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

Copyright © 1988 by
Max-Planck-Institut für Plasmaphysik
8048 Garching
All rights reserved
Gliederung

1. Einleitung ..................................................................................................................... 1

2. Die Architektur von MULTITOP .................................................................................. 6
   2.1 Parallele Architekturen .............................................................................................. 7
   2.1.1 Gemeinsamer Speicher und gemeinsame Variable .................................................. 8
   2.1.2 Mischformen der Kommunikation ......................................................................... 11
   2.2 Das Kommunikationskonzept von MULTITOP ......................................................... 12
   2.3 Die dynamisch variable Topologie .......................................................................... 13

3. Das modulare Koppelnetz von MULTITOP ................................................................... 21
   3.1 Bestehende Koppelnetze mit vollständiger Erreichbarkeit ......................................... 21
   3.2 Die Topologie des modularen Koppelnets ................................................................. 24
   3.3 Das Modell des Speichers mit Adressen und Inhalten ............................................. 25
   3.4 Der rechte Teil des Netzes ....................................................................................... 26
   3.4.1 Die Implementierung mit Hilfe von Kreuzschaltern ............................................. 26
   3.4.2 Die Topologie des rechten Teils .......................................................................... 27
   3.4.3 Bedingung für blockierungsfreies Arbeiten ......................................................... 28
   3.4.4 Der Begriff der Distanz ....................................................................................... 29
   3.5 Der linke Teil des Netzes ......................................................................................... 30
   3.5.1 Trennung von I₀ und I₁ auf die Distanz 2ʰ .......................................................... 31
   3.5.2 Einstellen aller Distanzen durch mehrfaches Trennen ......................................... 32
   3.5.3 Die Implementierung mit Hilfe von Kreuzschaltern ............................................. 33
   3.5.4 Topologie des linken Teils ................................................................................... 33
   3.5.5 Bedingung für blockierungsfreies Arbeiten ......................................................... 34
   3.6 Zusammenfassung ................................................................................................... 39
   3.7 Die modulare Erweiterung des Netzes ................................................................... 40
   3.8 Die Algorithmen zum Einstellen der Schalter ......................................................... 41

4. Die Parallelisierung der schnellen Fourier Transformation ......................................... 45
   4.1 Die Granularität des Problems ................................................................................. 45
   4.2 Die Zuordnung von Butterflies zu Prozessoren ....................................................... 46
   4.3 Die Beschleunigungserwartung bei Parallelisierung ............................................... 48
   4.4 Die Signalflußgraphen der verschiedenen Formen der FFT .................................. 49
   4.4.1 Datenflußanalyse der Permutationen zwischen den Stufen ................................. 51
4.4.2 Datenflußanalyse der verschiedenen Formen der FFT ........................................ 58
4.4.3 Der Zeitbedarf von Butterfly-Berechnung und Kommunikation .................... 60
4.4.4 Kommunikation bei variabler Topologie ....................................................... 61
4.5 Ein Mathematisches Modell für Berechnungszeiten und Wirkungsgrad ............... 63
4.6 Messergebnisse mit MULTITOP ................................................................. 64
4.7 Extrapolation auf eine sehr große Zahl von Prozessoren ................................ 68

5. Die parallele Spline-Approximation .................................................................. 73
5.1 Interpolation, Approximation und Glättung einer Folge von Messwerten .......... 73
5.2 Der Spline, die Kurve kleiner Krümmung ....................................................... 74
5.3 Spline Approximation ohne Datenreduktion .................................................. 75
5.4 Ein Verfahren zur Spline-Glättung mit Datenreduktion .................................. 76
5.5 Die Parallelisierung der die Messdaten reduzierenden Approximation .......... 79
5.6 Der Einfluß der Topologie ............................................................................. 80
5.6.1 Matrix parallel transponieren ................................................................. 80
5.6.2 Die parallele Matrix-Vektor-Multiplikation ............................................. 83
5.6.3 Die parallele Matrix-Matrix-Multiplikation in einem systolischen Array .... 86
5.6.4 Paralleles Lösen eines linearen Gleichungssystems ................................. 87

6. Zusammenfassung und Ausblick ....................................................................... 91

Im Anhang:

A.1 Realisierung des MULTITOP-Rechners ......................................................... 95
A.2 Grundlagen der schnellen Fourier-Transformation ....................................... 113
A.3 Grundlagen der Spline-Approximation ....................................................... 124
A.4 Literaturverzeichnis ..................................................................................... 129
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.


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

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

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:

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

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 "Multiple Instruction, Multiple Data"-Maschinen (MIMD) bezeichnet werden, allgemeiner sind als Vektorrechner, die man als "Single Instruktion, Multiple Data"-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 Parallelechner 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.

![Diagramm von MULTITOP](image)

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 Programmarchitektur ein Rechner, der gleichzeitig mehrere Prozesse bearbeitet.

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


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 Konstruktors 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.


<table>
<thead>
<tr>
<th>Gemeinsamer Speicher und gemeinsame Variable</th>
<th>Gemeinsamer Speicher und Botschaftenaustausch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kanäle und gemeinsame Variable</td>
<td>Kanäle und Botschaftenaustausch</td>
</tr>
</tbody>
</table>

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


![Diagramm 2.2a](image)

Bild 2.2a:

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

![Diagramm 2.2b](image)

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


Bei einer Kopplung von Prozessoren über einen gemeinsamen Speicher mit Hilfe eines Koppelnnetzes 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 Koppelnnetzes 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:

![Diagramm des VMEbus](image)

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 beschränkt. 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 \((m < n)\).


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.


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 Ablaufes die Prozessoren für Benutzerprozesse blockiert. Eine Verlangsamung des Zugriffs auf gemeinsame Variable um ein bis zwei Größenordnungen 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

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

2.2 Das Kommunikationskonzept von MULTITOP


Für dieses Kommunikationskonzept sprechen weitere Gründe:


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" bedeutet, 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 Signalfußgraph einer dieser Formen ist im nächsten Bild für den Fall von 16 zu transformierenden Werten dargestellt.

![Signalfußgraph einer schnellen Fourier-Transformation (FFT) mit minimaler Interprozessor- Kommunikation (Gentlemen-Sande FFT). Die Permutationsfunktionen zwischen den Stufen sind verschieden.](image)

Man sieht, wie sich die Topologie des Signalfusses 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:

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.


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.


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

\[
e = \frac{T_{\text{alg}}}{T_{\text{alg}} + T_{\text{kom}}}
\]

Im Falle der FFT ist:

\[ T_{\text{Kom}} = T_{\text{alg}}. \]

Daraus erhält man:

\[ e = 50\%. \]

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 Umschalzeit zwischen den Topologien klein sein im Vergleich zur Rechenzeit pro Topologie, d.h., es muß schnell umgeschaltet werden. Im einzelnen sei \( T_U \) die Umschalzeit, \( T_{\text{Spez}} \) die Rechenzeit bei einer dem Problem angepaßten Topologie und \( T_{\text{allg}} \) die Rechenzeit bei einer allgemeinen, festen Topologie wie z.B. einem Gitter (nearest neighbour). Ein Gewinn an Rechenzeit ergibt sich nur dann, wenn

\[ T_U + T_{\text{Spez}} < T_{\text{allg}} \]

ist. Bei MULTITOP kann eine bei der Übersetzung des Programms bereits bekannte Topologie in ca. 10 \( \mu s \) 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 erster 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:

\[ A^T A p = A^T y \]

Darin ist die Matrix, die sich aus den Meßdaten y und den Stützstellen des Splines ergibt, die Unbekannte p enthält die Gewichtungsfaktoren von der Linearcombination 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
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.
Das Konzept von MULTITOP

Welche Architektur muß nun ein Multiprozessor-System aufweisen, das völlig verschiedene Topologien gleichermaßen gut unterstützt 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:

![Diagramm der Architektur](image)

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:

![Diagramm der vollständigen Vermaschung](image)

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 zusammenschalten, einen Kreuzschienenverteiler zu verwenden. Dieser ist in Bild 2.10 zusammen mit den dazu gehörenden PMs dargestellt:

![Diagramm](Image)

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. Koppelnets vorgenommen, das mit \( O(N\log N) \) 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 Koppelnets unterstützt in wirtschaftlicher Weise alle möglichen Topologien gleich gut, indem die Topologie der Prozessorverbindungen der Topologie des Problems angepaßt wird.

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 Rechenmodulen 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 Koppelnetworks 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 (Modulen)


3.1 Bestehende Koppelnetze mit vollständiger Erreichbarkeit

Der Kreuzschienenverteiler

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

![Diagramm](image)


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"

## 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:

![Diagramm des Clos-Koppelnetzes](image)

**Aufwand:** $O(3N^{3/2})$

**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.


$$N = 2^{10}$$

<table>
<thead>
<tr>
<th>Kreuzschienenverteiler</th>
<th>Clos Netz</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{20}$</td>
<td>$3 \times 2^{15}$</td>
</tr>
</tbody>
</table>
Das Benes-Koppelnetz


![Diagramme](image)

Bild 3.3:

Man erhält für die Zahl der hier notwendigen Schalter:

\[
N = 2^{10}
\]

<table>
<thead>
<tr>
<th>Netz</th>
<th>Anzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clos Netz</td>
<td>96(\times2^{10})</td>
</tr>
<tr>
<td>Benes Netz</td>
<td>19(\times2^{10})</td>
</tr>
</tbody>
</table>


Das Lee-Koppelnetz

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:

![Diagramm](image)

Aufwand: O(2NlogN-N)

Bild 3.5:

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:

\[
\begin{pmatrix}
3 \\
2 \\
1 \\
0
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & 0 & 1 & 1 \\
1 & 1 & 2 & 2 \\
2 & 2 & 3 & 3 \\
3 & 3 & 0 & 0
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 \\
1 \\
2 \\
3
\end{pmatrix}
\]

Bild 3.6:


<table>
<thead>
<tr>
<th>Speicher Adresse</th>
<th>Inhalt</th>
<th>Adresse</th>
<th>Inhalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

vorher  nachher

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.

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: Sortierungsvorgang 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 $2^{n-6}$ Bits. Voraussetzung ist, daß $N = 2^n$ und $i = [0,1,\ldots,N-1]$ ist.]

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


- 26 -
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 niedrigeren Bits der Adressen in $2^n$-1 Adressteilbereichen dieser Permutation unterworfen werden.

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

3.4.2 Die Topologie des rechten Teils des Koppelnetszes

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 Koppelnetszes für $N = 8$ und einem Permutationsvektor $(3,4,7,1,5,2,0,6)$
3.4.3 Bedingung für blockierungsfreies Arbeiten


\[ A = a_{n-1}a_{n-2}...a_10 \]
\[ A' = a_{n-1}a_{n-2}...a_11 \]

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:

\[ I(s) = i_{n-1}i_{n-2}...i_1i_0 \]
\[ J(s) = j_{n-1}j_{n-2}...j_1j_0 \]

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:

\[ i_{n-1} = j_{n-1} \]
\[ i_{n-2} = j_{n-2} \]
\[ ... \]
\[ i_{n-(s-1)} = j_{n-(s-1)} \]

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):

\[ I(s) = i_{n-1}...i_{n-s}i_{n-(s+1)}i_{n-(s+2)}...i_0 \]
\[ J(s) = i_{n-1}...i_{n-s}j_{n-(s+1)}j_{n-(s+2)}...j_0 \]

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 komplementären 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:

\[
\begin{align*}
\text{Die Inhalte} & \quad I(s) = 000 \text{ mit } A(s) = 000 \\
& \quad J(s) = 001 \text{ mit } B(s) = 111
\end{align*}
\]

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

\[
A(s) = a_{n-1} \cdots a_{s+1} a_s a_{s-1} \cdots a_0
\]

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
\[ J(s) = i_{n-1} \ldots i_{n-s} i_{n-(s+1)} i_{n-(s+2)} \ldots i_0 \] 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 \( J(s) \) gehörige 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 gehörige 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 \( s = 1, 2, \ldots, n \) gilt. Die Aufgabe von \( NL \) ist es also, alle Paare \( I(s), J(s) \) in allen Stufen \( s \) mit

\[ I(s) = i_{n-1} \ldots i_{n-s} i_{n-(s+1)} i_{n-(s+2)} \ldots i_0 \]
\[ J(s) = i_{n-1} \ldots i_{n-s} i_{n-(s+1)} i_{n-(s+2)} \ldots i_0 \]

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.

\[
(I(s) = i_{n-1}i_{n-2}...i_1i_0, \quad J(s) = i_{n-1}i_{n-2}...i_1i_0)\]

b) Um blockierungsfrei vor der 1. Stufe zu sein, müssen $I(s)$ und $J(s)$ am Eingang von NR die Distanz $2^0$ haben.

\[
(I(s) = i_{n-1}i_{n-2}...i_1i_0, \quad J(s) = j_{n-1}j_{n-2}...j_1j_0)\]

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

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

\[
(I(s) = i_{n-1}...i_{n-s}i_{n-(s+1)}j_{n-(s+2)}...j_0) \quad \text{und} \quad \quad J(s) = i_{n-1}...i_{n-s}j_{n-(s+1)}j_{n-(s+2)}...j_0)\]

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:

![Diagramm](image)

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 $2^{n+1}$ 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 nach 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:

![Diagramm der Implementierung](image)

Bild 3.16:
Die Implementierung der Trennung zweier Paare \( I(s), J(s) \) auf die Distanz \( 2^8 \) 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 \( 2^8 \) auf alle nächstkleinere Distanzen führt auf die folgende Topologie für NL:

![Diagramm der Topologie](image)

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 Module 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, vorgeschaltet. Deshalb bleibt zu zeigen, daß NL ohne ein vorgeschaltetes Netz blockierungs-frei 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 verschiedenen 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 wiederholt, 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:

![Diagramm 1](image1)

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:

![Diagramm 2](image2)

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

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 ungeraden 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 unmarkierten 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.

\[\begin{array}{c}
I(s) \\
I′(s) \\
I′′(s) = J′(s)
\end{array}\]
\[\begin{array}{c}
J(s) \\
J′(s) \\
J′′(s)
\end{array}\]

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

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 an 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) wiederholt.

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 wiederholt, 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 (Hälfte, Viertel, Achtel 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 $(N/2)^*(2\log N - 1)$ Schaltern. Der technische Vorteil dieses Netzes gegenüber dem Benes-Netz liegt darin, daß es aus zwei gleichen Modulen 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:

![Diagramm](image)

**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.

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 Kreuzstellung*

(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:

<table>
<thead>
<tr>
<th>Adresse</th>
<th>Inhalt</th>
<th>Stufe 3</th>
<th>Adresse</th>
<th>Inhalt</th>
<th>Stufe 2</th>
<th>Adresse</th>
<th>Inhalt</th>
<th>Stufe 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 0</td>
<td>3 011</td>
<td>3 011</td>
<td>1 001</td>
<td>0 000</td>
<td>x</td>
<td>0 000</td>
<td>1 001</td>
<td></td>
</tr>
<tr>
<td>001 1</td>
<td>4 100</td>
<td>1 001</td>
<td>0 001</td>
<td>0 000</td>
<td>x</td>
<td>2 010</td>
<td>2 010</td>
<td></td>
</tr>
<tr>
<td>010 2</td>
<td>7 111</td>
<td>0 001</td>
<td>2 010</td>
<td>3 011</td>
<td>x</td>
<td>3 011</td>
<td>3 011</td>
<td></td>
</tr>
<tr>
<td>011 3</td>
<td>1 001</td>
<td>x</td>
<td>0 000</td>
<td>2 010</td>
<td>x</td>
<td>4 100</td>
<td>4 100</td>
<td></td>
</tr>
<tr>
<td>100 4</td>
<td>5 101</td>
<td>4 100</td>
<td>3 011</td>
<td>4 100</td>
<td>=</td>
<td>5 101</td>
<td>5 101</td>
<td></td>
</tr>
<tr>
<td>101 5</td>
<td>2 010</td>
<td>5 101</td>
<td>1 001</td>
<td>6 110</td>
<td>x</td>
<td>6 110</td>
<td>6 110</td>
<td></td>
</tr>
<tr>
<td>110 6</td>
<td>0 000</td>
<td>6 110</td>
<td>1 001</td>
<td>7 111</td>
<td>x</td>
<td>7 111</td>
<td>7 111</td>
<td></td>
</tr>
<tr>
<td>111 7</td>
<td>6 110</td>
<td>1 001</td>
<td>1 001</td>
<td>7 111</td>
<td>x</td>
<td>1 001</td>
<td>7 111</td>
<td></td>
</tr>
</tbody>
</table>

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:

$I(s) = i_{n-1} \cdots i_{n-s}i_{n-(s+1)}i_{n-(s+2)}\cdots i_0$

$J(s) = i_{n-1} \cdots i_{n-s}j_{n-(s+1)}j_{n-(s+2)}\cdots j_0$

$J(s)$ 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).

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.

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.


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:

\[
\begin{align*}
  a & \quad w^k & \quad c \\
  b & \quad \downarrow & \quad d \\
  c &= a + bw^k \\
  d &= a - bw^k
\end{align*}
\]

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

In der Schreibweise mit Real- und Imaginärteil:
\[c = cr + jci = ar + br*wr - bi*wi + j(ai + br*wi + bi*wr)\]
\[d = dr + jdi = ar - (br*wr - bi*wi) + j(ai - (br*wi + bi*wr))\]

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


4.2 Zuordnung von Butterflies zu Prozessoren

Es existieren drei verschiedene Möglichkeiten, die Butterflies Prozessoren zuzuordnen:

1. \(P = N\)


2. \(P \times k = \log_2 N, k = 1, 2, ..., \log_2 N\) (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 $N = 8$, $P = \log N = 3$ 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. $P \times k = N/2$, $k = 1,2,\ldots,N/2$

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

Bild 4.5:

4.3 Beschleunigungserwartung bei Parallelisierung

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.

<table>
<thead>
<tr>
<th>Zahl der Prozessoren</th>
<th>Rechenzeit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O(Nlog₂N)</td>
</tr>
<tr>
<td>log₂N</td>
<td>O(N) (pipelined)</td>
</tr>
<tr>
<td>N/2</td>
<td>O(log₂N)</td>
</tr>
<tr>
<td>(N/2)log₂N</td>
<td>O(1) (pipelined)</td>
</tr>
</tbody>
</table>

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 überlappenden 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 
zukönnen, 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 
differenten 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

\[ c = a + bw \]
\[ d = a - bw \]

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.


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:

Es gibt verschiedene Möglichkeiten zu klären, wie viele Daten bei der Shuffle-Permutation zwischen P Prozessoren ausgetauscht werden. Es zeigte sich, dass es am einfachsten ist, zu untersuchen, wie viele 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:

\[
S = a_n a_{n-1} \ldots a_1 \quad \rightarrow \quad A' = a_{n-1} a_{n-2} \ldots a_1 a_n
\]

Es gibt dann keinen Datenaustausch, wenn A und A' im selben Prozessor sind. Bei \( P = 2^p \) Prozessoren spezifizieren 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

\[
\begin{align*}
   a_n &= a_{n-1} \\
   a_{n-1} &= a_{n-2} \\
   & \quad \ldots \\
   a_{n-(p-1)} &= a_{n-p}
\end{align*}
\]

ist, bzw.:

\[
\begin{align*}
   a_n &= a_{n-1} = a_{n-2} = \ldots = a_{n-p}
\end{align*}
\]

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 Adressebits 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 \( 2^n - (2^n - p) = 2^p(1 - 2^{-p}) \) 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.

\[
\begin{array}{c}
p \text{ Bits} & n - p \text{ Bits} \\
\{a_n, \ldots, a_{n-p+1}, \ldots, a_k, \ldots, a_1\} & B_k \\
\text{k Bits} & \rightarrow \{a_n, \ldots, a_{n-p+1}, \ldots, a_1, \ldots, a_k\}
\end{array}
\]

2. \( k > n - p \)

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

\[
\begin{array}{c}
p \text{ Bits} & n - p \text{ Bits} \\
\{a_n, \ldots, a_k, \ldots, a_{n-p+1}, \ldots, a_1\} & B_k \\
\text{k Bits} & \rightarrow \{a_n, \ldots, a_1, \ldots, a_{n-p+1}, \ldots, a_k\}
\end{array}
\]

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 \( 2^p \cdot 2^n - p - 1 \) Daten invariant. Die Zahl der Daten, die durch \( B_k \) bewegt werden, ist insgesamt \( 2^n - 2^{n-1} = 2^{n-1} \).
\[ E_{Bk} = \begin{cases} 
0 & \text{für } k \leq n-p \\
2^{n-1} & \text{für } k > n-p 
\end{cases} \]

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 \(k > n-p\)).

**Reversal-Permutation**

Die Reversal-Permutation hat die folgende Topologie:

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:

   \[
   A = a_n \ldots a_{(n/2)+1} a_{(n/2)} \ldots a_1 \quad \rightarrow \quad A' = a_1 \ldots a_{(n/2)} a_{(n/2)+1} \ldots a_n
   \]

   \[ - \quad 54 \quad - \]
1.1 $p \leq n/2$

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

\[
a_n = a_1 \\
a_{n-1} = a_2 \\
... \\
a_{n-p+1} = a_p
\]

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:

\[
a_n = a_1 \\
a_{n-1} = a_2 \\
... \\
a_{(n/2)+1} = a_{n/2}
\]

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:

\[
A = a_n,..,a_{(n+3)/2},a_{(n+1)/2},a_{(n-1)/2},..,a_1 \quad \rightarrow \quad A' = a_{1},..,a_{(n-1)/2},a_{(n+1)/2},a_{(n+3)/2},..,a_n
\]

Mitte

Mitte

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

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

\[
a_n = a_1 \\
a_{n-1} = a_2 \\
... 
\]
\[ a_{n-p+1} = a_p \]

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:

\[
\begin{align*}
  a_n &= a_1 \\
  a_{n-1} &= a_2 \\
  & \vdots \\
  a_{(n+1)/2} &= a_{(n-1)/2} \\
  a_{(n+3)/2} &= a_{(n-1)/2} \\
  a_{(n+1)/2} &= a_{(n+1)/2}
\end{align*}
\]

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:

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

Ergebnis:

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

\([n/2]\) 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:

\[
\begin{align*}
\text{Shuffle:} & \quad E_S = 2^n \left(1 - 2^{-P}\right) \\
\text{Butterfly:} & \quad E_B = \begin{cases} 
0 & \text{für } k \leq n \cdot p \\
2^{n-1} & \text{für } k > n \cdot p 
\end{cases} \\
\text{Reversal:} & \quad E_R = \begin{cases} 
2^n \left(1 - 2^{-P}\right) & \text{für } p \leq \lfloor n/2 \rfloor \\
2^n \left(1 - 2^{-\lfloor n/2 \rfloor}\right) & \text{für } \lfloor n/2 \rfloor < p < n 
\end{cases}
\end{align*}
\]

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 \( \lfloor n/2 \rfloor > 1 \) und \( k = n \), dann gilt:

1. \( p = 1 \):
   \[ E_S = E_R = E_B \]

2. \( 1 < p \leq \lfloor n/2 \rfloor \):
   \[ E_S = E_R > E_B \]

3. \( \lfloor n/2 \rfloor < p < n \):
   \[ E_S > E_R > E_B \]

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

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 Permutationsfunktionen 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:

\[ E_{CT}(n,p) = \begin{cases} 
2^n(1-2^{-p}) & \text{für } p \leq \lfloor n/2 \rfloor \\
2^n(1-2^{-[n/2]}) & \text{für } n > p > [n/2]
\end{cases} + p \cdot 2^{n-1} + 2^n(1-2^{-p}) \]

E_R \quad E_{B(2)} \ldots E_{B(n)} \quad E_S

bzw.
\[ E_{CT}(n,p) = \begin{cases} 2^{n-1}(p+4-2^p+2) & \text{für } p \leq \lfloor n/2 \rfloor \\ 2^{n-1}(p+4-2^p+1-2^{\lfloor n/2 \rfloor}+1) & \text{für } n>p>\lfloor n/2 \rfloor \end{cases} \]

näherungsweise gilt:

\[ E_{CT}(n,p) = 2^{n-1}(p + 4) \]

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

Die Stockham FFT hat als Interprozessor-Kommunikation:

\[ E_{ST}(n,p) = E_S + B(n) + B(n-1) + \ldots + B(3) + 2E_S \]

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

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

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

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

\[ E_{PS}(n,p) = E_R + nE_S \]

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

\[ E_{PS}(n,p) = \begin{cases} 2^{n-1}.2(n+1-2^p+1) & \text{für } p \leq \lfloor n/2 \rfloor \\ 2^{n-1}.2(n+1-2^p+2-\lfloor n/2 \rfloor) & \text{für } n>p>\lfloor n/2 \rfloor \end{cases} \]
\[ \text{EPS}(n,p) = 2^{n-1} \cdot 2(n+1) \]

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

\[ E_{CT} = E_{GS} < E_{ST} < E_{PS} \text{ für } p+6 < 2(n+1). \]

Beispiel:

\begin{tabular}{|c|c|c|}
\hline
\text{p = 4:} & \text{p = 8:} \\
\hline
\multicolumn{3}{|c|}{E_{CT} / 2^{n-1}} \\
\multicolumn{3}{|c|}{E_{ST} / 2^{n-1}} \\
\multicolumn{3}{|c|}{E_{PS} / 2^{n-1}} \\
\hline
\text{n = 10:} & 8 & 10 & 22 \\
\text{n = 15:} & 8 & 10 & 32 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|}
\hline
\text{n = 10:} & 12 & 14 & 22 \\
\text{n = 15:} & 12 & 14 & 32 \\
\hline
\end{tabular}

Tabelle 2:

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 \(N = 16384\) 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

\[ n \cdot 2^{n-1}. \]

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 μs. 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

\[T_{CPU} = 40 \text{ ms und } T_{MOV} = 120 \text{ ms} = 3,0 \ T_{CPU}\]

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 % !

\[
\text{Effizienz} = \frac{T_{CPU}}{T_{CPU} + T_{MOV}}
\]

\[
T_{MOV1} : T_{CPU} = 3,0
\]

\[
T_{MOV2} : T_{CPU} = 6,0
\]

\[
\text{Effizienz}_1 = \frac{1}{4,0} = 0,25
\]

\[
\text{Effizienz}_2 = \frac{1}{7,0} = 0,14
\]

\[
\text{Effizienz}_2 = 0,57 \cdot \text{Effizienz}_1
\]

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

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

\[ T_{\text{MUL}} = 1 \text{us und } v_{\text{MOV}} = 1 \text{ MByte/s} \]

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:


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.

4.5 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 einem Prozessor ist:

   \[ T_{\text{BUT}} = a^2 n^{-1} \cdot n = a^2 n \cdot n \]

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

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

   \[ T_{\text{MOV}} = b^2 n^3 (1 + (p/4) - 2^p) \text{ für } 0 < p \leq \lfloor n/2 \rfloor \]

   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-Berechnungs- 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: \[ T_{\text{BUT}}(n,p) = a^2 n^{-p} \cdot n \]

Datentransportzeit: \[ T_{\text{MOV}}(n,p,d) = b^2 d^2 n^3 (1 + (p/4) - 2^p) \]

FFT Berechnungszeit: \[ T_{\text{FFT}} = T_{\text{BUT}} + T_{\text{MOV}} \]

\[ T_{\text{FFT}}(n,p,d) = 2^{n-P} (an + bd(1 + (p/4) - 2^p)) \]

Wirkungsgrad: \[ e = \frac{T_{\text{SEQ}}}{T_{\text{PAR}}} = \frac{T_{\text{BUT}}}{(T_{\text{BUT}} + T_{\text{MOV}})} \]
\[ e(n,p,d) = n \cdot \left[ n + cd(1+(p/4)-2^{-p}) \right], \text{ mit } c = b/a \]

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:


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. \( T_{\text{FFT}} \) kennzeichnet die Berechnungszeit ohne Daten Reversal am Ende der FFT
3. \( T_{\text{MOV}} \) 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 \( T_{\text{BUT}} \)
2. Der Speedup \( S_p \)
3. Der Wirkungsgrad \( e \)
4. Das Verhältnis von Daten-Transportzeit zu Gesamtzeit \( T_{\text{MOV}}/T_{\text{FFT}} \)

Das Ergebnis der Berechnungen ist in Tabelle 3 zusammengestellt:
<table>
<thead>
<tr>
<th>Gesamtzeit TFFT in ms:</th>
<th>Gesamtzeit TMOV in ms:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zahl der Prozessoren</td>
<td>Zahl der Prozessoren</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>2 944</td>
</tr>
<tr>
<td>32</td>
<td>7 296</td>
</tr>
<tr>
<td>64</td>
<td>16 768</td>
</tr>
<tr>
<td>128</td>
<td>40 064</td>
</tr>
<tr>
<td>256</td>
<td>93 312</td>
</tr>
<tr>
<td>512</td>
<td>214 592</td>
</tr>
<tr>
<td>1024</td>
<td>478 080</td>
</tr>
<tr>
<td>2048</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Butterfly-Berechnungszeit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zahl d. Punkte</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speedup/Wirkungsgrad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zahl d. Punkte</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>1024</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verhältnis Daten-Transportzeit zu Gesamtzeit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zahl d. Punkte</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
</tr>
</tbody>
</table>

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

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

**Wirkungsgrad:** $e = n / \left[ n + 2.01(1+(p/4)-2^{-p}) \right]$
Die Güte des Modells kann anhand der Abweichungen in den graphischen Darstellungen der gemessenen und berechneten Größen beurteilt werden. Für die Gesamtausführungszeit $T_{\text{Tot}}$ ist in der graphischen Darstellung keine Abweichung sichtbar (Bild 4.14):

![Graphik](image)

Bild 4.14:
Berechnungszeit der parallelen Gentleman-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:

![Graphik](image)

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 Gentleman-Sande FFT auf 4 Prozessoren in Meßwerten und Rechnung (durchgezogene Linien). \( n = \log_2 N \).

Bild 4.17:
Wirkungsgrad der parallelen Gentleman-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

\[
S_p = \frac{2^P n}{[n + cd(1 + (p/4) - 2^P)]}
\]

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:

\[
p = 4 \left[ \frac{n}{cd} - 1 \right]
\]

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:

$$d = 2 \left( 1 + 2 + \ldots + \frac{(P-1)}{2} \right).$$

Daraus erhält man:

$$d = \frac{(P^2 - 1)}{4}.$$ 

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

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

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:

$$d = 2 \left( 1 + 2 + \ldots + \frac{(P-2)}{2} \right) + P/2$$

Man erhält durch Summation:

$$d = \frac{P^2}{4}.$$ 

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:

$$
\begin{align*}
\text{a) } P & \text{ gerade:} & d_{\text{Ring}} &= \frac{P^2}{4 (P-1)} \\
\text{b) } P & \text{ ungerade:} & d_{\text{Ring}} &= \frac{(P^2 - 1)}{4 (P-1)}
\end{align*}
$$

Die Bechung 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:

\[ d_k = k \sum_{p \text{ über } k} \]

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

\[ d = \sum_{k}^{p} k \sum_{p \text{ über } k} \]

Nach zweifacher Substitution von \( p-1 \) durch \( p' \) und \( k-1 \) durch \( k' \) erhält man:

\[ d = p \sum_{0}^{n-1} [p' \text{ über } k'] \]

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

\[ d = p \times 2^{p-1} \]

Jeder Prozessor \( P_0 \) hat \( p-1 \) Nachbarn, so daß sich für die mittlere Entfernung ergibt:

\[ d_{\text{Hypercube}} = p \times 2^{p} / [2^{p} - 1] \]

Bei \( unendlich vielen Prozessoren \) streben die mittleren Entfernungen gegen die Grenzwerte:

\[ d_{\text{Cube}} \rightarrow p/2 \]
\[ d_{\text{Ring}} \rightarrow p/4 \]

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.
Zusammenfassung:


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 Löszen 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 eliminiert, bei gleichzeitiger Reduktion der Zahl der abzuspeicherden Datenmenge möglich.

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

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:

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

(= 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 aneinandergereihteten Parabelbögen 3. Ordnung besteht [Rein1]. Diese zweimal stetig differenzierbar aneinandergereihteten 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:

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

für \( x \) 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

\[ M \cdot c = y \]

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

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

(äquidistante Stützstellen)

Zusammenfassend läßt sich sagen, daß

\[ a_i = f_1(c_{i-1}, c_i, c_{i+1}) \]
\[ b_i = f_2(a_{i+1}, a_i, c_{i+1}, c_i) \]
\[ d_i = f_3(c_{i+1}, c_i) \]

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 Gleichungssystem 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.
Lösung: $Mc = y$, daraus: $c, d \Rightarrow a \Rightarrow b$

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.

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-Ap proximation

Sei \( s \) ein Spline und

\[
\mathbf{r} = \mathbf{y} - \mathbf{s}, \text{ mit } \mathbf{r} = (r_0, r_1, \ldots, r_{n-1})
\]

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

\[
\| \mathbf{r} \|_2^2 = r_0^2 + r_1^2 + \ldots + r_{n-1}^2
\]

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

\[
r_i = y_i - p_0B_0(x) - p_1B_1(x) - \ldots - p_{n-1}B_{n-1}(x).
\]

Und für alle Elemente von \( r \):

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

\[
\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{bmatrix}
\]

\[
\mathbf{B} = \begin{pmatrix}
B_0(x_0) & B_1(x_0) & \ldots & B_{n-1}(x_0) \\
B_0(x_1) & B_1(x_1) & \ldots & B_{n-1}(x_1) \\
\vdots & \vdots & \ddots & \vdots \\
B_0(x_{m-1}) & B_1(x_{m-1}) & \ldots & B_{n-1}(x_{m-1})
\end{pmatrix}
\]
\[ \mathbf{p} = \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{pmatrix} \]

Weiterhin ist:

\[ \| r \|_2^2 = \| y - \mathbf{Bp} \|_2^2 = y^T y - 2 p^T \mathbf{B}^T \mathbf{B} y + p^T \mathbf{B}^T \mathbf{Bp} \]

* Bemerkung:
1.) \( \mathbf{B}^T \) ist die transponierte Matrix,
2.) \( \| y - \mathbf{Bp} \|_2^2 \) wird analog zu \((a-b)^2 = a^2 - 2ab + b^2\) berechnet.

Nun ist eine notwendige und hinreichende Bedingung für

\[ \| r \|_2 = \text{MIN}, \]

wenn

\[ \frac{\partial}{\partial p_0} F = \frac{\partial}{\partial p_1} F = \ldots = \frac{\partial}{\partial p_{n-1}} F = 0, \]

für \( F = \| r \|_2^2 \).

Dies ist dann der Fall, wenn

\[ \frac{\partial}{\partial p_i} F = 0 - 2 e_i^T \mathbf{B}^T y + e_i^T \mathbf{B}^T \mathbf{B} p + p^T \mathbf{B}^T \mathbf{B} e_i \]

* Bemerkung:
1.) \( e_i \) ist der i. Einheitsvektor
2.) \( \frac{\partial}{\partial x} f \) erfolgt gemäß der Produktregel analog zu \( \frac{\partial}{\partial x} f g = f' g + f g' \),

mit

\[ p^T \mathbf{B}^T \mathbf{B} e_i = e_i^T \mathbf{B}^T \mathbf{B} p \]

wird

\[ \frac{\partial}{\partial p_i} F = 2 e_i (- \mathbf{B}^T y + \mathbf{B}^T \mathbf{B} p ) = 0, \]

daraus folgt:

\[ \mathbf{B}^T \mathbf{B} p = \mathbf{B}^T y. \]

 Damit ist der Algorithmus für eine Spline-Glättung mit Datenreduzierung hergeleitet.
5.5 Die Parallelisierung der die Messdaten reduzierenden und glättenden Spline Approximation

Seien

\[
\mathbf{y} = \begin{pmatrix}
    y_0 \\
    y_1 \\
    \vdots \\
    y_{m-1}
\end{pmatrix}
\]

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

\[
s(x) = p_0 B_0 (x) + p_1 B_1 (x) + \ldots + p_{n-1} B_{n-1} (x)
\]

der approximierende Spline mit den Basisfunktionen

\[
B_i(x) = \begin{cases}
0 & \text{für } x' \leq 0 \text{ und } x' = (x - x_i)/(x_{i+1} - x_i) \\
\frac{x^3}{6} & \text{für } 0 < x' \leq 1 \\
\frac{1}{2} (x' - 1) & \text{für } 1 < x' \leq 2 \\
B_i (4 - x') & \text{sonst}
\end{cases}
\]

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

\[
\mathbf{B}^T \ast \mathbf{B} \ast \mathbf{p} = \mathbf{B}^T \ast \mathbf{y}, \text{ mit }
\]

\[
\mathbf{B} = \begin{pmatrix}
    B_0(x_0) & B_1(x_0) & \ldots & B_{n-1}(x_0) \\
    B_0(x_1) & B_1(x_1) & \ldots & B_{n-1}(x_1) \\
    \vdots & \vdots & \ddots & \vdots \\
    B_0(x_{m-1}) & B_1(x_{m-1}) & \ldots & B_{n-1}(x_{m-1})
\end{pmatrix}
\]

\[
\mathbf{p} = \begin{pmatrix}
    p_0 \\
    p_1 \\
    \vdots \\
    p_{n-1}
\end{pmatrix}
\]

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. \( B^T \) mit dem Vektor \( y \) der Meßwerte multiplizieren
3b. Parallel dazu \( B^T \) mit \( B \) multiplizieren
4. Lineares Gleichungssystem lösen

Die Approximationsfunktion \( s \) ergibt sich dann zu:

\[
s = B^*p.
\]

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:
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:

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

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 } A^T \), bei der die Zeilen mit den Spalten vertauscht sind, und die deshalb vom Typ } [M,N] \) ist, sieht das Speicherbild folgendermaßen aus:

\[
\begin{array}{|c|c|}
\hline
\text{kleine Adresse:} & \text{Inhalt} \\
\hline
a_{11} & \\
a_{21} & \\
\vdots & \\
a_{N1} & \\
a_{12} & \\
a_{22} & \\
\vdots & \\
a_{N2} & \\
a_{1M} & \\
a_{2M} & \\
\vdots & \\
a_{NM} & \\
\hline
\end{array}
\]

\[
\text{Bild 5.5:} \\
\text{Spaltenweise abgespeicherte Matrix } A^T.
\]
Wenn man nun den Zeiger $p^T$ auf die Adressen der Elemente der transponierten Matrix betrachtet,

$$p^T = j_mj_{m-1}...j_1i_{n-1}...i_1,$$

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:**

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 Matrizen-Transposition 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 Unshuffle-Stufen zwischen den Butterfly-Operationen besteht. Und analog zur Pease-FFT kann der Signalfußgraph der Matrixtransposition zeilen- und/oder spaltenweise parallelisierter werden. Bei einer Matrix von \(N\times M\) Elementen ergeben sich \(NM\times \log_2(N)\) Datentransporte.

<table>
<thead>
<tr>
<th>Zahl d. Prozessoren</th>
<th>Berechnungszeit</th>
<th>Topologie</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(O(NM\times \log M))</td>
<td>-</td>
</tr>
<tr>
<td>(\log M)</td>
<td>(O(NM)) (pipelined)</td>
<td>Kette</td>
</tr>
<tr>
<td>(N)</td>
<td>(O(M\times \log M))</td>
<td>shuffle</td>
</tr>
<tr>
<td>(NM\times \log M)</td>
<td>(O(1)) (pipelined)</td>
<td>&quot;*&quot;, mehrfach hintereinander</td>
</tr>
<tr>
<td>(N/k, k =1,2,...,N)</td>
<td>(O(kM\times \log M))</td>
<td>shuffle</td>
</tr>
</tbody>
</table>

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


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 Ergebniszelement die Berechnung eines Skalarprodukts. Alle Skalarprodukte können gleichzeitig berechnet werden da sie nicht voneinander abhängen. Im Falle der Matrix-Vektor-Multiplikation \([n, m]\times [m]\) können die \(m\) Skalarprodukte von bis zu \(m\) Prozessoren gleichzeitig berechnet werden. Bei der Matrix-Matrix-Multiplikation sind \(n\times k\) 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.

\[
s_0 + s_1 + s_2 + s_3 + s_4 + s_5 + \ldots + s_{p-1}
\]

\[
s_0'' \quad s_2'' \quad s_4'' \quad \ldots \quad s_{p-2''}
\]

\[
\ldots \quad \ldots
\]

\[
\text{Resultat}
\]

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:

\[
\rightarrow P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_{p-1}
\]

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:

<table>
<thead>
<tr>
<th>Datenein/Ausgabe:</th>
<th>( O(n) + O(p) ) Schritte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datenverarbeitung:</td>
<td>( O(1+\log(p)) ) Schritte</td>
</tr>
</tbody>
</table>

Die Multiplikationen wurden dabei als \( \text{ein} \) Schritt gezählt. Die Datenein/Ausgabe dominiert also in diesem Beispiel. Das Beispiel läßt sich unter der Voraussetzung verallgemeinern, daß ein Datentransport 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 \( \text{nicht} \) 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 \( \text{Verarbeitung} \) 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 Ausnutzung 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 Größenordnung der Berechnungsdauer für die Matrix-Vektor-Multiplikation für verschiedene Prozessorzahlen und Topologien in einer Tabelle 2 angeben. Eine andere Möglichkeit als das Umschalten der Topologie, ist die Datenein/Ausgabe und Verarbeitung \( \text{überlappend} \) 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

\([m,n] \cdot [n] \rightarrow [m]\)

<table>
<thead>
<tr>
<th>Zahl d. Prozessoren</th>
<th>Zeitkomplexität</th>
<th>Topologie</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(O(mn))</td>
<td>-</td>
</tr>
<tr>
<td>(m)</td>
<td>(O(n))</td>
<td>Kette</td>
</tr>
<tr>
<td>(mn)</td>
<td>(O(\log n)) ohne I/O</td>
<td>(m) Ketten</td>
</tr>
<tr>
<td>(mn)</td>
<td>(O(n)) mit I/O</td>
<td>(m) Ketten</td>
</tr>
<tr>
<td>(mn)</td>
<td>(O(\log n)) mit I/O</td>
<td>(m) Ketten</td>
</tr>
</tbody>
</table>


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

\[C = A \ast B, \text{ mit}\]

\[c_{ik} = \sum_{l=0}^{m-1} a_{il} \ast b_{lk}\]

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:

![Matrixmultiplikation mit Hilfe eines systolischen Arrays am Beispiel zweier [4,4] Matrizen.](image-url)

Die Anordnung 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

\[
c_{ik} = \sum_{l=0}^{m-1} a_{il} \cdot b_{lk},
\]

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

\[
A[n,m] \cdot B[m,k] \rightarrow C[n,k]
\]

<table>
<thead>
<tr>
<th>Zahl d. Prozessoren</th>
<th>Zahl der Operationen</th>
<th>Topologie</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O(nk)</td>
<td>-</td>
</tr>
<tr>
<td>n</td>
<td>O(km)</td>
<td>Kette</td>
</tr>
<tr>
<td>nk</td>
<td>O(m) systolisch</td>
<td>Gitter</td>
</tr>
</tbody>
</table>

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

\[
A^*x = b
\]

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:

\[
\begin{align*}
\text{vorher:} \\
\sum_{i=1}^{n} a_{1i} x_i + a_{12} x_2 + \ldots + a_{1n} x_n &= b_1 \\
\sum_{i=1}^{n} a_{2i} x_i + a_{22} x_2 + \ldots + a_{2n} x_n &= b_2 \\
&\vdots\\
\sum_{i=1}^{n} a_{ni} x_i + a_{n2} x_2 + \ldots + a_{nn} x_n &= b_n
\end{align*}
\]

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

\[
\begin{align*}
\text{nachher:} \\
\sum_{i=1}^{n} \hat{a}_{ii} x_i + \sum_{i=1}^{n} \hat{a}_{1i} x_i &= \hat{b}_1 \\
\sum_{i=1}^{n} \hat{a}_{2i} x_i + \sum_{i=1}^{n} \hat{a}_{2n} x_n &= \hat{b}_2 \\
&\vdots\\
\sum_{i=1}^{n} \hat{a}_{ni} x_i &= \hat{b}_n
\end{align*}
\]

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

\[
\hat{a}_{ij} = a_{ij} - (a_{kj}/a_{kk})a_{jk} \quad \text{für alle } i,j,k \text{ mit } k < i, j \geq k
\]

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.

\[
\begin{align*}
\begin{bmatrix}
a_{11} & \cdots & x_1 \\
a_{22} & \cdots & x_2 \\
\vdots & & \vdots \\
a_{nn} & \cdots & x_n
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
\vdots \\
0
\end{bmatrix}
&= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{bmatrix}
\end{align*}
\]

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 gegenübergestellt:

<table>
<thead>
<tr>
<th>Zahl d. Prozessoren</th>
<th>Zeitabhängigkeit</th>
<th>Topologie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss</td>
<td>(O(n^3))</td>
<td>-</td>
</tr>
<tr>
<td>Gauss-Jordan</td>
<td>(O(n^3))</td>
<td>Kette *</td>
</tr>
<tr>
<td>(n)</td>
<td>(O(n^2+n))</td>
<td>(O(n^2))</td>
</tr>
<tr>
<td>(n^2)</td>
<td>(O(n+n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gitter</td>
</tr>
</tbody>
</table>

* 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.


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?


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:


2. Weiterhin ist neben einer Minimierung der Datenmenge 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 \( O(N \log N) \) 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.

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.


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 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 zwischen-zuspeichern.

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:

2. Auf dem Gebiet der Sprachen sind selbst parallelisierende Compiler vorstellbar, die die Partitionierung 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 innenwohnende Parallelität besser auszudrücken, so daß Fehler wie "gegenseitige Verklemmung" (Deadlock) oder "Nicht-Terminierung" (Liveloop) 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


\[
N = nk
\]

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 \( N \) physischen Kanälen unterscheiden, ähnlich wie dies in einem virtuellen Speichersystem mit virtuellen und physischen Seiten geschieht. Analog zur "Seitenersetzung" beim Speicherzugriff auf eine physisch 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 vorgenommen 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.


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 \( v_{\text{DAT}}(n) \):

\[ v_{\text{DAT}}(n) = \frac{n}{T(n)} = \frac{n}{T_s + nT_B}. \]

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

Man sieht, daß die effektive Geschwindigkeit \( v_{\text{Dat}} \) 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 \( T_s = 4 \mu s \), und \( T_B = 550 \text{ ns} \). 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 \( \mu s \) 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


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

Die Synchronisierungsplatine hat die Aufgaben:

1. Abtasten der asynchronen Linksignale, um diese taksynchron in das Koppelnetz einspeisen zu können;
2. Zwischenspeichern der Daten- und Rückmeldepakete, während der
Umkonfigurierung des Koppelnets;

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 Doppeleuropakarten, 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.

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.


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.

1. Prozessorkarte
Diese Karte enthält vier unabhängige Module, von denen jedes besteht aus:
- Transputer T414-15, T414-20 oder T800-20
- statischer Speicher a 128 KBytes mit 330 ns Zykluszeit bei 15 MHz Transputer bzw. 250 ns bei 20 MHz Transputer.
- 4 bidirektionale serielle Kanäle mit 10 oder 20 Mbit/s Übertragungsrate.
- Linkadapter mit 10 oder 20 Mbit/s Übertragungsrate
- max. 2 MFLOPS Gleitkomma-Rechenleistung bei einfach genauer Arithmetik (nur T800) und max. 10 MIPS Durchsatz.

Allen Moduln gemeinsam sind:
- 2x(16+4) Leitungstreiber für die Kommunikationskanäle (bidirektional)
- Taktszillator (40 MHz): reset, error und analyse Schaltung.

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

3. Koppelnetzkarte
- 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.
- Transputer T414-20 mit 128 KBytes RAM.
- 4x Linkadapter mit 10 oder 20 Mbit/s.
- 2x(16+4) Leitungstreiber, Oszillator, reset, error und analyse logik.
- OCCAM I Programm zum Setzen der Koppelnetzschalter in 40 μs. (In OCCAM II sind 10us möglich).
- 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

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 widersprächen. 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:


**Legende:**
- : Draht
- : INMOS Link
- : Bus
- : Linkadapter

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

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:

```plaintext
a ? x
```

Die Ausgabe wird geschrieben als

```plaintext
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:

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 Topologien, 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 Links signale, damit diese taktsynchron in das Koppelnetz eingespeist werden können.

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


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 $1,4 \times 10^{13}$ Bits ein Bitfehler auftritt ($16 \times 10^7$ Bits/s * 3600s/h * 24h = $1,4 \times 10^{13}$).

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.


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

Das zweite Problem (hohe Abtastrate) wurde auf die folgende Weise gelöst:

![Zeitdiagramm](image)

1. Abtastzeitpunkt: Taktflanke und Startbit "gleichzeitig"

2. Abtastzeitpunkt: Taktflanke 50 ns nach dem Startbit

Abtastzeitpunkte: 25 ns, nach ansteigender Flanke von delinp. Dann alle 100 ns später.

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 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


1. Das Link war in der Umschaltpause inaktiv.
2. Es wurde ein Datenpaket gesendet
3. Es wurde ein Rückmeldepaket gesendet.

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.
Zustands-Codierung des Automaten:

000 Stop Bit ausgeben (= Flag Bit, falls Flag = L) Transm := L
Hier ändert sich nur 1 Bit d.h., nur ein Flip Flop tastet das Signal ab!
(= Single Bit Transition)
dinp := H
Startbit
Waitstart
dinp abtasten, Transm := H
dinp ist der um 25 ns verzögerte input.
Startbin ausgeben, Transm := H

001 delinp := L
Waiting
dinp abtasten, Transm := delinp
Flagbit
delinp abtasten, Transm := H
Waitflag

010 Flag Bit ausgegeben, falls Flag = H Transm := H
delinp abtasten, Transm := H

011 Data Bit 0-7 ausgeben Transm := H
delinp abtasten, Transm := L
Stopbit

100
Waitflag

101
Bit 0
count > 0

110
Wait 0

111

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
drei möglichen Pakete auf dem Link zwischengespeichert und nach der Umschaltsphase ein
eventuell gefüllter Linkspeicher wieder korrekt ausgelesen wird. Ein Problem tritt für den Fall auf,
dazu bei der Umschaltung 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
der Automaten, der 32 verschiedene Zustände annehmen hat.
Bild A.1.10:
Der Ein-/Ausles-Automat. Das Problem dieses Automaten war es, ohne Zeitverlust in jedem Zustand des Auslesens ein neues Paket einlesen zu können.


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:

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

Fourier Analyse

Fourier Synthese

In diesen Integralgleichungen wurde der Eichfaktor 1/2π 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 Fourieranalyse 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 \( s(t) \):

\[
\begin{array}{c}
\text{Analyse} \\
\downarrow \\
S(f) \\
\downarrow \\
\text{Synthese}
\end{array}
\]

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 eine numerischen Auswertung der Integrale in der Praxis das unendlich große Integrations Intervall entgegenstellt. 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.

![Diagram](image)

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:

\[
S(nf_1) = \int_{-\infty}^{+\infty} s(t) e^{-j2\pi nf_1} dt = \sum_{n=-\infty}^{+\infty} S(nf_1) e^{+j2\pi nf_1} t
\]

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

Aufgrund des oben angegebenen Satzes gilt, daß ein periodisches Spektrum ein diskretes
Zeitsignal zur Folge hat!

\[ S(f) \text{ wird periodisch fortgesetzt. Dann wird das Zeitsignal diskret.} \]

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

\[ S(nf_1) = \sum_{m=0}^{T_1} s(mT_2).e^{-j2\pi mf_1T_2} \]
\[ s(mT_2) = \sum_{n=0}^{fQ} S(nf_1).e^{+j2\pi nmf_1T_2} \]

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:

\[ S(nf_1) = \sum_{m=0}^{T_1} s(mT_2).e^{j(2\pi/N)nm} \]
\[ s(mT_2) = \sum_{n=0}^{fQ} S(nf_1).e^{+j(2\pi/N)nm} \]

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.


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 (=\( f_0 \)) von \( s(t) \) ist.

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

\[ s(0) \]
\[ T_1 \]
\[ 1/2f_0 \]

\[ t \]

\[ T_1 \]
\[ 1/T_1 \]

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 \( \text{rect}(f_0, f) \) beschreiben.

\[ S(f) \rightarrow S(f) \cdot \text{rect}(f_0, f) \]
Anmerkung:
rectan(f₀,f) soll eine Rechteckfunktion sein, die identisch Null für f > f₀ ist.

Nach dem 2. Teil des Abtasttheorems kann rectan(f₀,f) ersetzt werden durch rectan(f₂/2,f), da S(f) = 0 für f₀ ≤ f ≤ f₂/2 ist.

Damit erhält man:

\[ S(f) \rightarrow S(f) \cdot \text{rectan}(f₂/2,f) \]

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

\[ s(t) = \sum_{n = -\infty}^{\infty} s(nT₂) \frac{\sin(2\pi(f₂/2)(t - nT₂))}{2\pi(f₂/2)(t - nT₂)} \]

(Abtasttheorem 3. Teil)

Anmerkung:

1. Ein rechteckförmiges Spektrum der Bandbreite f₀ hat eine \( \sin x / x \) Funktion gemäß \( \sin(2\pi f₀ t) / (2\pi f₀ t) \) zur Folge.

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

\[ f \ast g = \sum_{n = -\infty}^{+\infty} f(t) \cdot g(t - n) \]

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


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 nT₂.
\[ T_0 = 1/2f_0 = 1, \text{ Nullstelle von } \sin(x)/x \]
\[ f_0 = \text{Bandbreite des Spektrums} \]
\[ T_1 = 1/f_1: \text{Periode des Zeitsignals} \]
\[ f_1 = \text{Abstand der Spektrallinien} \]
\[ T_2 = 1/f_2: \text{Abstand der Abtastwerte} \]
\[ f_2/f_1 = T_1/T_2 = N = \text{Zahl der Abtastwerte} \]

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:

\[
s(mT_2) \rightarrow s_m = \sum_{n=0}^{N-1} S_n \cdot e^{j(2\pi/N)n m} \quad \text{Synthese} \]

\[
S(nf_1) \rightarrow S_n = \sum_{m=0}^{N-1} s_m \cdot e^{-j(2\pi/N)n m} \quad \text{Analyse} \]

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 Zeilenektors

\[
(e^{j(2\pi/N)0m}, e^{j(2\pi/N)1m}, ..., e^{j(2\pi/N)(N-1)m})
\]

mit dem Spaltenvektor
\[
\begin{pmatrix}
  s_0 \\
  s_1 \\
  \vdots \\
  s_{N-1}
\end{pmatrix}
\]
aufgefaßt werden.

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

\[
\begin{align*}
\text{Synthese} & : \quad \mathbf{s} = W^{-1} \cdot \mathbf{S} \quad \text{mit} \\
\text{Analyse} & : \quad \mathbf{S} = W \cdot \mathbf{s} \quad \text{mit}
\end{align*}
\]
\[
W^{-1} = (w_{nm}^{-1}) \quad \text{und} \\
W = (w_{nm}) \quad \text{und}
\]
\[
w_{nm}^{-1} = \frac{1}{w_{nm}} = e^{-j(2\pi/N)nm} \\
w_{nm} = e^{j(2\pi/N)nm}
\]

\[W\] ist eine Matrix vom Typ \([N,N]\).

Bild A.2.6: 
Die diskrete Fourier-Transformation ordnet einem Vektor \(\mathbf{s}\) von Abtastwerten einen Vektor \(\mathbf{S}\) 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:

\[T_{\text{DFT}} = N^2. \text{ Zeit pro Multiplikation} = O(N^2)\]

Die quadratische Abhängigkeit der Berechnungsduauer 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 \([N/2 \times N/2]\), als eine Matrix der Größe \([N \times N]\) 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 Vektorlängenschrittweise 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 stufenweise aus 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 $[N \times N]$ auf $[N/2 \times N/2]$

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 der Punkte 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 jedes für sich transformiert und ihre Spektren mit Hilfe sogenannter Drehfaktoren zum Spektrum aller Punkte addiert. Für den Fall einer Unterteilung in gerade und ungerade Punkte hat man für die Analyse:

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

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

\[
\text{Summation über gerade Indizes} \quad \text{Summation über ungerade Indizes}
\]

\[
= \text{für } 0 \leq n \leq (N/2)-1:
\]

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

\[
\text{Spektrum der geraden Punkte} \quad \text{Drehfaktor} \quad \text{Spektrum der ungeraden Punkte}
\]

bzw.

\[
\text{für } N/2 \leq n < N:
\]

- 120 -
Das Raster des Index \( m \) verhält sich dabei folgendermaßen:

- Ursprüngliches Raster: \( m = 0, 1, 2, 3, \ldots, m, \ldots, N-1 \), insgesamt \( N \) Stück
- Gerades Raster: \( 2m = 0, 2, 4, \ldots, N-2 \) \( N/2 \)
- Ungerades Raster: \( 2m + 1 = 1, 3, 5, \ldots, N-1 \), \( N/2 \)

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 \( N = 2^m \) Punkte sind dazu \( m \) 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 so genannten Reversal Permutation durchführt.

A.2.7 Zusammenfassung


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

\[ B_i(x) = \begin{cases} 
0 & \text{für } x' \leq x_i \text{ und } x' = (x - x_i)/(x_{i+1} - x_i) \\
\frac{x'^3}{6} & \text{für } x_i < x \leq x_{i+1} \\
\frac{1}{6}(-3x'^3 + 12x'^2 - 12x' + 4) & \text{für } x_{i+1} < x \leq x_{i+2} \\
B_i(4 - x') & \text{sonst}
\end{cases} \]

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.](image-url)
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 aneinandergereiht 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: \(k_i = n - 4\).

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:

![Diagramm der Raster der Stützstellen und der Meßwerte](image)

Bild A.3.2:
Das Raster der Stützstellen und der Meßwerte für \(n = m = 5\).
Für gleiche Intervallänge gilt:

\[(k_i + 1) \cdot 1 = a \cdot m\]

bzw.

\[a = (n-3)/m, \text{ mit } m > 3.\]

Beispiel:

\[m = 5\] Stützstellen und Interpolation, \(\Rightarrow n = 5, k_i = 1, a = 0,5\]

\[m = 6\] Stützstellen und Approximation mit \(n = 5, \Rightarrow k_i = 1, a = 0,4\]

Bild A.3.3:
Das Raster der Meßwerte und der Basis-Splines bei Interpolation und Approximation.

Im unteren Beispiel wird das Raster der Meßwerte feiner, die Zahl der Basis Splines 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 Linearerstellung für 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(nt) 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:

\[ (\sum f^2(x))^1/2 = \text{MIN}, \]

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.

\[ \| r \|_2 = \text{MIN}. \]

Literaturverzeichnis

Literaturhinweise zur Architektur von MULTITOP


- 129 -
Literaturhinweise zum modularen Koppelnetz


[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


Literaturhinweise zur Spline Approximation


