Max-Planck-Institut für Plasmaphysik

Garching bei München

Abt. Informatik, Prof. Dr. F. Hertweck

Multiprozessor mit dynamisch variabler Topologie

H.Richter

IPP R/43 27.3.1988

Gliederung

1. Einleitung

Diese Arbeit fügt sich ein in den Rahmen der Datenerfassung für das Großexperiment ASDEX-Upgrade des Max-Planck-Instituts für Plasmaphysik. Dort werden nach Inbetriebnahme des Experiments periodisch im Minutenabstand Millionen von Meßwerten anfallen, die selektiert und vorverarbeitet werden müssen. Dazu ist eine hohe Rechenleistung im experimentnahen Bereich notwendig, die kostengünstig nur mit Hilfe paralleler Architekturen erzielt werden kann. Weiterhin sind die Vorverarbeitungs-Algorithmen für die Daten in ihrem Rechenzeitbedarf sehr verschieden, so daß ein in seiner Leistung skalierbares Baukastensystem zur Datenverarbeitung vorteilhaft ist.

Der modulare Multiprozessor MULTITOP erfüllt diese Anforderungen. Darüberhinaus weist MULTITOP eine neuartige Architektur auf, deren Vorteil es ist, daß die Topologie der Prozessorverbindungen dynamisch, d.h. *während* der Abarbeitung eines parallelen Programms geändert werden kann. Diese neue Eigenschaft verleiht dem Entwickler paralleler Algorithmen ein Höchstmaß an Flexibilität und sichert zugleich eine hohe Effizienz der Programmausführung, die dadurch hervorgerufen wird, daß die Topologie der Prozessorverbindungen der Topologie des Problems angepaßt wird und nicht umgekehrt. Die bei Parallelverarbeitung immer notwendige Interprozessor-Kommunikation wird in MULTITOP dadurch gelöst, daß Erzeuger und Verbraucher direkt (ohne Zwischenstufen) verbunden sind. Somit können Daten schnell ausgetauscht werden; die für die Kommunikation benötigte Zeit bleibt kurz und damit die Effizienz des parallelen Programms hoch.

Diese Eigenschaft von MULTITOP der dynamisch variablen Topologie wird in den Kapiteln eins und zwei "Über die Architektur" näher erläutert und ihre Vorteile in den Kapiteln drei und vier über "Parallele Algorithmen" demonstriert. Zunächst wird an dieser Stelle erläutert, welche Gründe für eine Parallelverarbeitung im Rechnerbau sprechen, und wie das Konzept von MULTITOP in die Kategorien bestehender paralleler Architekturen einzuordnen ist. Zum Schluß der Einleitung werden die in dieser Arbeit neu gewonnenen Erkenntnisse stichpunktartig dargelegt.

In sequentiellen Rechnern sind im Gegensatz zu parallelen Architekturen per definitionem nur wenige funktionale Einheiten gleichzeitig aktiv. Insbesondere der Speicher weist trotz Millionen verschiedener Adressen zu einem Zeitpunkt je eine aktive Adresse auf. Eine gleichzeitige Nutzung mehrerer den Speicher bildender Bausteine bewirkt eine bessere "Ausnützung des Siliziums", wodurch die Rechenzeit eines Programms reduziert wird. Mit der Parallelverarbeitung ist es möglich, bei gleichem Aufwand an Silizium-Chipfläche wie bei einem sequentiellen Rechner, einen höheren Durchsatz zu erzielen. Ein weiteres Argument, das für Parallelverarbeitung spricht, ist die Tatsache, daß ein sequentieller Rechner prinzipiell nicht beliebig schnell werden kann. Es gibt zwei begrenzende Faktoren: Die Ausbreitungsgeschwindigkeit elektrischer Signale, die bei 20 cm/ns liegt, begrenzt bei Taktzeiten im ns-Bereich den nutzbaren Raum eines Rechner auf ein kleines Volumen. Zum anderen verhindert die bei solch kurzen Taktzeiten entstehende Wärme eine beliebige Volumenreduktion, da die Verlustleistung bei zu kleinen Volumina nicht mehr abgeführt werden kann. Ein Ausweg aus diesem Dilemma mit den Mitteln der vorhandenen Technologien

wird in der Parallelverarbeitung gesehen. Es ist möglich, dieselbe Rechenleistung wie bei einem sequentiellen Rechner durch Addition der Rechenleistung vieler kleiner und billiger Mikroprozessorchips zu erreichen, wenn es gelingt, diese effizient an *einem* Problem arbeiten zu lassen. Daraus resultiert ein Preis/Leistungsverhältnis , das es erlaubt, von der Konzeption eines Rechen*zentrums* abzugehen und sich einer dezentralen Datenverarbeitung zuzuwenden, die dort Rechenleistung erzeugt, wo sie verbraucht wird. Ein viertes Argument, das für Parallelverarbeitung spricht, ist der Umstand, daß ein Mehrprozessor-System ausfallsicherer konstruiert werden kann als ein Uniprozessor. Insbesondere kann der mögliche Ausfall eines oder mehrerer Prozessoren leicht durch zuvor eingebaute redundante Prozessoren kompensiert werden. Dies fällt bei Verzicht auf ein zentrales Bussystem umso leichter. Damit ist kein Verzicht auf große Wortbreite verbunden, wie durch MULTITOP demonstriert wird, bei dem im Prototyp-Aufbau 4x32 Bit gleichzeitig verarbeitet werden. Letztendlich liefert die belebte Natur das beste Vorbild zur Nachahmung in der Parallelverarbeitung. Ein ganzes Bild kann "in einem Blick" erfaßt oder ein gesprochener Satz in Echtzeit durch das menschliche Gehirn verarbeitet werden und das trotz einer Neuronen-Schaltzeit, die im Millisekundenbereich liegt.

Gleichwohl sind der Parallelverarbeitung auch Grenzen gesetzt. Man kann den sog. Beschleunigungsfaktor S_P gegenüber sequentieller Verarbeitung definieren als:

$$Sp = \frac{T_1}{T_p}$$

Darin ist T_1 die Zeit für ein Programm auf einem Prozessor und T_P die Zeit für dasselbe Programm, verteilt auf p Prozessoren.

Gene Amdahl formulierte als erster einen Maximalwert der Beschleunigung für den Fall der Vektorrechner. Er definierte dazu den "Parallelisierungsgrad" a als den Anteil in einem Programm, der nicht sequentiell ausgeführt werden muß. Das Amdahl´sche Gesetz kann auf Parallelrechner übertragen werden:

$$Sp = \frac{1}{(1 - a) + (a/p)}$$

Geht man jetzt z.B. von einem Programm mit 90% Parallelisierungsgrad aus, kann nach dem Amdahl´schen Gesetz selbst *bei unendlich vielen Prozessoren* ein *Beschleunigungsfaktor von maximal zehn* erreicht werden. Bei Parallelrechnern ist ein Parallelisierungsgrad von 90% leichter möglich als bei Vektorrechnern, da Parallelrechner, die auch als "*M*ultiple *I*nstruction, *M*ultiple*D*ata" -Maschinen (MIMD) bezeichnet werden, allgemeiner sind als Vektorrechner, die man als "*S*ingle *I*nstruktion, *M*ultiple *D*ata" -Maschinen (SIMD) bezeichnet.

Ein weiteres wichtiges Maß, das oft in dieser Arbeit verwendet wird, ist neben dem Beschleunigungsfaktor der Wirkungsgrad. Seine Definition lautet:

$$e = \frac{Sp}{p}$$

Beschleunigungsfaktor und Wirkungsgrad geben Auskunft darüber, ob eine Rechnerarchitektur die Abarbeitung paralleler Programme effizient unterstützt.

Wie läßt sich nun die Architektur von MULTITOP in den Rahmen bestehender Parallelrechner einordnen? Aus architektonischer Sicht ist sie ein Verbund fest gekoppelter Prozessor-Speicher-Module. Dies ist in Bild 1.1 dargestellt, das die Stellung von MULTITOP in der Hierarchie der Kopplung funktionaler Einheiten von Rechnern wiedergibt.

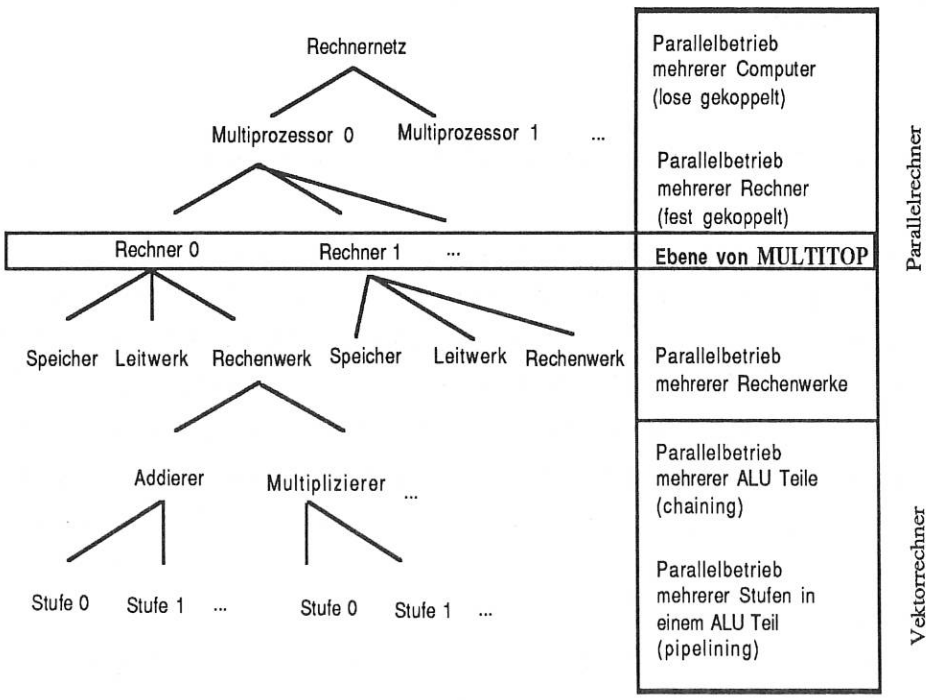


Bild 1.1:
MULTITOP ist in der Hierarchie der Kopplung funktionaler Einheiten auf der Ebene fest gekoppelter Prozessor-Speicher-Modulen anzusiedeln.

Aus logischer Sicht ist MULTITOP geeignet zur parallelen Verarbeitung mehrerer Programmeinheiten, die als Prozeduren oder Prozessen bezeichnet werden. Bild 1.2 erläutert die Position von MULTITOP in der Hierarchie von Programmen.

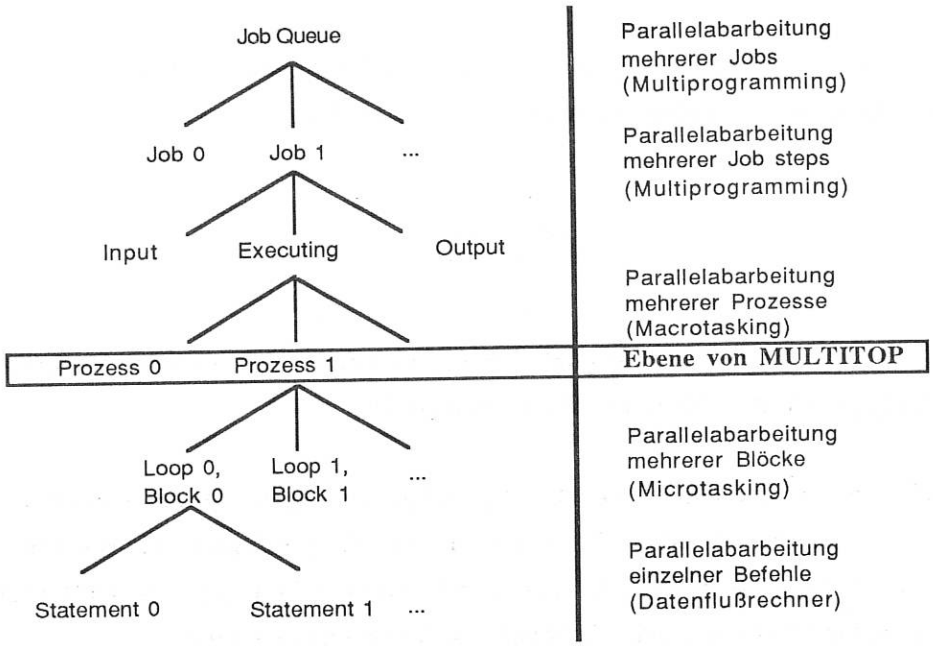


Bild 1.2:
MULTITOP ist aus logischer Sicht in der Programmhierarchie ein Rechner, der gleichzeitig mehrere Prozesse bearbeitet.

Die im Verlauf dieser Arbeit neu gewonnenen Erkentnisse lassen sich wie folgt zusammenfassen:

In Bezug auf die architektonische Seite wurde die Idee der variablen Topologie für einen Transputer-Multiprozessor eingeführt, durch den Bau von MULTITOP realisiert, und ihre Vorteile demonstriert. Der Begriff "Transputer" wird im nächsten Kapitel ausführlich erläutert. Wie bereits erwähnt wird bei MULTITOP, ähnlich wie bei einer Regelung, die Topologie der Prozessorverbindungen der dem Problem innewohnenden Topologie nachgeführt. Dadurch wird erreicht, daß der Datenaustausch zwischen den Prozessoren direkt (ohne Zwischenstufen) ablaufen kann, so daß sich eine hohe Recheneffizienz während des ganzen Programms ergibt. Die variable Topologie wird mit Hilfe eines neuartigen *modularen* Koppelnetzes realisiert, um die Forderung nach skalierbarer Rechenleistung zu erfüllen. Das Koppelnetz weist Vorteile gegenüber anderen Netzen bezüglich seiner Konstruktion und Produktion auf, da es aus zwei gleichen Modulen besteht, die darüberhinaus *ohne Änderung* als Bausteine in einem doppelt so großen Netz weiter verwendet werden können.

Zur Bewertung der algorithmischen Seite wurden verschiedene Formen der schnellen Fourier-Transformation auf ihre Parallelisierbarkeit hin untersucht und ihre Interprozessor-Kommunikation berechnet. Damit konnte das Verfahren, das den geringsten Datentransport erfordert, bestimmt werden. Dieses Verfahren stellt zugleich die am besten zu parallelisierende Form der schnellen Fourier-Transformation dar, da sich damit der geringste Zeitverlust beim Datentransport ergibt. Es wurde deshalb auf MULTITOP implementiert. Ein mathematisches Modell für die Berechnungsdauer und den Wirkungsgrad der schnellen Fourier-Transformation (FFT) wurde entwickelt und durch Messungen an MULTITOP bestätigt. Das Modell erlaubt Voraussagen über den Wirkungsgrad der FFT für sehr hohe Prozessorzahlen bei beliebigen Topologien der

Verbindungen. Schließlich wurde ein Verfahren entworfen, das in einem Schritt Meßdaten glättet und ihre Zahl auf das Wesentliche reduziert. Dazu wurde mit Hilfe von sog. Basissplines eine Ausgleichsrechnung zur Minimierung des Approximationsfehlers durchgeführt. Dieses Verfahren verwendet ausschließlich gut parallelisierbare Operationen aus der linearen Algebra wie Matrix-Vektor- und Matrix-Matrix-Multiplikationen.

In der weiteren Ausarbeitung werden die oben gemachten Aussagen zur architektonischen und algorithmischen Seite hergeleitet.

2. Die Architektur von MULTITOP

In diesem Kapitel wird der für die Parallelverarbeitung essentielle Punkt der Interprozessor-Kommunikation diskutiert, die variable Topologie erläutert und der Rechner im einzelnen beschrieben. Den Schluß dieses Kapitels bildet die Darstellung über die beim Bau des Rechners aufgetretenen elektrotechnischen Probleme.

Es gibt zwei grundlegende Arten einen Rechner zu betrachten: Zum einen aus der Perspektive des Benutzers die logische Sicht, und zum anderen aus der des Konstrukteurs die physikalische. Vom physikalischen Gesichtspunkt aus betrachtet, kann man zwei Arten der Interprozessor-Kommunikation unterscheiden:

1. Über einen gemeinsamen Speicher.
2. Über andere schaltungstechnische Hilfsmittel wie PIA, UART oder FIFO, die im weiteren als Kanäle bezeichnet werden.

Analog unterscheidet man zwei Varianten der Interprozessor-Kommunikation bei der Betrachtung aus logischer Sicht:

1. Über gemeinsame Variable.
2. Über Botschaftenaustausch.

Um die Rechenleistung in einem Multiprozessor-System effizient nutzen zu können, ist eine schnelle Interprozessor-Kommunikation notwendig. Aus diesem Grunde ist es sinnvoll, einen Parallelrechner danach zu beurteilen, wie die Kommunikation implementiert wird. Kombiniert man die oben dargestellten Varianten der Kommunikation in einer Matrix, erhält man das Bild 2.1, das eine Möglichkeit der Klassifikation von Parallelrechnern zeigt.

Gemeinsamer Speicher und gemeinsame Variable	Gemeinsamer Speicher und Botschaftenaustausch
Kanäle und gemeinsame Variable	Kanäle und Botschaftenaustausch

Bild 2.1:
Klassifikationsmatrix der Interprozessor-Kommunikation von Parallelrechnern aus logischer und physikalischer Sicht.

Viele Parallelrechner basieren auf dem Konzept der Kommunikation über einen gemeinsamen

Speicher und gemeinsame Variable. Die Vor- und Nachteile dieses Konzepts werden im folgenden gegeneinander abgewogen und daraus begründet, warum MULTITOP ein anderes Kommunikationsmodell verwendet.

2.1 Parallele Architekturen

Im Bild 2.2a wird das Prinzip der Kopplung mehrerer Prozessoren über einen gemeinsamen Speicher verdeutlicht. Der gemeinsame Speicher kann *über einen Bus oder über ein Koppelnetz* mit den Prozessoren verbunden sein. Im Allgemeinfall sind m gemeinsame Speicher vorhanden, auf die n Prozessoren zugreifen können. Bei gemeinsamem Speicher werden aus Software-Sicht zumeist auch gemeinsame Variable zur Interprozessor-Kommunikation verwendet.

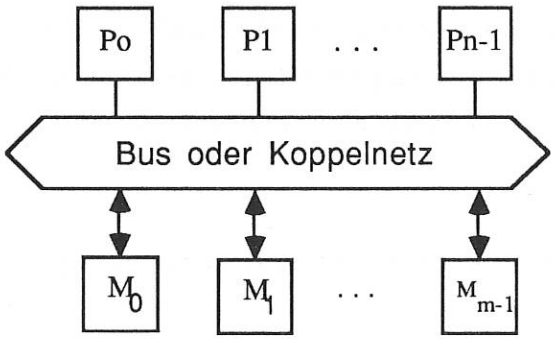


Bild 2.2a:
Die Interprozessor-Kommunikation vieler Multiprozessor-Systeme erfolgt über einen oder mehrere gemeinsame Speicher und gemeinsame Variable. Die Kopplung zwischen Speicher und Prozessoren kann über einen Bus oder ein Koppelnetz erfolgen.

Alternativ dazu ist die Kopplung über Kanäle und Botschaftenaustausch möglich, wie in Bild 2.2b verdeutlicht.

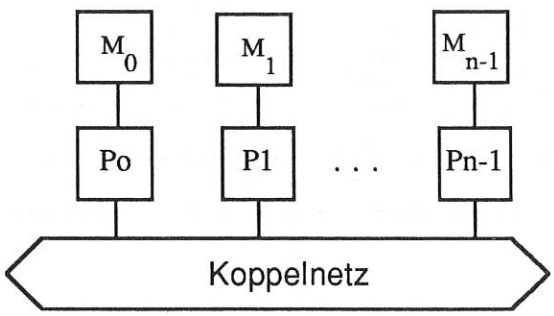


Bild 2.2b:
Prozessoren mit *lokalen Speichern* sind über ein Koppelnetz verbunden. Die Interprozessor-Kommunikation erfolgt über Kanäle und Botschaftenaustausch durch das Netz.

Die Koppelnetze selbst sind ebenfalls nicht einheitlich aufgebaut, sondern es existieren auch hier zwei grundlegende Konstruktionsprinzipien, die

1. Paketvermittlung
2. Leitungsvermittlung

genannt werden. Unter Paketvermittlung versteht man, daß ein mit einer Zieladresse versehenes Datenpaket durch das Netz geschleust wird, wobei keine feste Verbindung zwischen dem Sender und dem Empfänger besteht. Bei der Leitungsvermittlung wird, ähnlich wie bei einem Rangierbahnhof, physikalisch ein Weg zwischen Ein- und Ausgang des Netzes geschaltet, der für die Dauer der Kommunikation fest bleibt.

Bei einer Kopplung von Prozessoren über einen gemeinsamen Speicher mit Hilfe eines Koppelnetzes ist nur die Paketvermittlung sinnvoll, da sonst die Leitung für Zugriffe von verschiedenen Seiten jedesmal umgeschaltet werden müßte, was länger dauern würde als der Speicherzyklus selbst. Bei einer Kopplung über Kanäle mit Hilfe eines Koppelnetzes sind für den Datentransfer sowohl Leitungs- wie auch Paketvermittlung möglich.

2.1.1 Parallelisierung durch einen gemeinsamen Speicher und gemeinsame Variable

Bei dieser Art der Kopplung läßt die technisch begrenzte Speicherbandbreite nur eine geringe Zahl (4...8) von Prozessoren zu, die auf den gemeinsamen Speicher zugreifen können. Eine Verminderung der damit zusammenhängenden Zugriffskonflikte ist durch die Verwendung von zusätzlichen lokalen Speichern möglich. Ein Beispiel für das ein-Bus mehr-Prozessor-Konzept ist der VMEbus [VME], an den wie in Bild 2.3 dargestellt mehrere Prozessoren angeschlossen werden können:

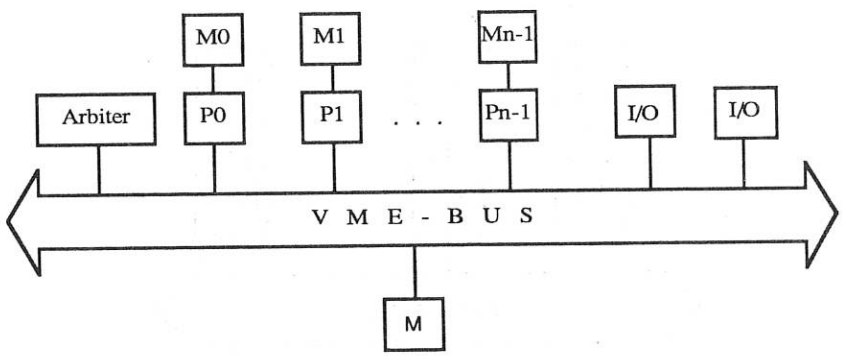


Bild 2.3:
Der VMEbus ist ein Beispiel für ein ein-Bus mehr-Prozessor-System.

Ein weiteres Beispiel für diese Art der Prozessorkopplung bildet der MULTIBUS II. Allen Buskopplungen liegt das Problem der begrenzten Speicherbandbreite und der begrenzten Busbandbreite zugrunde, die die Leistungsfähigkeit solcher Systeme auf die Leistung von ca. vier bis acht Prozessoren beschneidet. Bei mehr Prozessoren wird die Datenbelastung auf dem Bus so groß, daß eine Sättigung der Leistung eintritt. Weitere Prozessoren erbringen dann keinen Leistungszuwachs mehr. Um dem Problem der begrenzten Speicherbandbreite zu begegnen, können mehrere Busse parallel geschaltet werden. Ein Beispiel für eine solche Konfiguration ist in Bild 2.4 dargestellt:

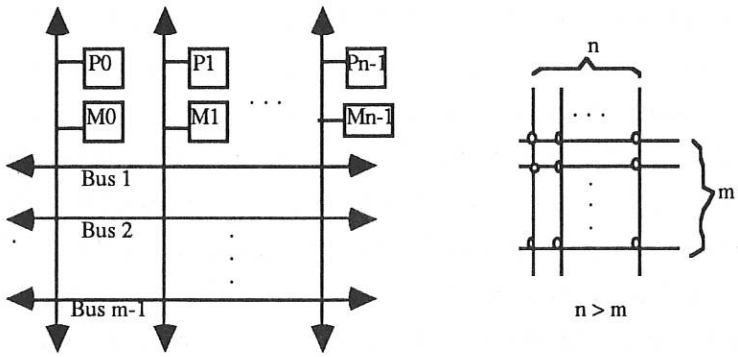


Bild 2.4:
Prinzip des Rechners POLYP (Heidelberg).
Durch Parallelschaltung von n vertikalen über m horizontale Bussen kann eine m-fache Bandbreite erzielt werden (m < n).

Diese Architektur arbeitet ähnlich einem Kreuzschienenverteiler. An jedem Kreuzungspunkt können Busse miteinander gekoppelt werden. Der Rechner POLYP der Universität Heidelberg ist so aufgebaut, dabei sind wenige (<5) vertikale und horizontale Busse realisiert. Neben den bereits erwähnten Kopplungsarten existiert analog zu einem Mehr-Bus-System das Mehrtor-Speicher-System. Es findet z.B. beim Rechner DIRMU [DIRM] der Universität Erlangen Verwendung.

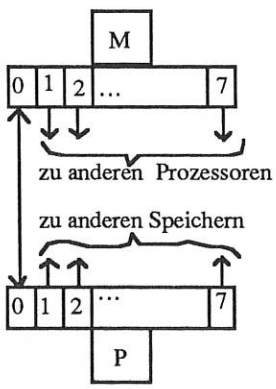


Bild 2.5:
Prinzip des Rechners DIRMU (Erlangen).
Fünfundzwanzig Prozessoren sind mit Hilfe von Achttor-Speichern gekoppelt. (Hier ist nur ein Prozessor-Speicher-Modul dargestellt.) Bei einem Achttor-Speicher kann für bis zu acht Prozessoren eine vollständige Vermaschung wie bei einem Buskonzept erreicht werden.

In Bild 2.5 ist ein Prozessor-Speicher-Modul von DIRMU dargestellt. Prozessor und Speicher weisen je acht Anschlüsse auf, davon sind sieben frei zur Kopplung mit anderen Moduln. Für bis zu acht Prozessoren kann mit diesem Konzept jeder Prozessor mit jedem verbunden werden. Dies wird als vollständige Vermaschung bezeichnet.

Bei allen soeben vorgestellten Architekturen existiert das folgende Problem: Wenn mehrere Prozessoren zugleich schreibend auf eine Variable zugreifen wollen, entsteht eine Konfliktsituation, und zwar darüber, welchen Wert die Variable annehmen soll. Eine Möglichkeit zur Vermeidung dieses Konfliktes ist, daß die Prozesse priorisiert werden. Der Prozess mit der höchsten Priorität bestimmt den Wert der Variablen. Diese Methode hat den Nachteil, daß ein Indeterminismus eingeführt wird, weil keinem schreibenden Prozess bekannt ist, ob er zur Zeit derjenige mit der höchsten Priorität ist , und ob deshalb sein Wert tatsächlich geschrieben wird. Das Problem des schreibenden Mehrfachzugriffs erinnert deshalb an das Problem eines sich selbst modifizierenden Programms, das Teile von sich überschreibt. Die damit verbundenen Probleme hinsichtlich der Testbarkeit sind bekannt. Ein anderer Weg den Mehrfachzugriff zu regeln ist, daß die Prozessoren sich gegenseitig vom Zugriff ausschließen (mutual exclusion). Dies ist nur mit zusätzlichen Einrichtungen im Rechner zu bewerkstelligen, die den Zugriff auf logischer und/oder physikalischer Ebene regeln. Will man z.B. auf physikalischer Ebene den wechselseitigen Auschluß mehrerer Prozessoren erreichen, muß ein neuer Befehl, der nicht unterbrochen werden kann, in den Befehlssatz der Prozessoren aufgenommen werden. Ein Beispiel dafür ist der TEST&SET Befehl in der 680XX Prozessorserie von MOTOROLA [MOTO]. Damit kann in *einem Schritt* der Zustand eines Bits überprüft, und, bei einem bestimmten Zustand, in einen anderen überführt werden. Wegen der Unteilbarkeit des Befehls kann kein anderer Prozessor während der Ausführung dieses Befehls einen Befehl für dasselbe Bit ausführen, so daß kein Indeterminismus entsteht. Die erwähnten Kennzeichnungs- oder Synchronisierungsbits werden als Semaphore (=Zeichenträger) bezeichnet, weil sie den Zustand einer gemeinsamen Variablen "frei für Schreiben" oder "nicht frei für Schreiben" anzeigen. Es gibt daneben andere Arten der Anzeige wie z.B. die beim HEP Rechner von DENELCOR [DENE]. Dort existiert für jede Speicherzelle eine Semaphore, die nach einem Schreiben in die Zelle automatisch auf "voll" und nach einem Lesen auf "leer" gesetzt wird. In eine volle Zelle kann nicht geschrieben, aus einer leeren Zelle nicht gelesen werden. Damit ist die Konsistenz des gesamten Speichers bei Mehrfachzugriff gesichert, man muß allerdings zusätzlichen schaltungstechnischen Aufwand in Kauf nehmen.

Die Synchronisation kann auch auf logischer statt auf physikalischer Ebene des Rechners durch ein Systemprogramm erfolgen. Ein solches Vorgehen verlangsamt aber den Zugriff auf die gemeinsame Variable erheblich, da das Monitor-Programm allgemein und damit aufwendig ist, und während seines Ablaufens die Prozessoren für Benutzerprozesse blockiert. Eine Verlangsamung des Zugriffs auf gemeinsame Variable um ein bis zwei Größenordungen im Vergleich zur hardwaremäßigen Synchronisation ist zu erwarten, da ca. zehn bis hundert Instruktionen des Monitors ablaufen müssen, bis der Zugriff geregelt ist.

Zusammenfassend liegen die Vorteile der Kommunikation über gemeinsamen Speicher und

gemeinsame Variable darin, daß:

1. Gleiche Daten und Programme nicht mehrfach gehalten werden müssen.
2. Datenzugriffe auf die gemeinsame Variable schnell sind.

Nachteile liegen in :

1. einer meist aufwendigen Synchronisation bei schreibendem Mehrfachzugriff bei vielen (> 8) Prozessoren,
2. der kleinen Zahl von koppelbaren Prozessoren, die von der Bandbreite begrenzt ist,
3. der Testbarkeit von Programmen bei schreibendem Mehrfachzugriff, die ähnlich schwierig ist wie bei einem sich selbst modifizierenden Programm.

Aus diesen Gründen kam für MULTITOP dieses Kommunikationskonzept nicht in Frage.

2.1.2 Mischformen der Kommunikation

Die Kommunikationen über gemeinsamen Speicher und Botschaftenaustausch sowie über Kanäle und gemeinsame Variable können als Mischformen bezeichnet werden. Sie haben bisher im Parallelrechner-Bau eine geringere Bedeutung erlangt. Eine dieser Mischformen ist bei UNIX-*Einprozessor-Rechnern* dagegen sehr häufig. Sie wird dort als "Kommunikation über pipes" bezeichnet und beschreibt den Datenaustausch über einen gemeinsamen Speicher und Kanäle. Der Mehrprozessor-Rechner SUPRENUM [SUPR] verwendet dasselbe Prinzip, um innerhalb eines sog. "SUPRENUM-Clusters" zu kommunizieren. Das SUPRENUM-Cluster besteht aus 16 Prozessor-Speicher-Modulen, die an einen Bus angeschlossen sind. Bei der Kommunikation über gemeinsamen Speicher und Kanälen wird das Synchronisationsproblem komplett dem Betriebssystem überlassen. Der Benutzer hat als Schnittstelle Kanäle. Die Betriebssystem- Programme müssen dafür sehr allgemein gehalten werden, um alle möglichen Benutzerfälle abzudecken. Dies wirkt sich auf die Geschwindigkeit der Datenübertragung aus. Eine Kommunikation über eine UNIX pipe z.B. benötigt auf einer RACOMP (NBI) Workstation ca. 1ms bis das Betriebssystem den Aufruf zur Kommunikation bearbeitet hat. Der Nachteil dieser Art der Kommunikation ist demnach die rel. geringe Geschwindigkeit der Datenübertragung besonders bei kleinen Datenmengen, die durch einen von der Datenmenge unabhängig (hohen) Verwaltungsaufwand begrenzt wird. Dieser Nachteil wird bei der Kommunikation über Kanäle mit Botschaftenaustausch vermieden, weil es dabei keinen Zugriffskonflikt auf einen gemeinsamen Speicher mehr gibt, und deshalb auch keinen Zeitverbrauch mit der Synchronisation. Zum anderen kann, weil die Kommunikation konfliktfrei ist, der Benutzer *direkt* (ohne ein Betriebsystem) auf seine Kanäle zugreifen. Ein Zeitverbrauch durch ein Betriebssystem somit auch hier nicht vor. Die sog. "Hypercube"-Rechner [HYPE] sind wohl z.Z.die häufigsten Parallelrechnertypen; sie werden von namhaften Firmen wie INTEL vermarktet. Diese Rechner verwenden genau das Prinzip der Kommunikation über Kanäle und Botschaftenaustausch. Auch der mit 2,5 GFLOPS

Dauerrechenleistung z.Z. wohl leistungsfähigste Parallelrechner, die "Connection Machine" [HILL] arbeitet nach diesem Kommunikationsprinzip.

2.2 Das Kommunikationskonzept von MULTITOP

Im Rahmen dieser Arbeit sollte unter anderem eine neue parallele Rechnerarchitektur mit auf dem Markt erhältlichen Mikroprozessoren entworfen werden. Weiterhin war und ist das geplante Einsatzgebiet von MULTITOP die schnelle Datenvorverarbeitung in einem plasmaphysikalischen Experiment, die eine *hohe Gleitkomma- Rechenleistung* erforderlich macht. Der Transputer (*Trans*istor + Com*puter*) der Fa. INMOS [IMS] erschien als Prozessor besonders geeignet, weil abzusehen war, daß eine Transputer-Version mit schnellem Gleitkomma-Rechenwerk *auf dem Chip* erscheinen wird. Bei dieser Lösung entfällt das Problem des schnellen Datentransfers zwischen Gleitkomma-Rechenwerk und Prozessor wie es bei einer mehr-Chip-Lösung auftritt. Der Transputer bevorzugt aufgrund seines inneren Aufbaus ein Kommunikationskonzept mit Kanälen und Botschaftenaustausch. Mit der Wahl des Transputers als Prozessorelement für MULTITOP stand auch das Kommunikationskonzept fest: *Kanäle und Botschaftenaustausch.*

Für dieses Kommunikationskonzept sprechen weitere Gründe:

1. Die Fehlertoleranz bei Hardware- oder Softwaredefekten ist größer als bei der Kommunikation über einen gemeinsamen Speicher. Da Daten und Programme lokal gespeichert sind, können sich Fehler zunächst nur lokal auswirken. Es existieren auch *viele* Punkt-zu-Punkt-Verbindungen, so daß bei Ausfall eines oder mehrerer Kanäle die Leistungsfähigkeit des Rechners nur langsam abnimmt (graceful degradation). Weiterhin können mit Hilfe des Koppelnetzes *zur Laufzeit* Umkonfigurationen vorgenommen werden, um so defekte Prozessoren auszublenden. Ausfälle des Koppelnetzes können nach einem Vorschlag von H.J.Siegel [SIEG] dadurch vermieden werden daß im Netz redundante Stufen vorhanden sind, die bei Bedarf dazugeschaltet werden.

2. Die Prozessorleistung ist auch zu hohen Leistungen hin gut skalierbar, da die Kommunikations-Bandbreite mit der Zahl der Kanäle zunimmt. Vorausetzung dazu ist ein skalierbares Koppelnetz, wie es in Kapitel drei vorgestellt wird.

3. Parallele Programme, die dieses Konzept verwenden, können leichter als bei anderen Konzepten getestet werden, da jeder Kanal eine eindeutig definierte Schnittstelle mit bestimmtem Protokoll darstellt. Durch Beobachten der Botschaften auf allen Schnittstellen kann im Fehlerfall viel Information über die Art des Fehlers gewonnen werden.

Nachteilig wirkt sich bei der Architektur der über Kanäle gekoppelten Prozessor-Speicher-Moduln aus, daß Kanäle meist langsamer als gemeinsame Speicher arbeiten und außerdem, daß gemeinsame Programme und Daten u.U. mehrfach gespeichert werden müssen. Letzteres wird

allerdings infolge des Preisverfalls bei Speichern bedeutungslos. Nach der Begründung für das gewählte Kommunikationsmodell wird jetzt auf den Kern der Architektur von MULTITOP eingegangen, die *dynamisch variable Topologie.*

2.3 Die dynamisch variable Topologie

Als variable Topologie eines Rechners bezeichnet man die Eigenschaft, daß die Kanäle der Prozessoren nicht fest verbunden sind, sondern daß ihre Topologie frei gewählt werden kann. Der Zusatz "dynamisch" heißt, daß die Topologie *während* der Ausführung eines Programms umgeschaltet werden kann.

Die folgenden Darstellungen über die parallelen Algorithmen von schneller Fourier-Transformation (FFT) und Spline-Approximation zeigen, warum eine variable Topologie sinnvoll ist.

Schnelle Fourier-Transformation

Es gibt zwei Formen der schnellen Fourier-Transformation, deren Interprozessor- Kommunikation, wie in Kapitel 4 nachgewiesen wird, minimal ist. Dadurch eignen sich diese besonders zur Parallelisierung, weil auch der mit der Kommunikation verbundene Zeitverlust minimal wird. Der Signalflußgraph einer dieser Formen ist im nächsten Bild für den Fall von 16 zu transformierenden Werten dargestellt.

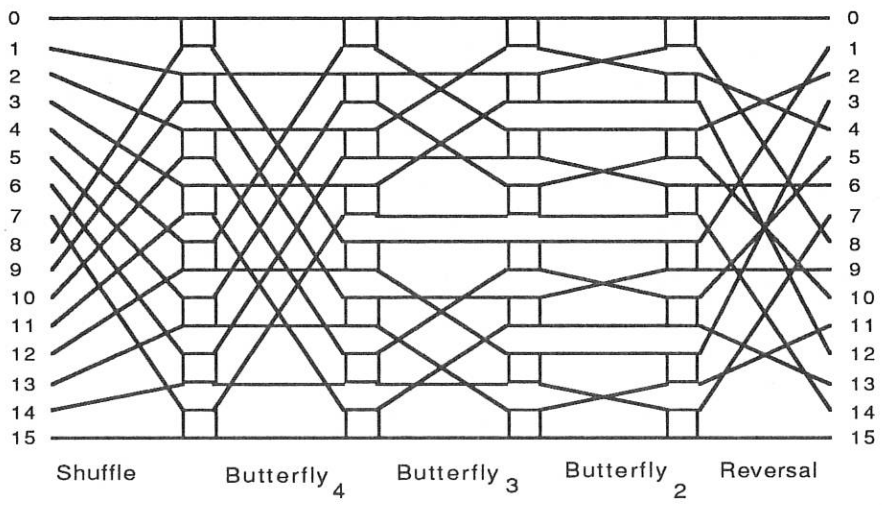


Bild 2.6:
Der Signalflußgraph einer schnellen Fourier-Transformation (FFT) mit minimaler Interprozessor-Kommunikation (Gentlemen-Sande FFT). Die Permutationsfunktionen zwischen den Stufen sind verschieden.

Man sieht, wie sich die Topologie des Signalflußes von Stufe zu Stufe ändert. Ordnet man zur Parallelisierung einzelne Zeilen des Graphen verschiedenen Prozessoren zu, müssen diese in der

folgenden Weise miteinander verbunden sein:

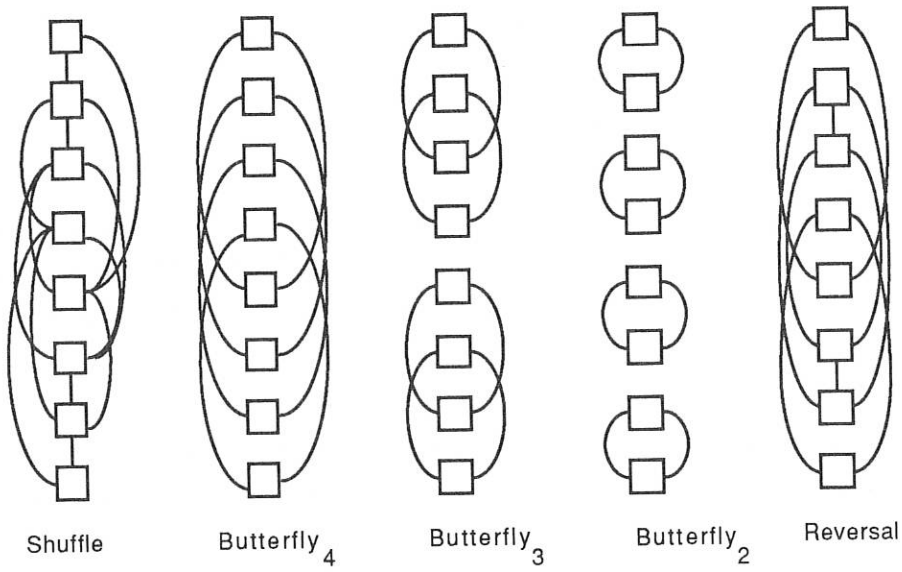


Bild 2.7:
Die zum Signalflußgraphen der FFT von Bild 2.6 gehörenden Prozessorverbindungen in jeder Stufe des Graphen bei zeilenweise Parallelisierung für den Fall von acht Prozessoren. Die Topologie der Verbindungen ändert sich von Stufe zu Stufe: Shuffle, Butterfly und Reversal.

Aus den Bildern 2.6 und 2.7 wird deutlich, daß es bei der gezeigten FFT drei verschiedene Topologien gibt, die sich nicht auseinander ableiten lassen: Shuffle, Butterfly und Reversal. Aus diesem Grunde gibt es auch mehrere Möglichkeiten, auf einem Rechner mit Botschaftenaustausch die FFT zu implementieren:

1. Es wird die Vereinigungsmenge aller Topologien verwendet, um die Prozessoren zu koppeln.

2. Es wird nur die Shuffle-Topologie verwendet, da nur sie einen zusammenhängenden Graphen ergibt. Dies bedeutet für die zwei verbleibenden Topologien, daß Daten über Zwischenstufen transportiert werden müssen, da für diese kein direktes Abbild in der Prozessortopologie existiert.

3. Es wird eine Topologie verwendet, die bezüglich Aufwand und Kommunikation zwischen Lösung 1 und 2 liegt und einen Kompromiß darstellt wie z.B. die Hypercube-Topologie.

4. Die Topologie ist variabel. In jeder Stufe der FFT wird die zu dieser Stufe passende Topologie der Prozessorverbindungen eingestellt.

Bei der ersten Methode resultiert aus der Vereinigung der Topologien ein komplexer Graph. Die Zahl der Links pro Prozessor wächst ungefähr proportional zur Zahl der Prozessoren. Dies wird

Zahl der Links pro Prozessor wächst ungefähr proportional zur Zahl der Prozessoren. Dies wird deutlich, wenn man die Bilder 2.6 und 2.7 für eine doppelt so große Zahl von Punkten und Prozessoren zeichnet. Die physikalische Implementierung eines solchen Graphen ist nicht möglich, weil bei realen Prozessoren die Zahl der Links pro Prozessor begrenzt ist.

Gegen die Anwendung der zweiten und dritten Möglichkeit spricht die Tatsache, daß bei der FFT die Zahl der zu transportierenden Daten der Zahl der zu berechnenden Werte entspricht. Eine Kommunikation über Zwischenstufen beeinflußt deshalb die Effizienz der Kommunikation und damit des gesamten Programms wesentlich. Die Höhe die Beeinflussung wird durch folgende Rechnung abgeschätzt:

Sei T_{Kom} die Zeit für Kommunikation und T_{alg} die Zeit für Berechnung, dann ergibt sich für die Effizienz eines parallelen Programms:

$$e = \frac{T_{alg}}{T_{alg} + T_{kom}}$$

Im Falle der FFT ist:

$$T_{Kom} \approx T_{alg}.$$

Daraus erhält man:

$$e \approx 50\ \%.$$

Dies bedeutet, daß für eine hohe Effizienz (> 50 %) die Methoden zwei und drei nicht anwendbar sind.

Die vierte Methode ist die *variable Topologie*. Für jede Stufe der FFT wird die Topologie der Prozessorverbindungen eingestellt, die dem Signalfluß dieser Stufe entspricht. Dies ist aus der Sicht der Algorithmen die ideale Lösung, da der Unterschied zwischen der Topologie des Algorithmus und des Rechners aufgehoben wird. Die Interprozessor-Kommunikation erfolgt dabei ohne Zwischenstufen also *direkt*. Die Effizienz der Programmausführung bleibt hoch, weil die Datentransporte minimal sind. Dazu muß allerdings die Umschaltzeit zwischen den Topologien klein sein im Vergleich zur Rechenzeit pro Topologie, d.h., es muß schnell umgeschaltet werden. Im einzelnen sei T_U die Umschaltzeit, T_{Spez} die Rechenzeit bei einer dem Problem angepaßten Topologie und T_{allg} die Rechenzeit bei einer allgemeine, festen Topologie wie z.B. einem Gitter (nearest neighbour). Ein Gewinn an Rechenzeit ergibt sich nur dann, wenn

$$T_U + T_{Spez} < T_{allg}$$

ist. Bei MULTITOP kann eine bei der Übersetzung des Programms bereits bekannte Topologie in ca. 10 µs eingestellt werden. Dies ist eine sehr kurze Zeit im Vergleich zur Berechnungsdauer einer FFT, die z.B. bei 1024 zu transformierenden Werten auf einem Transputer 500 ms beträgt. Eine erst zur Laufzeit bekannte Topologie benötigt zum Berechnen und Einstellen der Wege ca. 4 ms. In beiden Fällen lohnt sich demnach das Umschalten der Topologie.

Spline-Glättung

Neben der FFT ist die parallele Spline-Approximation als weiterer Algorithmus, der effizienter bei dynamisch variabler Topologie wird, zu nennen. Dort ist im wesentlichen das folgende lineare Gleichungssystem zu lösen:

$$\mathbf{A}^T\mathbf{A}\mathbf{p} = \mathbf{A}^T\mathbf{y}$$

Darin ist **A** die Matrix, die sich aus den Meßdaten **y** und den Stützstellen des Splines ergibt, die Unbekannte **p** enthält die Gewichtungsfaktoren von der Linearkombination aus Basissplines, wie sie in Kapitel 5 hergeleitet wird.Die Lösung dieses Gleichungssystems nach **p** erfordert sechs Schritte, bei denen sich von Schritt zu Schritt die Topologie des Signalflußes ändert:

1. Matrix **A** aufstellen:	gitterförmige Topologie
2. **A** transponieren:	shuffle Topologie
3. Matrix-Vektor multipl.:	Kette
4. Matrix-Matrix multipl.:	Gitter
5. lineare Gleich. lösen:	Kette oder Gitter
6. **p** ausgeben:	Stern

Aus dem bisher Dargelegten geht hervor, daß

1. die FFT und die Spline Approximation völlig verschiedene Topologien haben, die sich auch nicht voneinander ableiten lassen.

2. sich *innerhalb* der FFT oder der Spline-Approximation die optimale Topologie von Phase zu Phase des Algorithmus ändert.

Damit läßt sich das folgende Ergebnis formulieren:

> Die parallele schnelle Fourier-Transformation und die Spline-Approximation haben bezüglich der Minimierung der Interprozessor-Kommunikation eine optimale Folge von Topologien der Prozessorverbindungen.

Diese Aussage kann allgemeiner gefaßt werden, nämlich daß für *jeden* parallelen Algorithmus eine optimale Folge von Topologien existiert. In diesem Zusammenhang sei auf die Literatur über parallele Algorithmen wie [HOSS], [SCHE] oder [HELL] verwiesen.

Das Konzept von MULTITOP

Welche Architektur muß nun ein Multiprozessor-System aufweisen, das völlig verschiedene Topologien gleichermaßen gut unterstützten soll? Zunächst ergibt sich für eine solche hypothetische Architektur aufgrund des gewählten Kommunikationsmodells, das auf Kanälen mit Botschaftenaustausch basiert, daß die Prozessoren über Punkt-zu-Punkt-Verbindungen miteinander verbunden werden müssen. Weiterhin ergibt sich, daß insbesondere keine gemeinsamen Speicher vorhanden sind, so daß jeder Prozessor lokal Daten und Programme speichern muß. Der Verbund von Prozessor und lokalem Speicher soll als Prozessor-Speicher-Modul (PM) bezeichnet werden. Unter diesen Voraussetzungen läßt sich ein erstes grobes Bild der Architektur entwerfen, das in der Abbildung 2.8 dargestellt ist:

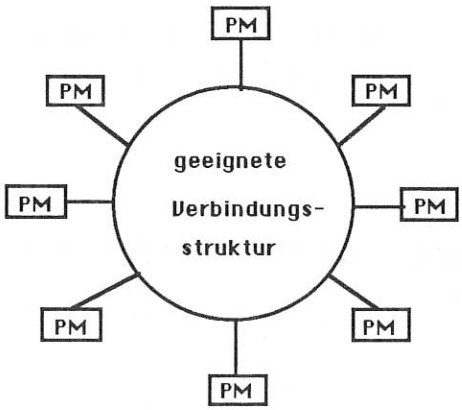


Bild 2.8:
Mehrere Prozessor-Speicher-Module (PMs) sind geeignet gekoppelt, um alle möglichen Punkt-zu-Punkt-Verbindungen zu realisieren.

Diese grobe Architektur kann dadurch verfeinert werden, daß die Art der Verbindungsstruktur näher spezifiziert wird. Eine naheliegende Möglichkeit, alle Punkt-zu-Punkt-Verbindungen zu realisieren ist, die Prozessor-Speicher-Module vollständig miteinander zu vermaschen, wie dies im Bild 2.9 gezeigt ist:

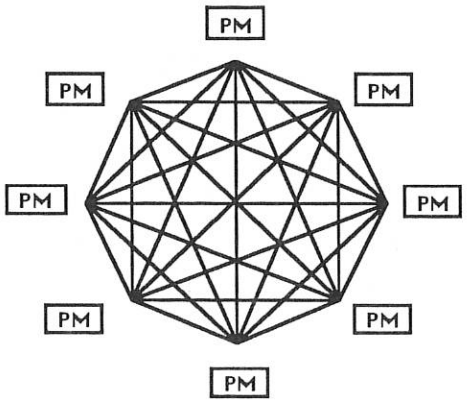


Bild 2.9:
Die vollständige Vermaschung aller Prozessor-Speicher-Module realisiert alle möglichen Punkt-zu-Punkt-Verbindungen.

Damit können alle möglichen Topologien gleich gut von der Architektur des Rechners unterstützt werden. Bei N Prozessoren ergeben sich N Links pro Prozessor und v = (1/2)N(N-1) Verbindungen insgesamt. Diese Lösung ist für große N deshalb nicht anwendbar, da ihr Aufwand mit $O(N^2)$ steigt.

In realen Programmen sind dagegen nie alle v Verbindungen gleichzeitig aktiv, so daß eine wirtschaftlichere Lösung darin besteht, nur die momentan tatsächlich genutzten Verbindungen zu realisieren. Diese zweite Lösung zielt also darauf ab, bei Bedarf, Prozessorlinks zu Verbindungen zusammenzuschalten.

Führt man weiterhin eine feste Zahl L von Links pro Prozessor ein (L << N), und unterteilt man jedes Prozessorlink in einen Eingabe- und Ausgabeteil, dann ist die einfachste Möglichkeit, Links zu Verbindungen zusammenzuschalten, einen Kreuzschienenverteiler zu verwenden. Dieser ist in Bild 2.10 zusammen mit den dazu gehörenden PMs dargestellt:

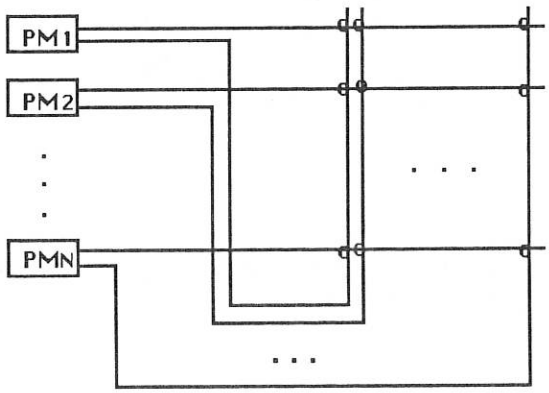


Bild 2.10:
Die vollständige Vermaschung kann mit Hilfe eines Kreuzschienenverteilers nachgebildet werden, indem bei Bedarf je zwei Prozessorlinks zu einer Verbindung zusammengeschaltet werden. (Im Bild ist der Fall L = 1 gezeigt.)

Da der Kreuzschienenverteiler aus $O(N^2)$ Schaltern besteht, ist er aus denselben Gründen wie der vollständig vermaschte Graph unbrauchbar. Die Lösung des Wirtschaftlichkeitsproblems wird im nächsten Kapitel anhand eines sog. Koppelnetzes vorgenommen, das mit O(NlogN) Schaltern auskommt. Ein blockierungsfreies Koppelnetz ermöglicht N! verschiedene Punkt-zu-Punkt-Verbindungen, so daß ebenfalls jeder Prozessor mit jedem verbunden werden kann. Damit läßt sich eine verfeinerte Architektur für die Lösung des Topologieproblems angeben. Sie ist in Bild 2.11 graphisch dargestellt.

Die Einführung von schaltbaren Punkt-zu-Punkt-Verbindungen mit Hilfe eines Koppelnetzes unterstützt in wirtschaftlicher Weise alle möglichen Topologien gleich gut, indem die Topologie der Prozessorverbindungen der Topologie des Problems angepaßt wird.

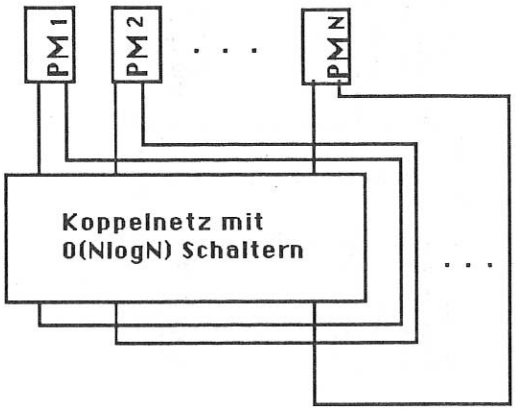


Bild 2.11:
Eine wirtschafltiche Lösung des Problems, verschiedene Topologien in einer Architektur zu unterstützen, liegt in der Verwendung eines Koppelnetzes, das mit O(NlogN) Schaltern auskommt. (Wiederum ist der Fall L = 1 gezeigt.)

Allerdings ist der mit der Durchschaltung der Verbindungen verbundene Zeitverlust, der durch das Setzen der Schalter des Netzes entsteht, so klein zu halten, daß er bei der Programmausführung nicht wesentlich ins Gewicht fällt. Dies führt zu der Idee, die Schalter des Netzes *parallel*, d.h. von mehreren Prozessoren gleichzeitig, setzen zu lassen. Im Prinzip könnten dazu dieselben Prozessoren PM1 - PMN verwendet werden, die auch dem Benutzerprogramm zur Verfügung stehen. Dieses würde dann allerdings in der Zeit, in der die Berechnung der Schalterstellungen vorgenommen wird, nicht ausgeführt werden können. Im nächsten Kapitel wird gezeigt, daß das Berechnen der Stellungen für eine neue Verbindung eine nichtriviale Aufgabe ist, die auf einem Transputer z.B. 4 ms Zeit benötigt. Um diesen Zeitverlust bei jeder neuen Verbindung zu vermeiden, können zusätzliche Prozessor-Speicher-Module (PM1´ - PMM´) eingesetzt werden, deren Zahl M wesentlich kleiner als N sein kann.Die M Module müssen gemäß dem Algorithmus des Schalter-Berechnens miteinander verbunden sein. Diese Art der Verbindung kann fest sein, da stets dasselbe Programm ausgeführt wird. Geeignet ist eine Baum- oder Gitter-Topologie. Aufgrund dieser Überlegungen läßt sich die Architektur weiter verfeinern (Bild 2.12).

Bei den bisherigen Betrachtungen wurde die Datenein-/ausgabe nicht berücksichtigt. Eine schnelle Verarbeitung ist allerdings ohne eine ebensolche Datenein-/ausgabe wirkungslos. Bei Rechnern erfolgt sie zumeist von einer zentralen Stelle aus - der Schnittstelle zur Außenwelt. An diesem Punkt werden Eingabedaten eingespeist, die danach auf alle Prozessoren verteilt werden müssen. Nach der Verarbeitungsphase werden Ergebniswerte an dieser Stelle gesammelt und von dort ausgegeben. Aus Gründen der Auslastung ist es zweckmäßig, die zentrale Datenein-/ausgabe mit Hilfe derselben Prozessor-Speicher-Moduln vorzunehmen, die auch für das Koppelnetz zuständig sind, da diese nur während der Verarbeitungsphase mit dem Herstellen neuer Topologien beschäftigt sind. Dazu müssen sie in einer bestimmten, festen Topologie mit den Benutzer-Moduln verbunden sein. Für das Verteilen und Sammeln von Daten eignet sich eine sternförmige Topologie besonders, im Zentrum des Sterns steht die Schnittstelle zur Außenwelt. Somit ist dem Aufbau von MULTITOP noch ein Stern von festen Verbindungen überlagert, wie es das Bild 2.13 zeigt:

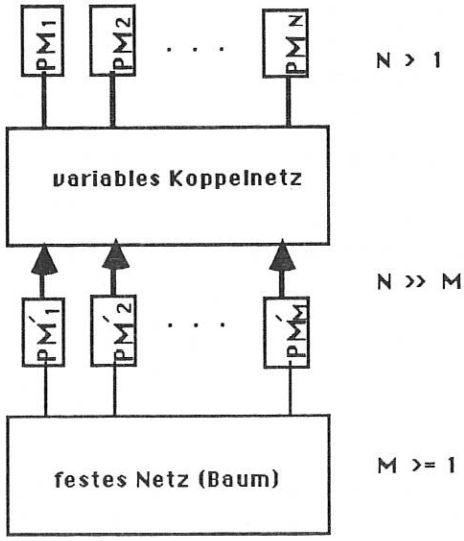


Bild 2.12:
Um Zeitverluste beim Berechnen und Setzen der Schalter des Netzes zu vermeiden, werden zusätzliche Prozessor-Speicher-Module (PM1´- PMM´) eingesetzt, die fest miteinander verbunden sein können, da sie nur einen einzigen Algorithmus ausführen.

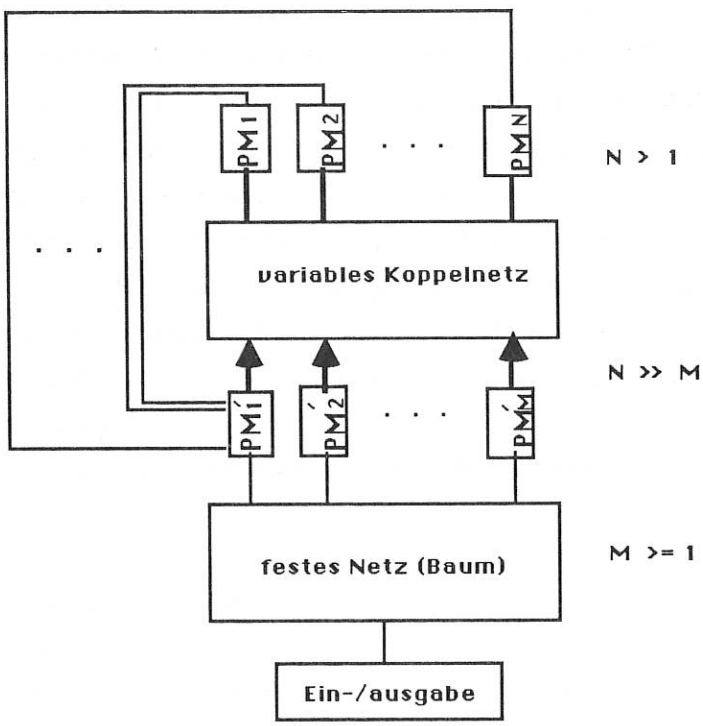


Bild 2.13:
Prinzipschaltbild des MULTITOP-Konzepts. Die Architektur umfaßt neben den Rechenmoduln PM1 - PMN weitere Module für die Schalter des Netzes, sowie eine sternförmige Topologie zur Datenein-/ausgabe.

Damit ist die Herleitung der Architektur von MULTITOP abgeschlossen. Im nächsten Kapitel wird die wesentliche Komponente der Architektur diskutiert: das modulare Koppelnetz.

3. Das modulare Koppelnetz des MULTITOP-Rechners

In diesem Kapitel wird die Topologie des modularen Koppelnetzes hergeleitet. Die Modularität des Netzes ist die Voraussetzung für ein in seiner Leistung skalierbares Multiprozessorsystem. Bei dem hier vorgestellten Netz ist die Modularität in doppeltem Sinne vorhanden:

1. Das Netz selbst besteht aus zwei gleichen Teilen (Moduln)
2. Das nächst größere Netz, mit der doppelten Zahl von Eingängen, enthält *zwei* Netze mit diesen Moduln als Bausteine.

Die erste Art der Modularität bietet den Vorteil billigerer Produktion, da die Stückzahl sich verdoppelt. Die zweite Art der Modularität ist für eine Systemerweiterung, die ohne Änderung der vorhandenen Teile vollzogen werden kann, unerläßlich. Diese Eigenschaften ergeben sich aus der Topologie des MULTITOP-Netzes, das aus zwei gleichen, hintereinandergeschalteten "Baseline"-Netzwerken besteht. Zum besseren Verständnis werden zuerst bestehende blockierungsfreie Netze und deren Eigenschaften dargestellt. Ein Netz heißt dann blockierungsfrei, wenn es stets einen Weg durch das Netz von einem freien Eingang zu einem freien Ausgang gibt. Blockierungsfreie Netze werden auch als vollständig erreichbare Netze bezeichnet.

3.1 Bestehende Koppelnetze mit vollständiger Erreichbarkeit

Der Kreuzschienenverteiler

Zum ersten Mal wurden Koppelnetze in der Telephonvermittlungstechnik eingesetzt, wie z.B. der Kreuzschienenverteiler (crossbar):

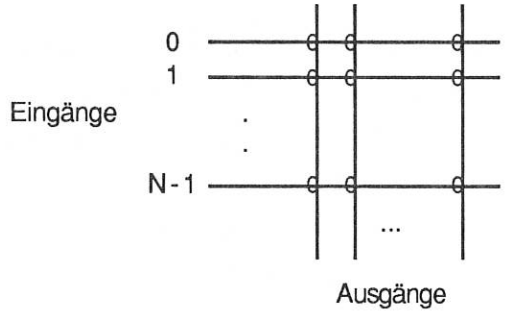


Bild 3.1:
Der Kreuzschienenverteiler: An jedem Kreuzungspunkt ist ein Schalter, der einen Eingang mit einem Ausgang verbinden kann. Es ist stets möglich, einen Weg von einem Eingang zu einem Ausgang herzustellen.

Der Kreuzschienenverteiler hat den Vorteil, daß es stets einen Weg von einem freien Eingang zu einem freien Ausgang durch den Verteiler gibt. Er wird deshalb auch als "strikt blockierungsfrei"

bezeichnet. Nachteilig ist seine sehr große Zahl von N^2 Schaltern bei N Eingängen. Für 1000 Eingänge z.B. müssen bereits 1 Million Schalter aufgewendet werden.

Das Clos-Koppelnetz

Man kann dadurch Schaltelemente sparen, daß man statt eines großen Matrixfeldes mehrere kleinere Kreuzschienenverteiler verwendet, die z.B. in der Art von C.Clos [CLOS] gekoppelt werden:

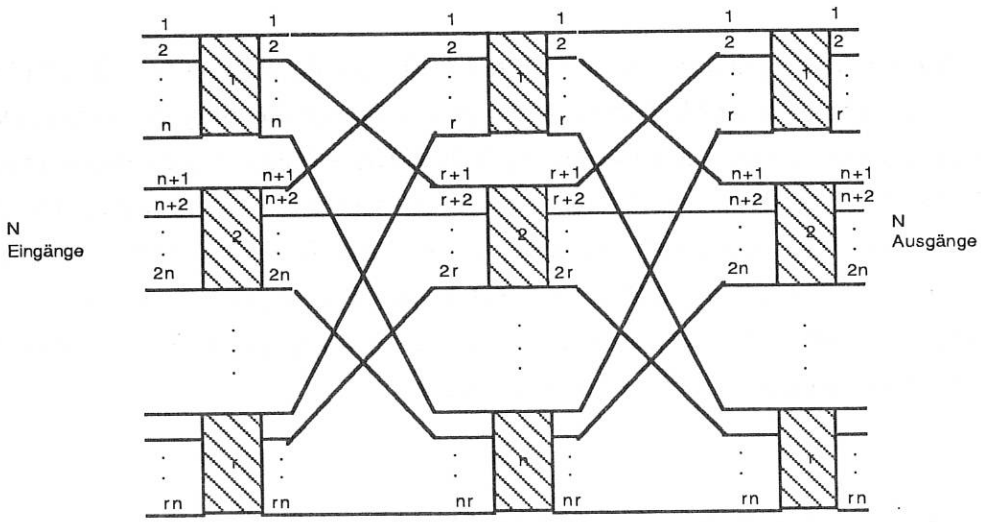


Bild 3.2:
Das Clos Koppelnetz (1953). Durch Umordnen bestehender Wege durch das Netz kann stets eine neue Verbindung zwischen einem Eingang und einem Ausgang hergestellt werden.

Das Clos Koppelnetz hat die Eigenschaft der strikten Blockierungsfreiheit verloren, aber Clos hat gezeigt, daß es stets eine Konfiguration von Schalterstellungen gibt, um alle möglichen Verbindungen von Eingängen mit Ausgängen zu realisieren. Dazu sind u.U. bestehende Verbindungen im Koppelnetz auf anderen Wegen durch das Netz zu legen. Diese Eigenschaft wird als "blockierungsfrei durch Umordnen" der internen Wege bezeichnet. Dadurch wurde eine erheblich Reduktion der Zahl der Schalter möglich, wie das Beispiel von 1024 Eingängen zeigt:

$$N = 2^{10}$$

Kreuzschienenverteiler	Clos Netz
2^{20}	$3*2^{15}$

Das Benes-Koppelnetz

Eine weitere deutliche Reduzierung der Schalter gelang V.E. Benes 1962 [BENE] mit einem Netzwerk, wie es in Bild 3.3 dargestellt ist.

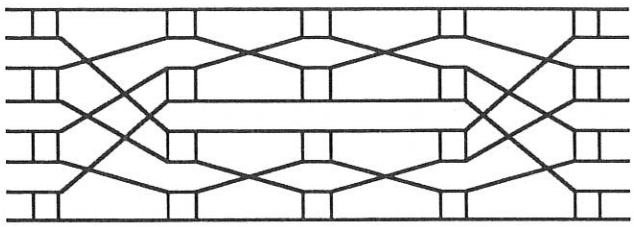


Bild 3.3:
Das Benes Netz (1962). Jedes Kästchen beinhaltet einen sog. Kreuzschalter, der entweder parallelen Durchgang (=) oder gekreuzten Durchgang (x) aufweist. Das Netz ist blockierungsfrei durch Umordnen interner Wege.

Man erhält für die Zahl der hier notwendigen Schalter:

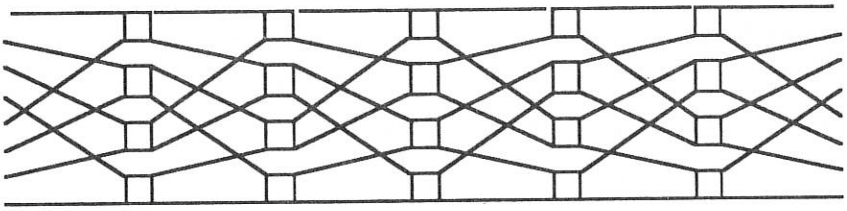
$$N = 2^{10}$$

Clos Netz	Benes Netz
$96*2^{10}$	$19*2^{10}$

Das Benes Netz weist für eine technische Realisierung einen Nachteil auf: aufgrund seiner Spiegelsymmetrie zur mittleren Stufe ist der rechte Teil und der linke Teil des Netzes verschieden, da bei einer Realisierung z.B. mit Gattern Eingänge mit Ausgängen vertauscht sind. Deshalb müssen für ein Benes Netz *zwei verschiedene Moduln* entworfen und gebaut werden. Die linke Hälfte des Benes-Netzes wird auch als "Baseline"-Netzwerk bezeichnet. Ein Baseline Netzwerk hat bereits die Eigenschaft, daß es von jedem Eingang zu einem Ausgang einen Weg durch das Netz gibt, aber nicht für alle Eingänge gleichzeitig! Ein Baseline Netzwerk allein ist deshalb nicht blockierungsfrei.

Das Lee-Koppelnetz

K.Y.Lee [LEE] stellte 1985 ein weiteres Koppelnetz vor, das die gleiche Zahl von Schaltern wie das Benes Netz hat, aber nicht mehr dessen leichte Erweiterbarkeit auf eine größere (doppelte) Zahl von Eingängen aufweist, da seine Verdrahtung zwischen den Stufen für jede mögliche Zahl von Eingängen neu gemacht werden muß, wie an der in Bild 3.3 dargestellten Topologie zu sehen ist. Ein weiterer Nachteil liegt in der hohen Vermaschung der Verdrahtung zwischen den einzelnen Stufen, die bei einer technischen Realisierung des Netzes auf Platinen eine höhere Zahl von Steckverbindern nötig macht als beim Benes Netz, da dort die Lokalität der Verbindungen zwischen den Stufen von Stufe zu Stufe zunimmt.

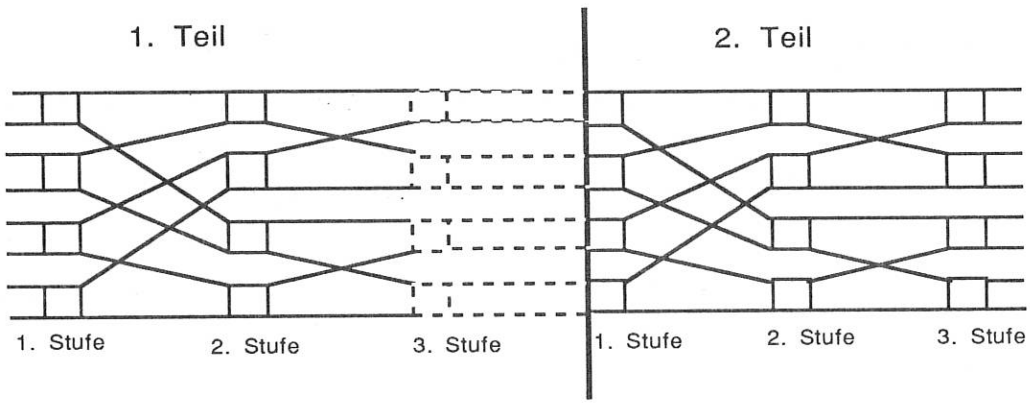


Aufwand: O(2NlogN-N)

Bild 3.4:
Das Lee Netz (1985). Durch Umordnen blockierungsfreies Netz. Zwischen den Stufen existiert eine starke Durchmischung der Verdrahtung.

3.2 Das MULTITOP-Koppelnetz

Das modulare Koppelnetz des MULTITOP-Rechners vermeidet bei *gleicher Zahl von Schaltern* wie beim Benes- oder Lee-Netz die starke Vermaschung des Lee-Netzes, und es besteht, anders als das Benes Netz, aus zwei gleichen Moduln. Zusätzlich weist es dieselbe Erweiterbarkeit wie des Benes-Netzes auf, die bewirkt, daß bei einer Verdopplung der Zahl der Eingänge ins Netz die vorhandene Teile weiterverwendet werden können. Diese Eigenschaften ergeben sich aus der Topologie des MULTITOP-Netzes, das aus zwei gleichen, hintereinandergeschalteten Baseline-Netzwerken besteht, wie sie in Bild 3.5 zu sehen sind:



Aufwand: O(2NlogN-N)

Bild 3.5:
Ein modulares Koppelnetz (Bsp.: N = 8): Das Netz besteht aus zwei gleichen, hintereinander geschalteten Baseline-Netzwerken. Die letzte Stufe des linken Teils ist redundant und kann deshalb weggelassen werden.

Das Koppelnetz besteht aus zwei gleichen Moduln. Die letzte Stufe des linken Teils ist redundant und wurde deshalb weggelassen. Die zwei Algorithmen dagegen, nach denen die Schalter der beiden Moduln gesetzt werden, sind völlig verschieden. Im weiteren werden die beiden Teile des Netzes und ihre Schalter-Algorithmen hergeleitet, und es wird die Blockierungsfreiheit des Netzes für alle N! Permutationen bewiesen.

3.3 Das Modell des Speichers aus Adressen und Inhalten

Die Verbindungen zwischen Ein- und Ausgängen eines Koppelnetzes kann man mit Hilfe eines Permutationsvektors definieren:

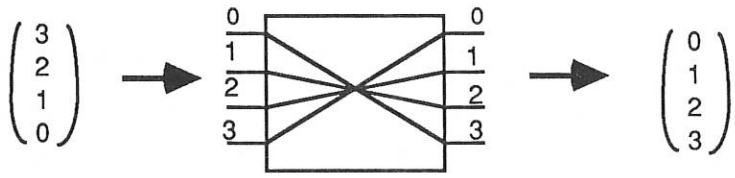


Bild 3.6:
Beispiel für einen Permutationsvektor von Verbindungen. Zu verbinden sind: Eingang 0 mit Ausgang 3, Eingang 1 mit Ausgang 2, Eingang 2 mit Ausgang 1 und Eingang 3 mit Ausgang 0.

Eine äquivalente Betrachtungsweise für die Funktion eines Koppelnetzes ist es, wenn man das Netz als eine Maschine zum *Sortieren von Zahlen* ansieht. Für das obige Beispiel angewandt würde das bedeuten, daß man ein Signal mit dem Zahlenwert 3 in den Eingang 0, ein Signal mit dem Wert 2 in den Eingang 1 usw. einzuspeist. Am Ausgang des Netzes erscheinen dann alle Signalwerte von 0 bis 3 ihrem Wert nach an den Ausgängen von 0 bis 3 sortiert. Die Fähigkeit, Zahlen zu sortieren, führt zu der Modellvorstellung eines Speichers mit Zahlen, die sortiert werden müssen. Der Speicher hat addressierbare Zellen, die den verschiedenen Eingängen des Netzes entsprechen:

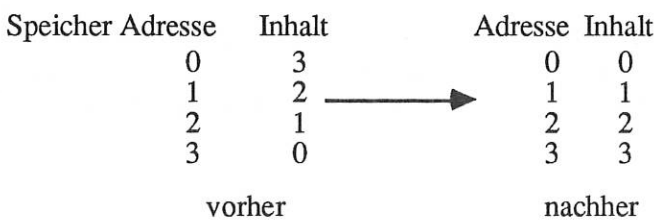


Bild 3.7:
Die Funktion eines Koppelnetzes kann man auch anhand eines Modells studieren. Das Modell besteht aus einem linear adressierbaren Speicher, dessen Inhalt sortiert werden soll.

Im Allgemeinfall sind N Zahlen zu sortieren, die eine Permutation von 0,1,...,N-1 sind. Das Problem des Koppelnetzes kann deshalb auf das Analogon, in einem Speicher Zahlen zu sortieren, übertragen werden, was einfacher zu diskutieren ist. Es ist nun ebenfalls zweckmäßig, Adressen und ihre *Inhalte binär anzugeben*, da sich dann Aussagen, wie " die 2 Eingänge eines Kreuzschalters ", oder " die obere und untere Hälfte der Eingänge " mit dem nieder- bzw. höchstwertigen Adressbit identifizieren lassen. Genauso läßt sich der Vorgang des Sortierens von Inhalten nach Adressen in einem Speicher besser binär veranschaulichen, da bitweise sortiert wird.

Im folgenden wird daher von Adressen der Kreuzschalter und von Inhalten von Adressen gesprochen. Weiterhin sei N stets eine Zweierpotenz: $N = 2^n$. Da die zwei Algorithmen zum Einstellen der Schalter völlig verschieden sind, betrachten wir beide Teile getrennt.

3.4 Der rechte Teil des modularen Koppelnetzes

Der rechte Teil des Netzes wird mit NR bezeichnet. Das Prinzip von NR besteht darin, daß in jeder Stufe nach je einem Bit des Inhalts entschieden wird, ob der Inhalt in die obere oder untere Hälfte der nachfolgenden Stufe wandert. Bei einer Wortbreite des Inhalts von n Bit nähert man sich so schrittweise der Adresse, die gleich dem Inhalt ist, und erreicht sie nach n Stufen. Dies wird auch als Prinzip der sukzessiven Approximation bezeichnet. Das folgende Bild verdeutlicht diesen Entscheidungsprozess für die erste Stufe:

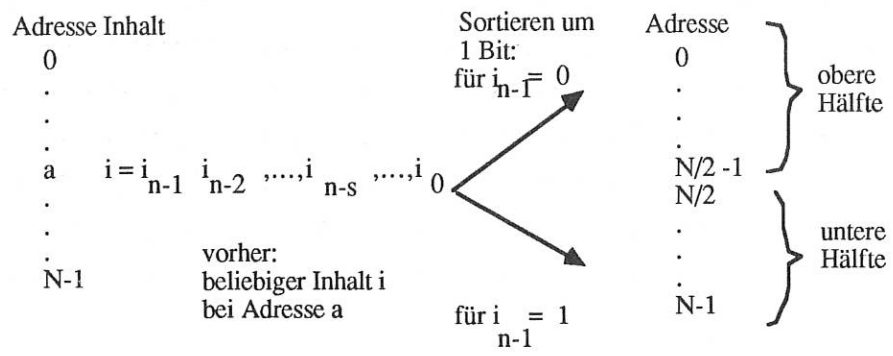


Bild 3.8:
Sortiervorgang in der 1. Stufe. Das 1. Bit des Inhalts entscheidet, ob der Inhalt in den oberen oder unteren Adressbereich der nachfolgenden Stufe transportiert wird. Das Beispiel läßt sich auf eine beliebige Stufe s übertragen. Der Adressraum der nachfolgenden Stufe s+1 ist dann 2^{n-s} Bits. Voraussetzung ist, daß $N = 2^n$ und $i = \{0,1,...,N-1\}$ ist.

3.4.1 Die Implementierung der sukzessiven Approximation mit Hilfe von Kreuzschaltern und der Unshuffle-Permutation

Das Sortieren nach dem Prinzip der sukzessiven Approximation wird mit Hilfe von Kreuzschaltern und der Unshuffle-Permutation S^{-1} realisiert, die, wie in Bild 3.9 dargestellt ist, gerade und ungerade Adressen in verschiedene Hälften verteilt. Schaltet man eine solche Verdrahtung hinter eine Spalte von Kreuzschaltern, erreicht man dadurch, daß je nach Stellung eines Kreuzschalters der Inhalt einer Adresse in den oberen oder unteren Adressbereich der nachfolgenden Stufe transportiert wird. Für die erste Stufe ist dieser Vorgang in Bild 3.10 dargestellt. Dies läßt sich auf eine Stufe i verallgemeinern: In der Stufe i werden gerade und ungerade Adressen eines Adress*teilbereichs* in die obere und untere Hälfte diese Bereichs verteilt.

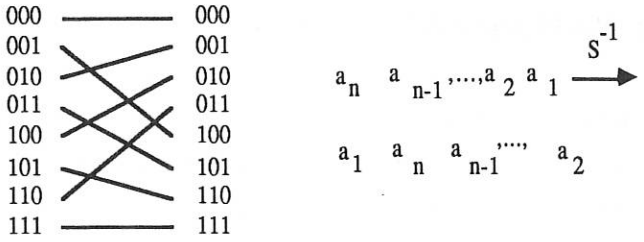


Bild 3.9:
Die Unshuffle Permutation S^{-1} läßt sich durch eine zyklische Rotation der Adressbits a_i um eine Position nach rechts darstellen. Im Beispiel ist für die erste Stufe der Fall von n = 3 dargestellt.

Die Definition von S^{-1} in der i.Stufe ist so, daß nur die i niederwertigen Bits der Adressen in 2^{n-i} Adressteilbereichen dieser Permutation unterworfen werden.

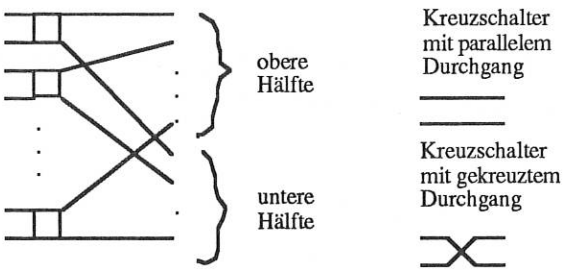


Bild 3.10:
Implementierung eines Schrittes der sukzessiven Approximation durch eine Stufe des Koppelnetzes. Im Bild ist die erste Stufe dargestellt.

3.4.2 Die Topologie des rechten Teils des Koppelnetzes

Durch schrittweises Verteilen eines Inhalts in die obere oder untere Hälfte des Adressbereichs der nachfolgenden Stufe kann man einen Inhalt zu der Adresse transportieren, die gleich ihrem Inhalt ist, d.h. einen Inhalt sortieren. Bei einem Inhalt mit n Bits Wortbreite sind dazu n Stufen erforderlich, so daß sich damit die folgende Topologie von NR ergibt:

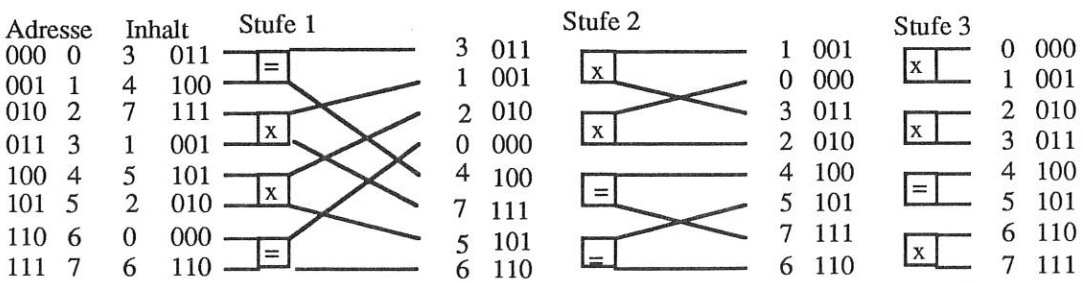


Bild 3.11:
Der rechte Teil des modularen Koppelnetzes für N = 8 und einem Permutationsvektor (3,4,7,1,5,2,0,6)

3.4.3 Bedingung für blockierungsfreies Arbeiten

An jedem Kreuzschalter liegen *zwei* Inhalte an, die sortiert werden müssen. Dazu müssen die zwei für diese Stufe relevanten Bits der zwei Inhalte komplementär sein, da aufgrund der Verdrahtung zwischen den Stufen Inhalte von einem Schalter stets in *verschiedene Hälften* transportiert werden. Die Bedingung für blockierungsfreies Arbeiten ist somit, daß alle Paare von Inhalten vor allen Schaltern ein, für die jeweilige Stufe relevantes, *komplementäres Bit* haben müssen. Daraus erhält man einen Satz von Bedingungen. Dieser Satz soll nun formal spezifiziert werden. Dazu numeriert man die Stufen des rechten Teils des Netzes von links nach rechts mit den Zahlen von 1 bis n. Die Zahl der Eingänge muß zusätzlich eine Zweierpotenz sein ($N = 2^n$). Weiterhin sei in binärer Darstellung

$$A = a_{n-1}a_{n-2},\ldots,a_1 0$$
$$A' = a_{n-1}a_{n-2},\ldots,a_1 1$$

ein Adresspaar vor der Stufe s, das zu den beiden Eingängen eines Kreuzschalters gehört. Diese benachbarten Adressen A, A´ sollen die Inhalte I(s), J(s) haben:

$$I(s) = i_{n-1}i_{n-2},\ldots,i_1 i_0$$
$$J(s) = j_{n-1}j_{n-2},\ldots,j_1 j_0$$

Der nachfolgende Tatbestand ist nun entscheidend:

Vor jeder Stufe s müssen die beiden Inhalte I(s) und J(s) s-1 gleiche höherwertige Bits haben, sonst wären sie dort nicht benachbart, da sie bereits s-1 Stufen durchlaufen haben und entsprechend ihrer s-1 Bits sortiert wurden!

Es gilt also für die Stufe s:

$$i_{n-1} = j_{n-1}$$
$$i_{n-2} = j_{n-2}$$
$$\ldots$$
$$i_{n-(s-1)} = j_{n-(s-1)}$$

Weiterhin muß das s. Bit der Inhalte für konfliktfreies Arbeiten komplementär sein:

Stufe s: $i_{n-s} = /j_{n-s}$, ("/" heißt: komplementär)

Damit erhält man für I(s), J(s):

$$I(s) = i_{n-1},\ldots,i_{n-s}i_{n-(s+1)}i_{n-(s+2)},\ldots,i_0$$
$$J(s) = i_{n-1},\ldots,i_{n-s}/i_{n-(s+1)}j_{n-(s+2)},\ldots,j_0$$

Beispiel: a) Vor der Stufe n (=letzte Stufe) liegen an einem Kreuzschalter Inhalte mit n-1 gleichen Bits an. Ihr niederwertigstes Bit ist komplementär.

b) Vor der ersten Stufe ist das höchstwertige Bit zweier benachbarter Inhalte komplementär.

Die Frage ist nun, welche Auswirkungen die Bedingungen der Stufe s auf den Eingang von NR haben. Diese Frage ist einfach zu beantworten:

Am Eingang von NR müssen zwei Inhalte mit s-1 gleichen höchstwertigen Bits und komplementärem Bit i_{n-s} *so weit voneinander getrennt* liegen, daß sie genau *in der Stufe s benachbart* werden. Dann ist die Bedingung für konfliktfreies Arbeiten der Stufe s erfüllt.

3.4.4 Der Begriff der Distanz

Dies führt auf den Begriff der Distanz als ein Maß für die Entfernung zweier Inhalte im Koppelnetz. Beispiel:

Die Inhalte I(s) = 000 mit A(s) = 000
J(s) = 001 mit B(s) = 111

haben die Distanz 2^2, da sie nach 2 Stufen benachbart sind.

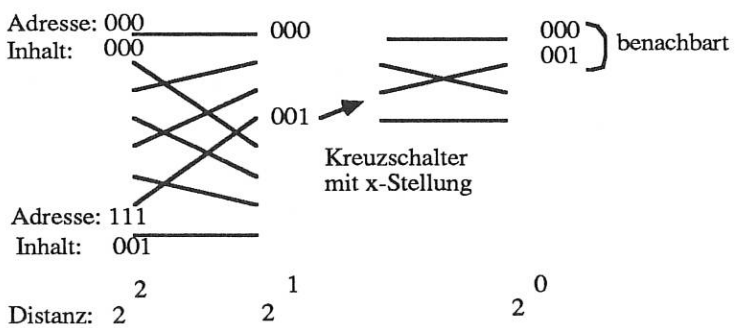


Bild 3.12:
Die Inhalte I(s), J(s) haben die Distanz 2^2 im Adressraum, da sie nach zwei Stufen benachbart sind.

Allgemein gilt für eine Distanz 2^s:

Zwei Inhalte I(s), J(s) haben am Eingang des Netzes die Distanz 2^s, wenn sie nach s Stufen benachbart sind.

Für die Adressen dieser Inhalte am Eingang des Netzes bedeutet dies, daß ihr *erstes unterschiedliches Adressbit* von links das Bit mit der Wertigkeit 2^s sein muß. I(s) und J(s) haben somit genau dann die Distanz 2^s, wenn für ein beliebiges I(s) bei der Adresse

$A(s) = a_{n-1},\ldots,a_{s+1}a_s a_{s-1},\ldots,a_0$ der Inhalt

$I(s) = i_{n-1},\ldots,i_{n-s}i_{n-(s+1)}i_{n-(s+2)},\ldots,i_0$ anliegt.

Und bei der Adresse

$B(s) = a_{n-1},\ldots,a_{s+1}/a_s b_{s-1},\ldots,b_0$ der Inhalt
$J(s) = i_{n-1},\ldots,i_{n-s}/i_{n-(s+1)}j_{n-(s+2)},\ldots,j_0$ anliegt.

J{s) hat mit I(s) s gleiche höherwertige Bits, das s+1. Bit ist komplementär, und die restlichen n-(s+1) Bits sind beliebig. Die zu J(s) gehörende Adresse B(s) dagegen hat mit A(s) n-(s+1) gleiche höherwertige Bits, das n-s. Bit ist komplementär, und die restlichen s Bits sind beliebig. Liegen diese Inhalte I(s), J(s) an *Eingängen* A(s), B(s) *von NR* an, sind sie nach s Stufen, d.h. vor der s+1. Stufe benachbart. Da die Zahl der Adressen gleich der Zahl der Eingänge in das Netz eine Zweierpotenz ist, und die Inhalte eine Permutation der Adressen sein sollen, existiert zu jedem Inhalt I(s) der dazu gehörende Inhalt J(s). Dieses J(s) hat aber im allgemeinen **nicht die Adresse B(s)**, so daß ein J(s) mit einer Adresse B(s) eine *Bedingung* darstellt.

3.5 Der linke Teil des Netzes

NL sei der Name des linken Teils des Netzes. Die Aufgabe von NL ist es, blockierungsfreies Arbeiten von NR zu ermöglichen. NL leistet diese Aufgabe durch ein Vorsortieren der Inhalte. Das Vorsortieren der Inhalte hat das Ziel, daß ein Inhaltspaar I(s), J(s) mit s gleichen höchstwertigen Bits am Eingang von NR die Distanz 2^s aufweist. Im Netz NR treffen sich dann I(s) und J(s) nach s Stufen vor einem Kreuzschalter der s+1. Stufe. Da ihr s+1. Bit ist komplementär ist, kann dieser Kreuzschalter konfliktfrei arbeiten. Das ganze Netz NR arbeitet dann konfliktfrei, wenn dies für alle Kreuzschalter in allen Stufen s = 1,2,...,n gilt. Die Aufgabe von NL ist es also, alle Paare I(s), J(s) in allen Stufen s mit

$$I(s) = i_{n-1},\ldots,i_{n-s}i_{n-(s+1)}i_{n-(s+2)},\ldots,i_0$$
$$J(s) = i_{n-1},\ldots,i_{n-s}/i_{n-(s+1)}j_{n-(s+2)},\ldots,j_0$$

so weit zu trennen, daß sie die Distanz 2^s haben.

Das Trennen von I(s) und J(s) auf die Distanz 2^s durch NL ist die gegenläufige Operation zum Zusammenführen von I(s) und J(s) durch NR nach s Stufen der sukzessiven Approximation. Wenn zuerst ein Trennen der Inhalte I(s) und J(s) mit s gleichen höherwertigen Bits auf die Distanz 2^s stattfindet, und dann ein Zusammenführen nach s Stufen, dann kann der Koppelschalter der s+1. Stufe von NL konfliktfrei arbeiten, da die s+1. Bits der Inhalte komplementär sind. Nach der s+1. Stufe trennen sich die Wege von I(s) und J(s) wieder, da sie dann in verschiedene Hälften des Adressbereichs der nachfolgenden Stufe transportiert werden. Das Prinzip von NL läßt sich graphisch anhand des blockierungsfreien Weges eines Paares I(s) und J(s) durch das Koppelnetz illustrieren:

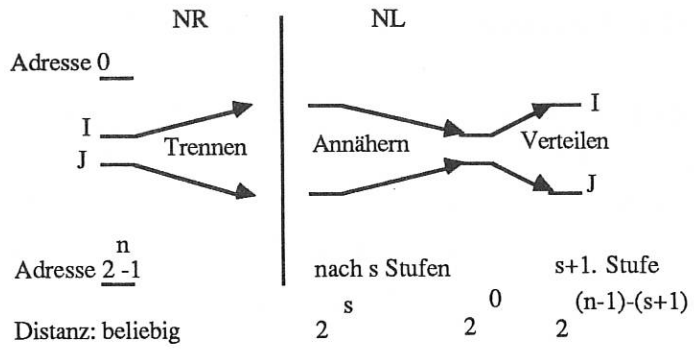


Bild 3.13:
Der blockierungsfreie Weg eines Paares I(s), J(s) mit s gleichen höherwertigen Bits durch das Koppelnetz.

Beispiel: a) Um blockierungsfrei vor der Stufe n (=letzte Stufe) zu sein, müssen I(s) und J(s) am Eingang von NR die Distanz 2^{n-1} haben.
($I(s) = i_{n-1}i_{n-2},\ldots,i_1i_0$, $J(s) = i_{n-1}i_{n-2},\ldots,i_1/i_0$)

b) Um blockierungsfrei vor der 1. Stufe zu sein, müssen I(s) und J(s) am Eingang von NR die Distanz 2^0 haben.
($I(s) = i_{n-1}i_{n-2},\ldots,i_1i_0$, $J(s) = /i_{n-1}j_{n-2},\ldots,j_1j_0$)

3.5.1 Trennen von I(s) und J(s) auf die Distanz 2^s

Zwei Inhalte I(s) und J(s) mit

$$I(s) = i_{n-1},\ldots,i_{n-s}i_{n-(s+1)}i_{n-(s+2)},\ldots,i_0 \quad \text{und}$$
$$J(s) = i_{n-1},\ldots,i_{n-s}/i_{n-(s+1)}j_{n-(s+2)},\ldots,j_0$$

erhalten die Distanz 2^s, indem man sie in einem Adressraum von s+1 Bits in die obere und untere Hälfte des Adressraums verteilt. Dabei ist es irrelevant, welcher Inhalt in welche Hälfte kommt.

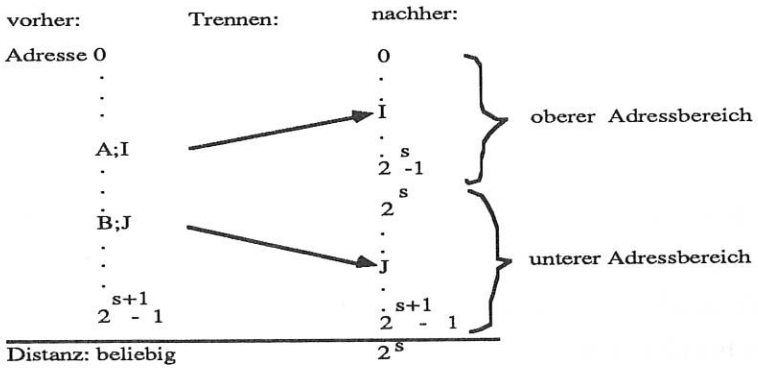


Bild 3.14:
Einstellen der Distanz 2^s von zwei Inhalten I(s), J(s) durch Verteilen in die zwei Hälften eines Adressraums von s+1 Bits.

3.5.2 Einstellen aller Distanzen durch mehrfaches Trennen

Das Einstellen aller Distanzen für alle Paare I(s), J(s) geschieht in mehreren Schritten. Zuerst wird für alle Paare die maximale Distanz 2^{n-1} (bei 2^n verschiedenen Adressen) eingestellt. Dazu wird I(s) und J(s) in verschiedene Hälften verteilt. In diesen Hälften wird das Verteilen in verschiedene Hälften rekursiv fortgesetzt, und damit nacheinander die Distanzen 2^{n-2}, 2^{n-3} usw. eingestellt. Wiederum ist es irrelevant, welcher Inhalt in welche Hälfte kommt:

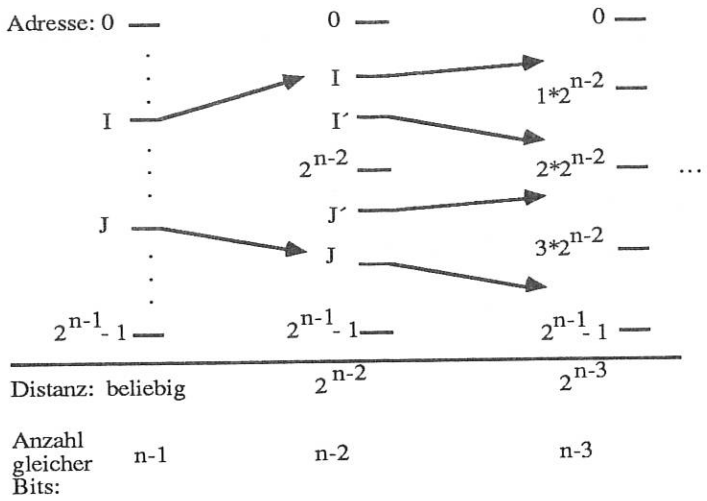


Bild 3.15:
Für s = n-1, n-2,...,1 werden alle Paare I(s), J(s) mit s gleichen höherwertigen Bits in obere und untere Adressbereichshälften verteilt, um alle Distanzen einzustellen.

Die Distanz 2^0 braucht nicht explizit hergestellt zu werden, da ein Trennen in obere und untere Hälfte in einem Adressraum von 2^{0+1} Bits implizit erfüllt ist, für je ein Paar I(s),J(s), da es in diesem Adressraum nur zwei Adressen gibt. Aus diesem Grunde kann in NL die letzte Stufe entfallen, da zur Erreichung aller Distanzen nur n-1 Stufen nötig sind.

In der ersten Stufe von NL, in der die Distanz 2^{n-1} eingestellt wird, gibt es, bei 2^n verschiedenen Inhalten, *zu jedem Inhalt I(s) genau einen Inhalt J(s)* mit n-1 gleichen höherwertigen Bits. Das n. Bit muß komplementär sein, sonst wären I(s) und J(s) identisch. Dieses Paar wird dann *in verschiedene Hälften verteilt*, so daß in diesen Hälften *kein Paar mit n-1 gleichen höherwertigen Bits* vorkommt.

In der zweiten Stufe von NL gibt es in jeder Hälfte 2^{n-1} verschiedene Inhalte und somit zu jedem I(s) genau ein J(s) mit n-2 gleichen höherwertigen Bits. Weniger als n-2 gleiche Bits gibt es nicht, weil in jeder Hälfte 2^{n-1} verschiedene Inhalte existieren. Das geschilderte Verhalten gilt rekursiv für alle alle Stufen bei abnehmender Zahl von Bits. Dies hat zur Folge, daß der Prozeß der Trennung von Paaren I(s), J(s) stets *eindeutig und vollständig* ablaufen kann.

Zusammenfassend kann man sagen, daß Inhalte betrachtet werden, denen *von rechts nachs links* in

jeder Stufe ein niederwertiges Bit abgeschnitten wird. Die Inhalte mit gleichen übrigbleibenden Bits werden verschiedene Hälften verteilt.

3.5.3 Implementierung der Trennung mit Hilfe von Kreuzschaltern und Unshuffle-Topologie

Die Implementierung der Trennung kann mit Hilfe derselben Schaltung, wie für die sukzessiven Approximation erreicht werden:

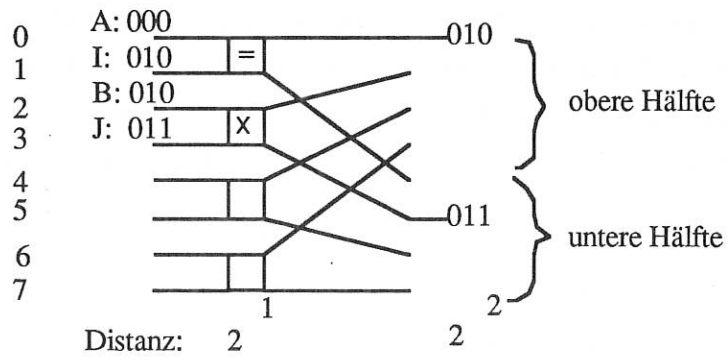


Bild 3.16:
Die Implementierung der Trennung zweier Paare I(s), J(s) auf die Distanz 2^s kann wie bei der sukzessiven Approximation erfolgen. Bsp. n = 3, s = 2, I(s) = 010, J(s) = 011, A(s) = 000, B(s) = 010.

3.5.4 Die Topologie des linken Teils des Netzes

Die rekursive Anwendung der angegebenen Implementierung der Trennung für die Distanz 2^s auf alle nächstkleinere Distanzen führt auf die folgende Topologie für NL:

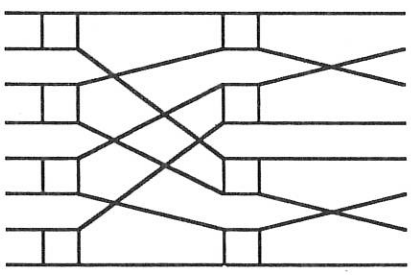


Bild 3.17:
Topologie des linken Teils des Netzes für N = 8 Eingänge.

Damit sind NL und NR bis auf die letzte Stufe von NL, die redundant ist und deshalb weggelassen wurde, gleich. Dieser Punkt und die Tatsache, daß aufgrund der Selbstähnlichkeit der Topologie

des modularen Koppelnetzes die Moduln NL und NR eines bestehendes Netz bei Verdopplung der Zahl der Eingänge weiterverwendet werden können, sind die beiden technischen Vorteile des modularen Koppelnetzes.Es bleibt jetzt noch zu zeigen, das die Schalter von NL ihrerseits stets blockierungsfrei gesetzt werden können.

3.5.5 Die Blockierungsfreiheit des linken Teils

Die Situation ist die folgende: Zur Blockierungsfreiheit von NR wurde diesem Netz ein zweites, NL, vorgesschaltet. Deshalb bleibt zu zeigen, daß NL ohne ein vorgeschaltetes Netz blockierungsfrei arbeitet. Daß dieser Beweis geführt werden kann, hängt u.a. damit zusammen, daß es für die Trennung von I(s) und J(s) unerheblich ist, welcher Inhalt in welche Hälfte kommt. Weiterhin kann ein Algorithmus zur Einstellung der Schalter von NL angegeben werden, bei dem es, im Gegensatz zu NR, eine überschaubare Zahl von Möglichkeiten für die Schalterstellungen gibt. Der Beweis für die Blockierungsfreiheit von NL wird so geführt, daß für jeden Fall einzeln gezeigt wird, daß ein Trennen von I(s) und J(s) möglich ist.

Der Beweis für die Blockierungsfreiheit von NL

Die Schalter des Netzes NL können verschiedene Zustände haben, nämlich den Zustand "gesetzt" oder "nicht gesetzt". Die an den Schaltern anliegenden Inhalte können ebenfalls die Zustände "verarbeitet" oder "noch nicht verarbeitet" einnehmen. Ein gesetzter Schalter und ein verarbeiteter Inhalt werden zur genauen Buchführung markiert. In dieser Art der Buchführung ist ein unmarkierter Schalter noch nicht gesetzt, und ein unmarkierter Inhalt noch nicht verarbeitet. Zum Beweis der Blockierungsfreiheit müssen weiterhin sechs verschiedene Arten von Inhalten berücksichtigt werden:

I(s), J(s) seien Komplementärpartner der s. Stufe und müssen voneinander getrennt, d.h. in verschieden Hälften ihres Adressbereichs verteilt werden.

I´(s), I´´(s) seien die Inhalte an den zweiten Eingängen der Kreuzschalter von I(s) und J(s).

J´(s), J´´(s) seien die Komplementärpartner zu I´(s) bzw. I´´(s).

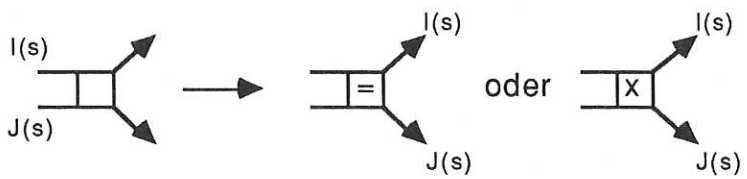
Der Beweis wird so geführt, daß *für jeden möglichen Fall* gezeigt wird, daß die Komplementärpartner mit Hilfe der Kreuzschalter *konfliktfrei* getrennt werden können. Dazu sind drei Schritte auszuführen. Beim ersten Schritt wird ein Komplementärpaar I(s), J(s) gesucht und die dazu gehörenden Schalter so gesetzt, daß I(s) von J(s) getrennt wird. I(s) und J(s) sowie ihre Schalter werden markiert. Im zweiten Schritt werden die Komplementärpartner J´(s) bzw. J´´(s) der zweiten Inhalte I´(s) bzw. I´´(s) von diesen getrennt. I´(s) und I´´(s) bzw. J´(s) und J´´(s) sowie

ihre Schalter werden markiert. Beim dritten Schritt wird der Vorgang eins und zwei solange wiederhohlt, bis alle Schalter und Inhalte in allen Stufen markiert sind.

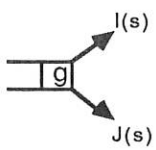
Am Anfang dieses Verfahrens sind alle Schalter und alle Inhalte unmarkiert. Dann wird ein beliebiger Inhalt I(s) der Stufe s = n-1 ausgewählt und sein Komplementärpartner J(s) gesucht. Für die Lage des Paares I(s) und J(s) zueinander gibt es nur zwei Möglichkeiten: sie liegen an *einem* Schalter an oder nicht.

1. I(s) und J(s) sind am selben Kreuzschalter

In diesem Fall kann entweder I(s) am oberen Eingang und J(s) am unteren Eingang des Schalters sein oder umgekehrt. Aus Symmetriegründen braucht nur eine Variante betrachtet werden, z.B. die, daß I(s) am oberen Eingang anliegt. Da per definitionem I(s) und J(s) Komplementärpartner der Stufe s sind, müssen sie voneinander getrennt werden. Dazu gibt es zwei mögliche Schalterstellungen, die zusammen mit der Ausgangssituation im nächsten Bild dargestellt sind:



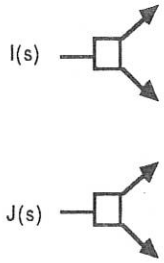
Zunächst liegt die Stellung des Kreuzschalters nicht fest. Für beide Schalterstellungen werden I(s) und J(s) in verschiedene Hälften transportiert. Der Schalter wird z.B. so gesetzt, daß I(s) nach oben transportiert wird. *Es ist also möglich, die Komplementärpartner voneinander zu trennen.* Der gesetzte Kreuzschalter wird mit einem "g" markiert, wie dies im nächsten Bild dargestellt ist:



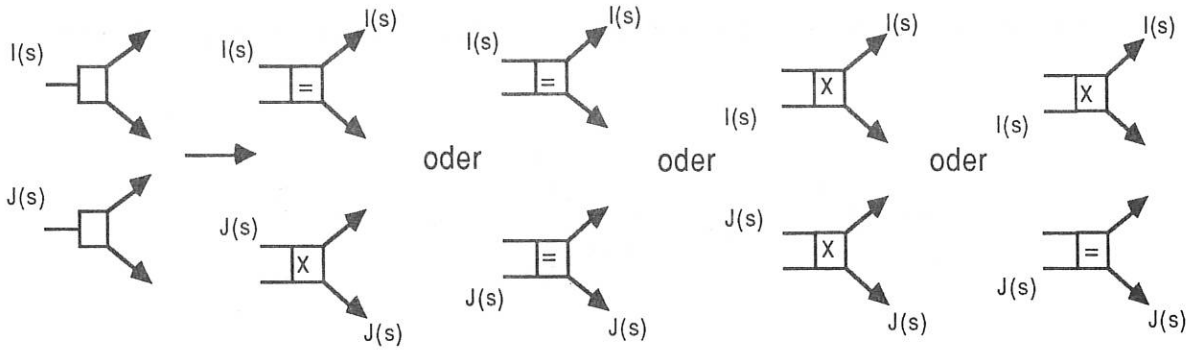
Nach dieser Trennung wird Schritt eins für einen neuen Inhalt I(s) wiederhohlt.

2. I(s) und J(s) sind nicht am selben Schalter

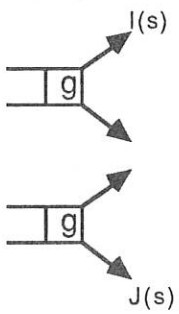
Wiederum kann I(s) und J(s) am oberen oder unteren Eingang seines Kreuzschalters anliegen. Diese Ausgangssituation wird graphisch so dargestellt, daß ein Kreuzschalter mit nur einem Eingang gezeichnet wird, der aber als oberer *oder* unterer Eingang des Schalters zu interpretieren ist.



Zunächst sind die Schalter von I(s) und J(s) nicht gesetzt. Wenn beide Inhalte entweder nur am geraden oder ungeraden Eingang ihrer Schalter anliegen, müssen die Schalter gegensinnig gesetzt werden. Wenn die Inhalte an verschiedenen Eingängen (z.B. I(s) am geraden, J(s) am *un*geraden Eingang) ihrer Schalter anliegen, müssen die Schalter gleichsinnig gesetzt werden. Aus den zwei möglichen Schalterstellungen "gleichsinnig" bzw. "gegensinnig" ergeben sich mit zwei zu setzenden Schaltern vier mögliche Kombinationen, I(s) und J(s) von einander zu trennen:



Da beide Schalter noch nicht gesetzt sind, kann ihre Stellung so festgelegt werden, daß *die Inhalte getrennt sind.* Man kann sie z.B.so setzen, daß I(s) nach oben und J(s) nach unten wandert:



Als zweiter Schritt müssen jetzt die zu I´(s) und I´´(s) komplementären Inhalte J´(s), bzw. J´´(s) gesucht werden. Würde man dies nicht tun, würde sehr schnell die Situation entstehen, daß mehr als zwei gesetzte Schalter existieren, an denen ein noch nicht verarbeiteter Inhalt anliegt, der aber, *anders als die bereits gesetzten Schalter es erlauben,* transportiert werden muß. Um diese Konfliktsituation zu vermeiden, darf höchstens je ein unmarkierter Inhalt an zwei markierten Schaltern anliegen. Die Strategie des Schalter-Setzens liegt also darin, maximal zweimal einen unmarkierten Inhalt an einem markierten Schalter zuzulassen. Wenn die Komplementär-Partner dieser beide Inhalte an anderen als den beiden markierten Schaltern liegen, müssen diese Schalter unmarkiert sein. Über deren Schalterstellung kann dann verfügt werden, und die Komplementär-

partner werden somit von den ummarkierten Inhalten getrennt.

Aus diesem Grunde werden im 2. Schritt die J´(s) und J´´(s) zu I´(s) bzw. I´´(s) gesucht. Dabei können zwei Fälle auftreten:

2.1 I´(s) und I´´(s) sind bereits komplementär

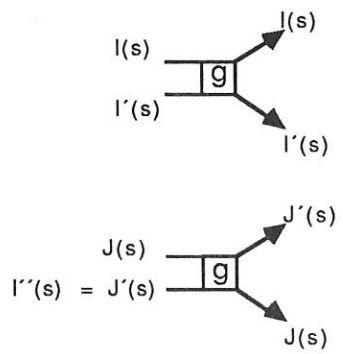
Da die Schalter von I(s) und J(s) im ersten Schritt so gesetzt wurden, daß I(s) und J(s) getrennt werden, werden im zweiten Schritt auch I´(s) und I´´(s) automatisch getrennt. Der Beweis dafür ist einfach: Das Ziel von I´(s) ist die obere *oder* untere Hälfte des Adressbereichs der nachfolgenden Stufe. Aber es ist aufgrund der Unshuffle-Verdrahtung zwischen den Stufen sicher, daß das *Ziel von I´(s) komplementär zu dem von I(s)* ist. Dieselbe Überlegung gilt für I´´(s) und J(s). Wenn aber auch *I(s) und J(s) komplementäre Ziele* haben, was nach Schritt eins sichergestellt ist, sind

dadurch die Ziele von I´(s) und I´´(s) komplementär. Der Grund dafür ist die dreifache Negation, die sich formal mit Hilfe des Operators "<->" darstellen läßt, der heißen soll: "haben komplementäre Ziele".

Wenn I´ <-> I und I <->J und J <-> I´´ ist, dann ist I´<-> I´´.

q.e.d.

Eine der möglichen Kombinationen, die diesen Sachverhalt exemplarisch widerspiegelt, ist im nächsten Bild dargestellt, wobei angedeutet ist, daß I´´(s) = I´(s) ist.



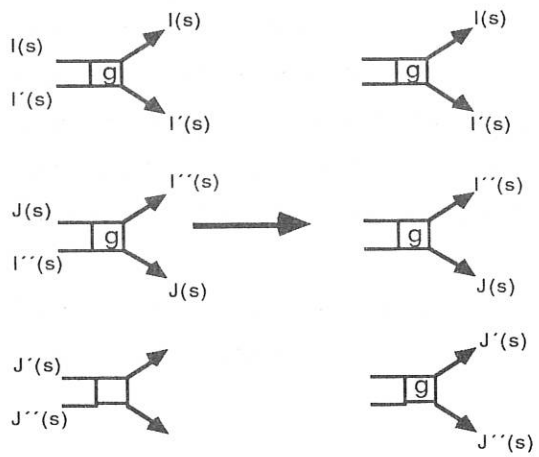
Nach dieser Trennung wird Schritt eins für ein neues I(s) wiederhohlt.

2.2 I´(s) und I´´(s) sind nicht komplementär

In diesem Fall existieren zwei weitere Inhalte J´(s) und J´´(s) und es müssen zwei Unterfälle unterschieden werden: J´(s) und J´´(s) liegen am selben (unmarkierten) Schalter oder nicht.

2.2.1 J´(s) und J´´(s) liegen am selben Schalter

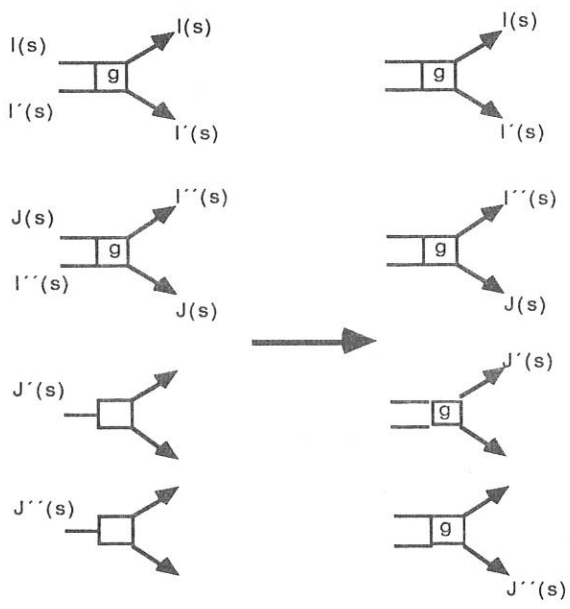
Der Schalter, an dem J´(s) und J´´(s) anliegen, ist nach Voraussetzung unmarkiert. Es kann also über seine Stellung verfügt werden. Für diese gibt es, da I´(s) und I´´(s) bereits in verschiedene Hälften weisen, genau eine Position, so daß J´(s) und I´(s) bzw. J´´(s) und I´(´s) voneinander getrennt werden. Die einzustellende Position des bislang unmarkierten Schalters ist durch die Ziele von I´(s) und I´´(s) eindeutig bestimmt. Wiederum können die Komplementärpartner voneinander getrennt werden. Dieser Trennvorgang ist graphisch dargestellt:



Nach dieser Trennung wird Schritt eins für ein neues I(s) wiederhohlt.

2.2.2 J´(s) und J´´(s) liegen an verschiedenen Schaltern

Im diesem Fall gibt *zwei* unmarkierte Schalter. Sie können so gesetzt werden, daß die Komplementärpartner in verschiedenen Hälften zu liegen kommen. Die Position der unmarkierten Schalter wird eindeutig durch das Ziel von I´(s) und I´´(s) festgelegt. Wiederum können die Komplementärpartner getrennt werden. Dieser Fall ist im nächsten Bild veranschaulicht:



Vor dem dritten Schritt besteht die folgende Situation: Die beiden oberen Schalter der letzten beiden Bilder sind markiert, sowie alle an ihnen anliegenden Inhalte. Die unteren Schalterpaare in diesen Bildern sind ebenfalls markiert, haben aber je einen markierten und unmarkierten Inhalt anliegen. Letztere sind als neue I´(s) und I´´(s) zu betrachten. Für diese neuen I´(s) bzw. I´´(s) sind die Fälle 2.1 oder 2.2 gültig, so daß ihr Fall auf einen bereits vorhandenen Fall zurückgeführt ist. Wenn alle Eingänge von markierten Schaltern ebenfalls markiert sind, wird ein neuer unmarkierter Inhalt an einem unmarkierten Schalter in der Stufe s gesucht . Danach wird Schritt eins und zwei wiederhohlt, solange, bis alle Paare I(s), J(s) getrennt sind. Dies ist aufgrund der gezeigten Strategie immer möglich. Wenn keine unmarkierten Inhalte mehr existieren, terminiert der Algorithmus für die Stufe s.
Damit sind alle Möglichkeiten, die bei dem geschilderten Verfahren der Paartrennung in einer Stufe von NL auftreten können, erfaßt und für jede einzeln gezeigt, daß eine Trennung mit Hilfe von NL konfliktfrei möglich ist.
Dieses Ergebnis kann auf alle Stufen von NL übertragen werden, mit dem Zusatz, daß man sich auf die jeweils gültigen Adressräume (Hälften, Vierteln, Achteln usw.) innerhalb einer Stufe bezieht. Somit ist die blockierungsfreie Anwendung des Verfahrens der Paartrennung auf dem ganzen Netz NL gezeigt.

3.6 Zusammenfassung

Es wurde ein Koppelnetz vorgestellt, das bei N Eingängen ($N = 2^n$) alle N! Permutationen von Verbindungen zwischen Ein- und Ausgängen herstellen kann. Es bedient alle Eingänge gleichzeitig und verbindet sie mit Ausgängen in Form von Punkt-zu-Punkt-Verbindungen. Es eignet sich daher besonders z.B. für Verbindungen zwischen Prozessoren, die alle gleichzeitig arbeiten, oder für Verbindungen zwischen Prozessoren und Speichern. Es besteht, wie das Benes Netz, aus (N/2)* (2logN - 1) Schaltern. Der technische Vorteil dieses Netzes gegenüber dem Benes-Netz liegt darin, daß es aus *zwei gleichen Moduln* besteht. (Die letzte Stufe des ersten Moduls ist redundant und kann weggelassen werden). Im Vergleich zum Lee-Netz sind die *Verdrahtungen zwischen den Stufen wesentlich lokaler*, da die Vermaschung von Stufe zu Stufe abnimmt. Dies bedeutet Kostenersparnis für ein Netz, das auf mehreren Platinen realisiert ist, da weniger Steckverbinder notwendig sind:

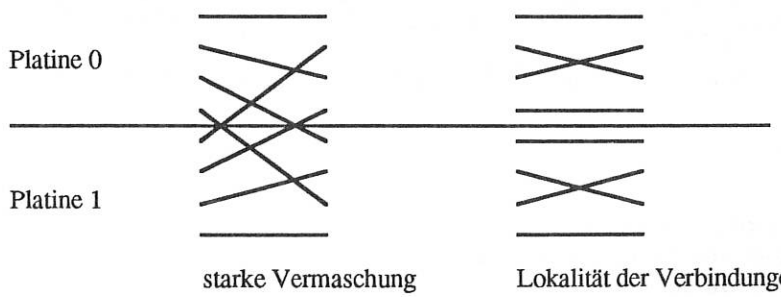


Bild 3.18:
Ein Vorteil des modularen Koppelnetzes gegenüber dem Lee Netz ist die geringere Vermaschung der Verdrahtung zwischen den Stufen. Dadurch können Kosten für Steckverbinder etc. eingespart werden.

Weiterhin besteht, wie beim Benes Netz, eine *Selbstähnlichkeit der Topologie*. Dies bedeutet, daß die Teile NL und NR eines bestehenden Netzes in einem Netz mit der doppelten Zahl von Eingängen weiter verwendet werden können.

3.7 Die modulare Erweiterung des Netzes am Beispiel von 4, 8 und 16 Eingängen

Das Netz wird als Beispiel für seine Modularität exemplarisch von vier Eingängen über acht auf 16 Eingänge erweitert :

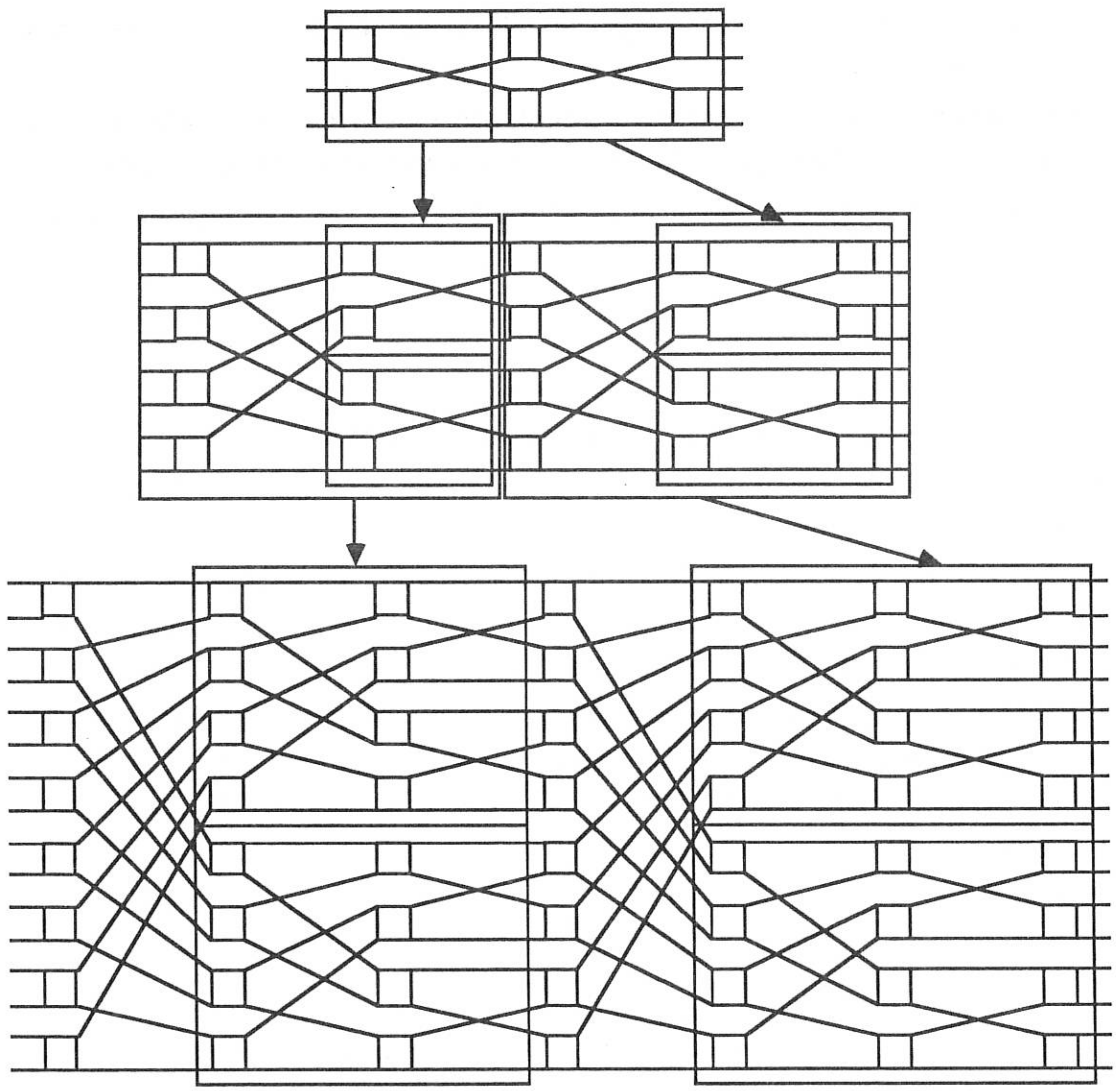


Bild 3.19:
Topologie des modularen Koppelnetzes für 4, 8 und 16 Eingänge. Aufgrund der Selbstähnlichkeit der Topologie können die beiden Teile eines Netzes für ein Netz mit doppelter Zahl von Eingängen weiterverwendet werden.

3.8 Die Algorithmen zur Einstellung der Schalter des Netzes

Der Algorithmus für NR

Die Stufen von NR werden von links nach rechts mit n-1 bis 0 numeriert. Für einen Schalter der Stufe s ist das Bit der Wertigkeit 2^s von einem der beiden Inhalte I(s) oder J(s), die an diesem Schalter anliegen, relevant. Es sei das Bit i_s von I(s) relevant. Dann ist der Algorithmus der folgende:

Wenn Bit i_s von I(s) gleich Null, dann Schalter parallel stellen, sonst Schalter auf Kreuzsstellung

(I(s) ist der am geraden Eingang des Schalters anliegende Inhalt)
Dieser Algorithmus führt zu einer eindeutigen Lösung für die Schalterstellungen von NR.

Beispiel für ein gesetztes Netz NR:

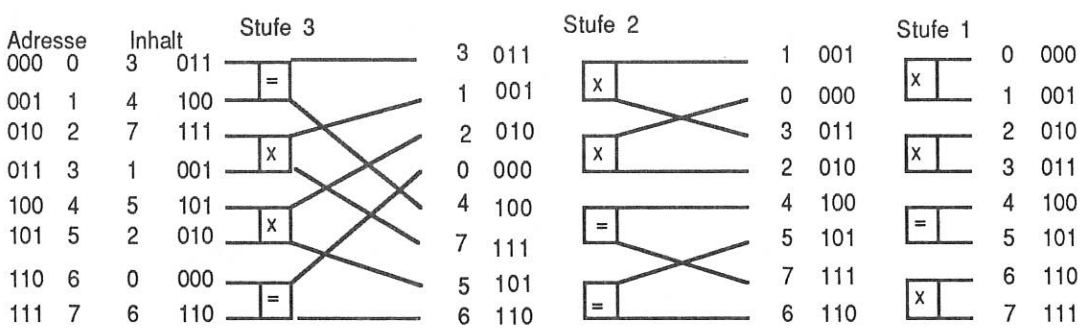


Bild 3.20:
Die Schalterstellungen von NR für das Beispiel (3,4,7,1,5,2,0,6). Am Ausgang von NR ist der Permutationsvektor sortiert.

Der Algorithmus für NL

Wie bei NR werden die Stufen s von NL von links nach rechts mit n-1 bis 1 numeriert. Die Bezeichnungen I(s), J(s) sollen zum Ausdruck bringen, daß von den Inhalten I, J nur die Bits i_{n-1} bis $i_{n-(s+1)}$ relevant sind. Beim Fortschreiten von Stufe zu Stufe werden daher immer mehr Bits rechts von $i_{n-(s+1)}$ abgeschnitten, da die Stufennummer s abnimmt .

Bezeichnungen: $I(s) = i_{n-1},\dots,i_{n-s}i_{n-(s+1)}i_{n-(s+2)},\dots,i_0$
$J(s) = i_{n-1},\dots,i_{n-s}/i_{n-(s+1)}j_{n-(s+2)},\dots,j_0$
J(s) wird auch als Komplementärpartner zu I(s) bezeichnet.
I´(s), I´´(s) sind die zweiten Inhalte an den zweiten Eingängen der Kreuzschalter von I(s) und J(s).
J´(s), J´´(s) sind die Komplementärpartner zu I´(s) und I´´(s).

Zustände: Da die Schalter nicht sequentiell nacheinander gesetzt werden, muß über den jeweiligen Zustand der Schalter Buch geführt werden: Ein Schalter, der noch nicht gesetzt wurde, ist markiert. Ebenso wird ein Inhalt, der noch nicht verarbeitet wurde, markiert. Ein markierter Inhalt ist bereits in seine Hälfte verteilt und braucht im Algorithmus nicht mehr berücksichtigt zu werden.

Bemerkungen: 1. Der Algorithmus ist bezüglich seiner Schalterstellungen nicht eindeutig, da es irrelevant ist, welcher Inhalt in welche Hälfte kommt.

2. Der Algorithmus muß von links nach rechts für alle Stufen von NL durchgeführt werden. Für die erste Stufe von links ist s = n-1, und es werden 2^{n-1} Paare I(s), J(s) unter 2^n verschiedenen Inhalten gesucht und verteilt. Für die zweite Stufe ist s = n-2, und es werden in der oberen und unteren Hälfte 2^{n-2} Paare gesucht, unter je 2^{n-1} verschiedenen Inhalten. Für die dritte Stufe ist s = n-3, und es werden in jedem Viertel 2^{n-3} Paare gesucht und verteilt u.s.w..

3. Die mit einem Kreis gekennzeichneten Zahlen geben im Beweis der Blockierungsfreiheit den jeweiligen untersuchten Fall an. Das folgende Bild 3.21 zeigt den Algorithmus zur Einstellung von NL in Form eines Flußdiagramms.

4. Die Blöcke "Betrachte I_S' " bzw. Betrachte I_S'' " in Bild 3.21 sind so zu verstehehen, daß die zweiten Inhalte der *soeben* gesetzten Schalter zu betrachten sind. Beim ersten Durchlaufen des Signalflußdiagramms sind dies diejenigen Schalter, an denen I_S und J_S anliegen. Bei allen weiteren Durchläufen können es auch die Schalter von J_S' und J_S'' sein. In diesem Fall wird der Teil "Parallel Ausführen" mehrfach durchlaufen. Dies drückt eine *rekursive Anwendung* dieser Prozedur auf die zweiten Inhalte, der durch diese Prozedur soeben gesetzen Schalter aus.

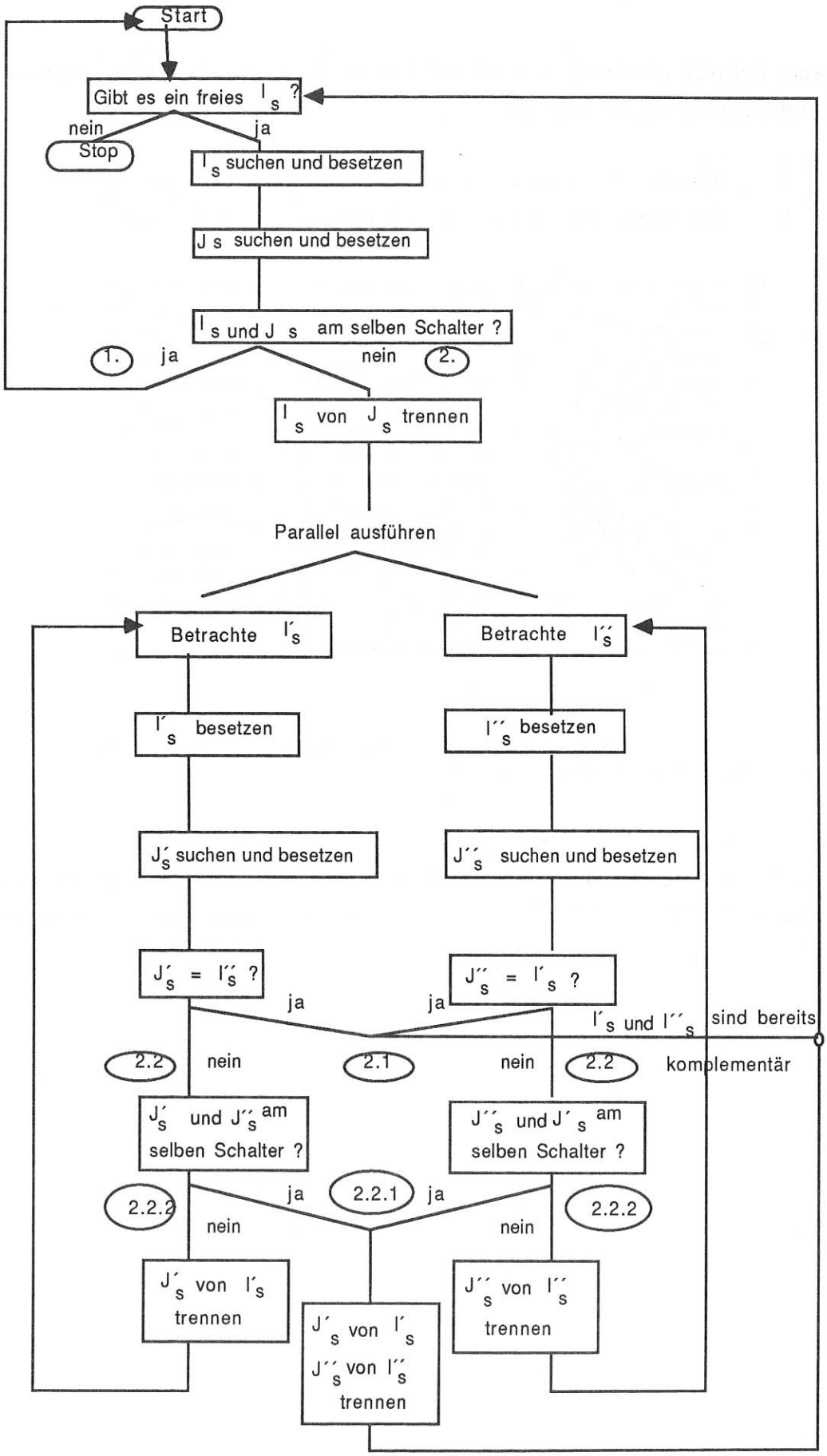


Bild 3.21:
Der Algorithmus zur Einstellung der Schalter des linken Teils des Netzes.

Das folgende Beispiel soll eine der möglichen Schalterstellungen von NL zeigen. Dabei wurden *zusätzlich* folgende Konvention eingehalten:

1. Absuchen der Schalter nach Inhalten in einer Stufe von oben nach unten.
2. Voreinstellung für einen zu setzenden Schalter ist paralleler Durchgang

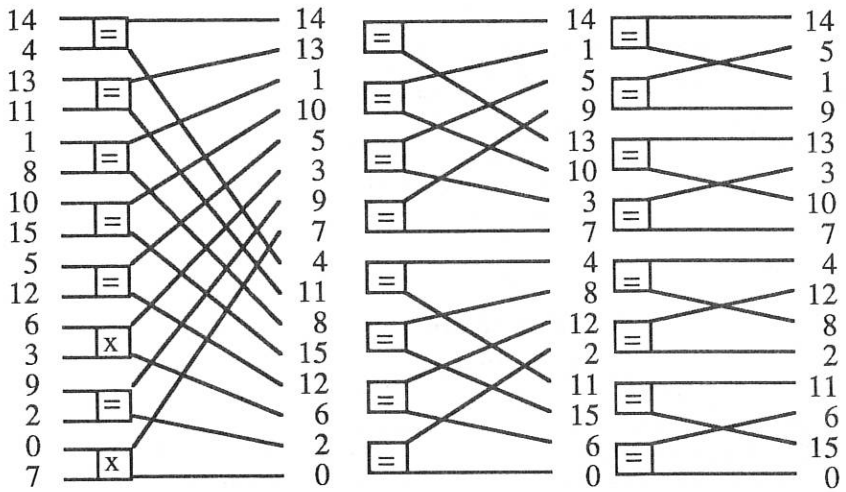


Bild 3.22:
Beispiel der Schalterstellungen von NL bei 16 Eingängen für den Permutationsvektor (14,4,13,11,1,8,10,15,5,12,6,3,9,2,0,7).

Damit ist die Darstellung über das modulare Koppelnetz beendet. Im nächsten Kapitel werden zwei Anwendungen von MULTITOP diskutiert: die schnelle Fourier-Transformation und die Spline-Approximation.

4. Die Parallelisierung der schnellen Fourier-Transformation

In diesem Kapitel wird davon ausgegangen, daß die Grundlagen über die Fourier-Transformation und ihre numerisch effiziente Ausführung, die schnelle Fourier-Transformation (FFT), bekannt sind. Es soll hier nur auf den Anhang wie auf [BRIG], [GOLD] und [SWAR] verwiesen werden.

Zur Parallelisierung der FFT muß eine Beschreibung in Termen von Prozessen gefunden werden, die dann Prozessoren zugeordnet werden können. Eine Modifikation der Signalflußgraphen-Schreibweise, die es erlaubt, die Operationen der FFT in einzelnen Blöcken zu konzentrieren, ist die Funktionsblockschreibweise. Die Blöcke werden als "Butterflies" bezeichnet.

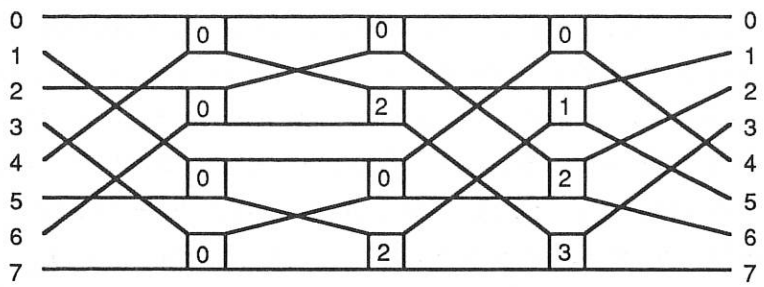
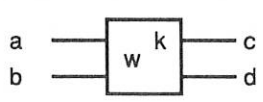


Bild 4.1:
Funktionsblockschreibweise der schnellen Fourier-Transformation. Links werden die Abtastwerte angelegt, rechts erscheint ihr Spektrum. Die Ziffern in den Blöcken geben die Exponenten des Drehfaktors w^k an.

4.1 Granularität des Problems

Bei der Parallelisierung ist die Frage nach der kleinstmöglichen Verfeinerung des Problems für parallele Verarbeitung, der sogenannten Granularität, zu stellen. Bei der FFT bietet sich der Butterfly als elementarer Prozeß an, der parallel zu anderen Butterflies ausgeführt wird. Ein Butterfly besteht aus den folgenden Operationen mit komplexen Operanden:



$$c = a + bw^k$$

$$d = a - bw^k$$

Bild 4.2:
Der Butterfly Prozeß der Cooley-Tukey FFT.

In der Schreibweise mit Real- und Imaginärteil:

$$c = cr + j.ci = ar + br*wr - bi*wi + j.(ai + br*wi + bi*wr)$$
$$d = dr + j.di = ar - (br*wr - bi*wi) + j.(ai - (br*wi + bi*wr))$$

Ein Butterfly enthält 4 verschiedene skalare Multiplikationen und 6 verschiedene skalare Additionen.

Die Wahl des elementaren Prozesses ist von der Architektur des Rechners abhängig. Für einen Signalprozessor z.B wäre es sinnvoll, einen kleineren elementaren Prozeß als einen Butterfly zu wählen, da ein Signalprozessor aufgrund seiner Architektur wie getrennter Programm- und Datenspeicher (= Harvard Architektur), getrenntes Addier- und Multiplizierwerk, intern parallel arbeiten kann. Damit ist in einem Signalprozessor eine überlappende Bearbeitung von Multiplikation, Akkumulation und evtl. Indexberechnung, sowie Daten-Ein/Ausgabe möglich. Für einen konventionellen von Neumann-Rechner (sequentielle Architektur) dagegen ist ein Butterfly das kleinste Teil, das parallel mit anderen Butterflies auf anderen von Neumann Rechnern berechnet werden kann. Wie viele Butterflies sinnvollerweise auf jeden von Neumann Rechner innerhalb eines Multiprozessorsystems entfallen sollten, hängt neben dem Preis für Prozessoren, von dem Verhältnis zwischen Datenberechnung und Datenaustausch ab. Wenn die Kommunikation zwischen zwei Prozessoren im Vergleich zur Berechnung eines Butterflies viel Zeit kostet, lohnt es sich nicht, nur wenige Butterflies in jedem Prozessor zu berechnen, da für jeden Butterfly zwei Eingabe-Daten benötigt und zwei Ausgabe-Daten produziert werden, und diese ausgetauscht werden müssen, was einer hohen Kommunikation entspricht. Im folgenden wird der allgemeine Fall von N Punkten bei P Prozessoren diskutiert.

4.2 Zuordnung von Butterflies zu Prozessoren

Es existieren drei verschiedene Möglichkeiten, die Butterflies Prozessoren zuzuordnen:

1. P = N

Jedem Butterfly wird ein Prozessor zugeordnet. Die Topologie ihrer Verbindungen ist die Topologie des Graphen der FFT in der Funktionsblockschreibweise. Die Prozessoren müssen zwei Eingabe, zwei Ausgabe und einen Zwei-Worte-Speicher für die Eingabewerte a,b enthalten, sowie eine Datenkommunikation in vergleichbarer Geschwindigkeit mit der Datenberechnung. Sie führen alle dasselbe Programm aus, nur unterschieden durch ihre Position in dem Graphen der FFT mit einem anderen Drehfaktor. Dies ist zwar die größtmögliche Parallelisierung der FFT, die aber i.a. aus Kostengründen nicht realisierbar sein wird.

2. P * k = $\log_2 N$, k = 1, 2, ...,$\log_2 N$ (k Stufen je Prozessor)

Die Butterflies werden spaltenweise Prozessoren zugeordnet. Es existieren maximal so viele Prozessoren, wie es Stufen gibt, bzw. für k > 1 werden die Operationen der einzelnen Stufen gleichmäßig auf die Prozessoren verteilt, so daß jeder Prozessor dasselbe Quantum an Arbeit erhält.

Wenn k die Zahl der Stufen der FFT mit Rest teilt, ergibt sich das Problem der Minderauslastung eines Prozessors, was die Effizienz reduziert.

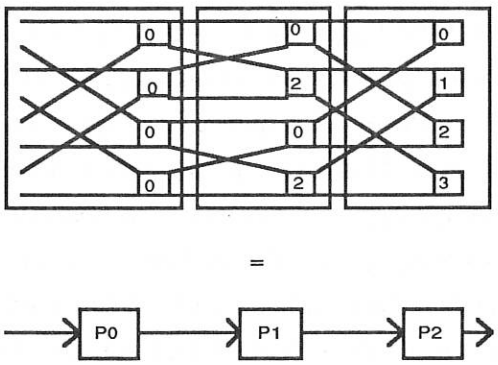


Bild 4.3:
Spaltenweise Parallelisierung der FFT für das Beispiel N = 8, P = logN = 3 und k = 1.

Diese Art der Aufteilung erlaubt Flexibilität bei der Zahl der Prozessoren, setzt aber einen kontinuierlichen Dateneingangsstrom voraus, da es sich um eine Fließbandverarbeitung (Pipelining) handelt, die bei kontinuierlichem Betrieb der Pipeline am Eingang mit jedem Takt am Ausgang ein Ergebnis produziert.

3. P * k = N/2, k = 1,2,...,N/2

Die Butterflies werden zeilenweise Prozessoren zugeordnet. Es existieren maximal so viele Prozessoren, wie es Zeilen in dem Graphen der FFT gibt, bzw. für k > 1, werden die Zeilen gleichmäßig auf die Prozessoren verteilt (Gleichmäßige Verteilung ist nur möglich, wenn N ohne Rest durch k teilbar ist).

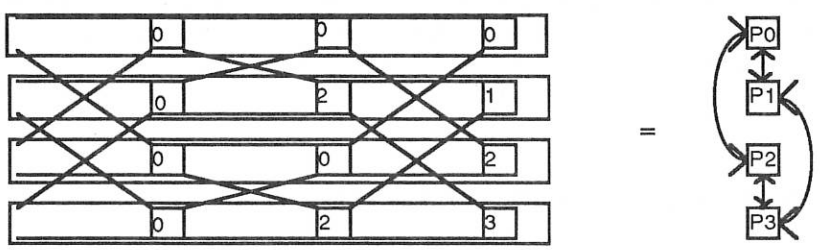


Bild 4.4:
Zeilenweise Zuordnung der Operationen FFT zu Prozessoren und deren Topologie der Verbindungen für das Beispiel N = 8, P = 4 und l = 1.

Die zeilenweise Parallelisierung ermöglicht nichtkontinuierlichen parallelen Betrieb der Prozessoren. Darüberhinaus ermöglicht sie eine wesentlich größere Freiheit bei der Wahl der Zahl der Prozessoren, da in praktischen Anwendungen $N/2 >> \log_2 N$ ist, und es deshalb mehr Unterteilungsmöglichkeiten gibt. Als Nachteil ist die kompliziertere Topologie der Verdrahtung der

Prozessoren untereinander anzusehen. Zudem unterliegt die Topologie der Verbindungen in der ersten und letzen Stufe der FFT einem anderen Bildungsgesetz als in den Stufen dazwischen. Bei der Implementierung dieser Form der parallelen FFT müssen die tatsächlichen Verbindungen zwischen den Prozessoren entweder alle drei Bildungsgesetze, für erste Stufe, den Mittelteil und die letzte Stufe, enthalten, oder die Topologie der Verbindungen muß variabel sein. Ein Nachteil ist, daß bei dieser Art der Parallelisierung die maximale Zahl von Links pro Prozessor sehr groß werden kann, wenn die Zahl der Prozessoren anwächst; Dies zeigt das folgende Beispiel von N = 16 Punkten und P = 8 Prozessoren. Dort ist die größte Zahl von Links pro Prozessor gleich sechs. Das Beispiel läßt sich leicht verallgemeinern. Reale Prozessoren sind in der Zahl ihrer Links begrenzt, der Transputer zum Beispiel auf vier Links. Dieser Nachteil läßt sich durch ein dynamisches Umschalten der Topologie aufheben, weil dadurch jeder Sender oder Empfänger von Daten *direkt* erreichbar wird. Darin liegt der Vorteil der variablen Topologie.

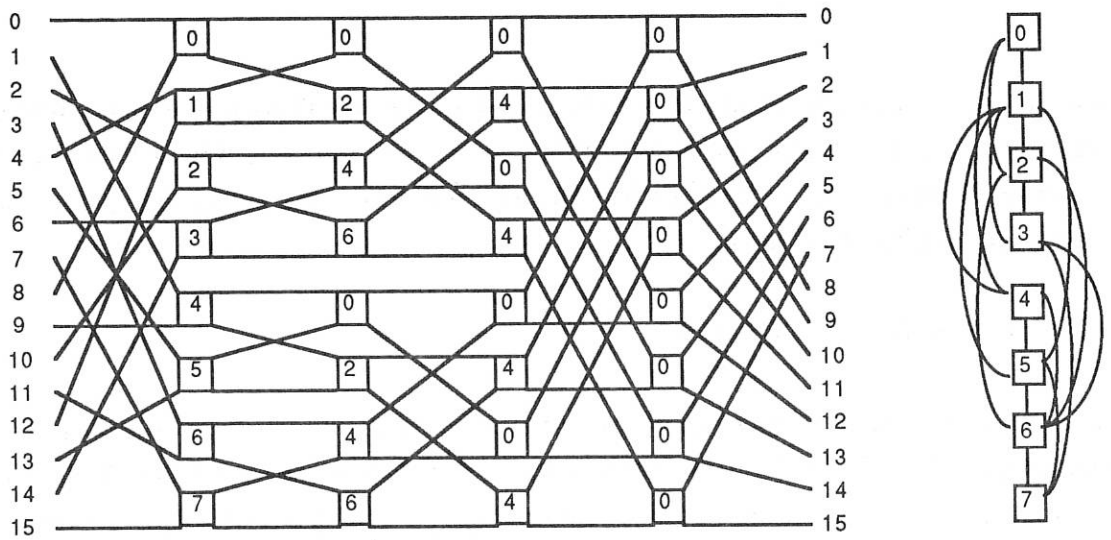


Bild 4.5:
Zeilenweise Parallelisierung der Cooley-Tukey-FFT am Beispiel N = 16, P = 8 ohne Pipelining. Die zeilenweise Parallelisierung gewährt die größtmögliche Flexibilität bei der Zahl der Prozessoren. Der Nachteil ist, daß die Topologie der Verbindungen von Stufe zu Stufe nicht konstant ist, und die maximale Zahl von Links pro Prozessor mit N wächst.

4.3 Beschleunigungserwartung bei Parallelisierung

Zusammenfassend ist in Tabelle 1 der zu erwartenden Zeitbedarf der parallelen FFT *unter Vernachlässsigung der Kommunikation* angeben. Aus den Zahlen könnte man schließen, daß die FFT sich gut parallelisieren läßt, da die Rechenzeiten linear mit der Zahl der Prozessoren skaliert werden können. Ob dieses angegebene Zeitverhalten in der Praxis tatsächlich erreicht wird, hängt von der Implementierung der FFT auf einem konkreten Multiprozessor ab. Insbesondere ist die Zeit des Datenaustauschs ja nicht berücksichtigt worden. Diese Vernachlässigung ist dann unzulässig,

wenn
1. der Datenaustausch in der selben Größenordnung wie die Datenberechnung liegt,
2. der Datenaustausch nicht voll überlappend mit der Berechnung implementiert wurde, oder nicht implementierbar ist, so daß seine Zeit mehr oder weniger additiv in die Berechnungsdauer eingeht.

Zahl der Prozessoren	Rechenzeit
1	$O(N\log_2 N)$
$\log_2 N$	O(N) (pipelined)
N/2	$O(\log_2 N)$
$(N/2)\log_2 N$	O(1) (pipelined)

Tabelle 1:
Vorhersage der Rechenzeiten in Abhängigkeit der Zahl der eingesetzten Prozessoren. Die Interprozessor-Kommunikation wurde dabei vernachlässigt.

Bei hoher Parallelisierung (P >> 1) ist bei nicht überlappender Kommunikation der *Datenaustausch der bestimmende Faktor des Wirkungsgrades*, und es lohnt sich, wie auch sonst in der parallelen Programmierung, eine genaue Datenflußanalyse zu machen, um die Effizienz im voraus abschätzen zu können, ohne den jeweiligen Algorithmus erst implementieren zu müssen. Im folgenden wird eine Datenflußanalyse für die verschiedenen kanonischen Formen der FFT (Gentleman-Sande FFT, Cooley-Tukey FFT, Stockham FFT und Pease FFT) gemacht. Dabei wird nur der für die Praxis interessanteste Fall der zeilenweisen Parallelisierung untersucht. Es zeigt sich, daß die verschiedenen Formen der FFT sequentiell zwar alle gleich (=kanonisch), parallel aber durchaus unterschiedlich bezüglich ihres Datenverkehrs zwischen den Prozessoren sind. Um die Datenflußanalyse machen zu können, werden zuerst die Signalflußgraphen der kanonischen Formen der FFT in Funktionsblockschreibweise angegeben. Danach werden die Permutationsfunktionen zwischen den einzelnen Butterfly-Stufen definiert, und damit die Gesamtzahl der zu transportierenden Daten für den Fall von N Punkten bei P Prozessoren berechnet.

4.4 Die Signalflußgraphen von Cooley-Tukey,Gentleman-Sande, Stockham und Pease FFT

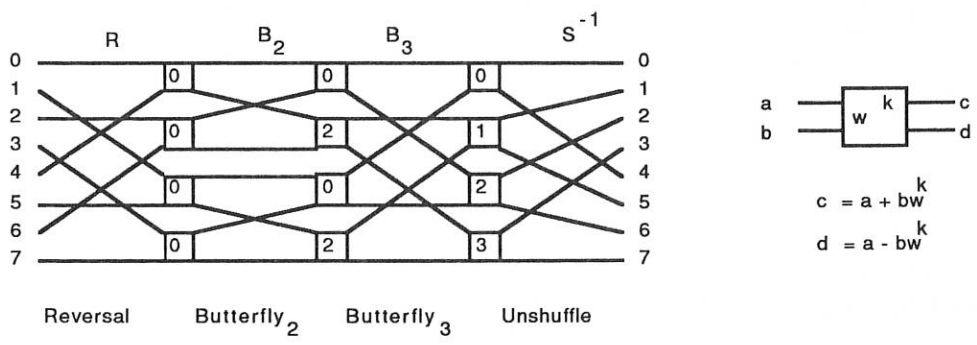


Bild 4.6:
Cooley-Tukey FFT für N = 8 Punkte.

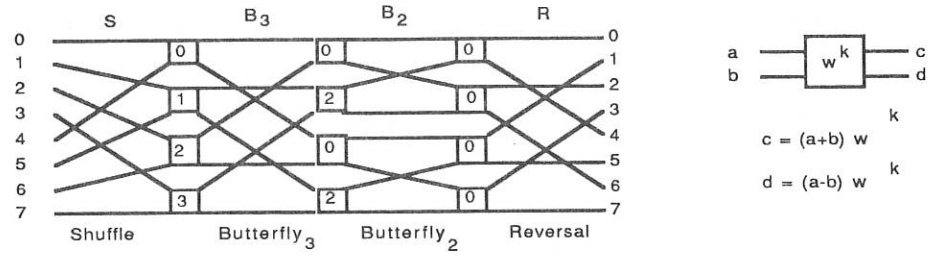


Bild 4.7:
Gentleman-Sande FFT für N = 8 Punkte.

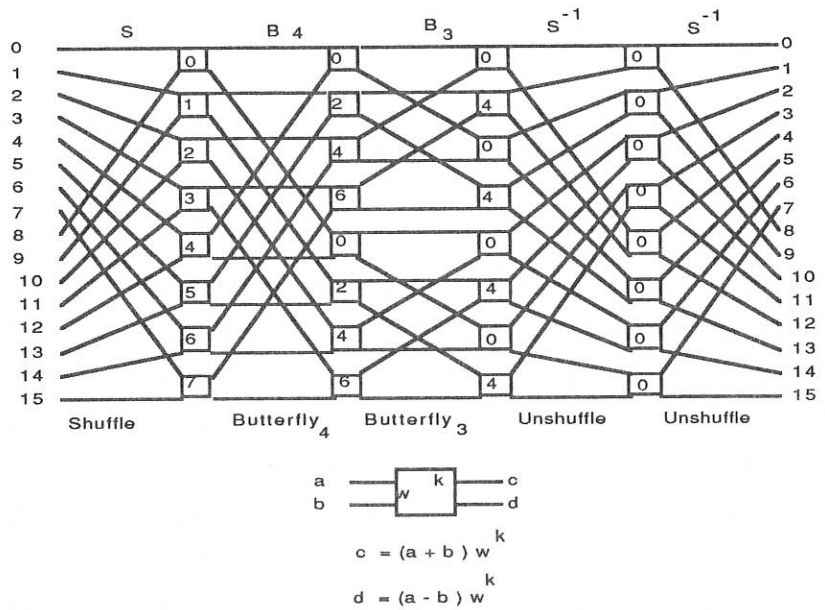


Bild 4.8:
Stockham FFT für N = 16 Punkte.

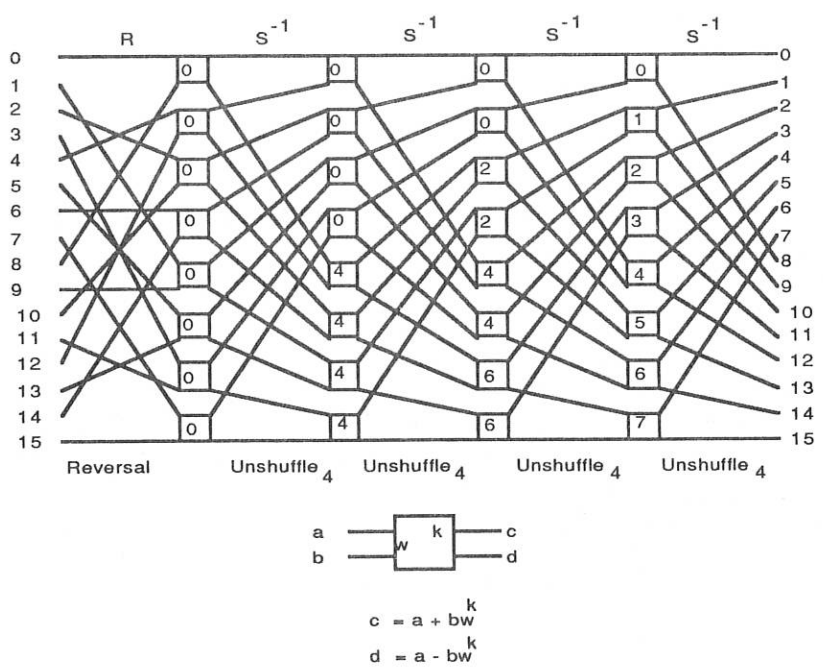


Bild 4.9:
Pease FFT für N = 16.

Die Signalflußgraphen zeigen:

1. Die Gentleman-Sande FFT erfordert bezüglich der Interprozessor-Kommunikation denselben Aufwand wie die Cooley-Tukey FFT, denn ihre Graphen gehen durch Spiegelung auseinander hervor.

2. Die Durchmischung von Daten bei der Gentleman-Sande FFT nimmt von Stufe zu Stufe ab, so daß von einer gewissen Stufe an, die abhängig von der Zahl der Prozessoren und Punkte ist, keine Daten mehr ausgetauscht werden müssen.

3. Die Durchmischung ist bei der Pease FFT offensichtlich am größten, so daß diese Form der FFT, die sequentiell den anderen zwar gleichwertig ist, sicher nicht die für Parallelverarbeitung beste FFT sein wird, da am meisten Daten ausgetauscht werden müssen.

Um eine exakte Auswertung der Zahl der zwischen den Prozessoren auszutauschenden Daten durchzuführen, ist die mathematische Definition der Permutationsfunktionen Shuffle, Butterfly und Reversal sowie die Zahl der dort auszutauschenden Daten notwendig. Die Datenflußanalyse von Shuffle, Butterfly und Reversal-Permutation wird im nächsten Abschnitt durchgeführt. Das Prinzip, nach dem die Datenflußanalyse gemacht wird, ist, daß von der Gesamtmenge an Daten diejenigen Daten subtrahiert werden, die nicht einer Interprozessor-Kommunikation unterliegen, so daß die zu transportierenden Daten indirekt ermittelt werden. Diese Methode hat sich am einfachsten erwiesen, da dazu nur die Fixpunkt-Menge der Permutations-Abbildungen bestimmt werden muß.

4.4.1 Die Datenflußanalyse von Shuffle, Butterfly und Reversal-Permutation

Shuffle-Permutation

Die Shuffle-Permutation hat die folgende Topologie:

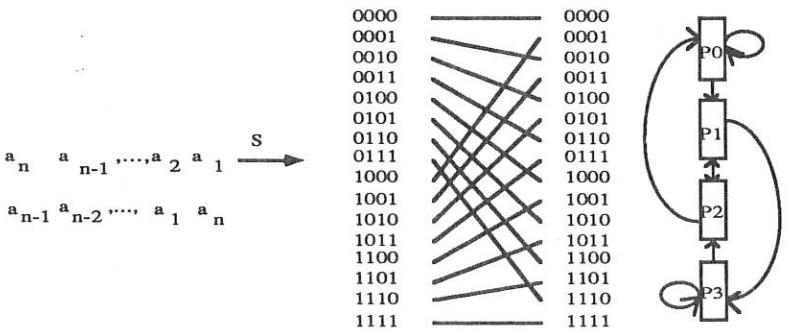


Bild 4.10:
Die Shuffle-Permutation S, die Topologie ihrer Prozessorverbindungen für N=16, P=4 und ihre Definition in Binärschreibweise. S kann als eine zyklische Rotation der Bits von A um eine Bitstelle nach links aufgefaßt werden.

Es gibt verschiedene Möglichkeiten zu klären, wieviele Daten bei der Shuffle-Permutation zwischen P Prozessoren ausgetauscht werden. Es zeigte sich, daß es am einfachsten ist, zu untersuchen, wieviele Daten nicht ausgetauscht werden müssen und diese von der Gesamtzahl N aller Daten abzuziehen. Im folgenden wird, wie bereits zuvor, von $N = 2^n$ ausgegangen. Nach der Definition von S gilt:

$$A = a_n a_{n-1},\ldots,a_1 \quad \xrightarrow{S} \quad A' = a_{n-1} a_{n-2},\ldots,a_1 a_n$$

Es gibt dann keinen Datenaustausch, wenn A und A' im selben Prozessor sind. Bei $P = 2^p$ Prozessoren spezifieren p Bits eindeutig einen Prozessor aus der Menge von P Stück. A und A' sind dann im selben Prozessor, wenn die ersten p Bits von A und A' identisch sind, d.h. wenn

$$\begin{aligned} a_n &= a_{n-1} \\ a_{n-1} &= a_{n-2} \\ &\ldots \\ a_{n-(p-1)} &= a_{n-p} \end{aligned}$$

ist, bzw.:

$$a_n = a_{n-1} = a_{n-2} = \ldots = a_{n-p}$$

gilt. Dies ist eine Bedingung, die p+1 Bits miteinander verknüpft. Wählt man den Wert eines dieser Bits, liegen alle anderen p Bits damit fest. In der Adresse A sind n Addressbits vorhanden, so daß insgesamt noch n - p Bits frei wählbar sind. Daraus folgt, daß es 2^{n-p} verschiedene Adressen A gibt, die invariant gegenüber der Abbildung S sind. Bei 2^n Daten insgesamt und 2^{n-p} invarianten Daten müssen demnach $2^n - 2^{n-p} = 2^n(1 - 2^{-p})$ Daten ausgetauscht werden. Die Zahl E der auszutauschenden Daten bei N Punkten und P Prozessoren bei der Shuffle Permutation ist damit:

$$E_S = 2^n(1 - 2^{-p})$$

Butterfly-Permutation

Die Butterfly-Permutation hat die folgende Topologie:

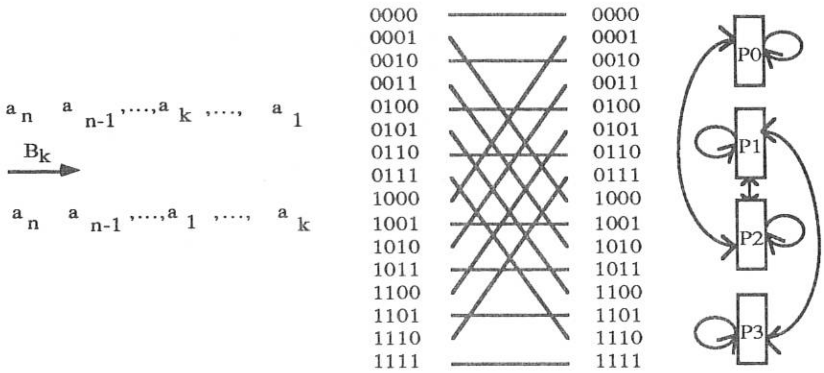


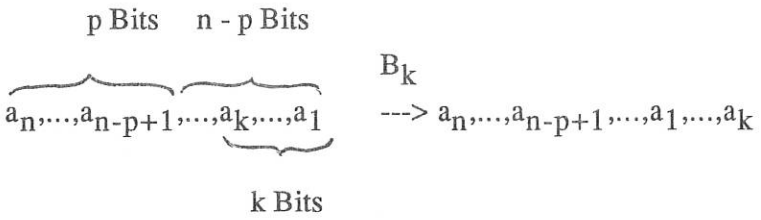
Bild 4.11:
Die Butterfly Permutation B_k, die Topologie ihrer Prozessorverbindungen für N=16, P=4, und ihre Definition in Binärschreibweise. B_k entspricht einem Vertauschen des k. mit dem 1. Bit.

Zur Klärung der Frage, wieviele Daten bei der Butterfly Permutation zwischen den Prozessoren ausgetauscht werden müssen, gehen wir wie bei der Shuffle Permutation vor, indem wir untersuchen, wieviele Daten nicht ausgetauscht werden müssen, und diese werden dann von der Gesamtzahl der Daten abgezogen.
A' ist dann im selben Prozessor, wenn die p höchstwertigen Bits von A' identisch mit denen von A sind.

Es müssen 2 Fälle unterschieden werden:

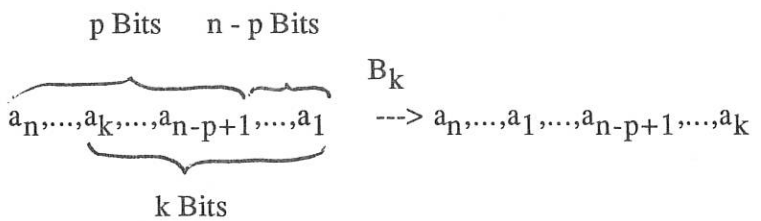
1. $k \leq n - p$

Das heißt, daß das k-te Bit rechts der p höchstwertigen Bits liegt und somit B_k keine externe Kommunikation bewirkt.



2. $k > n - p$

In diesem Fall liegt das k-te Bit innerhalb der höchstwertigen p Bits von A. Ein Austauschen von a_k mit a_1 bewirkt daher eine Adressänderung der Prozessoradresse und somit Interprozessor Kommunikation.



Die p linken Bits der Prozessoradresse von A sind invariant gegenüber B_k, wenn gilt:

$$a_k = a_1$$

D.h. für einen ganz bestimmten Prozessor (= $a_n,\ldots,a_{n-p-1}$ fest !), ist auch a_1 festgelegt auf $a_1 = a_k$. Damit bleiben von den n Bits von A noch n - p - 1 Bits zur freien Verfügung, so daß es innerhalb dieses Prozessors 2^{n-p-1} Daten gibt, die invariant gegenüber B_k sind. Bei 2^p Prozessoren sind demnach $2^p.2^{n-p-1}$ Daten invariant. Die Zahl der Daten, die durch B_k bewegt werden, ist insgesamt $2^n - 2^{n-1} = 2^{n-1}$.

$$E_{Bk} = \begin{cases} 0 & \text{für } k \leq n\text{-}p \\ 2^{n-1} & \text{für } k > n\text{-}p \end{cases}$$

Bemerkung:

Die Butterfly-Permutation hat die interessante Eigenschaft, daß sie unabhängig von der Prozessorzahl und unabhängig davon, welches Bit a_k ausgetauscht wurde, stets denselben Verkehr von der halben Zahl der Punkte aufweist (für $k > n\text{-}p$).

Reversal-Permutation

Die Reversal-Permutation hat die folgende Topologie:

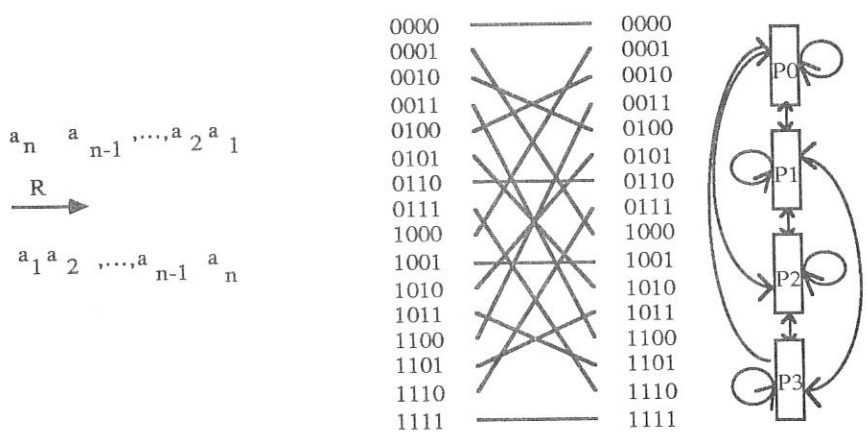


Bild 4.12:
Die Reversal-Permutation R, die Topologie ihrer Prozessorverbindungen und ihre Definition in Binärschreibweise. R bewirkt ein Umkehren der Bit-Reihenfolge.

Das Umkehren der Reihenfolge der Bits kann man auch durch Punktspiegelung der Bits an der Mitte des Bitstrings von A darstellen. Wenn man dies tut, erkennt man, daß für $p < n/2$ alle höchstwertigen p Prozessorbits durch 'neue' Bits rechts von der Mitte ausgetauscht werden. Dies ist nicht der Fall für $p > n/2$, da dann Prozessorbits rechts von der Mitte des Bitstrings liegen und diese durch die Punktspiegelung links von der Mitte zu liegen kommen und sich deshalb die neue Prozessoradresse teilweise aus Bits der alten Prozessoradresse zusammensetzt.
Um genau die Mitte in einem Bitstring festzulegen muß man zwei Fälle unterscheiden:

1. n ist gerade, dann gilt:

$$A = a_n,\ldots,a_{(n/2)+1}a_{(n/2)},\ldots,a_1 \xrightarrow{R} A' = a_1,\ldots,a_{(n/2)}a_{(n/2)+1},\ldots,a_n$$

1.1 $p \leq n/2$

Für diesen Fall liegen A und A' im selben Prozessor, wenn gilt:

$$a_n = a_1$$
$$a_{n-1} = a_2$$
$$\ldots$$
$$a_{n-p+1} = a_p$$

Damit sind p der n Bits festgelegt. Es ergeben sich somit 2^{n-p} Möglichkeiten für invariante Adressen. Die Zahl der bewegten Daten ist demnach $2^n - 2^{n-p}$, oder

$$E_R = 2^n(1 - 2^{-p}) \text{ für } p \geq n/2$$

1.2 $p > n/2$

Es ist derselbe Prozessor, wenn gilt:

$$a_n = a_1$$
$$a_{n-1} = a_2$$
$$\ldots$$
$$a_{(n/2)+1} = a_{n/2}$$

Das sind Bedingungen für n/2 Bits. Somit gibt es noch $2^{n-(n/2)} = 2^{n/2}$ Möglichkeiten für invariante Adressen. Für nicht invariante Adressen bleiben dann $2^n - 2^{n/2} = 2^n(1 - 2^{-n/2})$ Möglichkeiten übrig. Es gilt also:

$$E_R = 2^n(1 - 2^{-n/2}) \text{ für } p > n/2$$

2. n ist ungerade, dann gilt:

$$A = a_n,\ldots,a_{(n+3)/2}a_{(n+1)/2}a_{(n-1)/2},\ldots,a_1 \xrightarrow{R} A' = a_1,\ldots,a_{(n-1)/2}a_{(n+1)/2}a_{(n+3)/2},\ldots,a_n$$

Mitte (bei A) Mitte (bei A')

2.1 $p \leq (n-1)/2$

Für diesen Fall liegen A und A' im selben Prozessor, wenn gilt:

$$a_n = a_1$$
$$a_{n-1} = a_2$$
$$\ldots$$

$$a_{n-p+1} = a_p$$

Dieser Fall ist identisch mit 1.1, so daß das Ergebnis lautet:

$$E_R = 2^n(1 - 2^{-p}) \text{ für } p \geq (n-1)/2$$

2.2 p > (n-1)/2

Es ist derselbe Prozessor, wenn gilt:

$$a_n = a_1$$
$$a_{n-1} = a_2$$
$$\ldots$$
$$a_{(n+3)/2} = a_{(n-1)/2}$$
$$a_{(n+1)/2} = a_{(n+1)/2}$$

Das sind Bedingungen für (n-1)/2 Bits, denn die letzte Gleichung gilt stets. Somit gibt es noch $2^{n-((n-1)/2)} = 2^{(n+1)/2}$ Möglichkeiten für invariante Adressen. Für nicht invariante Adressen bleiben dann $2^n - 2^{(n+1)/2} = 2^n(1 - 2^{(1-n)/2})$ Möglichkeiten übrig. Es gilt also:

$$E_R = 2^n(1 - 2^{(1-n)/2}) \quad \text{für } p > (n-1)/2$$

Ergebnis:

$$E_R = \begin{cases} 2^n(1 - 2\text{-}p) & \text{für } p \leq [n/2] \\ 2^n(1 - 2^{-[n/2]}) & \text{" } p > [n/2] \end{cases}$$

[n/2] ist die nächste ganze Zahl kleiner gleich n/2

Bemerkung:
Die Reversal-Permutation R hat die interessante Eigenschaft, daß von p > [n/2] an, die Interprozessor-Kommunikation nicht mehr mit der Zahl der Prozessoren zunimmt, sondern konstant bleibt!

Bewertung von Shuffle, Butterfly und Reversal Permutation hinsichtlich der Interprozessor-Kommunikation

Für diese Permutationen sind bei $N = 2^n$ Daten und $P = 2^p$ Prozessoren die folgende Zahl E an Daten auszutauschen :

Shuffle: $E_S = 2^n(1 - 2^{-p})$

Butterfly: $E_B = \begin{cases} 0 & \text{für } k \leq n-p \\ 2^{n-1} & \text{für } k > n-p \end{cases}$

Reversal: $E_R = \begin{cases} 2^n(1 - 2^{-p}) & \text{für } p \leq [n/2] \\ 2^n(1 - 2^{-[n/2]}) & \text{für } [n/2] < p < n \end{cases}$

Bemerkung:
Der Parameter k bei der Butterfly-Permutation war eine Erweiterung der Definition für Permutationsfunktionen. Er gibt an, daß sich die Permutationsfunktion nicht über alle n Bits einer Adresse A erstreckt, sondern nur über die k niederwertigen Bits. Shuffle und Reversal-Permutation erstrecken sich in unseren untersuchten Fällen der FFT stets über alle n Bits, so daß bei diesen Permutationen der Parameter k entfallen kann.

Es läßt sich damit die folgende Bewertung der Interprozessor-Kommunikation vornehmen:

Sei $[n/2] > 1$ und $k = n$, dann gilt:

1. $p = 1$: $E_S = E_R = E_{Bn}$

2. $1 < p \leq [n/2]$: $E_S = E_R > E_{Bn}$

3. $[n/2] < p < n$: $E_S > E_R > E_{Bn}$

Das heißt, daß die Butterfly-Permutation B_n die kleinste Interprozessor-Kommunikation aufweist und demnach für eine Parallelisierung am besten geeignet ist.

Die graphische Darstellung der Ungleichungen ist in Bild 4.13 abgebildet. Als Ergebnis der Datenflußanalyse von Shuffle-, Butterfly- und Reversal-Permutation läßt sich hinsichtlich seiner Größe sagen daß, die Menge des Datenflusses *bis zum Faktor 2 differieren* kann.

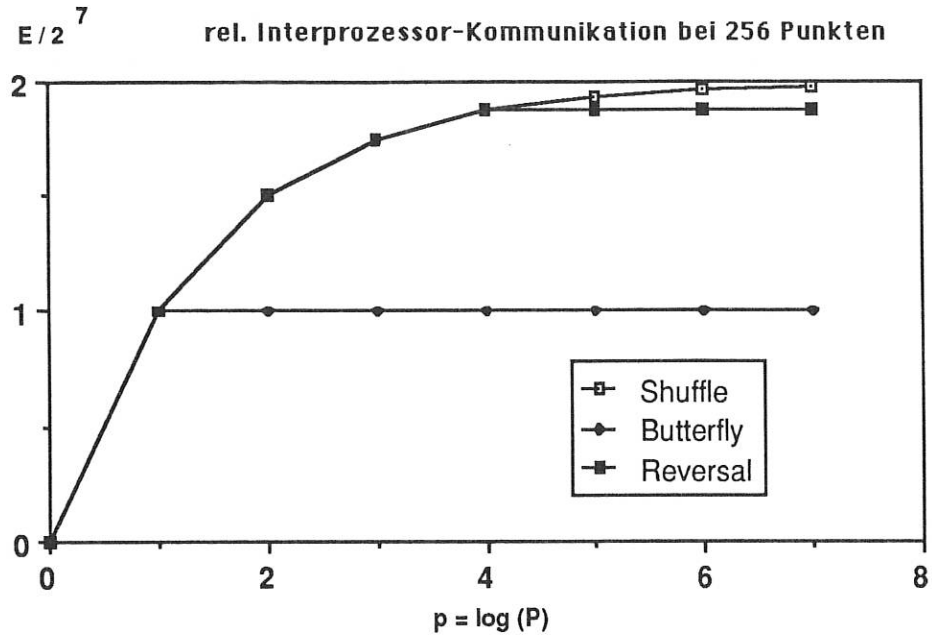


Bild 4.13:
Vergleich der Interprozessor-Kommunikation bei Butterfly, Reversal und Shuffle-Permutation für den von Fall von $N = 2^n$ Punkten.

4.4.2 Datenflußanalyse bei Cooley-Tukey, Gentleman-Sande, Stockham und Pease FFT

Die gesamte Datenmenge, die zwischen Prozessoren bei der FFT ausgetauscht werden muß, ist gleich der Summe der Daten zwischen den einzelnen Stufen. Die bezüglich der Ausführungszeit sequentiell gleichwertigen Formen der FFT verwenden unterschiedliche Permutationsfuntionen in den einzelnen Stufen, so daß die Interprozessor-Kommunikation sich unterscheidet. Z.B. aufgrund der graphischen Darstellung der FFT läßt sich die Art und Abfolge der Permutationsfunktionen angeben. So gilt für die Zahl der auszutauschenden Daten bei der Cooley-Tukey FFT:

$$E_{CT}(n,p) = E_R + E_{B(2)} + E_{B(3)} + \ldots + E_{B(n)} + E_S$$

Dies läßt sich nach dem vorangegangenen Kapitel darstellen als:

$$E_{CT}(n,p) = \underbrace{\left\{ \begin{array}{ll} 2^n(1-2^{-p}) & \text{für } p \le [n/2] \\ 2^n(1-2^{-[n/2]} & \text{für } n>p>[n/2] \end{array} \right\}}_{E_R} + \underbrace{p.2^{n-1}}_{E_{B(2)}\ldots E_{B(n)}} + \underbrace{2^n(1-2^{-p})}_{E_S}$$

bzw.

$$E_{CT}(n,p) = \begin{cases} 2^{n-1}(p+4-2^{-p+2}) & \text{für } p \le [n/2] \\ 2^{n-1}(p+4-2^{-p+1}-2^{-[n/2]+1}) & \text{für } n>p>[n/2] \end{cases}$$

näherungsweise gilt:

$$E_{CT}(n,p) \approx 2^{n-1}(p+4)$$

Für die Gentleman-Sande FFT ist $E_{GS} = E_{CT}$, da beide Formen spiegelsymmetrisch sind.

Die Stockham FFT hat als Interprozessor-Kommunikation:

$$E_{ST}(n,p) = E_S + B(n) + B(n-1) + ... + B(3) + 2.E_S$$

$$E_{ST}(n,p) = 3.2^n(1-2^{-p}) + \begin{cases} p.2^{n-1} & \text{für } n-p \ge 2 \\ (p-1).2^{n-1} & \text{für } n-p = 1 \end{cases}$$

$$E_{ST}(n,p) = \begin{cases} 2^{n-1}(p+6-6.2^{-p}) & \text{für } n-p \ge 2 \\ 2^{n-1}(p+5-6.2^{-p}) & \text{für } n-p = 1 \end{cases}$$

$$E_{ST}(n,p) \approx \begin{cases} 2^{n-1}(p+6) & \text{für } n-p \ge 2 \\ 2^{n-1}(p+5) & \text{für } n-p = 1 \end{cases}$$

Und für die Pease FFT gilt bezüglich der Interprozessor-Kommunikation:

$$E_{PS}(n,p) = E_R + n.E_S$$

$$E_{PS}(n,p) = \begin{cases} 2^n(1-2^{-p}) & \text{für } p \le [n/2] \\ 2^n(1-2^{-[n/2]}) & \text{für } n>p>[n/2] \end{cases} + n.2^n(1-2^{-p})$$

$$E_{PS}(n,p) = \begin{cases} 2^{n-1}.2(n+1-2^{-p+1}) & \text{für } p \le [n/2] \\ 2^{n-1}.2(n+1-2^{-p-2-[n/2]}) & \text{für } n>p>[n/2] \end{cases}$$

$$E_{PS}(n,p) \approx \frac{2^{n-1}.2(n+1)}{}$$

Damit kann man die folgende Rangfolge bezüglich der Interprozessor-Kommunikation aufstellen:

$$\mathbf{E_{CT} = E_{GS} < E_{ST} < E_{PS} \quad für\ p+6 < 2(n+1).}$$

Beispiel :

p = 4:				p = 8:			
	$E_{CT}/2^{n-1}$	$E_{ST}/2^{n-1}$	$E_{PS}/2^{n-1}$		$E_{CT}/2^{n-1}$	$E_{ST}/2^{n-1}$	$E_{PS}/2^{n-1}$
n = 10:	8	10	22	n = 10:	12	14	22
n = 15:	8	10	32	n = 15:	12	14	32

Tabelle 2:
Die relative Interprozessor-Kommunikation von Cooley-Tukey, Stockham und Pease FFT in Abhängigkeit von der Zahl der Punkte N und der Prozessoren P ($P = 2^p$, $N = 2^n$).

Ergebnis:

Die Cooley-Tukey bzw. Gentleman-Sande FFT ist von den untersuchten Formen (Gentleman-Sande, Cooley-Tukey, Stockham und Pease) diejenige mit der geringsten Interprozessor-Kommunikation. Das Verhältnis zwischen der besten und der schlechtesten Form der FFT bezüglich der auszutauschenden Datenmenge beträgt:

2,75 für N = 1024 und P = 16, bzw. 4 für N = 16 384 und P = 16).

4.4.3 Der Zeitbedarf von Butterfly-Berechnung und Kommunikation

Bei $N = 2^n$ Punkten ist die Zahl der zu berechnenden Butterflies

$$n*2^{n-1}.$$

Bei $P = 2^p$ Prozessoren und der FFT mit der geringsten Interprozessor-Kommunikation liegt die Zahl der auszutauschenden Daten bei:

2^{n-1} (p+4).

Das bedeutet, daß Kommunikation und Berechnung in derselben Größenordnung liegen, und daß damit *die Kommunikation wesentlich die Effizienz der parallelen FFT beeinflußt*, sofern nicht überlappend mit der Berechnung Daten ausgetauscht werden (können).

Beispiel:

Bei dem Prototyp von MULTITOP mit T414 Transputern liegt die Zeit für eine Festkomma-Multiplikation bei ca. 2 µs, und für eine Addition bei nur 100 ns. Ein Butterfly, der aus 4 verschiedenen skalaren Multiplikationen und 6 Additionen besteht, dauert demnach

ca. 8 µs.

Die Zeit, ein Byte zwischen zwei Prozessoren auszutauschen, beträgt bei MULTITOP ca. 5 us. Ein Daten-Transport besteht aus der Übertragung einer komplexen Zahl mit je 4 Bytes für Real- und Imaginärteil, so daß ein Daten-Transport

ca. 40 µs dauert.

Für N = 1024 Punkte und P = 4 Prozessoren ist demnach

$$T_{CPU} \approx 40 \text{ ms und } T_{MOV} \approx 120 \text{ ms} = 3{,}0\ T_{CPU}\ !$$

4.4.4 Kommunikation bei variabler Topologie

MULTITOP hat die Eigenschaft, daß Daten stets direkt, mit Hilfe eines Koppelnetzes, vom Erzeuger zum Verbraucher *ohne Zwischenstufe* transportiert werden. Bei nur einer Zwischenstufe verdoppelt sich bereits die Zeit für den Daten-Transport, da dann doppelt so viele Daten-Transportiert werden müssen. Die Effizienz ist davon wesentlich beeinflußt. Unter Zugrundelegung der vorigen Zahlen $T_{MOV} : T_{CPU} = 3{,}0$ sinkt die Effizienz bei Einführung einer Zwischenstufe um 43 % !

$$\text{Effizienz} = T_{CPU}/(T_{CPU} + T_{MOV})$$
$$T_{MOV1} : T_{CPU} = 3{,}0$$
$$T_{MOV2} : T_{CPU} = 6{,}0$$
$$\text{Effizienz}_1 = 1/4{,}0 = 0{,}25$$
$$\text{Effizienz}_2 = 1/7{,}0 = 0{,}14$$
$$\text{Effizienz}_2 = 0{,}57.\text{Effizienz}_1$$

Ein weiteres Argument für eine variable Topologie ist das folgende:

Die FFT hat von Stufe zu Stufe eine andere Topologie. Für die erste und letzte Stufe sowie die

Mittelteile gibt es jeweils andere Bildungsgesetze: Reversal, Shuffle, Butterfly. Dies bedeutet, daß es verschiedene inkompatible Topologien gibt, deren Vereinigungsmenge einen Graphen mit einer sehr großen Zahl von Links pro Prozessor erfordern würde. Die maximale Zahl von Links pro Prozessor steigt zudem mit der Zahl von Prozessoren. Dies bedeutet, daß es von einer gewissen Zahl von Prozessoren an technisch unmöglich ist, bei fester Topologie, Daten ohne Zwischenstufe zu übertragen, da dazu eine zu große Zahl von Links pro Prozessor nötig wäre.

Hinzu kommt, daß bei den heutigen Mikroprozessoren die Geschwindigkeit für eine Multiplikation nicht wesentlich langsamer ist als die Daten-Übertragungsgeschwindigkeit des Prozessors:

$$T_{MUL} \approx 1us \text{ und } v_{MOV} \approx 1 \text{ MByte/s}$$

Und es müssen

$$O(2^{n-1}.n)$$

Multiplikationen bzw.

$$O(2^{n-1}(p+4))$$

Daten-Transporte durchgeführt werden, was ebenfalls in derselben Größenordnung liegt. Das bedeutet, daß die Kommunikation entscheidend für Geschwindigkeit und Effizienz ist, denn die additive Rechenleistung kann durch Hinzufügen neuer Prozessoren leicht erhöht werden. Ein Multiprozessor mit fester Topologie ist deshalb im normalen Anwendungsbereich, wo $P << N$ ist, hinsichtlich der parallelen Fast Fourier-Transformation ineffizient.

Zusammenfassung:

1. Die kleinste parallele Einheit für ein Ensemble aus von Neumann Rechnern ist bei der FFT ein Butterfly.

2. Die größte Flexibilität in der Wahl der Prozessoren und damit der Ausführungsgeschwindigkeit (= Skalierbarkeit), sowie hohe Leistung ohne Pipelining ermöglicht nur die zeilenweise Parallelisierung der FFT.

3. Die sequentiell gleichwertigen Formen der FFT unterscheiden sich bei paralleler Ausführung hinsichtlich ihrer Interprozessor-Kommunikation um ca. den Faktor 2 - 5! Die Formen mit der geringsten Kommunikation sind die Cooley-Tukey- und die Gentleman-Sande-FFT.

4. Die Interprozessor-Kommunikation beeinflußt bei der FFT wesentlich die Effizienz des Systems. Ein Multiprozessor mit fester Topologie ist ineffizienter als einer mit variabler Topologie.

4.5 Ein Mathematisches Modell der Berechnungszeiten und des Wirkungsgrades

Die Motivation dafür, ein mathematisches Modell für die Kenngrößen eines parallelen Algorithmus aufzustellen liegt darin, für eine größere Zahl von Prozessoren als real zur Verfügung stehen extrapolieren zu können, und damit *Voraussagen* machen zu können, ob ein großes System zu bauen lohnt oder nicht.

Dieses Modell basiert auf den folgenden Überlegungen:

1. Die Gesamtzeit zur Berechnung aller Butterflies auf <u>einem</u> Prozessor ist:

   $$T_{BUT} = a'2^{n-1}n = a*2^n*n$$

   Hierin ist a eine dimensionsbehaftete Proportionalitätskonstante der Einheit [sec].

2. Die Gesamtzeit aller Datentransporte *würde auf einem Prozessor* dauern:

   $$T_{MOV} = b*2^n(1+(p/4)-2^{-p}) \text{ für } 0 < p \leq [n/2]$$

   Wiederum ist b eine Proportionalitätskonstante.

3. Diese Zeiten verteilen sich bei einer zeilenweisen Parallelisierung gleichmäßig auf $P = 2^p$ Prozessoren.

4. Die Gesamtzeit ist die Summe aus Butterfly-Berechnung- und Daten-Transportzeit.

5. Wenn der Daten-Transport nicht direkt, sondern über d Zwischenstufen stattfindet, muß das d-fache an Daten-Transportiert werden.
   Damit kennzeichnet also *d die mittlere Entfernung zwischen zwei Prozessoren.*

Aus diesen Überlegungen erhält man die folgenden Formeln:

Butterfly Zeit: $T_{BUT}(n,p) = a*2^{n-p}*n$

Datentransportzeit: $T_{MOV}(n,p,d) = b*d*2^{n-p}(1+(p/4)-2^{-p})$

FFT Berechnungszeit: $T_{FFT} = T_{BUT} + T_{MOV}$

$$T_{FFT}(n,p,d) = 2^{n-p}(an + bd(1+(p/4)-2^{-p}))$$

Wirkungsgrad: $e = T_{SEQ}/T_{PAR} = T_{BUT} / (T_{BUT} + T_{MOV})$

$$e(n,p,d) = n / [n + cd(1+(p/4)-2^{-p})] \text{ , mit } c = b/a$$

Speedup: $S_P = e*2^p$

Im nächsten Kapitel werden die Konstanten a,b anhand zweier Meßpunkte bestimmt, und damit die Güte des Modells für die übrigen Meßwerte überprüft.

Zusammenfassung:

1. Es wurde ein allgemeines Modell der Berechnungszeiten und des Wirkungsgrades der Cooley-Tukey bzw. Gentleman Sande FFT angegeben.

2. Das Modell ist für eine beliebige Anzahl Punkte, Prozessoren und Verbindungen zwischen den Prozessoren geeignet. Dazu ist die mittlere Entfernung zwischen zwei Prozessoren als Masszahl anzugeben.

4.6 Messergebnisse mit MULTITOP

Mit dem Multiprozessor MULTITOP wurden für die parallele Gentleman-Sande FFT Messungen über die Ausführungszeit der FFT und über die reine Berechnungszeit ohne Kommunikation gemacht. Dabei waren die Zahl der Prozessoren und die Zahl der Punkte Parameter. Die genauen Messbedingungen waren:

1. Dasselbe Programm (GS4PFRM.OCC) für 1, 2 und 4 Prozessoren
2. T_{FFT} kennzeichnet die Berechnungszeit ohne Daten Reversal am Ende der FFT
3. T_{MOV} ist die Daten-Transportzeit aus lokaler *und* globaler Kommunikation. (Im Modell wurde die lokale Kommunikation vernachlässigt)

Aus diesen Meßwerten ergeben sich als abgeleitete Größen durch Rechnung:

1. Die Butterfly Berechnungszeit T_{BUT}
2. Der Speedup S_P
3. Der Wirkungsgrad e
4. Das Verhältnis von Daten-Transportzeit zu Gesamtzeit T_{MOV}/T_{FFT}

Das Ergebnis der Berechnungen ist in Tabelle 3 zusammengestellt:

Gesamtzeit TFFT in ms:			
	Zahl der Prozessoren		
	1	2	4
16	2 944	2 048	1 344
32	7 296	4 800	3 072
64	16 768	10 688	6 400
128	40 064	24 448	14 080
256	93 312	55 424	31 552
512	214 592	123 585	69 568
1024	478 080	276 480	152 960
2048	-	604 416	334 976

Gesamtzeit TMOV in ms:			
	Zahl der Prozessoren		
	1	2	4
16	1 280	1 216	960
32	3 200	2 752	2 048
64	7 552	6 080	4 096
128	17 856	13 632	8 704
256	40 960	30 080	19 264
512	96 256	65 408	40 704
1024	215 360	144 960	88 256
2048	-	315 392	190 144

Butterfly-Berechnungszeit

Zahl d. Punkte	Zahl der Prozessoren		
	1	2	4
16	1 664	832	384
32	4 096	2 048	1 024
64	9 216	4 608	2 304
128	22 208	10 816	5 376
256	52 352	25 344	12 288
512	118 336	58 176	28 864
1024	262 720	131 520	64 704
2048	-	289 024	144 832

Speedup/Wirkungsgrad

Zahl d. Punkte	Zahl der Prozessoren	
	2	4
16	1,44/0,72	2,19/0,54
32	1,52/0,76	2,38/0,59
64	1,57/0,78	2,62/0,66
128	1,64/0,82	2,85/0,71
256	1,68/0,84	2,96/0,74
512	1,74/0,86	3,08/0,77
1024	1,73/0,87	3,13/0,78

Verhältnis Daten-Transportzeit zu Gesamtzeit

Zahl d. Punkte	Zahl der Prozessoren		
	1	2	4
16	0,43	0,59	0,71
32	0,44	0,57	0,67
64	0,45	0,57	0,64
128	0,45	0,56	0,62
256	0,44	0,54	0,61
512	0,45	0,53	0,59
1024	0,44	0,52	0,58
2048	-	0,52	0,57

Tabelle 3:
Meßwerte der Ausführungszeit der FFT und der Daten-Transportzeit sowie daraus berechnete Größen.

Anhand der gewonnenen Werte ist es möglich, die Güte des Modells für Berechnungszeiten zu testen. Dazu muß man die Konstanten a, b anhand zweier Meßpunkte bestimmen. Aus n = 10 und p = 0 (1 Prozessor) erhält man: a = 46.7. Und aus n = 10 und p = 1 (2 Prozessoren): b = 97.3.

Die Gleichungen für MULTITOP (d = 1) sind demnach:

FFT Berechnungszeit: $T_{FFT} = 2^{n-p}(46.7 + 97.3(1+(p/4)-2^{-p}))$

Daten-Transportzeit: $T_{MOV} = 97.3*2^{n-p}(1+(p/4)-2^{-p})$

Wirkungsgrad: $e = n / [\, n + 2.01(1+(p/4)-2^{-p}) \,]$

Die Güte des Modells kann anhand der Abweichungen in den graphischen Darstellungen der gemessenen und berechneten Größen beurteilt werden. Für die Gesamtausführungszeit T_{Tot} ist in der graphischen Darstellung keine Abweichung sichtbar (Bild 4.14):

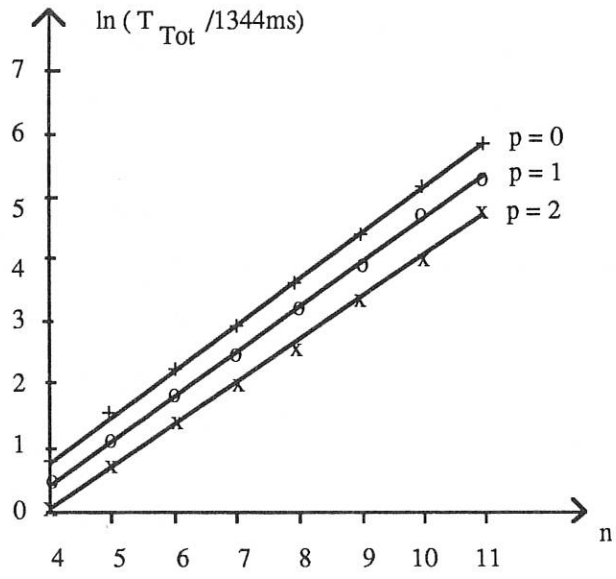


Bild 4.14:
Berechnungszeit der parallelen Gentlemen-Sande FFT für 32 bis 2048 Punkte bei 1,2 und 4 Prozessoren in Meßwerten und Rechnung (durchgezogene Linien).

Ähnlich gut schneidet der berechnete Beschleunigungsfaktor bei zwei Prozessoren ab, der in Bild 4.15 zusammen mit den gemessenen Werten dargestellt ist:

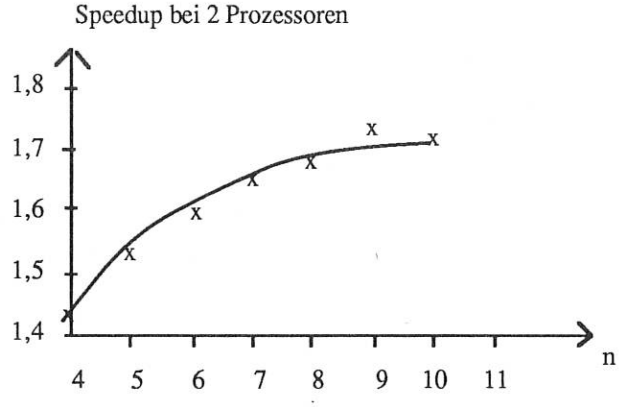


Bild 4.15 :
Speedup der parallelen FFT von 32 bis 2048 Punkten bei 2 Prozessoren nach Messung und Rechnung.

Bei vier Prozessoren werden bei einer kleineren Zahl von Punkten (bis ca. 127) Abweichungen zwischen Messung und Rechnung sichtbar.

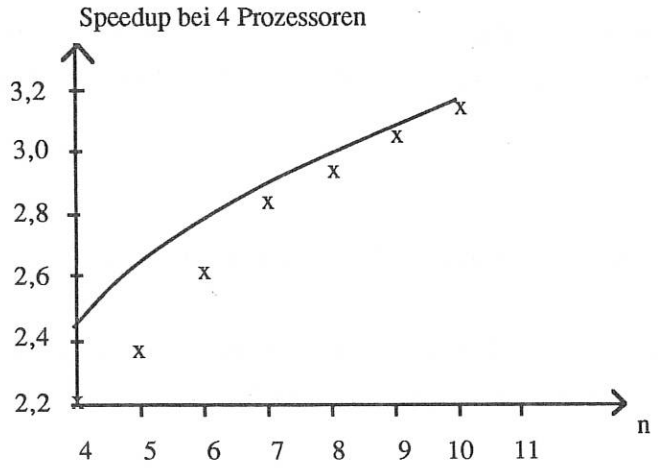


Bild 4.16:
Beschleunigungsfaktor der parallelen Gentlemen-Sande FFT auf 4 Prozessoren in Meßwerten und Rechnung (durchgezogene Linien). $n = \log_2 N$.

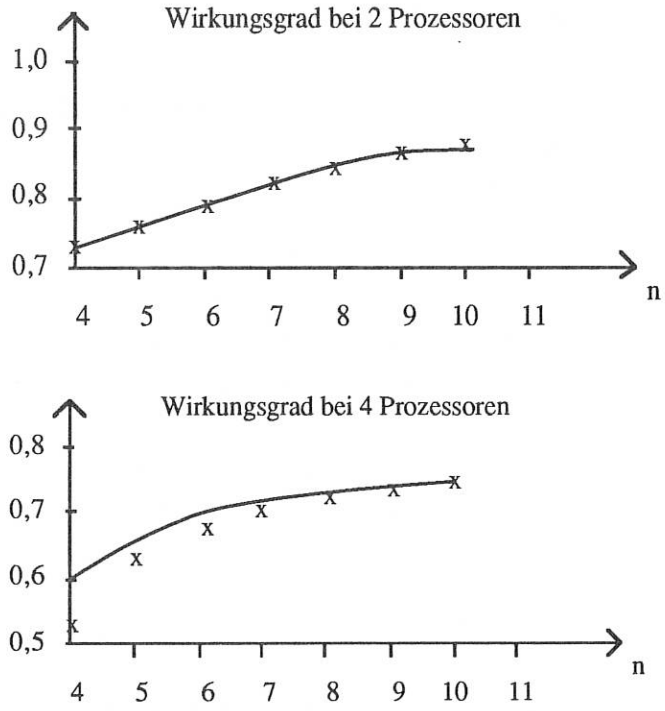


Bild 4.17:
Wirkungsgrad der parallelen Gentlemen-Sande FFT auf 2 und 4 Prozessoren in Meßwerten und Rechnung (durchgezogene Linien). $n = \log_2 N$.

Ergebnis:

1. Die berechneten Werte stimmen z.T. sehr gut mit den gemessenen überein. Die größte

Abweichung ist beim Wirkungsgrad mit 12% bei n=4) zu verzeichnen. Die Brauchbarkeit des Modells ist damit erwiesen.

2. Die zeilenweise Parallelisierung der Gentleman-Sande FFT hat bei direkter Kommunikation zwischen den Prozessoren Beschleunigungsfaktoren von 1.7 bzw. 3.2 (bei n=10 und 2 bzw. 4 Prozessoren).

4.7 Extrapolation auf eine sehr große Zahl von Prozessoren

Die gute Übereinstimmung des Modells mit der Wirklichkeit erlaubt die Extrapolation auf P > 4 Prozessoren mit Hilfe des Modells. Insbesondere sind dabei 2 Fragen interessant:

1. Von welcher Prozessorzahl an ist kein Leistungszuwachs mehr zu erwarten ?

2. Bei welcher Prozessorzahl ist der Wirkungsgrad < 50% (Grenze der Wirtschaftlichkeit) ?

Die Beantwortung der ersten Frage liefert die folgende Überlegung:

Die Leistung steigt ab dem Punkt nicht mehr an, bei dem sich bei Verdoppeln der Prozessorzahl der Wirkungsgrad halbiert. Bei einem Beschleunigungsfaktor von

$$S_P = 2^p * n \; / \; [n + cd(1 + (p/4) - 2^{-p})]$$

wird dieser Punkt, zumindest theoretisch, nicht erreicht, da der Zähler schneller als der Nenner des Wirkungsgrades wächst. Die Leistung würde also beliebig zunehmen.

Um die Frage nach der Prozessorzahl für 50 % Wirkungsgrad zu beantworten, wird die Gleichung für den Wirkungsgrad näherungsweise nach p aufgelöst und e = 0,5 gesetzt:

$$p = 4 [(n/cd) - 1]$$

Daraus erhält man für ein cd von 2,08, wie es bei MULTITOP der Fall ist, und für z.B. 1024 Punkte (n = 10) ein p von:

$$p = 16.$$

D.h., ab 65 536 Prozessoren würde der Wirkungsgrad kleiner als 50% sein. Natürlich können bei 1024 Punkten maximal 512 Prozessoren tatsächlich verwendet werden.

Das Modell erlaubt auch Aussagen über den Wirkungsgrad bei anderen Topologien von Prozessor-Verbindungen zu machen. Dazu muss nur die mittlere Entfernung zwischen zwei Prozessoren

bekannt sein. Es wird jetzt exemplarisch die mittlere Entfernung bei einer Ring- und einer Hypercube-Topologie hergeleitet. Zur Bestimmung der mittleren Entfernung in einem Ring aus P Prozessoren muß man zwei Fälle unterscheiden:

a) *P ist ungerade* , dann gilt:
Von einem beliebigen Prozessor P_0 aus gibt es in jeder Richtung des Rings die gleiche gerade Zahl von Nachbarn, nämlich:

$$(P -1) / 2.$$

Weiterhin habe jeder Prozessor den Abstand eins zu seinem Nachbarn. Die Summe der Abstände aller P-1 Prozessoren zum Prozessor P_0, läßt sich wegen der Symmetrie so berechnen, daß die Summe entlang einer Richtung im Ring bestimmt und diese verdoppelt wird:

$$d = 2 (1 + 2 + ... + (P-1)/2).$$

Daraus erhält man: $$d = (P^2-1) / 4.$$

b) *P ist ungerade*
In diesem Fall gibt es *einen* ausgezeichneten Prozessor, der den größten Abstand von P_0 hat:

$$d_{max} = P/2.$$

Wiederum kann die Summe aller Entfernungen dadurch bestimmt werden, daß der Wert für eine Richtung im Ring berechnet und verdoppelt wird. Hinzu kommt die Entfernung zum weitesten Prozessor, so daß sich ergibt:

$$d = 2 (1 + 2 + ... + (P-2)/2) + P/2$$

Man erhält durch Summation:

$$d = P^2/4.$$

Damit steht die Summe der Entfernungen fest. Zur Berechnung der mittleren Entfernung muß die Summe durch die Zahl der Nachbarn dividiert werden. Man erhält also:

a)	P gerade:	$d_{Ring} = P^2 / [4 (P-1)]$
b)	P ungerade:	$d_{Ring} = (P^2 - 1) / [4 (P-1)]$

Die Bechnung der mittleren Entfernung in einem n-dimensionalen Hypercube wird so vollzogen, daß zuerst alle $P = 2^p$ Prozessoren von 0 bis P-1 numeriert werden. Die Numerierung vollzieht sich in der Weise, daß sich die Nummern der Nachbarn eines beliebigen Prozessors P_0 um genau ein Bit im Vergleich zur Nummer von P_0 unterscheiden. Die Nummer eines Prozessors kann deshalb

als die geometrische Position dieses Prozessors im Überwürfel angesehen werden, weil benachbarte Prozessoren auch benachbarte Nummern haben. Wenn der Prozessor P_0 die Nummer

$$I = i_p i_{p-1},\ldots,i_1$$

hat, erhält man die Positionen der Nachbarn, die den Abstand eins zum Prozessor P_0 haben, durch Invertieren *eines* der p Adressbits von P_0. P_0 hat also p Nachbarn mit Abstand eins. Analog erhält man die Zahl der Nachbarn von P_0 mit Abstand k, indem man k der p Bits von P_0 invertiert. Es gibt n über k Möglichkeiten dies tun, also hat P_0 n über k Nachbarn mit der Entfernung k. Die Summe aller Entfernungen k erhält man dadurch, daß man k mit der Zahl der Nachbarn bei dieser Entfernung multipliziert:

$$d_k = k * [\text{ p über k}].$$

Die Gesamtsumme aller Entfernungen erhält man durch Summation aller d_k über alle k:

$$d = \sum_{1}^{p} k * [\text{ p über k}] .$$

Nach zweifacher Substitution von p-1 durch p´ und k-1 durch k´ erhält man:

$$d = p * \sum_{0}^{n-1} [\text{p´über k´}].$$

Damit ist die Summation auf die bereits bekannte Summierung der Zahlen eines Pascal´schen Dreiecks zurückgeführt, so daß sich ergibt:

$$d = p*2^{p-1}.$$

Jeder Prozessor P_0 hat P-1 Nachbarn, so daß sich für die mittlere Entfernung ergibt:

$$d_{Hypercube} = p*2^p / [2^p - 1].$$

Bei *unendlich vielen Prozessoren* streben die mittleren Entfernungen gegen die Grenzwerte:

$$d_{Cube} \rightarrow p/2$$
$$d_{Ring} \rightarrow P/4.$$

Bild 4.18 zeigt den Beschleunigungsfaktor und den Wirkungsgrad der FFT für diese Topologien in Abhängigkeit von der Zahl der Prozessoren und zum Vergleich das Verhalten von MULTITOP am Beispiel von N = 1024 Punkten.

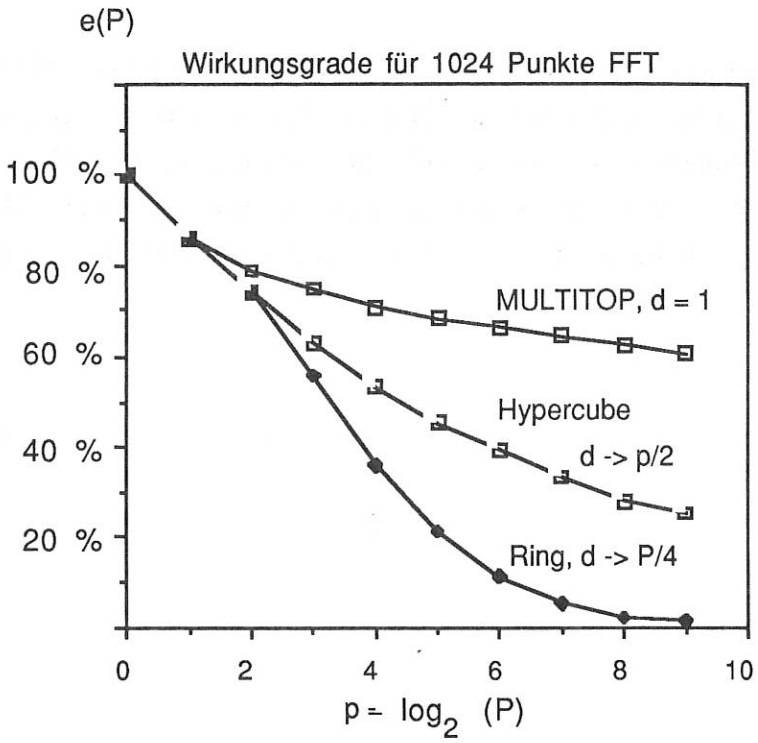


Bild 4.18:
Die Wirkungsgrade der FFT bei sehr hohen Prozessorzahlen (bis zu 512 Prozessoren bei 1024 Punkten). Es tritt bei größeren Prozessorzahlen ein starker Abfall des Wirkungsgrades ein, sobald die mittlere Entfernung zwischen den Prozessoren größer als eins ist ($p = \log_2 P$).

Zusammenfassung:

Die einzelnen Varianten der schnelle Fouriertransformation wurden einer Datenflußanalyse hinsichtlich der Menge und der Richtung des Datenflusses unterzogen. Es konnte daraus die Cooley-Tukey bzw. Gentleman-Sande als die FFT mit der geringsten Interprozessor-Kommunikation bestimmt werden. Diese eignen sich deshalb besonders zur Parallelisierung, weil die Effizienzverluste durch Datentransfer minimal sind. Die Gentleman-Sande wurde auf dem MULTITOP-Rechner implementiert. Weiterhin wurde ein mathematisches Modell zur Extrapolation des Wirkungsgrades der FFT bei hohen Prozessorzahlen entwickelt und durch Messungen an MULTITOP für 1,2 und 4 Prozessoren bestätigt. Es zeigte sich, daß die parallele schnelle Fouriertransformation einen hohen Anteil von Interprozessor-Kommunikation im Vergleich zur Berechnungszeit für die Butterflies aufweist. Der Wirkungsgrad bei paralleler Berechnung wird somit wesentlich von der Interprozessor-Kommunikation bestimmt und fällt bei höheren Prozessorzahlen (> 8) und einer mittleren Entfernung zwischen den Prozessoren, die größer als eins ist, stark ab.

Mit der variablen Topologie und dem schnellen Umschalten der Topologien bei MULTITOP kann nach dem mathematischen Modell zur Extrapolation bei bis zu 65 536 Prozessoren ein Wirkungsgrad von > 50 % erzielt werden. Dies kann als ein Beweis für die *Steigerung der*

Effizienz bei der Ausführung paralleler Programme durch den MULTITOP-Rechner angesehen werden.

Die Programmierung paralleler Prozesse wird durch die Flexibilität von MULTITOP ebenfalls erleichtert. Dies wurde am Beispiel der FFT deutlich, bei der die Implementierung der Interprozessor-Kommunikation sich deshalb als einfach erwies, weil stets direkt, ohne Einschaltung von Zwischenstufen, Daten ausgetauscht werden konnten. Das Konzept von MULTITOP wurde somit für den Fall der parallelen schnellen Fouriertransformation insgesamt bestätigt.

5. Die parallele Spline-Approximation

In diesem Kapitel wird anhand des Beispiels der Spline-Approximation gezeigt, wie man ein weiteres numerisches Problem parallelisieren kann, um dadurch seine Ausführungszeit zu reduzieren. Es werden dabei Kenntnisse aus der Approximationstheorie vorausgesetzt. In diesem Zusammenhang sei auf den Anhang und das Literaturverzeichnis verwiesen.

Die Spline-Approximation hat die Anwendung, daß man mit ihr Daten (Meßwerte) glätten und reduzieren kann. Ihre Parallelisierung führt auf die Grundelemente paralleler Algorithmen aus der linearen Algebra, wie Skalarprodukt berechnen, Matrix-Multiplikation und lineares Gleichungssystem lösen. Im folgenden wird zuerst der theoretische Hintergrund der Splines angegeben, dann wird ein gut parallelisierbarer Algorithmus zur parallelen Spline-Approximation mit Datenreduzierung hergeleitet. Zuletzt werden die erwarteten Ausführungszeiten der Grundelemente, auf die der Algorithmus aufbaut, unter besonderer Berücksichtigung der Topologie bei verschiedenen Prozessorzahlen untersucht. Es zeigt sich, daß die parallele Matrixmultiplikation, -transposition und das Lösen eines linearen Gleichungssystems jeweils eine eigene optimale Topologie bezüglich ihrer Ausführungszeit aufweisen.

5.1 Interpolation, Approximation und Glättung einer Folge von Messwerten

In Bild 5.1 sind die Messwerte (x) eines verrauschten Sinussignals dargestellt. Mit Hilfe des vorgestellten Verfahrens ist eine Glättung der Messwerte, die das Rauschen elimiert, bei gleichzeitiger Reduktion der Zahl der abzuspeicherden Datenmenge möglich.

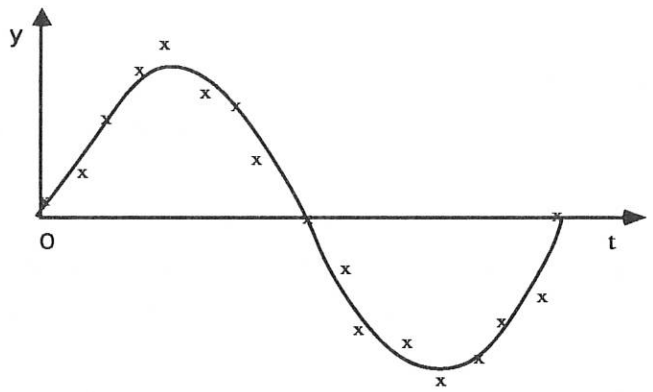


Bild 5.1:
Glättung eines verrauschten Sinussignals mit Hilfe kubischer Spline-Approximation. Dabei wurden die Meßwerte im Beispiel auf 1/3 der urspünglichen Menge reduziert!

Als Interpolation einer durch Stützstellen (Meßwerte) gegebenen Funktion bezeichnet man eine Kurve, die sowohl zwischen den Stützstellen existiert, als auch die Stützstellen exakt enthält. Die

Approximation einer Folge von Stützstellen ist eine Ausgleichskurve, die z.B. nach dem Prinzip der kleinsten Quadrate konstruiert wird, und die sowohl zwischen den Stützstellen definiert ist, als auch die Abweichung zwischen ihr und den Stützstellen minimiert; bei einer vorgegeben Randbedingung, wie z.B. dem Grad der Kurve. Die Glättung von Messwerten erfolgt dadurch, daß man eine "von Natur aus" glatte Kurve durch die Meßwerte legt.

5.2. Der Spline, die Kurve kleinster Krümmung

Die Glattheit einer Kurve drückt sich in der Summe all ihrer Krümmungen aus. Man kann nun zeigen [Rein1], daß es genau eine Kurve gibt, die optimal glatt, d.h. minimal gekrümmt ist: die Spline Funktion.

Aufgabe:
Unter all den Funktionen g(x) soll eine gefunden werden so, daß gilt:

$$\int_{x_0}^{x_n} g''(x)^2 \, dx = \text{Minimum}$$

(= möglichst glatt)

unter der Randbedingung:

$$\sum_{i=1}^{n} (g(x_i) - y_i)^2 \leq s, \quad g \text{ aus } C^2[x_1,x_n]$$

(= möglichst nahe an den Messwerten)

Wobei die x_i die Abszissen und die y_i die dazugehörenden Ordinaten der Meßwerte sind, und s eine vorzugebende obere Schranke für die größte Fehlersumme der Approximation sein soll.

Die Lösung dieser Minimierungsaufgabe ist eine Funktion, die aus zweimal stetig differenzierbar aneinandergehefteten Parabelbögen 3. Ordnung besteht [Rein1]. Diese zweimal stetig differenzierbar aneinandergehefteten Parabelbögen werden als kubische Splines bezeichnet.

In der Natur erscheinen Splines immer dann, wenn man elastische Medien (z.B. Balken) um Ecken und Kanten biegt. Diese Medien verformen sich so, daß ihre Verformungsenergie, und damit ihre Krümmung, minimal ist, d.h. sie biegen sich so wie ein Spline.

5.3. Spline-Approximation ohne Datenreduktion

Die gebräuchlichste Definition der Splines besteht darin, die einzelnen Parabelbögen direkt anzugeben:

$$s_i(x) = a_i + b_i\,(x - x_i) + c_i\,(x - x_i)^2 + d_i\,(x - x_i)^3$$

$$\text{für } x \text{ aus } [x_i, x_{i+1}],\ i = 0,1,\ldots,n-1,$$

wobei zur Bestimmung der Koeffizientenvektoren **a**, **b**, **c**, **d** die folgende Prozedur auszuführen ist [Jord1] :

1. Bestimmen von **c** durch Lösen eines linearen Gleichungssystems der Art

$$\mathbf{M} * \mathbf{c} = \mathbf{y}$$

Genauer gilt für äquidistante Stützstellen:

$$c_0 = 0$$
$$6\,c_{i-2} - 23\,c_{i-1} + 40\,c_i - 23\,c_{i+1} + 6\,c_{i+2} = 3(\,y_{i+1} - 2\,y_i + y_{i-1}\,)$$
$$i = 1,2,\ldots,n-1$$

2. Bestimmen von **a**, **b** und **d** durch

$$a_0 = y_0 - 2c_1$$
$$a_i = y_i - 2\,(\,c_{i-1} - 2\,c_i + c_{i+1}\,),\ i = 1,2,\ldots,n-1$$
$$b_i = (a_{i+1} - a_i) - 1/3\,(c_{i+1} + 2\,c_i),\ i = 0,1,\ldots,n-1$$
$$d_i = 1/3\,(\,c_{i+1} - c_i),\ i = 0,1,\ldots,n-1$$

(äquidistante Stützstellen)

Zusammenfassend läßt sich sagen, daß

$$a_i = f_1(c_{i-1}, c_i, c_{i+1})$$
$$b_i = f_2(a_{i+1}, a_i, c_{i+1}, c_i)$$
$$d_i = f_3\,(c_{i+1}, c_i) \text{ ist.}$$

Es liegt somit keine Rekursion vor, die zu Schwierigkeiten bei der Parallelisierung führen würde. Die sequentielle Prozedur zur Berechnung der Spline Koeffizienten läßt sich mit Hilfe des in Bild 5.2 angegebenen Schemas zusammenfassen. Die einzelnen Teile der Spline-Berechnung, wie lineares Gleichungsystem lösen, **a**, **b**, und **d** berechnen, lassen sich sehr gut parallelisieren und die Berechnung von **a**, **b**, und **d** auch vektorisieren. Mit dem geschilderten Verfahren ist allerdings keine Datenreduktion möglich, so daß im weiteren darauf nicht mehr eingegangen wird.

Löse: **Mc = y** , daraus: **c, d => a => b**

Bild 5.2:
Prozedur der Spline-Berechnung ohne Datenreduktion.

5.4. Ein Verfahren zur Approximation mit Datenreduktion

Eine alternative Methode zur Definition eines Splines ist die, ihn als Linearkombination von Basisfunktionen anzugeben:

$$s(x) = p_0 B_0(x) + p_1 B_1(x) + ... + p_{n-1} B_{n-1}(x)$$

Die Basisfunktionen $B_i(x)$ sind in diesem Fall sog. Basissplines. Ein Basisspline an der Stelle i ist nur zwischen den Stellen x_i und x_{i+4} von Null verschieden, so daß er den kürzestmöglichen symmetrischen Spline darstellt, der überhaupt konstruiert werden kann. Aufgrund seiner lokalen Begrenztheit wird er als Basisspline bezeichnet (die genaue Definition des Basissplines ist im Anhang enthalten). Man benötigt vier Parabelbögen, um einen Basisspline zu konstruieren:

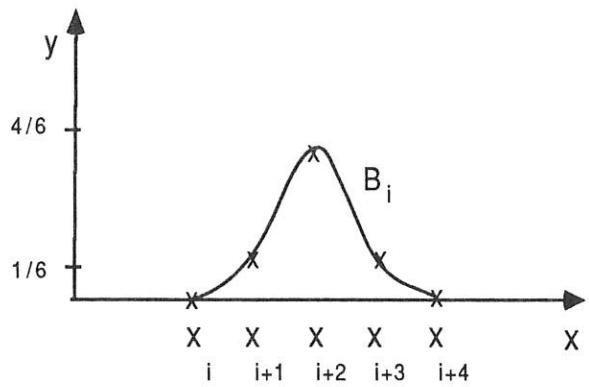


Bild 5.3:
Der Basisspline B_i beginnt bei der Stelle x_i und endet vier Stützstellen weiter. Er ist der kürzestmögliche symmetrische Spline und besteht aus vier Parabelbögen.

Durch gewichtete Addition einer gewissen Zahl gegeneinander verschobener Basissplines lassen sich beliebige Funktionswerte F(x) interpolieren oder approximieren. Die Interpolation einer Funktion ist in Bild 5.4 exemplarisch dargestellt. Entscheidend für eine gute Parallelisierbarkeit ist, nach welchem Verfahren die Gewichte p_i berechnet werden können. Es wirrd jetzt eine gut parallisierbare Methode zur Berechnung der Gewichtsfaktoren hergeleitet. Die dazu notwendigen Begriffe wie "Residuum" und "Norm" etc. werden im Anhang erklärt.

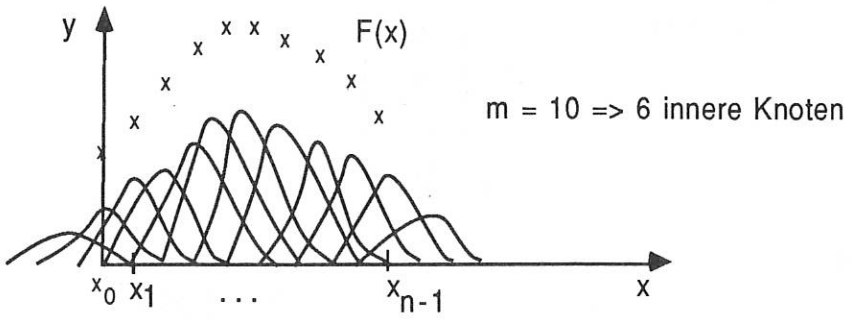


Bild 5.4:
Interpolation einer Funktion F(x) $[x_0,x_{n-1}]$ durch gewichtete Addition verschobener Basisfunktionen B_i. Die Funktion F(x) ist durch x an diskreten Stellen x_i dargestellt.

Herleitung des Algorithmus der Spline-Approximation

Sei s ein Spline und

$$\mathbf{r} = \mathbf{y} - \mathbf{s}, \text{ mit } \mathbf{r} = (r_0, r_1,...,r_{n-1})$$

der Vektor der Abweichung (Residuum) , dann gilt für dessen euklidische Norm:

$$\| r \|_2^2 = r_o^2 + r_1^2 + ... + r_{n-1}^2$$

Für eine optimale Approximation im obigen Sinne muß die Norm des Residuums minimiert werden [Rein3]. Für ein Element des Residuums gilt:

$$r_i = y_i - p_0B_0(x) - p_1B_1(x) - ... - p_{n-1}B_{n-1}(x).$$

Und für alle Elemente von r:

$$\mathbf{r} = \mathbf{y} - \mathbf{Bp}, \text{ mit}$$

$$\mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ ... \\ y_{m-1} \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} B_0(x_0) & B_1(x_0) & ... & B_{n-1}(x_0) \\ B_0(x_1) & B_1(x_1) & ... & B_{n-1}(x_1) \\ & ... & & \\ B_0(x_{m-1}) & B_1(x_{m-1}) & ... & B_{n-1}(x_{m-1}) \end{pmatrix}$$

$$\mathbf{p} = \begin{pmatrix} p_0 \\ p_1 \\ \dots \\ p_{n-1} \end{pmatrix}$$

Weiterhin ist:

$$\begin{aligned} \| \mathbf{r} \|_2^2 &= \| \mathbf{y} - \mathbf{B}\mathbf{p} \|_2^2 \\ &= \mathbf{y}^T\mathbf{y} - 2\mathbf{p}^T\mathbf{B}^T\mathbf{y} + \mathbf{p}^T\mathbf{B}^T\mathbf{B}\mathbf{p} \quad * \end{aligned}$$

* Bemerkung: 1.) $\mathbf{B}^T$ ist die transponierte Matrix,

2.) $\| \mathbf{y} - \mathbf{B}\mathbf{p} \|_2^2$ wird analog zu $(a - b)^2 = a^2 - 2ab + b^2$ berechnet.

Nun ist eine notwendige und hinreichende Bedingung für

$$\| \mathbf{r} \|_2 = \text{MIN},$$

wenn

$$\partial/\partial p_0 F = \partial/\partial p_1 F = \dots = \partial/\partial p_{n-1} F = 0,$$
für $F = \| \mathbf{r} \|_2^2$.

Dies ist dann der Fall, wenn

$$\partial/\partial p_i F = 0 - 2\mathbf{e}_i^T\mathbf{B}^T\mathbf{y} + \mathbf{e}_i^T\mathbf{B}^T\mathbf{B}\mathbf{p} + \mathbf{p}^T\mathbf{B}^T\mathbf{B}\mathbf{e}_i \quad *$$

* Bemerkung: 1.) $\mathbf{e}_i$ ist der i. Einheitsvektor

2.) $\partial/\partial p_i F$ erfolgt gemäß der Produktregel analog zu $\partial/\partial x\, f*g = f'g + fg'$,

mit

$$\mathbf{p}^T\mathbf{B}^T\mathbf{B}\mathbf{e}_i = \mathbf{e}_i^T\mathbf{B}^T\mathbf{B}\mathbf{p}$$

wird

$$\partial/\partial p_i F = 2\mathbf{e}_i(-\mathbf{B}^T\mathbf{y} + \mathbf{B}^T\mathbf{B}\mathbf{p}) = 0,$$

daraus folgt:

$$\mathbf{B}^T\mathbf{B}\mathbf{p} = \mathbf{B}^T\mathbf{y}.$$

==========

Damit ist der Algorithmus für eine Spline-Glättung mit Datenreduzierung hergeleitet.

5.5 Die Parallelisierung der die Messdaten reduzierenden und glättenden Spline Approximation

Seien

$$\mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{m-1} \end{pmatrix}$$

die m Stützstellen (=Meßwerte) einer Funktion y. Sei ferner

$$s(x) = p_0 B_0(x) + p_1 B_1(x) + \dots + p_{n-1} B_{n-1}(x)$$

der approximierende Spline mit den Basisfunktionen

$$B_i(x) = \begin{cases} 0 \text{ für } x' \le 0 \text{ und } x' = (x - x_i)/(x_{i+1} - x_i) \\ x'^3/6 \text{ für } 0 < x' \le 1 \\ 1/6(-3x'^3 + 12x'^2 - 12x' + 4) \text{ für } 1 < x' \le 2 \\ B_i(4 - x') \text{ sonst .} \end{cases}$$

Dann müssen die Parameter p_i des Splines so gewählt werden, daß

$$\mathbf{B}^T * \mathbf{B} * \mathbf{p} = \mathbf{B}^T * \mathbf{y}, \quad \text{mit}$$

$$\mathbf{B} = \begin{pmatrix} B_0(x_0) & B_1(x_0) \dots & B_{n-1}(x_0) \\ B_0(x_1) & B_1(x_1) \dots & B_{n-1}(x_1) \\ \dots & & \\ & & \\ B_0(x_{m-1}) & B_1(x_{m-1}) \dots & B_{n-1}(x_{m-1}) \end{pmatrix}$$

$$\mathbf{p} = \begin{pmatrix} p_0 \\ p_1 \\ \dots \\ p_{n-1} \end{pmatrix}$$

Für n = m wird interpoliert (ohne Datenreduktion)
Für n < m wird approximiert, der Datenreduktionsfaktor ist n/m.

B ist eine Bandmatrix, deren Bandbreite von n und m abhängt, und zwar sind für ein gewisses x_i alle die Basissplines identisch null, die x_i nicht als Abszisse enthalten.

Damit zerfällt der Algorithmus der parallelen Spline-Approximation mit Datenreduktion in die folgenden Schritte:

1. Matrix **B** aufstellen. Dabei sind alle B_i außerhalb der Bandbreite der Matrix identisch null.
2. Matrix **B** transponieren.

3a. $\mathbf{B}^T$ mit dem Vektor **y** der Meßwerte multiplizieren

3b. Parallel dazu $\mathbf{B}^T$ mit **B** multiplizieren

4. Lineares Gleichungssystem lösen

Die Approximationsfunktion s ergibt sich dann zu:

$$\mathbf{s = B*p.}$$

Damit wurde das Problem der parallelen Spline-Approximation mit Datenreduktion zurückgeführt auf Elemente der linearen Algebra, die sich, wie im folgenden gezeigt wird, sehr gut parallelisieren lassen.

5.6 Einfluß der Topologie

In diesem Kapitel wird diskutiert, wie sich die vier Grundelemente der linearen Algebra

- Matrix transponieren
- Matrix mit Vektor multiplizieren
- Matrix mit Matrix multiplizieren
- lineares Gleichungssystem lösen

parallelisieren lassen. Es werden die Algorithmen und ihre Beschleunigungsfaktoren gegenüber sequentieller Verarbeitung für verschiedene Prozessorzahlen angegeben. Es zeigt sich, daß jeder Algorithmus eine ihm eigene optimale Topologie bezüglich der Ausführungszeit aufweist.

5.6.1 Matrix parallel transponieren

Sei **A** eine Matrix vom Typ [N,M] und a_{ij} ein Element von **A**. Weiterhin seien die Elemente a_{ij} im Speicher des Rechners zeilenweise abgespeichert:

kleine Adresse:	Inhalt
	a_{11}
	a_{12}
	...
	a_{1M}
	a_{21}
	a_{22}
	...
	a_{2M}
	...
	a_{N1}
	a_{N2}
	...
goße Adresse:	a_{NM}

Bild 5.5:
Zeilenweise abgespeicherte Matrix **A**.

Betrachtet man nun die Indizes i,j in ihrer binären Schreibweise und beschränkt man sich auf $N = 2^n$ und $M = 2^m$ (n,m > 0), so erhält man:

$$i = i_n i_{n-1},\ldots,i_1$$
$$j = j_m j_{m-1},\ldots,j_1, \text{ wobei}$$
i_i, j_k die Bits der Indizes i,j sind.

Es wird nun der "Zeiger P auf die Adressen aller Elemente a_{ij}" eingeführt:

$$P = i_n i_{n-1},\ldots,i_1 j_m j_{m-1},\ldots,j_1$$

Man erkennt, daß die höherwertigen Bits von P sich aus den Zeilenbits i_i, die niederwertigen aus den Spaltenbits zusammensetzen. Bei der transponierten Matrix $\mathbf{A}^T$, bei der die Zeilen mit den Spalten vertauscht sind, und die deshalb vom Typ [M,N] ist, sieht das Speicherabild folgendermaßen aus:

kleine Adresse:	Inhalt
	a_{11}
	a_{21}
	...
	a_{N1}
	a_{12}
	a_{22}
	.....
	a_{N2}
	...
	a_{1M}
	a_{2M}
	...
goße Adresse:	a_{NM}

Bild 5.5:
Spaltenweise abgespeicherte Matrix $\mathbf{A}^T$.

Wenn man nun den Zeiger P^T auf die Adressen der Elemente der transponierten Matrix betrachtet,

$$P^T = j_m j_{m-1},\ldots,j_1 i_n i_{n-1},\ldots,i_1,$$

stellt man fest, daß beide Zeiger gleich bis auf eine zyklische Rotation der Bits um n bzw. m Stellen nach rechts bzw. links sind.Dies führt zur Idee [Hoss], daß das Matrix Transponieren als ein Sortiervorgang zu betrachten ist, der in n bzw. m Schritten durchgeführt werden kann und auf alle Elemente der Matrix *gleichzeitig* angewendbar ist. Bild 5.7 zeigt für den Fall von A[4,4] wie die Transposition in $\log_2(4)$ Schritten abläuft:

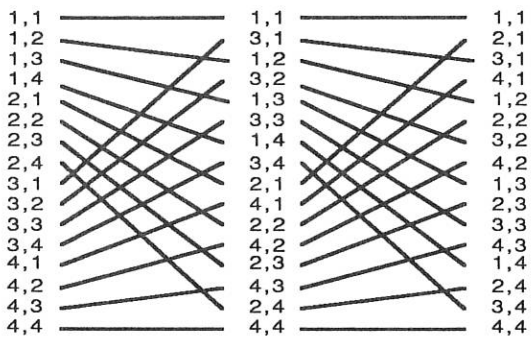


Bild 5.7:
Die Matrix **A** [4,4] wird in zwei Stufen durch Umsortieren der Elemente transponiert.

Entscheidend beim Umsortieren der Matrix ist, daß man die Pfade, entlang denen die Elemente der Matrix wandern, als die Verbindungen zwischen Prozessoren ansehen kann, so daß das Transponieren von Matrizen auf einen Datentransport zwischen Prozessoren, die gemäß der perfect- shuffle-Topologie verbunden sind, zurückgeführt wird.

Topologie der parallelen Matrixtransposition

Die Topologie der Verbindungen zwischen den Prozessoren ist die perfect shuffle Permutation s, die als die zyklische Rotation der Adressbits um eine Position nach links definiert ist:

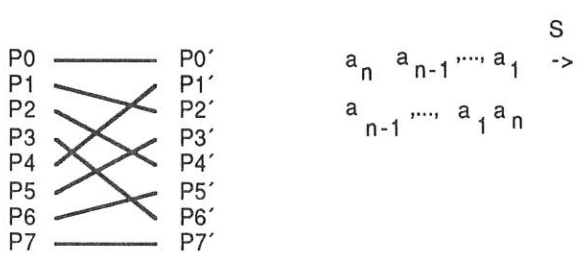


Bild 5.8:
Die für Matrizentransposition optimale Topologie ist die perfect shuffle Permutation.

Beschleunigungsfaktoren

Die vorgestellte parallele Matrixtransposition weist in ihrer *Shuffle*-Topologie eine Analogie zur Pease-FFT auf, die aus Un*shuffle*-Stufen zwischen den Butterfly-Operationen besteht. Und analog zur Pease-FFT kann der Signalflußgraph der Matrixtransposition zeilen- und/oder spaltenweise parallelisiert werden. Bei einer Matrix von NxM Elementen ergeben sich $NM*\log_2(N)$ Datentransporte.

Parallele Matrixtransposition $\mathbf{A}[N,M] \rightarrow \mathbf{A}^T[M,N]$

Zahl d. Prozessoren	Berechnungszeit	Topologie
1	O(NM*logM)	-
logM	O(NM) (pipelined)	Kette
N	O(M*logM)	shuffle
NMlogM	O(1) (pipelined)	", mehrfach hintereinander
N/k, k =1,2,...,N	O(kM*logM)	shuffle

Tabelle 1:
Ausführungszeiten und Topologien für verschiedene Prozessorzahlen bei der parallelen Matrixtransposition.

Diese Transfers können zwischen Zeilen oder Spalten von Prozessoren stattfinden. Bei zeilenweiser Parallelisierung sind die Prozessoren in einer Shuffle Topologie verbunden, so daß die Transfers zwischen Sender und Empfänger ohne Einschalten von Zwischenstufen stattfinden. Bei spaltenweiser Parallelisierung werden die Prozessoren linear in einer Kette miteinander verbunden. Um diese "pipeline" von Prozessoren nutzen zu können, ist ein kontinuierlicher Dateneingangsstrom erforderlich. In Tabelle 1 sind die sich ergebenden Größenordnungen für die Berechnungszeit der Matrixtransposition nach dem Shuffle-Verfahren aufgelistet.

5.6.2 Die parallele Matrix-Vektor-Multiplikation

Die Multiplikation einer Matrix [n,m] mit einem Vektor [m] oder mit einer Matrix [m,k] erfordert für jedes Ergebniselement die Berechnung eines Skalarprodukts. Alle Skalarprodukte können gleichzeitig berechnet werden da sie nicht voneineander abhängen. Im Falle der Matrix-Vektor-Multiplikation [n,m]*[m] können die m Skalarprodukte von bis zu bis zu n Prozessoren gleichzeitig berechnet werden. Bei der Matrix-Matrix-Multiplikation sind n*k Prozessoren einsetzbar. Der Grad der Parallelisierung kann dadurch erhöht werden, daß die Berechnung eines Skalarprodukts von mehr als einem Prozessor ausgeführt wird. Dies ist z.B. mit Hilfe der sog. Kaskadenaddition möglich, bei der für ein Skalarprodukt von m Summanden bis zu m/2 Prozessoren verwendet werden können. In Bild 5. 9 ist das Prinzip der Kaskadenaddition für p Summanden, die in log(p)

Schritten aufaddiert werden, dargestellt.

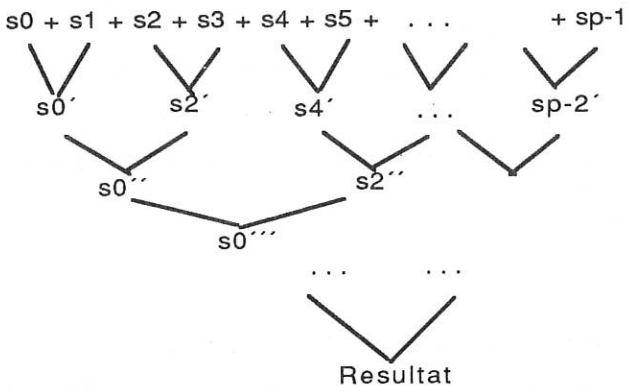


Bild 5.9:
Prinzip der Kaskadenaddition. Eine Summe aus p Summanden kann von p Prozessoren in *log(p) Schritten* berechnet werden, indem die Zahl der Summanden in jedem Schritt halbiert wird.

Bei den bisherigen Betrachtungen wurde die Zeit für die Ein/Ausgabe nicht berücksichtigt. Wie stark die Datenein/ausgabe ins Gewicht fallen kann, sieht man an dem Beispiel der Bildung eines Skalarprodukts, bei dem die Summation mit Hilfe der Kaskadenaddition durchgeführt wird. Dazu sei die Topologie der Prozessoren eine Kette, wie sie in Bild 5.10 dargestellt ist:

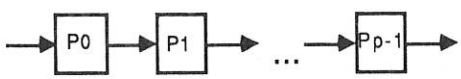


Bild 5.10:
Kette von Prozessoren zur Berechnung einer Skalarprodukts mit Kaskadenaddition.

Die Kette von Prozessoren werde von links mit Eingabedaten versorgt, und gebe auf der rechten Seite das Ergebnis aus. Zur Dateneingabe in die Kette sind bei einer Vektorlänge von n Elementen O(n) Zeitschritte erforderlich, da jedes Element einzeln in die Kette eingespeist werden muß. Danach kann das Skalarprodukt berechnet werden:

$$\sum_{i=0}^{n-1} a_i b_i = a_0 b_0 + a_1 b_1 + \ldots + a_{n-1} b_{n-1}$$

Dazu werden von allen Prozessoren gleichzeitig Teilskalarprodukte berechnet, so daß man nach dem 1. Schritt folgendes Zwischenergebnis erhält:

$$\sum_{i=0}^{p-1} a_i b_i = s_0 + s_1 + \ldots + s_{p-1}$$

Anschließend werden die Resultate mit Hilfe der Kaskadenaddition paarweise in log(p) Schritten aufaddiert. Der Abtransport des Resultats erfordert O(p) Schritte. Die Bilanz dieses Beispiels sieht so aus:

Datenein/ausgabe:	O(n) + O(p) Schritte
Datenverarbeitung:	O(1+log(p)) Schritte

Die Multiplikationen wurden dabei als *ein* Schritt gezählt. Die Datenein/Ausgabe dominiert also in diesem Beispiel. Das Beispiel läßt sich unter der Voraussetzung verallgemeinern, daß ein Datentrasnport unerläßlich ist. Er ist in den Fällen, in denen im Laufe mehrerer Berechnungen die Operanden in den Prozessoren bereits vorliegen, nicht notwendig. Ein Fall, bei dem der Datentransport *nicht* entfällt ist, wenn sequentielle Programme parallele Unterprogramme aufrufen. In diesem Fall müssen alle Ergebnisdaten der Unterprogramme nacheinander in den Prozessor transportiert werden, auf dem das sequentielle Programm läuft. Günstiger verläuft der Datentransfer der Ein/Ausgabe dann, wenn die Prozessoren nicht linear, sondern in einem Stern angeordnet sind. Diese Topologie ist in Bild 5.11 dargestellt. Der in Zentrum des Sterns sitzende Prozessor hat gleichen Zugriff auf alle anderen Prozessoren. Die Verwendung des Sterns als günstige Ein/Ausgabe-Topologie setzt aber voraus, daß zur *Verarbeitung* der Daten die Topologie umgeschaltet werden kann. Im Falle der Kaskadenaddition ist die Kette eine günstige Verarbeitungstopologie.

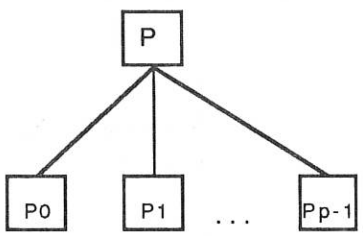


Bild 5.11:
Die für die Datenein/ausgabe günstigste Topologie ist der Stern. Der zentrale Prozessor an der Spitze muß bei voller Ausnützung aller p Prozessoren deren p fache Bandbreite besitzen.

Das Umschalten vom Stern zur Kette lohnt sich dann, wenn die Verarbeitungszeit bei der Kette inklusive der Umschaltzeit kürzer als die Verarbeitungszeit bei einer sternförmigen Topologie ist. Damit sich das Umschalten weiterhin lohnt muß der zentrale Ein/ausgabe-Prozessor genügend Ein/Ausgabe-Bandbreite aufweisen, um die Prozessoren in einem Stern mit Daten versorgen zu können. Unter diesen Voraussetzungen läßt sich die Größenordung der Berechnungsdauer für die Matrix-Vektor-Multitplikation für verschiedene Prozessorzahlen und Topologien in einer Tabelle 2 angeben. Eine andere Möglichkeit als das Umschalten der Topologie, ist die Datenein/ausgabe und Verarbeitung *überlappend* zu organisieren, wie es z.B. bei den systolischen Arrays geschieht. Um das Prinzip der überlappenden Verarbeitung zu demonstrieren, wird jetzt die Ausführung der Matrix-Matrix-Multiplikation mit Hilfe eines systolischen Arrays diskutiert.

Parallele Matrix-Vektor-Multiplikation

[m,n] * [n] -> [m]

Zahl d. Prozessoren	Zeitkomplexität	Topologie
1	O(mn)	-
m	O(n)	Kette
mn	O(logn) ohne I/O	m Ketten
mn	O(n) mit I/O	m Ketten
mn	O(logn) mit I/O	m Ketten + Stern f. I/O

Tabelle 2:
Ausführungszeiten und Topologien bei verschiedenen Prozessorzahlen bei der parallelen Matrix-Vektor-Multiplikation.

5.6.3 Die parallele Matrix-Matrix-Multiplikation in einem systolischen Array

Gegeben seien zwei Matrizen **A**[n,m], **B**[m,k], die miteinander multipliziert werden sollen so, daß

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \text{ , mit}$$

$$c_{ik} = \sum_{l=0}^{m-1} a_{il} * b_{lk} \text{ ist.}$$

D.h. für jedes Element c_{ik} aus **C** soll das Skalarprodukt aus dem i. Zeilenvektor von **A** und dem k.Spaltenvektor von **B** berechnet werden. Dazu ist die folgende Prozessor-Topologie günstig:

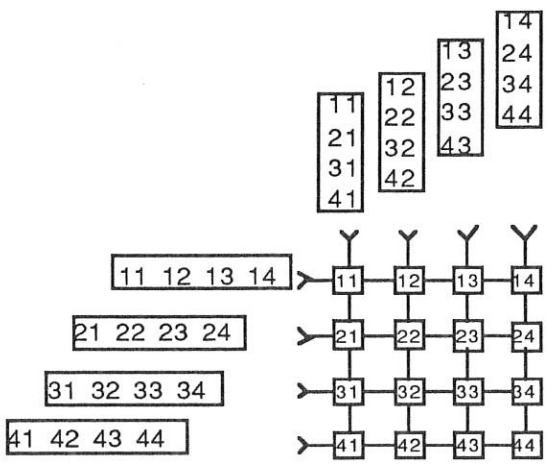


Bild 5.12:
Matrixmultiplikation mit Hilfe eines systolischen Arrays am Beispiel zweier [4,4] Matrizen.

Die Anordung von Prozessoren, wie Bild 5.12 sie zeigt, kann in einer sog. "systolischen", d.h. gepulsten, Betriebsweise bedient werden. Dies geschieht in der folgenden Art und Weise: zum Zeitpunkt T=0 werden in Prozessor 11 gleichzeitig die Elemente a_{14} und a_{41} eingespeist und dann miteinander multipliziert. Zur Zeit T=1 werden die Elemente a_{13} und a_{31} eingespeist, multipliziert und zu $a_{14}b_{41}$ addiert. Gleichzeitig werden die Operanden a_{14} und a_{41} an die Prozessoren 21 und 12 nach unten und links weitergeleitet, so daß diese dort rechtzeitig mit den Operanden a_{24} bzw. a_{42} multipliziert werden können. So wird die Matrix **A** zeilenweise und die Matrix **B** spaltenweise mit jedem Multiplizier-Akkumulier-Takt (=Puls) in das Prozessorfeld geschoben. Um die Laufzeiten der Operanden durch das Array zu eliminieren, müssen die Zeilen und Spalten von **A** und **B** zeitverschoben werden, so daß sich das oben dargestellte Bild 5.12 ergibt. Nach O(n) Schritten enthält jeder Prozessor ik genau das Skalarprodukt

$$c_{ik} = \sum_{l=0}^{m-1} a_{il} * b_{lk},$$

Die für die parallele Matrixmultiplikation ideale Topologie ist beim systolischen Prinzip eine gitterförmige Verbindung der Prozessoren (nearest neighbour). Für diese Topologie und für die Kette erhält man die in Tabelle 3 zusammengestellten Beschleunigungsfaktoren:

Matrix-Matrix-Multiplikation

A[n,m] * **B**[m,k] -> **C**[n,k]

Zahl d. Prozessoren	Zahl der Operationen	Topologie
1	O(nk)	-
n	O(km)	Kette
nk	O(m) systolisch	Gitter

Tabelle 3:
Ausführungszeiten und Topologien bei der parallelen Matrix-Matrix-Multiplikation.

5.6.4 Paralleles Lösen eines linearen Gleichungssystems

Es existieren viele Verfahren (direkte und iterative), ein lineares Gleichungssystem

$$\mathbf{A*x = b}$$

zu lösen [West]. Das bekannteste direkte unter ihnen ist das Gauss Verfahren (mit und ohne Pivotisierung), bei dem durch Äquivalenzumformungen des Gleichungssystems unterhalb der

Hauptdiagonalen Nullen erzeugt werden, und anschließend durch Rückwärtseinsetzen schrittweise nach den Unbekannten aufgelöst wird:

vorher:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \quad | * \; a21/a11$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$
$$\ldots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

Bild 5.13a:
Das Gauss Verfahren zur Lösung eines linearen Gleichungssystems.

nachher:

$$a'_{11}x_1 + a'_{12}x_2 + \ldots + a'_{1n}x_n = b'_1$$
$$a'_{22}x_2 + \ldots + a'_{2n}x_n = b'_2$$
$$\ldots$$
$$a'_{nn}x_n = b'_n$$

Bild 5.13b
Das Gauss Verfahren transformiert eine Matrix in eine obere Dreiecksgestalt.

Das Gauss Verfahren läßt sich derart parallelisieren, daß gleichzeitig in allen Zeilen unterhalb der Hauptdiagonalen Nullen erzeugt werden. Dazu können bis zu n-1 Prozessoren in jeder Zeile für die Grundoperation

$$a'_{ij} = a_{ij} - (a_{kj}/a_{kk})a_{jk} \qquad \text{für alle i,j,k mit } k < i,\ j \geq k$$

verwendet werden. Insgesamt bis zu $(n-1)^2$ Prozessoren in n-1 Zeilen. Der Nachteil dieses Verfahrens liegt darin, daß die Zahl der zu erzeugenden Nullen stetig abnimmt, je größer der Spaltenindex wird, so daß mit fortschreitendem Index immer weniger Prozessoren beschäftigt sind. Dieser Nachteil wird dadurch behoben, daß sowohl unterhalb als auch oberhalb der Hauptdiagonalen Nullen erzeugt werden, so daß die Summe aller Nullen bei wachsendem Zeilenindex konstant bleibt. Man bezeichnet diese Variante als Gauss-Jordan Verfahren [Hoss]. Es ist in Bild 5.14 dargestellt. Beim Gauss-Jordan Verfahren entfällt das "Rückwärtseinsetzen " zur Ermittlung der Unbekannten x_1 bis x_n, das aufgrund seiner rekursiven Abhängigkeit die Zeitkomplexität O(n) aufweist. Hinzu kommt bei diesem Verfahren, daß in n Spalten oberhalb der Hauptdiagonalen Nullen zu erzeugen sind, so daß in der Summe bei *beiden Verfahren dieselbe Zahl von Operationen auszuführen sind.* Das Gauss-Jordan-Verfahren ist aber besser parallelisierbar, da das Erzeugen der Nullen oberhalb der Hauptdiagonalen keine zusätzliche Zeit kostet. Es lassen

sich die erwarteten Beschleunigungswerte gemäß Tabelle 4 angeben.

$$
\begin{array}{lll}
a'_{11}x_1 & 0 & = b'_1 \\
a'_{22}x_2 & & = b'_2 \\
\ldots & & \\
0 & a'_{nn}x_n & = b'_n
\end{array}
$$

Bild 5.14:
Das Gauss-Jordan Verfahren eignet sich sehr gut zur Parallelisierung, da die Zahl der zu erzeugenden Nullen in jeder Spalte konstant ist.

In Tabelle 4 wird das Zeitverhalten von Gauss- und Gauss-Jordan-Verfahren gegenüber gestellt:

Paralleles Lösen eines linearen Gleichungssystems
[n,n] nach Gauss und nach Gauss-Jordan

Zahl d. Prozessoren	Zeitabhängigkeit		Topologie
	Gauss	Gauss-Jordan	
1	$O(n^3)$	$O(n^3)$	-
n	$O(n^2+n)$	$O(n^2)$	Kette *
n^2	$O(n+n)$	$O(n)$	Gitter

* Bei zeilenweiser oder spaltenweiser Parallelisierung

Tabelle 4:
Zeitkomplexität beim parallelen Lösen eines linearen Gleichungssystems nach dem Gauss- und dem Gauss-Jordan-Verfahren bei verschiedenen Prozessorzahlen und Topologien.

Zusammenfassung:

Das Beispiel der parallelen Lösung eines linearen Gleichungssystems zeigt, daß zwei Algorithmen, die sequentiell gleichwertig sind, da sie die gleiche Zahl von Operationen aufweisen, sich hinsichtlich ihrer parallelen Ausführung unterscheiden können.

Anhand der Matrix-Vektor-Multiplikation wurde deutlich, daß für die Ein/Ausgabe von manchen Algorithmen eine größere Zeit notwendig ist, als für deren parallele Verarbeitung. Ein Umschalten zwischen Ein/Ausgabe-Topologie und Verarbeitungstopologie kann die Zeitverhältnisse umkehren und ist deshalb sinnvoll. Voraussetzung ist, daß die Topologien "schnell" umgeschaltet wird.

Die Matrix-Matrix.Multiplikation und die Matrix-Transposition führte vor Augen, daß es für parallele Algorithmen "günstige" Topologien von Prozessorverbindungen gibt, die von

Algorithmus zu Algorithmus verschieden sein können. Die Shuffle-Topologie ist für die Matrix-Transposition günstig, während für die Matrix-Matrix-Multiplikation die nearest-neighbour Topologie günstig ist.

Wie ein größeres numerisches Problem parallelisiert werden kann, wurde am Beispiel der Spline-Approximation gezeigt. Das für dieses Problem hergeleitete Verfahren verwendet nur Elemente aus der linearen Algebra, wie Matrix-Matrix- und Matrix-Vektor-Multiplikationen, und ist somit gut parallelisierbar.

Es haben sich aus der Betrachtung der parallelen Spline-Approximation wie auch der Fourier-Transformation zwei Anforderungen an die Architektur eines Parallelrechners ergeben:

1. Die Möglichkeit, die Topologie der Prozessorverbindungen der Topologie des Problems anpassen zu können.

2. Die Möglichkeit *innerhalb eines Programms* schnell die Topologie wechseln zu können.

Die Architektur von MULTITOP trägt mit seiner *dynamisch variablen Topologie* diesen Anforderungen Rechnung.

Die schnelle Fouriertransformation wurde in der Sprache OCCAM auf MULTITOP für einen, zwei und vier Prozessoren implementiert. Das Verfahren der parallelen Spline- Approximation wurde in einer Implementierung für einen Prozessor anhand zahlreicher Beispiele getestet.

Im Anhang zu dieser Arbeit sind die Grundlagen der schnellen Fourier-Transformation und der Spline-Approximation dargestellt, sowie ein Literaturverzeichnis enthalten. Im Zusatzteil zu dieser Ausarbeitung finden sich die Dokumentation der Schaltung von MULTITOP einschließlich der PAL- und der OCCAM-Programme.

6. Zusammenfassung und Ausblick

Was wurde erreicht?

Am Beispiel der parallelen schnellen Fourier-Transformation (FFT) und einer Spline-Approximation wurde der Einfluß der Interprozessor-Kommunikation auf den Wirkungsgrad der Programmausführung untersucht. Dabei zeigte es sich, daß die Kommunikation zwischen den Prozessoren die Effizienz *wesentlich* beeinflußt.

Daraus ergeben sich Konsequenzen sowohl für die Wahl der untersuchten parallelen Algorithmen als auch für die Architektur des Multiprozessor-Systems, das diese Algorithmen optimal unterstützen soll:

1. Für eine hohe Effizienz der Programmausführung ist, bei mehreren möglichen Lösungen, derjenige Algorithmus mit der kleinsten Interprozessor-Kommunikation durch eine Datenfluß-Analyse zu bestimmen. Eine solche Analyse wurde exemplarisch für die verschiedenen Formen der FFT vorgenommen und daraus zwei Formen mit minimaler Kommunikation bestimmt.

2. Weiterhin ist neben einer Minimierung der Daten*menge* auch die *Zeit* für den Transport von Daten zwischen einem Sender (Prozessor) und einem Empfänger zu minimieren. Dazu müssen Daten *direkt*, d.h. ohne Einschalten von Zwischenstufen, vom Erzeuger zum Verbraucher geleitet werden können.

Für eine kleine Zahl von Prozessoren ist dies mit Hilfe einer vollständig vermaschten Topologie von Verbindungen möglich, so daß von jedem Sender zu jedem Empfänger ein Weg führt. Da bei N Prozessoren die Zahl der Verbindungen proportional zu N^2 wächst, ist aus Kostengründen wie aus technischen Gegebenheiten dieses Verfahren begrenzt.

Da aber nie alle N^2 Verbindungen *gleichzeitig* aktiv sind, kann ihre Zahl dadurch verkleinert werden, daß *schaltbare* Punkt-zu-Punkt-Verbindungen eingeführt werden, die *bei Bedarf* einen Weg zwischen einem Sender und einem Empfänger herstellen. Dies bewirkt, daß die Topologie der Prozessorverbindungen, ähnlich wie bei einer Regelung, dynamisch, d.h. *während* eines Programmablaufs, der Topologie des Problems nachgeführt wird.

Die Realisierung einer dynamisch variablen Topologie kann nicht durch einen Kreuzschienenverteiler erfolgen, der genauso wie der vollständig vermaschte Graph einen Aufwand von $O(N^2)$ erfordert. Deshalb wurde bei MULTITOP ein sog. Koppelnetz verwendet, das mit O(NlogN) Schaltern auskommt. Zu diesem Zweck wurde ein für alle Permutationen blockierungsfreies Netz sowie ein Verfahren zum Setzen der Schalter des Netzes hergeleitet, das darüberhinaus den Vorzug der Modularität bietet.

Die Modularität ist Voraussetzung für eine leichte Erweiterbarkeit der Zahl der Ein- und Ausgänge des Netzes. So kann unter Verwendung der Teile eines bestehenden MULTITOP-Netzes bestimmter Größe die Zahl der Eingänge leicht verdoppelt oder vervierfacht usw. werden, weil keine Änderungen am bestehenden Netz vorzunehmen sind. Dies erlaubt ein in seiner Leistung skalierbares Multiprozessor-System, weil eine beliebige Zahl von Prozessoren gekoppelt werden kann.

Beim Durchschalten einer neuen Verbindung einsteht ein Zeitverlust durch das Setzen der Schalter des Netzes. Ein MULTITOP-Netz mit z.B. 16 Eingängen kann in ca. 10 µs gesetzt werden, was eine kurze Zeit im Vergleich zur Ausführungszeit eines Programms ist, die mindestens im ms-Bereich liegt. Das schnelle Umkonfigurieren ist also eine wichtige Eigenschaft des MULTITOP-Rechners.

Beim Prototypen dieses Rechners werden vier *Transputer* mit einer Leistung von 4x2 Mio. Gleitkomma-Operationen pro Sekunde als Rechnerkerne eingesetzt. Die Interprozessor-Kommunikation geschieht über geschaltete bidirektionale und bitserielle Links, die mit 10-20 MBit/s arbeiten.

Für die Ausführungszeit der parallelen FFT z.B. ergab sich ein Wirkungsgrad von über 90% bei vier Prozessoren. Zum Vergleich wurden die Wirkungsgrade auf einer (festen) Hypercube-Topologie und einem Ring berechnet, die, abhängig von der Zahl der eingesetzten Prozessoren, deutlich niedriger lagen und bei bei 512 Prozessoren z.B. noch ca. 27% (Hypercube) bzw. 2% (Ring) betrugen, während die Effizienz bei MULTITOP auf 61% extrapoliert wurde.

Zusammenfassend kann gesagt werden, daß die variable Topologie sich hinsichtlich der effizienten parallelen Ausführung der untersuchten Algorithmen bewährt hat. Diese Tatsache läßt sich auf andere Verfahren aus der Numerik übertragen, bei denen die Interprozessor-Kommunikation einen wesentlichen Anteil an der Programmausführung hat. Der relativ hohe Aufwand, der mit einer variablen Topologie verbunden ist, kann durch den Einsatz kundenspezifischer Schaltkreise (ASICs) reduziert werden, so daß dieses Gegenargument entkräftet ist. Zusätzlich erwies sich das Koppelnetz als wertvoll beim Entwurf paralleler Programme sowie bei deren Test, da das Netz zum eine eine hohe Flexibilität bei der Auswahl der Algorithmen garantiert, und da es zum anderen leicht möglich ist, die vom Netz transportierten Botschaften in diesem zu Testzwecken zwischenzuspeichern.

Ausblick

Als Ausblick sollen die verbleibenden Probleme und ihre möglichen Lösungen betrachtet werden. Es bleibt insbesondere auf den Gebieten der Rechner, der Sprachen und der Algorithmen zu nennen:

1. Auf dem Gebiet der Rechner ist es notwendig, ein wesentlich größeres System als bisher mit mindestens 32 Prozessoren aufzubauen, um den Abfall des Wirkungsgrades bei hohen Prozessorzahlen zu untersuchen. (Der Prototyp von MULTITOP erlaubt bis zu 8 Prozessoren zu koppeln).

2. Auf dem Gebiet der Sprachen sind selbst parallelisierende Compiler vorstellbar, die die Partionierung der bei SIMD-Anwendungen auftretenden Vektoren automatisch vornehmen. Außerdem könnte der Sprachumfang paralleler Sprachen in der Art erweitert werden, daß nicht nur Vektoren sondern auch Operatoren zu deren Handhabung implementiert werden, wie dies bei einigen sequentiellen Sprache bereits der Fall ist. Weiterhin sind bei MIMD-Anwendungen Sprachen wünschenswert, die es gestatten, die einem Problem innewohnende Parallelität besser auszudrücken, so daß Fehler wie "gegenseitige Verklemmung" (Deadlock) oder "Nicht-Terminierung" (Livelock) weniger oft auftreten bzw. früher erkannt werden.

3. Auf dem Gebiet der Algorithmen ist festzustellen, daß bisherige parallele Algorithmen auf einigen wenigen, allgemeinen Topologien von Prozessorverbindungen, wie z.B. einem Stern, Baum oder Hypercube beruhen. Mit einer dynamisch variablen Topologien ist es möglich, sehr spezielle Topologien zu wählen, was sich auf die Art der verwendeten Algorithmen auswirkt. Es insbesondere eine neue Klasse von Algorithmen denkbar, die sich aus sehr vielen Teilalgorithmen verschiedener Topologien zusammensetzen.

Insgesamt lassen sich Parallelrechner sicher als ein Gebiet bezeichnen, dem in Zukunft mehr Bedeutung zukommen wird.

Anhang

A.1 Die Realisierung des MULTITOP-Rechners

A.1.1 Die zwei wesentlichen Komponenten von MULTITOP

Das *Koppelnetz* und die *Transputer* sind die zwei wesentlichen Komponenten des Rechners. Das Koppelnetz ist ein Netzwerk, das es gestattet, Ein- und Ausgänge des Netzes beliebig miteinander zu verbinden. Ein Koppelnetz gibt einem also die Möglichkeit, eine variable Topologie der Prozessorverbindungen zu realisieren. Wie bereits ausgeführt wurde, unterscheidet man Koppelnetze mit Paket- und Leitungsvermittlung. Bei Koppelnetzen mit Leitungsvermittlung muß dem Datenpaket keine Zieladresse vorangestellt werden, da das Ziel explizit festliegt. Für kurze Paketlängen spart dies Zeit, da die Übertragung der Zieladresse entfällt. Bei den Kanälen des Transputers besteht ein Paket aus 11 Bits. Um in einem größeren Koppelnetz von z.B. 256 Ausgängen *einen* Ausgang des Netzes zu adressieren, müssten den 11 Paketbits 8 Adressbits vorangestellt werden, deren Übertragung die ursprüngliche Übertragungszeit um 72 % erhöhen würde. Um diese zusätzliche Zeit zu sparen, wurde das Prinzip der Leitungsvermittlung verwendet. Diese Wahl der Datenübertragung hat Konsequenzen bezüglich der Verbindungsfähigkeit der Prozessoren untereinander: Geht man von einer festen Zahl von Kanälen pro Prozessor, z.B. beim Transputer von vier aus, können damit höchstens vier andere Prozessoren gleichzeitig über das Koppelnetz verbunden werden. Wenn n die Zahl der Prozessoren, 2k die Zahl der Anschlüsse pro Prozessor (k Ein- und k Ausgänge) ist, dann können maximal

$$N = nk$$

Kanäle gleichzeitig aktiv sein. Die Zahl der potentiellen Verbindungen dagegen beträgt

$$v = (n/2)(n - 1),$$

wenn jeder Prozessor mit jedem verbunden ist. Dies wird als vollständige Vermaschung bezeichnet. Der dazu gehörende Graph ist in Bild A.1.1 für den Fall n = 8 dargestellt:

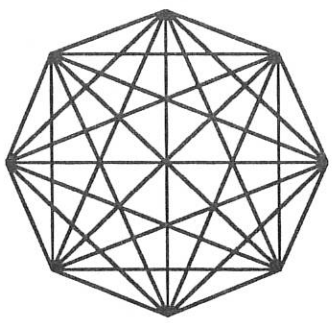


Bild A.1.1:
Die maximale Zahl von Verbindungen enthält der vollständig vermaschte Graph, der hier für n = 8 Prozessoren dargestellt ist.

Im Falle der Transputer ist jede Verbindung zwischen zwei Prozessoren bidirektional. Aus diesem Grunde ist die Zahl V der möglichen Kanäle doppelt so groß wie die Zahl v der möglichen

Verbindungen.

$$V = 2v.$$

Ist die Zahl der möglichen Kanäle größer als die Zahl der vorhandenen (V > N), muß das V/N-fache an Verbindungen durch Systemprogramme emuliert werden: Es sind nie alle V Verbindungen gleichzeitig aktiv. Man kann deshalb zwischen V virtuellen und N physikalischen Kanälen unterscheiden, ähnlich wie dies in einem virtuellen Speichersystem mit virtuellen und physikalischen Seiten geschieht. Analog zur "Seitenersetzung" beim Speicherzugriff auf eine physikalisch nicht vorhandene Speicherseite (page) muß eine "Kanalersetzung" dann erfolgen, wenn ein virtueller Kanal angesprochen wird, der kein reales Gegenstück hat. Das Verfahren, nach dem die Kanalersetzung vorgenommem wird, kann z.B die Verdrängung durch Alterung sein (least recently used). Die Aufgabe der "Kanalersetzung" wird von einem Systemprogramm vorgenommen.

Mit dem Konzept der virtuellen Kanäle sind somit alle möglichen Topologien realisierbar.

Anders als bei einem virtuellen Speichersystem herrscht beim virtuellen Kanalsystem *nicht* das sog. Lokalitätsprinzip, das besagt, daß die Adressen der Programmausführung sich so ändern, daß ständige Seitenersetzungen selten sind. Vielmehr können die Topologien der Prozessor-Verbindungen z.B. von einem Gitter zu einem Stern und wieder zurück "springen". Aber auch unabhängig von der Größe der Änderungen sind bei einem Koppelnetz für *jeden neuen Kanal alle Schalter des Netzes* zu setzen. Ein schneller Zugriff auf die Schalter ist deshalb wichtig. Aus diesem Grunde ist ein virtuelles Kanalsystem nur dann praktikabel, wenn das Koppelnetz schnell umgeschaltet werden kann. Erleichternd kommt allerdings hinzu, daß eine einmal eingestellte Topologie normalerweise "lange" benützt werden kann. Dies entspricht bei einem virtuellen Speichersystem der Tatsache, daß ein physikalisch vorhandene Seite i.A. solange im Speicher bleibt, bis alle Adressen dieser Seite durchlaufen sind. Bei den im Rahmen dieser Arbeit untersuchten Anwendungen aus der parallelen Numerik zeigte es sich, daß ein "Springen" der Topologien weniger bei der Verarbeitung sondern eher bei der schnellen Ein- und Ausgabe von Daten auftritt. Das Verteilen von Eingabewerten und das Sammeln von Ergebnisdaten erfordert, "reihum" einen Kanal von jedem Prozessor zu einem zentralen Ein-/Ausgabegerät zu schalten.

Neben dem Koppelnetz ist die zweite wesentliche Komponente von MULTITOP der sog. Transputer der Fa. INMOS [IMS]. Der Transputer T800 ist ein 32 Bit Mikroprozessor, der *auf dem Chip* neben einem schnellen Gleitkomma-Rechenwerk und einem Schreib/Lese-Speicher von 4 Kbytes vier serielle Hochgeschwindigkeitskanäle integriert hat. Sein Blockschaltbild ist in Bild A.1.2 dargestellt. Das Gleitkomma-Rechenwerk des T800 leistet ca. zwei Millionen Operationen pro Sekunde. Seine 32 Bit Zentraleinheit (CPU) basiert auf einer sogenannten "RISC"-Architektur. Er erreicht damit einen Durchsatz von bis zu 10 Millionen einfacher Instruktionen pro Sekunde. Die bidirektionalen Kanäle arbeiten mit je 20 Mbit/s über vier "Direct memory access" (DMA)-Steuerungen unabhängig voneinander und von der CPU und können ganze Folgen von Bytes selbstständig übertragen. Der Speicher hat eine Zykluszeit von 50 ns.

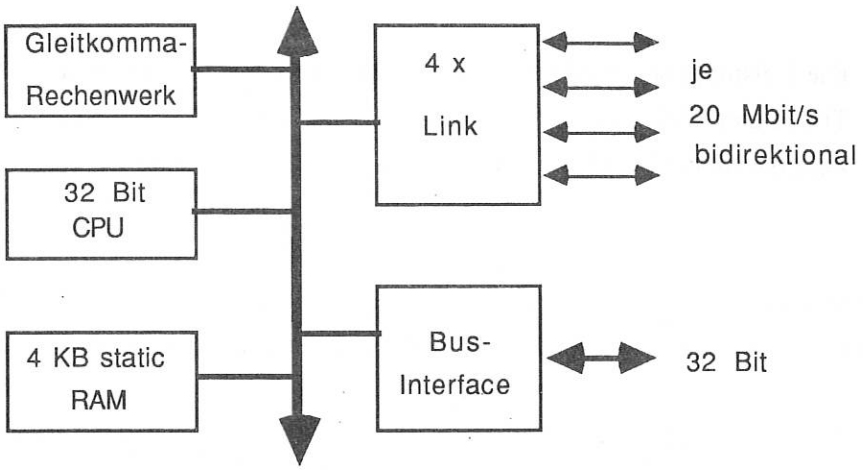


Bild A.1.2:
Das Blockschaltbild des Transputers T800. Dieser Baustein ist eine wesentliche Komponente von MULTITOP.

Der Transputer weist auch Nachteile auf: Wegen seiner Begrenzung von vier Links pro Pozessor kann damit z.B. maximal ein vierdimensionaler Hypercube realisiert werden. Dieser Nachteil ist aber über ein programmierbares Koppelnetz zu beheben.

Wegen des Transputers basiert bei MULTITOP die Kommunikation auf Kanälen und Botschaftenaustausch. Ein Kanal ist eine spezielle Hardware-Einrichtung zur Kommunikation, die beim Transputer als Link bezeichnet wird. Das Transputer-Link ist ein bitserieller Hochgeschwindigkeitskanal, der über DMA (Direct Memory Access) bedient wird. Mit Links werden Punkt-zu-Punkt-Verbindungen zwischen je zwei Prozessoren aufgebaut. Wichtig ist nun, daß die Zahl der Links, und damit die Kommunikationskapazität des Rechners, linear mit der Zahl der Prozessoren wächst, da jeder Prozessor mit einer festen Zahl von Links versehen ist. Bei MULTITOP ist die *nutzbare* Bandbreite ebenfalls linear mit der Zahl der Prozessoren, da über das programmierbare Koppelnetz jeder Prozessor direkt mit jedem verbunden werden kann, so daß keine Kommunikations-Kapazität durch Zwischenstufen verloren geht. Schließlich existiert auf jedem Link ein Protokoll, das eine synchrone Kommunikation a priori sicherstellt. Denn, ähnlich einem Rendez-Vous, findet auf einem Link dann und nur dann ein Botschaftenaustausch statt, wenn *beide* Partner der Punkt-zu-Punkt-Verbindung dazu bereit sind. Dies ist schaltungstechnisch so implementiert, daß für jedes gesendete Byte eine Rückmeldung vom Empfänger kommen muß, daß er bereit für ein nächstes Byte ist. Weil zur Kommunikation keine gemeinsamen Variablen verwendet werden, entfällt auch das Problem der Synchronisierung bei Mehrfachzugriff. Die Implementierung der Kommunikation über Kanäle durch Transputerlinks stellt somit eine schnelle und sichere Interprozessor-Kommunikation dar. Zwei weitere Gründe sprechen für den Einsatz von Transputern im MULTITOP Rechner:

1. Das MULTITOP zugrunde liegende Kommunikationsmodell kann einfach mit Hilfe der vier Links realisiert werden. Da die Links seriell sind, ist der Aufwand für das Koppelnetz geringer als bei parallelen Kanälen.

2. Die Leistungsfähigkeit der Kommunikation ist wegen der geringen "Setup"-Zeit der Transputer-Links, die ca. 4 µs pro Datenpaket beträgt, hoch. (Die "Setup"-Zeit ist die Zeit, die für die Initialisierung eines Datentransfers erforderlich ist.)

Das folgende lineare Modell zur Übertragung eines Datenpakets von n Bytes auf einem Link zeigt, daß die "Setup"-Zeit ein entscheidendes Maß für die Kommunikationsleistung ist:

T(n) sei die Zeit zur Übertragung von n Bytes, T_S die "Setup"-Zeit der Datenübertragung und T_B die Zeit, um ein Byte zu übertragen. Für T(n) gilt:

$$T(n) = T_S + nT_B.$$

Daraus erhält man für die Datenübertragungs- Geschwindigkeit $v_{DAT}(n)$:

$$v_{DAT}(n) = \frac{n}{T(n)} = \frac{n}{T_S + n\,T_B}$$

Der Graph dieser Funktion ist in Bild A.1.3 dargestellt.

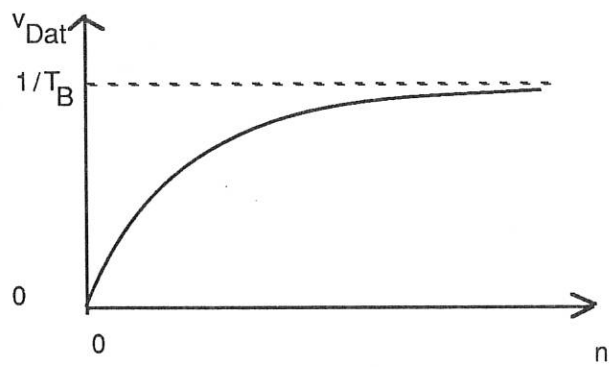


Bild A.1.3:
Die effektive Geschwindigkeit v_{DAT} der Datenübertragung bei einem linearen Kommunikations-Modell in Abhängigkeit von der Paketlänge n. Durch die Setup-Zeit bei der Kommunikation wird die maximale Geschwindigkeit erst bei größeren Paketlängen erreicht.

Man sieht, daß die effektive Geschwindigkeit v_{Dat} nur für eine ausreichend große Paketlänge der physikalisch möglichen Übertragungsgeschwindigkeit $1/T_B$ nahekommt. Als quantitatives Maß für "ausreichend große Paketlänge" kann z.B. die Länge gewählt werden, bei der *die halbe maximale* Geschwindigkeit $v_{1/2}$ erreicht wird. Für diese Länge gilt:

$$n_{1/2} = T_S / T_B.$$

$n_{1/2}$ gibt also an, *ab welcher Paketlänge die Kommunikation effektiv wird.*

Beim Transputer ist $T_S \approx 4$ µs, und $T_B = 550$ ns. Daraus ergibt sich ein $n_{1/2}$ von:

$$n_{1/2} = 7{,}27 \text{ Bytes.}$$

Ein kleines $n_{1/2}$ ist für ein Kommunikationssystem, das auf Botschaftenaustausch basiert, wichtig. Wegen des sehr kleinen $n_{1/2}$ von MULTITOP (nur zwei Transputer-Worte!) können die Programme auf diesem Rechner hoch parallelisiert werden, ohne daß die Effektivität der Kommunikation darunter leidet.

In einem weiteren Punkt hebt sich MULTITOP klar von anderen Rechnern ab: die Geschwindigkeit, mit der die Topologie umgeschaltet werden kann, ist sehr hoch. Bei MULTITOP liegt die Umschaltzeit bei ca. 10 µs für die zur Übersetzungszeit des Programms bekannten Topologien. Damit kann eine besonders *fein abgestufte* Anpassung der Topologie des Rechners an

die Topologie des Problems erfolgen. So kann man nicht nur für die drei Grundphasen der Datenverarbeitung

Eingabe - Verarbeitung - Ausgabe

die jeweils optimale Topologie eingestellen, sondern Wechsel von Programm zu Programm oder sogar von Programmphase zu Programmphase sind möglich. Die Umschaltzeit soll als die *Strukturzykluszeit* der dynamisch variablen Topologie bezeichnet werden. Die Strukturzykluszeit ist also ein Maß für die *Flexibilität* eines Rechners mit dynamisch variabler Topologie.

A.1.2 Der Aufbau von MULTITOP

Der Rechner besteht aus den drei Grundbausteinen:

1. Prozessorplatine
2. Koppelnetzplatine
3. Synchronisierungsplatine

Aufgrund des *modularen* Konzepts kann jede Platine auch mehrfach vorhanden sein. Dadurch ist eine Skalierung der Rechenleistung möglich, bei der sich die Größe des Systems nach dem jeweiligen Leistungsbedarf richtet. Prozessor- und Synchronisierungsplatine sind *linear* skalierbar. Die Zahl der Koppelnetzplatinen dagegen ist für das nächstgrößere (doppelt so große) Netz jeweils *zu verdoppeln.*, wobei noch eine hier nicht näher dargestellte Mischer-Einheit hinzukommt, die zwei gleich große Netze miteinander verbindet. Daß ein Netz mit doppelt so vielen Eingängen durch Kopplung zweier kleinerer Netze gewonnen werden kann, ist der Grund dafür, daß die

Rechenleistung skalierbar wird. Das Konzept der Kopplung von zwei kleineren Netzen, die aufgrund der *Selbstähnlichkeit* der Topologie des Netzes möglich ist, wird in Kapitel 3 hergeleitet. Hier wird auf den technischen Aufbau der drei Platinen und deren Funktion in der Architektur von MULTITOP eingegangen werden. Einen Überblick über den Aufbau gibt Bild A.1.4.

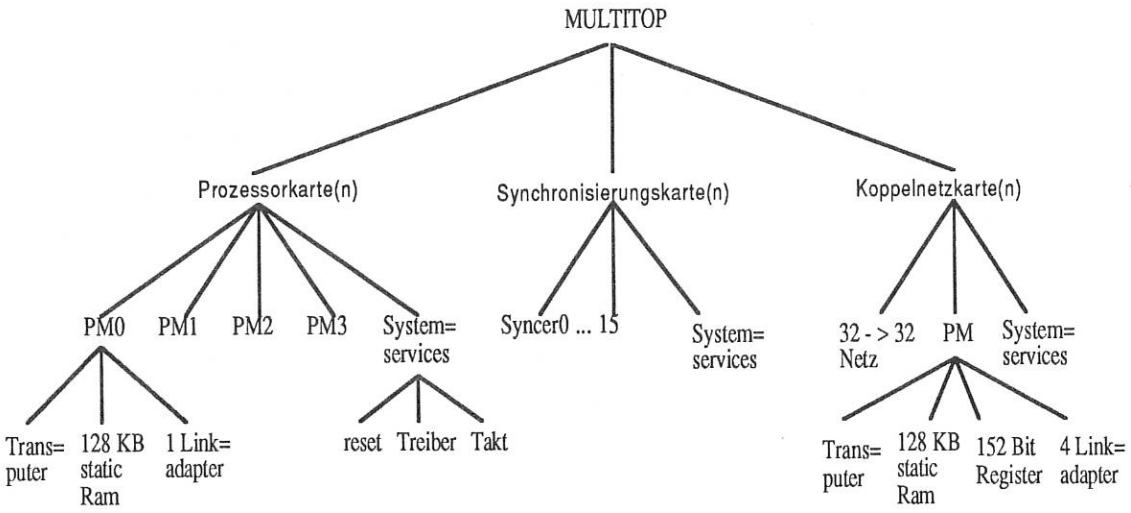


Bild A.1.4:
Der Aufbau von MULTITOP. Der Rechner besteht aus den drei Einheiten Prozessorplatine, Koppelnetzplatine und Synchronisierungsplatine. Jede Platine kann mehrfach existieren, so daß die Rechenleistung des Systems einstellbar wird. PM steht für Prozessor-Speicher-Modul.

In diesem Diagramm sind die wesentlichen Bestandteile und ihre Komponenten hierarchisch dargestellt. Die Abkürzung PM steht für Prozessor-Speicher-Modul. Auf der Prozessorplatine sind vier solcher Moduln untergebracht. Alle Transputerlinks werden auf einem Stecker am Rand der Prozessorplatine herausgeführt. Die Synchronisierungsplatine enthält 16 identische Synchronisierer für die 4x4 Links einer Prozessorkarte. Die Koppelnetzplatine schließlich hat 32 Ein- und Ausgänge und kann somit zwei Synchronisierungs- und zwei Prozessorplatinen bedienen. Die physikalische Plazierung der in Bild A.1.4 aufgelisteten Komponenten ist im Bild A.1.5 dargestellt. Alle drei Platinen sind über ein Flachbandkabel mit einzeln verdrillten Adern miteinander verbunden. Der P1-Stecker des VMEbus wird in der zur Zeit existierenden Version zur Stromversorung verwendet. Die Aufgaben von Prozessorplatine bzw. Koppelnetzplatine sind:

1. Bereitstellen von Prozessorleistung
2. Realisierung der dynamisch variablen Topologie

Die Synchronisierungsplatine hat die Aufgaben:

1. Abtasten der asynchronen Linksignale, um diese taktsynchron in das Koppelnetz einspeisen zu können;

2. Zwischenspeichern der Daten- und Rückmeldepakete, während der

Umkonfigurierung des Koppelnetzes;

3. Auslesen gespeicherter Daten gleichzeitig zum Einlesen neuer Daten nach dem Umschalten des Netzes.

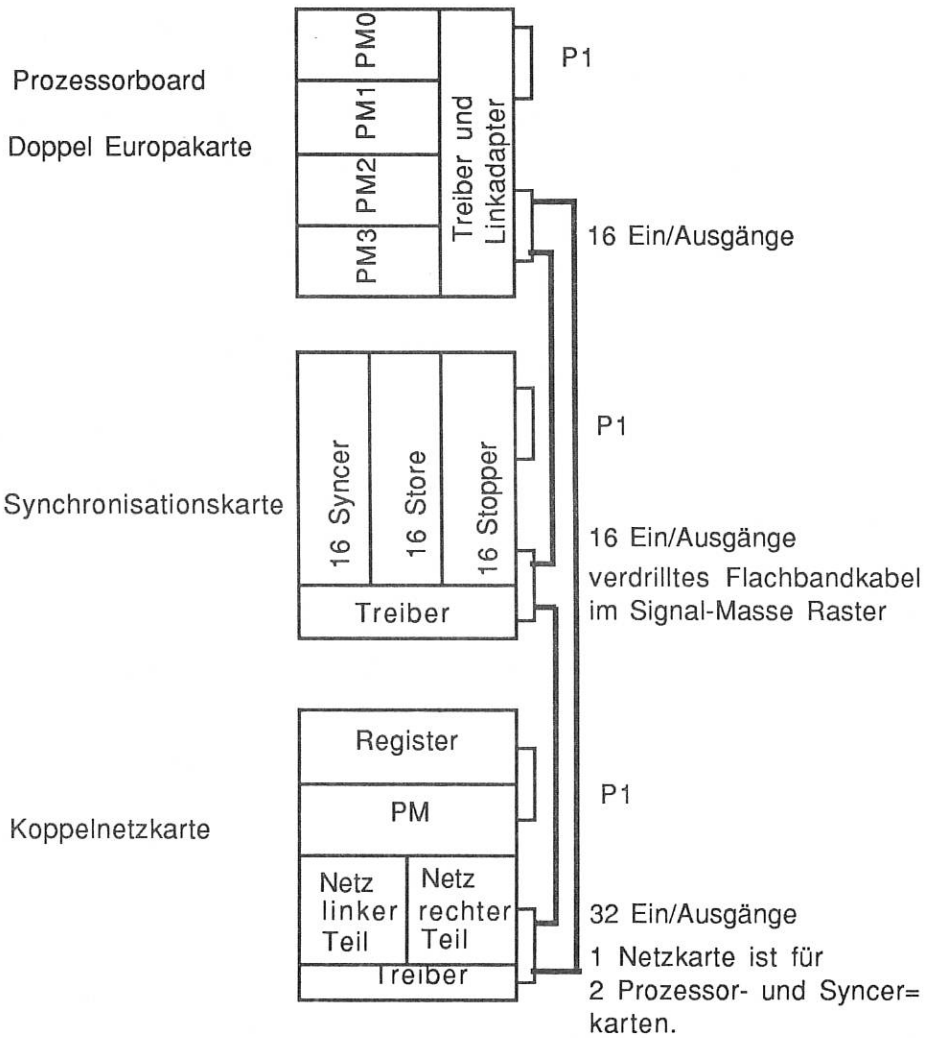


Bild A.1.5:
Das physikalische Layout von MULTITOP. Der Rechner besteht aus mindestens drei Doppeleuropa-Karten, die die Prozessor-Speicher-Module, die Synchronisierer und das rekonfigurierbare Koppelnetz enthalten. Mit P1 wird der VMEbus-Anschluß bezeichnet, der hier nur zur Stromversorgung dient.

Die oben dargestellten Aufgaben ergeben sich zum einen aus der Notwendigkeit, die Transputerlinks hardwaremäßig vor dem Umschalten des Netzes zu synchronisieren und zum anderen aus den Eigenschaften des verwendeten Netzes selbst. Wie in dem Kapitel über das modulare Netz ausgeführt wird, gehört das Netz zur Klasse der Netze, die durch Umordnen interner Wege blockierungsfrei sind. Das heißt, daß für einen neuen Weg durch das Netz u.U. *alle* bestehenden Wege intern neu gelegt werden müssen. Wenn sich während des Umschaltens ein Paket im Netz befände, ginge es den falschen Weg. Aus diesem Grund darf während des Umschaltens kein Paket im Netz sein, d.h. das Netz muß "leer" sein. Es gibt nun zwei

Möglichkeiten, dies sicherzustellen:

1. Softwaremäßige Synchronisation: Jeder Prozessor sendet z.B. ein Signal zu einer zentralen Instanz, daß er inaktiv, d.h. bereit zum Umschalten ist. Sobald alle Prozessoren bereit sind, wird umgeschaltet.
2. Hardwaremäßige Synchronisation: Das Netz kann zu beliebigen Zeiten umgeschaltet werden. Gerade im Netz befindliche Pakete werden noch bis zum Ausgang des Netzes transportiert, dann wird umgeschaltet. Neue Pakete werden während dieser Zeit nicht in das Netz eingespeist sondern zwischengespeichert.

Die softwaremäßige Synchronisation hat die Nachteile:

1. Der Programmierer muß explizit in sein Programm Synchronisationspunkte einführen.

2. Durch das Warten auf die Bereitschaft aller Prozessoren, entsteht ein Zeitverlust, der bis zur Blockade aller Prozessoren durch einen einzigen gehen kann, da gewartet werden muß bis auch der letzte inaktiv ist.

Die hardwaremäßige Synchronisation hat den Nachteil, daß der Schaltungsaufwand zu ihrer Realisierung sehr hoch ist. So muß für jeden seriellen Kanal ein ausreichender Speicher während der Umschaltzeit zur Verfügung stehen. Glücklicherweise existiert auf den Transputerlinks ein Protokoll [IMS], das für jedes gesendete Byte eine Rückmeldung vom Empfänger vorschreibt. Solange keine Rückmeldung kommt, wird kein neues Byte gesendet. Dies begrenzt die notwendige Speichertiefe pro Kanal auf ein Daten- und ein Rückmeldepaket. Da nur die acht Bits des Datenpakets und zwei weitere Flag-Bits gespeichert werden müssen, reicht die Kapazität *eines* PALs mit 10 Flipflops für diesen Speicher aus. Dieser Umstand, der die technische Realisierung der hardwaremäßigen Synchronisation erst praktikabel macht, und der Vorteil, daß dann für den Benutzer die Umkonfiguration transparent ist, war der Grund, bei MULTITOP hardwaremäßig zu synchronisieren. Die Synchronisierungsplatine trägt also wesentlich zum Komfort und zur Effizienz beim Umschalten bei. Die hardwaremäßige Synchronisierung warf aber zugleich elektrotechnische Probleme auf: Jede Prozessorplatine hat einen eigenen Taktgenerator, da die Verteilung eines zentralen Takts über ein großes System aufgrund der Signallaufzeiten schwierig ist. Die Prozessorplatinen weisen also bezüglich der Phasenlage ihres Taktes Differenzen untereinander und zum Takt der Koppelnetzplatine auf. Es ist deshalb notwendig, vor dem Einspeisen der Linksignale in das Koppelnetz, diese abzutasten und gleichphasig zu machen. Um einen sicheren Betrieb des Rechners zu garantieren, muß man mit einer Fehlerwahrscheinlichkeit von weniger als 10^{-14} abtasten. Weiterhin sind Daten, die während der Umschaltphase des Netzes eintreffen, zwischenzuspeichern und nach beendetem Umschalten auszulesen, während gleichzeitig neue Signale eingelesen werden können müssen. In Abschnitt 2.6 sind diese Probleme und ihre Bewältigung im einzelnen dargestellt. Die folgende Aufstellung gibt eine tabellarische Übersicht über die einzelnen Bestandteile des Rechners.

1. Prozessorkarte

Diese Karte enthält vier unabhängige Module, von denen jedes besteht aus:

- o Transputer T414-15, T414-20 oder T800-20
- o statischer Speicher a 128 KBytes mit 330 ns Zykluszeit bei 15 MHz Transputer bzw. 250 ns bei 20 MHz Transputer.
- o 4 bidirektionale serielle Kanäle mit 10 oder 20 Mbit/s Übertragungsrate.
- o Linkadapter mit 10 oder 20 Mbit/s Übertragungsrate
- o max. 2 MFLOPS Gleitkomma-Rechenleistung bei einfach genauer Arithmetik (nur T800) und max. 10 MIPS Durchsatz.

Allen Moduln gemeinsam sind:

- o 2x(16+4) Leitungstreiber für die Kommunikationskanäle (bidirektional)
- o Taktoszillator (40 MHz); reset, error und analyse Schaltung.

2. Synchronisierungskarte.

- o 16 Synchronisierer, Speicher und Ablaufsteuerung in CMOS UV erasable PAL Technik
- o voll duplex Betrieb der Kanäle
- o 10 MBit/s pro Link Übertragungsrate.
- o Einzelschrittsteuerung der Linkspeicher (Jeder der 16 Linkspeicher kann gezielt in das Netz eingespeist werden).
- o 2x16 Leitungstreiber, 40 MHz Oszillator und reset.

3. Koppelnetzkarte

- o Koppelnetz in CMOS UV erasable PAL Technik mit 32 Ein-/Ausgängen und 152 Bit Speicher für die Schalterstellungen. 10 oder 20 MBit/s Transportgeschwindigkeit durch das Netz.
- o Transputer T414-20 mit 128 KBytes RAM.
- o 4x Linkadapter mit 10 oder 20 Mbit/s.
- o 2x(16+4) Leitungstreiber , Oszillator, reset, error und analyse logik.
- o OCCAM I Programm zum Setzen der Koppelnetzschalter in 40 µs. (In OCCAM II sind 10us möglich).
- o OCCAM I Programm zum Berechnen einer neuen Topologie und Setzen der Schalter in 4 ms.

Stromversorgung: 2,5 A bei +5 V (CMOS Technik)

Tabelle A.1.1:

Zusammenstellung der Einzelteile von MULTITOP und ihre qualitative und quantitative Spezifikation.

Auf allen drei Platinen befinden sich Leitungstreiber für die Links, die über ein Flachbandkabel mit einzeln verdrillten Adern von der Prozessorplatine zur Synchronisierkarte und von dort zum

Koppelnetz und wieder zurück zur Prozessorplatine geführt werden. Die Verkabelung ist also nicht wie bei einem Bus, sondern zirkular. Weiterhin befindet sich auf allen drei Platinen ein Taktgenerator und eine "power-up-reset"-Schaltung, damit jede Platine für sich getestet werden kann. Im gemeinsamen Betrieb haben die Synchronisierungskarte und das Koppelnetz *einen* Takt. Die Linkadapter der Prozessorplatine sind mit den vier Linkadaptern der Koppelnetzplatine verbunden, die an den 32 Bit-Bus des Transputers angeschlossen sind. Diese Konfiguration von Linkadaptern ergibt eine *sternförmige* Topologie, die der variablen Topologie der Prozessorverbindungen überlagert ist. Die dadurch gebildeten "fünften Links" an jedem Transputer dienen zur Steuerung der Prozessor-Speicher-Module, denn es können vom Transputer im Zentrum des Sterns "Broadcast"-Funktionen an alle ausgübt werden. Wenn alle 32 Eingänge in das Koppelnetz verwendet werden, sind 8 Transputer an das Netz angeschlossen. Um einen vollständigen Stern daraus bilden zu können, sind auf der Koppelnetzplatine 8 Linkadapter notwendig. Aus Platzgründen konnten nur vier untergebracht werden, so daß bei voller Bestückung eine "Clusterbildung" von mehreren Transputern zu einer Einheit notwendig wird, um Linkadapteranschlüsse zu sparen.

Bild A.1.6 zeigt den Aufbau von MULTITOP aus logischer Sicht. Es wird der *allgemeine Fall* von N Rechen- und M Koppelnetzsetz-Transputern dargestellt, wie er sich aus einem hochskalierten MULTITOP-Rechner ergibt.

Als Host wird ein IBM-PC verwendet, als weitere Schnittstelle zur Außenwelt ein VMEbus-Interface. Im unteren Teil des Bildes sind zwei weitere Komponenten eingezeichnet:

1. ein festes Verbindungsnetzwerk
2. die das Koppelnetz setzenden Transputer T0´-T(M-1)´.

Die Aufgabe dieser beiden Teile ist die folgende:
Das programmierbare Koppelnetz besteht im wesentlichen aus Schaltern und Drähten zwischen den Schaltern. Um einen Weg durch das Netz zu legen, müssen die Schalter die entsprechende Stellung haben. Zum Berechnen der Schalterstellungen und zum Setzen wird eine zweite Klasse von Transputern verwendet: T0´-T(M-1)´. Sie erledigen diese Aufgabe parallel, um nicht einen sequentiellen Engpass zu bilden, der dem Prinzip der Skalierbarkeit widerspräche. Da ihre Aufgabe fest ist, können sie auch fest miteinander verbunden werden. Wenn zur Laufzeit keine schnelle Berechnung der Schalterstellung notwendig ist, können diese Transputer *bis auf einen entfallen.*, da zum *Berechnen und Setzen der Schalter nminimal ein Transputer erforderlich ist.* Der Aufbau von MULTITOP kann für den allgemeinen Fall von N Rechen- und M Koppelnetzsetz-Transputern so zusammengefaßt werden:

Das Rückgrat des Rechners bildet ein festes Verbindungsnetzwerk, auf das sich ein variables Netz aufbaut. Je nach Art der Parallelisierung des Algorithmus, der die Schalterstellungen des variablen Netzes berechnet, kann das feste Netz als Kette oder Gitter ausgeführt werden. Es verbindet die das Koppelnetz setzenden Transputer miteinander. Diese legen einen Weg zwischen zwei Rechentransputern durch das variable Netz. Da keine Zwischenstufen über andere

Rechentransputer eingeschaltet werden, ist der Datentransport minimiert. Über den Rechentransputern steht der Host, der ihnen ihr Programm lädt, und das VMEbus-Interface, das eine zweite Schnittstelle zur Außenwelt bildet. *Beim Prototypen von MULTITOP ist N = 4 und M= 1*. Dieser Fall ist in Bild A.1.7 dargestellt.

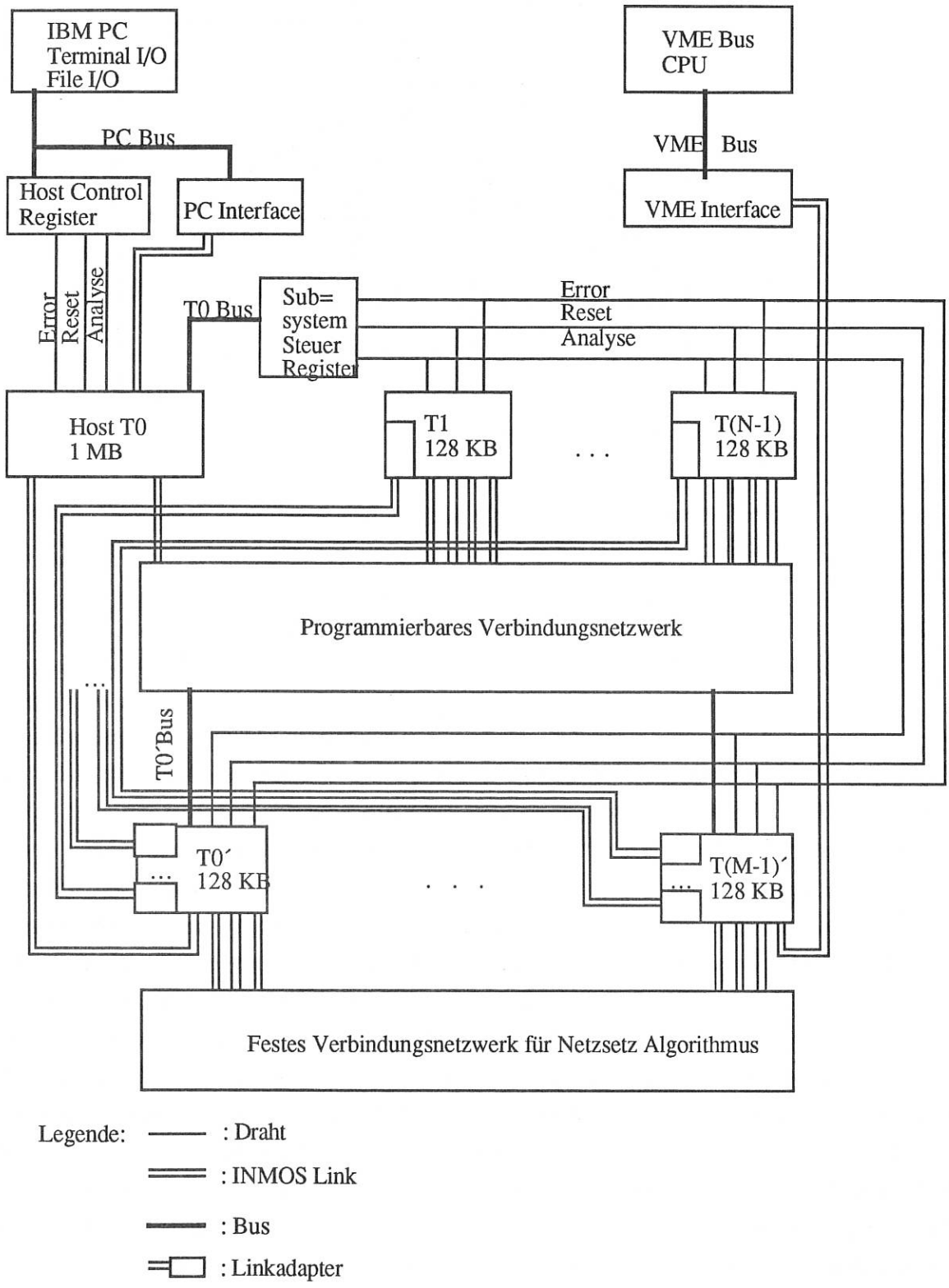
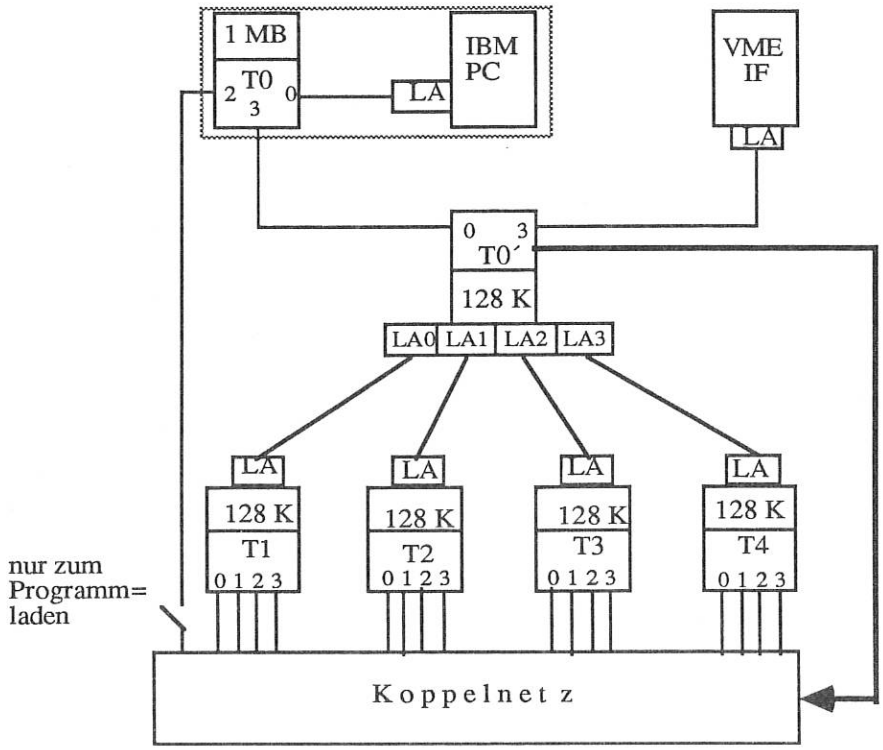


Bild A.1.6:
Prinzipschaltbild von MULTITOP für den allgemeinen Fall von N Rechentransputern und M Koppelnetzsetz-Transputer.



Legende:
LA : Linkadapter

Bild A.1.7:
MULTITOP aus der Sicht des Benutzers. Der variablen Topologie ist ein Stern überlagert, in dessen Zentrum der Koppelnetzsetz-Transputer ist. Dieser Transputer übernimmt auch die Datenein- und -ausgabe. Der Transputer T0 ist auf einer käuflichen Platine im IBM-PC untergebracht.

Im Zentrum des Sterns steht der Koppelnetzsetz-Transputer. Aufgrund der sternförmigen Topologie, die der variablen Topologie überlagert ist, kann der Koppelnetz-Transputer eine weitere Funktion erfüllen: Er übernimmt die Verteilung und Sammlung der Daten bei Ein- und Ausgabe mit Hilfe seiner Linkadapter. Weiterhin senden die Rechentransputer alle Meldungen, die auf dem Bildschirm des IBM-PC angezeigt werden sollen, an ihn. Dieser serialisiert die Botschaften und bringt sie zur Anzeige auf dem Schirm des PC. Dieser Vorgang wird als Multiprozessor-Bildschirmanzeige bezeichnet. Sie ist für den Test paralleler Programme unerläßlich. Ein anderes Testhilfsmittel für parallele Programmierung ist die sog. Einzelschritt-Betriebsweise des Koppelnetzes. Damit ist gemeint, daß ein Testhilfe-Programm *einen Eingang* in das Netz ausgewählt, der Daten in das Netz einspeisen kann. Alle anderen Eingänge sind dabei blockiert. Das Netz nimmt die Daten dieses Eingangs Paket für Paket auf und hält nach jedem Paket an, so daß das Testhilfe-Programm das Paket lesen und auf dem Bildschirm anzeigen kann. Die Kommunikation des zu testenden Programms läuft mit diesem Hilfsmittel in einzelnen Schritten ab, die so langsam aufeinander folgen, daß sie beobachtet werden können. Die Einzelpaket-Steuerung gibt einem so die Kontrolle über den Verkehr auf dem Links. Insbesondere kann damit eindeutig festgestellt werden:

1. Welchen Inhalt ein gesendetes Paket hat.

2. Wer der Absender und wer der Empfänger eines Pakets ist.
3. Wann ein Paket abgeschickt wird.

Die Einzelpaket-Steuerung hilft, die Fragen, die bei einem auf Botschaftenaustausch basierenden Kommunikationssystem auftreten können, zu beantworten.

Die Programme für MULTITOP wurden in OCCAM geschrieben. OCCAM ist eine strukturierte Programmiersprache, die PASCAL oder C ähnlich ist. Es fehlen darin allerdings strukturierte Datentypen und Rekursion. Dafür existieren neue Sprachelemente, wie das PAR oder ALT Schlüsselwort, das für parallele Programmierung unerläßlich ist. Die Links der Transputer sind in den Sprachumfang von OCCAM eingebettet. So wird eine Eingabe auf einem *Kanal a* in eine*Variablen x* so codiert:

a ? x

Die Ausgabe wird geschrieben als

a ! x

OCCAM ist eine Implementierung von CSP [HOAR], die von der Fa. INMOS für die Transputer vorgenommen wurde. CSP ist ein mathematisches Kalkül mit weitreichenden Möglichkeiten. So kann damit z.B. exakt *bewiesen* werden, ob ein paralleles Programm seine Spezifikation erfüllt oder nicht. Ein solcher mathematischer Beweis wird als Verifikation des Programms bezeichnet. Programme, die in OCCAM geschrieben sind, haben den Vorteil, daß sie leichter in CSP verifiziert werden können als Programme in anderen Sprachen.

Zusammenfassend kann über die Architektur von MULTITOP gesagt werden, daß sie die folgenden wesentlichen Merkmale aufweist:

1. Die Verwendung von *Transputern* als Rechnerkerne, und damit zusammenhängend das auf *Botschaftenaustausch basierende Kommunikationssystem* mit geringer Setup-Zeit.

2. Die Verwendung eines *modularen Koppelnetzes* , das eine hohe Flexibilität bei der parallelen Programmierung und eine Skalierung der Rechenleistung im Baukastenstil ermöglicht.

3. Die *schnelle Umschaltung* zwischen den Topolgien, die die Effizienz bei der Ausführung paralleler Programme erhöht.

A.1.3 Die Synchronisierungseinrichtung

Die Synchronisierungseinrichtung stellt das komplizierteste Teil des Rechners dar. Sie hat die Aufgaben:

1. Abtasten der asynchronen Linksignale, damit diese taktsynchron in das Koppelnetz eingespeist werden können.

2. Speichern der gesendeten Daten- und Rückmeldepakete, während der Umschaltphase des Netzes.

3. Auslesen von gespeicherten Daten nach dem Umschalten, während neue Daten eingelesen werden.

Diesen drei Aufgaben entsprechend besteht die Synchronisierungskarte aus drei funktionalen Einheiten, die für jedes Link vorhanden sind:

1. Abtaster
2. Speicher
3. Ein-/Auslese-Automat

Bei jeder funktionalen Einheit traten besondere Probleme auf, deren Lösung hier dargestellt wird.

Abtaster

Das zuverlässige Abtasten der Links hat sich als ein besonderes Problem erwiesen, da die Daten *asynchron* gesendet werden, und die Datenübertragungs-Geschwindigkeit mit 10 MBit/s rel. hoch ist. Aufgrund der Tatsache, daß jedes Link autonom arbeitet, werden zusätzliche Forderungen an die Zuverlässigkeit des Abtasters gestellt: wenn alle 16 Links gleichzeitig arbeiten, multipliziert sich die Fehlerwahrscheinlichkeit mit der Zahl der Links. Um im Mittel einen störungsfreien Betrieb über mindestens einen Tag zu gewährleisten, ist es notwendig, daß nur alle $1{,}4*10^{13}$ Bits ein Bitfehler auftritt ($16*10^7$ Bits/s * 3600s/h * 24h = $1{,}4 * 10^{13}$).

Zusätzlich mußten beim Design des Abtaster noch die folgenden Faktoren beachtet werden:

1. Aus funktionalen Erwägungen wie aus Platzgründen müssen alle 16 Synchronisierungsschaltungen auf *einer Platine* untergebracht werden.

2. Das 10 Mbit/s Signal muß nach dem Abtasttheorem mit *mindestens 20 MHz* abgetastet werden.

3. Bei einem Abtastflipflop können *metastabile Zustände* auftreten, wenn durch das asynchrone Eingangssignal die Setup-Zeit nicht eingehalten werden kann.

Das erste Problem (Packungsdichte) wurde durch Verwendung der PAL-Technik gelöst. Der PAL Baustein CYPRESS 22V10 enthält ca. 1200 Gatteräquivalente. Er ist in CMOS-Technologie

lieferbar und kann durch Einstrahlen von UV-Licht gelöscht werden. Aufgrund dieser Vorzüge wurde er für alle Funktionen des Synchronisierers verwendet.

Das zweite Problem (hohe Abtastrate) wurde auf die folgende Weise gelöst:
Das Abtast-PAL wird mit 20 MHz getaktet. Dieser Wert liegt an der Grenze der für das PAL möglichen Geschwindigkeit. Um eine höhere zeitliche Auflösung, als durch den Takt vorgegeben, zu erreichen, wurde das Eingangssignal *und ein um eine Gatterlaufzeit (=25 ns) verzögertes Eingangssignal* abgetastet. Zum *Erkennen des Startbits* im Signal wird das nichtverzögerte Signal verwendet,verarbeitet dagegen wird das verzögerte Signal. Durch diesen Trick wird erreicht, daß das verzögerte Signal 25 oder 75 ns nach seiner ansteigenden Flanke sicher abgetastet werden kann. Die Abtastung erfüllt die Setup-Zeit des Abtastflipflops mit 25 ns und die Hold-Zeit des Flipflops mit ebenfalls 25 ns. Das daraus resultierende Zeitverhalten ist in Bild A.1.8 dargestellt.

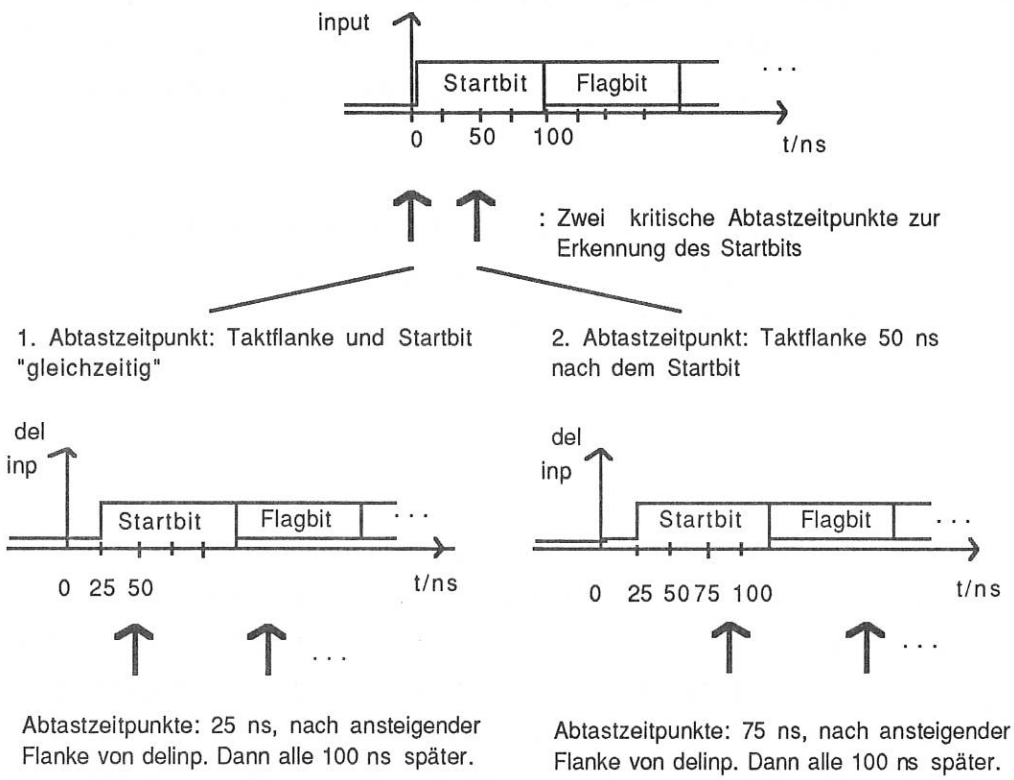


Bild A.1.8:
Das Zeitdiagramm von input und delinp. Erkannt wird das Signal input, abgetastet das um 25 ns verzögerte input Signal (delinp). Dadurch wird entweder 25 oder 75 ns nach der steigenden Flanke des Signals abgetastet. Zu diesen Zeitpunkten findet sicher keine Signaländerung statt, so daß die Abtastung sicher ist.

Das dritte Problem (metastabiler Zustand) des Abtastflipflops wurde durch einen *endlichen Automaten* gelöst. Im Ablaufplan des Automaten ist das Auftretens eines metastabilen Zustands von bis zu 50 ns Dauer mit einberechnet. Weiterhin wurde zur Minimierung möglicher metastabiler Zustände eine besondere Zustandscodierung des Automaten angewandt: Die zwei Zustände des Automaten "Warten auf das Startbit" und "Starbit erkannt" werden mit den Zahlen 000 und 001 codiert. Damit entscheidet nur *ein einziges Flipflop* des Automaten, ob ein Startbit vorliegt oder nicht. Würde der Automat diese zwei Zustände z.B. als 001 und 010 codieren, würden *zwei* Flipflops ihren Zustand beim Erkennen des Startbits ändern müssen. Aufgrund winzigster

Unterschiede in den Signallaufzeiten dieser Flipflops, die auch dann bestehen, wenn sie physikalisch auf demselben Chip sind, kommt es bei dieser Codierung ab und zu zu unterschiedlichen Beurteilungen des Eingangssignals. Es wurde gemessen, daß dieser Fall durchschnittlich alle 10^4 Pakete eintritt. Deshalb wurde eine Codierung der Zustände mit "single bit transition" (000 -> 001) verwendet. Erschwerend kommt hinzu, daß das Abtastflipflop dann einen metastabilen Zustand annehmen kann, wenn die Setup-Zeit des Flipflops wegen des asynchronen Auftretens des Startbits nicht eingehalten werden kann [AMD]. Der kritische Zeitbereich dafür beträgt zwischen 5 und 50 ps. Das Kennzeichen eines metastabilen Zustands ist, daß man nicht weiß, wie lange er anhält. Er endet aber spätenstens mit der nächsten Takflanke des Flip Flops, vorausgesetzt, das Eingangssignal ist dann stabil. Spätestens einen Takt (50 ns) später wird das Startbit als stabil erkannt. Ein metastabiler Zustand bewirkt also die zeitliche Unsicherheit von einer Taktperiode. Wenn dies beim Abtasten des Bitstroms einkalkuliert wird, werden die *Datenbits* sicher abgetastet. Aufgrund der Unsicherheit einer Taktperiode wird das um eine halbe Taktperiode verzögerte Eingangssignal entweder 25 oder 75 ns nach der steigenden Flanke des Startbits erkannt. Zu diesen Zeitpunkten ändert es sich nicht, so daß kein metastabiler Zustand auftreten kann.

Das Resultat all dieser Überlegungen schlug sich in der Konzeption eines *endlichen Automaten* enthalten, der in einem PAL Baustein Platz findet. Das Bild A.1.9 zeigt das Flußdiagramm dieses Automaten und die Codierung seiner Zustände in binärer Form.

Speicher

Die Transputerlinks sind bidirektionale und autonome funktionale Einheiten. Auf einem Link können deshalb spontan *ein* Daten- und *ein* Rückmeldepaket auftreten. Weitere Datenpakete können erst nach entsprechenden Rückmeldungen gesendet werden. Während der Umschaltphase des Netzes steht das Netz für neue Pakete nicht zur Verfügung, so daß eine Speichermöglichkeit vorgesehen werden muß. Das Datenpaket besteht aus 11 und das Rückmeldepaket aus 2 Bit. Leider gibt es kein serielles 13 Bit TTL-Schieberegister mit kleinem Gehäuse zur Speicherung dieser Daten. Auch bei PALs sind 10 Flipflops pro Baustein das Maximum. Das Problem war gelöst, als die genaue Zahl der auf einem Link möglichen Zustände während der Umschaltphase des Netzes ermittelt war. Die Zustände sind:

1. Das Link war in der Umschaltphase inaktiv.
2. Es wurde ein Datenpaket gesendet
3. Es wurde ein Rückmeldepaket gesendet.
4. Es wurde ein Daten-*und* Rückmeldepaket gesendet. Die genaue Reihenfolge der Pakete ist irrelevant, da die Pakete unabhängig voneinander sind.

Diese vier Zustände können in 2 Bits codiert werden. Die Zahl der zu speichernden Bits beträgt somit 8 + 2 = 10 Bits. Dies ist genau die Größe, die in einem PAL-Baustein maximal Platz findet.

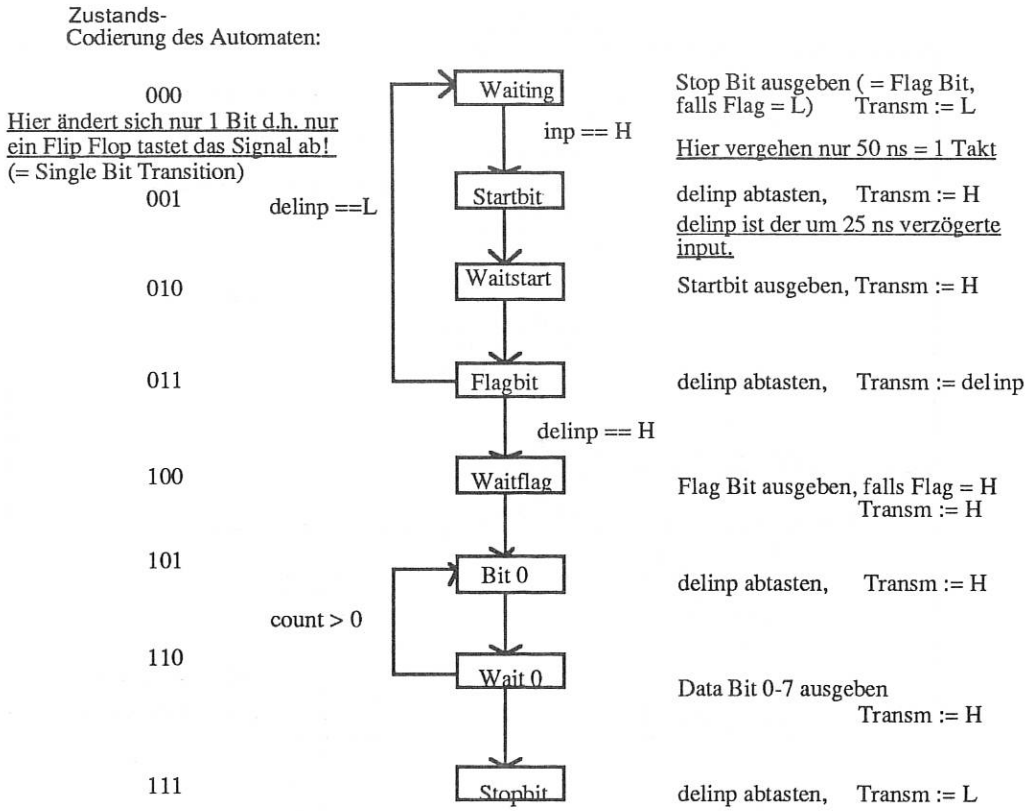


Bild A.1.9:
Flußdiagramm des Abtasters. Diese Funktion kann inklusive des Bit-Zählers, der die Abbruchbedingung "count = 0" der Schleife liefert, in einem PAL-Baustein mit sechs Flipflops untergebracht werden.

Ein-/Auslese-Automat

Die Aufgabe dieser funktionalen Einheit ist, daß während des Umschaltens des Netzes die maximal zwei möglichen Pakete auf dem Link zwischengespeichert und nach der Umschaltphase ein eventuell gefüllter Linkspeicher wieder korrekt ausgelesen wird. Ein Problem tritt für den Fall auf, daß während des Umschaltens *ein* Paket gespeichert wurde. Nach dem Umschalten wird dieses wieder ausgelesen. Es kann aber *während* des Auslesens des ersten Pakets ein zweites gesendet werden! Dies muß, damit es nicht verloren geht, ebenfalls zwischengespeichert werden, während das erste Paket in das Koppelnetz ausgelesen wird. Zur Steuerung dieser Ein-/Auslese-Prozedur ist ein komplexer Automat erforderlich, der *ohne Verzögerung* auf eine äußere Unterbrechung reagieren muß, sobald ein neues Datenpaket eintrifft. Bild A.1.10 zeigt das Flußdiagramm des endlichen Automaten, der 32 verschiedene Zustände annehmen hat.

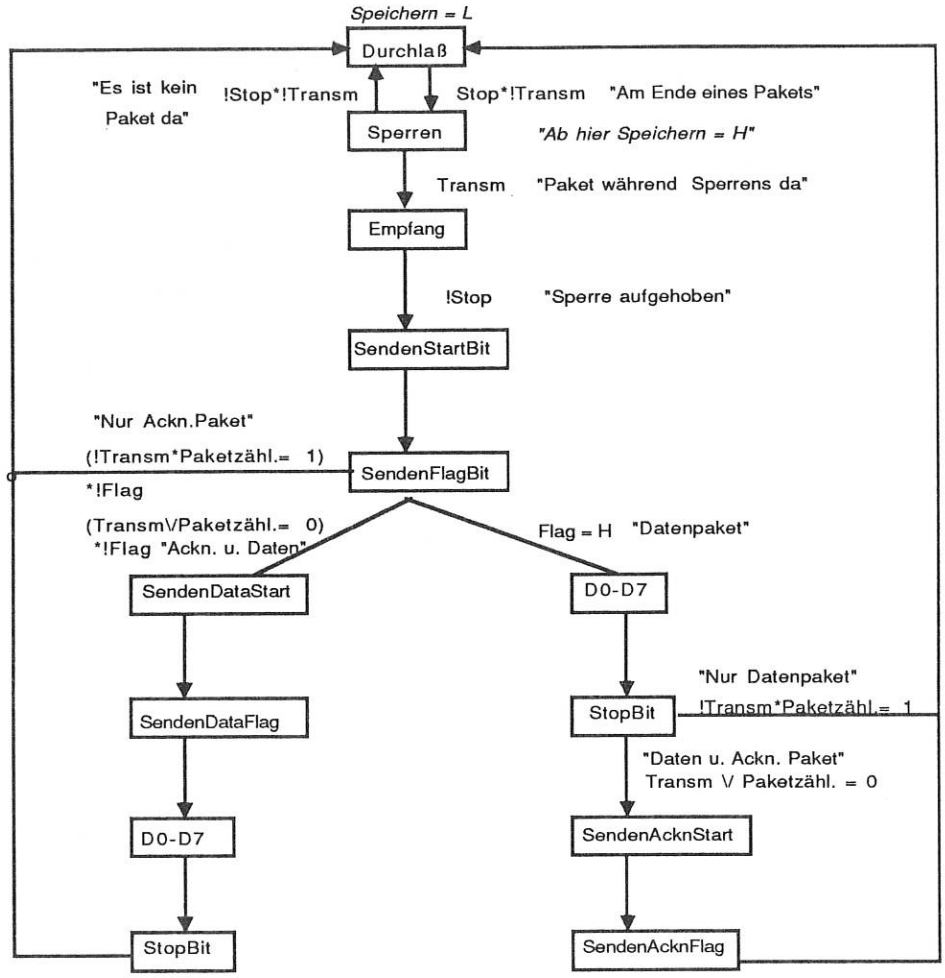


Bild A.1.10:
Der Ein-/Auslese-Automat. Das Problem dieses Automaten war es, *ohne Zeitverlust* in jedem Zustand des Auslesens ein neues Paket einlesen zu können.

Der komplette Synchronisierer, der aus den Einheiten Abtaster, Zwischenspeicher und Ein/Auslese-Automat besteht, wurde mehrfach einem einwöchigen Dauertest unter wechselnden Bedingungen unterworfen. Bei keinem Test trat ein Bitfehler auf. Die Zielvorgabe von $7{,}14*10^{-14}$ Fehlerausfallsrate wurde somit übertroffen. Sie kann auf unter 10^{-14} geschätzt werden.

Damit ist die Darstellung über den Aufbau von MULTITOP abgeschlossen. In den verbleibenden zwei Kapiteln des Anhangs werden die theoretischen Grundlagen zur schnellen Fourier-Transformation und zur Spline-Approximation dargestellt.

A.2 Grundlagen der schnellen Fourier-Transformation

A.2.1 Die Fourier Integrale

Jean Baptiste Joseph Fourier gab 1826 zwei Gleichungen zur sogenannten Fourier Analyse bzw. Synthese an:

$$S(f) = \int_{-\infty}^{+\infty} s(t) \cdot e^{-j2\pi ft}\, dt \qquad s(t) = \int_{-\infty}^{+\infty} S(f) \cdot e^{+j2\pi ft}\, df$$

Fourier Analyse — Fourier Synthese

In diesen Integralgleichungen wurde der Eichfaktor $1/2\pi$ wurde der Übersichtlichkeit wegen weggelassen. Als Beispiel einer Fourier-Transformierten ist das Spektrum einer sin(t)/t Schwingung in Bild A.2.1 dargestellt.

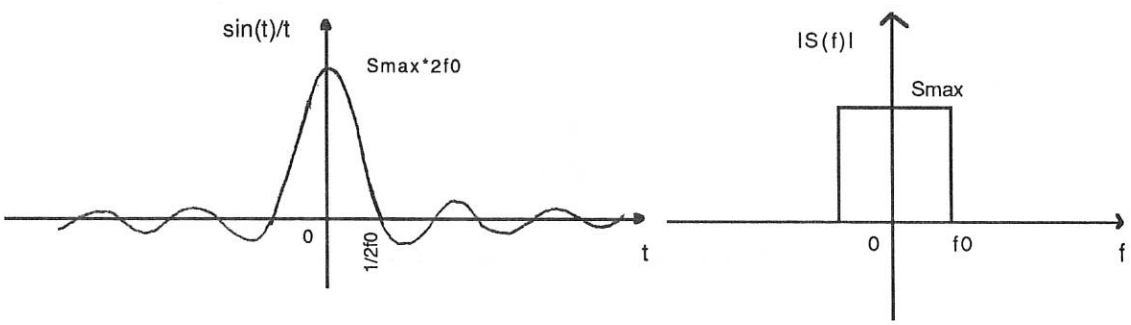


Bild A.2.1:
Eine sin(t)/t Schwingung hat ein rechteckförmiges Spektrum.

Die Fouriersynthese erlaubt es, ein Signal s(t) als Funktion von harmonischen Oszillatoren (sin, cos) von bestimmter Frequenz, Amplitude und Phase darzustellen. Die Fourieranalyse berechnet die Frequenz, Amplitude und Phase der harmonischen Oszillatoren. Bei den Fourier Integralen handelt es sich eine umkehrbar eindeutige Abbildung einer Funktion s(t):

Analyse
s(t) ----------> S(f)
<----------
Synthese

Dabei wird s(t) als Funktion der orthogonalen Basisfunktionen $e^{j\Omega t}$ dargestellt, die einen Funktionenraum, analog einem Vektorraum, aufspannen. S(f) kann als das Skalarprodukt

zwischen s(t) und den Basisfunktionen $e^{-j\Omega t}$ angesehen werden und gibt die Komponenten von s(t) entlang der Basisfunktionen an, analog der Komponentendarstellung eines Vektors in einem Vektorraum.

A.2.2 Die diskrete Fouriertransformation

Die Fourier Integrale haben zwei für die Praxis wichtige Eigenschaften. Zum einen sind die Transformationen bis auf das Minuszeichen im Exponenten identisch, so daß eine Eigenschaft, die für die eine Transformation gilt, auf die andere Transformation übertragbar ist. Zum anderen ist es so, daß einer numerischen Auswertung der Integrale in der Praxis das unendlich große Integrations Intervall entgegensteht. Man beschränkt sich daher auf Signale s(t), die nach einer endlichen Zeit T_1 periodisch fortgesetzt werden, so daß die Integration über die Zeit T_1 genügt.

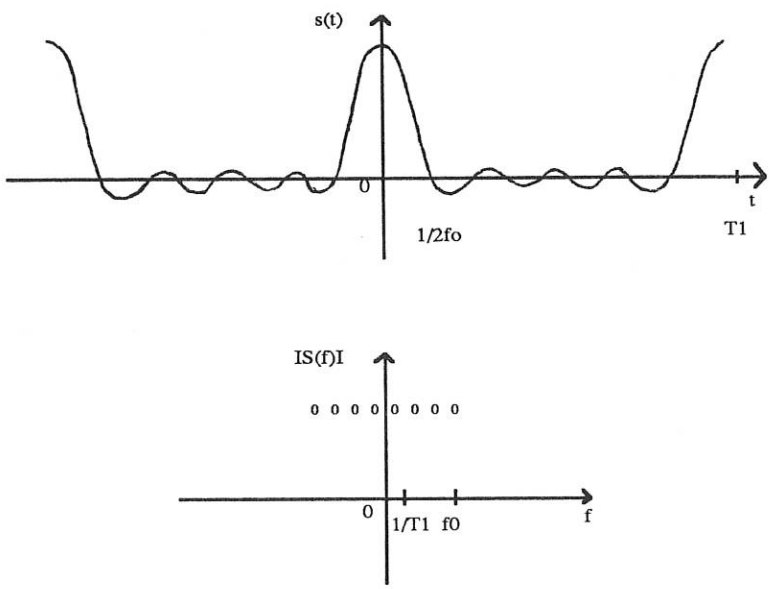


Bild A.2.2:
s(t) wird periodisch mit der Zeit T_1 fortgesetzt. Dann wird das Spektrum diskret.

Man kann zeigen, daß folgender wichtiger Satz gilt: Ein periodisches Zeitsignal s(t) hat stets ein diskretes Spektrum S(f) zur Folge und umgekehrt.

Für den Spezialfall periodischer Zeitsignale vereinfachen sich die Fourier Integrale zu:

$$S(nf_1) = \int_{-\infty}^{+\infty} s(t).e^{-j2\pi nf_1 t}\,dt \qquad s(t) = \sum_{-\infty}^{+\infty} S(nf_1).e^{+j2\pi nf_1 t}$$

Die rechte Gleichung (Synthese) wird oft auch als Fouriersumme bezeichnet.

Aufgrund des oben angegegebenen Satzes gilt, daß ein periodisches Spektrum ein diskretes

Zeitsignal zur Folge hat!

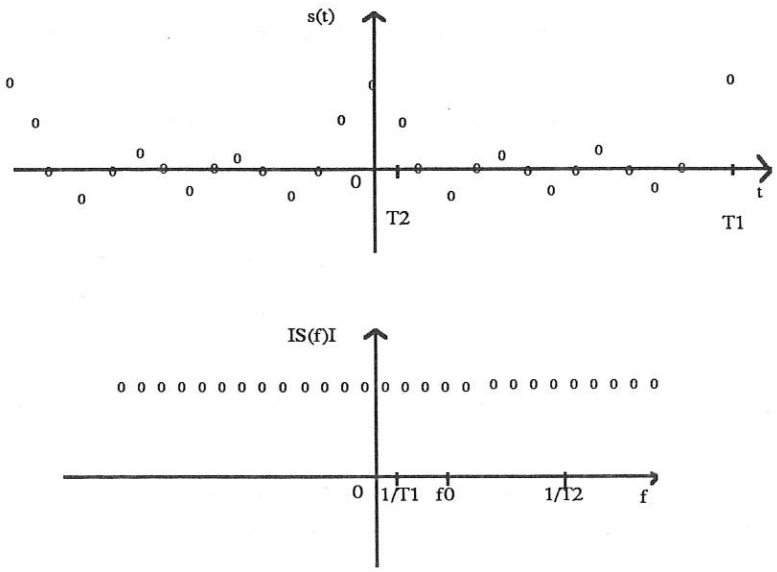


Bild A.2.3:
S(f) wird periodisch fortgesetzt. Dann wird das Zeitsignal diskret.

Für den Spezialfall periodischer Spektren vereinfachen sich die Transformationen weiter zu:

$$S(nf_1) = \sum_{m=0}^{T_1} s(mT_2).e^{-j2\pi nmf_1T_2} \qquad s(mT_2) = \sum_{n=0}^{f_0} S(nf_1).e^{+j2\pi nmf_1T_2}$$

Die Summation muß jetzt nur noch bis f_2 erfolgen, da das Spektrum periodisch mit f_2 fortgesetzt wird und damit keine neue Information mehr enthält.

Anmerkung:
$1/(f_2T_2) = T_1/T_2$ wird normalerweise so gewählt, daß der Quotient ganz ist und damit der Zahl der Abtastpunkte entspricht: $N = T_1/T_2 = f_2/f_1$. Mit dieser weiteren Vereinfachung erhält man:

$$S(nf_1) = \sum_{m=0}^{T_1} s(mT_2).e^{-j(2\pi/N)nm} \qquad s(mT_2) = \sum_{n=0}^{f_0} S(nf_1).e^{+j(2\pi/N)nm}$$

Diese Summen werden als diskrete Fouriertransformation bezeichnet.

A.2.3 Das Abtasttheorem

Die diskrete Fouriertransformation, als Vereinfachung der Fourier Integrale, hat aufgrund der folgender Eigenschaft analoger Signale sehr große Bedeutung:

1. Ein Signal s(t) kann unter gewissen Bedingungen ersetzt werden durch eine Folge von diskreten Werten, ohne daß dabei Information verloren geht.

2. Die diskreten Werte kann man numerisch verarbeiten (digitale Signalverarbeitung), z.B. mit Hilfe der diskreten Fourier-Transformation.

3. Ein analoges Signal läßt sich unter gewissen Bedingungen aus einer Folge von diskreten Werten rekonstruieren, ohne daß dabei Information verloren geht.

Die dazu notwendigen Bedingungen (Abtasttheorem) sind:

1. Das Spektrum von s(t) muß bandbegrenzt sein, damit es periodisch wiederholbar ohne Überlappung der Teilspektren ist.

2. Um eine Überlappung der Teilspektren ebenfalls zu vermeiden, muß weiterhin $f_1 > 2f_0$ sein. Dies bedeutet, daß s(t) mit mindestens der doppelten Frequenz abgetastet werden muß, als die Frequenz des höchsten spektralen Anteils (=f0) von s(t) ist.

Nach einer erfolgten numerischen Verarbeitung des Signals kann ein kontinuierliche Signal dadurch rückgewonnen werden, daß die periodische Wiederholung des Spektrums durch ein Filter verhindert wird (Rekonstruktions-Tiefpaß).

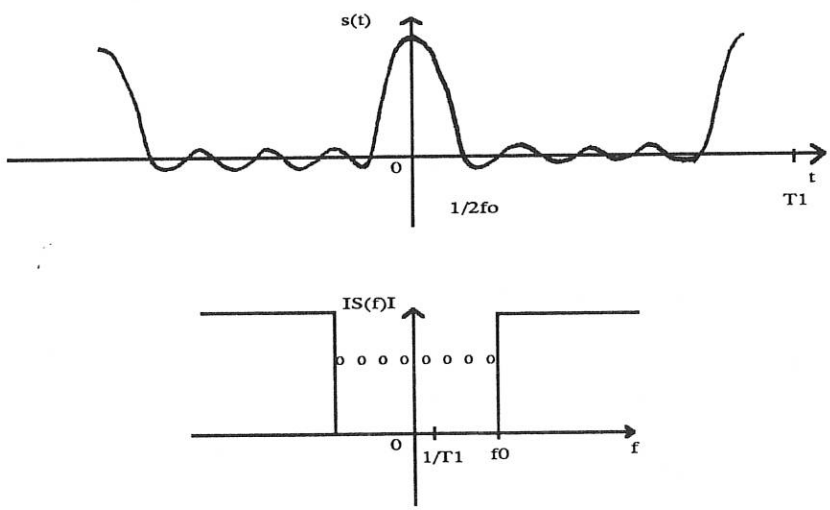


Bild A.2.4:
Rekonstruktion des kontinuierlichen Signals durch Tiefpaßfilterung. Da das Spektrum nicht periodisch ist, ist das Signal nicht diskret.

Die Tiefpaßfilterung des periodischen Spektrums läßt sich formal durch Multiplikation des periodischen Spektrums mit einer Fenster- oder Rechteckfunktion rectan(f_0,f) beschreiben.

$$S(f) \rightarrow S(f) \cdot \mathrm{rectan}(f_0,f)$$

Anmerkung:
rectan(f_0,f) soll eine Rechteckfunktion sein, die identisch Null für $f > f_0$ ist.

Nach dem 2. Teil des Abtasttheorems kann rectan(f_0,f) ersetzt werden durch rectan($f_2/2$,f), da $S(f) = 0$ für $f_0 \leq f \leq f_2/2$ ist.

Damit erhält man:

$$S(f) \rightarrow S(f) \cdot \text{rectan}(f_2/2,f)$$

Es läßt sich zeigen, daß einer Multiplikation im Frequenzbereich eine Faltung der Zeitsignale entspricht, S(f) und damit s(t) geschrieben werden kann als:

$$s(t) = \sum_{n=-\infty}^{\infty} s(nT_2) \frac{\sin(2\pi(f_2/2)(t - nT_2)}{2\pi(f_2/2)(t - nT_2)}$$

(Abtasttheorem 3. Teil)

Anmerkung:

1. Ein rechteckförmiges Spektrum der Bandbreite f_0 hat eine sinx/x Funktion gemäß $\sin(2\pi f_0 t)/(2\pi f_0 t)$ zur Folge.

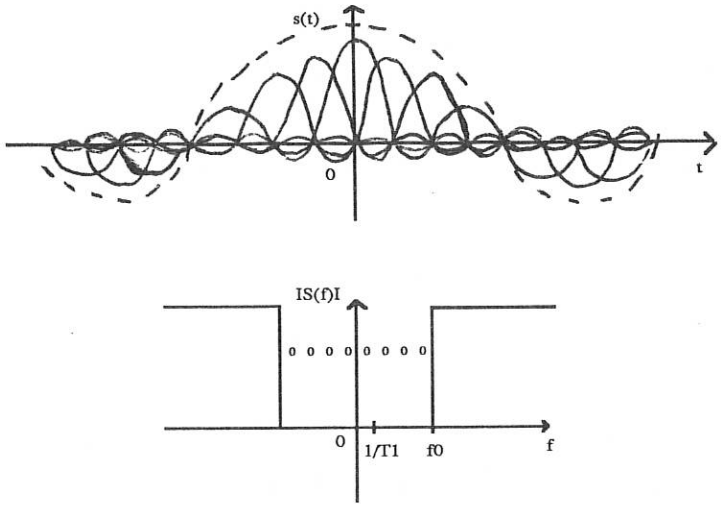
2. Die diskrete Faltung zweier Funktionen f(t) und g(t) ist:

$$f * g = \sum_{n=-\infty}^{+\infty} f(t) \cdot g(t - n)$$

3. Das Zeitsignal des diskreten und periodischen Spektrums S(nf1) ist das diskrete und periodische Zeitsignal s(nT2).

4. Damit erhält man den oben angegeben 3. Teil des Abtasttheorems.

Danach läßt s(t) auch darstellen als eine gewichtete Linearkombination von zeitverschobenen sin(t)/t Funktionen, wobei die Gewichte besonders einfach sind: sie sind die Werte von s(t) an den Stellen nT_2.



$T_0 = 1/2f_0$ = 1. Nullstelle von sin(x)/x — f_0 = Bandbreite des Spektrums

$T_1 = 1/f_1$: Periode des Zeitsignals — f_1 = Abstand der Spektrallinien

$T_2 = 1/f_2$: Abstand der Abtastwerte — f_2 = Periode des Spektrums = Nf_1

$$f_2/f_1 = T_1/T_2 = N = \text{Zahl der Abtastwerte}$$

Bild A.2.5:
Rekonstruktion des kontinuierlichen Zeitsignals mittels gewichteter und verschobener sin(x)/x Funktionen.

A.2.4 Matrizenschreibweise der diskreten Fouriertransformation

Die Fourier-Transformierten $s(mT_2)$ bzw. $S(nf_1)$ lassen sich auch mit Hilfe eines Index anstelle einer unabhängigen Variablen darstellen, sofern man eine normierte Zeit- bzw. Frequenzachse voraussetzt:

$$s(mT_2) \rightarrow s_m = \sum_{n=0}^{N-1} S_n \cdot e^{+j(2\pi/N)nm} \qquad \text{Synthese}$$

$$S(nf_1) \rightarrow S_n = \sum_{m=0}^{N-1} s_m \cdot e^{-j(2\pi/N)nm} \qquad \text{Analyse}$$

Anmerkung: Es gilt $T_1 = N \cdot T_2$ bzw. $f_2 = N \cdot f_1$.

Diese Indexschreibweise läßt z.B. s_m als ein Element eines Vektors von N Elementen erscheinen. Die Berechnung dieses Elements kann als eine Skalarprodukt des Zeilenvektors

$$(e^{j(2\pi/N)0m},\ e^{j(2\pi/N)1m},\ \ldots,\ e^{j(2\pi/N)(N-1)m})$$

mit dem Spaltenvektor

$$\begin{pmatrix} s_0 \\ s_1 \\ \dots \\ s_{N-1} \end{pmatrix}$$

aufgefaßt werden.

Damit läßt sich die diskrete Fourier-Transformation auch schreiben als:

Synthese	Analyse
$\underline{s} = W^{-1} \cdot \underline{S}$ mit	$\underline{S} = W \cdot \underline{s}$ mit
$W^{-1} = (w^{-1}_{nm})$ und	$W = (w_{nm})$ und
$w^{-1}_{nm} = 1/w_{nm} = e^{+j(2\pi/N)nm}$	$w_{nm} = e^{-j(2\pi/N)nm}$

W ist eine Matrix vom Typ [N,N].

Bild A.2.6:
Die diskrete Fourier-Transformation ordnet einem Vektor $\underline{s}$ von Abtastwerten einen Vektor $\underline{S}$ von Spektralwerten zu und umgekehrt.

A.2.5 Zeitbedarf der diskreten Fourier-Transformation

Die Multiplikation einer Matrix vom Typ [N,N] mit einem Vektor der Länge N erfordert für jedes Element des Vektors die Berechnung eines Skalarprodukts von N Faktoren. Unter der meistens berechtigten Annahme, daß die Zahl der Multiplikationen in diesem Skalarprodukt die weitaus größte Zeit im Vergleich zu den Additionen benötigt, erhält man die gesamte Berechnungszeit der DFT näherungsweise zu:

$$T_{DFT} = N^2 \cdot \text{Zeit pro Multiplikation} = O(N^2)$$

Die quadratische Abhängigkeit der Berechnungsdauer der DFT von der Zahl der Abtastpunkte macht eine numerische Auswertung der DFT für große N unmöglich.

A.2.6 Die schnelle Fourier-Transformation

Die schnelle Fourier-Transformation ist ein numerisches Verfahren zur beschleunigten Berechnung der diskreten Fourier-Transformation. Sie macht von der Tatsache Gebrauch, daß es weniger Zeit erfordert, zwei Matrizen der Größe [N/2xN/2], als eine Matrix der Größe [NxN] mit einem Vektor zu multiplizieren. Sie versucht demnach, die Berechnung eines Spektralvektors der Länge N auf die

Berechnung zweier Teilspektren der Länge N/2 zu reduzieren und diese dann mit geringem Aufwand zu dem Gesamtspektrum zu addieren. Und tatsächlich lassen sich z.B die Teilspektren der geraden bzw. ungeraden Abtastwerte zu einem Gesamtspektrum bei phasenrichtiger Addition zu einem Spektrum zusammenfügen, wie im folgenden gezeigt wird. Der zweite Trick, der bei der FFT angewandt wird, ist der, daß die Vektorlängeschrittweise bis zur Länge 1 halbiert wird (nur möglich für $N = 2^{n)}$ und dann das Spektrum von einzelnen Punkten zu berechnen übrig bleibt. Das Spektrum eines einzelnen Punktes ist aber der Punkt selbst, so daß der eigentliche Transformationsprozess bei der FFT entfällt. Jetzt müssen nur noch stufenweiseaus je zwei kleineren Spektren, angefangen von einzelnen Punkten, das nächst größere Spektrum durch phasenrichtige Addition der Teilspektren erzeugt werden, und so fort, bis man das Gesamtspektrum zusammengesetzt hat. Dies wird im folgenden genauer dargestellt.

a) Reduktion der Größe der Matrix W von [NxN] auf [N/2 x N/2]

Der Vektor der Abtastwerte wird in zwei gleichgroße Teilvektoren zerlegt, die z.B. die geraden bzw. die ungeraden Punkte, oder die 1. und die 2. Hälfte derPunkte enthalten. Dann muß man 2 . $O((N/2)^2)$ Operationen ausführen, was die Hälfte verglichen mit $O(N^2)$ ist. Die Teilmengen der Punkte werden jede für sich transformiert und ihre Spektren mit Hilfe sogenannter Drehfaktoren zumSpektrum aller Punkte addiert. Für den Fall einer Unterteilung in gerade und ungerade Punkte hat man für die Analyse:

Gesamtspektrum

$$S_n = \sum_{m=0}^{N-1} s_m \cdot e^{-j(2\pi/N)mn}$$

$$= \underbrace{\sum_{m=0}^{(N/2)-1} s_{2m} \cdot e^{-j(2\pi/N)2mn}}_{\text{Summation über gerade Indizes}} + \underbrace{\sum_{m=0}^{(N/2)-1} s_{2m+1} \cdot e^{-j(2\pi/N)(2m+1)n}}_{\text{Summation über ungerade Indizes}}$$

= für $0 \leq n \leq (N/2)-1$:

$$\underbrace{\sum_{m=0}^{(N/2)-1} s_{2m} \cdot e^{-j(2\pi/(N/2))mn}}_{\text{Spektrum der geraden Punkte}} + \underbrace{e^{-j(2\pi/N)n}}_{\text{Drehfaktor}} \cdot \underbrace{\sum_{m=0}^{(N/2)-1} s_{2m+1} \cdot e^{-j(2\pi/(N/2))mn}}_{\text{Spektrum der ungeraden Punkte}}$$

bzw.

= für $N/2 \leq n < N$:

$$\sum_{m=0}^{(N/2)-1} s_{2m} \cdot e^{-j(2\pi/(N/2))m(n-(N/2))} + e^{-j(2\pi/N)(n-(N/2))} \cdot \sum_{m=0}^{(N/2)-1} s_{2m+1} \cdot e^{-j(2\pi/(N/2))m(n-(N/2))}$$

Das Raster des Index m verhält sich dabei folgendermaßen:

ursprüngliches Raster:	m = 0,1,2,3,...,m,...,N-1 ,	insgesamt N Stück
gerades Raster:	2m = 0, 2, 4,...,N-2,	" N/2 "
ungerades Raster:	2m + 1 = 1, 3, 5,..., N-1,"	N/2 "

Tabelle A.2.1:
Der Summierung der DFT wird zerlegt in die Bildung zweier Teilsummen aus Summanden mit geradem bzw. ungeradem Index.

b) mehrfache Anwendung der Reduktion

Das zweite Prinzip, auf dem die FFT beruht, ist, daß die Reduktion der Größe der Transformationsmatrix W um die Hälfte schrittweise bis zur Größe eins fortgeführt wird. Dazu muß die Zahl der Abtastpunkte eine Zweierpotenz sein. Bei der Größe eins ist das Spektrum gleich dem Punkt selbst, und das nächstgrößere Spektrum von zwei Punkten entsteht aus der phasenrichtigen Addition der beiden Punkte mit Hilfe des Drehfaktors

$$e^{-j(2\pi/N)n}$$

Das nächstgrößere Spektrum von 4 Punkten entsteht aus der phasenrichtigen Addition der Spektren von zwei Punkten und so fort. Das Gesamtspektrum wird so schrittweise aus den Teilspektren aufgebaut. Für $N = 2^n$ Punkte sind dazu n Stufen erforderlich. Innerhalb einer Stufe müssen ca. N Multiplikationen komplexer Zahlen vorgenommen werden, so daß der Gewinn der FFT ist:

DFT direkt:	$O(N^2)$
DFT mit Hilfe der FFT:	$O(N\log_2 N)$

Für N = 1024 zum Beispiel sind für die DFT ca. 1 Mio. Multiplikationen auszuführen, bei der Berechnung der DFT über die FFT nur 5000!
Diese zwei Prinzipien der FFT, schrittweises Halbieren der Zahl der zu transformierenden Punkte und schrittweises Zusammensetzen von kleineren zu größeren Spektren, ist in dem folgenden Bild A.2.7 für das Beispiel N = 8 Punkte zusammengestellt. In diesem Bild wird die Notation von Signalflußgraphen verwendet.

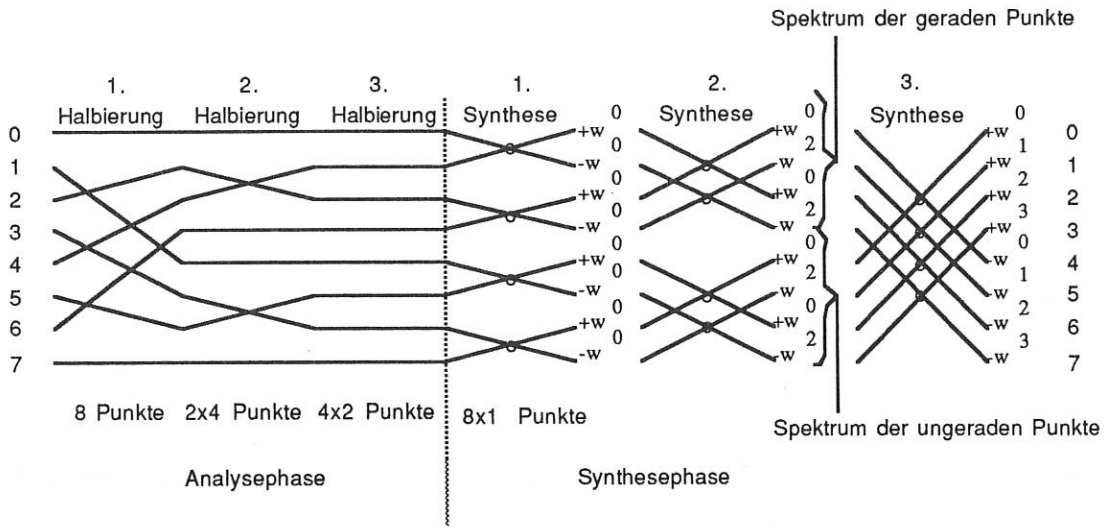


Bild A.2.7:
Die zwei Prinzipien der schnellen Fourier-Transformation, schrittweises Halbieren der Zahl der zu transformierenden Punkte und schrittweises Verdoppeln der Größe der Spektren.

Der angegebene Signalflußgraph entspricht der von Cooley-Tukey angegebenen FFT, wenn man die schrittweisen Halbierungen in einem Schritt ausführt:

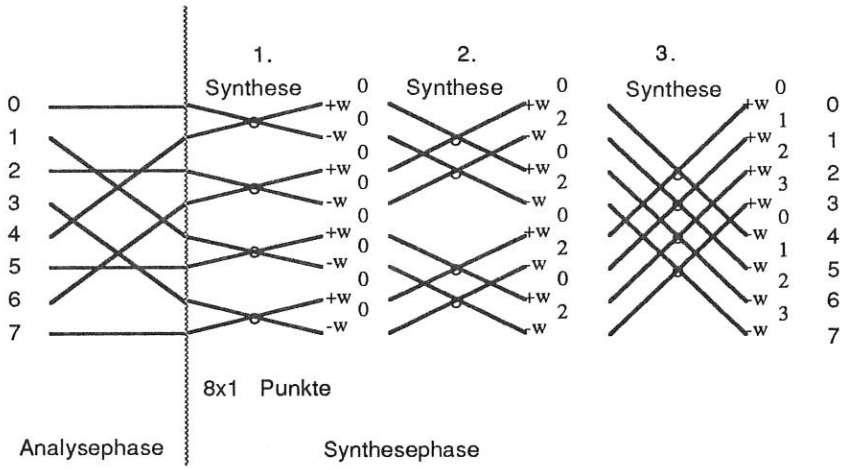


Bild A.2.8:
Die schnelle Fourier-Transformation nach Cooley-Tukey entsteht, wenn man die Halbierungen der Operanden in einem Schritt, der sogenannten Reversal Permutation durchführt.

A.2.7 Zusammenfassung

1. Ein periodisches Zeitsignal hat ein diskretes Spektrum zur Folge und umgekehrt. Ein periodisches Spektrum ist eine notwendige und hinreichende Bedingung für ein diskretes Zeitsignal.

2. Die diskrete Fourier-Transformation ist ein Spezialfall der Fourier-Integrale für periodische

Zeitfunktionen, die zudem ein periodisches Spektrum haben. Die Periodizität eines Zeitsignals ist nur näherungsweise erfüllbar, da in der Praxis Signale stets zeitbegrenzt sind.

3. Das Abtasttheorem legt die Bedingungen fest, unter denen ein kontinuierliches Zeitsignal ohne Informationsverlust durch ein diskretes Zeitsignal ersetzt werden kann (Periodisches Spektrum, ohne Überlappung der Teilspektren), und wie aus einem diskreten Signal ein kontinuierliches Signal rekonstruiert werden kann.

4. Die schnelle Fourier-Transformation ist ein effizientes Verfahren zur Berechnung der diskreten Fourier-Transformation. Sie beruht auf zwei Prinzipien:

   a) schrittweises Halbieren der Zahl der zu transformierenden Punkte bis zur Größe einzelner Punkte.
   b) schrittweises Verdoppeln der Länge der transformierten Punkte durch phasenrichtige Addition zweier Teilspektren zu einem größeren gemeinsamen Spektrum.

Damit ist die einführende Darstellung über die Fourier-Transformation und ihre numerisch effiziente Berechnung (FFT) abgeschlossen.

A.3 Grundlagen der Spline-Approximation

A.3.1. Die zur Interpolation notwendige Zahl von Basissplines

Ein Spline besteht aus zweimal stetig differenzierbar aneinandergehefteten Parabelbögen dritten Grades. Er läßt sich alternativ auch als Linearkombination von gewichteten und gegeneinander verschobenen Basissplines darstellen:

$$s(x) = p_0 B_0(x) + p_1 B_1(x) + \ldots + p_{n-1} B_{n-1}(x)$$

mit

$$B_i(x) = \begin{cases} 0 \text{ für } x' \le x_i \text{ und } x' = (x - x_i)/(x_{i+1} - x_i) \\ x'^3/6 \text{ für } x_i < x \le x_{i+1} \\ 1/6(-3x'^3 + 12x'^2 - 12x' + 4) \text{ für } x_{i+1} < x \le x_{i+2} \\ B_i(4 - x') \text{ sonst} \end{cases}$$

Die Frage ist nun, wie groß n gewählt werden muß, damit s(x) exakt durch eine gegebene Zahl von m Stützstellen geht.

Man kann zeigen [Pren], daß zur Interpolation von m Stützstellen genau m Basissplines notwendig sind.

Eine Datenreduzierung erfolgt somit für n < m durch eine Approximation des Splines an das Original.

Beispiel:
Es sollen m=4 gleiche Meßwerte durch einen Spline interpoliert werden. Welche Basissplines sind dazu notwendig?

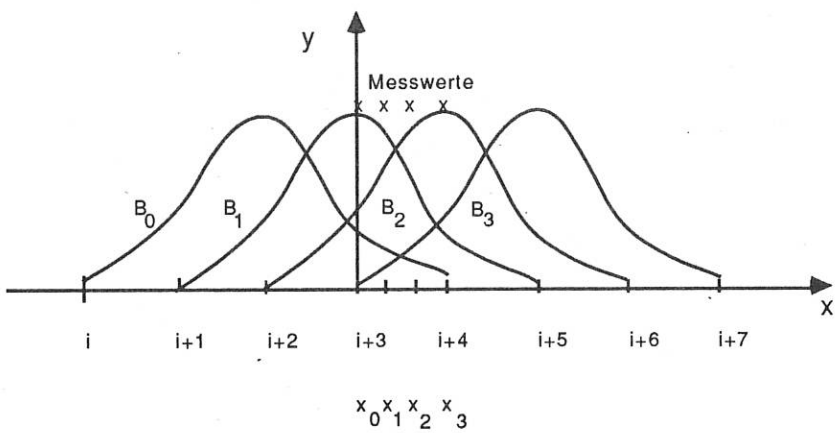


Bild A.3.1:
Interpolation von vier Meßwerten durch Basissplinefunktionen.

Man sieht an diesem Beispiel, daß das Intervall $[x_0,x_3]$ Teilmenge des Intervalls [i,i+7] sein muss, und daß das Raster der Basissplines ungleich dem Raster der Meßwerte ist ! Dies ist für die praktische Anwendung der Splines von Bedeutung und führt zum Begriff der inneren Knoten eines Splines.

A.3.2. Die inneren Knoten eines Splines

Gegeben seien in einem Intervall $[x_0,x_{m-1}]$ m äquidistante Stützstellen (Meßwerte) einer Funktion f(x), die durch einen Spline s(x) interpoliert oder approximiert werden sollen.

Sei $x_{i+1} - x_i = a$, das Raster der Meßwerte

und

$$s(x) = p_2B_2 + p_3B_3 + \ldots + p_{n-3}B_{n-3} + p_0B_0 + p_1B_1 + p_{n-2}B_{n-2} + p_{n-1}B_{n-1}$$

der gesuchte Spline.
Als Knoten des Splines werden die Stellen, an denen die Parabelbögen aneinandergeheftet werden, bezeichnet. Die inneren Knoten eines Splines sind diejenigen Konten, die im Inneren des Intervalls der Meßwerte liegen. Im obigen Beispiel haben B_0, B_1, B_{n-2} und B_{n-1} keine inneren Knoten.

Allgemein gilt für die Zahl k_i der inneren Knoten: $\mathbf{k_i = n - 4}$.

A.3.3. Das Raster der Knoten und der Meßwerte

Der Abstand zwischen den Abszissen der Meßwerte sei a und zwischen den Abszissen der Basissplines 1. Die Zahl der inneren Knoten muß deshalb verwendet werden, um den Wert a zu ermitteln, denn die Intervallänge, die aus der Zahl der inneren Knoten bestimmt wird, muß mit der Intervallänge der Meßwerte übereinstimmen.
Beispiel:

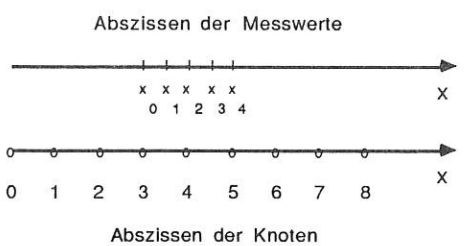


Bild A.3.2:
Das Raster der Stützstellen und der Meßwerte für n = m = 5.

Für gleiche Intervallänge gilt:

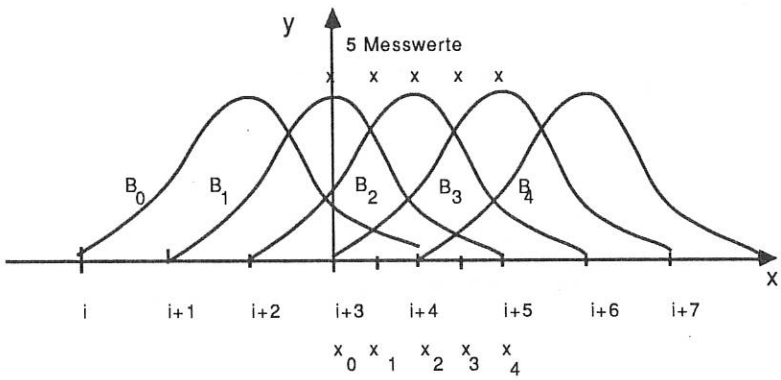
$$(k_i + 1) * 1 = a * m$$

bzw.

$$a = (n-3)/m, \text{ mit } m > 3.$$

Beispiel:

m = 5 Stützstellen und Interpolation, => n = 5, $k_i = 1$, a = 0,5



m = 6 Stützstellen und Approximation mit n = 5, => $k_i = 1$, a = 0,4

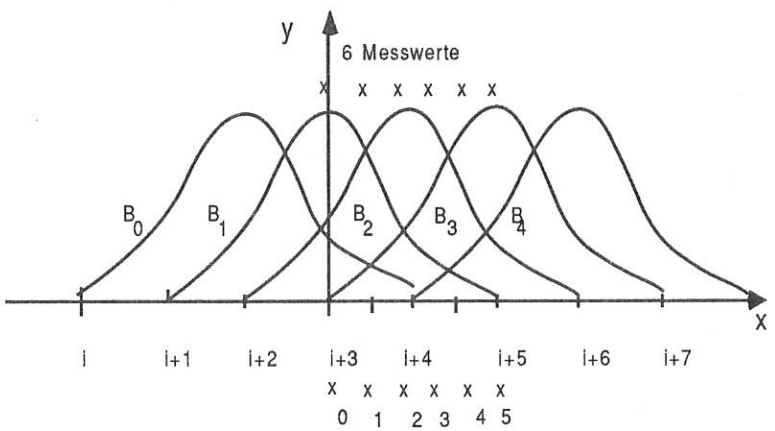


Bild A.3.3:
Das Raster der Meßwerte und der Basissplines bei Interpolation und Approximation.

Im unteren Beispiel wird das Raster der Meßwerte feiner, die Zahl der Basisplines bleibt gleich. Dies ist der Fall der Datenreduktion.

Damit ist die Phase der Definition und Beschreibung der Splines abgeschlossen. Es folgt nun auf informelle Art die Einführung des Begriffs der Norm, um dann den Algorithmus der Spline-Approximation mit Datenreduktion herzuleiten.

A.3.4. Approximation im Funktionenraum

Funktionen stellen das kontinuierliche Analogon zu Vektoren dar. Analog zu Vektorräumen lassen sich für Funktionen daher auch Funktionenräume definieren [Hell]. In einem solchen Funktionenraum läßt sich analog zum Abstand im Vektorraum ein Maß für die "Entfernung" zweier Funktionen definieren. Dieses Maß wird als "Norm" bezeichnet. Für die Approximation einer Funktion durch eine andere ist die Norm ein wichtiges Maß, da es die Güte der Annäherung angibt. Weiterhin lassen sich in Funktionenräumen analog zu den Vektorräumen beliebige Funktionen durch eine Linearkombination von Basisfunktionen darstellen. Wegen ihrer unendlich feinen Rasterung sind dazu allerdings i.a. auch unendlich viele Basisfunktionen notwendig. Bekannte Beispiele für Basisfunktionen sind sin(nt), cos(mt) in der Fouriersynthese. Funktionen können nun, wie Vektoren, senkrecht aufeinander stehen, wenn ihr "Skalarprodukt", das das kontinuierliche Analogon zum Vektor Skalarprodukt ist, verschwindet:

$$\int f_1(x) f_2(x)\, dx = 0$$

Das Skalarprodukt von Funktionen ist besonders nützlich, um den Anteil einer Basisfunktion f_1 an einer beliebigen Funktion f_2 zu bestimmen. Im Falle der Spline-Approximation wurden als Basisfunktionen die sog. Basissplines verwendet, die zwar nicht orthogonal aufeinander stehen, dafür aber lokal begrenzt sind.

A.3.5. Approximation im euklidischen Funktionenraum

Es gibt nun Funktionenräume, in denen eine sog. euklidische Norm definiert ist [Hell], die das kontinuierliche Analogon zum geometrischen Abstand zweier Punkte im Raum darstellt:

$$\| f \|_2 = \left(\int f^2 dx\right)^{1/2}$$

Es soll nun das Prinzip der Approximation erläutert werden, das hier verwendet wird.

Aus der Fehlerausgleichsrechnung in der Statistik ist das Prinzip der kleinsten Quadrate bekannt:

$$\left(\Sigma f^2(x)\right)^{1/2} = \text{MIN},$$

das demnach nichts anderes ist, als das Minimieren des Abstandes zwischen einem Messwertvektor und dem Vektor der Ausgleichskurve in einem m-dimensionalen Vektorraum (bei m Messwerten).

Nach diesem Prinzip soll auch hier approximiert werden. Das bedeutet, daß die euklidische Norm des Fehlers

$$r = y - s$$

minimal sein soll, wobei y die Originalfunktion und s die Annäherung ist, d.h.

$$\| r \|_2 = \text{MIN.}$$

Damit ist die Einführung in die Grundlagen der Spline-Approximation abgeschlossen. Am Schluß des Anhangs findet sich das Literaturverzeichnis. Die gesamte Dokumentation des MULTITOP-Rechners (Schaltpläne, PAL- und OCCAM Programme) ist im Zusatzteil zu dieser Arbeit enthalten.

Literaturverzeichnis

Literaturhinweise zur Architektur von MULTITOP

[Ims] Inmos Ltd., The Transputer Reference Manual, Bristol, 1987.

[Hoar] C.A.R. Hoare, Communicating Sequentiel Processes, Prentice-Hall, 1985.

[VME] Mostek Corp., VMEbus Technical Specification, Brüssel, 1982.

[Dirm] Händler, W.; Mähle, E.; Wirl,K.; DIRMU Multiprocessor Configurations. Proc. 1985 Int. Conf. on Parallel Processing, p.652-656, St.Charles 1985.

[Moto] Motorola Inc., MC 68 000 16-/32 Bit Microprocessor, Glasglow, 1985.

[Dene] Denelcor Inc., Heterogeneous Element Processor: Principles of Operation, April 1981.

[Hyp] C.E.Seitz, The Cosmic Cube, Com. of the ACM, Jan. 1985 Vol. 28 Nr. 1, p.22-33.

[Hill] W.D.Hillis, The Connection MAchine, Scientific American, 1987.

[Sieg] H.J.Siegel, G.B.Adams, D.P.Agrawal, A survey and comparison of fault-tolerant multistage interconnection networks, Computer, June 1987, p.14-27.

[Supr] U.Trottenberg, On the SUPRENUM Conception, SUPRENUM Report 1, SUPRENUM GmbH, 1987.

[Hoss] F.Hoßfeld, parallele Algorithmeen, Informatik Fachberichte, Springer 1983.

[Sche] U.Schendel, Einführung in die parallele Numerik, Oldenbourg, 1981.

[Hell] D.Heller, A survey of parallel Algorithms in num. Lin. Algebra, SIAM Review, Vol. 20, No. 4, Oct. 1978, p.740-777.

[AMD] AMD Corp., Bipolar Microprocessor Logic and Interface, p.8-2, AMD Sunnyvale CA., 1985.

Literaturhinweise zum modularen Koppelnetz

[Bene]: V.E.Benes, Math. theory of connecting networks and telphone traffic, Academic Press, 1965.

[Lee]: K.Y.Lee, IEEE Transactions on computers, Vol. c-34, No.5, May 1985.

[Wu1]: C.Wu, T.Feng, Interconnection networks for parallel and distributed processing, IEEE Computer society press, Silver spring, 1984.

[Hock]: Hockney, Jesshope, Parallel Computers, Adam Hilger Bristol, 1981.

[Wu2]: C.Wu, Tutorial note on parallel processing and interconnection networks, Dep. of electr. and comp. engineering, University of Texas at Austin, 1985.

Literaturhinweise zur schnellen Fourier-Transformation

[Brig] E.O.Brigham, The Fast Fourier Transform, Prentice-Hall, 1974.

[Gold] L.R.Rabiner, B.Gold, Theory and application of digital signal processing, Prentice-Hall, 1975.

[Swar] P.N.Swarztrauber, FFT algorithms for vector computers, Parallel Computing 1 (1984) p.45-63.

Literaturhinweise zur Spline Approximation

[Rein1]: C.H.Reinsch, " Smoothing by Spline Functions",, Numerische Mathematik 1967 Bd. 10, p.177-183.

[Rein2]: C.H.Reinsch, " Smoothing by Spline Functions II",Numerische Mathematik 1971 Bd. 16, p.451-454.

[Rein3]: C.H.Reinsch, "Skript zur Vorlesung Einführung in die numerische Mathematik", TU München 1980, p. 72-76, p.85-86.

[Jord1]: G.Jordan-Engeln,F.Reutter, "Formelsammlung zur numerischen Mathematik", BI Hochschultaschenbücher Band 106. p. 145-164.

[Jord2]: G.Jordan-Engeln,F.Reutter, "Numerische Mathematik für Ingenieure", BI Hochschultaschenbücher Band 104., p. 236-237.

[Hell]: J.Hellauer, "Die Verwendung v. kubischen Interpolations- u. Ausgleichssplines z. Meßdatenverarbeitung", MPI f. Plasmaphysik, R-41, 1983.

[Hoss]: F.Hossfeld, "Parallele Algorithmen", Springer, Informatik Fachberichte Bd. 64, 1983, p.151

[Spät1]: H. Späth, "Algorithmen für elementare Ausgleichsmodelle", R.Oldenbourg, p. 59-67.

[Spät2]: H. Späth, "Spline-Algorithmen zur Konstruktion glatter Kurven und Flächen ", R.Oldenbourg, p.76-85.

[NAG]: Numerical Algorithms for FORTRAN Programs, "E02BAF", NAGLIB 1984, p.1-10.

[Dege]: W.Degen, "Skript zur Vorlesung Numerische Mathematik für Ingenieure", TU Stuttgart 1976, p.84.

[Schu]: L.L.Schumaker, "Spline Functions: Basic Theory", J.Wiley & sons, p. 77-91,118-139.

[Pren]: P.M.Prenter, "Splines and variational methods", Wiley & sons, p.77-91.

[Haye1]: J.G.Hayes, " Numerical methods for curve and surface fitting", National Physical Laboratory 1974, p. 144-153.

[Haye2]: M.G.Cox, J.G.Hayes, " A guide and suite of algorithms for the non-specialist user", National Physical Laboratory 1973 Report NAC 26.

[Cox]: M.G.Cox, "The numerical evaluation of B-Splines", J.Inst.Maths. Aplics 197210, p. 134-149.

[Boor]: C.de Boor, "On calculating with B-Splines", J. of Approx. Theory 1972 6, p 50-62.

[Evan]: D.J.Evans, "Software for numerical mathematics", Academic Press 1974, p. 235-251.

[Wood]: C.H.Wood, "An Algorithm for data smoothing using spline functions", BIT 1970 10, p.501-510.

[Ichi1]: K.Ichida, F.Yoshimoto, T.Kiyono, " Curve fitting by a picewise cubic polynomial", Computing 16 , Springer 1976, p.329-338.

[Ichi2]: K.Ichida, F.Yoshimoto, T.Kiyono, " Curve fitting by a one pass method with a picewise cubic polynomial", ACM Transactions on Math. Software, Vol 3. No. 2, 1977, p.164-174.

[West]: J.R.Westlake, " A handbook of numerical matrix inversion and solution of linear equations", Wiley & sons.