sich gegenseitig ausschließen.

```
Priorität   111111111111111111111111111111111111111111111111
Sektion Bank
  0  -  0  |111.........222111.........222111.........22
  1  -  1  |2<<111.........*22<111.........*22<111.........*
  2  -  2  |.222111.........222111.........222111.........
  0  -  3  |..222111.........222111.........222111........
  1  -  4  |...*22<111.........*22<111.........*22<111.....
  2  -  5  |.....222111.........222111.........222111.....
  0  -  6  |......222111.........222111.........222111....
  1  -  7  |.......*22<111.........*22<111.........*22<111..
  2  -  8  |.........222111.........222111.........222111.
  0  -  9  |..........222111.........222111.........222111
  1  - 10  |...........*22<111.........*22<111.........*22<1
  2  - 11  |............222111.........222111.........222
            |-------------------------------------------->
                                                      Takt
```

a) feste Priorität: $b_{eff}=3/2$

```
Priorität   111222111222111222111222111222111222111222111222
Sektion Bank
  0  -  0  |111........222.111.....222.111.....222.111.....2
  1  -  1  |2<<*111.....222.111.....222.111.....222.111.....
  2  -  2  |.222.111.....222.111.....222.111.....222.111....
  0  -  3  |..222.111.....222.111.....222.111.....222.111...
  1  -  4  |...222.111.....222.111.....222.111.....222.111..
  2  -  5  |....222.111.....222.111.....222.111.....222.111.
  0  -  6  |.....222.111.....222.111.....222.111.....222.111
  1  -  7  |......222.111.....222.111.....222.111.....222.11
  2  -  8  |.......222.111.....222.111.....222.111.....222.1
  0  -  9  |........222.111.....222.111.....222.111.....222.
  1  - 10  |.........222.111.....222.111.....222.111.....222
  2  - 11  |..........222.111.....222.111.....222.111.....22
            |-------------------------------------------->
                                                      Takt
```

b) zyklische Priorität: $b_{eff}=2$

Abb.4.13 Einfluß der Prioritätenregelung auf das Konfliktverhalten

Eine andere Möglichkeit, Verbundkonflikte zu vermeiden wird in [CuS 84] vorgeschlagen; jeweils m/s aufeinanderfolgende Bänke werden in einer Sektion zusammengefaßt.

Für das Beispiel aus Abb.4.13 ist die dann entstehende Situation in Abb.4.14 dargestellt. ("*" bedeutet ein Pfadkonflikt und ">", daß "2" einen Bankkonflikt durch "1" erfährt; "1" habe die höhere Priorität.)

Der Vorteil dieser Methode liegt darin, daß Verbundkonflikte unabhängig von der Prioritätenregelung aufgelöst werden, sofern die beiden Zugriffsströme die Gl.(4.28) erfüllen. Insbesondere ermöglicht diese Methode für den Fall $n_c=s$ das Erreichen der maximalen Speicherauslastung von $u_m=1$.

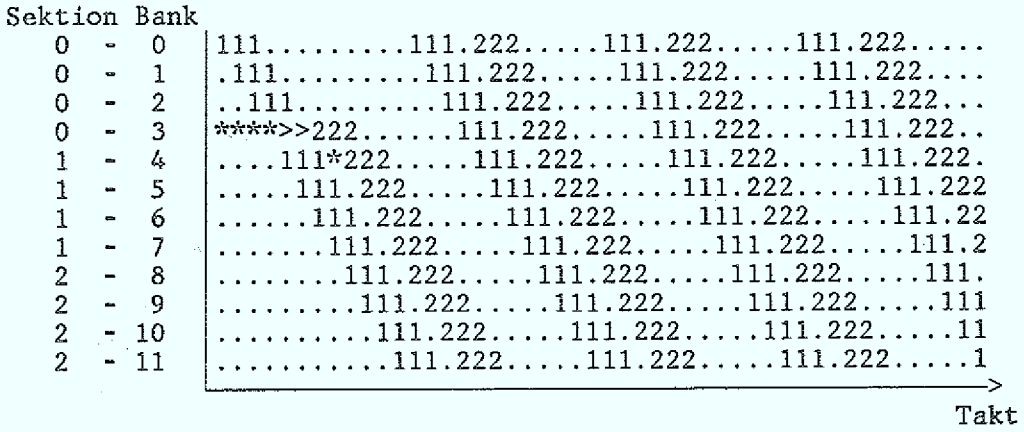


Abb.4.14 Vermeiden eines Verbundkonflikts durch andere Zuordnung der Bänke zu den Sektionen [CuS 84]

Nachteilig wird diese Methode bei kleiner Anzahl von Sektionen, da dann ein Zugriffsstrom eine Sektion relativ lange "festhält", wodurch die Einlaufphase in den Zyklus verlängert wird. Weiterhin ist es, abgesehen von dem ohnehin ungünstigen Fall, $d_I=m$, nicht möglich, disjunkte Pfadmengen zu erzielen.

4.4 GENERELLE BERECHNUNG DER EFFEKTIVEN BANDBREITE

Im vorigen Abschnitt sind für zwei Zugriffsströme die Bedingungen zur Erlangung konfliktfreier Zyklen sowie die wichtigsten Konfliktsituationen analysiert worden. Allerdings zeigte sich, daß die Analyse unter Berücksichtigung von Sektionen bereits recht komplex wird, insbesondere dann, wenn unterschiedliche Prioritätenregelungen sowie Methoden des Zusammenfassens der Bänke zu Sektionen betrachtet werden.

Für den allgemeinen Fall einer beliebigen Anzahl von Ports über die jeweils ein Zugriffsstrom mit einer beliebigen Distanz d_I läuft, ist eine überschaubare Analyse nicht mehr möglich. Deshalb wird ein einfaches Modell vorgeschlagen, mit dessen Hilfe die effektive Bandbreite sowie die Speicherauslastung einer gegebenen Konstellation ermittelt werden kann.

Dieses Modell besteht darin, beginnend im Zeitpunkt $t=0$, Takt für Takt ein Fünftupel $Q_t(W_t,D_t,V_t,X_t,t_p)$ zu erzeugen; (betrachtete Zeitpunkte t sind Vielfache der Taktzeit τ).

Die einzelnen Komponenten des Fünftupels haben dabei folgende Bedeutung:

- W_t ist ein Vektor der Länge p (*Bankvektor*). Die i-te Komponente w_i, $i=1,2,\ldots,p$, gibt die Adresse der Bank an, auf die der Zugriffswunsch des i-ten Zugriffsstroms im Zeitpunkt t erfolgt.

- D_t ist ein Vektor der Länge p (*Distanzvektor*). Die i-te Komponente, $i=1,2,\ldots,p$, ist die Distanz d_i des i-ten Zugriffsstroms.

- V_t ist ein Vektor der Länge m (*Restbelegungsvektor*). Die i-te Komponente v_i, $i=1,2,\ldots,m$, gibt die Anzahl der Takte an, die die Bank noch belegt ist, bevor wieder ein Zugriff zugelassen werden kann.

- X_t ist ein boolescher Vektor der Länge s (*Pfadvektor*). Die i-te Komponente x_i, $i=1,2,\ldots,s$, gibt an, ob zum Zeitpunkt t der Pfad in die i-te Sektion bereits belegt ist (x_i=*wahr*) oder nicht (x_i=*falsch*).

- t_p gibt die im Zeitpunkt t noch verbleibende Zeit bis zu einem Wechsel der Priorität an. Im Falle einer festen Prioritätenregelung wird im Zeitpunkt $t=0$ diese Zeit auf einen negativen Wert gesetzt und nicht mehr verändert.

Der Anfangszustand $Q_0(W_0,D_0,V_0,X_0,t_p)$ wird hergestellt, indem W_0 mit den p Startbänken, D_0 mit den p Distanzen, alle m Komponenten von V_0 mit 0 sowie alle s Komponenten von X_0 mit *falsch* initialisiert werden; t_p erhält im Falle einer zyklischen Priorität den Wert n_c, andernfalls den Wert -1.

Die Komponenten der Vektoren W_t und D_t sind in der Reihenfolge der Prioritäten der Ports geordnet, wobei der Port mit der höchsten Priorität an erster Stelle steht. Bei einem Wechsel der Priorität werden die beiden Vektoren entsprechend neu geordnet.

Mit Hilfe des in Abb.4.15 in einem Struktogramm dargestellten Algorithmus läßt sich ein Zyklus der Dauer $\Delta t = t_2 - t_1$ finden; er ist erreicht, falls

$$Q_{t_1} = Q_{t_2}. \tag{4.59}$$

Die Prozedur AUSWERTUNG ermittelt abschließend, wieviele Zugriffe im Zeitraum von 0 bis t_1 und von t_1 bis t_2 zugelassen worden sind.

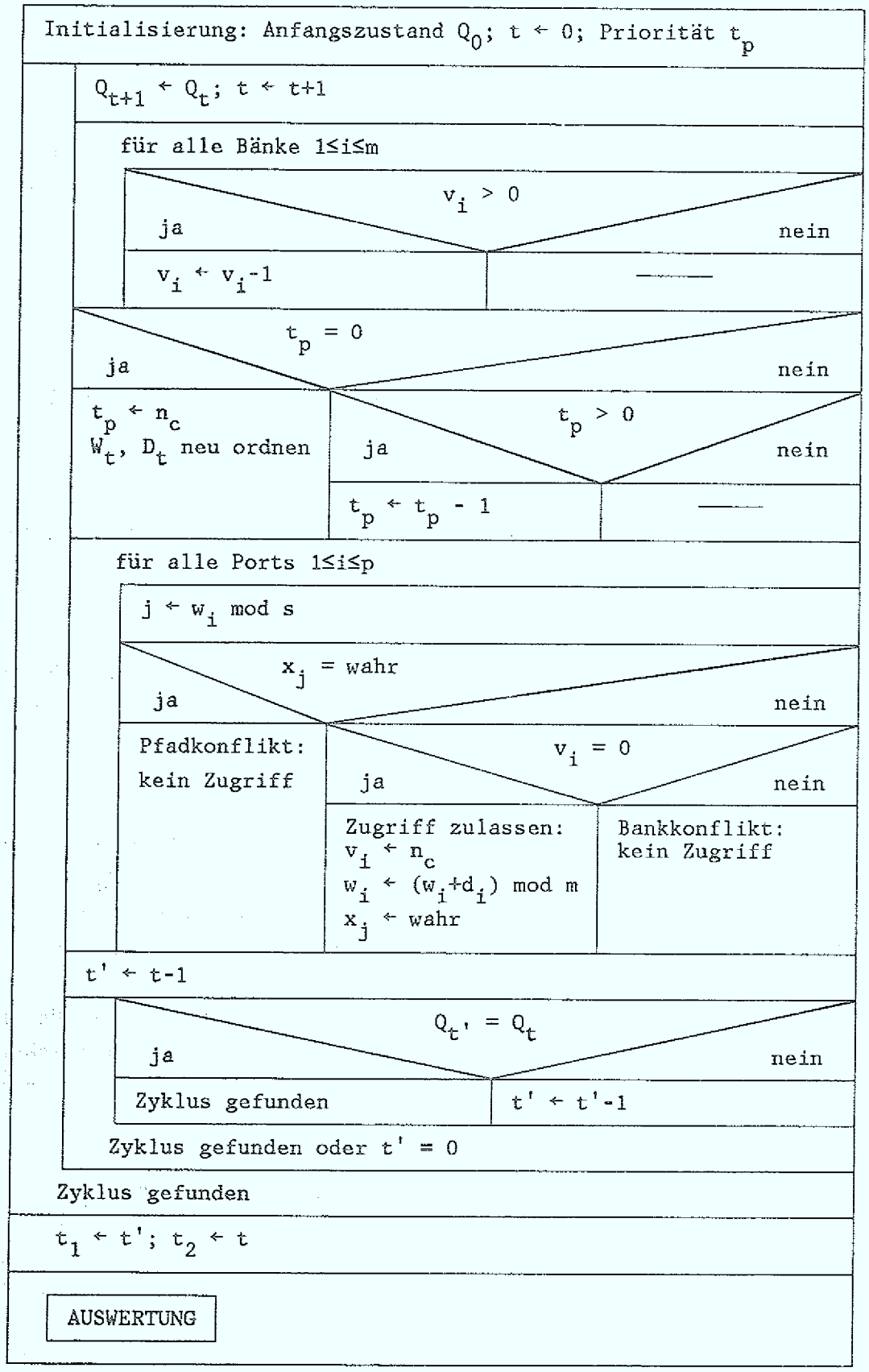


Abb.4.15 Struktogramm eines Algorithmus zum Auffinden von Zugriffszyklen auf ein *m*-fach verschränktes Speichersystem

Es läßt sich nun ein auf zwei Zustände, den *Ausgangszustand* Z_a und den *zyklischen Zustand* Z_z reduzierter Zustandsgraph angeben, in dem alle Zwischenzustände durch eine Kantenbewertung ersetzt worden sind. In Abb.4.16 ist der reduzierte Graph dargestellt, dessen beide Kanten mit der Anzahl der benötigten Takte zum Übergang vom einen in den anderen Zustand sowie der Anzahl der innerhalb des Übergangs zugelassenen Zugriffe bewertet sind. Dabei ist $n_{az}=t/\tau$ die Anzahl der benötigten Takte von Z_a nach Z_z und $n_{zz}=(t_2-t_1)/\tau$ die Anzahl der benötigten Takte zur Rückkehr nach Z_z; z_{az} ist die Anzahl der innerhalb von n_{az} und z_{zz} die Anzahl der innerhalb von n_{zz} zugelassenen Zugriffe.

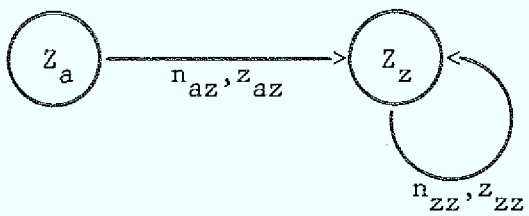


Abb.4.16 Reduzierter Zustandsgraph

Damit ergibt sich die effektive Bandbreite im zyklischen Zustand zu

$$b_{eff} = z_{zz}/n_{zz}, \tag{4.60}$$

woraus sich mit Gl.(3.3) die Speicherauslastung u_m berechnen läßt.

Tab.4.1 m=12, s=m, n_c=3, d_1=1 fest, b_1=0, b_2=1, feste Priorität

d_2	r_2	n_{zz}	z_{zz}	b_{eff}	u_m	Kommentar
0	1	12	15	1.25	31.25%	$r_2<n_c$
1	12	12	24	2.00	50.00%	konfliktfrei
2	6	12	18	1.50	37.50%	Barriere-Situation
3	4	12	16	1.33	33.33%	Barriere-Situation
4	3	12	15	1.25	31.25%	Barriere-Situation
5	12	16	24	1.50	37.50%	
6	2	14	18	1.29	32.14%	$r_2<n_c$
7	12	12	24	2.00	50.00%	konfliktfrei
8	3	12	18	1.50	37.50%	$r_2<2n_c$
9	4	16	24	1.50	37.50%	$r_2<2n_c$
10	6	12	18	1.50	37.50%	
11	12	16	24	1.50	37.50%	d_2=-1

Der oben beschriebene Algorithmus ist als FORTRAN-Programm implementiert worden und in Tab.4.1 bis Tab.4.3 sind die mit Hilfe dieses Programms ermittelten Ergebnisse für drei Beispiele wiedergegeben. Zum Vergleich mit den Untersuchungen des vorigen Abschnitts wird stets der Fall zweier Zugriffsströme betrachtet.

Das erste Beispiel besteht aus zwölf Läufen des Programms, deren jeweilige Ergebnisse in Tab.4.1 zusammengestellt sind. Das betrachtete Speichersystem ist mit $m=12$, $n_c=3$, $s=m$ sowie fester Prioritätenregelung gegeben. Der erste Zugriffsstrom hat stets die Distanz $d_1=1$ sowie die höhere Priorität, während der zweite Zugriffsstrom den Bereich $0 \le d_2 \le m-1$ durchläuft. Die Adressen der Startbänke sind stets $b_1=0$ und $b_2=1$.

Während sich die Barriere-Situation für $d_2=2$ und $d_2=3$ aufgrund der Theoreme 4.8 bis 4.10 eindeutig einstellen, liegt für den Fall $d_2=4$ lediglich eine geeignete Wahl der Startbänke vor.

Das zweite Beispiel unterscheidet sich vom ersten dadurch, daß mit $s=3$ jeweils vier Bänke zu einer Sektion zusammengefaßt sind (zyklische Zusammenfassung der Bänke). Wie die in Tab.4.2 zusammengestellten Ergebnisse zeigen, fallen die beiden konfliktfreien Paare beide in einen Verbundkonflikt. Für das Paar $d_1=1, d_2=1$ liegt der Grund hierfür in der ungünstigen Wahl der Startbänke, während das Paar $d_1=1, d_2=7$ die Gl.(4.58) nicht erfüllt. Die Barriere-Situation hingegen bleibt davon unberührt.

Tab.4.2 m=12, s=3, n_c=3, d_1=1 fest, b_1=0, b_2=1, feste Priorität

d_2	r_2	n_{zz}	z_{zz}	b_{eff}	u_m	Kommentar
0	1	13	15	1.15	28.85%	$r_2<n_c$
1	12	16	24	1.50	37.50%	Verbundkonflikt
2	6	12	18	1.50	37.50%	Barriere-Situation
3	4	12	16	1.33	33.33%	Barriere-Situation
4	3	12	15	1.25	31.25%	Barriere-Situation
5	12	16	24	1.50	37.50%	
6	2	16	18	1.13	28.13%	$r_2<n_c$
7	12	15	24	1.60	40.00%	Verbundkonflikt
8	3	27	36	1.33	33.33%	$r_2<2n_c$
9	4	18	24	1.33	33.33%	$r_2<2n_c$
10	6	33	48	1.45	36.36%	
11	12	27	36	1.33	33.33%	$d_2=-1$

Das dritte Beispiel schließlich unterscheidet sich vom zweiten durch eine zyklische anstelle einer festen Prioritätenregelung. Ein Vergleich der in Tab.4.3 zusammengestellten Ergebnisse mit den Ergebnissen der beiden anderen Tabellen zeigt, daß aufgrund der zyklischen Priorität der Verbundkonflikt des Paares $d_1=1, d_2=1$ aufgelöst wird, die anderen Paare hingegen meist zu einer geringeren effektiven Bandbreite führen.

Tab.4.3 m=12, s=3, n_c=3, d_1=1 fest, b_1=0, b_2=1, zyklische Priorität

d_2	r_2	n_{zz}	z_{zz}	b_{eff}	u_m	Kommentar
0	1	18	17	0.94	23.61%	$r_2<n_c$
1	12	12	24	2.00	50.00%	konfliktfrei
2	6	72	90	1.25	31.25%	
3	4	72	80	1.11	27.78%	$r_2<2n_c$
4	3	24	27	1.13	28.13%	$r_2<2n_c$
5	12	48	72	1.50	37.50%	
6	2	24	24	1.00	25.00%	$r_2<n_c$
7	12	30	48	1.60	40.00%	Verbundkonflikt
8	3	18	21	1.17	29.17%	$r_2<2n_c$
9	4	24	24	1.00	25.00%	$r_2<2n_c$
10	6	66	96	1.45	36.36%	
11	12	18	24	1.33	33.33%	d_2=-1

Der Vorteil einer zyklischen Priorität liegt also hauptsächlich darin, daß ein "fair share" zwischen den Zugriffsströmen gewährleistet wird. Ansonsten ist zur Verhinderung von Verbundkonflikten die Zusammenfassung von jeweils *m/s* hintereinanderliegender Bänke zu jeweils einer Sektion [CuS 84] die bessere Lösung.

5 ANALYSE DES ARBEITSSPEICHERS DES VEKTORRECHNERS CRAY X-MP

Hinsichtlich der Analyse des Speicherverhaltens eines speziellen Vektorrechnersystems sind die Systeme der CRAY X-MP Serie [CRA 84b] insofern interessant, als sie über mehrere, voneinander unabhängige Ports zum Arbeitsspeicher verfügen.

Dazu wurden auf dem im *Zentralinstitut für Angewandte Mathematik der Kernforschungsanlage Jülich GmbH* installierten Doppelprozessorsystem CRAY X-MP Messungen vorgenommen. Weiterhin wurde auf der Grundlage dieses Systems ein Simulator entwickelt, mit dessen Hilfe Simulationen durchgeführt werden können, die den Experimenten auf der realen Maschine entsprechen und Auskunft über Größen geben, die den Messungen unzugänglich sind. Die Ergebnisse der Messungen und Simulationen werden mit denen der im vorigen Kapitel entwickelten analytischen Modelle verglichen.

5.1 ARCHITEKTUR EINER CPU DES VEKTORRECHNERS CRAY X-MP

In diesem Abschnitt werden die wesentlichen Merkmale des Doppelprozessorsystems CRAY X-MP (Taktzeit τ=9.5*n*sec) vorgestellt soweit sie im Rahmen dieser Arbeit relevant sind. Die Architektur einer CPU des Doppelprozessorsystems CRAY X-MP ist in Abb.5.1 wiedergegeben [CRA 82].

5.1.1 ARBEITSSPEICHER

Der 2MWorte umfassende Arbeitsspeicher des Systems ist *16*-fach verschränkt, hat eine Bankzykluszeit von $t_c=4\tau$ (Speicherelemente in ECL-Bipolar Technologie) sowie eine Speicherzugriffszeit von $t_a=14\tau$ und ist in $s=4$ Sektionen unterteilt. Jede CPU verfügt über drei Ports, die für die CPU 0 jeweils mit *1,2* und *3* und für die CPU 1 jeweils mit *4,5* und *6* bezeichnet werden sollen. (In [CRA 82] werden die Ports einer CPU jeweils mit *A,B* und *C* bezeichnet. Die davon abweichende Bezeichnung wird jedoch zur besseren Unterscheidung und in Anlehnung an die Bezeichnungen des

vorangegangenen Kapitels gewählt.) Weiterhin verfügt jede CPU über je einen E/A-Port, der nicht weiter betrachtet wird.

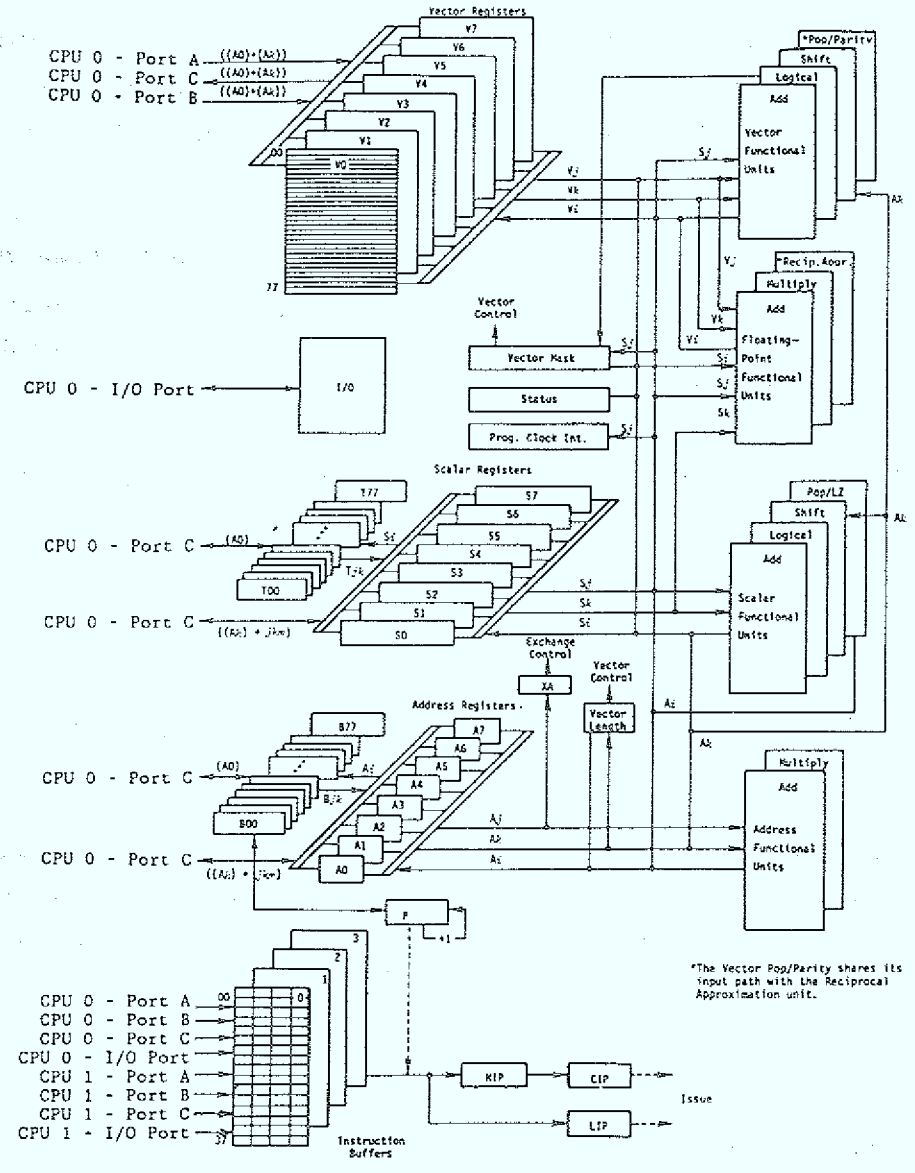


Abb.5.1 eine CPU des Doppelprozessorsystems CRAY X-MP [CRA 82] (Zahlenangaben oktal)

Mögliche Zugriffskonflikte in diesem System sind:

- Der *Bankkonflikt*, der durch Verzögerung des Zugriffswunsches im Port gelöst wird.
- Der *Simultankonflikt*, der dann erfolgt, falls *beide* CPUs im gleichen Takt auf die gleiche inaktive Bank zugreifen wollen. Die Auflösung des

Simultankonflikts erfolgt aufgrund einer zyklischen Priorität, die alle vier Takte zwischen den CPUs wechselt.

- Der *Pfadkonflikt*, der innerhalb einer CPU zwischen den drei Ports auftreten kann. Die Auflösung eines Pfadkonflikts erfolgt über eine Kombination aus fester Priorität, Bankkonflikt und Simultankonflikt. Es wird der Zugriffswunsch des Ports mit der höchsten Priorität ohne Bankkonflikt und ohne Simultankonflikt zugelassen.

Die feste Priorität zwischen den Ports innerhalb einer CPU wird folgendermaßen festgelegt:

Ports, die mit ungerader Distanz zugreifen wollen, haben grundsätzlich höhere Priorität gegenüber Ports, die mit gerader Distanz zugreifen wollen. Haben danach zwei oder mehr Ports die gleiche Priorität, so wird zwischen diesen Ports aufgrund des Startzeitpunkts unterschieden, wobei der frühere Startzeitpunkt die höhere Priorität erhält. Ein gleichzeitiger Startzeitpunkt ist innerhalb einer CPU, bedingt durch die sequentielle Instruktionsinitiierung (s. Abschn.5.1.2), nicht möglich.

Durch diese Regelung werden innerhalb einer CPU die Ports mit maximaler Rückkehrzahl gegenüber den Ports mit kleinerer Rückkehrzahl bevorzugt. Da es sich bei $m=16$ Bänken um eine Zweier-Potenz handelt, bedeutet eine ungerade Distanz die maximale Rückkehrzahl m.

Alle Ports, die in einem Takt einen Bankkonflikt erfahren, werden von jeder weiteren Konfliktauflösung innerhalb des gleichen Taktes ausgeschlossen. Bei der Auflösung eines Pfadkonflikts erhält jedoch nicht notwendigerweise der Port mit der höchsten Priorität den Zugriff, sondern der, der auch keinen Simultankonflikt erfährt.

An einem Beispiel soll diese Situation verdeutlicht werden. Mit $1(i)$ werde dabei der Port 1 bezeichnet, der einen Zugriff auf die i-te Bank tätigen will; entsprechendes gelte für die jeweils anderen Ports. Weiterhin bedeute $1(i)>2(j)$, daß der Port 1 gegenüber dem Port 2 die höhere Priorität besitzt.

1(0) > 2(4) > 3(8) - CPU 0
4(5) > 5(7) > 6(4) - CPU 1

Hat die CPU 0 die höhere Priorität gegenüber der CPU 1, so verfügt in diesem Beispiel insgesamt der Port 1 über die höchste und der Port 6 über die niedrigste Priorität.

Außer der Bank 0 seien alle anderen Bänke inaktiv. Dadurch erfährt der Port 1 einen Bankkonflikt und wird nicht weiter berücksichtigt.

Da die Bänke 4 und 8 sich innerhalb der gleichen Sektion befinden, ergibt sich zwischen den verbleibenden Ports der CPU 0 ein Pfadkonflikt; zwischen den Ports der CPU 1 ergeben sich keine Pfadkonflikte. Weiterhin besteht ein Simultankonflikt zwischen dem Port 2 und dem Port 6.

Würde ausschließlich auf der Grundlage der Prioritäten über die Zulassung eines Zugriffs entschieden, so würden die Ports 2,4 und 5 den Zugriff erhalten. Der Port 3 erhielte keinen Zugriff aufgrund eines Pfadkonflikts mit Port 2; entsprechend erhielte der Port 6 keinen Zugriff aufgrund eines Simultankonflikts mit dem Port 2.

Die im System CRAY X-MP getroffene kombinierte Regelung ermöglicht jedoch den Zugriff aller Ports der CPU 1. In der CPU 0 wird anstelle des Ports 2 trotz niedrigerer Priorität der Port 3 zugelassen, da er auf keinen Simultankonflikt trifft. Diese Methode stellt einerseits sicher, daß kein Port in die Situation gelangen kann, niemals zugreifen zu können, andererseits daß pro Takt möglichst viele Zugriffe auf freie Bänke zugelassen werden.

5.1.2 INSTRUKTIONSTEIL

Der Instruktionsteil des Systems CRAY X-MP besteht aus dem Instruktionspuffer, dem Programmzähler und dem Instruktionsdekodierer.

Bevor eine Instruktion ausgeführt werden kann, muß sie sich in einem der vier Instruktionspuffer, über die jede CPU verfügt, befinden. Sie werden dorthin aus dem Arbeitsspeicher unter Verwendung aller acht verfügbaren Ports des Gesamtsystems geladen. Jeder der Instruktionspuffer kann 32 Speicherwörter aufnehmen, wobei ein Wort in vier Parzellen (*Parcels*) unterteilt ist. Die Instruktionen sind entweder *1-Parcel* (16 Bits)- oder *2-Parcel* (32 Bits)-Instruktionen.

Der Programmzähler *P* (*Programm Address Register*) zeigt auf das nächste Parcel, das zur Ausführungsvorbereitung in das 16 Bit lange NIP-Register (*Next Instruction Parcel Register*) geleitet wird.

Das 16 Bit lange CIP-Register (*Current Instruction Parcel Register*) enthält die Instruktion, die auf ihre Initiierung (*issue*) wartet. Handelt es sich um eine 2-Parcel-Instruktion, so befindet sich der zweite (*untere*) Teil der Instruktion im 16 Bit langen LIP-Register (*Lower Instruction Parcel Register*). 1-Parcel-Instruktionen können mit einer Rate von einer Instruktion pro Takt, 2-Parcel-Instruktionen mit jedem zweiten Takt initiiert werden.

Folgende Konfliktsituationen verhindern eine Initiierung von Instruktionen:

- Die benötigte *Funktionseinheit* ist von einer laufenden Instruktion belegt.
- Das benötigte *Resultatregister* wird von einer laufenden Instruktion als *Operandenregister* oder als *Resultatregister* benutzt.
- Ein benötigtes *Operandenregister* wird von einer laufenden Instruktion als *Operandenregister* benutzt.
- Ein benötigter *Datenpfad* ist von einer laufenden Instruktion belegt.

Die Konflikte werden von der Hardware durch *Reservierungen* der entsprechenden Komponenten gelöst. Liegt für eine benötigte Komponente eine Reservierung vor, wird die Instruktion bis zum Ende der Konfliktsituation im CIP-Register zurückgehalten. Damit werden auch alle nachfolgenden Instruktionen entsprechend verzögert.

5.1.3 ADREßTEIL

Der Adreßteil einer CPU des Systems CRAY X-MP besteht aus 8 24-Bit langen *A-Registern*, 64 24-Bit langen *B-Registern*, dem 24-Bit langen XA-Register (*Exchange Control Register*), dem 8-Bit langen VL-Register (*Vector Length Register*) sowie den beiden Adreßfunktionseinheiten *Add* (2-Segment Pipeline) und *Multiply* (4-Segment Pipeline), die ausschließlich 24-Bit Inte-

gergrößen aus den A-Registern verarbeiten. Die B-Register sind lediglich zur Zwischenspeicherung verwendbar.

Die benötigten Operanden werden aus dem Arbeitsspeicher in die A-Register oder B-Register mit einer Rate von maximal einem Operanden pro Takt geladen, wobei von den 64 Bits die 24 niederwertigen (0-23) übertragen werden.

Dabei wird ausschließlich der Port 3 bzw. 6 einer CPU verwendet. Weiterhin müssen alle Aktivitäten auf den anderen Ports der betreffenden CPU beendet sein, um mögliche Abhängigkeitsrelationen [NAG 84] mit Daten laufender Speicherzugriffe im Vektormodus nicht zu verändern (*Speicherzugriff im Skalarmodus*).

Die einzig mögliche Konfliktsituation ist dadurch gegeben, daß ein benötigtes Operandenregister in einer vorherigen Instruktion als Resultatregister angegeben ist und dieses Ergebnis noch nicht vorliegt.

5.1.4 SKALARTEIL

Der Skalarteil einer CPU des Systems CRAY X-MP besteht aus einem 8-Bit langen *Status Register*, 8 64-Bit langen *S-Registern*, 64 64-Bit langen *T-Registern* sowie den Funktionseinheiten *Integer Add* (3-Segment Pipeline), *Logical* (1 Segment), *Shift* (3-Segment Pipeline) und *Population/ Leading Zero Count* (4-Segment Pipeline). Die benötigten Operanden werden aus dem Speicher in die S-Register oder T-Register mit einer Rate von maximal einem Operanden pro Takt geladen. Gearbeitet wird ausschließlich auf den S-Registern, die T-Register sind lediglich zur Zwischenspeicherung verwendbar.

Die Speicherzugriffe erfolgen im Skalarmodus, d.h. es wird ausschließlich der Port 3 bzw. 6 einer CPU verwendet und alle Aktivitäten auf den anderen Ports der betreffenden CPU müssen beendet sein.

Konfliktsituationen sind zum einen dadurch gegeben, daß ein benötigtes Operandenregister in einer vorherigen Instruktion als Resultatregister angegeben ist und dieses Ergebnis noch nicht vorliegt, zum anderen dadurch, daß eine benötigte Gleitkommafunktionseinheit von einer laufenden Vektorinstruktion belegt ist.

5.1.5 VEKTORTEIL

Der Vektorteil einer CPU des Systems CRAY X-MP besteht aus einem 64-Bit langen VM-Register (*Vector Mask Register*), 8 Sätzen zu je 64 64-Bit langen V-Registern (*Vector Register*) sowie den Funktionseinheiten *Integer Add* (3-Segment Pipeline), *Logical* (2-Segment Pipeline), *Shift* (4-Segment Pipeline) und *Population/Leading Zero Count* (6-Segment Pipeline).

Hinzu kommen noch die Gleitkommafunktionseinheiten *Add* (6-Segment Pipeline), *Multiply* (7-Segment Pipeline) und *Reciprocal Approximation* (14-Segment Pipeline), in der der Kehrwert eines Wertes *y* in drei Iterationsschritten nach der Newtonschen Fixpunktmethode auf 30 Bit-Stellen genau approximiert wird. Zur vollen Genauigkeit von 48 Bit-Stellen sind zwei Nachiterationen im Multiplizierer nötig.

Die Gleitkommafunktionseinheiten werden auch vom Skalarteil mitbenutzt. Allerdings nimmt der Skalarteil keine Reservierung dieser Einheiten vor, was der Vektorteil zur Vermeidung von Konflikten für mindestens so viele Takte wie die aktuelle Vektorlänge tun muß.

Die benötigten Operanden werden mit einer Rate von maximal zwei Operanden pro Takt aus dem Speicher in die V-Register geladen. Parallel dazu kann noch ein Operand pro Takt in den Speicher geschrieben werden. Für die Speicherzugriffe im Vektormodus verwendet die CPU 0 die Ports *1* und *2* zum *Laden* und den Port 3 zum *Speichern*. Entsprechend verwendet die CPU 1 die Ports *4,5* zum Laden und den Port *6* zum Speichern. Somit beträgt die maximale Bandbreite im Vektormodus $b_w=3$ pro CPU.

Gearbeitet wird ausschließlich auf den V-Registern. Da ein Vektorregister lediglich aus einem Satz von 64 Elementen besteht, kann eine einzelne Vektoroperation für maximal 64 Elemente durchgeführt werden. Bei größeren Vektorlängen müssen entsprechend viele Vektoroperationen auf Teilvektoren zu je 64 Elementen sowie gegebenenfalls einem Restvektor kleinerer Länge ausgeführt werden. Die *aktuelle* Vektorlänge wird jeweils zum Zeitpunkt der Initiierung einer Vektorinstruktion dem VL-Register entnommen.

Der CRAY-FORTRAN-Compiler erzeugt den Code so, daß der Restvektor zu Beginn bearbeitet wird, d.h. aus der FORTRAN-Schleife

```
      DO 1 I = 1,100
    1 C(I) = A(I) + B(I)
```

wird prinzipiell folgender Code erzeugt:

```
        VL ← 100 mod 64
  $1A   lade A
        lade B
        C ← A + B
        speichere C
        falls ENDE springe nach $1B
        VL ← 64
        springe nach $1A
  $1B   ...
```

Mit "ENDE" wird hierbei symbolisch angedeutet, daß entsprechende Instruktionen ausgeführt werden, um festzustellen, ob die vorgegebene Vektorlänge vollständig abgearbeitet ist. Die Bearbeitung eines Vektors der Länge n erfordert also $\lceil n/64 \rceil$ Schleifendurchläufe.

Mögliche Konfliktsituationen sind dadurch gegeben, daß ein benötigtes Operandenregister in einer vorherigen Instruktion als Resultatregister angegeben ist, und dieses Ergebnis noch nicht vorliegt, eine benötigte Funktionseinheit von einer laufenden Vektorinstruktion belegt ist, oder ein benötigter Datenpfad belegt ist.

Bei den Vektorregistern handelt es sich um Registersätze. Jeder Registersatz verfügt über zwei *Zeiger*, von denen der eine auf das aktuelle Element beim Lesen des Vektorregisters und der andere auf das aktuelle Element beim Schreiben des Vektorregisters zeigt. Diese beiden Zeiger ermöglichen es, daß bei Registerkonflikten die entsprechende Instruktion nicht im CIP-Register zurückgehalten werden muß, sondern vor Ort warten kann. Somit können weitere Instruktionen iniitiert werden, sofern ihrerseits keine Konflikte vorliegen.

Zur Verbindung der Funktionseinheiten untereinander ist die Verkettung, nicht aber die Rückkopplung möglich (vgl. Kap.2). Die Verkettung von Funktionseinheiten geschieht automatisch über die ganze Kette Arbeitsspeicher - Register - Funktionseinheit - Register - Arbeitsspeicher, wobei auch mehrere Funktionseinheiten involviert sein können. Die Verkettung wird auch bei Zugriffskonflikten auf den Arbeitsspeicher nicht verhindert, es kommt lediglich zu einem "Stottern" innerhalb der Funktionseinheiten, da die Operanden nicht mit jedem Takt eintreffen.

5.2 MESSUNG UND SIMULATION DES SPEICHERVERHALTENS

In diesem Abschnitt werden die verwendeten Hilfsmittel und Methoden vorgestellt, mit denen das Zugriffsverhalten auf den Arbeitsspeicher des Vektorrechners CRAY X-MP analysiert worden ist.

5.2.1 MEßMETHODEN

Die derzeit einzige Möglichkeit, software-mäßig an dem System CRAY X-MP eine Messung durchzuführen, ist die Messung der Zeit. Dazu kann einerseits die Realzeit (*wall-clock-time*) und andererseits die zur Ausführung eines bestimmten Code-Stücks benötigte CPU-Zeit (*CPU-time*) gemessen werden.

Da Rückschlüsse auf den Speicherzugriff gezogen werden sollen, kommt in diesem Falle ausschließlich die Messung der verbrauchten CPU-Zeit in Frage. Das geschieht mit Hilfe der CRAY-Bibliotheksroutine SECOND [CRA 84c], die beim jeweiligen Aufruf die bis dahin seit Beginn der Ausführung des Programms verbrauchte CPU-Zeit in Sekunden angibt:

```
CALL SECOND(T1)
CALL SECOND(T2)
T0 = T2 - T1
CALL SECOND(T1)
   ...
   ...          zu messendes Code-Stück
   ...
CALL SECOND(T2)
TG = T2 - T1 - T0
```

Mit *T0* wird der Eigenbedarf der Routine SECOND berücksichtigt, der von der gemessenen Zeitdifferenz zwischen dem Beginn des zu messenden Code-Stücks und dessen Ende abgezogen wird, um somit die tatsächlich benötigte Ausführungszeit *TG* zu erhalten; vgl. Gl.(2.1).

Da die Vergabe der Betriebsmittel durch das Betriebssystem erfolgt, müssen besondere Vorkehrungen getroffen werden, damit die Meßergebnisse möglichst genau sind. Der wichtigste Gesichtspunkt ist dabei die Reproduzierbarkeit eines Experiments, was in einem Doppelprozessorsystem unter

der normalen Arbeitslast nicht möglich ist. Die Messungen müssen daher im Testbetrieb auf einer dedizierten Maschine durchgeführt werden.

Allerdings besteht auch dann keine Möglichkeit, beispielsweise den gleichzeitigen Beginn zweier Programme auf jeweils einer CPU sicherzustellen. Die Grundlage der Experimente wird also stets die Messung der Ausführungszeit der interessierenden Code-Stücke auf lediglich einer CPU sein. Gleichzeitig muß gewährleistet werden, daß währenddessen auf der anderen CPU ein Programm läuft, das ein gewünschtes Zugriffsverhalten besitzt, bzw. daß währenddessen keinerlei Zugriffe auf den Speicher erfolgen. Letzteres wird am einfachsten dadurch erreicht, daß zunächst ein "Dummy-Programm" der Form

```
1  GOTO 2
2  GOTO 1
```

zur Ausführung an das Betriebssystem abgeschickt wird. Nachdem dieses Programm läuft, wird das zu messende Programm abgeschickt, dem vom Betriebssystem die andere, noch freie CPU zugeteilt wird. Sobald die Messung beendet ist, kann die Dummy-Routine abgebrochen werden, was entweder von der Operateurkonsole aus oder durch das Betriebssystem bei Überschreiten der in der Job-Karte angegebenen CPU-Zeitschranke erfolgt. Entsprechend werden auch die Programme, die ein bestimmtes Zugriffsmuster auf den Speicher besitzen über das Betriebssystem an eine CPU abgeschickt, um somit eine bestimmte Umgebung für das zu messende Code-Stück herzustellen.

Als Anhaltspunkt zur Brauchbarkeit eines durchgeführten Experiments dienen schließlich der Mittelwert und die Standardabweichung der Zeiten, die sich jeweils bei mehrfacher Ausführung des zu messenden Code-Stücks ergeben.

5.2.2 STRUKTUR UND IMPLEMENTATION DES SIMULATORS

Während Untersuchungen im realen System auf Zeitmessungen beschränkt bleiben, ermöglicht die Methode der Simulation eine detailliertere Analyse von Speicherzugriffen. Bei hinreichend genauer Modellierung des zugrunde liegenden Systems lassen sich die auftretenden Zugriffskonflikte aufzeichnen sowie die effektive Bandbreite und die Speicherauslastung be-

rechnen. Darüberhinaus können Voraussagen über die Auswirkungen von Modifizierungen des Systems getroffen werden.

Zur Untersuchung des Zugriffsverhaltens auf den Arbeitsspeicher eines Rechnersystems vom Typ CRAY X-MP ist ein Takt-gesteuertes Simulationsmodell entwickelt und in FORTRAN-77 implementiert worden. Der Vorteil der Taktsteuerung (*time-driven simulation*) gegenüber der Ereignissteuerung (*event-driven simulation*) liegt darin, daß die in jedem Takt auftretenden Bank-, Pfad- und Simultankonflikte aufgezeichnet werden können. Nachteilig dagegen ist der große Aufwand, der sich in entsprechend langer Ausführungszeit niederschlägt. Erfahrungsgemäß ist für den hier beschriebenen Simulator bei Ausführung auf einem Großrechner (etwa CRAY X-MP oder IBM 3081, die aufgrund der zahlreichen, prinzipiell nicht vektorisierbaren, logischen Ausdrücke des Simulationsprogramms in etwa die gleiche CPU-Zeit zur Ausführung des Simulationsprogramms benötigen) ein Faktor von 10^3 bis 10^6 für die Rechenzeit zu veranschlagen. Das bedeutet, daß die Simulation eines Programms, das im realen System 1µsec CPU-Zeit benötigt, u.U. eine Sekunde CPU-Zeit eines Großrechnersystems erfordert. Die Simulation umfangreicher Programme ist somit kaum möglich, was für die Untersuchungen dieser Arbeit allerdings auch nicht nötig ist.

Die im Simulationsmodell realisierbare Systemkonfiguration lehnt sich an die in Abschn.5.1 beschriebene Architektur eines Vektorrechnerssystems vom Typ CRAY X-MP an und bietet folgende Möglichkeiten:

- Das System kann aus einer beliebigen Anzahl identischer CPUs bestehen.

- Der Arbeitsspeicher kann in beliebig viele Bänke mit beliebig vielen Sektionen im Bereich von $1 \le s \le m$ unterteilt sein. Die Bänke sind den Sektionen zyklisch zugeordnet. Die Anzahl der benötigten Takte n_c für einen Bankzyklus sowie n_a für einen Speicherzugriff sind frei wählbar.

- Von jeder CPU aus ist der Zugriff zum Arbeitsspeicher über drei Ports möglich. Bei Zugriffskonflikten, deren Auflösung durch eine Prioritätenregelung erfolgt, werden die in Abschn.5.1.1 beschriebenen Prioritätenregeln verwandt. Darüberhinaus besteht die Möglichkeit, die Priorität zwischen allen Ports zyklisch zu verändern. Die Anzahl der Takte nach denen jeweils ein Prioritätswechsel erfolgt ist frei wählbar.

- Der Instruktionsteil initiiert 1-parcel Pseudo-Instruktionen. Jede CPU kann in jedem Takt gemäß Abschn.5.1.2 eine Instruktion initiieren.

- Der Adreßteil einer CPU (vgl. Abschn.5.1.3) ist im Simulationsmodell nicht berücksichtigt.

- Als Skalarinstruktionen (vgl. Abschn.5.1.4) sind skalare Speicherzugriffe sowie skalare Gleitkommaoperationen implementiert. Bei den Speicherzugriffen wird zwischen *Load* und *Store* nicht unterschieden. Die Adresse der Bank, auf die ein Speicherzugriff im Skalarmodus erfolgt, wird zufällig ausgewählt.

- Der Vektorteil des Systems (vgl. Abschn.5.1.5) ist relativ detailliert nachgebildet. Jede CPU verfügt über 8 Vektorregister der Länge 64 sowie die Gleitkommafunktionseinheiten *Add, Multiply* und *Reciprocal Approximation*. Als Vektorinstruktionen sind die beiden Speicheroperationen *Load* und *Store* sowie vektorielle Gleitkommaoperationen implementiert. Die Speicherinstruktionen enthalten die Startbank, auf die der erste Zugriff erfolgen soll, die aktuelle Vektorlänge sowie die Distanz.

Jede aktive CPU führt die Pseudo-Instruktionen eines "Programms" aus. Als Anhaltspunkt dient dazu zweckmäßigerweise der vom CRAY-FORTRAN-Compiler [CRA 83] erzeugte Assembler-Code [CRA 82]. Das interessierende Code-Stück wird Instruktion für Instruktion in den Pseudo-Code umgesetzt, sofern ein Äquivalent dafür vorhanden ist. Nicht als Pseudo-Instruktion implementierte Instruktionen werden durch Pseudo-NOPs (*no operation*) berücksichtigt, um vergleichbare Ausführungszeiten zu erhalten.

Da, insbesondere bei der Simulation eines Mehrprozessorsystems, der Simulator eine Einschwingphase besitzt, ist es vorteilhaft, die zu untersuchenden "Programme" mehrfach auszuführen und jeweils den Mittelwert und die Standardabweichung der interessierenden Werte zu ermitteln. Dabei kann ein Intervall aus der Gesamtzahl der Einzelläufe angegeben werden, das in die Statistik miteinbezogen wird. Um das Einpendeln in einen festen Zyklus zu verhindern, wird zum einen die Bankadresse für skalare Speicherzugriffe zufällig nach einer Gleichverteilung ausgewählt, zum anderen kann die Startbank für Speicherzugriffe im Vektormodus zufällig ausgewählt sowie die Anzahl der Pseudo-NOPs zufällig nach einer Exponentialverteilung bei vorgegebenem Mittelwert variiert werden. Vor

allem letzteres ist ein gutes Hilfsmittel, um synthetische Arbeitslasten (*synthetic workload*) zu erzeugen [KOB 78].

Nach Beendigung eines Simulationslaufes werden für jedes aktive "Programm" folgende Ergebnisse ausgegeben:

- die Anzahl der zur Ausführung benötigten Takte bzw. die entsprechende CPU-Zeit;
- die Anzahl der "blocked issues", d.h. die Anzahl der Fehlversuche eine Instruktion zu initiieren;
- die Anzahl der Bank-, Pfad- und Simultankonflikte;
- die durchschnittliche effektive Bandbreite, worunter das Verhältnis der Anzahl der "übertragenen Daten" zu der Anzahl der zur Ausführung benötigten Takte verstanden wird.

Weiterhin werden die Aktivitäten auf dem Speicher für ein bestimmtes Intervall ausgegeben. Neben der Speicherauslastung erscheint für jede einzelne Bank die Anzahl der darauf stattgefundenen Zugriffe sowie der Bank-, Pfad- und Simultankonflikte, die beim Versuch, auf diese Bank zuzugreifen, entstanden sind. Zusätzlich können Zugriffsdiagramme für ein vorgegebenes Intervall ausgedruckt werden, die ein detailliertes Bild über den Ablauf der Zugriffe auf den Speicher sowie der Konflikte vermitteln.

5.3 VERGLEICH: MESSUNG - SIMULATION - MODELLIERUNG

Eine der wichtigsten Grundoperationen stellt die sogenannte *Triade* dar:

```
      DO 2 I = 1,N*INC,INC
    2 A(I) = B(I) + C(I)*D(I)
```

INC bedeutet hierbei das *Inkrement*, was in FORTRAN im Falle des Zugriffs auf eindimensionale Felder bzw. auf Spalten mehrdimensionaler Felder der Distanz entspricht, mit der die einzelnen Feldelemente im Speicher auseinander liegen. Die Schreibweise *N*INC* soll andeuten, daß die Vektoroperation unabhängig vom Inkrement die Länge *n* besitzt.

Idealerweise laufen bei der Ausführung einer Triade vier Zugriffsströme parallel, drei um *B,C* und *D* in die Vektorregister zu laden sowie einen um den Resultatvektor *A* zu speichern. Die benötigten Gleitkommafunktionseinheiten *Multiply* und *Add* können verkettet arbeiten (vgl. Abschn.2.1). Da im System CRAY X-MP jedoch lediglich drei Ports, zwei zum Laden und einer zum Speichern (vgl. Abschn.5.1.5), zur Verfügung stehen, erfolgt das Laden der Operanden nicht mit der gewünschten Bandbreite. Prinzipiell müssen folgende Instruktionen (unter Nichtberücksichtigung von Adreßoperationen) ausgeführt werden:

```
      VL ← N mod 64
$2A   lade D in Vektorregister V7
      lade C in Vektorregister V6
      V5 ← V6 * V7
      lade B in Vektorregister V4
      V0 ← V4 + V5
      speichere V0 nach A
      falls ENDE springe nach $2B
      VL ← 64
      springe nach $2A
$2B   ...
```

Mit "ENDE" wird symbolisch angedeutet, daß entsprechende Instruktionen ausgeführt werden, um festzustellen, ob die vorgegebene Vektorlänge *n* vollständig abgearbeitet ist. Da die Vektorregister maximal *64* Elemente aufnehmen können (vgl. Abschn.5.1.5), sind $\lceil n/64 \rceil$ Schleifendurchläufe nötig. Im folgenden werden die am Beispiel einer solchen Triade durchgeführten Experimente beschrieben.

Um die relative Position der Felder zueinander im Speicher festlegen zu können, werden die Felder in einem COMMON-Block abgelegt:

```
COMMON// A(IDIM),B(IDIM),C(IDIM),D(IDIM)
```

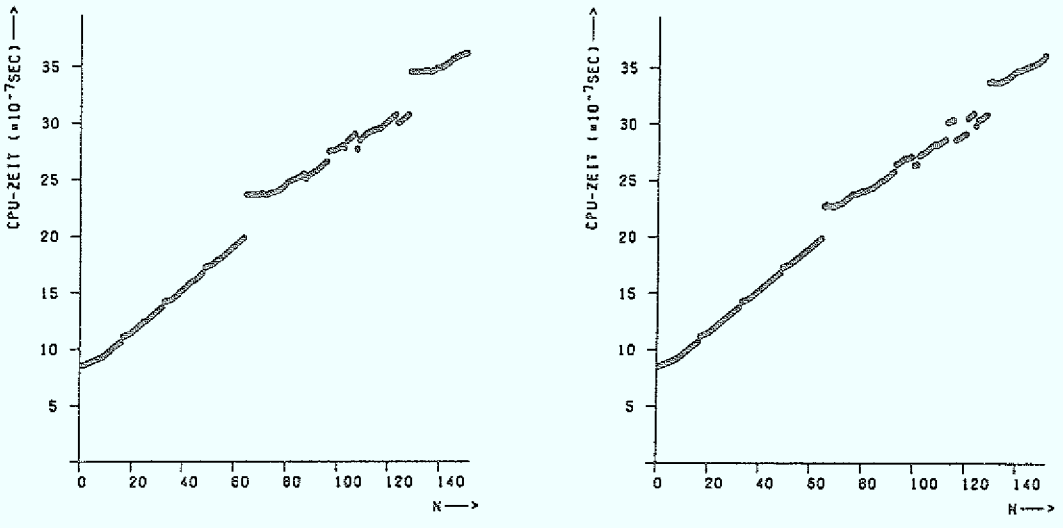
Über *IDIM* wird die Dimensionierung der Felder jeweils dem durchzuführenden Experiment angepaßt.

Experiment 5.1: Die Triade wird für die Vektorlängen $1 \le n \le 150$ auf einer CPU ausgeführt; die andere CPU führt die Dummy-Routine aus und somit erfolgen von dort keine Zugriffe auf den Speicher. Durch die Wahl von *IDIM=16*16=256* befinden sich die jeweils ersten Elemente der Felder auf der gleichen Bank.

Aufgrund der für Pipeline-Rechner i.allg. verwendeten Formel [HuJ 81], ist die optimale Ausführungszeit proportional zur Vektorlänge n:

$$t_o = (a + bn)\tau. \qquad (5.1)$$

In Abb.5.2a ist der Verlauf der tatsächlich benötigten CPU-Zeit über der Vektorlänge n aufgetragen, der zwar im wesentlichen linear ist, jedoch einige Sprünge enthält.



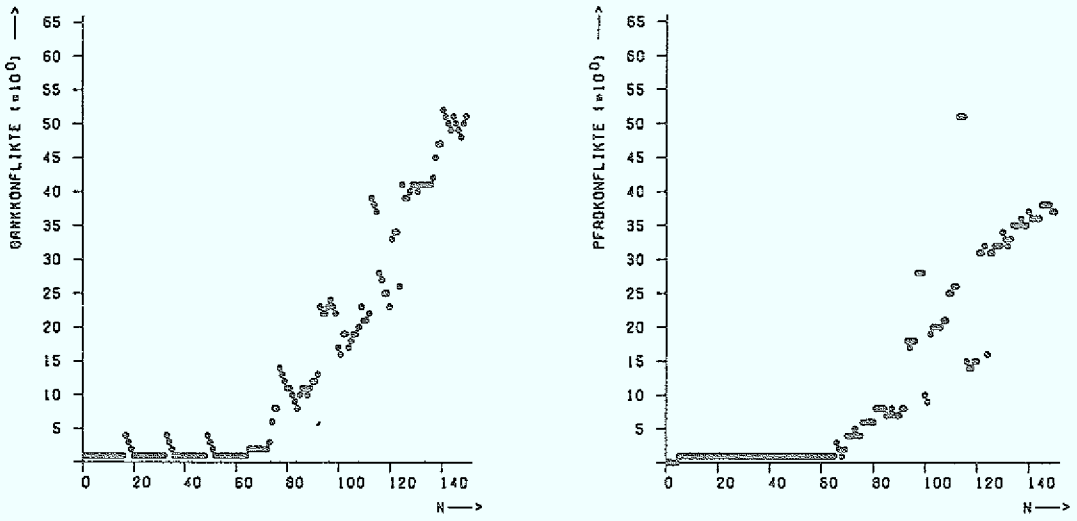
a) Messung auf CRAY X-MP b) Simulation

Abb.5.2 Ausführungszeit für eine Triade *1≤n≤150*

Unmittelbar auffallend ist der Sprung bei *n=65*, was auf das notwendige Nachladen der Vektorregister zurückzuführen ist, da ein neuer Schleifendurchlauf beginnt. Mit wachsender Vektorlänge nähert sich der Verlauf wieder dem theoretischen Wert nach Gl.(5.1) an. Bei *n=129* ist erneut ein Nachladen der Vektorregister erforderlich, wodurch wiederum ein Sprung erfolgt. Weiterhin sind kleinere Sprünge bei *n=17*, *n=33* sowie *n=49* erkennbar. Auffallend ist auch der ungleichmäßige Verlauf für Werte von *n>64*. Insbesondere ist es merkwürdig, daß es Fälle gibt, bei denen mit wachsender Vektorlänge die Ausführungszeit wieder abnimmt. So beträgt beispielsweise die Ausführungszeit für *n=107 2.89µsec* und für *n=108 2.76µsec*.

Aufschluß über diese Phänomene kann mit Hilfe der Simulation gewonnen werden. In Abb.5.2b ist zum Vergleich mit den Meßergebnissen die durch den Simulator ermittelte Ausführungszeit aufgetragen. Es zeigt sich eine

recht gute Übereinstimmung der gemessenen mit den simulierten Werten, vor allem an den markanten Stellen. Die in Abb.5.3 dargestellten, durch die Simulation ermittelten, Bank- und Pfadkonflikte lassen erkennen, daß die sprunghaften Änderungen der Ausführungszeit in enstprechenden Zugriffskonflikten ihre Begründung finden.



a) Bankkonflikte b) Pfadkonflikte

Abb.5.3 Simulation einer Triade $1 \leq n \leq 150$

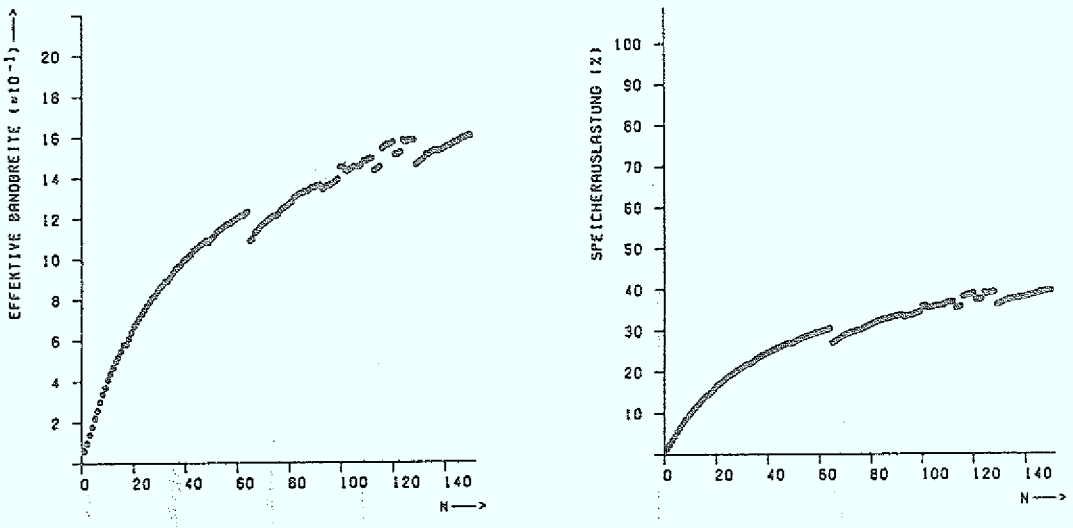
Während für $n \leq 64$ in der Regel nur ein Bankkonflikt auftritt, springt die Anzahl der Bankkonflikte jeweils bei $n=17$, $n=33$ sowie $n=49$ auf vier an, um dann wieder mit wachsendem n bis auf einen abzunehmen (s. Abb.5.3a). Diese Sprünge ergeben sich dadurch, daß nach jeweils *16* Zugriffen wiederum auf die Startbank zugegriffen wird, die offensichtlich noch nicht wieder frei ist. Da bei den Pfadkonflikten (s. Abb.5.3b) in diesem Bereich keine entsprechenden Sprünge vorliegen, ist ein Verbundkonflikt ausgeschlossen. Die maximal drei parallel laufenden Zugriffsströme müssen also gemäß Gl.(4.46) mindestens einen Takt Abstand voneinander haben. Da die jeweilige Initiierung der Zugriffsströme jedoch nicht notwendigerweise im optimalen Zeitpunkt erfolgt, ist wegen $3(n_c+1)=15$ die Wahrscheinlichkeit relativ hoch, bei lediglich sechzehn Bänken bei der Rückkehr auf die Startbank Bankkonflikte zu erhalten.

Für Werte von $n>64$ nehmen sowohl die Bank- als auch die Pfadkonflikte drastisch zu. Aufgrund des erneuten Schleifendurchlaufs verschieben sich noch aktive gegen neu initiierte Zugriffsströme, was zu einer Neufestsetzung der Prioritäten führt. Je nachdem wie, ungünstig diese Verschiebung ausfällt, treten mal mehr mal weniger Konflikte auf. Vor allem eine Häu-

fung von Verbundkonflikten, die sich nicht synchronisieren (vgl. Abschn.4.3.2) wirkt sich nachteilig auf die Ausführungszeit aus. Somit läßt sich auch die beobachtete Anomalie, daß trotz wachsender Vektorlänge die Ausführungszeit u.U. wieder abnimmt, erklären. Bei der Simulation ist beispielsweise eine solche Anomalie für *n=99* und *n=100* zu beobachten:

n	t_g	Bankkonflikte	Pfadkonflikte
99	2.71µsec	22	28
100	2.63µsec	17	10

Schließlich zeigt Abb.5.4, daß die bei der Ausführungszeit beobachteten Sprünge und Anomalien sich entsprechend in der effektiven Bandbreite (s. Abb.5.4a) sowie der Speicherauslastung (s. Abb.5.4b) niederschlagen.



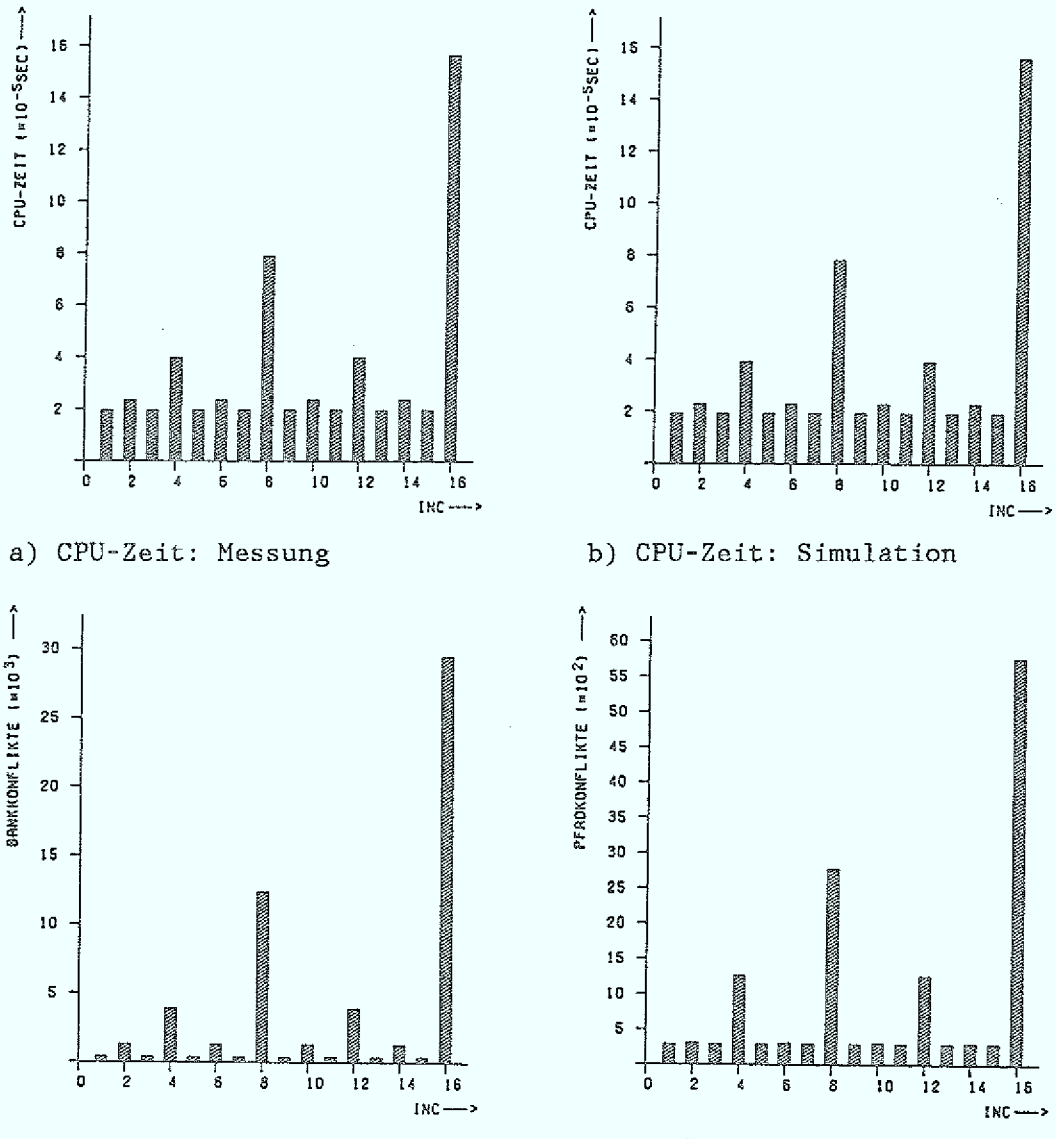
a) effektive Bandbreite b) Speicherauslastung

Abb.5.4 Simulation einer Triade *1≤n≤150*

Experiment 5.2: Die Triade wird für das Inkrement *1≤INC≤16* bei jeweils gleicher Vektorlänge *n=1024* auf einer CPU ausgeführt; die andere CPU führt die Dummy-Routine aus und somit erfolgen von dort keine Zugriffe auf den Speicher. Durch die Wahl von *IDIM=16*1024=16384* befinden sich die jeweils ersten Elemente der Felder auf der gleichen Bank.

In Abb.5.5a ist der Verlauf der tatsächlich benötigten CPU-Zeit über dem Inkrement *INC* aufgetragen, während in Abb.5.5b die entsprechenden, durch die Simulation ermittelten CPU-Zeiten dargestellt sind. Auch in diesem

Beispiel zeigt sich wiederum eine sehr gute Übereinstimmung zwischen Messung und Simulation.



a) CPU-Zeit: Messung

b) CPU-Zeit: Simulation

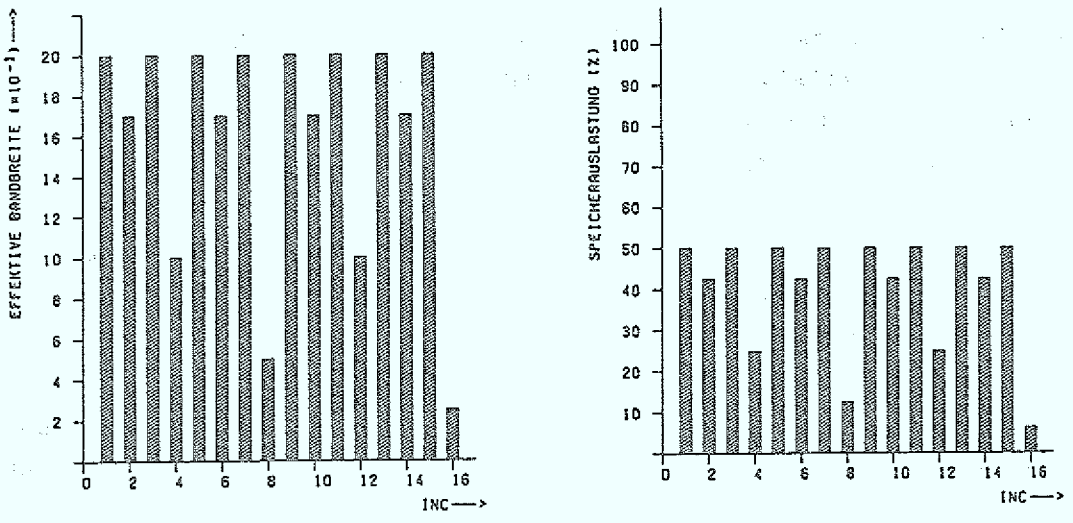
c) Bankkonflikte

d) Pfadkonflikte

Abb.5.5 Ausführung einer Triade (*n=1024*) bei unterschiedlichem Inkrement und gleicher Startbank

Diese Ergebnisse lassen sich mit den in Kap.4 gewonnenen theoretischen Resultaten erklären. Aufgrund der gleichen Startbank sind die Fälle mit jeweils gleicher Rückkehrzahl isomorph und somit ergibt sich für die Inkremente (*1,3,5,7,9,11,13,15*) jeweils die gleiche Ausführungszeit, die gleiche Anzahl an Bank- und Pfadkonflikten (s. Abb.5.5c und Abb.5.5d) sowie die gleiche mittlere effektive Bandbreite (s. Abb.5.6a) und die

gleiche Speicherauslastung (s. Abb.5.6b). Entsprechendes gilt für die Inkremente (*2,6,10,14*), (*4,12*), (*8*) und (*16*).



a) effektive Bandbreite b) Speicherauslastung

Abb.5.6 Ausführung einer Triade (*n=1024*) bei unterschiedlichem Inkrement und gleicher Startbank

Die niedrigste Ausführungszeit sowie die geringste Anzahl an Bank- und Pfadkonflikten wird jeweils bei ungeradem Inkrement erreicht. In diesem Fall besitzen alle Zugriffsströme die maximale Rückkehrzahl von $r_I=m=16$. Trotz dreier Ports zum Speicher beträgt die mittlere effektive Bandbreite lediglich $b_{eff}=2$, da von den vier erforderlichen Datenströmen maximal drei parallel aktiv sein können und auf den vierten dann gewartet werden muß.

Für die Gruppe (*2,6,10,14*) ist ein leichter Anstieg der Ausführungszeit, bedingt durch einen Anstieg der Bankkonflikte, festzustellen; die Pfadkonflikte nehmen gegenüber dem günstigsten Fall nicht zu. Da von den sechzehn verfügbaren Bänken nur acht benutzt werden, kann wegen $2n_c=8$ die effektive Bandbreite nicht größer als zwei werden. Allerdings zeigte sich bereits für den günstigsten Fall $b_{eff}=2$ als eine Grenze, wenn auch aus anderen Gründen. Deswegen bleibt der Anstieg der Ausführungszeit in diesem Fall gegenüber dem günstigsten relativ niedrig.

Anders hingegen ist der Sachverhalt bei den verbleibenden Fällen. Die Gruppe (*4,12*) greift lediglich auf vier der sechzehn verfügbaren Bänke zu, (*8*) auf zwei Bänke und (*16*) gar nur noch auf eine Bank, was zu einem starken Anstieg der Bank- und Pfadkonflikte führt. Entsprechend sinkt die

effektive Bandbreite auf *1* (*2,12*), auf *1/2* (*8*) bzw. auf *1/4* (*16*) ab. Im gleichen Maße verdoppelt, vervierfacht bzw. verachtfacht sich die Ausführungszeit gegenüber dem günstigsten Fall.

Dieses Experiment zeigt also, daß möglichst große Rückkehrzahlen anzustreben sind. Parallele Algorithmen [HOß 83] verlangen jedoch öfters Inkremente, die Vielfache von zwei sind, insbesondere falls rekursive Datenabhängigkeiten aufgelöst werden sollen [OED 84]. Da die Anzahl der Bänke ebenfalls eine Zweierpotenz ist, ergeben sich kleinere Rückkehrzahlen. Trotzdem läßt sich eine hohe effektive Bandbreite erreichen, falls die Anordnung der Felder disjunkte Zugriffsmengen zur Folge hat, was das nächste Experiment zeigt.

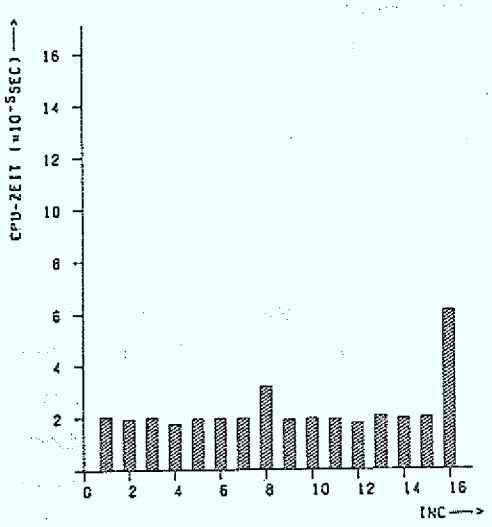
Experiment 5.3: Die Triade wird für das Inkrement *1≤INC≤16* bei jeweils gleicher Vektorlänge *n=1024* auf einer CPU ausgeführt; die andere CPU führt die Dummy-Routine aus und somit erfolgen von dort keine Zugriffe auf den Speicher. Durch die Wahl von *IDIM=16*1024+1=16385* befinden sich die jeweils ersten Elemente der Felder auf jeweils einer anderen Bank.

Auch hier zeigt sich eine gute Übereinstimmung zwischen den gemessenen Ausführungszeiten (s. Abb.5.7a) und den simulierten Ausführungszeiten (s. Abb.5.7b). Da die relative Position der Startbänke für alle Inkremente erhalten bleibt, sind die unterschiedlichen Fälle innerhalb der Gruppen mit gleicher Rückkehrzahl nicht notwendigerweise isomorph. Dadurch ergeben sich die geringfügigen Schwankungen der Ausführungszeit bzw. der Bank- (s. Abb.5.7c) und Pfadkonflikte (s. Abb.5.7d).

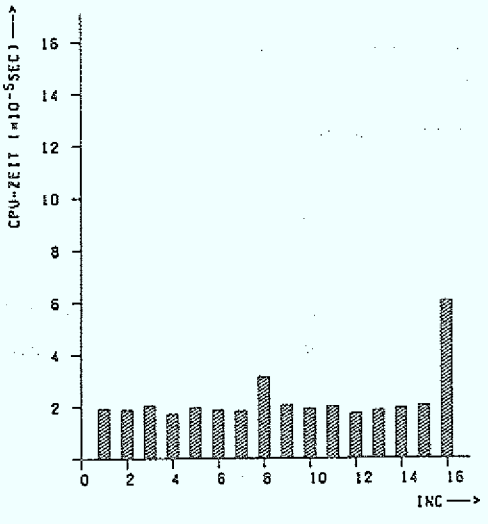
Das Verhalten der Gruppe (*1,3,5,7,9,11,13,15*) ist im wesentlichen insensitiv gegenüber der relativen Positionierung der Felder. Dies ist auch zu erwarten, da die Zugriffsströme dieser Gruppe ohnehin auf alle Bänke zugreifen.

Anders hingegen ist die Situation bei den verbleibenden Gruppen, wo die Zugriffe aufgrund der Verschiebung günstiger über die verfügbaren Bänke verteilt werden als im Fall von Experiment 5.2.

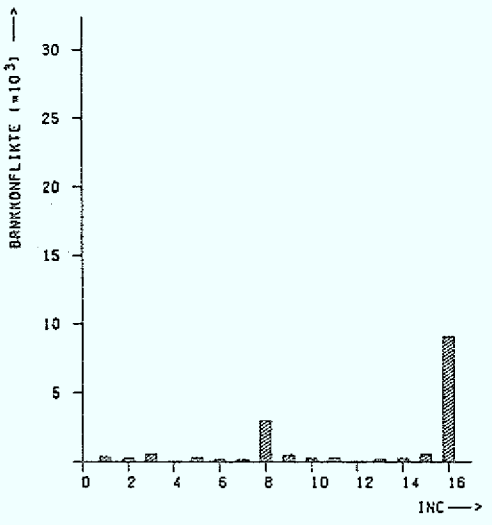
Der optimale Fall liegt für die Gruppe (*4,12*) vor, bei der die Zugriffsmengen aller vier auftretenden Zugriffsströme disjunkt sind, und die Rückkehrzahlen mit $r_I=n_c$ Bedingung (4.11) erfüllen. Somit treten weder Bank- noch Pfadkonflikte auf.



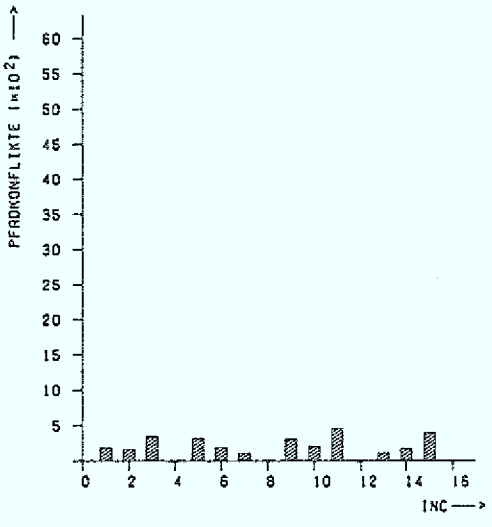
a) CPU-Zeit: Messung



b) CPU-Zeit: Simulation



c) Bankkonflikte



d) Pfadkonflikte

Abb.5.7 Ausführung einer Triade (*n=1024*) bei unterschiedlichem Inkrement und unterschiedlicher Startbank

Die Fälle (*8*) und (*16*) verfügen ebenfalls über disjunkte Zugriffsmengen, so daß ebenfalls keine Pfadkonflikte entstehen, doch nehmen die Bankkonflikte wegen zu kleiner Rückkehrzahlen zu. Trotzdem sind die Konflikte und damit auch die Ausführungszeiten gegenüber der Anordnung der Felder in Experiment 5.2 erheblich reduziert.

Während in den bislang aufgeführten Experimenten keine weiteren Zugriffe auf den Speicher seitens der anderen CPU erfolgten, soll noch ein abschließendes Experiment in einer Umgebung mit Zugriffen durch die andere CPU durchgeführt werden.

Experiment 5.4: Die Triade wird für das Inkrement $1 \leq INC \leq 16$ bei jeweils gleicher Vektorlänge $n=1024$ auf einer CPU ausgeführt. Durch die Wahl von $IDIM=16*1024+1=16385$ befinden sich die jeweils ersten Elemente der Felder auf jeweils einer anderen Bank. Die andere CPU führt ein Programm aus, das über alle drei Ports möglichst oft Vektor-Speicherinstruktionen mit der Distanz *1* initiiert.

Die Realisierung dieses Experiments gestaltet sich etwas schwieriger als die der vorherigen Experimente, da sich nicht feststellen läßt, was "möglichst oft auf den Speicher zugreifen" in der realen Maschine bedeutet. Deshalb wurden die Läufe für die einzelnen Inkremente stets fünfundzwanzigmal in direkter Aufeinanderfolge ausgeführt. Diese Gruppen zu fünfundzwanzig Läufen wurden solange wiederholt, bis die Standardabweichung innerhalb einer Gruppe genügend klein war. Somit ergibt sich wiederum eine gute Übereinstimmung zwischen den gemessenen Ausführungszeiten (s. Abb.5.8a) und den durch die Simulation ermittelten Ausführungszeiten (s. Abb.5.8b).

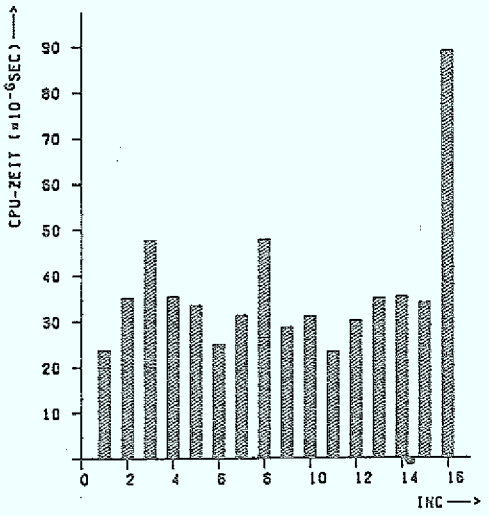
Da in diesem Experiment Speicherzugriffe von zwei CPUs aus erfolgen, treten auch Simultankonflikte (s. Abb.5.8e) auf, deren Anzahl im Vergleich zu den Bank- (s. Abb.5.8c) und Pfadkonflikten (s. Abb.5.8d) jedoch relativ gering ist. Lediglich für die, wegen ihrer niedrigen Rückkehrzahl ohnehin ungünstigen Gruppen (*8*) und (*16*) liegt der Anteil der Simultankonflikte oberhalb von 10%.

Die geringsten Störungen durch das auf der anderen CPU gegenlaufende Programm erfährt die Triade für die Inkremente *1,6* und *11*.

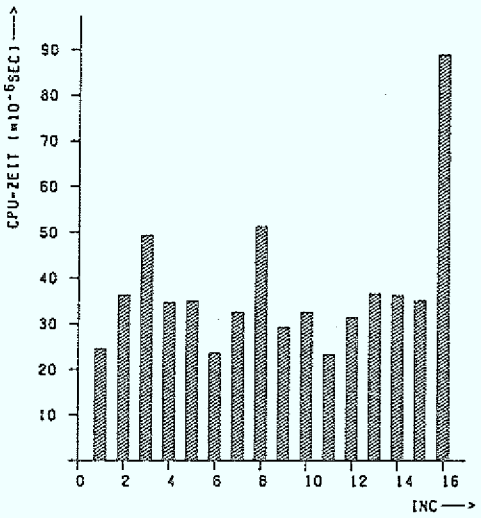
Der Fall *INC=1* ist insofern günstig, da alle Zugriffsströme mit der gleichen Distanz und maximaler Rückkehrzahl auf den Speicher zugreifen.

Das günstige Verhalten von *INC=6* in der Umgebung anderer Zugriffsströme mit Distanz *1* ist auf eine Barriere-Situation zurückzuführen. Der Fall *6* => *1* ist isomorph (s. Anhang) zu dem Fall *2* => *3*, d.h. die Triade mit *INC=6* verläuft relativ ungestört, während das Umgebungsprogramm mit *INC=1* verzögert wird (s. Theorem 4.7).

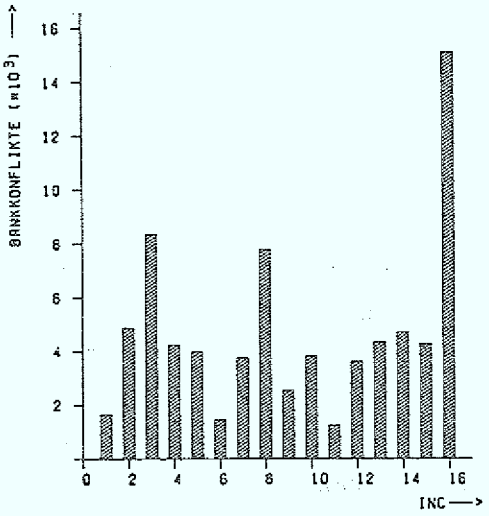
INC=11 schließlich stellt ebenfalls eine Barriere-Situation dar, die aufgrund der Isomorphie mit *1* => *3* identisch ist. In diesem Experiment kann die Triade mit *INC=11* relativ ungestört zugreifen, während das Gegenprogramm sehr viele Störungen erfährt.



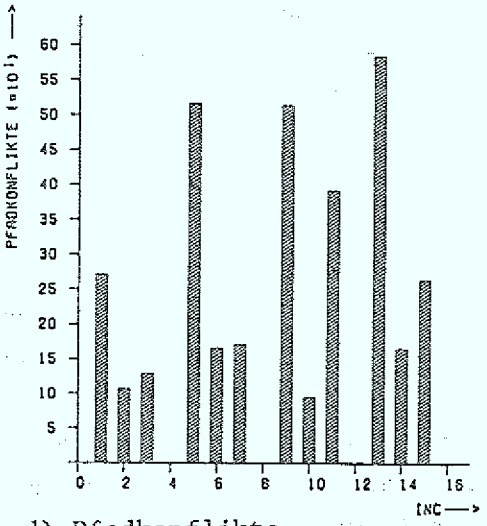
a) CPU-Zeit: Messung



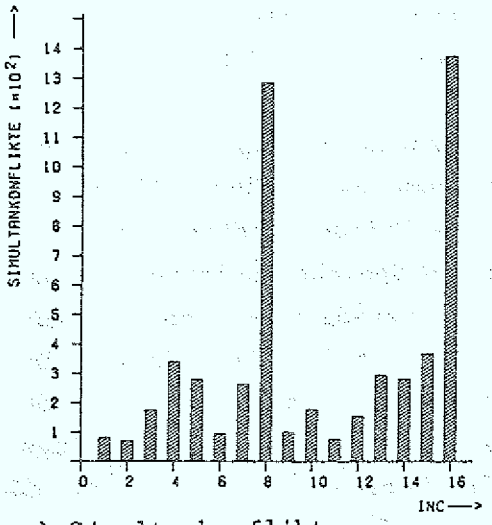
b) CPU-Zeit: Simulation



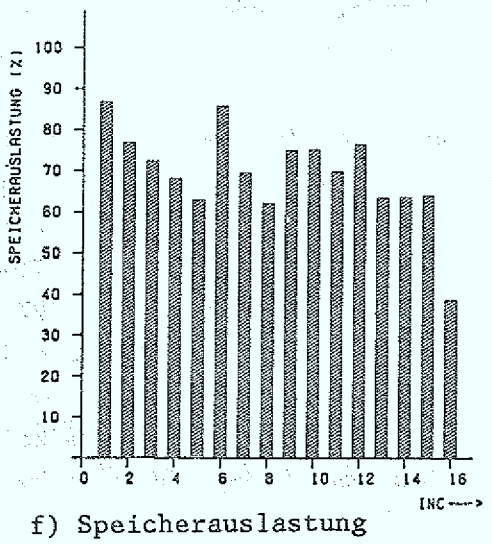
c) Bankkonflikte



d) Pfadkonflikte



e) Simultankonflikte



f) Speicherauslastung

Abb.5.8 Ausführung einer Triade (*n=1024*) bei unterschiedlichem Inkrement und unterschiedlicher Startbank; zweite CPU greift mit Distanz *1* zu

Entsprechend handelt es sich bei *INC=2* und *INC=3* um Barriere-Situationen, bei der die Triade durch das Gegenprogramm verzögert wird. Der starke Anstieg der Ausführungszeit um etwa 50% (*2*) bzw. 100% (*3*) gegenüber dem günstigsten Fall vermittelt sehr eindrucksvoll die Folgen einer solchen Situation.

Auch die in Experiment 5.4 günstigste Gruppe (*4,12*), die aufgrund disjunkter Zugriffsmengen sowie ausreichender Rückkehrzahl sich selbst keinerlei Konflikte erzeugt, wird durch das Gegenprogramm ziemlich stark beeinträchtigt. Ebenso verhält es sich mit den verbleibenden Inkrementen, wobei für *INC=9* eigentlich geringere Störungen erwartet werden, da das Aufeinandertreffen der Distanzen *1* und *9* nach Gl.(4.28) ebenfalls konfliktfrei verläuft. Allerdings versuchen in dem Experiment mitunter alle sechs Zugriffsströme auf den Speicher zuzugreifen, was zwangsläufig zu Konflikten führen muß, da $6n_c=24>m=16$. Das Verhalten in solchen Fällen wird durch die analytischen Modelle aus Kap.4 jedoch nicht mehr beschrieben.

Schließlich sei noch auf die in Abb.5.8f dargestellte Speicherauslastung verwiesen. Wegen $m=16$ und $n_c=4$ ist unmittelbar einsichtig, daß der Speicher bereits bei vier optimal verlaufenden Zugriffsströmen, d.h. alle mit maximaler Rückkehrzahl und konfliktfrei, zu 100% ausgelastet ist. Da in diesem Experiment mitunter bis zu sechs Zugriffsströme gleichzeitig um den Zugriff auf den Speicher miteinander konkurrieren, entstehen auch aus dieser Sicht zwangsläufig Zugriffskonflikte. Die dadurch bedingten Wartezeiten verhindern wiederum ein kontinuierliches Zugreifen auf den Speicher, so daß die Speicherauslastung in der Regel unter 100% bleibt.

Am höchsten liegt die Speicherauslastung mit annähernd 90% für die beiden günstigen Fälle (*1*) und (*6*). Die Zugriffsströme der beiden CPUs stören sich also nur wenig gegenseitig. Der für die Triade günstige Fall (*11*) hingegen läßt durch die bei etwa 70% liegende Speicherauslastung erkennen, daß die Zugriffe der anderen CPU erheblich verzögert werden. Letzters trifft offensichtlich auch für die Inkremente *4,5,13,14* und *15* zu. Die Bankkonflikte und entsprechend die Ausführungszeit haben gegenüber dem Fall, daß von der anderen CPU aus keine Zugriffe erfolgen erheblich zugenommen. Allerdings weist das Absinken der Speicherauslastung auf etwa 65% auch auf erhebliche Verzögerungen in dem gegenlaufenden Programm hin.

Weiterhin ist das Absinken der Speicherauslastung im Falle der beiden Barriere-Situationen (*2*) und (*3*) gut zu beobachten, während die niedrige

Speicherauslastung der Inkremente *8* und *16* wiederum ein Ergebnis des ohnehin schlechten Zugriffsverhaltens dieser beiden Fälle ist.

Das letzte Experiment zeigt den mitunter beachtlichen Einfluß auf die Ausführungszeit eines Programms in Abhängigkeit von der Umgebung. Untersuchungen über "typische" Programme (z.B. [KNU 71] und [KMC 72]) haben gezeigt, daß das Inkrement *1* in FORTRAN-Programmen in mehr als 90% aller Schleifen verwendet wird. Allerdings bedeutet dies auch nur dann mit Sicherheit eine Distanz von *1*, falls es sich entweder um eindimensionale Felder handelt oder die Zugriffe in mehrdimensionalen Feldern über die Spalten erfolgen.

Bezeichnet *INC* das in der FORTRAN-Schleife angegebene Inkrement, und soll der Zugriff über die *(k+1)*-te Dimension eines Feldes erfolgen, so ergibt sich die zugehörige Distanz *d* zu

$$INC \cdot \prod_{i=0}^{k} J_i \bmod m = d, \qquad (5.2)$$

wobei J_i die Anzahl der Elemente der *i*-ten Dimension bezeichnet, mit $J_0=1$.

Mittlerweile sind jedoch viele Algorithmen umgestellt worden, um Datenabhängigkeiten aufzulösen, die eine Vektorisierung verhindern. Einige solcher Algorithmen, die sehr häufig eine Distanz von *2* verlangen sind beispielsweise in [HuJ 81], [HOß 83] und [OED 84] zu finden. Greift nun die Mehrzahl der Programme mit der Distanz *1* auf den Speicher zu, so werden in Mehrprozessorsystemen (z.B. CRAY X-MP) die Programme, die mit der Distanz *2* auf den Speicher zugreifen erheblich verzögert, da eine Barriere-Situation vorliegt.

Handelt es sich dabei um sehr umfangreiche Programme und sind die Verzögerungen sehr gravierend, bietet sich die Möglichkeit des Multitasking [CRA 84] an, sofern das Programm geeignet zerlegt werden kann. Sobald ein solches Programm alle verfügbaren CPUs verwendet, ist die Speicherumgebung bekannt und die Zugriffe erfahren bei entsprechend geeigneter Dimensionierung der Felder erheblich weniger Konflikte.

Eine andere Möglichkeit, eine Barriere-Situation zu vermeiden, besteht darin, für die verschiedenen Datenstrukturen Abbildungsvorschriften (z.B. mit Hilfe der Skewing-Schemata [LAW 75], [SHA 78], [LuW 83]) zu entwickeln, so daß alle Zugriffsströme mit der Distanz *1* zugreifen können.

5.4 ARCHITEKTONISCHE UND TECHNOLOGISCHE ÄNDERUNGEN

Neben dem für Messungen zur Verfügung stehenden Doppelprozessorsystem CRAY X-MP werden Maschinen dieses Typs seit kurzem auch als Ein- und als Vier-Prozessorsysteme angeboten [CRA 84b]. Dabei verfügt das Ein-Prozessorsystem über einen Speicher in MOS-Technologie mit einer Bankzykluszeit von $t_c=8\tau$ (τ=9.5$n$sec), während in dem Vier-Prozessorsystem, das ausschließlich mit $m=64$ Bänken angeboten wird, jeweils 16 Bänke nach der in [CuS 84] vorgeschlagenen Methode zu $s=4$ Sektionen zusammengefaßt sind.

Daher ist es interessant, die Auswirkungen der unterschiedlichen Änderungen gegenüber dem vorgestellten Doppelprozessorsystem sowie einiger weiterer Alternativen dazu zu analysieren. Aufgrund seiner flexiblen Implementation ist der in Abschn.5.2.2 beschriebene Simulator dafür ein geeignetes Hilfsmittel.

Zur Durchführung der weiteren Experimente wird auf jeder aktiven CPU bei Simulationsbeginn das folgende "Programm" gestartet:

```
      X = Y*Z
      DO 10 I = 1,10
   10 A(I) = B(I) + C(I)
      DO 20 I = 1,200
      B(I) = A(I)*B(I)
   20 A(I) = (B(I) + C(I))*(B(I) + D(I))
      X = Y+Z
      DO 30 I = 1,200,2
   30 A(I) = B(I) + C(I)*D(I)
      DO 40 I = 1,1000
   40 A(I) = B(I) + C(I)
```

Diese *synthetische Arbeitslast* [KOB 78] erhebt allerdings keinen Anspruch darauf, für eine bestimmte Umgebung repräsentativ zu sein. Es sollen lediglich kürzere (DO-Schleife 10), mittlere (DO-Schleife 20 und 30) und größere Vektorlängen (DO-Schleife 40) sowie Skalaroperationen vertreten sein.

Im Falle mehrerer CPUs wird das Programm jeweils dreimal ausgeführt. Um das Erreichen eines festen Zyklus zu verhindern, werden bei jedem Eintritt in eine Schleife die Startbänke der zugehörigen Vektor-Speicherzugriffe zufällig ausgewählt. Ausgewertet wird schließlich der zweite Lauf von

einer CPU. Da auf jeder CPU das gleiche Programm ausgeführt wird, sind die Ergebnisse auf den anderen CPUs vergleichbar und werden daher nicht weiter berücksichtigt.

5.4.1 ANZAHL DER BÄNKE - ANZAHL DER CPUS

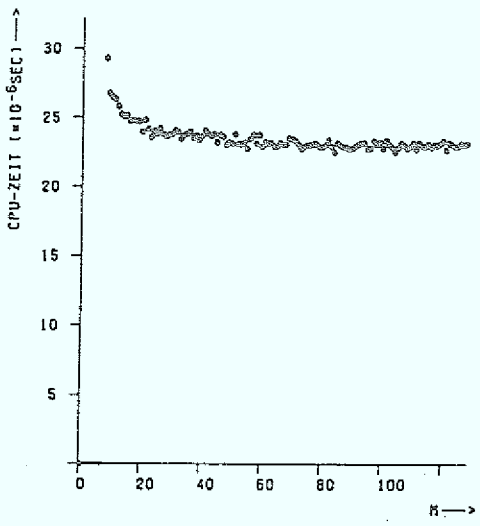
Bereits bei der Durchführung von Experiment 5.4 zeigte sich die Beschränkung der Bandbreite, da 16 Bänke für ein Doppelprozessorsystem mit jeweils bis zu drei gleichzeitig aktiven Ports nicht ausreichen (vgl. Def.3.9).

Experiment 5.5: Die synthetische Arbeitslast wird auf einer CPU für $8 \le m \le 128$ Bänke des Arbeitsspeichers ausgeführt. (Die Anzahl der Sektionen ist stets $s=4$ und die Bänke sind zyklisch darauf verteilt. In diesem Experiment ist m meist kein Vielfaches von s, d.h. in diesen Fällen ist die Anzahl der Bänke pro Sektion nicht gleich.)

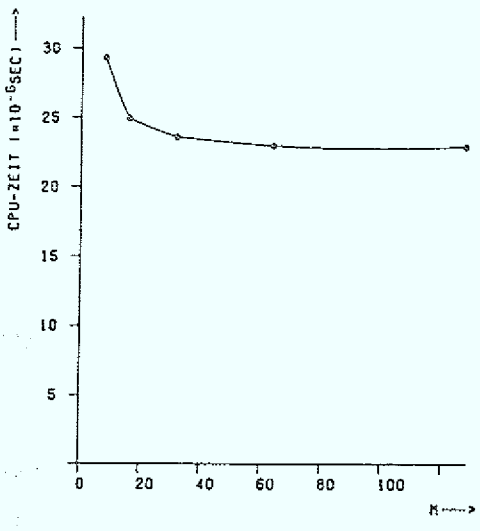
In Abb.5.9a ist der Verlauf der CPU-Zeit über m aufgetragen. Da aus Gründen einfacher Adressierungs-Hardware m in der Regel eine Zweier-Potenz ist, kommt den Ergebnissen für $m=8,16,32,64,128$ eine besondere Bedeutung zu (s. Abb.5.9b). Der Vergleich von Abb.5.9a mit Abb.5.9b zeigt ohnehin, daß sich kein Wert für m in irgendeiner Weise auszeichnet. Deshalb sollen im folgenden ausschließlich Zweier-Potenzen für m betrachtet werden.

Da bis zu drei Zugriffsströme gleichzeitig aktiv sind, müssen zum Erreichen der notwendigen Bandbreite mindestens $m=3n_c=12$ Bänke vorhanden sein. Diese Bedingung wird im Falle von 8 Bänken nicht erfüllt und macht sich durch den deutlichen Anstieg der CPU-Zeit bemerkbar.

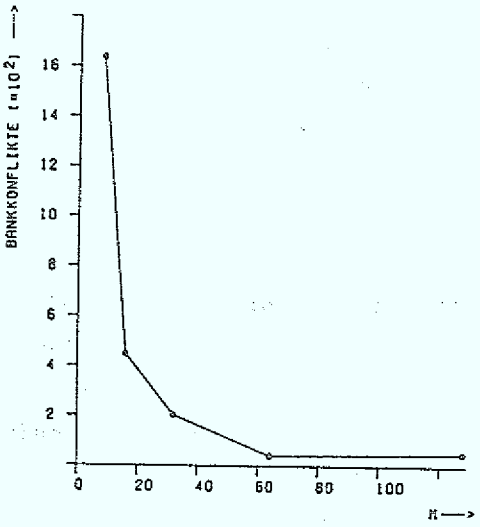
Im Fall von 16 Bänken ist die Bandbreite ausreichend, was zu einer starken Verkürzung der Ausführungszeit gegenüber dem Fall von 8 Bänken führt. Hat das System 32 Bänke, so ist eine weitere Verbesserung der Ausführungszeit um fast 10% festzustellen. Diese Verbesserung läßt sich mit Hilfe der analytischen Untersuchungen nicht erklären. Mit zunehmender Anzahl von Bänken entsteht allerdings auch "mehr Platz" im Speicher, wodurch die Wahrscheinlichkeit von Bank- (s. Abb.5.9c) und Pfadkonflikten (s. Abb.5.9d) geringer wird. Der Einsatz von 64 oder gar 128 Bänken bewirkt keine nennenswerten Verbesserungen, so daß für ein Ein-Prozessorsystem 32 Bänke als völlig ausreichend angesehen werden können.



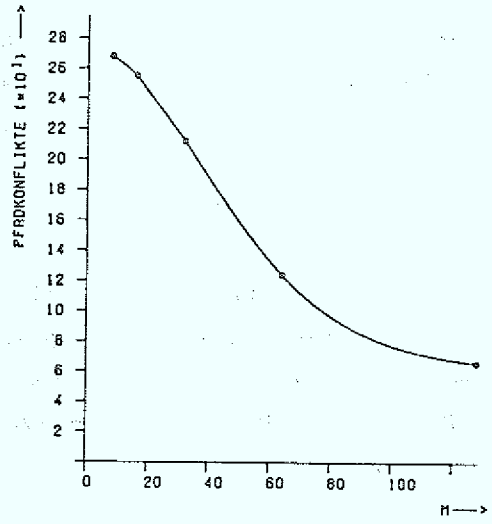
a) CPU-Zeit: $8 \leq m \leq 128$



b) CPU-Zeit: $m=8,16,32,64,128$



c) Bankkonflikte

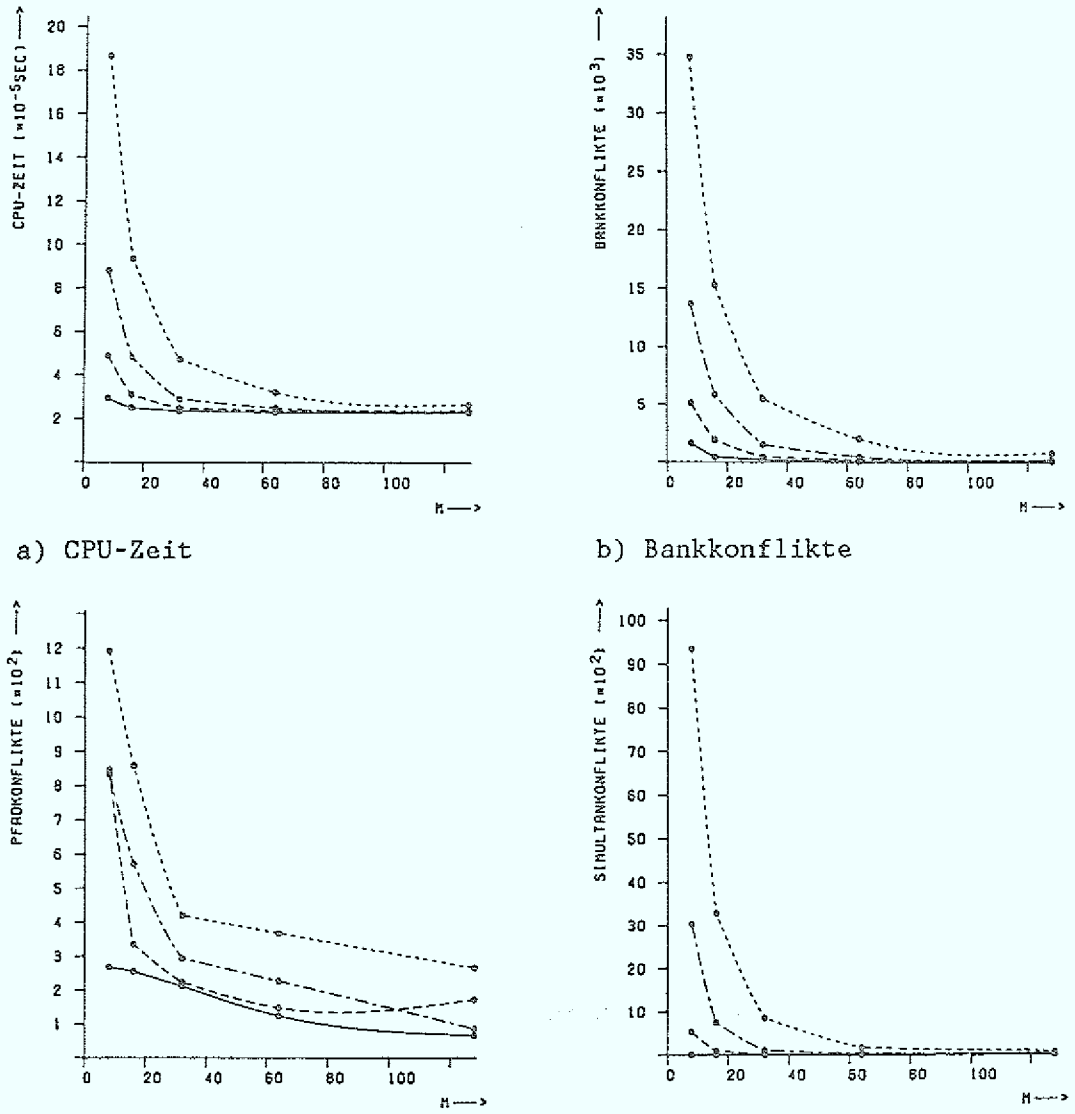


d) Pfadkonflikte

Abb.5.9 Einfluß der Anzahl *m* der Bänke bei einer aktiven CPU

Experiment 5.6: Die synthetische Arbeitslast wird auf 1,2,4 und 8 CPUs für jeweils 8,16,32,64 und 128 Bänke des Arbeitsspeichers ausgeführt.

Die Ausführungszeiten (s. Abb.5.10a) bleiben erwartungsgemäß in etwa gleich, falls bei Verdoppelung der Anzahl aktiver CPUs die Anzahl der verfügbaren Bänke ebenfalls verdoppelt wird. Ähnlich gilt dies auch für die Anzahl der Bank- und Simultankonflikte (s. Abb.5.10b und Abb.5.10d). Für die Pfadkonflikte (s. Abb.5.10c) wird dieser Zusammenhang hingegen nicht mehr beobachtet. Dies ist darauf zurückzuführen, daß die Pfadkonflikte lediglich zwischen den Ports innerhalb einer CPU auftreten (vgl. Abb.3.3) und die Anzahl *s* der Sektionen sich nicht verändert hat. In Abschn.5.4.3 wird noch näher auf diese Zusammenhänge eingegangen.



a) CPU-Zeit

b) Bankkonflikte

c) Pfadkonflikte

d) Simultankonflikte

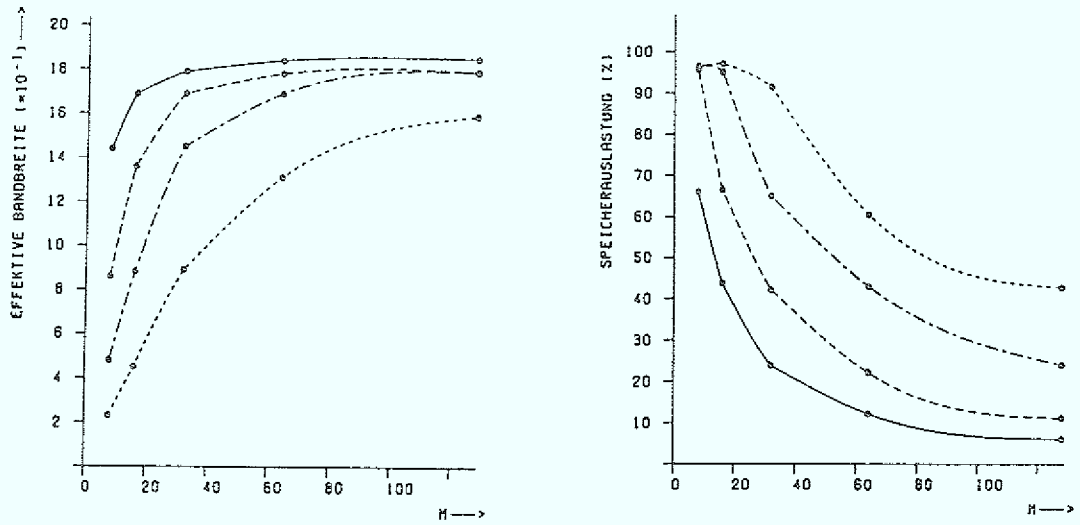
q=1: ——— q=2: - - - - - q=4: — — — — q=8: .....

Abb.5.10 Abhängigkeit zwischen Anzahl Bänke *m* und Anzahl CPUs *q*

In Abb.5.11 sind noch die effektive Bandbreite sowie die Speicherauslastung für das Experiment 5.6 wiedergegeben. Zwei Beobachtungen erscheinen dabei bemerkenswert:

- Der asymptotische Wert der effektiven Bandbreite in einem Ein-Prozessorsystem wird selbst bei Vergrößerung der Anzahl *m* der Bänke von den Mehrprozessorsystemen nicht erreicht. Die gegenseitigen Störungen lassen sich durch Erhöhung der Anzahl der Bänke zwar stark reduzieren, jedoch nicht gänzlich ausschließen.

- Eine gute Performance, d.h. annähernd optimale Ausführungszeit, wird erst bei einer Speicherauslastung von unter 50% erreicht.



a) effektive Bandbreite b) Speicherauslastung

q=1: ——— q=2: ----- q=4: -·-·- q=8: ·····

Abb.5.11 Abhängigkeit zwischen Anzahl Bänke *m* und Anzahl CPUs *q*

5.4.2 VERHINDERUNG VON VERBUNDKONFLIKTEN

Verbundkonflikte, die sich nicht synchronisieren (vgl. Abschn.4.3.2), entstehen leicht, falls die Anzahl der Takte für einen Bankzyklus n_c, die Anzahl der Sektionen s und die auftretenden Distanzen d_i nicht teilerfremd sind. Da *1* die am häufigsten auftretende Distanz ist, und im System CRAY X-MP $s=n_c=4$, sind Verschlechterungen der Performance durch Verbundkonflikte gegeben. Es sollen daher verschiedene Methoden zur Verhinderung von Verbundkonflikten diskutiert werden.

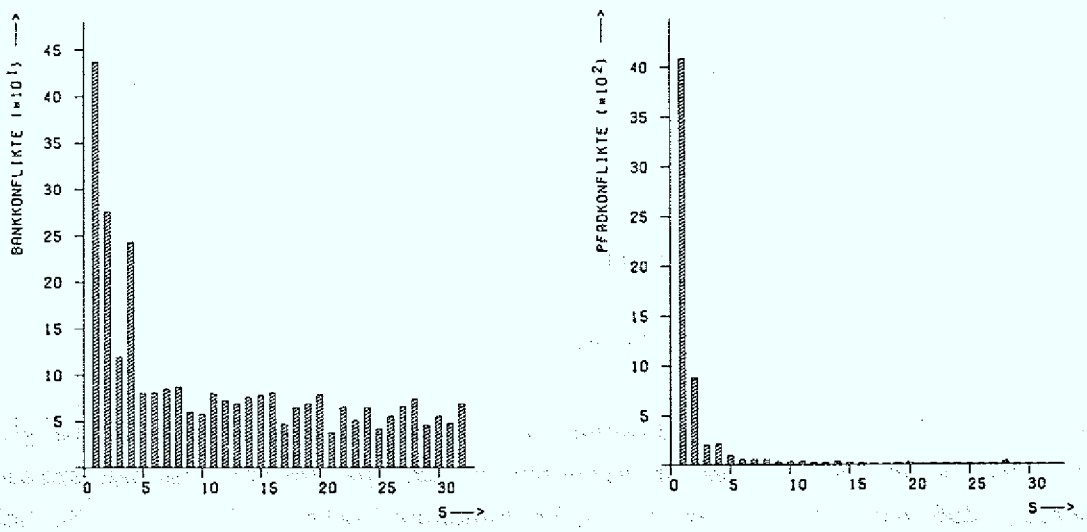
Da ein Verbundkonflikt zwischen Bank- und Pfadkonflikt ausschließlich zwischen den Zugriffsströmen einer CPU entstehen kann (s. Def.3.5), werden die Untersuchungen lediglich für eine CPU durchgeführt.

Eine Möglichkeit, Verbundkonflikte zu verhindern, ist, die Anzahl der Takte für einen Bankzyklus n_c zu verändern. Für $n_c=3,4$ und *5* ergeben sich folgende Resultate bei Ausführung der synthetischen Arbeitslast auf einem System mit $m=32$ Bänken und $s=4$ Sektionen:

n_c	t_g	Bankkonflikte	Pfadkonflikte
3	22.43µsec	52	85
4	23.97µsec	239	204
5	23.09µsec	131	52

Für $n_c=4$ zeigt sich eine deutlich höhere Anzahl sowohl der Bank- als auch der Pfadkonflikte im Vergleich zu $n_c=3$ und $n_c=5$. Darüberhinaus hat die Wahl von $n_c=3$ noch den Vorteil, daß einerseits nach einem Zugriff die referierte Bank schneller wieder freigegeben wird und andererseits die Verkürzung der Bankzykluszeit auch eine entsprechende Verkürzung der Speicherzugriffszeit nach sich zieht. Nachteilig dagegen ist, daß eine Verkürzung der Bankzykluszeit nur durch schnellere, und das bedeutet in der Regel teurere, Schaltkreise erreicht werden kann. Entsprechend führt eine Verlängerung der Bankzykluszeit auch zu verlängerten Speicherzugriffszeiten, was sich insbesondere bei Mehrprozessorsystemen nachteilig bemerkbar macht (vgl. Abschn.5.4.3).

Eine weitere Möglichkeit, Verbundkonflikte zu vermeiden, besteht darin, die Anzahl der Sektionen zu verändern. In Abb.5.12 sind die bei Ausführung der synthetischen Arbeitslast in einem System mit $m=32$, $n_c=4$ und $1 \le s \le m$ auftretenden Bank- und Pfadkonflikte über der Anzahl der Sektionen s aufgetragen.

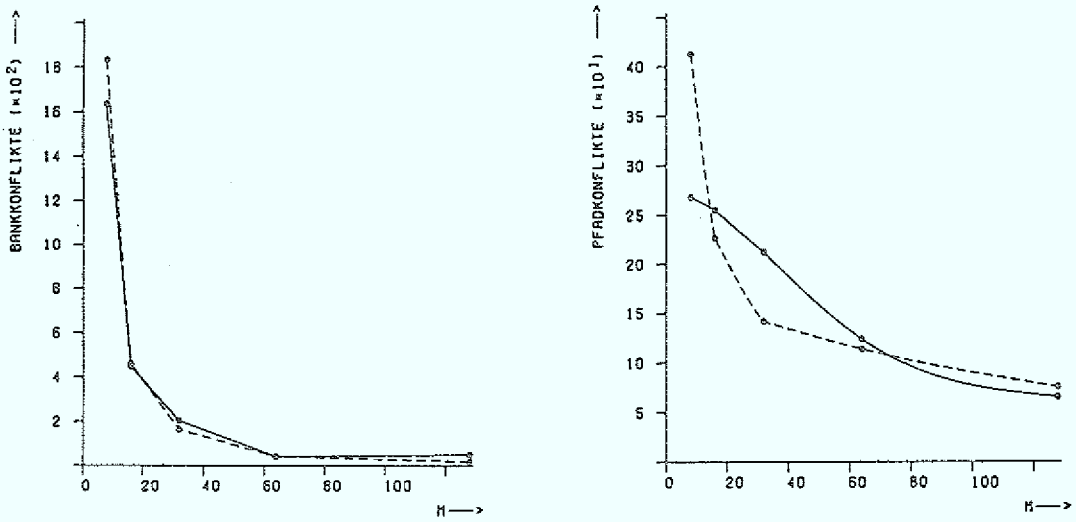


a) Bankkonflikte b) Pfadkonflikte

Abb.5.12 Einfluß der Anzahl Sektionen s; $m=32$, $n_c=4$

Die hohe Anzahl von Konflikten bei *s=1* und *s=2* ist nicht überraschend, da von den maximal drei gleichzeitig aktiven Zugriffsströmen lediglich einer bzw. zwei zugreifen können. Ab *s=3* sinken die Konflikte sehr stark ab, abgesehen von *s=4*. Hierbei treten Verbundkonflikte häufiger auf. Aus pragmatischen Gründen ist jedoch eine Anzahl von Sektionen, die kein Teiler der Anzahl der Bänke ist, nicht erstrebenswert. Daher müßte der Speicher statt in 4, in mindestens 8 Sektionen unterteilt werden. Allerdings wäre diese Maßnahme mit einem höheren Kostenaufwand verbunden.

Relativ leicht sowie ohne zusätzlichen Kostenaufwand läßt sich die Verwendung einer zyklischen Priorität für alle Ports realisieren. Die Auswirkungen auf die Anzahl der Bank- und Pfadkonflikte bei Ausführung der synthetischen Arbeitslast sind in Abb.5.13 für *m=8,16,32,64* und *128* Bänke aufgetragen; der Speicher ist jeweils in 4 Sektionen unterteilt.



a) Bankkonflikte b) Pfadkonflikte

feste Priorität: ——— zyklische Priorität: ------

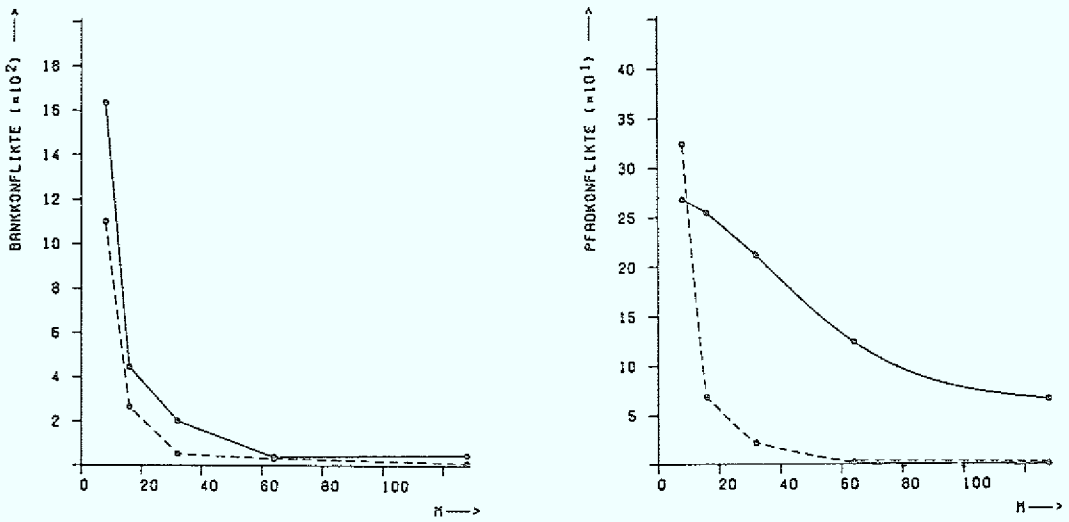
Abb.5.13 Einfluß der Prioritätenregelung; *s=4*, n_c=4

Offensichtlich ist eine zyklische Priorität nicht in jedem Falle günstiger als eine feste. Die Untersuchungen in Abschn.4.3.2 haben bereits gezeigt, daß eine Auflösung eines Verbundkonflikts nach dieser Methode "zusätzlich Platz" im Speicher benötigt. Daher ist dieses Verfahren nur bei einer relativ hohen Anzahl von Bänken erfolgversprechend.

Die Zusammenfassung von jeweils *m/s* Bänken mit aufeinanderfolgenden Adressen zu einer Sektion ist ebenfalls eine kostenneutrale Methode, Ver-

bundkonflikte zu verhindern [CuS 84]. Für den Fall, daß *m* und *s* Zweier-Potenzen sind, ist der einzige Unterschied zu der zyklischen Verteilung der Bänke auf die Sektionen, daß sich die Adressen der Sektionen aus den *ld s* Bit-Positionen beginnend bei *ld (m/s)* anstatt aus den *ld s* niederwertigen Bit-Positionen ergeben.

Abb.5.14 zeigt, im Vergleich der beiden Methoden, die Anzahl der Bank- und Pfadkonflikte bei Ausführung der synthetischen Arbeitslast für *m=8,16,32, 64* und *128* Bänke; der Speicher ist jeweils in 4 Sektionen unterteilt.



a) Bankkonflikte b) Pfadkonflikte

Adressen der Sektionen *mod s*: ——— Adressen der Sektionen *m/s*: -----

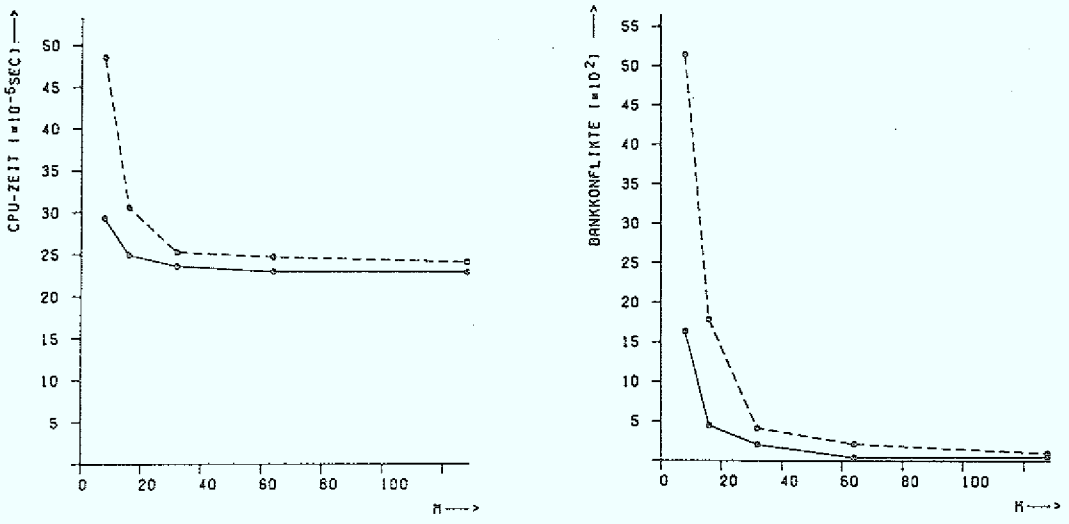
Abb.5.14 Unterschiedliche Zusammenfassung der Bänke zu Sektionen

Von all den vorgestellten Möglichkeiten, Verbundkonflikte zu verhindern, ist die letztere als optimal anzusehen, da sie einerseits ohne zusätzliche Kosten realisiert werden kann und andererseits guten Erfolg zeigt.

5.4.3 SPEICHERELEMENTE

Von den Systemen der CRAY X-MP Serie [CRA 84b] ist der Arbeitsspeicher des Ein-Prozessor-Systems mit MOS-Schaltkreisen bestückt. Deren Zykluszeit ist mit 8τ doppelt so lang wie die der ECL-Bipolar-Schaltkreise, die in den Zwei- und Vier-Prozessor-Systemen verwendet werden.

Abb.5.15 zeigt die für die Ausführung der synthetischen Arbeitslast für *m=8,16,32,64* und *128* Bänke benötigte CPU-Zeit sowie die Anzahl der Bankkonflikte; einmal für ECL-Bipolar-Speicherelemente, n_c=4, und zum Vergleich für MOS-Speicherelemente, n_c=8.



a) CPU-Zeit b) Bankkonflikte

ECL-Bipolar; n_c=4: ——— MOS; n_c=8: -----

Abb.5.15 Einsatz unterschiedlicher Speicherelemente

Es zeigt sich, daß bei Verwendung langsamerer Speicherelemente mindestens 32 Bänke für eine CPU zur Verfügung stehen sollten. Weiterhin ist erkennbar, daß selbst bei wachsender Anzahl von Bänken die Ausführungszeit und auch die Anzahl der Bankkonflikte im Falle der MOS-Speicherelemente um 10% bis 15% über den vergleichbaren Werten bei Verwendung von ECL-Bipolar-Speicherelementen liegen. Dies ist zum einen auf die längere Belegzeit der Bänke und zum anderen auf die entsprechend verlängerte Speicherzugriffszeit zurückzuführen.

6 ZUSAMMENFASSUNG UND AUSBLICK

Ziel dieser Arbeit war die Untersuchung der Vorgänge bei Zugriffen auf den Arbeitsspeicher von Vektorrechnersystemen. Dabei sind sowohl Ein- als auch Mehrprozessorsysteme, die auf einen gemeinsamen Arbeitsspeicher zugreifen, betrachtet worden.

Mit Hilfe analytischer Methoden konnten die Bedingungen ermittelt werden, unter denen ein Zugriffsstrom mit der Distanz d_1 und zwei Zugriffsströme mit den Distanzen d_1 und d_2 konfliktfrei auf einen m-fach verschränkten Arbeitsspeicher mit s Sektionen zugreifen können.

Von Bedeutung ist dabei die Rückkehrzahl, die für den i-ten Zugriffsstrom gegeben ist mit

$$r_i = m/GGT(m,d_i). \tag{6.1}$$

Ein Zugriffsstrom verläuft konfliktfrei, falls seine Rückkehrzahl größer gleich der Anzahl n_c der benötigten Takte für einen Bankzyklus ist.

In einem Speichersystem mit gleicher Anzahl s von Sektionen und Anzahl m von Bänken verlaufen zwei Zugriffsströme konfliktfrei, falls ihre beiden Zugriffsmengen disjunkt sind und $r_1, r_2 \geq n_c$. Eine notwendige Bedingung für disjunkte Zugriffsmengen ist

$$GGT(m,d_1,d_2) > 1. \tag{6.2}$$

Im Falle nichtdisjunkter Zugriffsmengen erreichen zwei Zugriffsströme unabhängig von ihrer relativen Startposition in einem Speichersystem mit gleicher Anzahl s von Sektionen und Anzahl m von Bänken einen konfliktfreien Zyklus, falls

$$GGT(m/f,(d_2-d_1)/f) \geq 2n_c \tag{6.3}$$

gegeben ist, mit $f=GGT(m,d_1,d_2)$. Es wird hier und im folgenden $d_1 | m$ angenommen; andere Werte für d_1 sind dazu isomorph (s. Anhang).

In einem Speichersystem mit kleinerer Anzahl s von Sektionen als Anzahl m von Bänken verlaufen zwei Zugriffsströme konfliktfrei, falls ihre beiden Zugriffsmengen disjunkt sind, $r_1, r_2 \geq n_c$ und

$$GGT(s, d_2 - d_1) > 1. \qquad (6.4)$$

Im Falle nichtdisjunkter Zugriffsmengen erreichen zwei Zugriffsströme unabhängig von ihrer relativen Startposition in einem Speichersystem mit kleinerer Anzahl *s* von Sektionen als Anzahl *m* von Bänken einen konfliktfreien Zyklus, falls Gl.(6.3) erfüllt ist und außerdem für beliebiges *k* gilt

$$n_c d_1 \neq ks. \qquad (6.5)$$

Falls Gl.(6.5) nicht erfüllt ist, können relative Startpositionen gefunden werden, so daß zwei Zugriffsströme mit nichtdisjunkten Zugriffsmengen in einem Speichersystem mit kleinerer Anzahl *s* von Sektionen als Anzahl *m* von Bänken konfliktfrei verlaufen, falls

$$GGT(m/f, (d_2 - d_1)/f) \geq 2(n_c + 1), \qquad (6.6)$$

mit $f = GGT(m, d_1, d_2)$. Eine geeignete relative Startposition ist durch $b_2 = b_1 + (n_c + 1)d_1$ gegeben.

Die Aussage, "es können relative Startpositionen gefunden werden" bedeutet, daß sich bei ungeeigneter relativer Startposition trotz Erfüllung von Gl.(6.6) der konfliktfreie Zyklus nicht einstellt. Stattdessen entsteht ein "Verbundkonflikt" im Wechsel zwischen Bank- und Pfadkonflikten.

Verbundkonflikte lassen sich seitens des Programmierers in der Regel nicht verhindern, da er i.allg. keinen Einfluß auf die relativen Initiierungszeitpunkte der Instruktionen hat. Einfache Veränderungen am Speichersystem verschaffen hierbei jedoch Abhilfe. Eine Möglichkeit ist die Verwendung einer zyklischen Prioritätenregelung, eine andere das Zusammenfassen von je *m/s* aufeinanderfolgender Bänke zu jeweils einer Sektion.

In der Praxis lassen sich möglichst konfliktfreie Zugriffsströme durch geeignete Dimensionierung der Datenfelder erreichen. Bezeichnet *INC* das in der FORTRAN-Schleife angegebene Inkrement, und soll der Zugriff über die *(k+1)*-te Dimension eines Feldes erfolgen, so ergibt sich die zugehörige Distanz *d* zu

$$INC \cdot \prod_{i=0}^{k} J_i \bmod m = d, \qquad (6.7)$$

wobei J_i die Anzahl der Elemente der i-ten Dimension bezeichnet, mit $J_0=1$.

Für die Anzahl m der Bänke eines Speichersystems ergibt sich die Forderung

$$m \geq pn_c, \tag{6.8}$$

falls gleichzeitig über p Speicher-Ports zugegriffen werden soll.

Eine weitere, spezielle Konfliktsituation stellt die "Barriere-Situation" dar. In diesem Fall verläuft der eine Zugriffsstrom ungestört und behindert den anderen Zugriffsstrom dergestalt, daß dieser auf eine "Barriere" trifft ohne überholen zu können und sehr stark verzögert wird. Die Barriere-Situation wird als "eindeutig" bezeichnet, wenn unabhängig von den Startbänken der Zugriffsstrom mit der Distanz d_1 ungestört verläuft und den anderen Zugriffsstrom mit der Distanz $d_2>d_1$ verzögert. Die Barriere-Situation kann bei nichtdisjunkten Zugriffsmengen und $r_1 \geq 2n_c$, $r_2>n_c$ entstehen, falls

$$(d_2-d_1)/f < n_c, \tag{6.9}$$

mit $f=GGT(m,d_1,d_2)$. Im Falle einer festen Prioritätenregelung tritt die Barriere-Situation auch bei $(d_2-d_1)/f=n_c$ auf, falls der Zugriffsstrom mit der Distanz d_1 über die höhere Priorität verfügt.

Ist Gl.(6.9) erfüllt, so stellt sich die Barriere-Situation eindeutig ein, falls

$$(n_c-1)\cdot(d_2+d_1) < m. \tag{6.10}$$

Gl.(6.10) stellt eine obere Schranke für die Mindestanzahl an Bänken dar, für die sich die Barriere-Situation auf jeden Fall eindeutig einstellt. Anhand verschiedener Kriterien (vgl. Theoreme 4.8 und 4.10) kann auch für eine kleinere Anzahl m von Bänken über die Eindeutigkeit der Barriere-Situation entschieden werden.

Die Barriere-Situation ist besonders in Vektorrechnersystemen mit mehreren CPUs, die auf einen gemeinsamen Arbeitsspeicher zugreifen, von Bedeutung. Viele numerische Verfahren lassen sich nur vektorisieren, falls im Algorithmus sogenannte "Schachbrettmuster" verwendet werden, um rekursive Datenabhängigkeiten aufzulösen. Solche Programme können, je nach Zugriffsverhalten der konkurrierenden Programme auf den anderen CPUs, in ihren Speicherzugriffen sehr stark behindert werden, was zu ent-

sprechend verlängerten Ausführungszeiten führt. Eine Barriere-Situation läßt sich durch architektonische Maßnahmen nicht verhindern, sondern lediglich durch Anpassung der Zugriffsumgebung.

Dabei kann einerseits die Möglichkeit des Multitasking in Betracht gezogen werden, um somit eine einheitliche Zugriffsumgebung herzustellen. Eine andere interessante Möglichkeit besteht darin, für die verschiedenen Datenstrukturen Abbildungsvorschriften zu entwickeln (z.B. mit Hilfe der Theorie der Skewing-Schemata), so daß alle Zugriffsströme mit der Distanz 1 zugreifen können. Weitere Untersuchungen hierzu müßten Aufschluß darüber geben, für welche Datenstrukturen solche Abbildungsvorschriften gefunden werden können und welche Implikationen deren Umsetzung in die Praxis mit sich bringen.

Die Ergebnisse der analytischen Untersuchungen wurden durch Messungen an einem Doppelprozessor-Vektorrechnersystem CRAY X-MP demonstriert. Vergleichende Simulationen lieferten darüberhinaus Auskunft über Größen, die den Messungen unzugänglich sind.

Schließlich wurden mit Hilfe des Simulators einige architektonische und technologische Alternativen analysiert. Dabei zeigte sich, daß die Zusammenfassung von je m/s aufeinanderfolgender Bänke zu jeweils einer Sektion [CuS 84], eine optimale Methode zur Verhinderung von Verbundkonflikten darstellt.

Für ein Vektorrechnersystem CRAY X-MP sollten bei Verwendung von ECL-Bipolar-Speicherelementen (Bankzykluszeit 4τ) pro CPU 16 Bänke zur Verfügung gestellt werden, bei Verwendung von MOS-Speicherelementen (Bankzykluszeit 8τ) sollten pro CPU 32 Bänke zur Verfügung gestellt werden. Aufgrund der verlängerten Speicherzugriffszeit der MOS-Speicherelemente gegenüber der Speicherzugriffszeit der ECL-Bipolar-Speicherelemente ist, auch bei genügend hoher Anzahl von Bänken, typischerweise mit einer Verlängerung der Ausführungszeit von 10% bis 15% zu rechnen.

Der Einfluß der gewählten Prioritätenregelung auf die effektive Bandbreite ist nur exemplarisch behandelt worden. An einigen Beispielen konnte gezeigt werden, daß eine zyklische Prioritätenregelung in bestimmten Situationen Vorteile gegenüber einer festen Prioritätenregelung bietet (z.B. Auflösung eines Verbundkonflikts), jedoch offensichtlich nicht im allgemeinen Fall. Um ein tiefergehendes Verständnis dieser Problematik zu erhalten, wären spezielle Untersuchungen über die Bedeutung der Prioritätenregelung nötig.

VERZEICHNIS DER VERWENDETEN SYMBOLE

$\emptyset$	leere Menge
$\cap$	geschnitten mit
$\in$	ist Element von
$\notin$	ist kein Element von
$\subseteq$	Teilmenge von
$\forall$	Allquantor
$\exists$	Existenzquantor
$a \mid b$	a ist Teiler von b
$\lceil a \rceil$	kleinste ganze Zahl $\geq a$
$\equiv$	identisch mit
$\not\equiv$	nicht identisch mit
$\leftarrow$	Zuweisung
$\rightarrow$	daraus folgt
b_{eff}	effektive Bandbreite
B_i	Startbank des i-ten Zugriffsstroms
b_i	Adresse der Startbank des i-ten Zugriffsstroms
b_w	Bandbreite
d_i	Distanz des i-ten Zugriffsstroms
D_t	Distanzvektor im Zeitpunkt t
GGT	größter gemeinsamer Teiler
ld	Logarithmus dualis
m	Anzahl Speicherbänke
mod	Modulo Funktion
M_r	Menge der Rückkehrzahlen
n	Vektorlänge
n_a	Anzahl benötigter Takte für einen Speicherzugriff
n_{az}	Anzahl benötigter Takte für den Übergang von Zustand Z_a in den Zustand Z_z
n_c	Anzahl benötigter Takte für einen Bankzyklus
n_{zz}	Anzahl benötigter Takte für die Rückkehr in den Zustand Z_z aus dem Zustand Z_z
p	Anzahl Speicher-Ports
$\prod$	Produkt
P_i	Pfadmenge des i-ten Zugriffsstroms
q	Anzahl Prozessoren (CPUs)
Q_t	Knoten des Zustandsgraphen im Zeitpunkt t
r_i	Rückkehrzahl des i-ten Zugriffsstroms
s	Anzahl Sektionen

Σ	Summe
τ	Maschinentaktzeit [sec]
t_a	Speicherzugriffszeit [sec]
t_c	Bankzykluszeit [sec]
t_g	gesamte Ausführungszeit [sec]
t_o	optimale Ausführungszeit [sec]
t_p	Restzeit bis zum Prioritätswechsel [sec]
t_v	Verzögerung der Ausführungszeit [sec]
u_m	Speicherauslastung
v_i	i-te Komponente des Restbelegungsvektors V_t
V_t	Restbelegungsvektor im Zeitpunkt t
w_i	i-te Komponente des Bankvektors W_t
W_t	Bankvektor im Zeitpunkt t
x_i	i-te Komponente des Pfadvektors X_t
X_t	Pfadvektor im Zeitpunkt t
Z	Menge der ganzen Zahlen
Z_m	Restklasse von Z *mod* m
Z_a	Ausgangszustand
z_{az}	Anzahl der innerhalb von n_{az} Takten zugelassenen Speicherzugriffe
Z_i	Zugriffsmenge des i-ten Zugriffsstroms
Z_z	zyklischer Zustand
z_t	Anzahl der bis zum Zeitpunkt t zugelassenen Speicherzugriffe
z_{zz}	Anzahl der innerhalb von n_{zz} Takten zugelassenen Speicherzugriffe

LITERATURVERZEICHNIS

AuS 83 Andrews G.R, Schneider F.B.
Concepts and Notations for Concurrent Programming
ACM Comp. Surveys, vol.15 no.1, 1983, 3-43

BHA 75 Bhandarkar D.P.
Analysis of Memory Interference in Multiprocessors
IEEE Trans. on Comp., vol.C-24 no.9, 1975, 897-908

Bua 68 Barnes G.H. u.a.
The ILLIAC IV Computer
IEEE Trans. on Comp., vol.C-17 no.8, 1968, 746-757

BuC 70 Burnett G.J., Coffman C.G.
A Study of Interleaved Memory Systems
Proc. AFIPS SJCC vol.36, 1970, 467-474

BuD 71 Briggs F.A., Davidson E.S.
Organization of Semiconductor Memories for Parallel-Pipelined Processors
IEEE Trans. on Comp., vol.C-20 no.12, 1971, 1566-1569

BuH 83 Broomell G., Heath J.R.
Classification Categories and Historical Development of Circuit Switching Topologies
ACM Comp. Surveys, vol.15 no.2, 1983, 95-133

BuK 71 Budnik P., Kuck D.J.
The Organization and Use of Parallel Memories
IEEE Trans. on Comp., vol.C-20 no.12, 1971, 1566-1569

BuM 68 Birkhoff G., Mac Lane S.
A Survey of Modern Algebra
Macmillan 1968

BuS 76 Baskett F., Smith A.J.
Interference in Multiprocessor Computer Systems with Interleaved Memory
Comm. ACM vol.19 no.6, 1976, 327-334

CKL 77 Chang D.Y., Kuck D.J., Lawrie D.H.
On the Effective Bandwidth of Parallel Memories
IEEE Trans. on Comp., vol.C-26 no.5, 1977, 480-489

CRA 82 CRAY Research Inc.
CRAY X-MP Computer Systems
CRAY X-MP Series Mainframe Reference Manual - HR-0032, 1982

CRA 83 CRAY Research Inc.
CRAY-1 and CRAY X-MP Computer Systems
FORTRAN (CFT) Reference Manual - SR-0009, 1983

CRA 84a CRAY Research Inc.
CRAY X-MP Computer Systems
Multitasking User's Guide - SN-0222, 1984

CRA 84b CRAY Research Inc.
The CRAY X-MP Series of Computer Systems
Cray Research Inc. 1984

CRA 84c CRAY Research Inc.
CRAY X-MP Computer Systems
Library Reference Manual - SR-0014, 1984

CuS 84 Cheung T., Smith J.E.
An Analysis of the CRAY X-MP Memory System
Proc. 1984 Int. Conf. on Parallel Processing, 499-505

DET 84 Detert U.
Performance Comparison for CRAY-1/S and CRAY X-MP by Means of FORTRAN Kernels and User Programs
Proc. CRAY User Group Meeting, Paris 25.-27.4. 1984, 66-76

DuB 83 Du H.C., Baer J.L.
On the Performance of Interleaved Memories with Non-Uniform Access Probabilities
Proc. 1983 Int. Conf. on Parallel Processing, 429-436

FuB 77 Folberth O.G., Bleher J.H.
Grenzen der digitalen Halbleitertechnik
NTZ, Bd.30 (1977), Heft 4, 307-314

FuM 82 Fraude K., an Mey D.
Die Vektormaschine CYBER 205
Bericht des Rechenzentrums der RWTH Aachen, Jan.1982

Gua 82 Gottlieb A. u.a.
The NYU Utracomputer - Designing a MIMD Shared-Memory Parallel Machine
Proc. 9th Annual Symposium on Computer Architecture 1982, 27-42

HEL 67 Hellerman H.
Digital Computer System Principles
McGraw-Hill 1967

HOO 77 Hoogendoorn C.H.
A General Model for Memory Interference in Multiprocessors
IEEE Trans. on Comp., vol.C-26 no.10, 1977, 998-1005

HOß 83 Hoßfeld F.
Parallele Algorithmen
Springer-Verlag 1983

HOß 84 Hoßfeld F.
Parallelrechner: Wechselwirkungen zwischen Architektur, Algorithmen und Technologie
Entwicklungsperspektiven mittlerer Rechnersysteme, Proebster und Remshardt (Hrsg.), R.Oldenbourg 1984, 269-292

HuJ 81 Hockney R.W., Jesshope C.R.
Parallel Computers
Adam Hilger 1981

JES 82 Jesshope C.R.
Programming with a High Degree of Parallelism in FORTRAN
Computer Physics Communications vol.26, June 1982, 237-246

KMC 72 Kuck D.J., Muraoka Y., Chen C.
On the Number of Operations Simultaneously Executable in FORTRAN-Like Programs and Their Resulting Speedup
IEEE Trans. on Comp., vol.C-21 no.12, 1972, 1293-1310

KNU 71 Knuth D.E.
An Empirical Study of FORTRAN Programs
Software-Practice and Experience, vol.1, 1971, 105-133

KOB 78 Kobayashi H.
Modeling and Analysis: An Introduction to System Performance Evaluation Methodology
Addison-Wesley 1978

KOG 81 Kogge P.M.
The Architecture of Pipelined Computers
McGraw-Hill 1981

KUC 77 Kuck D.J.
A Survey of Parallel Machine Organization and Programming
ACM Comp. Surveys, vol.9 no.1, 1977, 29-59

KUC 82 Kuck D.J.
High-Speed Machines and their Compilers
Parallel Processing Systems, Evans (Hrsg.)
Cambridge University Press 1982, 193-214

KuS 82 Kuck D.J., Stokes R.A.
The Burroughs Scientific Processor (BSP)
IEEE Trans. on Comp., vol.C-31 no.5, 1982, 363-376

LAK 84 Lee K.Y., Abu-Sufah W., Kuck D.J.
On Modeling Performance Degradation Due to Data Movement in Vector Machines
Proc. 1984 Int. Conf. on Parallel Processing, 269-277

LAW 75 Lawrie D.H.
Access and Alignment of Data in an Array Processor
IEEE Trans. on Comp., vol.C-24 no.12, 1975, 1145-1155

LuV 82 Lawrie D.H., Vora C.R.
The Prime Memory System for Array Access
IEEE Trans. on Comp., vol.C-31 no.5, 1982, 435-442

LuW 83 van Leeuwen J., Wijshoff H.A.G.
Data Mappings in Large Parallel Computers
Proc. GI - 13.Jahrestagung, Informatik Fachberichte,
Springer Verlag 1983, 8-20

MuU 84 Miura K., Uchida K.
FACOM Vector Processor VP-100/VP-200
NATO ASI Series, vol.F7, High Speed Computation, Kowalik (Hrsg.)
Springer Verlag 1984, 127-138

NAG 84 Nagel W.
Ein Preprocessor zur Unterstützung vektorisierender Compiler
Diplomarbeit - Lehrgebiet für Allgemeine Elektrotechnik und
Datenverarbeitungssysteme der RWTH Aachen, gedruckt als
Jül-Spez-270, Aug. 1984, Kernforschungsanlage Jülich GmbH

OED 84 Oed W.
The Treatment of Some Recursive Problems on Vector Processors
Proc. SEAS Spring Meeting, Knokke 9.-13.4., 1984, 306-313

RAU 79a Rau B.R.
Program Behavior and the Performance of Interleaved Memories
IEEE Trans. on Comp., vol.C-28 no.3, 1979, 191-199

RAU 79b Rau B.R.
Interleaved Memory Bandwidth in a Model of a
Multiprocessor Computer System
IEEE Trans. on Comp., vol.C-28 no.9, 1979, 678-681

RAV 72 Ravi C.V.
On the Bandwidth and Interference in Interleaved Memory System
IEEE Trans. on Comp., vol.C-21 no.8, 1972, 899-901

RUS 78 Russell R.M.
The CRAY-1 Computer System
Comm. ACM, vol.21 no.1, 1978, 63-72

RuW 81 Ramamoorthy C.V., Wah B.W.
An Optimal Algorithm for Scheduling Requests on Interleaved Memories for a Pipelined Processor
IEEE Trans. on Comp., vol.C-30 no.10, 1981, 787-800

SHA 78 Shapiro H.D.
Theoretical Limitations on the Efficient Use of Parallel Memories
IEEE Trans. on Comp., vol.C-27 no.5, 1978, 421-428

SOM 83 Sommer A.
Scheduling-Strategien für Pipeline-Rechner
Diplomarbeit - Lehrgebiet für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen, gedruckt als Jül-Spez-190, Jan. 1983, Kernforschungsanlage Jülich GmbH

SuD 79 Sethi A.S., Deo N.
Interference in Multiprocessor Systems with Localized Access Probabilities
IEEE Trans. on Comp., vol.C-28 no.2, 1979, 157-163

WIN 82 Winkens H.D.
Pipeline-Rechner: Architektur und Programmierung
Diplomarbeit - Lehrgebiet für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen, gedruckt als Jül-Spez-164, Juli 1982, Kernforschungsanlage Jülich GmbH

WUL 74 Wulf W.A.
C.mmp: A Multi-Mini-Processor
Computer Design: International Computer State of the Art Report Maidenhead 1974, 585-600

ANHANG

In diesem Anhang sind in Anlehnung an [BuM 68] die notwendigen mathematischen Grundlagen zusammengefaßt.

TEILBARKEIT

Alle betrachteten Zahlen sind aus $\mathbb{Z}$.

Eine Zahl b ist teilbar durch eine Zahl a, falls es eine Zahl d gibt so, daß $b=ad$. In diesem Fall schreibt man $a|b$ und nennt b ein *Vielfaches* von a und a einen *Teiler* von b. Es gilt

$$a|b \text{ und } b|c \rightarrow a|c. \tag{A.1}$$

Eine Zahl d ist der *größte gemeinsame Teiler* von a und b, d.h. $GGT(a,b)=d$, falls d ein gemeinsamer Teiler von a und b und ein Vielfaches jedes anderen gemeinsamen Teilers von a und b ist. d hat also die Eigenschaft

$$d|a;\ d|b;\ c|a \text{ und } c|b \rightarrow c|d. \tag{A.2}$$

Ist $GGT(a,b)=1$, so sind a und b *teilerfremd*.

$$GGT(a,b) = 1 \text{ und } a|c \text{ und } b|c \rightarrow ab|c. \tag{A.3}$$

$$GGT(a,b) = 1 \text{ und } b|ac \rightarrow b|c. \tag{A.4}$$

$$GGT(a,b,c) = d \rightarrow GGT(a,b,b-c) = d, \tag{A.5}$$

denn aus $d|a,b,b-c$ folgt, daß $a=df_1$, $b=df_2$ und $(b-c)=df_3$. Durch Einsetzen ergibt sich $(df_2-c)=df_3$ und daraus folgt mit $c=d(f_2-f_3)$ $d|a,b,c$.

EUKLIDISCHER ALGORITHMUS

Der euklidische Algorithmus wird dazu verwandt, den größten gemeinsamen Teiler zwischen zwei Zahlen a und b, den $GGT(a,b)$ zu finden. Dabei wird $a,b>0$ angenommen, denn negative a oder b können im Algorithmus durch $-a$ bzw. $-b$ ersetzt werden, ohne dadurch den $GGT(a,b)$ zu verändern.

Die Division von a durch b ergibt

$$a = bq_1 + r_1, \tag{A.6}$$

mit $0 \le r_1 < b$. Ist $r_1 = 0$, so ist q_1 der $GGT(a,b)$. Jede ganze Zahl, die a und b teilt, teilt auch r_1, d.h. jeder *gemeinsame* Teiler von b und r_1 ist auch ein Teiler von a in Gl.(A.6). Daher sind die gemeinsamen Teiler von a und b die gleichen wie von b und r_1 und somit

$$GGT(a,b) = GGT(b,r_1). \tag{A.7}$$

Diese Reduktion kann entsprechend mit b und r_1 fortgesetzt werden

$$\begin{aligned} b &= r_1 q_2 + r_2 \qquad 0 < r_2 < r_1 \\ r_1 &= r_2 q_3 + r_3 \qquad 0 < r_3 < r_2 \\ &\cdots \\ &\cdots \\ r_{n-1} &= r_n q_{n+1} \end{aligned}$$

Da die Reste r_i, $i=1,2,\ldots,n$ monoton fallend sind, wird schließlich ein Rest $r_{n+1}=0$ erreicht. Die Weiterführung von Gl.(A.7) ergibt

$$GGT(a,b) = GGT(b,r_1) = GGT(r_1,r_2) = \ldots = GGT(r_{n-1},r_n). \tag{A.8}$$

Daraus folgt, daß r_n selbst ein Teiler von r_{n-1} ist und somit der $GGT(a,b)$ gerade r_n ist.

Weiterhin läßt sich der $GGT(a,b)$ als eine Linearkombination von a und b darstellen

$$GGT(a,b) = k_1 a + k_2 b. \tag{A.9}$$

Dies ergibt sich, falls die jeweiligen Reste r_i sukzessive nach a und b aufgelöst werden

$$r_1 = a - bq_1 = a + (-q_1)b$$
$$r_2 = b - r_1q_2 = (-q_2)a + (1+q_1q_2)b$$
$$\ldots$$
$$\ldots$$

Die Koeffizienten k_1 und k_2 aus Gl.(A.4) enthalten entsprechend der Auflösung die Quotienten q_i, $i=1,2,\ldots,n$.

ISOMORPHIE DER DISTANZEN

Mit der Schreibweise "$d_1 \Rightarrow d_2$" wird angedeutet, daß ein Zugriffstrom mit der Distanz d_1 *gegen* einen anderen Zugriffsstrom mit der Distanz d_2 läuft.

Für d_1 brauchen lediglich solche Werte betrachtet werden, die Teiler von der Anzahl m der Bänke sind, da dadurch alle möglichen Rückkehrzahlen berücksichtigt sind. Alle anderen Kombinationen sind hierzu *isomorph*, da sie nur eine Umnumerierung der Bänke bedeuten und ergeben sich durch die Beziehung

$$d_1 \Rightarrow d_2 \equiv kd_1 \Rightarrow kd_2 \mod m, \qquad \text{(A.10)}$$

mit $GGT(k,m)=1$.

Beispiel: $m=16$

$$1 \Rightarrow 3 \equiv 5 \Rightarrow 15 \equiv 11 \Rightarrow 1 \mod 16$$
$$2 \Rightarrow 3 \equiv 6 \Rightarrow 9 \equiv 14 \Rightarrow 5 \mod 16$$