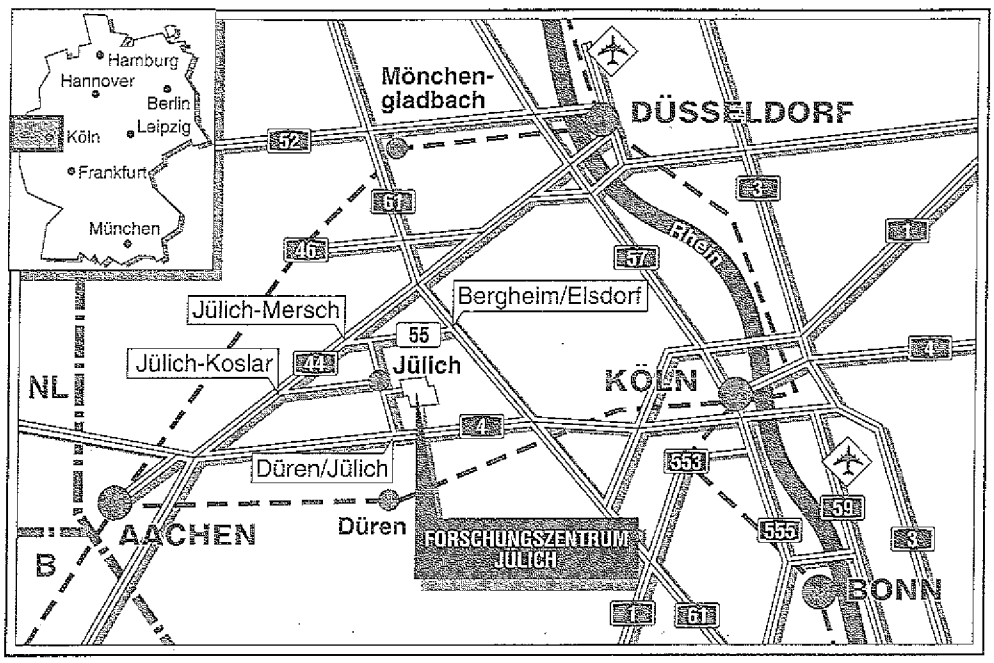
FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Zentralinstitut für Angewandte Mathematik

Aufbau und Eigenschaften von Hypercube-Rechnern und Untersuchung von Datentransferalgorithmen

Stefan Rödder



Berichte des Forschungszentrums Jülich ; 2428
ISSN 0366-0885
Zentralinstitut für Angewandte Mathematik Jül-2428

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

Aufbau und Eigenschaften von Hypercube-Rechnern und Untersuchung von Datentransferalgorithmen

Stefan Rödder

Zusammenfassung

Hyercube-Rechner fallen in die Klasse der aus Mikroprozessoren aufgebauten Multicomputer mit verteiltem Speicher, deren Entwicklung durch fortschreitende Erfolge auf dem Gebiet des VLSI-Designs vielversprechend erscheint. Allgemein zeichnen sich Rechner dieser Klasse durch einfachen und kostengünstigen Aufbau und gute Erweiterbarkeit aus. Jegliche Kommunikation zwischen den Prozessoren wird durch Message-Passing abgewickelt, wodurch Lokalität in Berechnungen effektiv genutzt und die bei Systemen mit gemeinsamem Speicher auftretenden Speicherkonflikte vermieden werden.
Hypercube-Rechner bestehen aus Prozessoren, die nach Art der Hypercube-Struktur untereinander direkt verbunden sind. Diese Struktur zeichnet sich durch eine Vielzahl gut untersuchter Eigenschaften aus, erlaubt die Abbildung vieler Problemstrukturen und die effektive Durchführung vieler häufig benötigter Kommunikationsarten.
Die Arbeit geht auf Eigenschaften der Hypercube-Struktur ein und stellt die existierenden Hypercube-Rechner verschiedener Firmen und Forschungsgruppen vor. Den Schwerpunkt der Arbeit bildet eine Untersuchung verschiedener Arten von Interprozessorkommunikation, deren effektive Durchführung einen entscheidenden Einfluß auf den erzielbaren Speedup hat. Untersucht werden z.B. Unicast-, Multicast-, Broadcast- und Scatter-Algorithmen.

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

1 Einleitung

Seit dem Bau der ersten elektronischen Digitalrechner in den vierziger Jahren hat sich die Leistungsfähigkeit dieser Anlagen sowohl durch Verbesserung der einzelnen Bauelemente als auch durch Weiterentwicklung verschiedenster Konzepte für Rechnerarchitektur und Programmierung drastisch vergrößert. Weiteren Geschwindigkeitssteigerungen durch Entwicklung schnellerer Schaltelemente ist jedoch mit der Lichtgeschwindigkeit eine physikalische Schranke gesetzt, der man sich nur noch unter großen und weiter steigenden Kosten annähern kann.
Es existiert jedoch eine Vielzahl komplexer Probleme, die sich teilweise oder ganz einer Lösung in akzeptabler Zeit mit Hilfe heutiger digitaler Rechenanlagen entziehen. Einige Beispiele solcher Probleme sind [Quin88]

- langfristige, genaue Wettervorhersage,
- Windkanalmodellierungen,
- Simulation chemischer Prozesse,
- Festkörperberechnugen,
- Berechnung seismischer Modelle und
- Probleme aus der künstlichen Intelligenz
  (z. B. Mustererkennung, Verarbeitung natürlicher Sprache).

Neben Verbesserungen bestehender Algorithmen versprechen neue architektonische Konzepte für die Zukunft das einzige Mittel zur Erreichung der für diese und viele andere Probleme benötigten Rechenleistung zu sein. Ein solches Konzept ist die *Parallelverarbeitung*, d. h. die gleichzeitige Ausführung von Prozessen, die zusammen ein einziges Problem lösen. Rechner, die Parallelverarbeitung auf Programmebene durch Bereitstellung mehrerer einfacher oder komplexer Prozessoren gestatten, werden Parallelrechner (*parallel computers*) genannt [Quin88, Dunc90].
Zwar läßt sich durch den Einsatz von Parallelrechnern eine teilweise erhebliche Beschleunigung (*Speedup*) der Lösung eines Problems erzielen, jedoch erfordert die Programmierung von Parallelrechnern im Vergleich zu herkömmlichen, seriellen Rechnern einen erhöhten Aufwand.

Zum einen muß ein Algorithmus für ein gegebenes Problem auf seine parallelisierbaren Teile hin untersucht werden. Dazu ändert man entweder einen schon bekannten seriellen Algorithmus so ab, daß er in Teile zerfällt, die parallel ausgeführt werden können, oder man entwickelt einen völlig neuen parallelen Algorithmus. Letzteres ist besonders dann sinnvoll, wenn die gegebenen seriellen Algorithmen durch einen hohen Anteil seriell auszuführender Anweisungen nur in unzureichender Weise parallelisierbar sind und daher die zu erwartende Beschleunigung durch Parallelisierung nur gering ist [Amda67].

Zum anderen muß man Prozesse, die sowohl identisch als auch verschieden sein können, auf die gegebenen Prozessoren durch Lastverteilung (*load balancing*) möglichst so verteilen, daß eine gleichmäßige Auslastung aller Prozessoren erreicht und somit die gesamte Rechenzeit minimiert wird. Zu beachten ist dabei besonders, daß die entstehenden Kosten (*Overhead*) für Kommunikation zwischen den Prozessoren und Verwaltung der Prozesse minimiert wird, denn der Speedup S läßt sich durch den Overhead σ und die Anzahl der beteiligten Prozessoren N abschätzen durch $S \leq \frac{N}{1+\sigma}$.

Der erste Parallelrechner wurde 1958 am National Bureau of Standards Pilot gebaut und umfaßte drei unabhängig voneinander operierende Rechner, die zusammenarbeiten konnten [Ensl77]. Der Anfang der siebziger Jahre in Betrieb genommene Illiac IV [Bouk&&72] hatte bereits 64 Prozessoren, die in einem zweidimensionalen Feld verbunden waren.

Schon 1963 wurde jedoch ein Parallelrechner auf der Basis eines n-dimensionalen *Hypercubes* vorgeschlagen [Sq&Pa63, Stou90]. 1977 wurde dann eine Idee für den Aufbau eines, die Verbindungsstruktur des Hypercube simulierenden, speziellen Schaltnetzes erstmalig veröffentlicht (*indirect binary n-cube* [Peas77]), die als Grundlage für die Ankündigung des ersten *Hypercube-Rechners* durch IMS Associates genommen wurde [Mill75, Ho&Je88]. Dieser Versuch verlief zwar erfolglos, wurde aber Anfang der achtziger Jahre durch den *Cosmic Cube* des California Institute of Technology (Caltech) [Seit85] und später durch den ersten verfügbaren kommerziellen Hypercube-Rechner der Firma Intel [Fox&&88] erfolgreich wieder aufgegriffen.

Seitdem wurden viele Hypercube-Rechner sowohl zu Forschungs- als auch zu kommerziellen Zwecken entwickelt und Probleme aus vielen, hauptsächlich wissenschaftlich-technischen Anwendungegebieten auf ihnen gelöst. Zu den Anwendungsgebieten gehören beispielsweise Biologie, Chemie, Physik, Geologie, Weltraumforschung, Astronomie, Astrophysik, Mathematik und Informatik [Angu&&90, Sh&Fi89].

Der Begriff „Hypercube" bezeichnet zwar nur eine spezielle Verbindungsstruktur, wird aber auch synonym für das gesamte, derartig verbundene Rechnersystem verwendet. In dieser Arbeit wird ein solches System zur deutlicheren Abgrenzung mit *Hypercube-Rechner* und die zugrundeliegende Verbindungsstruktur mit *Hypercube-Struktur* bezeichnet. Hypercube-Rechner werden aus 2^n, von 0 bis 2^n-1 numerierten, Prozessoren aufgebaut, die jeweils mit genau den n Prozessoren verbunden sind, deren Binärdarstellungen sich von der eigenen in genau einer Ziffer unterscheiden. Man nennt n die Dimension des Hypercube-Rechners

bzw. der Hypercube-Struktur. Man kann sich einen n-dimensionalen Hypercube-Rechner als einen n-dimensionalen Würfel denken, dessen Knoten je einen Prozessor und dessen Kanten Verbindungen zwischen ihnen repräsentieren. Eine dreidimensionale Hypercube-Struktur ist in Abbildung 2.8 dargestellt.
Hypercube-Rechner werden in der Regel als MIMD-Rechner [Flyn66] klassifiziert, die aus identischen Prozessoren mit ausschließlich lokalem Speicher gebildet werden und somit durch den Austausch von Nachrichten (*Message-Passing*) mit den direkt verbundenen Prozessoren kommunizieren müssen (*nearest neighbour communication*). Nachrichten an nicht benachbarte Prozessoren (*remote communication*) werden mittels mehrerer direkter Kommunikationsschritte durch einen einfachen, verklemmungsfreien Algorithmus [Kats88] ans Ziel geleitet (*Routing*). Offensichtlich beansprucht Kommunikation mit direkt verbundenen Prozessoren weniger Zeit als ein Routing. Daher werden Algorithmen mit hohem Anteil direkter Kommunikation am gesamten Kommunikationsaufwand einen höheren Speedup erwarten lassen als solche, deren entsprechender Anteil klein ist.
Die vorliegende Arbeit beschäftigt sich mit verschiedenen Aspekten zum Thema Hypercube-Rechner. Nach einem generellen Überblick über Architekturen von Parallelrechnern und deren Klassifikation wird der Begriff Hypercube-Struktur genauer definiert, verschiedene ihrer Eigenschaften diskutiert und mit anderen Verbindungsstrukturen verglichen. Anschließend werden die derzeit verfügbaren Hypercube-Rechner und deren Aufbau, Unterschiede und Gemeinsamkeiten aufgeführt. Abschließend folgt eine Untersuchung von Datentransferalgorithmen für Interprozessorkommunikation in Hypercube-Rechnern, die aufgrund ihrer Bedeutung für die Leistungsfähigkeit von Parallelrechnern den Schwerpunkt dieser Arbeit bildet.

2 Parallele Rechnerarchitekturen

Zur vollständigen Beschreibung eines parallelen Rechnersystems gehört sowohl die Betrachtung der verschiedenen Bauteile, aus denen der Rechner aufgebaut ist als auch der Konzepte, nach denen er arbeitet. Interessant sind z.B. Anzahl, Aufbau und Leistungsfähigkeit der verwendeten Prozessoren, über wieviel Speicher sie verfügen und wie schnell Ein- und Ausgabe von ihnen unterstützt wird. Wichtig ist ebenfalls die Art der Verbindungen (Bus oder Kanal), deren Struktur (Erweiterbarkeit, Partitionierbarkeit, Fehlertoleranz,...) und welche Synchronisationsmechanismen (SIMD, gemeinsamer Speicher, Message-Passing) benutzt werden [Alma85].

2.1 Die Klassifikation nach Flynn

Ein erster Ansatz zur Klassifikation der Vielzahl heute existierender Rechnersysteme ist die Einteilung nach den Möglichkeiten, ein- oder vielfache Instruktions- und Datenströme zu verarbeiten [Flyn66]. Demnach existieren vier Klassen von Rechnern:

SISD (single instruction stream, single data stream) Mit diesem Rechnertyp wird ein von Neumann-Rechner bezeichnet.

SIMD (single instruction stream, multiple data stream) Alle Rechner, die in der Lage sind, verschiedene Datenströme mit den gleichen Befehlen zu bearbeiten, fallen unter die SIMD-Kategorie. Die Synchronität der Ausführung der Instruktionen auf jedem Datenstrom und die zentrale Kontrolle aller ausführenden Einheiten sind Merkmale dieser Rechner. Illiac IV [Bouk&&72] ist ein Rechner dieses Typs.

MISD (multiple instruction stream, single data stream) Gleichzeitiges Ausführen verschiedener Instruktionen auf einem Datum wird als wenig sinnvoll erachtet [Dunc90]. Daher fällt kein Rechner unter diese Kategorie.

MIMD (multiple instruction stream, multiple data stream) Hier sind Multiprozessorsysteme gemeint, deren Einzelprozessoren autonom, mit möglicherweise verschiedenen Programmen, auf jeweils verschiedenen Daten arbeiten. Im Gegensatz zu SIMD-Rechnern geschieht dies asynchron, und Kontrollmechanismen für die Prozessoren sind dezentral implementiert. Jeder Prozessor kann daher als ein eigenständiger Rechner angesehen werden, so daß in diesem Zusammenhang auch der Begriff *Multicomputer* benutzt wird. Die meisten Hypercube-Rechner sind MIMD-Rechner, z.B. der iPSC/2 von Intel.

2.2 Parallelismus auf unteren Ebenen

Die Abgrenzung zwischen Parallelrechnern und seriellen Rechnern ist bei genauerer Untersuchung des Aufbaus heutiger Rechner nicht ganz einfach, denn sie stellen eine Vielzahl von Möglichkeiten zur Leistungssteigerung zur Verfügung, die zwar über das Konzept des seriellen von Neumann-Rechners hinausgehen, aber noch nicht die Kriterien für einen Parallelrechner erfüllen. Eine Auswahl besteht aus [Dunc90, Quin88]

- dem Befehlspipelining (*instruction pipelining*), die z.B. in Reduced Instruction Set Computers (RISC-Rechner) Anwendung findet. Hierbei wird die Ausführung aller Instruktionen in eine Folge von n elementaren Befehlen zerlegt, die nacheinander von verschiedenen hintereinandergeschalteten Funktionseinheiten des Prozessors (*Pipeline*) in jeweils gleicher Zeit t_0 ausgeführt werden.
  Nach der Ausführung des ersten Teilschritts eines Befehls in einer anfangs leeren Pipeline können der zweite Schritt dieses Befehls und der erste Schritt des nächsten Befehls parallel ausgeführt werden. Danach können Schritt eins des dritten, Schritt zwei des zweiten und Schritt drei des ersten Befehls parallel ausgeführt werden (Abb. 2.1). Bis zum n-ten Schritt (Startzeit, *startup-time*) liefert die Pipeline noch kein

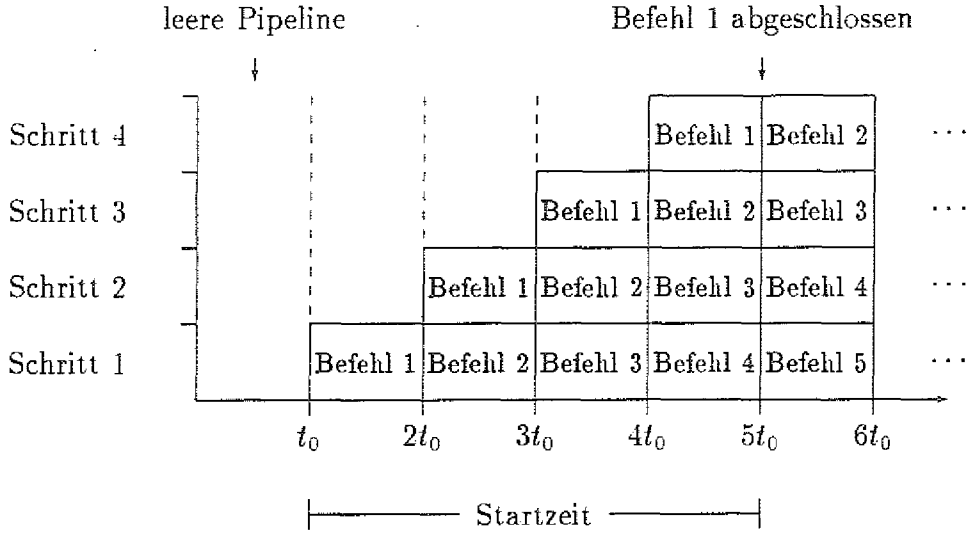


Abb. 2.1: Zeitdiagramm zum Befehlspipelining

Ergebnis, von da an jedoch wird mit jedem Schritt eine komplette Instruktion abgeschlossen.
Eine mögliche Zerlegung ($n = 4$) ist zum Beispiel die Aufteilung in

1. Instruktion laden,
2. Instruktion dekodieren,
3. Operand laden und
4. Instruktion ausführen.

- Verschiedene Prozessor-Funktionseinheiten (*multiple CPU functional units*) für arithmetische-, logische- und Vektoroperationen, die parallel arbeiten können und
- separate Ein- und Ausgabeprozessoren (*I/O-processors*), die für die Kommunikation von und zu externen Speichern zuständig sind, führen ebenfalls zu teilweise erheblichen Leistungssteigerungen des gesamten Rechners.

Systeme, die nur über derartige Möglichkeiten verfügen, werden nicht Parallelrechner genannt, denn ein Parallelrechner soll die Verarbeitung paralleler Programme durch die Bereitstellung mehrerer Prozessoren gestatten, die zusammen gleichzeitig an einer Problemlösung arbeiten [Dunc90].

2.3 Klassifikation paralleler Rechner

Zur Klassifikation von Parallelrechnern (Abb. 2.2) unterscheiden wir nun auf oberster Ebene in Ergänzung der Flynn'schen Klassifikation zwischen Synchronen-, MIMD- und MIMD-basierten Rechnern. Synchrone Systeme werden in Vektorrechner, SIMD-Rechner und Systolische Felder, MIMD-Rechner in geschaltete und Netzwerk-verbundene Systeme und MIMD-basierte Rechner in Datenflußrechner, Reduktionsarchitekturen, assoziative Prozessoren und Wavefront Array Rechner eingeteilt [Dunc90, Ho&Je88].

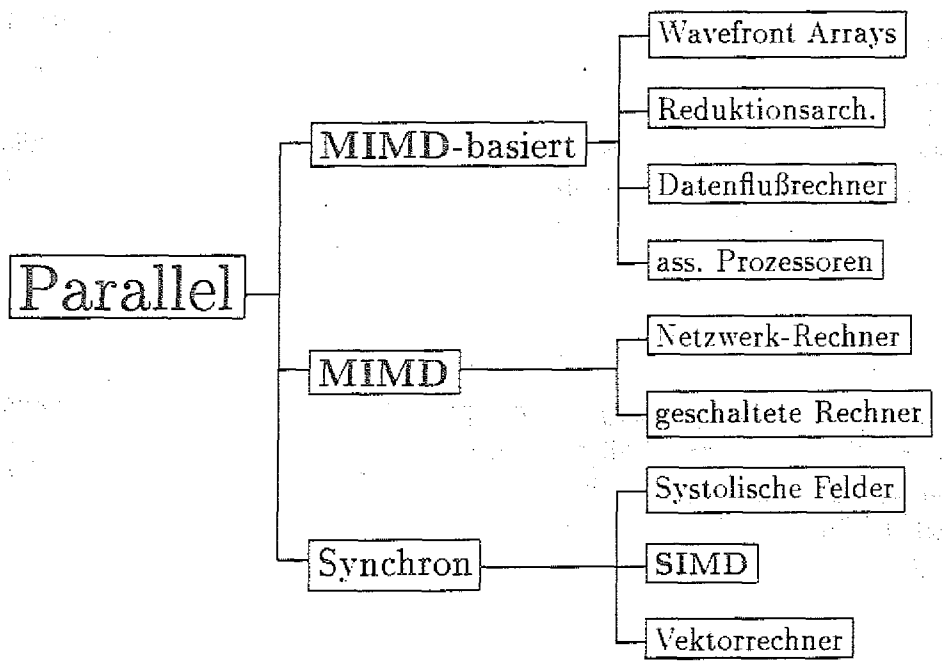


Abb. 2.2: Klassifikation paralleler Rechner

2.3.1 Synchrone Rechner

- Vektorrechner (*pipelined vectorcomputers*)
  - Register-Register Maschinen
  - Memory-Memory Maschinen

Vektorrechner verfügen über mehrere, parallel arbeitende Funktionseinheiten für Skalar- und Vektoroperationen, die wiederum in einer Pipeline hintereinandergeschaltet werden können. *Register-Register Maschinen* verfügen über spezielle, schnelle Vektorregister, die sowohl Operanden als auch Ergebnisse aufnehmen können (z.B. CRAY Y-MP). *Memory-Memory Maschinen* benutzen spezielle Puffer statt Vektorregister (z.B. CYBER 205).

- SIMD-Rechner
  - Prozessorfelder
  - Assoziative-Speicher-Prozessoren

  Prozessorfeldrechner sind die bekanntesten Beispiele für SIMD-Rechner. Hierbei sind die Prozessoren in einem Feld fester Dimension verbunden (z.B. Illiac IV, zweidimensionales Feld).
  Assoziative-Speicher-Prozessoren greifen auf gespeicherte Daten inhaltsabhängig zu. Alle Worte des Speichers werden dabei gleichzeitig, bitweise auf einen bestimmten Inhalt hin überprüft (z.B. Bell Laboratories Parallel Element Processing Ensemble). Einsatzmöglichkeiten dieser Rechner liegen im Datenbankbereich, z.B. für Suchzwecke.

- systolische Felder
  Systolische Felder (*systolic arrays*) sind sehr speziell auf ein Problem hin ausgelegte, regelmäßige Netzwerke mit hoher Parallelität, die Daten aus dem Speicher lesen, synchron mittels Pipelining durch das Netzwerk laufen lassen und die veränderten Daten wieder in den Speicher schreiben. Im allgemeinen kann ein systolisches Feld nur für die Lösung desjenigen Problems eingesetzt werden, für das es konstruiert wurde.

2.3.2 MIMD-Rechner

- geschaltete Rechner (*switched computers*)
  - gemeinsamer Speicher (*shared memory*) (Abb. 2.3)
  - verteilter Speicher (*distributed memory*) (Abb. 2.4)

  Als geschaltete Rechner werden Systeme mit einem zentralen, dynamischen Schaltnetz bezeichnet, an das eine Reihe von Prozessoren durch je eine Leitung angeschlossen sind. Verbindet das Schaltnetz die Prozessoren untereinander, so spricht man von einem System mit *verteiltem Speicher*, während die Verbindung von Prozessoren mit

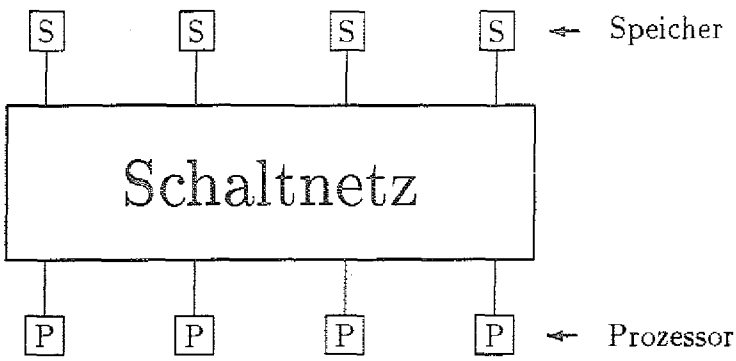


Abb. 2.3: Geschalteter Rechner mit gemeinsamem Speicher

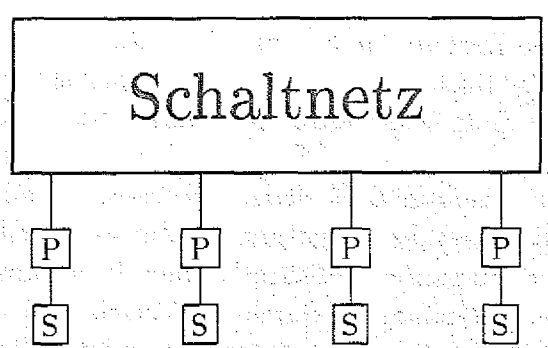


Abb. 2.4: Geschalteter Rechner mit verteiltem Speicher

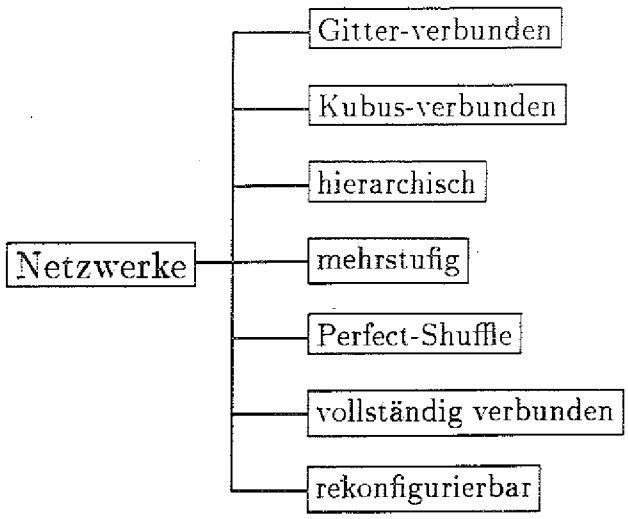


Abb. 2.5: Klassifikation von Netzwerken

einer oder mehreren Speichereinheiten zu einem System mit *gemeinsamem Speicher* führt. Letzteres hat zur Folge, daß die Kommunikation der Rechner über Variablen im gemeinsamen Speicher abgewickelt werden muß. Bei Systemen mit großer Anzahl von Prozessoren kann daraus ein Engpaß (Flaschenhals, *bottleneck*) resultieren, der die Leistungsfähigkeit des gesamten Rechners einschränkt.

- netzwerkverbundene Rechner (*networks*) (Abb. 2.5)
  Netzwerkverbundene Rechner sind Multiprozessorsysteme mit identischen Prozessoren, deren Verbindungen dezentral realisiert sind. Jeder Prozessor verfügt über einen lokalen Speicher, der von keinem anderen Prozessor angesprochen werden kann (verteiltes Speicherkonzept). Jeder Prozessor ist mit einigen anderen Prozessoren (*nearest-neighbours*) direkt verbunden. Zur eventuell erforderlichen Synchronisation und zum Datenaustausch während einer Programmausführung kommunizieren die Prozessoren untereinander durch das Verschicken von Nachrichten (*Message-Passing*). Kommunikation mit nicht direkt verbundenen Prozessoren wird durch mehrere Schritte über Prozessoren, die zwischen Sender und Empfänger liegen, abgewickelt. Das Finden eines Weges zwischen zwei Prozessoren nennt man *Routing*. Die Einfachheit des Routings ist ebenso wie die maximale und die durchschnittliche auftretende Weglänge, sowie die Anzahl alternativer Wege ein Bewertungskriterium für ein Netzwerk.

  Beispiele für Netzwerkstrukturen sind:

– Gitterstrukturen verschiedener Dimension mit und ohne äußere Verbindungen (*wrap-around*) (Abb. 2.6 und 2.7)

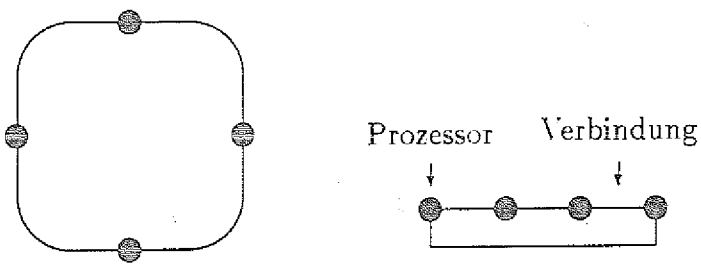


Abb. 2.6: Zwei Darstellungen einer eindimensionalen Gitterstruktur mit äußeren Verbindungen (Ring)

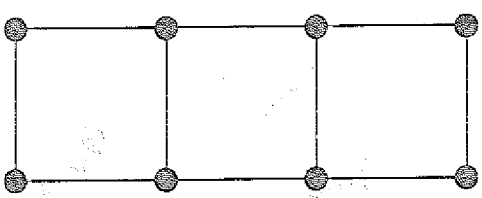


Abb. 2.7: Zweidimensionale Gitterstruktur ohne äußere Verbindungen

– Würfelnetzwerke und verwandte Strukturen
  (a) Hypercube-Strukturen (binary n-cube, boolean n-cube oder n-cube) (Abb. 2.8) [Heat85b]
  (b) Cube Connected Cycles (Abb. 2.9) [Pr&Vu81, Leng82]
  (c) Generalized Hypercubes (Abb. 2.10), base-b cubes [Bh&Ag84, La&Dh88]
  (d) Nearest Neighbour Mesh Hypercube (Abb. 2.11) [Bh&Ag84]
  (e) Banyan-Hypercubes [Yo&Na90]
  (f) Folded Hypercubes [La&El89]
  (g) Twisted Hypercubes [Efe&&88]
  (h) Bridged Hypercubes [El&La90]
– hierarchische Strukturen [Hayn&&82]

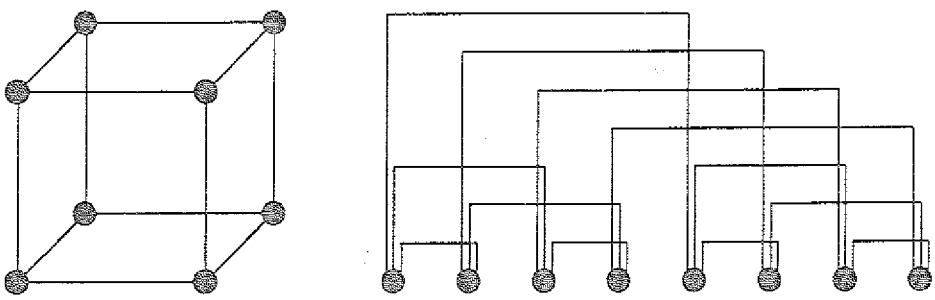


Abb. 2.8: Drei- und zweidimensionale Darstellung einer dreidimensionalen Hypercube-Struktur

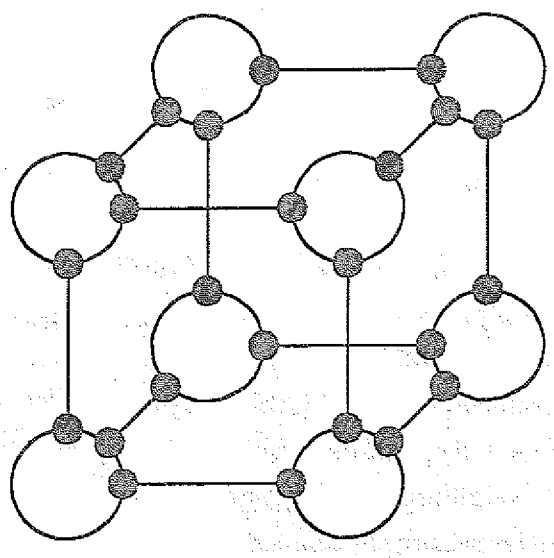


Abb. 2.9: Dreidimensionaler Cube Connected Cycle

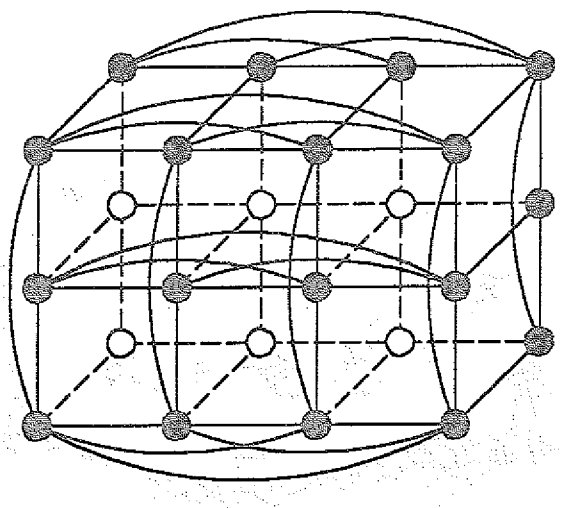


Abb. 2.10: Generalized Hypercube

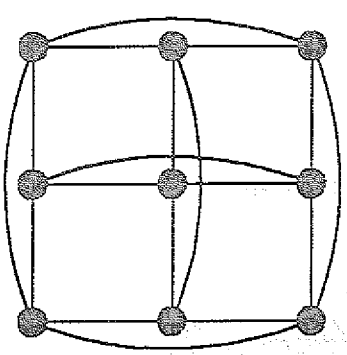


Abb. 2.11: Nearest Neighbour Mesh Hypercube

(a) Bäume
(b) Pyramiden [Quin88]
(c) Cluster
(d) Hypertrees (Abb. 2.12) [Go&Se81]

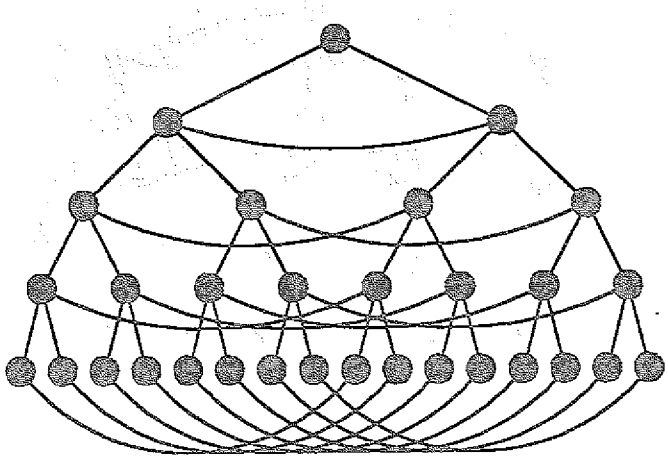


Abb. 2.12: **Hypertree**

- mehrstufige Netzwerke [Feng81]
  (a) Multistage Cube Networks [Sieg&&87a, Sieg&&87b]
    · mehrstufige Perfect-Shuffle
    · Omega [Lawr75]
    · baseline
    · delta
    · indirect binary n-cube [Peas77]
    · flip
    · SW-banyan
  (b) extra-stage cube networks
  (c) dynamic redundancy networks
- Perfect-Shuffle [Ston71]
- vollständige Verbindung
- rekofigurierbare Netzwerke

Ringe sind Beispiele für eine (eindimensionale) *Gitterstruktur* mit äußeren Verbindungen.

Ein *Hypercube-Rechner* der Dimension n besteht aus 2^n Prozessoren, die in einer n-dimensionalen Würfelstruktur mit je zwei Prozessoren pro Dimension verbunden sind. Damit ist jeder Prozessor mit n anderen Prozessoren verbunden. Eine Verallgemeinerung der Hypercube-Struktur ist der *Cube Connected Cycle* (CCC). In dieser Verbindungsstruktur ersetzt man jeden einzelnen Prozessor eines Hypercube-Rechners durch einen Ring mit n Prozessoren und verbindet nur je einen Prozessor eines Ringes mit einem Prozessor der n benachbarten Ringe. Die Anzahl von Verbindungen pro Prozessor wird damit für beliebige Dimension n auf eine Kubus- und zwei Ringverbindungen reduziert. Beispiele weiterer Verallgemeinerungen der Hypercube-Struktur sind der *Generalized Hypercube* (GHC) bzw. der base-b cube, der *Nearest Neighbour Mesh Hypercube* (NNMHC), der *Banyan-Hypercube*, der *Folded Hypercube*, der *Twisted Hypercube* sowie der *Bridged Hypercube*. Der GHC besteht in der i-ten Dimension, $0 \leq i \leq n-1$, aus m_i Prozessoren, die untereinander vollständig verbunden sind. Aus Gründen der Übersichtlichkeit wurden in Abbildung 2.10 nicht alle Verbindungen eingezeichnet. Für $m_i = 2, 0 \leq i \leq n-1$, erhält man eine Hypercube-Struktur. Ein NNMHC besteht aus m Prozessoren pro Dimension, die innerhalb einer Dimension mit ihren beiden Nachbarn auch über äußere Leitungen verbunden sind. Auch hier erhält man bei zwei Prozessoren pro Dimension ($m = 2$) eine Hypercube-Struktur. Banyan-Hypercubes sind eine Mischform aus Banyan-Netzen und der Hypercube-Struktur. Folded Hypercubes sowie Twisted Hypercubes sind leicht abgewandelte Hypercube-Strukturen mit verbesserten Eigenschaften bezüglich Routing und Durchmesser. In Bridged Hypercubes schließlich wird der Durchmesser des Netzwerks durch Einfügen einiger weniger Verbindungen in eine herkömmliche Hypercube-Struktur in etwa halbiert.

Pyramiden, *Bäume* und *Cluster* bezeichnet man als *hierarchische Strukturen*. Bäume können für viele verschiedene Algorithmen verwendet werden und sind sehr leicht dynamisch erweiterbar. Eine Verbindung dieser Eigenschaften mit denen der Hypercube-Struktur erreicht man durch *Hypernets* [Hw&Gh87] oder *Hypertrees*. Hypertrees sind wie binäre Bäume aufgebaut mit zusätzlichen Verbindungen zwischen einigen, jedoch nicht allen Knoten einer Ebene, deren Binärdarstellungen sich in genau einem Bit unterscheiden.

Mehrstufige Netzwerke können in verschiedenen Familien eingeteilt werden. Die oben aufgeführten Vertreter der *Multistage Cube Networks* sind topologisch äquivalent.
Extra Stage Cube Networks können als Multistage Cube Networks mit sehr einfacher Fehlertoleranz aufgefaßt werden, während *Dynamic Redundancy Networks* auch bei Ausfall eines beliebigen Prozessors einige Systemeigenschaften erhalten können.

Rekonfigurierbare Netzwerke schließlich können ihre Verbindungsstruktur dynamisch ändern und sind somit flexibel einsetzbar.

Ein Überblick über weitere Verbindungsstrukturen findet sich in [Feng81].

2.3.3 MIMD-basierte Rechner

- Als MIMD/SIMD-Rechner bezeichnet man MIMD-Rechner, die in verschiedene Bereiche aufgeteilt werden können, die jeweils wie ein SIMD-Rechner arbeiten [Sieg&&87a].
- *Wavefront Array Prozessoren* sind im Prinzip wie systolische Felder aufgebaut, jedoch arbeiten die Prozessoren asynchron und kommunizieren untereinander während einer Programmausführung.
- Reduktionsarchitekturen (*reduction architectures*, *demand-driven architectures*) führen eine Anweisung aus, falls deren Resultate für die Berechnung einer anderen, zur Ausführung anstehenden Anweisung benötigt werden (z.B. Newcastle Reduction Maschine).
- *Datenflußrechner* basieren auf der Idee, eine Instruktion genau dann auszuführen, wenn alle für die Ausführung benötigten Operanden zur Verfügung stehen.

2.4 Leistungsparameter und weitere Begriffe

Speedup: Unter dem *Speedup* S eines parallelen Algorithmus versteht man das Verhältnis der Ausführungszeit T_S des schnellsten sequentiellen Algorithmus auf einem Parallelrechner zur Ausführungszeit T_P des parallelen Algorithmus auf dem gleichen Parallelrechner:

$$S = \frac{T_S}{T_P}$$

Eine obere Schranke für den Speedup ist die Anzahl der verwendeten Prozessoren N.

Alternative Definitionen für S sind das Verhältnis der Ausführungszeiten des parallelen Algorithmus auf einem Prozessor und auf einem Parallelrechner oder das Verhältnis der Ausführungszeit des effizientesten, seriellen Algorithmus auf dem schnellsten seriellen Rechner zu der Zeit, die der parallele Algorithmus auf einem parallelen Rechner benötigt.
Die erste alternative Definition ist jedoch insofern irreführend, als daß sie nicht berücksichtigt, daß sequentielle Algorithmen meist einen wesentlich kleineren Verwaltungsaufwand enthalten als parallele Algorithmen. Daher wird der in diesem Fall resultierende Speedup stets größer ausfallen als in der ersten Definition.
Die zweite Definition setzt den Zugang zu den schnellsten seriellen Rechnern voraus und ist deshalb zumeist unbrauchbar [Quin88].

Effizienz: Die *Effizienz* ϵ eines parallelen Algorithmus ist das Verhältnis von Speedup S zur Anzahl N der verwendeten Prozessoren:

$$\epsilon = \frac{S}{N}$$

Es gilt $0 < \epsilon \leq 1$.

Kosten: *Kosten* werden definiert als das Produkt aus der Zeitkomplexität κ eines Algorithmus und der Anzahl N der verwendeten Prozessoren:

$$\text{Kosten} = \kappa \cdot N$$

κ wird meist in einer O-Notation [Knut75] in Abhängigkeit von der Problemgröße angegeben.

Overhead: Der Speedup läßt sich auch mittels des Begriffs *Overhead* σ durch

$$S \leq \frac{N}{1+\sigma}$$

abschätzen. Overhead bezeichnet den durch Parallelverarbeitung entstehenden zusätzlichen Aufwand. Darunter fällt z.B. die Kommunikation zwischen den Prozessoren und der Verwaltungsaufwand für die verschiedenen Prozesse.

Granularität: Unter Granularität (*grain size*) eines Problems versteht man die Größe parallel abarbeitbarer Prozesse, in die ein Problem aufgeteilt werden kann. Statt kleiner Granularität spricht man häufig auch von feiner- und statt großer auch von grober Granularität.
Der Begriff Granularität wird in der Literatur in vielfältiger Weise benutzt. Beispielsweise kann von der Granularität eines Parallelrechners oder eines Programms gesprochen werden, womit die Leistungsfähigkeit der verwendeten Prozessoren bzw. die Größe der Prozesse, aus denen das Programm besteht, gemeint ist [Dall88].

Amdahl's Gesetz: Als *Amdahl's Gesetz* wird die Tatsache bezeichnet, daß der maximal zu erzielende Speedup durch den Anteil k der sequentiell auszuführenden Befehle eines Algorithmus beschränkt ist [Amda67]:

$$S = \frac{T_S}{T_P} = \frac{1}{k + \frac{1-k}{N}} \leq \frac{1}{k}$$

k beschränkt damit die Anzahl der sinnvoll nutzbaren Prozessoren auf $1/k$, d.h. bei einem Anteil von $k = 0.1$ kann kein Speedup erreicht werden, der größer ist als $1/k = 10$.

Diese Begriffe stammen aus den Quellen [Ho&Je88, Seit85, Quin88, Fox&&88]. Dort finden sich noch weitere Begriffe, die in dieser Arbeit jedoch nicht verwendet werden.

3 Hypercube-Strukturen und deren Eigenschaften

Hypercube-Rechner fallen in der Regel in die Klasse der netzwerkverbundenen MIMD-Rechner (Kap. 2.3.2). Vorteile dieser Systeme gegenüber der anderen Klasse von MIMD-Rechnern, den geschalteten Rechnern, sind die Art der Kommunikation (Message-Passing), die Einfachheit des Aufbaus, die leichte und im Prinzip unbegrenzte Erweiterbarkeit und die Möglichkeit der Nutzung von Lokalität bei der Kommunikation (nearest-neighbour communication).
Durch Message-Passing wird der Aufbau eines großen, zentralen Schaltnetzes und die bei Systemen mit großer Prozessorzahl auftretende Konkurrenz beim Zugriff auf einen gemeinsamen Speicher vermieden. Bei einer Erweiterung des Rechners müssen die neuen Prozessoren nur mit dem System verbunden werden. Die bestehende Verbindungsstruktur kann — logisch gesehen — vollständig übernommen werden. Die in Kapitel 2.3.2 vorgestellten Netzwerke lassen eine einfache Abbildung eines Problems oder Algorithmus auf den gesamten Rechner zu. Für Matrixoperationen beispielsweise sind zweidimensionale Gitterstrukturen sehr gut geeignet [Bern89]. Viele Netzwerke lassen weiterhin die Abbildung einer Vielzahl anderer Netzwerke etwa gleicher Knotenzahl in die gegebene Struktur zu, wobei alle Nachbarschaftsbeziehungen erhalten werden können. Im Fall der Hypercube-Struktur sind dies z.B. Ringe gerader Knotenzahl, Gitter verschiedener Dimensionen mit und ohne äußere Verbindungen und Bäume.
Rechner mit zentralem Schaltnetz und gemeinsamem Speicher hingegen erlauben für den Benutzer einfacheren Datenzugriff wegen des linearen Speicherbereichs. Damit kann diese Art von Rechner auf ähnliche Weise wie ein sequentieller Rechner programmiert werden. Weiterhin erlaubt das dynamische Schaltnetz, bei möglicherweise größerem Zeitaufwand, eine Simulation aller Netzwerkstrukturen. Da jedoch das Schaltnetz mit zunehmender Prozessorzahl komplexer wird und Speicherkonflikte häufiger auftreten, ist diese Art von Rechner für Systeme mit sehr vielen Prozessoren nicht geeignet [Sa&Sc88].
Im folgenden werden einige Eigenschaften der Hypercube-Struktur und die Einbettung verschiedener anderer Strukturen in diese Struktur behandelt. Abschließend folgt ein tabellarischer Vergleich der aufgeführten Eigenschaften für die Verbindungsstrukturen Ring, zweidimensionales Gitter, binärer Baum, Hypercube, Cube Connected Cycle und vollständige Verbindung.

3.1 Definition und Eigenschaften der Hypercube-Struktur

Hypercube-Strukturen finden schon seit längerer Zeit Anwendung bei der Lösung von Schaltungsproblemen und in der Kodierungstheorie [Gilb58] und werden meist in einer graphentheoretischen Notation angegeben. Eine Hypercube-Struktur (*n-cube*, binary n-cube, boolean n-cube) besteht aus 2^n, von 0 bis $2^n - 1$ numerierten Knoten (*nodes*). Zwischen zwei Knoten existiert genau dann eine Verbindung (Kante, *edge*), wenn sich die Adressen (*labels*) der Knoten in den durch Voranstellung von Nullen auf n Ziffern erweiterten Binärdarstellungen in genau einem Bit unterscheiden. Daraus folgt, daß jeder Knoten mit genau n anderen Knoten verbunden ist. Diese Struktur läßt sich als n-dimensionale Würfelstruktur darstellen. Daher heißt n die *Dimension* der Hypercube-Struktur.

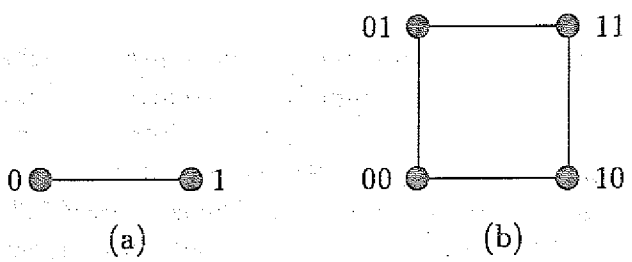


Abb. 3.1: (a) eindimensionale Hypercube-Struktur, (b) zweidimensionale Hypercube-Struktur

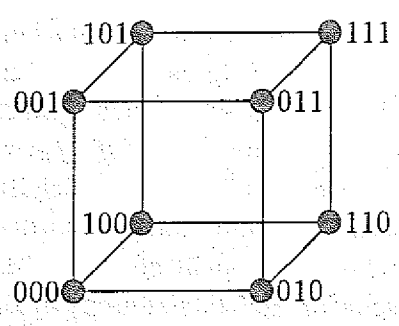


Abb. 3.2: Dreidimensionale Hypercube-Struktur

Hypercube-Strukturen der Dimensionen eins bis vier sind in den Abbildungen 3.1, 3.2 und 3.3 dargestellt. Die dort gewählten Bezeichnungen der Knoten sind nicht eindeutig. In

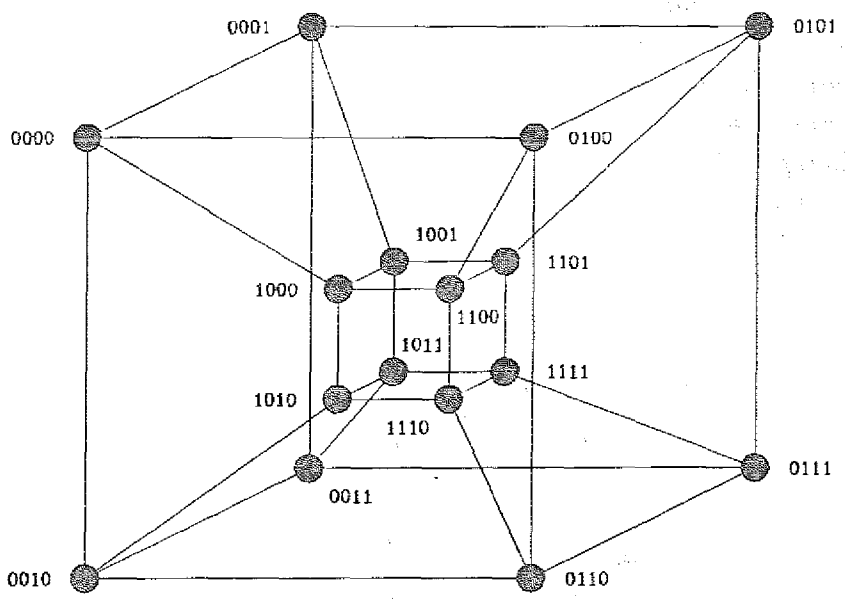


Abb. 3.3: Vierdimensionale Hypercube-Struktur

Abbildung 3.1 (b) beispielsweise hat man anfangs $4 = 2^n$ Möglichkeiten, eine beliebige Knotennummer (Knotenadresse) vorzugeben. Die beiden Nachbarknoten können dann auf zwei Arten numeriert werden, während die Nummer des nicht benachbarten Knotens bereits mit der Wahl der ersten Nummer festliegt. Damit ergeben sich insgesamt $8 = 4 \cdot 2$ verschiedene Numerierungen. Für beliebiges n existieren $2^n \cdot n!$ gültige Numerierungen [Sa&Sc88].
Man erkennt bei Hypercube-Strukturen bis zu einer Dimension von drei, daß jeder Knoten über alle Dimensionen mit genau einem weiteren Knoten verbunden ist. Meist numeriert man die Dimensionen von 0 bis $n-1$ derart, daß man eine Verbindung zwischen zwei Knoten über die i-te Dimension an den verschiedenen Einträgen in den i-ten Stellen der jeweiligen Binärdarstellungen erkennt.
Eine ausführliche Diskussion verschiedener Eigenschaften der Hypercube-Struktur findet sich in [Sa&Sc85a, Sa&Sc88].

3.1.1 Rekursiver Aufbau

Eine Eigenschaft der Hypercube-Struktur ist der *rekursive Aufbau*, d.h. man kann eine Hypercube-Struktur der Dimension $n+1$, $n > 0$ aus Hypercube-Strukturen der Dimension n konstruieren. Da eine $(n+1)$-dimensionale Hypercube-Struktur aus 2^{n+1} Knoten besteht, benötigt man für ihre Konstruktion genau zwei n-dimensionale Hypercube-Strukturen.

Die Konstruktion ist sehr einfach, denn es müssen lediglich 2^n neue Verbindungen — von jedem Knoten der einen Hypercube-Struktur genau eine Verbindung zu dem Knoten gleicher Nummer der anderen Hypercube-Struktur — gezogen werden. Danach werden die Knoten neu numeriert. Eine einfache Umnumerierung besteht in der Einfügung einer null an einer beliebigen, festen Stelle i, $0 \leq i \leq n-1$, der alten Knotenadressen der ersten Hypercube-Struktur und einer eins an der gleichen Stelle i aller Knotenadressen der zweiten Hypercube-Struktur.

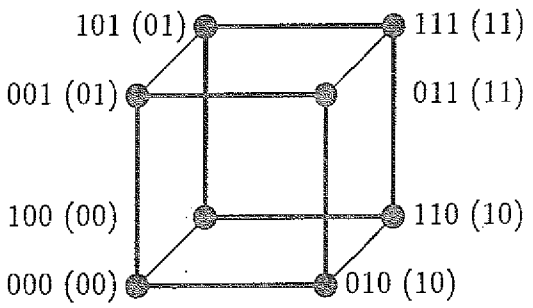


Abb. 3.4: **Aufbau einer dreidimensionalen Hypercube-Struktur**

Abbildung 3.4 zeigt die Zusammensetzung einer dreidimensionalen Hypercube-Struktur aus zwei zweidimensionalen Hypercube-Strukturen (Abb. 3.1 (b)). Die dünneren Linien stehen für neu zu errichtende Verbindungen. Die Adressen in Klammern sind die alten Knotenadressen. Die neuen Adressen entstehen in diesem Beispiel aus den alten durch Voranstellen einer null bzw. eins.
Die Eigenschaft des rekursiven Aufbaus beinhaltet auch das Aufspalten (*tearing*) einer Hypercube-Struktur in Unterwürfel (*subcubes*), d.h. in Hypercube-Strukturen kleinerer Dimension [Sa&Sc88]. Aus obiger Beschreibung folgt, daß eine n-dimensionale Hypercube-Struktur in 2^{n-j} j-dimensionale Hypercube-Strukturen aufgespalten werden kann, falls $0 \leq j < n$. Für $j = n-1$ heißt dies, daß man zwei Unterwürfel der Dimension $n-1$ findet. Für diese spezielle Aufspaltung existieren n Möglichkeiten, denn eine Aufspaltung kann mittels Weglassen einer der n binären Ziffer in den Knotenadressen durchgeführt werden. Aufspaltungen von Hypercube-Rechnern können z.B. für Mehrbenutzerbetrieb, Debugging auf kleineren Konfigurationen und feinere Problemaufspaltungen genutzt werden.

3.1.2 Routing

Mit *Routing* bezeichnet man das Finden eines Weges zwischen zwei Knoten s und t eines Netzwerks. Der Weg wird meist als eine Folge von zu durchlaufenden Knoten angegeben, die

mit s beginnt und mit t endet. Das Prinzip eines Routing-Algorithmus für eine Hypercube-Struktur ist sowohl einfach zu verstehen als auch einfach zu programmieren und kann von jedem Knoten selbständig durchgeführt werden. D.h. jeder Knoten, der eine Nachricht erhält, kann nur mit Hilfe der Knotenadresse des Empfängers und seiner eigenen Adresse entscheiden, wohin die Nachricht weitergeleitet werden muß.
Dem Algorithmus liegt der in der Definition der Hypercube-Struktur festgelegte Sachverhalt zugrunde, daß sich die Adressen benachbarter Knoten an genau einer Stelle unterscheiden. Vergleicht man nun seine eigene Adresse mit der des Empfängers bitweise in beliebiger Reihenfolge, und unterscheiden sie sich an der Stelle i, so leitet man die Nachricht über die Verbindung der i-ten Dimension weiter [Raab&&88]. Die Adresse des empfangenden Prozessors unterscheidet sich an der Stelle i nun nicht mehr von der des Empfängers der Nachricht. Sind beide Adressen in allen Stellen gleich, so hat die Nachricht ihr Ziel erreicht. Die Anzahl ungleicher Bits (*Hamming-distance*, Hamming-Abstand) im bitweisen Vergleich beider Knotenadressen ergibt demnach die Anzahl der Schritte im Routing-Algorithmus. Maximal müssen n Schritte ausgeführt werden.

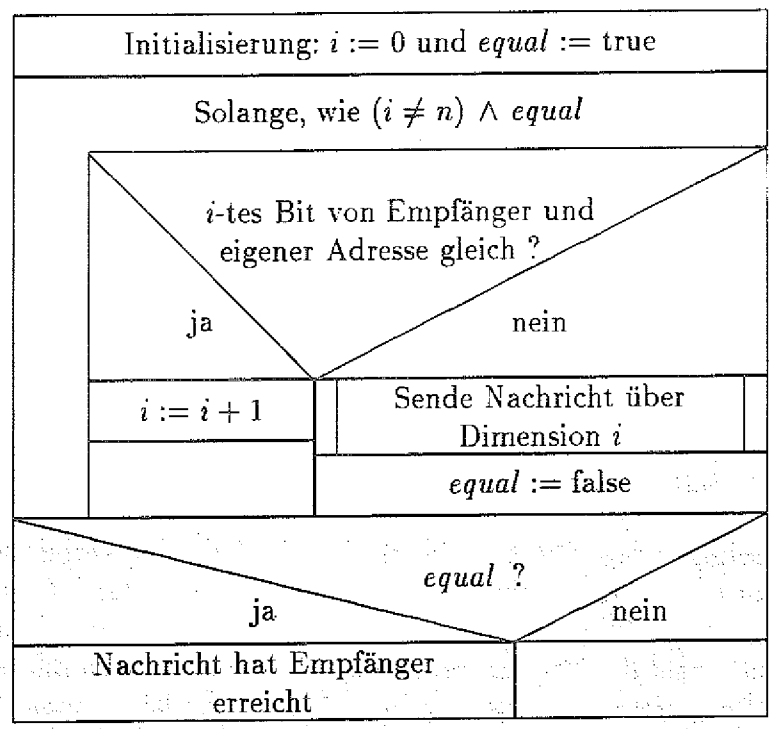


Abb. 3.5: Routing-Algorithmus für einen Hypercube-Rechner

Eine genauere Formulierung des Algorithmus, der in jedem Knoten nach Empfang einer

Nachricht abläuft, zeigt das Struktogramm in Abbildung 3.5. Der Vergleich der Bits beginnt mit der Stelle null und endet mit der Stelle $n-1$. Nach Abschluß des Algorithmus gibt der Wert der boolschen Variablen *equal* an, ob der Knoten Empfänger der Nachricht ist oder nicht.

3.1.3 Abstände

Als Abstand (*distance*) zwischen zwei verschiedenen Knoten s und t bezeichnet man die kleinste Anzahl von Zwischenknoten einschließlich s, die eine Nachricht von s nach t passieren muß. Dies entspricht der Anzahl der dabei benutzten Verbindungen.
Wie aus dem in Kapitel 3.1.2 vorgestelltem Routing-Algorithmus ersichtlich ist, ergibt die Länge n der Binärdarstellung der Knotenadressen den maximalen Abstand zweier Knoten, den Durchmesser (*diameter*) eines Hypercube-Rechners [Raab&&88].
Der durchschnittliche Abstand DA (*durchschnittliche Weglänge*) [Raab&&88] zwischen zwei Knoten berechnet sich aus dem Quotienten der Summe aller Weglängen[1] SW mit

$$SW \overset{\text{Hypercube-Struktur}}{=} \sum_{i=1}^{n} \binom{n}{i} \cdot i$$

und der Anzahl der erreichten Knoten, die der um eins verringerten Gesamtknotenzahl N entspricht. Damit ist

$$\begin{aligned} DA &= \frac{SW}{N-1} \\ &\overset{\text{Hypercube-Struktur}}{=} \frac{1}{2} \cdot \frac{n \cdot 2^n}{2^n - 1} \\ &\approx \frac{n}{2}. \end{aligned}$$

3.1.4 Anschlüsse pro Knoten

Die Zahl der Anschlüsse pro Knoten (*Ports*), d.h. die Zahl der Verbindungsleitungen, die an jeden Prozessor angeschlossen sind, spielt für die technische Realisierung des Prozessors eine wichtige Rolle und sollte möglichst klein sein. Ebenfalls von Vorteil ist die Unabhängigkeit von der Gesamtzahl der Prozessoren des Netzwerks. Dadurch erfordert nicht nur die Herstellung von Prozessoren für Systeme beliebiger Größe konstanten Aufwand, sondern man kann im Prinzip auch ein bestehendes Netzwerk unter Weiterverwendung der darin benutzten Prozessoren durch einfaches Verbinden mit weiteren Prozessoren erweitern.

[1] Die Anzahl der Knoten einer Hypercube-Struktur im Abstand i zu einem beliebigen Knoten ist gleich der Anzahl der Möglichkeiten, i Bits aus der n Ziffern langen Binärdarstellung der Adressen zu wählen.

Für n-dimensionale Hypercube-Strukturen gilt, daß jeder Knoten mit n Knoten verbunden ist, also über n Verbindungen verfügt. Dadurch ist eine logarithmische Abhängigkeit der Verbindungen pro Knoten von der Gesamtzahl der Knoten gegeben. Dieser Nachteil bedingt, daß Prozessoren aus Hypercube-Rechnern niedriger Dimension in Hypercube-Rechnern höherer Dimension im allgemeinen nicht verwendet werden können, obwohl die Hypercube-Struktur rekursiv aufgebaut ist [Raab&&88].

3.1.5 Parallele Wege

Parallele Wege (*node-disjoint paths, parallel paths*) sind knotendisjunkte Wege zwischen zwei Knoten. Solche Wege können zur schnelleren, parallelen Datenübertragung oder zur Umgehung eines fehlerhaften Knotens genutzt werden. Eine hohe Anzahl paralleler Wege kann als ein entscheidender Vorteil eines Netzwerks angesehen werden.
Eine obere Grenze für die Zahl paralleler Wege ist die Zahl der Anschlüsse pro Knoten. Bei Hypercube-Rechnern ist dies die Dimension n. Angenommen, die Adresse des Senders unterscheidet sich von der des Empfängers in j Stellen, so kann der Sender zwischen j Möglichkeiten für den ersten Kommunikationsschritt wählen. Die dadurch entstehenden j Wege gleicher Länge sind, bis auf Sender und Empfänger, knotendisjunkt, falls alle weiteren Knoten den bekannten Algorithmus aus Abbildung 3.5 mit einigen Modifikationen durchführen (Abb. 3.6) [Sa&Sc88].
Als Beispiel seien die so entstehenden, parallelen Wege zwischen zwei Knoten eines vierdimensionalen Hypercube-Rechners aufgeführt:

Beispiel 3.1.1 : Wege zwischen den Knoten 0001 und 1111 (Hamming-Abstand drei)

1. $0001 \Rightarrow 1001 \Rightarrow 1101 \Rightarrow 1111$
2. $0001 \Rightarrow 0101 \Rightarrow 0111 \Rightarrow 1111$
3. $0001 \Rightarrow 0011 \Rightarrow 1011 \Rightarrow 1111$

Der erste Schritt für jeden der drei Wege ist durch den Sender 0001 frei wählbar. Die restlichen beiden Schritte sind durch den modifizierten Routing-Algorithmus vorgegeben.

Man erkennt, daß die weiteren Bits nach der Wahl des ersten Bit in allen Knoten nach einem bestimmten Muster abgearbeitet werden müssen. Dieses Muster kann jedoch frei gewählt werden. So führt auch die Bearbeitung in umgekehrter Reihenfolge in allen Prozessoren zu parallelen Wegen:

Beispiel 3.1.2 :

1. $0001 \Rightarrow 1001 \Rightarrow 1011 \Rightarrow 1111$

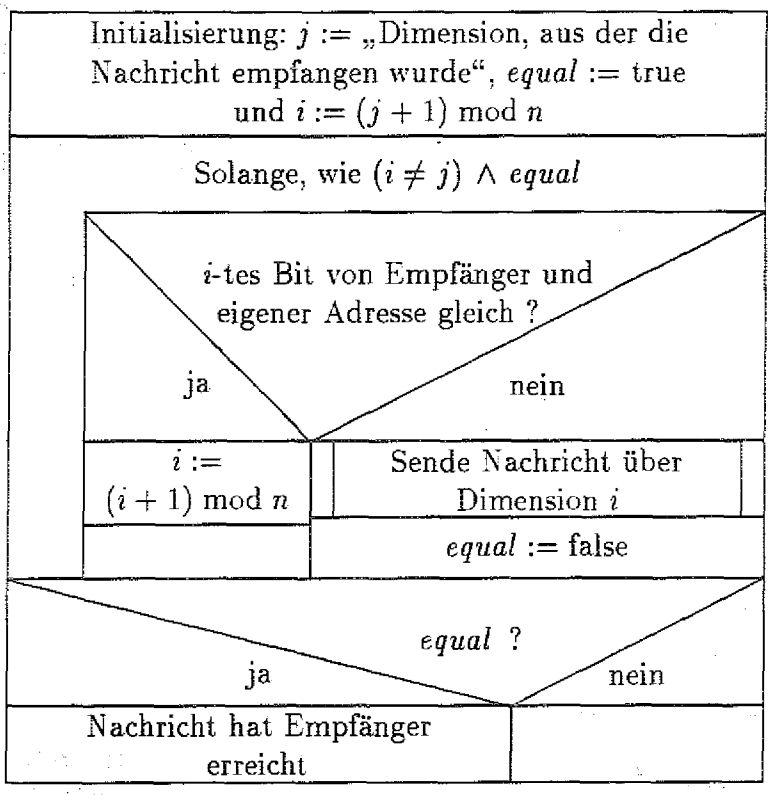


Abb. 3.6: **Routing-Algorithmus für einen Hypercube-Rechner**

2. $0001 \Rightarrow 0101 \Rightarrow 1101 \Rightarrow 1111$
3. $0001 \Rightarrow 0011 \Rightarrow 0111 \Rightarrow 1111$

Auch hier wurde der jeweils erste Schritt frei gewählt. Die restlichen Bits wurden nun „von rechts nach links" bearbeitet.

Werden aber verschiedene Reihenfolgen benutzt, so erhält man i.a. keine parallelen Wege:

Beispiel 3.1.3 :

1. $0001 \Rightarrow 1001 \Rightarrow 1101 \Rightarrow 1111$
2. $0001 \Rightarrow 0101 \Rightarrow 1101 \Rightarrow 1111$
3. $0001 \Rightarrow 0011 \Rightarrow 1011 \Rightarrow 1111$

Hier sind die Wege eins und zwei nicht knotendisjunkt, denn Knoten 1001 hat die Bits von „links nach rechts" und Knoten 0101 von „rechts nach links" verglichen.

Neben den j parallelen Wegen der Länge j können noch $n-j$ Wege der Länge $j+2$ gefunden werden, so daß alle n Wege knotendisjunkt sind [Sa&Sc88]. Somit können alle Verbindungen eines Prozessors gleichzeitig zur Kommunikation mit einem anderen Prozessor genutzt werden. Man erhält einen der $n-j$ neuen Wege durch die Übermittlung der Nachricht über eine der $n-j$ Dimensionen, deren Bit in Sender und Empfänger gleich ist. Danach verfährt man wie in obigem Algorithmus und sendet zum Schluß die Nachricht wieder über die anfangs gewählte Dimension. Somit benötigt man zusätzlich zu der Anzahl der Schritte des Routing-Algorithmus zwei weitere Schritte.
Man erhält damit für Beispiel 3.1.1 den zusätzlichen Weg

$$0001 \Rightarrow \underbrace{0000 \Rightarrow 1000 \Rightarrow 1100 \Rightarrow 1110}_{\text{Weg nach erstem Algorithmus}} \Rightarrow 1111.$$

Da nach dem ersten Algorithmus verschiedene Wege existieren, können auch andere „Mittelstücke" für den obigen Weg gewählt werden, z.B.

$$0000 \Rightarrow 0010 \Rightarrow 0110 \Rightarrow 1110$$

3.1.6 Fehlertoleranz

Unter Fehlertoleranz (*fault tolerance*) versteht man die maximal mögliche Anzahl beliebiger Knoten, die ausfallen können, ohne den Zusammenhang der intakten Knoten zu zerstören [Raab&&88].

Da jeder Knoten einer Hypercube-Struktur über genau n Verbindungen verfügt, ist n eine obere Schranke für dessen Fehlertoleranz, denn der Ausfall aller n Nachbarknoten isoliert einen Knoten vom restlichen System. Andererseits existieren n knotendisjunkte Pfade zwischen zwei beliebigen Knoten. Damit existiert selbst bei Ausfall von $n-1$ Knoten noch mindestens ein Weg von einem beliebigen Knoten zu allen anderen Knoten. Die Fehlertoleranz einer n-dimensionalen Hypercube-Struktur beträgt damit $n-1$.

3.1.7 Durchschnittliche Verkehrsdichte

Die *durchschnittliche Verkehrsdichte* DVD ist ein Maß für die auftretende Belastung der Verbindungsleitungen eines Netzwerks. Sie ist definiert durch das Produkt von durchschnittlichem Abstand DA (Seite 26) und Knotenzahl N, dividiert durch die Anzahl aller Verbindungen VL [Raab&&88]. Mit $VL = \frac{1}{2} \cdot n \cdot N$ für die Hypercube-Struktur ergibt sich

$$\begin{aligned} DVD &= \frac{DA \cdot N}{VL} \\ &\overset{\text{Hypercube-Struktur}}{=} \frac{\frac{1}{2} \cdot \frac{n \cdot N}{N-1} \cdot N}{\frac{1}{2} \cdot n \cdot N} \\ &= \frac{N}{N-1}. \end{aligned}$$

Je größer der Wert der DVD ist, desto größer wird die Wahrscheinlichkeit für die Verzögerung einer Nachricht aufgrund der Blockierung einer Verbindung durch eine andere Nachricht. Da

$$\lim_{N \to \infty} \left(\frac{N}{N-1} \right) = 1,$$

ist die durchschnittliche Verkehrsdichte für Hypercube-Rechner, unabhängig von deren Dimension, sehr gering und sogar nahezu konstant.

3.1.8 Gesamtbandbreite

Die maximale Anzahl der zu einem Zeitpunkt gleichzeitig versendbaren Pakete wird mit Gesamtbandbreite GB eines Netzwerks (*bandwidth*) [Kris89] bezeichnet. Eine große Gesamtbandbreite kann für einen schnellen Datentransfer zwischen den Knoten des Netzwerks benutzt werden.
Die Gesamtbandbreite der Hypercube-Struktur berechnet sich unter der Vorraussetzung von bidirektionalen Verbindungsleitungen durch das Produkt aus Knotenzahl N und der Anschlüsse pro Knoten n:

$$GB = n \cdot N$$

3.2 Abbildung von Netzwerken in die Hypercube-Struktur

Die Abbildung von Netzwerken in eine gegebene Verbindungsstruktur unter Erhalt aller Nachbarschaftsbeziehungen zwischen den Knoten kann sich bei der Entwicklung von Algorithmen als äußerst nützlich erweisen. Läßt sich die Struktur eines Problems leicht auf die Struktur eines Netzwerks abbilden, so läßt sich ebenfalls leicht ein Algorithmus zur Lösung des Problems auf diesem Netzwerk angeben, welcher bezüglich Kommunikation effizient ist. Ist man nun in der Lage, eine Abbildung dieses Netzwerks in ein zweites Netzwerk zu definieren, so hat dies den Vorteil, daß man den ursprünglich für das erste Netzwerk entwickelten Algorithmus nur durch Zuhilfenahme der Abbildung für das zweite Netzwerk übernehmen kann. Als Beispiel hierfür können Matrixalgorithmen angegeben werden, die häufig für zweidimensionale Gitterstrukturen definiert sind [Bern89, Fox&&87]. Diese können ohne Schwierigkeiten in eine Hypercube-Struktur abgebildet werden [Sa&Sc88].
Im folgenden werden die Abbildungen verschiedener, spezieller Netzwerke in die Hypercube-Struktur betrachtet. Für die Verbindungsstrukturen Ring, Gitter und Baum kann dies mit Erhalt der Nachbarschaftsbeziehungen geschehen. Bei Perfect-Shuffle- und Shuffle-Exchange-Netzwerken ist dies nicht möglich. Für jeden dieser speziellen Fälle allerdings kann die Abbildung auf einfache Weise definiert werden. Hingegen wird für das allgemeinere Problem der Feststellung, ob ein beliebiger Graph ein Teilgraph einer Hypercube-Struktur ist, in [Krum&&85] nachgewiesen, daß es NP-vollständig ist.
Statt Abbildung eines Netzwerks in ein anderes spricht man auch von Einbettung (*embedding*) oder Simulation eines Netzwerks.

3.2.1 Ringe

Es soll hier das Problem der Abbildung von Ringen (Abb. 2.6, Seite 13) der Knotenzahl l in n-dimensionale Hypercube-Strukturen betrachtet werden, so daß jedem Knoten der Hypercube-Struktur höchstens ein Knoten des Ringes zugeordnet wird.
Zunächst läßt sich feststellen, daß l gerade sein muß, denn es existieren keine Zykel, d.h. in sich abgeschlossene Folgen von Knoten, ungerader Länge in einer Hypercube-Struktur. Dies folgt aus der Tatsache, daß sich die Adressen benachbarter Knoten in genau einem Bit unterscheiden. Die *Paritäten*[2] benachbarter Knoten sind also unterschiedlich. Angenommen, es existiert ein Zykel ungerader Länge mit den Knoten $s_1, \ldots, s_{l+1} = s_1$. Mit jedem Wechsel von s_i zu s_{i+1} ändert sich die Parität des gerade besuchten Knotens. Wäre $l+1$ nun ungerade, so würde dies bedeuten, daß s_{l+1} eine andere Parität besäße als s_1. Da

[2]Die Parität eines Knotens ist als positiv definiert, falls die Anzahl gesetzter Bits in dessen Adresse gerade ist, und negativ sonst.

$s_1 = s_{l+1}$, ist dies ein Widerspruch.
Ringe gerader Länge lassen sich mit Hilfe binärer Gray Codes (*binary reflected Gray codes*) auf Hypercube-Strukturen abbilden. Binäre Gray Codes [Sa&Sc88] sind Folgen aller binärer Zahlen einer konstanten Länge und können wie folgt rekursiv definiert werden:

$$\begin{array}{lcll} G_1 & = & (\ 0, & 1\ ) & \text{einstelliger Gray Code} \\ G_{n+1} & = & (\ 0G_n, & 1G_n^R\ ) & (n+1)\text{-stelliger Gray Code} \end{array}$$

G_n^R steht für den n-stelligen Gray Code in umgekehrter (reverse) Reihenfolge. Der $(n+1)$-stellige Gray Code entsteht also aus der Aneinanderreihung der Elemente des n-stelligen Gray Codes in ursprünglicher mit den gleichen Elementen in umgekehrter Reihenfolge. Den Elementen der ersten Folge wird dabei eine null und denen der zweiten Folge eine eins vorangestellt. Der Gray Code der Stellenzahl n wird auch n-bit Gray Code genannt.
Beispielsweise lauten die Gray Codes der Stellenzahl zwei und drei:

$$\begin{array}{lcl} G_2 & = & (\ 00,\ 01,\ 11,\ 10\ ) \\ G_3 & = & (\ 000,\ 001,\ 011,\ 010,\ 110,\ 111,\ 101,\ 100\ ) \end{array}$$

Mit einem n-bit Gray Code ist die Abbildung eines Ringes der Länge 2^n in eine Hypercube-Struktur gleicher oder größerer Knotenzahl direkt gegeben, denn aufeinanderfolgende Elemente eines binären Gray Codes unterscheiden sich — wie die Adressen benachbarter Knoten einer Hypercube-Struktur — an genau einer Stelle.

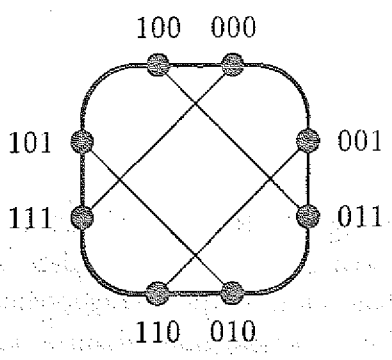


Abb. 3.7: Abbildung eines Ringes der Länge acht in eine dreidimensionale Hypercube-Struktur

Beispiel 3.2.1 : Abbildung 3.7

Die dick gezeichneten Linien stellen Hypercube-Verbindungen für die Simulation des Ringes dar. Die dünnen Linien sind die übrigen Verbindungen der Hypercube-Struktur.

Die Abbildung in eine m-dimensionale Hypercube-Struktur mit $m > n$ geschieht auf die gleiche Weise. In diesem Fall kann man die verbleibenden $m-n$ Stellen der Knotenadressen der Hypercube-Struktur beliebig, aber fest wählen, d.h. es wird nur ein n-dimensionaler Unterwürfel der Hypercube-Struktur genutzt. Die restlichen $2^{m-n} - 1$ n-dimensionalen Unterwürfel können für andere Zwecke eingesetzt werden.
Bis jetzt wurde nur beschrieben, wie solche Ringe in Hypercube-Strukturen abgebildet werden, deren Länge l sich als eine Zweierpotenz mit positivem, ganzzahligem Exponenten darstellen läßt. Ringe mit beliebiger, gerader Länge lassen sich jedoch auch, unter Verwendung von *partiellen Gray Codes*, in Hypercube-Strukturen abbilden.
Ein partieller n-bit Gray Code $G_n(m)$ mit $m < 2^n$ besteht aus den ersten m Folgengliedern des n-bit Gray Codes, z.B. $G_2(3) = (00, 01, 11)$.
Sei l die beliebige, aber gerade Länge eines Ringes und n so, daß $l \leq 2^n$. Eine Folge von Knotenadressen der Hypercube-Struktur, die einen Ring bilden, ist durch

$$(0G_{n-1}(m),\ 1G_{n-1}(m)^R),\quad m = \frac{l}{2}$$

gegeben.

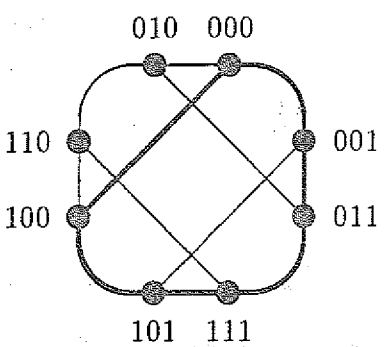


Abb. 3.8: **Abbildung eines Ringes der Länge sechs in eine dreidimensionale Hypercube-Struktur**

Beispiel 3.2.2 : $l = 6$, d.h. $m = 3, n \geq 3$, Abbildung 3.8 mit $n = 3$

$$\underbrace{000 \Rightarrow 001 \Rightarrow 011}_{0G_2(3)} \Rightarrow \underbrace{111 \Rightarrow 101 \Rightarrow 100}_{1G_2(3)^R}\ (\Rightarrow 000)$$

n, die Dimension der Hypercube-Struktur, wurde hier minimal gewählt. Die Verwendung einer höherdimensionalen Struktur bereitet auch hier keinerlei Schwierigkeiten. Obige Bemerkung über nicht genutzte Unterwürfel gilt analog.

3.2.2 Gitter

Ringe können als eindimensionale Gitter mit äußeren Verbindungen aufgefaßt werden (Abb. 2.6, Seite 13). Die Simulation von Gittern beliebiger Dimension und Ausdehnung ohne äußere Verbindungen läßt sich durch wiederholte Anwendung der Simulation von Ringen durchführen. Gitter mit äußeren Verbindungen in der Dimension i müssen in dieser eine gerade Anzahl von Knoten besitzen.
Sei eine d-dimensionale Gitterstruktur gegeben mit m_i Knoten in der i-ten Dimension, $0 \leq i < d$. Um eine Zuweisung von höchstens einem Knoten des Gitters an einen Knoten der Hypercube-Struktur zu erreichen, benötigt man eine Hypercube-Struktur der Dimension $n \geq \sum_{i=0}^{d-1} \lceil \log_2(m_i) \rceil$ [Sa&Sc85a, Sa&Sc88].
Jeder Dimension $i, 0 \leq i \leq d-1$, wird nun ein $\lceil \log_2(m_i) \rceil$-stelliger Gray Code zugeordnet. Dieser bildet die Verbindungen innerhalb der i-ten Dimension ab. Analog zur Abbildung von Ringen wird eine Kombination partieller, $\lceil \log_2(m_i) - 1 \rceil$-stelliger Gray Codes verwendet, falls m_i keine ganzzahlige Zweierpotenz ist. Allerdings darf dann die Anzahl der Knoten einer Dimension, in der äußere Verbindungen existieren, nicht ungerade sein, denn Ringe ungerader Länge lassen sich nicht in eine Hypercube-Struktur abbilden (Kap. 3.2.1). Die Abbildung dieser (eindimensionalen) Verbindungsstrukturen entspricht im übrigen genau der Abbildung eines Ringes. Verfügt das Gitter über keine äußeren Verbindungen, so betrachtet man die entsprechende Verbindung vom ersten zum letzten Knoten in der Hypercube-Struktur als für die Abbildung nicht relevant. Die endgültigen Knotenadressen der Hypercube-Struktur erhält man schließlich aus der Konkatenation je eines Elements der Gray Codes aller Dimensionen. Eine genauere Formulierung des Algorithmus findet sich in Abbildung 3.9.

Beispiel 3.2.3 : 6×4-Gitter, d.h. $d = 2$, $m_0 = 6$, $m_1 = 4$, Abbildung 3.10 auf Seite 36

> Entlang der nullten Dimension ist der partielle $2 = \lceil \log_2(6) - 1 \rceil$-stellige Gray Code $(0G_2(\frac{m_0}{2}), 1G_2(\frac{m_0}{2})^R)$ und entlang der ersten Dimension der $2 = \lceil \log_2(4) \rceil$-stellige Gray Code eingetragen. In kleiner Schrift sind die Knotenadressen der $n = \lceil \log_2(6) \rceil + \lceil \log_2(4) \rceil = 5$-dimensionalen Hypercube-Struktur eingetragen.
> Man erkennt, daß die Adressen benachbarte Knoten in genau einer Stelle differieren und nicht alle fünf Verbindungen pro Knoten in der Hypercube-Struktur pro Knoten genutzt werden.

3.2.3 Bäume

Bäume spielen eine wichtige Rolle als Daten- und Kommunikationsstrukturen. Als Datenstrukturen sind sie leicht manipulierbar und unterstützen effektiv vielfältige Operationen

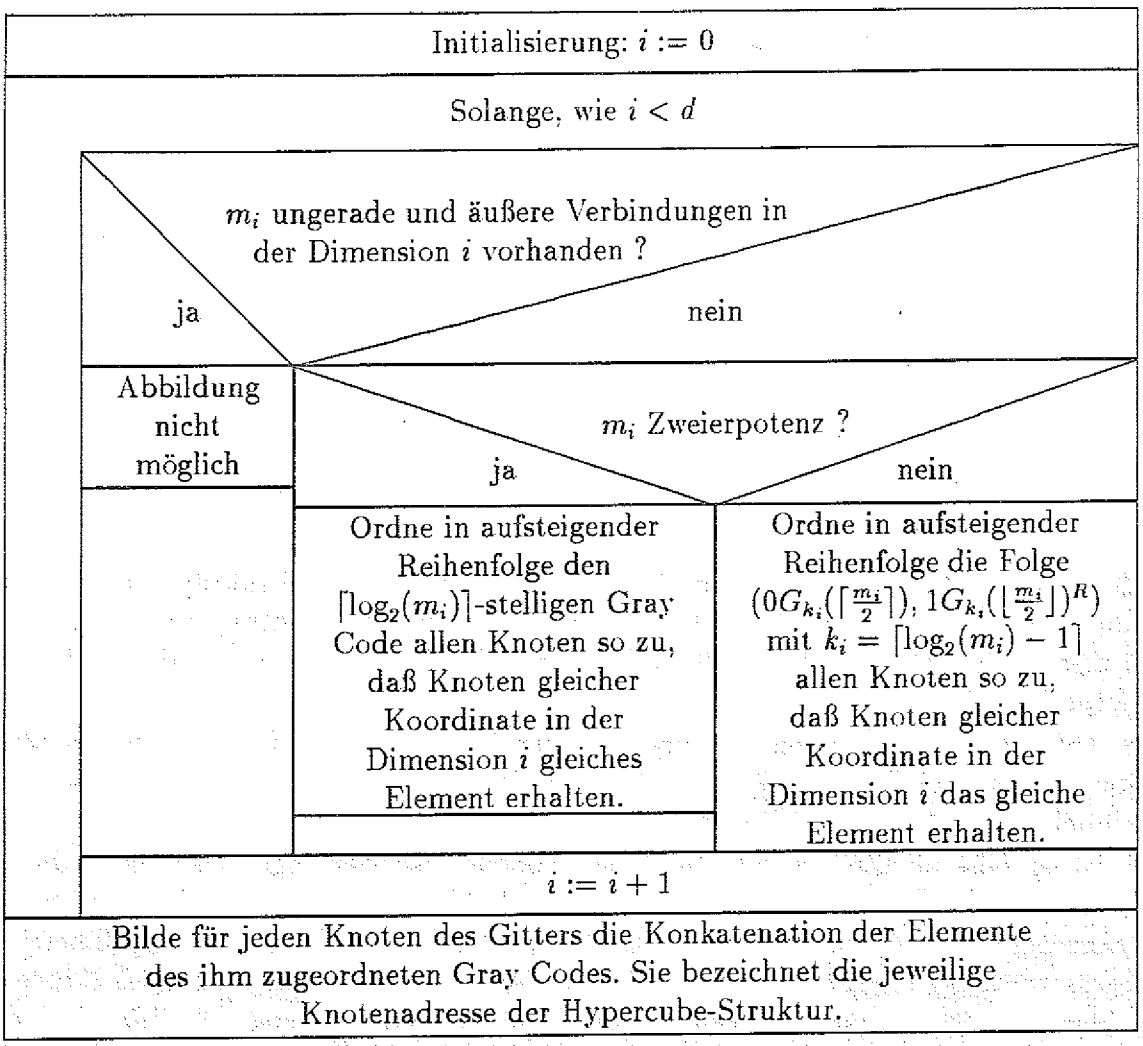


Abb. 3.9: **Algorithmus zur Abbildung eines Gitters in eine Hypercube-Struktur**

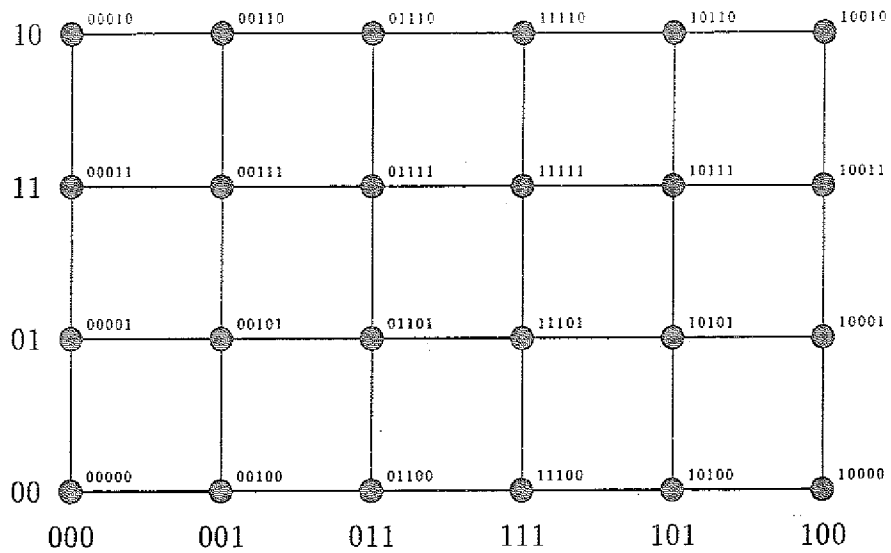


Abb. 3.10: Abbildung einer Gitterstruktur in eine Hypercube-Struktur

(z.B. Suchen und Sortieren). Weiterhin repräsentieren sie die Struktur von Divide and Conquer Algorithmen und können somit als Kommunikationsstruktur für Implementierungen dieser Algorithmen auf Parallelrechnern genutzt werden [Wagn89, Bi&Lo89, Wu85, Bh&Ip85].

Die Abbildung von Bäumen in Hypercube-Strukturen wirft aufgrund unterschiedlicher Knotenzahlen pro Ebene größere Probleme auf, als die Abbildung von — in diesem Sinne regelmäßig aufgebauten — Gittern.

Ziel ist es, eine injektive Abbildung f der Knoten eines beliebigen Baumes in die Knoten einer Hypercube-Struktur mit maximaler Prozessornutzung, minimaler Kommunikationsverzögerung und gleichmäßiger Lastverteilung in den Verbindungen zu definieren. Die Prozessornutzung (*expansion*) wird durch das Verhältnis von Knotenzahl des Hypercube-Rechners zur Knotenzahl des Baumes, die Kommunikationsverzögerung (*dilation*) durch die maximale Länge aller kürzesten Wege $(f(s), f(t))$ zwischen zwei benachbarten Knoten s, t des Baumes und die Lastverteilung (*load factor*) durch die maximale Anzahl von Benutzungen einer Verbindung des Hypercube-Rechners durch Wege $(f(s), f(t))$ zwischen benachbarten Knoten s, t des Baumes definiert [Wagn89]. Für den Fall des Erhalts aller Nachbarschaftsbeziehungen des Baumes ergibt sich eins als Wert sowohl für die Kommunikationsverzögerung als auch für die Lastverteilung.

Es werden in den folgenden Abschnitten einige allgemeine Ergebnisse zum beschriebenen

Abbildungsproblem behandelt. In Kapitel 5 wird auf die Abbildung spezieller, spannender Bäume und Graphen eingegangen, die dort als Kommunikationsstrukturen Verwendung finden.

Binäre Bäume

Zunächst wird nur die Abbildung vollständiger, binärer Bäume der Höhe n mit $2^n - 1$ Knoten in Hypercube-Strukturen betrachtet. Da an die Abbildung f die Bedingung der Injektivität gestellt wird, d.h. daß jedem Knoten der Hypercube-Struktur höchstens ein Knoten des Baumes zugeordnet wird, wird für $n > 1$ als Zielbereich von f mindestens eine Hypercube-Struktur der Dimension n benötigt. In diesem Fall können jedoch nicht alle benachbarten Knoten des Baumes auf benachbarte Knoten der Hypercube-Struktur abgebildet werden, denn die Paritäten (Kap. 3.2.1) der Bilder zweier Knoten aus benachbarten Ebenen des Baumes sind verschieden. Unter der Annahme, daß n gerade (ungerade) ist und die Wurzel des Baumes ohne Einschränkung auf den Knoten mit Adresse null abgebildet wird, existieren

$$\sum_{\substack{i=0 \\ i \text{ ungerade (gerade)}}}^{n} 2^i > 2^{n-1}, n > 2$$

Baumknoten, deren Bilder negative (positive) Parität haben. Jedoch finden sich nur

$$\sum_{\substack{i=0 \\ i \text{ ungerade (gerade)}}}^{n} \binom{n}{i} = 2^{n-1}$$

entsprechende Knoten der n-dimensionalen Hypercube-Struktur. Es existieren also nicht genügend viele Hypercube-Knoten, die die Voraussetzung erfüllen, als Bilder der Baumknoten aus geraden (ungeraden) Ebenen in Frage zu kommen. Mit anderen Worten bedeutet dies, daß keine Abbildung der Knoten eines vollständigen binären Baumes der Höhe $n > 2$ in eine Hypercube-Struktur der Dimension n mit Kommunikationsverzögerung eins und Prozessornutzung kleiner als zwei existiert.
Abbildung 3.11 verdeutlicht dies für die Höhen eins, zwei und drei. Der Abstand zwischen s und t beträgt für den Baum der Höhe drei im Bildbereich zwei. Die Kommunikationsverzögerung beträgt also ebenfalls zwei.
In [Bh&Ip85] wird jedoch gezeigt, daß alle bis auf eine Verbindung eines vollständigen, binären Baumes der Höhe n mit Kommunikationsverzögerung eins in eine Hypercube-Struktur der Dimension n eingebettet werden können. Für die eine Ausnahme kann eine Kommunikationsverzögerung mit Wert zwei garantiert werden, und zusätzlich wird der dafür benötigte Knoten für keine weitere Baumverbindung benutzt.

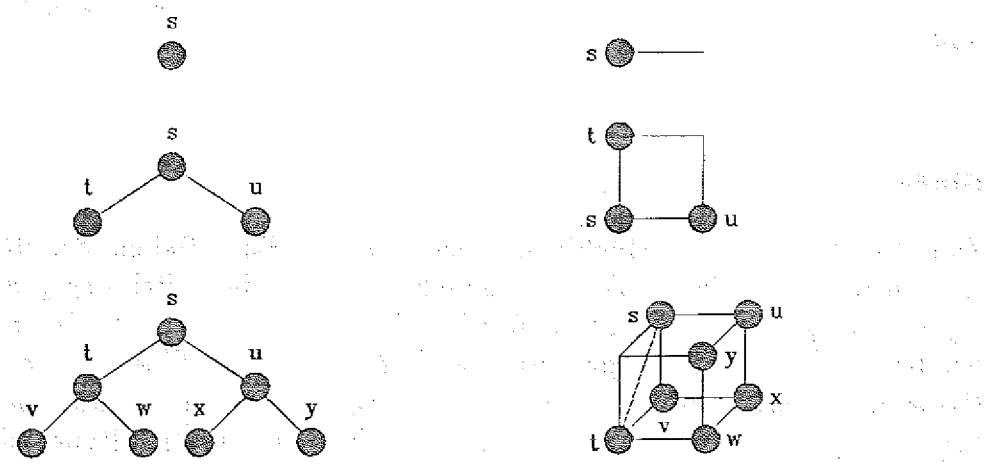


Abb. 3.11: Einbettungen binärer Bäume in Hypercube-Strukturen minimaler Dimension

Verwendet man als Zielbereich von f nun eine $(n+1)$-dimensionale Hypercube-Struktur, so kann man f so definieren, daß die Nachbarschaftsbeziehungen erhalten bleiben (Kommunikationsverzögerung gleich eins) und die Lastverteilung eins beträgt. Unabhängig von f beträgt die Prozessornutzung $\frac{2^{n+1}}{(2^n-1)} \approx 2$.

Ein Algorithmus für diesen Fall aus [Wu85] beginnt mit der Einbettung der Blätter des Baumes in jeweils einen Unterwürfel mit einem Knoten, so daß die Bilder beider Blätter einen gemeinsamen, benachbarten, noch nicht verwendeten Knoten haben. Dieser wird als Bild der Wurzel beider Blätter benutzt. Hat man auf diese Art bereits zwei (Teil-)Bäume abgebildet, so verwendet man einen freien, benachbarten Knoten der ersten Wurzel als neue Wurzel beider Teilbäume und verschiebt ggf. den zweiten Baum, damit dessen Wurzel zur neuen Wurzel benachbart ist.

Abbildung 3.12 verdeutlicht die Einbettung für Bäume der Höhen eins, zwei und drei in Hypercube-Strukturen der Dimensionen zwei, drei und vier. Man sieht, daß für Höhen kleiner als zwei je eine Hypercube-Struktur gleicher Dimension ausreicht, für Höhen größer oder gleich zwei jedoch nicht mehr. Im Fall der Höhe drei erkennt man, daß der Teilbaum mit Wurzel t im linken Würfel und der Teilbaum mit Wurzel u im rechten Würfel abgebildet ist. Statt der eingezeichneten Wahl von s kann man auch den anderen Nachbarknoten von t und u wählen.

Hätte man t und u wie in Abbildung 3.13 gewählt, so wäre die oben erwähnte Verschiebung eines Baumes erforderlich geworden, denn t und u haben keinen gemeinsamen Nachbarknoten.

Die Einbettung unvollständiger, binärer Bäume läßt sich auf genau die gleiche Weise durchführen. Gegebenenfalls lassen sich jedoch durch geschickte Wahl einige Dimensionen

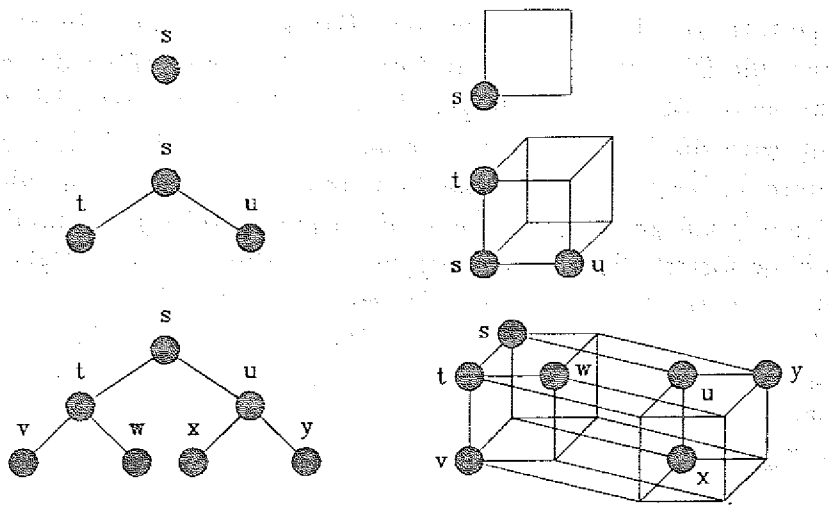


Abb. 3.12: Einbettung mit Erhalt der Nachbarschaftsbeziehungen

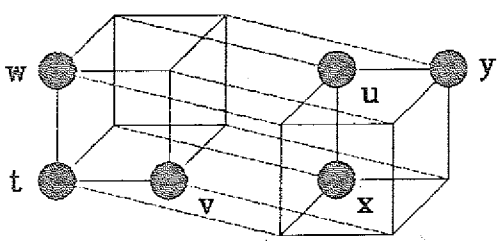


Abb. 3.13: Einbettung zweier binärer Bäume in Hypercube-Strukturen

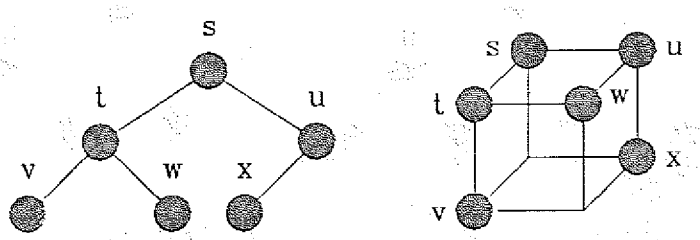


Abb. 3.14: Einbettung eines unvollständigen binären Baumes

der verwendeten Hypercube-Struktur einsparen, wie z.B. in Abbildung 3.14.
Weitere Resultate zur Einbettung binärer Bäume in Hypercube-Strukturen und den Möglichkeiten, die Dimension der Hypercube-Struktur gegenüber der damit erreichbaren Kommunikationsverzögerung abzuwägen, finden sich in [Bh&Ip85, Bhat&&86].
In [Bi&Lo89] wird die Einbettung von binären Bäumen, die als Kommunikationsmuster verwendet werden, besprochen und die Annahme getroffen, daß zu jedem Zeitpunkt ausschließlich Knoten aus genau zwei Ebenen des Baumes aktiv sind und die Berechnung von Ebene zu Ebene fortschreitet. Damit ergeben sich verschiedene Möglichkeiten zur Abbildung der Baumknoten auf die Knoten der Hypercube-Struktur. In Schleifeneinbettungen (*loop embeddings*) wird die Zuordnung eines Baumknotens und einer seiner Nachfolger auf einen Knoten der Hypercube-Struktur zugelassen. Ebeneneinbettungen (*level embeddings*) hingegen erfordern die Disjunktion der Bildermengen der Knoten zweier aufeinanderfolgender Ebenen. Zuordnungen von Knoten nicht aufeinanderfolgender Ebenen an einen Knoten hingegen sind erlaubt.

k-näre Bäume

k-näre Bäume mit höchstens $k > 2$ Nachfolgern pro Knoten lassen sich durch Einfügen von $\lceil \log_2(k) \rceil - 1$ neuen Ebenen mit höchstens $2^{\lceil \log_2(k) \rceil} - 2$ neuen Knoten zwischen einem Knoten und dessen Nachfolgern in binäre Bäume umwandeln [Wu85].

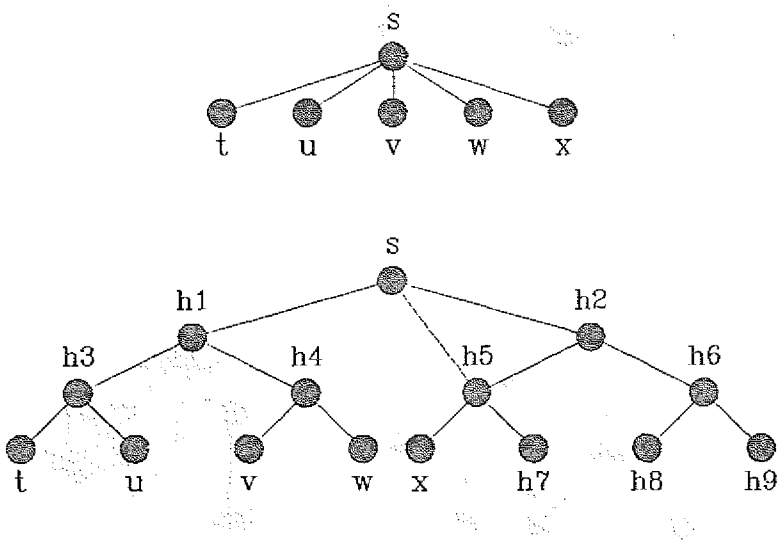


Abb. 3.15: Umwandlung eines k-nären Baumes in einen binären Baum

Der Abstand zwischen ehemals benachbarten Knoten beträgt dann $\lceil \log_2(k) \rceil$. Die Höhe

des binären Baumes wächst auf

$$n + (n-1)\left(\lceil \log_2(k) \rceil - 1\right) = \lceil \log_2(k) \rceil n - \lceil \log_2(k) \rceil + 1.$$

Abbildung 3.15 zeigt ein Beispiel einer Umwandlung eines Baumes mit fünf Nachfolgern. Es werden zwei neue Ebenen benötigt. Alle mit „H" beginnenden Knoten werden neu eingefügt. Die Knoten H2, H6, H8 und H9 sowie alle mit ihnen verbundenen Kanten können weggelassen und statt dessen die gestrichelte Verbindung benutzt werden. Man erhält in diesem Fall einen unvollständigen binären Baum.
Der umgewandelte Baum kann nun gemäß obigem Kapitel in eine Hypercube-Struktur abgebildet werden. Unter Beibehaltung der Nachbarschaftsbeziehungen beispielsweise beträgt die minimale Dimension der Hypercube-Struktur im Fall eines vollständigen binären Baumes

$$\lceil \log_2(k) \rceil n - \lceil \log_2(k) \rceil + 2.$$

3.2.4 Perfect-Shuffle

Das bekannte *Perfect-Shuffle Netzwerk* (PS-Netzwerk) wurde 1971 von Harold S. Stone eingeführt [Ston71]. Er erwähnte damals bereits die Bedeutung dieser Verbindungsstruktur für die Parallelverarbeitung und die enge Verwandtschaft mit der Hypercube-Struktur. Beide Systeme können sich wechselseitig — allerdings ohne Erhalt der Nachbarschaftsbeziehungen — simulieren.
Ein PS-Netzwerk besteht aus 2^n Knoten, die von 0 bis $2^n - 1$ numeriert werden. Es sei $s = (s_{n-1} \ldots s_0)$ die binäre Darstellung einer Knotenadresse. Dann existieren Verbindungen zwischen je zwei Knoten, deren Adressen durch eine der beiden Abbildungen PS (Perfect-Shuffle) und US (Perfect-Unshuffle) aufeinander abgebildet werden.

$$\begin{aligned} PS(s) &= (s_{n-2} \ldots s_0 s_{n-1}) \\ US(s) &= (s_0 s_{n-1} \ldots s_1) \end{aligned}$$

Kommuniziert man über eine Verbindung, die durch die Abbildung PS bzw. US gegeben ist, so spricht man von einem PS-Schritt über eine Perfect-Shuffle-Verbindung bzw. von einem US-Schritt über eine Perfect-Unshuffle-Verbindung. Ein PS-Netzwerk mit $n = 3$ ist in Abbildung 3.16 dargestellt.
Erweitert man das PS-Netzwerk um Verbindungen, die analog zu Perfect-Shuffle- und Perfect-Unshuffle-Verbindungen durch eine Abbildung EX (Exchange) definiert sind mit

$$EX(s) = (s_{n-1} \ldots s_1 \overline{s_0}) \qquad \text{und} \qquad \overline{s_0} = \begin{cases} 1 & \text{,falls } s_0 = 0 \\ 0 & \text{,falls } s_0 = 1 \end{cases}$$

so erhält man ein *Shuffle-Exchange Netzwerk* (Abb. 3.17).

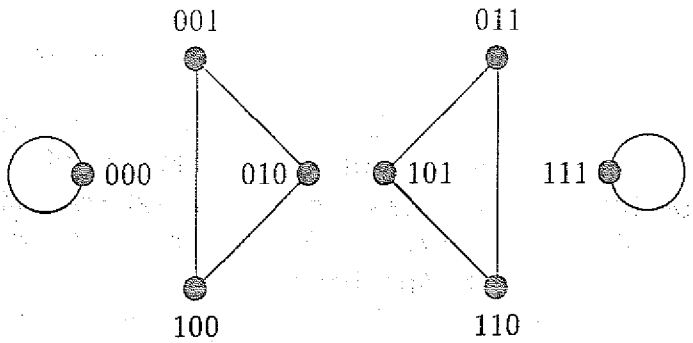


Abb. 3.16: Perfect-Shuffle Netzwerk

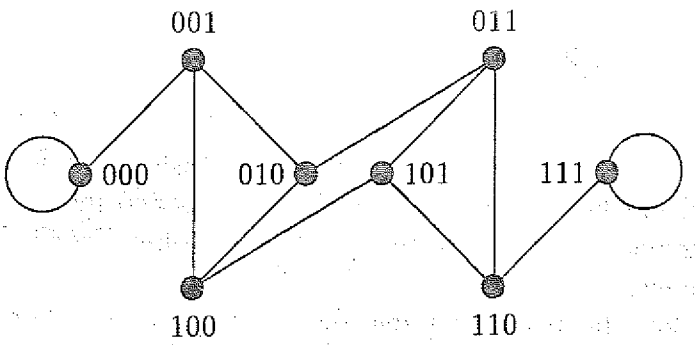


Abb. 3.17: Shuffle-Exchange Netzwerk

Ausgangspunkt	nach PS-Schritt 1	nach PS-Schritt 2	nach PS-Schritt 3
000	000	000	000
001	010	100	001
010	100	001	010
011	110	101	011
100	001	010	100
101	011	110	101
110	101	011	110
111	111	111	111

Tabelle 3.1: Perfect-Shuffle-Schritte

Tabelle 3.1 zeigt in jeder Zeile diejenigen Knoten, die nach dem i-ten PS-Schritt, $1 \leq i \leq n$, von einem bestimmten Knoten aus erreicht werden.
Interessant sind folgende Feststellungen:

1. Betrachtet man Paare von Knoten, die über eine Exchange-Verbindung direkt miteinander kommunizieren können, so erkennt man, daß nach dem i-ten PS-Schritt von jedem Paar aus ein Paar erreicht wird, deren Knotenadressen sich genau an der Stelle i mod n unterscheiden.

2. Nach n PS-Schritten erreicht man wieder die Ausgangsposition.

Daraus folgt, daß ein PS-Netzwerk die Verbindung über die Dimension $i, 0 \leq i < n$, einer Hypercube-Struktur simulieren kann durch

- $n-i$ PS-Schritte oder i US-Schritte, (Verschiebung des i-ten Bits an die Stelle null),
- einen EX-Schritt (Invertieren des Bits an der Stelle null) und
- i PS-Schritte oder $n-i$ US-Schritte, (Verschiebung des nullten Bits an die Stelle i).

Ein Hypercube-Rechner der Dimension n kann die PS-, US- und EX-Schritte eines PS-Netzwerks mit 2^n Knoten wie folgt simulieren:

PS-Schritt: Vergleiche von $i = n-1$ bis 0 paarweise benachbarte Bits $(i, (i+1) \bmod n)$ des Startknotens und kommuniziere über Dimension $(i+1) \bmod n$, falls die Bits ungleich sind.

US-Schritt: Analog zu PS-Schritt, jedoch kommuniziere im Fall differierender Bits über Dimension i.

EX-Schritt: Kommuniziere über Dimension null.

3.3 Vergleich von Netzwerken

Tabelle 3.2 zeigt einen Vergleich verschiedener Eigenschaften der n-dimensionalen Hypercube-Struktur mit Ringen, Gittern und vollständigen Verbindungen der Knotenzahl 2^n, Cube Connected Cycles mit $n \cdot 2^n$ und binären Bäumen mit $2^n - 1$ Knoten [Do&Ja87, John88, Raab&&88].

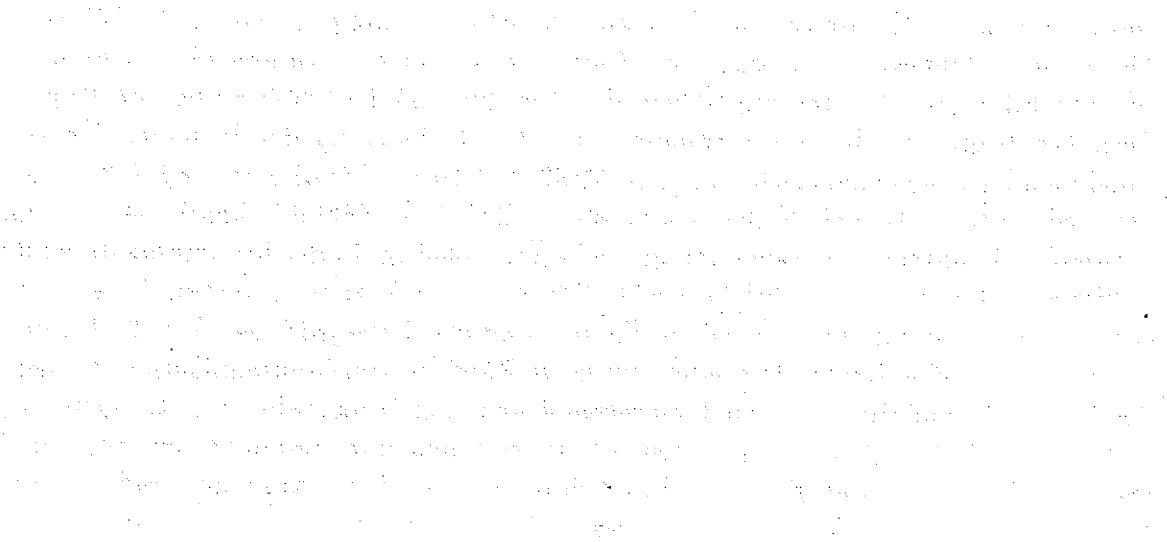
Eigenschaften	Netzwerke					
	Gitterstrukturen			Würfelstrukturen		
	Ring	Torus [1]	bin. Baum	n-cube	CCC	vollst.
Knotenzahl	2^n	2^n	$2^n - 1$	2^n	$n \cdot 2^n$	2^n
Dimension	1	2	$n - 1$ [2]	n	n	—
modularer Aufbau	nein	nein	ja	ja	nein	ja
Erweiterbarkeit [3]	ja	ja	ja	nein	ja	nein
Routing	einfach	einfach	einfach	einfach	aufwendig	trivial
DA	$\approx 2^{n-2}$	$\approx 2^{\frac{n}{2}-1}$	— [4]	$\approx \frac{n}{2}$	$\approx \frac{7}{4} \cdot n - 3$	1
Durchmesser	2^{n-1}	$\approx 2^{\frac{n}{2}}$	$2 \cdot (n-1)$	n	$\leq \frac{5 \cdot n - 4}{2}$	1
Ports	2	4	≤ 3	n	3	$2^n - 1$
Fehlertoleranz	1	3	0	$n - 1$	2	$2^n - 2$
VL	2^n	2^{n+1}	$2^n - 2$	$n \cdot 2^{n-1}$	$3 \cdot n \cdot 2^{n-1}$	$\approx 2^{2 \cdot n - 1}$
DVD	$\approx 2^{n-2}$	$\approx 2^{\frac{n}{2}-2}$	—	≈ 1	$\approx n$	$\approx \frac{1}{2^{n-1}}$
GB	2^{n+1}	2^{n+2}	$2^{n+1} - 4$	$n \cdot 2^n$	$3 \cdot n \cdot 2^n$	$\approx 2^{2 \cdot n}$

[1] betrachte hier $\sqrt{2^n} \times \sqrt{2^n}$ Torus
[2] Höhe des Baumes
[3] im Sinne der Möglichkeit der Weiterverwendung von Bauteilen (Prozessoren) aus kleinen Netzwerken in größeren zu verstehen
[4] Definition des durchschnittlichen Abstands für inhomogene Netzwerke nicht anwendbar

Tabelle 3.2: Vergleich von Eigenschaften verschiedener Netzwerke

Neben einfachem Routing, modularem Aufbau und geringer durchschnittlicher Verkehrsdichte DVD zählen die gute Fehlertoleranz, der geringe Durchmesser sowie der geringe durchschnittliche Abstand DA zu den Stärken der Hypercube-Struktur. Abgesehen von der in der Praxis bei großer Prozessorzahl technisch kaum noch realisierbaren vollständigen Verbindung besitzt die Hypercube-Struktur für diese Eigenschaften die günstigsten Werte

und eine, von der Knotenzahl unabhängige, konstante, durchschnittliche Verkehrsdichte. Als Nachteil hingegen wird meist die logarithmisch in der Knotenzahl wachsende Zahl der Anschlüsse pro Knoten angesehen, die eine dynamische Erweiterung von Hypercube-Rechnern begrenzt oder gar verhindert und die Realisierung der einzelnen Knoten mit zunehmender Dimension erschwert [Go&Mr89, Su&Ba77, Raab&&88, Do&Ja87]. L.G. Valiant jedoch führt in [Vali88] das Argument an, daß diese Zahl der Anschlüsse pro Knoten notwendig ist unter der Voraussetzung, daß keine Lokalität in der Kommunikation auftritt. In diesem Fall wird eine Nachricht gewöhnlich über eine Anzahl von Verbindungen übertragen, die dem Durchmesser des Netzwerks n entspricht. Falls jeder der $N = 2^n$ Knoten nun pro Zeiteinheit eine Nachricht sendet und keine Häufung von Kommunikation in einem Teil des Netzwerks auftritt, so ist die Gesamtbandbreite der Hypercube-Struktur genau auf die $n \cdot N$ pro Zeiteinheit unter den gegebenen Voraussetzungen im gesamten Netzwerk zu übertragenden Nachrichten ausgerichtet. Fraglich ist jedoch, ob die Annahme, daß während der Berechnungen keine Lokalität auftritt, realistisch ist, denn tatsächlich tritt sie in vielen Algorithmen aus den verschiedensten Gebieten auf, und schließlich zählt die Möglichkeit der leichten Nutzung von Lokalität zu den Stärken von Multicomputern [Sa&Sc85a, Sa&Sc88].



4 Hypercube-Rechner und deren Aufbau

In diesem Kapitel wird der Aufbau von Hypercube-Rechnern beschrieben. Besondere Berücksichtigung werden dabei die Rechner des auf diesem Gebiet wohl bedeutendsten Herstellers (Intel) erfahren.

4.1 Allgemeine Charakterisierungen

Ein typischer Hypercube-Rechner ist ein Multicomputer, d.h. er ist aus eigenständigen, meist identischen, Rechnern (Knoten) aufgebaut. Diese sind untereinander statisch nach Art der Hypercube-Struktur mittels bidirektionaler Leitungen (Kap. 3) vernetzt. Kommunikation zur Synchronisation und zum Datentransfer findet zwischen je zwei verbundenen Knoten statt (nearest-neighbour communication). Nachrichten für nicht verbundene Knoten werden von Knoten zu Knoten durch das Netzwerk ans Ziel geleitet (Routing). Der Hauptspeicher des Systems ist lokal in den Knoten realisiert, und jeder Knoten hat nur auf den eigenen Speicherbereich Zugriff. Die Leistungsfähigkeit der verwendeten Prozessoren wird im allgemeinen mit „medium" bezeichnet und entspricht in etwa der einer Workstation. Meist besitzt jeder Knoten eigene Einheiten für Kommunikation und numerische Berechnungen und verfügt über eine Kopie des Betriebssystemkerns und des Anwendungsprogramms. Die Programmierung des Hypercube-Rechners erfolgt über einen *Host-Rechner*, der mit einem oder mehreren Knoten verbunden ist, und diese sowohl mit Programmen als auch mit Daten versorgt [Do&Du89, Wile87, Fox&&88].

4.2 Spezielle Bewertungskriterien

Für die Bewertung von Hypercube-Rechnern können die folgenden Merkmale herangezogen werden [Bo&Ro89]:

1. Wieviele und welche Prozessoren werden verwendet und wie leistungsfähig sind sie?
2. Wie ist der übrige Knoten aufgebaut?
   - Sind Koprozessoren für numerische Berechnungen vorhanden?
   - Existierern Prozessoren für Kommunikationsaufgaben?
   - Über wieviel Hauptspeicher verfügt der Knoten?
3. Wie schnell können die Knoten miteinander kommunizieren?
   - Wie groß ist die Startzeit t_{start} einer Übertragung?
   - Wie groß ist die (asymptotische) Kommunikationsgeschwindigkeit t_{comm}?
4. Wie ist das Verhältnis von Kommunikationszeit pro übertragenem Wort bei k insgesamt übertragenen Worten $t_{send}(k) = (t_{start} + k \cdot t_{comm})/k$ zur Rechenzeit pro Gleitkommaberechnung t_{calc}?
5. Welche Host-Rechner und Peripheriegeräte können verwendet werden?
6. Welche Programmiersprachen, Anwendungsprogramme und Entwicklungsumgebungen (Kompiler,Debugger, Optimierer,...) sind verfügbar?
7. Wieviel kostet der Rechner?

Die Startzeit t_{start} hängt neben der Bandbreite der benutzten Kommunikationsverbindungen von der Aufwendigkeit des benutzten Protokolls, z.B. den darin verwendeten Algorithmen zur Fehlererkennung, ab.
Der vierte Punkt gibt Aufschluß darüber, ob sich ein gegebenes Problem der Größe k in sinnvoller Weise auf mehrere Prozessoren aufteilen läßt. Dies ist der Fall, falls

$$\frac{t_{send}(k)}{t_{calc}} < 1.$$

Man spricht in diesem Fall vom einem Rechner, dessen Kommunikations- und Rechenleistung „balanciert" ist [Bo&Ro89]. Es lohnt sich dann, die Daten an einen Nachbarprozessor zu senden und die Berechnungen dort durchzuführen, anstatt selbst zu rechnen. Je kleiner der Maximalwert von k ist, der obige Bedingung erfüllt, desto einfacher ist die Aufteilung

von auszuführenden Berechnungen und der dafür benötigten Daten auf verschiedene Knoten. Dieser Maximalwert kann nur durch Messungen von $t_{send}(k)$ für verschiedene Werte von k ermittelt werden, denn t_{comm} wird meist nur für große Mengen zu übertragender Daten annähernd erreicht.
Weiterhin ist von Bedeutung, ob der Rechner kommerziell vertrieben wird, ob er sich noch im Prototypstadium befindet oder ob er nur zu Forschungszwecken eingesetzt wird und welche Firma/Forschungsgruppe ihn ab wann und in welchen Ausbaustufen anbietet/entwickelt.

4.3 Kommerzielle Rechner

4.3.1 iPSC

iPSC/1 und iPSC/2

1985 brachte die 1984 gegründete Firma iSC (intel Scientific Computing) den ersten kommerziellen Hypercube-Rechner *iPSC/1* auf den Markt. Er ging aus dem bei Caltech (California Institute of Technologie) entwickelten Cosmic Cube hervor und besteht aus maximal 128 Intel 80286 Prozessoren mit numerischem Koprozessor 80287 und jeweils 0.5 MByte Hauptspeicher [Ratt85]. Das Verhältnis von $t_{send}(k)$ zu t_{calc} wird ab ca. 100 Byte kleiner als eins [Bo&Ro89].
Seit Dezember 1987 wird der *iPSC/2* angeboten, der gegenüber dem iPSC/1 in einigen Punkten wesentliche Verbesserungen erfahren hat. Die maximal 128 Knoten des iPSC/2 sind mit den Prozessoren Intel 80386/80387 ausgerüstet, die eine numerische Rechenleistung von 0.35 MFLOPS (Million Floating Point Operations per second, Millionen Gleitkomma Operationen pro Sekunde) bei einer Taktrate von 16 MHz liefern. Der Hauptspeicher kann von einem bis zu 16 MByte ausgerüstet werden, ein Cache der Größe 64 KByte steht zur Verfügung, und es kann sowohl eine Skalar- (*Intel SX scalar extension module*) als auch eine kanalisierte Vektoreinheit (*vector co-processor*) nachgerüstet werden, die einfach durch einen Bus mit dem restlichen Knoten verbunden werden. Die erweiterten Systeme werden iPSC/2-SX bzw. iPSC/2-VX genannt. Abbildung 4.1 zeigt den Knotenaufbau des iPSC/2-VX [Clos88].
Die Kommunikation zwischen Prozessoren wird ausschließlich über das *Direct Connect Module* (DCM) auf Hardware-Ebene abgewickelt, d.h. kein Prozessor eines Knotens wird mit Kommunikationsaufgaben aufgehalten. Abbildung 4.2 zeigt den Aufbau des DCM [Nuge88]. Es besteht aus acht voneinander unabhängigen, bitseriellen, bidirektionalen Kanälen, die durch je ein Routing-Element gesteuert werden und durch zwei unidirektionale Busse mit dem restlichen Knoten verbunden sind. Ein Kanal ist für I/O zu externen Speichern und für Kommunikation mit dem Host-Rechner reserviert. Dementsprechend können bis zu $2^7 = 128$ Knoten in Hypercube-Struktur verbunden werden. Pfade werden dynamisch aufgebaut und exklusiv für eine Übertragung reserviert, d.h. es wird kein Multiplex-Verfahren benutzt, und damit sind die beteiligten Kanäle für die Dauer der Kommunikation blockiert. Die Übertragung wird nach Aufbau des gesamten Pfads nach Art eines *circuit-switched* Netzwerks durchgeführt. Kennzeichnend für diese Art der Übertragung ist das Senden der Nachricht in einem Paket nach Aufbau des gesamten Pfads. Damit entfällt die Notwendigkeit, Daten in Zwischenknoten puffern zu müssen, wie dies beim *wormhole-routing* [Da&Se87, Gr&Re89] notwendig ist. Außerdem wird im Gegensatz zum *store-and-forward-routing* [Gr&Re89] kein Zwischenknoten in der Ausführung seiner Tätigkeiten beeinträchtigt, denn lediglich das DCM ist an der Übertragung beteiligt. Insgesamt wird der Da-

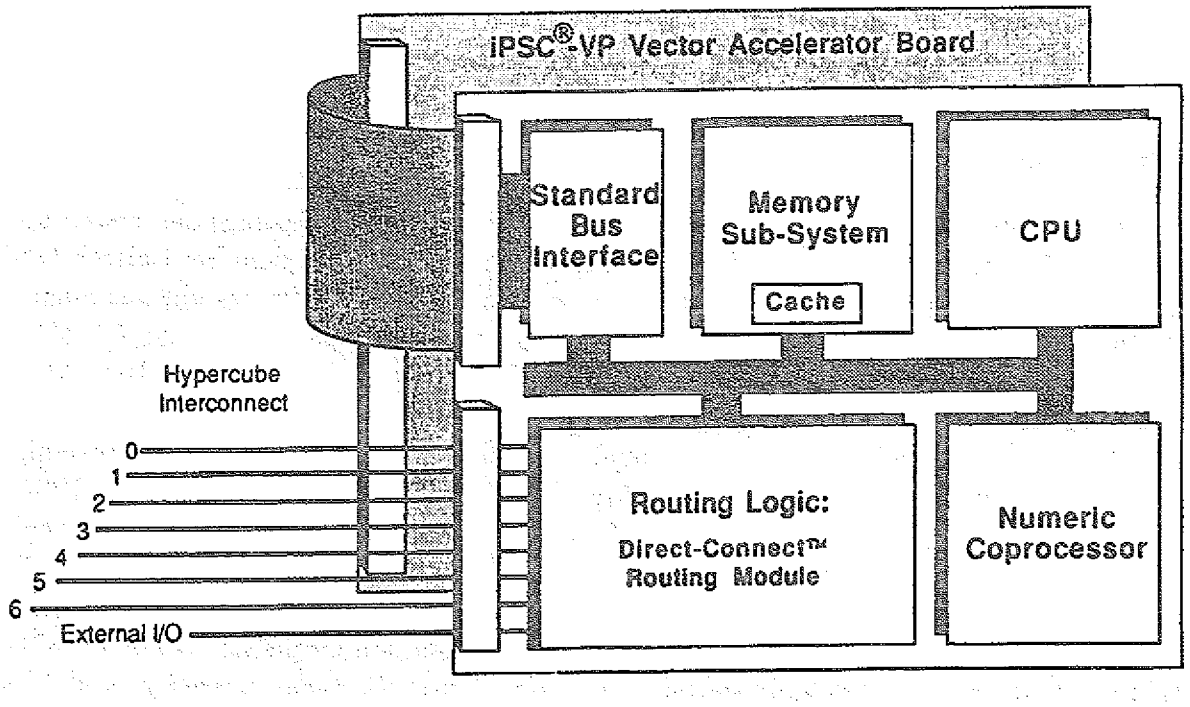


Abb. 4.1: Knotenaufbau eines iPSC/2-VX

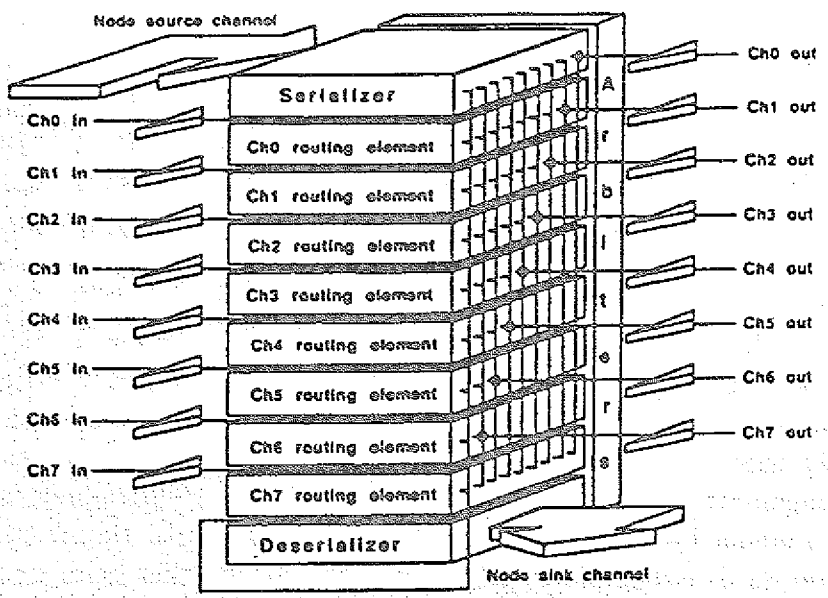


Abb. 4.2: Aufbau des Direct Connect Module (DCM)

tentransfer wesentlich beschleunigt und nur ausgesprochen wenig Speicher für den Puffer benötigt. Dieser hat die relativ geringe Größe von zwei mal vier KByte pro DCM und arbeitet nach dem FIFO-Prinzip.

Verschickte Nachrichten können beliebig lang sein und mit 2.8 MByte/sec bidirektional gesendet werden. t_{start} beträgt bei Nachrichten kleiner als 100 Byte 350μs und für größere Nachrichten 660μs. Die beiden verschiedenen Werte werden durch das benutzte Kommunikationsprotokoll hervorgerufen, welches eine unterschiedliche Behandlung von Nachrichten kleinerer und größerer Länge als 100 Byte vorsieht. Nachrichten kleiner als 100 Byte werden in speziell hierfür vorgesehenen Puffern gespeichert, die jeder Knoten für alle anderen Knoten bereitstellt. Jeder Knoten hat Kenntnis über die Anzahl der freien Puffer, die die anderen Knoten für ihn zur Verfügung stellen. In regelmäßigen Abständen werden alle Knoten über die Anzahl der für sie freigewordenen Puffer unterichtet. Durch diese Pufferverwaltung kann eine Nachricht kleinerer Länge als 100 Byte direkt gesendet werden, falls der Empfänger noch über einen freien Puffer für den Sender verfügt. Nachrichten größerer Länge hingegen erfordern eine Anfrage auf genügend Speicherplatz beim Empfänger und müssen erst dessen Bestätigung abwarten und gegebenenfalls solange auf das Senden warten, bis genügend Speicherplatz frei geworden ist [Pier88].

Mit den obigen Werten von t_{start} und t_{comm} wird das Verhältnis von $t_{send}(k)$ zu t_{calc} ab ca. $k = 512$ übertragenen Byte kleiner als eins [Bo&Ro89]. Dies bedeutet, daß aufgrund des kleineren Werts von $k = 100$ für den iPSC/1 die Aufteilung eines Problems auf diesem Rechner einfacher zu realisieren ist.

Alle Intel Hypercube-Rechner der Serie iPSC/2 sind mit dem DCM ausgerüstet, während die Knoten des iPSC/1 — einem Hypercube der sogenannten ersten Generation — gemäß dem *packet-switched* Modell kommunizierten. Dies bedeutet, daß ein gesendetes Paket von einem Knoten erst vollständig empfangen worden sein muß, bevor es weitergeleitet werden kann.

Als Host-Rechner (*System Resource Manager*, SRM) können SUN-Workstations unter UNIX benutzt werden. Seit September 1988 steht das CFS (Concurrent File System) zur Verfügung, welches UNIX-ähnliche Funktionen zur Verfügung stellt, jedoch mit einer Vielzahl von Platten arbeitet, auf die parallel zugegriffen wird. Beispielsweise können Teile einer Datei auf verschiedenen Platten gespeichert werden und durch verschiedene Wege parallel in den Hauptspeicher eines Knotens geladen werden. Die Platten, die dem Benutzer wie eine einzige logische Platte erscheinen, sind mit in der Zahl erweiterbaren, speziellen I/O-Knoten über Busse verbunden.

Der iPSC/2 kann in disjunkte Unterwürfel aufgeteilt und diese verschiedenen Benutzern zugeteilt werden. Als Knotenbetriebssystem wird NX/2 [Pier88] verwendet, welches maximal 20 Prozesse pro Knoten unterstützt und Kommunikationsaufrufe für Kommunikation mit anderen Prozessoren und für Ein- und Ausgabe bereitstellt. Als Programmiersprachen sind FORTRAN, C, Common LISP und ADA verfügbar. Ein Vektorisierer für FORTRAN

Programme, ein sog. MIMD'izer, welcher herkömmliche Programme in MIMD-Programme umwandeln kann, sowie ein Debugger auf Quellcodeebene (DECON, [Pa&Ja88]) und verschiedene Bibliotheken zur Lösung numerischer Probleme werden ebenfalls angeboten [INTEL88].
Bis 1989 wurden weltweit 175 iPSC/2 verkauft [Arch90]. In der Bundesrepublik Deutschland verfügen z.B. die GMD, St. Augustin, die Universität Stuttgart, AEG Berlin und die Gesellschaft für Strahlen- und Umweltforschung (GSF), München über Intel Hypercube-Rechner verschiedener Ausführungen [Fox&&88, Bo&Ro89, Gehr&&88, INTEL90, Wile87].

iPSC/860

Der *iPSC/860*, das jüngste Produkt von iSC, ist das erste kommerzielle Produkt des Touchstone Projekts von Intel und DARPA (Defense Advanced Research Projects Agency), welches zum Ziel hat, bis zum Jahr 1992 Parallelrechner mit bis zu 2048 Knoten und einer maximalen Rechenleistung von 150 GFLOPS zu bauen. Der iPSC/860 wurde im Januar 1990 der Öffentlichkeit vorgestellt und an die ersten Kunden ausgeliefert. Auffälligster Unterschied zu seinen Vorgängern ist die stark angestiegene Rechenleistung pro Knoten. Ein iPSC/860 umfaßt acht bis 128 Knoten mit jeweils einem Prozessor Intel 80860 (i860). Herausragende Merkmale des i860 sind seine RISC-Architektur (siehe Kap. 2.2) und verschiedene Funktionseinheiten, die in einem Kanal zusammengeschaltet werden können und zu einer maximalen numerischen Rechenleistung von 60 MFLOPS pro Knoten mit 64 Bit Genauigkeit und 80 MFLOPS pro Knoten mit 32 Bit Genauigkeit bei einer Taktrate von 40 Mhz führen. Die gesamte Rechenleistung des iPSC/860 liegt also bei maximal 10.24 GFLOPS. Pro Knoten können ein bis 16 MByte Hauptspeicher installiert werden und maximal sieben Leitungen stehen für Kommunikation mit anderen Knoten zur Verfügung. Eine Leitung ist für Plattenzugriff oder Verbindung mit dem, auf 80386/80387 Prozessoren basierenden, unter UNIX System V betriebenen SRM reserviert.
Als Kommunikationssystem wird weiterhin das DCM verwendet, welches sich in Ausbau und Leistungsfähigkeit im Vergleich zum iPSC/2 nicht grundlegend geändert hat. Lediglich die Kontrolle des Datentransfers wird beim iPSC/860 durch den äußerst leistungsfähigen, zentralen Prozessor selbst ausgeführt, während diese Aufagbe beim iPSC/2 zwei *DMA Controller* (Direct Memory Access Controller) übernehmen. Beim iPSC/860 verbessern sich die Startzeiten t_{start} deshalb auf 69 μs für Nachrichten kleinerer Länge als 100 Byte und auf 178 μs für längere Nachrichten. Die Übertragungsrate von 2.8 MByte/sec bleibt erhalten [Berr&&90].
Mit der asymptotisch im Vergleich zum iPSC/2 unveränderten Kommunikationsgeschwindigkeit und der stark angestiegenen Rechenleistung der Einzelprozessoren ($t_{calc} = 0.2\mu$s) hat sich das Verhältnis von Rechen- zu Kommunikationsgeschwindigkeit nachteilig für die Kommunikation geändert. Dies äußert sich auch darin, daß das Verhältnis von $t_{send}(k)$

zu t_{calc} für kein k kleiner als eins ist. Damit ist der iPSC/860 nach der Terminologie aus Kapitel 4.2 für kein k balanciert. Detailiertere Leistungsbewertungen finden sich in [Berr&&90, Heat&&90].
Der iPSC/860 kann z.B. über TCP/IP und VMS-Link an verschiedene Netzwerke angeschlossen werden. Als Knotenbetriebssystem kann weiterhin NX/2 verwendet werden. Generell ist Quellcodekompatibilität zu den Vorgängermodellen gegeben.
Folgende Software-Pakete werden angeboten:

- C- und vektorisierender FORTRAN-Kompiler
- ein Programm zur Portierung von FORTRAN-Programmen mittels Abhängigkeitsanalysen (FORGE)
- UNIX Tools
- Concurrent Debugger
- eine mathematische Bibliothek
- ein System Simulator u.a.

Zur Zeit verfügt in der Bundesrepublik z.B. die GSF, München über einen iPSC/860 [INTEL90, Marg90].

4.3.2 NCUBE

Seit 1985 bietet die Firma *NCUBE*, Beaverton/Oregon einen gleichnamigen Hypercube-Rechner an. Er kann in drei Ausbaustufen von 16 bis zu 1024 Knoten mit je 512 KByte Hauptspeicher umfassen. Jeder Knoten ist mit einem, von NCUBE selbst entwickelten, Prozessor auf 32-bit Basis ausgerüstet und kann mit einer kanalisierten Vektoreinheit nachgerüstet werden. Die verwendeten Host-Rechner basieren auf Intel 80x86 Prozessoren. Je 64 Knoten sind auf einer Prozessorkarte untergebracht. Für je zwei Prozessorkarten wird eine I/O-Karte benötigt, die aus 16 Prozessoren besteht. Jeder Prozessor einer Prozessorkarte ist durch einen Kanal direkt mit einer I/O-Karte verbunden [Palm85, Palm88].
Abbildung 4.3 zeigt die drei Ausbaustufen des NCUBE und Abbildung 4.4 ein Prozessor-Board eines NCUBE/four, der bis zu 16 Prozessoren umfaßt. Die Prozessor- und I/O-Boards der NCUBE/ten und NCUBE/seven sind identisch, während die Prozessorkarten für den NCUBE/four als Einsteckkarten für IBM PC-AT kompatible Rechner angeboten werden, welcher auch als Host-Rechner benutzt wird.
Kompiler für FORTRAN 77, C und ein UNIX-ähnliches Betriebssystem werden angeboten. Die Knoten des Rechners können aufgeteilt und verschiedenen Benutzern zugewiesen werden.

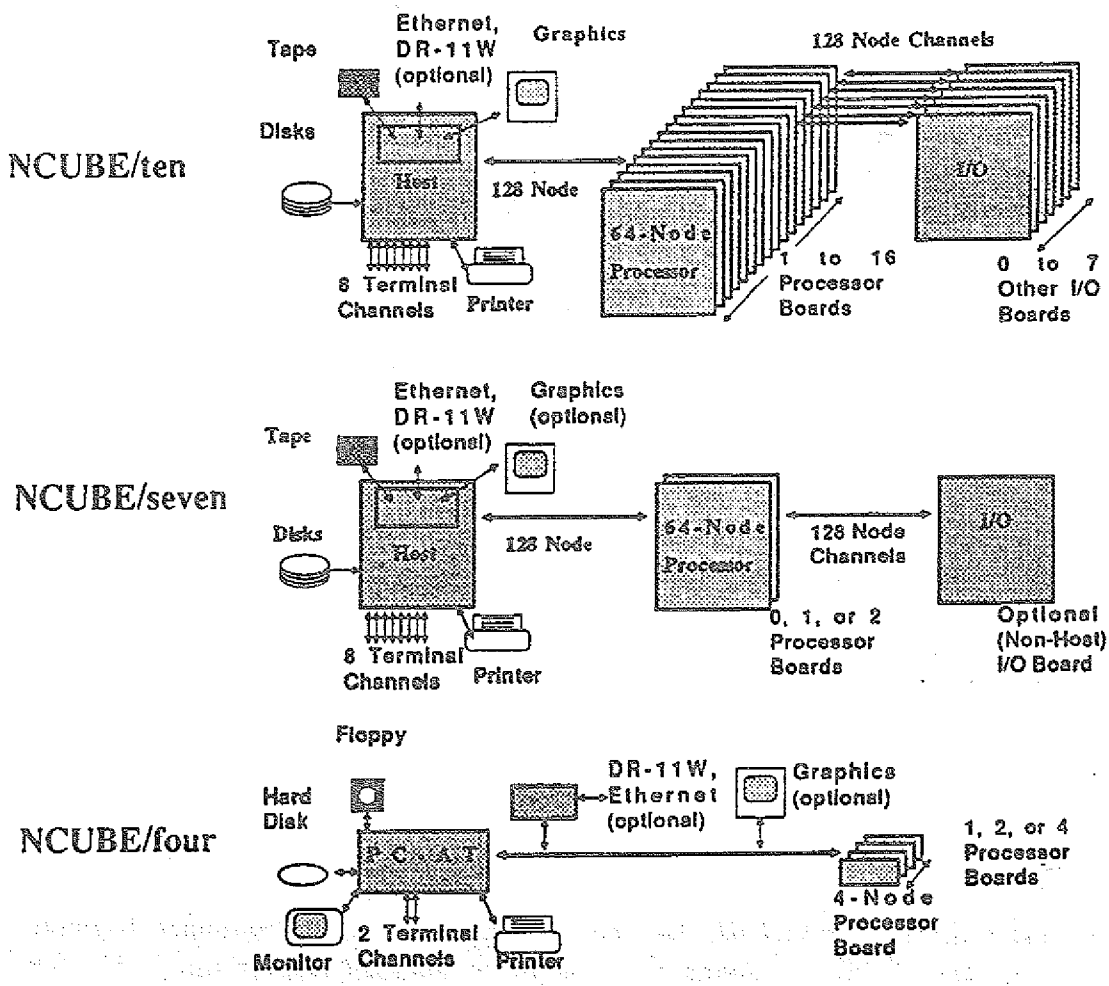


Abb. 4.3: Angebotene Ausbaustufen des NCUBE

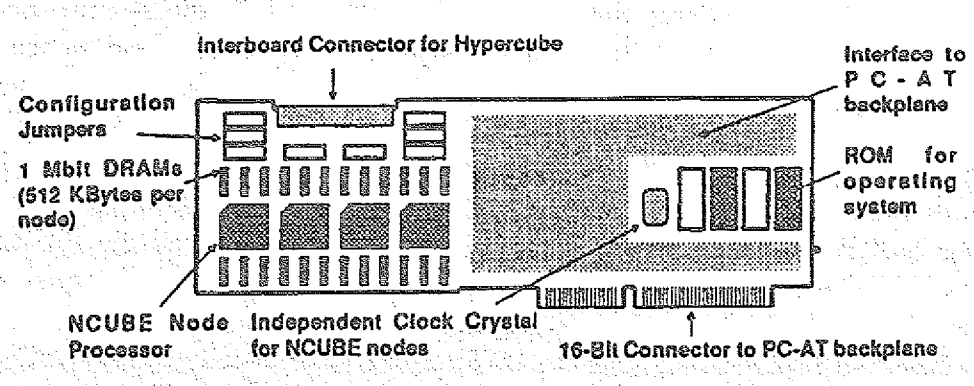


Abb. 4.4: Aufbau eines Prozessor-Boards des NCUBE/four

Mit einer Rechenleistung von 0.3-0.5 MFLOPS pro Knoten und einer maximalen Kommunikationsgeschwindigkeit von acht MBit/sec pro Verbindungsleitung ist der NCUBE weit weniger leistungsfähig als die Intel Hypercube-Rechner iPSC/2 und iPSC/860 [Do&Du89, Gehr&&88]. Jedoch ist das Verhältnis von $t_{comm}(k)$ zu t_{calc} ähnlich wie beim iPSC/1 ab ca. 100 Byte kleiner als eins [Bo&Ro89] und liegt damit günstiger als für die Systeme iPSC/2 und iPSC/860. Eine genauere Analyse der Leistungsfähigkeit des NCUBE im Vergleich mit dem iPSC/1 findet sich in [Duni86, Re&Fu87, Sh&Fi89].
Bis 1989 wurden ca. 88 NCUBE-Rechner verschiedener Ausführungen installiert [Arch90]. Die Firma NCUBE bietet laut [Heat&&90] inzwischen eine neue Produktreihe von Hypercube-Rechnern kommerziell an, deren Leistungsfähigkeit bis in den GFLOPS-Bereich reichen soll.

4.3.3 Connection Machine

Thinking Machines in Cambridge, USA bietet seit 1986 die *Connection Machine* (CM) an, die seit April 1987 in der zweiten Version CM-2 verkauft wird. Die Connection Machine arbeitet auf SIMD-Basis und ist somit kein typischer Vertreter eines Hypercube-Rechners. Maximal 65536 Ein-bit Prozessoren können in Gruppen zu 16 Stück auf je einem Chip untergebracht werden. Die Prozessoren eines Chips mit je acht KByte Hauptspeicher sind vollständig verbunden und nur die Verbindungen der Chips untereinander weisen eine Hypercube-Struktur auf. Die Kommunikationsgeschwindigkeit beträgt maximal drei GBit/sec. Numerische Koprozessoren können nachgerüstet werden, und als Host-Rechner stehen z.B. SUN-Workstations zur Verfügung.
Bis 1989 wurden ca. 55 Connection Machines der ersten und ca. 15 Connection Machines der zweiten Version installiert.
Für die Zukunft ist der Bau einer Connection Machine mit bis zu einer Million Prozessoren geplant [Fren86, McBr89, Arch90].

4.4 Projekte, Prototypen, Sonstige

4.4.1 FPS - T-Serie

Ab 1986 bot Floating Point Systems in Beaverton/Oregon einen maximal 16384 Knoten umfassenden Hypercube-Rechner an, dessen Vertrieb inzwischen eingestellt wurde. Jeder Knoten bestand aus einem INMOS Transputer für Kommunikationsaufgaben, einer Vektoreinheit, einem zentralen Prozessor und einem MByte Hauptspeicher. Pro Knoten konnte eine Rechenleistung von 16-20 MFLOPS erreicht werden [Fren86, Gehr&&88, Do&Du89].

4.4.2 Cosmic Cube

Der von Caltech Anfang der achtziger Jahre entwickelte *Cosmic Cube* war der erste funktionierende Prototyp eines Hypercube-Rechners. Er bestand aus 64 Knoten mit jeweils 128 KByte Hauptspeicher und den Prozessoren Intel 8086/8087. Die Kommunikationsgeschwindigkeit betrug zwei MBit/sec. Erste Messungen der Rechenleistung ergaben einen Speedup von ca. 50 gegenüber der Rechenleistung eines Knotens [Seit85]. Der Erfolg dieses Projekts gab den ersten Anstoß zur Entwicklung kommerziell vertriebener Hypercube-Rechner.

4.4.3 Mark II, Mark IIIfp

Die Hypercube-Rechner *Mark II* und *Mark IIIfp* von JPL (Jet Propulsion Laboratory) und Caltech sind Weiterentwicklungen des Cosmic Cube und erfuhren im wesentlichen Verbesserungen bezüglich Rechenleistung (Mark II: maximal 64 80286/80287 Prozessoren, Mark IIIfp: maximal 1024 68020/68881 Prozessoren mit weiteren numerischen Co- und Vektorprozessoren) und Kapazität des Hauptspeichers (256 KByte bzw. vier MByte). Mark IIIfp unterstützt weiterhin Kommunikation auf Hardware-Basis [Fox&&88, Wile87, Fox&&87]. Der Knotenaufbau eines Mark IIIfp ist in Abbildung 4.5 dargestellt [Tuaz&&88].
In [Gr&Re86] werden die Leistungsfähigkeiten der Rechner Mark IIIfp, iPSC/1 und System 14, Ametek verglichen.

4.4.4 Navier-Stokes Computer

Der *Navier-Stokes Computer* (Princeton University, CCI Corporation) ist ein speziell für numerische Probleme mit großer Vektorlänge, z.B. Windkanalmodellierungen, gebauter Hypercube-Rechner. Die maximal 128 Knoten sind sehr leistungsstark und umfassen jeweils 32 arithmetische Funktionseinheiten, die in einem Kanal mit einer Rechenleistung von 640 MFLOPS zusammengeschaltet werden können. Der Hauptspeicher beträgt zwei GByte und die Kommunikationsgeschwindigkeit ein GByte/sec [McBr88, Nose&&86].

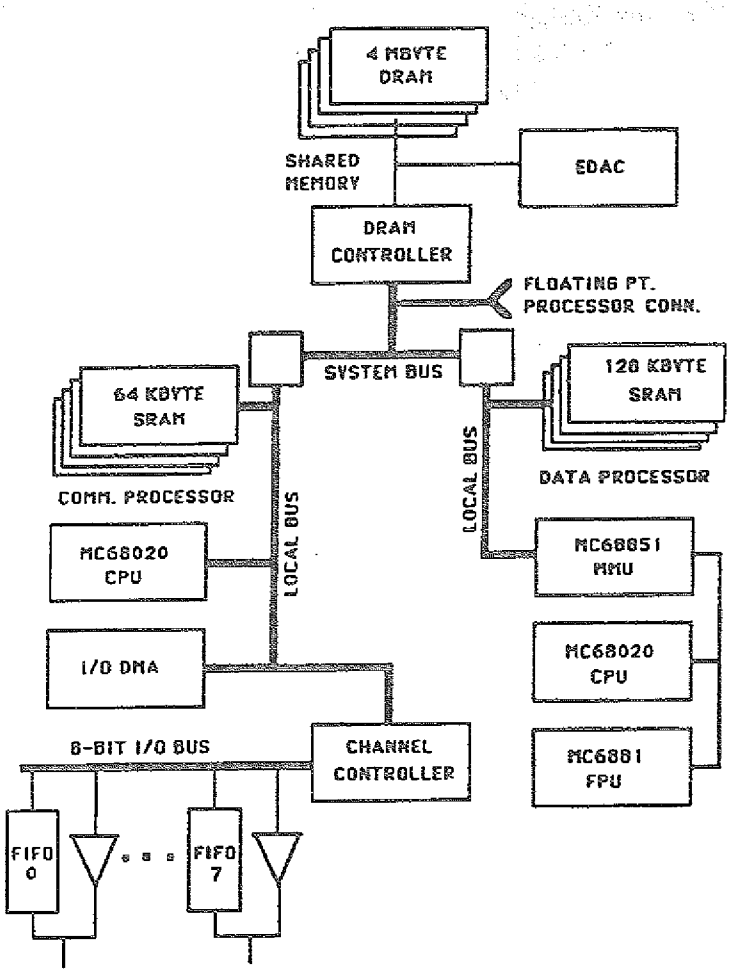


Abb. 4.5: Knotenaufbau eines Mark IIIfp

Laut [Nose&&86] plant CCI Corporation, eine kommerzielle Version dieses Rechners anzubieten.

4.4.5 Sonstige

Weitere Rechner auf der Basis der Hypercube-Struktur sind *CHoPP* (Columbia University) [Alma85], *Gemini* (Paralex Research Inc.) [McBr88], *UNMHC* (University of New Mexico, Los Alamos Laboratory) [Poll&&86, Hong&&85], *BTL-Hypercube* (AT&T Bell Laboratories) [Losz88], der ehemals kommerziell angebotene *System 14* (Ametek Inc.) [Trel88], sowie die auf der Basis von Personal Computern bzw. Apple Macintosh aufgebauten *PC-CUBE* [Ho&&88a] bzw. *MAC-CUBE* [Ho&&88b].

5 Datentransferalgorithmen für Hypercube-Rechner

Aufgrund der verteilten Realisation des Speichers in Multicomputern können Synchronisation und Datentransfer nur durch Interprozessorkommunikation mittels Message-Passing realisiert werden. Die Kommunikationsgeschwindigkeit ist jedoch im Vergleich zur Rechenleistung heutiger Rechner nur gering und hat dementsprechend einen großen Anteil am entstehenden Overhead (Kap. 2.4). Dieser kann, neben einer guten und möglichst gleichmäßigen Lastverteilung, durch eine effiziente Durchführung der Kommunikation entscheidend verringert werden [Lan&&90, Ch&Sh90].
Verglichen mit anderen Multicomputern verfügen Hypercube-Rechner aufgrund der leistungsfähigen Verbindungsstruktur und der daraus resultierenden großen Gesamtbandbreite (Kap. 3.1.8) über gute Vorraussetzungen zum schnellen Datentransfer für viele verschiedene Kommunikationsarten [Bh&Ip85].
Die Minimierung des Verkehrsaufkommens (*traffic*) — gemessen in der Anzahl der insgesamt versendeten Pakete — und des Zeitbedarfs (*time*) sind die wichtigsten Anforderungen an einen Kommunikationsalgorithmus. In Kapitel 5.5 wird anhand des Beispiels 5.5.1 gezeigt, daß Zeitbedarf und Verkehrsaufkommen voneinander abhängig sein können und man sich in diesem Fall für die Minimierung eines Kriteriums entscheiden muß. Hier wird in der Regel die Minimierung des Zeitbedarfs vorgezogen. Dementsprechend sind alle Algorithmen aus den Kapiteln 5.4–5.9 zeitminimiert. Jedoch wird auch eine abgeschwächte Form der Verkehrsminimierung garantiert, denn alle Daten werden genau einmal, jedoch aufgrund von Pipelining oder der Benutzung paralleler Wege meist nicht in einem Paket, auf kürzestem Weg vom Sender zum Empfänger geschickt.
Neben der Vorstellung von verteilten, d.h. ohne globale Kontrolle oder Synchronisation ausführbaren, Datentransferalgorithmen für spezielle Kommunikationsarten wird in Kapitel 5.3 auf einige allgemeine Verfahren (*Routing-Verfahren*) eingegangen, die zur Realisation beliebiger Kommunikationsarten geeignet sind. Da in allgemeinen Verfahren jedoch kein Gebrauch von problemspezifischen Informationen gemacht wird, sondern entweder keine oder nur netzwerkspezifische Parameter herangezogen werden, sind diese Algorithmen im Vergleich zu speziellen Algorithmen weit weniger leistungsfähig.
Weitere, in dieser Arbeit nicht behandelte, teilweise sehr spezielle Kommunikationsarten finden sich in [St&Wa86, St&Wa90, Fo&Fu86, Fo&Fu88a, Fo&Fu88b, Ho&Jo88].

5.1 Grundlegende Definitionen

In diesem Kapitel werden einige zur Beschreibung der Algorithmen benötigte Notationen und graphentheoretische Begriffe aus [Jo&Ho89] eingeführt.
Sei $N = 2^n$ die Anzahl der Knoten eines Hypercube-Rechners, $\mathcal{N} = \{0, 1, \ldots, N-1\}$ die Menge der binären Knotenadressen, $\mathcal{D} = \{0, 1, \ldots, n-1\}$ die Menge der Dimensionen, $\oplus$ bezeichne die bitweise EXKLUSIV-ODER Operation zweier Binärzahlen und $|A|$ sei die Kardinalität einer Menge A. Die Kardinalität der zu übertragenden Datenmenge sei M.
Eine nur aus Nullen (Einsen) bestehende binäre Zahl wird mit 0 (1) abgekürzt. Verbindungen eines Hypercube-Rechners werden entsprechend der Stelle des differierenden Bit der Adressen der durch sie verbundenen Knoten numeriert, d.h. entsprechend der Dimension, in der die Verbindung liegt (Kap. 3.1).

Definition 5.1.1 **(gerichtete Graphen)**

Gerichtete Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ werden durch eine *Knotenmenge* $\mathcal{V}$ und eine *Kantenmenge* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ gerichteter Kanten definiert. Die Länge oder Gewichtung der Kanten in $\mathcal{G}$ sei hier für alle Kanten konstant und gleich eins. Ist $(s,t) \in \mathcal{E}$, so nennt man s *Anfangs-* und t *Endknoten* der Kante (s,t). Knoten, die nicht Anfangsknoten einer Kante sind, heißen *Endknoten*, während Knoten, die keine Endknoten sind, *Anfangsknoten* des Graphen genannt werden. Die Anzahl der Nachbarn eines Knotens wird *Grad* des Knotens genannt, und der *Durchmesser* von $\mathcal{G}$ sei das Maximum aller Weglängen in $\mathcal{G}$.
Es werden im folgenden nur zusammenhängende Graphen betrachtet.

Definition 5.1.2 **(Bäume, spannende Bäume)**

Ein *Baum* ist ein zykelfreier, gerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit genau einem Anfangsknoten (Wurzel, *root*), von der aus es einen eindeutigen Weg zu jedem Knoten des Graphen geben muß. Endknoten eines Baumes heißen *Blätter*, und ein Knoten, die weder Blatt noch Wurzel ist, wird als *Zwischenknoten* bezeichnet. Das Maximum aller Weglängen von der Wurzel zu allen Blättern wird *Höhe* h des Baumes genannt. Die Knoten eines Baumes werden in $h+1$ *Ebenen* eingeteilt. Jede Ebene i mit $0 \leq i \leq h$ enthält genau diejenigen Knoten des Baumes, die den jeweils gleichen Abstand i zur Wurzel haben. Ebene null beispielsweise enthält nur die Wurzel selbst.
Spannende Bäume $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ sind Teilgraphen eines gerichteten Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit einer Knotenmenge $\mathcal{V}' = \mathcal{V}$ und einer Kantenmenge $\mathcal{E}' \subseteq \mathcal{E}$, so daß $\mathcal{G}'$ ein Baum ist. Ein spannender Baum mit der Wurzel $s \in \mathcal{V}$ heißt abstandstreu (*greedy*), falls alle Knoten einer Ebene i, $0 \leq i \leq n$, den gleichen Hamming-Abstand (Kap. 3.1.2) zu s haben.

Definition 5.1.3 **(Hypercube-Graph)**

Ein *Hypercube-Graph* ist ein gerichteter Graph, der aus der Knotenmenge $\mathcal{V} = \mathcal{N}$ und der Kantenmenge $\mathcal{E} = \{(s,t) \in \mathcal{V} \times \mathcal{V} \mid s \ominus t = 2^d, d \in \mathcal{D}\}$ besteht. Gemäß obiger Notation liegen die Kanten (s,t) und (t,s) mit $s \ominus t = 2^d$ in der Dimension d und verbinden die Knoten s und t miteinander über die d-te (bidirektionale) Verbindungsleitung.

Definition 5.1.4 **(Rotation, Translation)**

Unter der *Rotation* $Ro(s)$ eines Knotens s wird eine Rechtsrotation seiner binären Adresse $(s_{n-1}s_{n-2}\ldots s_0)$ verstanden, d.h. $Ro(s) = (s_0 s_{n-1}\ldots s_1)$. Die Rotation eines Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist definiert durch $Ro(\mathcal{G}) = (Ro(\mathcal{V}), Ro(\mathcal{E}))$ mit

$$\begin{aligned} Ro(\mathcal{V}) &= \{Ro(s) \mid s \in \mathcal{V}\} \\ Ro(\mathcal{E}) &= \{(Ro(s), Ro(t)) \mid (s,t) \in \mathcal{E}\}. \end{aligned}$$

Ro^k ist für $k > 1$ als $Ro \circ Ro^{k-1}$ definiert. $Ro^0(s)$ sei s selbst.
Die *Translation* $Tr(s,t)$ eines Knotens t mit einem Knoten s ist definiert durch $Tr(s,t) = s \oplus t$. Unter der Translation eines Graphen $Tr(s,\mathcal{G})$ wird analog zur Rotation der Graph $(Tr(s,\mathcal{V}), Tr(s,\mathcal{E}))$ verstanden mit

$$\begin{aligned} Tr(s,\mathcal{V}) &= \{Tr(s,t) \mid t \in \mathcal{V}\} \\ Tr(s,\mathcal{E}) &= \{(Tr(s,t), Tr(s,u)) \mid (t,u) \in \mathcal{E}\} \end{aligned}$$

Tr^k ist definiert durch $Tr \circ Tr^{k-1}$ für $k > 1$ und es sei wiederum $Tr^0(s) = s$.
Durch Rotation und Translation wird der Hamming-Abstand zweier Knoten nicht verändert. Eine k-fache Rotation Ro^k bildet alle Kanten in der Dimension $d \in \mathcal{D}$ eines Graphen in die Dimension $(d-k) \bmod n$ ab, während eine k-fache Translation Tr^k die Dimension jeder Kante erhält. Jedoch ändert eine Translation die Richtung einer Kante, falls an der m-ten Stelle von s eine eins steht.
Alle sonstigen Eigenschaften eines Graphen, insbesondere die Verbindungsstruktur der Knoten, bleiben bestehen.

Definition 5.1.5 **(Periode, zyklische Knoten)**

Die *Periode* $P(s)$ einer n-stelligen binären Zahl s ist definiert als

$$\min\{i \mid 0 < i \leq n \wedge Ro^i(s) = s\}$$

s heißt *zyklisch*, falls $P(s) < n$. (Beispiel: $s = 0101, P(s) = 2$, d.h. s ist zyklisch)
t heißt *zyklischer Knoten* bezüglich einer Wurzel s, falls $s \oplus t$ zyklisch ist. $s \oplus t$ wird auch *relative Adresse* von t bezüglich s genannt.

5.2 Arten, Strategien und Modelle der Kommunikation

5.2.1 Kommunikationsarten

In den Kapiteln 5.4–5.9 werden Algorithmen für die folgenden häufig benutzten Kommunikationsarten in Parallelrechnern aus [Jo&Ho89, Bert&&89, Lan&&90] vorgestellt:

Unicast: Das Senden von Daten eines Knotens an einen anderen Knoten wird *Unicast* genannt.

Multicast: Unter *Multicast* versteht man das Senden von jeweils gleichen Daten eines Knotens an mehrere, aber nicht alle anderen Knoten [Lan&&90].

Broadcast: *Broadcast* (Verbreitung) heißt das Senden von jeweils gleichen Daten eines Knotens an alle anderen Knoten. Die Datenmenge wird von einem empfangenden Knoten genau so oft vervielfältigt wie er sie an Nachbarknoten weiterleiten muß.

Accumulation: *Accumulation* (Sammlung) bedeutet das Senden von je einer Datenmenge jedes Knotens an einen Knoten unter der Voraussetzung, daß die Datenmengen jederzeit so kombiniert werden können, daß die Transferzeiten der ursprünglichen und der kombinierten Menge gleich bleiben. Accumulation ist die zu Broadcast in dem Sinne inverse Operation, daß das Benutzen der Wege eines Algorithmus für Broadcast (Accumulation) in umgekehrter Richtung zu einem Algorithmus für Accumulation (Broadcast) führt [Bert&&89]. Ein Beispiel für Accumulation ist die Addition von über das gesamte System verteilten Zahlen in einem Knoten. Die Kombination besteht hier in der Addition von jeweils zwei Zahlen zu einem Zwischenergebnis, und die Transferzeiten eines Summanden und eines Zwischenergebnisses sind gleich.

Multibroadcast: Sendet jeder Knoten eine jeweils gleiche Datenmenge an alle anderen Knoten, so spricht man von *Multibroadcast* (complete broadcast) oder von Durchführung eines Broadcast jedes Knotens.

Multiaccumulation: *Multiaccumulation* ist die zu Multibroadcast inverse Kommunikationsart und beinhaltet eine Accumulation jedes Knotens.

Scatter: Mit *Scatter* (personalized communication, distributing) wird das Verschicken von jeweils verschiedenen Datenmengen eines Knotens an alle anderen Knoten bezeichnet. Hier ist keine Vervielfältigung der Mengen sinnvoll.

Gather: Analog zu Accumulation wird *Gather* die zu Scatter inverse Kommunikationsart genannt. Ein Knoten empfängt je eine — mit anderen Mengen nur durch Verdoppelung der Transferzeit zusammenfaßbare — Datenmenge.

Total Exchange: *Total Exchange* (complete exchange) schließlich bezeichnet eine Scatter- oder eine Gather-Operation von jedem Knoten aus, d.h. jeder Knoten erhält je eine Datenmenge von jedem anderen Knoten bzw. schickt je eine Datenmenge an jeden anderen Knoten. Im Unterschied zu Multibroadcast sind alle gesendeten Datenmengen untereinander verschieden.

Kommunikationsart	Anwendung
Multicast	Schaltungssimulation Berechnung mechanischer Modelle Simulation von Rechnernetzen
Broadcast/Accumulation	Gaußsche Elimination mit Pivotisierung Berechnung von inneren Produkten Matrixmultiplikation Berechnung transitiver Hüllen
Scatter/Gather	Lösung tridiagonaler Systeme
Multibroadcast	Iterative Algorithmen
Total Exchange	Matrixtransposition

Tabelle 5.1: Anwendungen von Kommunikationsarten

In Tabelle 5.1 werden Probleme aufgelistet, zu deren Lösung auf Parallelrechnern einige der obigen Kommunikationsarten auf sinnvolle Weise eingesetzt werden können [Bert&&89, Ho&Jo86, Lan&&90].
Neben der oben erwähnten inversen Beziehung bestehen noch weitere Beziehungen zwischen den Kommunikationsarten derart, daß eine Kommunikationsart A eine Kommunikationsart B einschließt, d.h. ein Algorithmus für A führt ohne zusätzlichen Aufwand ebenfalls B durch. Beispielsweise führt ein Algorithmus für Multibroadcast auch Gather durch, denn bei Multibroadcast erhält ein beliebiger Knoten eine Datenmenge von jedem anderen Knoten. Ein Algorithmus für Gather wiederum kann ebenfalls für Accumulation eingesetzt werden. Es ergibt sich insgesamt eine Hierarchie der Kommunikationsarten, die in Abbildung 5.1 für die beschriebenen Arten dargestellt ist. Horizontale Pfeile sind zwischen zueinander inversen Arten eingezeichnet. Alle anderen Pfeile zeigen von einer Kommunikationsart zu einer darin enthaltenen [Bert&&89].

5.2.2 Kommunikationsmodelle

Folgende Annahmen werden bei der Beschreibung der Algorithmen zugrunde gelegt:

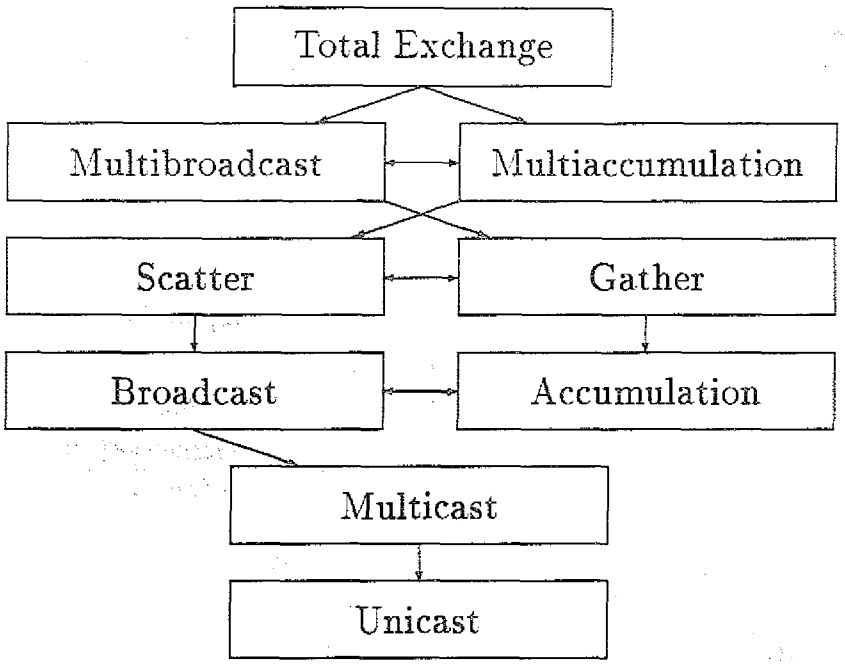


Abb. 5.1: Hierarchie von Kommunikationsarten

1. Die Anzahl der zu übertragenden Daten pro Knoten ist (auch für Scatter, Gather und Total Exchange) konstant.
2. Die Übertragung der Daten ist fehlerfrei.
3. Die Zeit, die zur Manipulation (Vervielfältigung, Kombination) der zu transferierenden Daten von einem Knoten benötigt wird, wird vernachlässigt.
4. Ein Paket kann von einem Knoten erst dann weiterverwendet werden, wenn es vollständig übertragen wurde (*store-and-forward* oder *packet-switched model*).
5. t_{comm} und t_{start} (Kap. 4.2) sind, im wesentlichen unabhängig von der Anzahl der übertragenen Daten und der benutzten (bidirektionalen) Verbindungen, konstant für alle Knoten. In existierenden Hypercube-Rechnern gilt meist $t_{comm} \ll t_{start}$.
6. Die zur Übertragung eines Datenpakets der Größe M von einem Knoten zu einem Nachbarknoten benötigte Zeit beträgt $M \cdot t_{comm} + t_{start}$. Die Übertragung eines Pakets wird *routing cycle* (Routing-Schritt) genannt. Den Zeitanteil, der durch t_{start} hervorgerufen wird, nennt man *Startzeit*, während der Term mit t_{comm} die (asymptotische) Kommunikationsgeschwindigkeit, im folgenden *reine Übertragungszeit* genannt, bezeichnet.

7. Entweder können alle Knoten zu einem Zeitpunkt auf je einer Verbindung Daten empfangen und senden, wobei die sendende und die empfangende Verbindung nicht identisch sein müssen (*one-port-* oder *processor-bound communication*, Ein-Kanal Kommunikation), oder alle Knoten können auf allen Verbindungen gleichzeitig senden und empfangen (*n-port-* oder *link-bound communication*, n-Kanal Kommunikation).

Die ersten vier Annahmen sind in allen Arbeiten zum Thema Datentransferalgorithmen zu finden, während in den Punkten fünf bis sieben Unterschiede festzustellen sind. Die hier aufgeführten Annahmen finden sich z.B. in [Sa&Sc89a, Jo&Ho89, St&Wa90] und sind sinnvoll, denn die Werte für t_{comm} und t_{start} können für spezielle Hypercube-Rechner ermittelt werden (Kap. 4).
Ähnliche Annahmen wie oben werden in [Bert&&89, Lan&&90] benutzt. Jedoch wird hier nicht nur die insgesamt zu übertragende Datenmenge, sondern auch die Größen der einzeln gesendeten Pakete während des gesamten Algorithmus als konstant angenommen, und dementsprechend sind alle Transferzeiten pro Paket konstant, d.h. es kann angenommen werden, daß t_{comm} konstant und t_{start} gleich null ist. Zeitminimierung bedeutet in diesem Fall die Minimierung der für die Übertragung benötigten Routing-Schritte. Dies heißt, daß aufgrund der konstanten Paketgröße keine der im nächsten Kapitel vorgestellten Strategien, Pipelining oder vielfache Wege benutzt werden.
In [Frai89, Frai90] hingegen wird angenommen, daß die Startzeit t_{start} von der Zahl der benutzten Verbindungen abhängig ist. Diese kann zwischen eins und der maximal möglichen Zahl schwanken.

5.2.3 Kommunikationsstrategien

Es existieren einige Standardtechniken, deren Verwendung in Datentransferalgorithmen zu einer teilweise erheblichen Verringerung der Gesamttransferzeit bzw. des Verkehrsaufkommens führt. Dazu gehören nach [St&Wa86]:

Pipelining: Die Übertragung eines Datenpakets der Länge M über $n-1$ Zwischenknoten benötigt bei n Routing-Schritten mit jeweiliger Übertragung aller M Daten in einem Paket die Zeit

$$n \cdot (M \cdot t_{comm} + t_{start}).$$

Unterteilt man das Paket jedoch in $\lceil \frac{M}{B} \rceil$ kleinere Pakete etwa gleicher Länge B und schickt in je einem Routing-Schritt ein Paket auf den Weg zum Zielknoten, so wird die Gesamtbandbreite (Kap. 3.1.8) des Hypercube-Rechners durch Benutzung mehrerer Verbindungen pro Routing-Schritt besser genutzt und die Transferzeit T verringert sich zu

$$T = \left(\left\lceil \frac{M}{B} \right\rceil + n - 1 \right) \cdot (B \cdot t_{comm} + t_{start})$$

$$= (M + (n-1) \cdot B) \cdot t_{comm} + \left(\left\lceil \frac{M}{B} \right\rceil + n - 1 \right) \cdot t_{start},$$

denn $\lceil \frac{M}{B} \rceil$ Pakete werden nacheinander verschickt, und das letzte Paket muß noch $(n-1)$ Schritte bis zum Ziel zurücklegen.

Optimiert man die Paketgröße bezüglich minimaler Transferzeit durch Nullsetzen der ersten Ableitung nach B, so erhält man als optimale Paketgröße

$$B_{opt} = \sqrt{\frac{M \cdot t_{start}}{(n-1) \cdot t_{comm}}}$$

mit der minimalen Transferzeit

$$\begin{aligned} T_{min} &= M \cdot t_{comm} + (n-1) \cdot t_{start} + 2 \cdot \sqrt{M \cdot (n-1) \cdot t_{comm} \cdot t_{start}} \\ &= \left(\sqrt{M \cdot t_{comm}} + \sqrt{(n-1) \cdot t_{start}} \right)^2 . \end{aligned}$$

Jedoch wird das Verkehrsaufkommen durch Pipelining von ursprünglich n auf $B \cdot n$ versendete Pakete gesteigert.

Es sei darauf hingewiesen, daß die hier erstmals auftretenden Brüche in Gaußklammern im folgenden genauso wie Brüche in herkömmlichen Klammern behandelt werden. Insbesondere werden Terme der Form $\lceil \frac{M}{B} \rceil \cdot B$ zu M vereinfacht. Falls der Bruch $\frac{M}{B}$ keine ganze Zahl ergibt, kann der durch diese Vereinfachung entstehende Fehler für kleine Werte von M und B zwar groß werden. Für größere Werte kann man ihn jedoch vernachläßigen. Des weiteren würde eine exakte Behandlung aller Gaußklammern einen großen Aufwand durch Fallunterscheidungen erfordern und nur einen unwesentlichen Beitrag zum Verständnis der Algorithmen leisten.

vielfache Wege: Das Aufspalten eines Datenpakets in mehrere, möglichst gleich große Teilpakete und deren gleichzeitiger Transfer über kantendisjunkte Wege gleicher Länge kann im Fall der n-Kanal Kommunikation zu einer um den Faktor n schnelleren Transferzeit führen und nutzt die Bandbreite des Hypercube-Rechners besser aus als die Übertragung über nur einen Weg.

Aber auch hier wird der schnellere Transfer nur durch ein um den Faktor n erhöhtes Verkehrsaufkommen erreicht.

dynamische Paketgröße: Pakete können nicht nur in der Quelle und im Empfänger, sondern auch in Zwischenknoten zusammengefaßt oder aufgespalten werden. Ein Beispiel hierfür ist das Aneinanderhängen zweier Pakete aus verschiedenen Knoten, die dasselbe Ziel haben. Dadurch wird ein Routing-Schritt, d.h. die Zeit t_{start} gespart.

Umgekehrt können Pakete mit einem anfänglich gemeinsamen Wegstück, die jedoch für verschiedene Empfänger bestimmt sind, in einem Paket zusammengefaßt und erst am Ende des gemeinsamen Weges wieder aufgespalten werden.

Durch diese Strategie wird nicht nur die benötigte Zeit, sondern auch das Verkehrsaufkommen um die um eins verringerte Anzahl der zusammengefaßten Pakete reduziert.

Kommunikationsalgorithmen werden im folgenden durch die Angabe aller Pfade von Sendern zu Empfängern (*Kommunikationsstruktur*) und einer oder mehrerer der obigen Strategien definiert. Im Fall der Ein-Kanal Kommunikation muß zusätzlich für alle Quellen angegeben werden, in welcher Reihenfolge mit den jeweiligen Nachbarn kommuniziert werden soll. Die Menge der Pfade wird im Fall von Broadcast, Multibroadcast, Scatter und Total Exchange durch einen spannenden Baum und im Fall von Multicast durch einen Baum angegeben. Falls mehrere Sender auftreten (Multibroadcast, Total Exchange), wird jeder Sender als Wurzel eines entsprechenden Baumes angenommen..
Nicht explizit behandelte Algorithmen für Kommunikationsarten aus Kapitel 5.2.1 (Gather, Accumulation, Multiaccumulation) ergeben sich durch einfache Umkehrung aller Pfade der Kommunikationsstruktur von Algorithmen für das jeweils inverse Problem.

5.3 Routing-Verfahren

Neben der in Kapitel 3.1.2 beschriebenen Möglichkeit, Nachrichten zwischen zwei beliebigen Knoten unabhängig von der Belastung des Netzwerks auf immer gleichen Wegen zu übertragen (*fixed-path routing*), verspricht besonders in Parallelrechnern mit leistungsfähiger Verbindungsstruktur die Benutzung des flexiblen *adaptiven Routings* schnelle Interprozessorkommunikation. Bei dieser Art des Routings kann nach verschiedenen Kriterien, z.B. in Abhängigkeit vom Status des Netzwerks, von der Länge des noch zurückzulegenden Weges oder von der Länge des gesamten Weges gewählt werden, an welchen Nachbarknoten eine Nachricht weitergeleitet werden soll [Ragh&&88]. Bei *verteiltem* adaptiven Routing trifft jeder Knoten seine Entscheidung nur aufgrund eigener lokaler Informationen. Der Status des restlichen Netzwerks bleibt unberücksichtigt. Im Gegensatz dazu werden in *zentralisierten* adaptiven Routing-Verfahren globale Informationen in einem oder auch mehreren Knoten gesammelt und aufgrund dieser Informationen ein Weg gesucht. Durch das Sammeln von Informationen kann zwar eine genauere Einschätzung der tatsächlichen Belastung des Netzwerks erfolgen, jedoch muß ein großer zusätzlicher Overhead in Kauf genommen werden.

Die Anzahl von Nachbarn, zu denen jeder Knoten eines Hypercube-Rechners eine Nachricht weiterleiten kann, ist gleich dem Hamming-Abstand des Knotens vom Ziel der Nachricht, falls die Nachricht auf dem kürzesten Weg weitergeleitet werden soll. Beispielsweise kann die Nachricht über diejenige Verbindung übertragen werden, deren Warteschlange mit noch zu übertragenden Nachrichten über die wenigsten Einträge verfügt (*shortest queue routing*). Weitere adaptive Routing-Verfahren aus [Ki&Re88] sind das *NRCC routing* (ein zentralisiertes Verfahren, welches Informationen über das Netzwerk in einem Knoten sammelt und in bestimmten Zeitintervallen Tabellen mit den besten Pfaden an alle anderen Knoten weiterleitet), das *delta routing* (eine Mischung aus NRCC und shortest queue routing, bei der für jedes Ziel verschiedene Wege vorgeschlagen werden und ein Knoten einen dieser Wege nur aufgrund lokaler Informationen auswählt) und das *hybrid weighted routing* (ähnlich wie delta routing, nur entscheidet jeder Knoten mit Hilfe von lokalen und globalen Informationen, welcher Weg eingeschlagen wird).

Ein Routing-Verfahren, welches zwar auf jeglichen Gebrauch von Netzwerkinformationen verzichtet, aber Nachrichten dennoch nicht auf festen Pfaden überträgt, wird in [Vali82, Va&Br81, Vali80] vorgeschlagen. Die Idee dieses Routing-Verfahrens ist es, zufällig einen der für eine Übermittlung in Frage kommenden Nachbarn auszusuchen. Es ist besonders dafür geeignet, die Kommunikation von einem Knoten zu einem anderen Knoten gleichzeitig für alle Knoten eines Hypercube-Rechners zu realisieren, falls ein ungleichmäßiges und nicht vorhersehbares Kommunikationsaufkommen zu erwarten ist. Es kann aber auch jede beliebige andere Kommunikationsart unterstützt werden.

Das Verfahren besteht aus zwei Phasen. In der ersten Phase wird für jedes zu übertragende

Paket und für jede Dimension der Verbindungsstruktur des Hypercube-Rechners zufällig entschieden, ob das Paket über eine Verbindung in dieser Dimension übertragen werden soll oder nicht. Nach der Übertragung über die gewählten Dimensionen sind die Pakete mit großer Wahrscheinlichkeit gleichmäßig über das System verteilt, und in der zweiten Phase wird jedes Paket durch adaptives Routing auf zufällig ausgesuchten, aber kürzesten Wegen zu seinem Ziel geschickt.
Dieses Verfahren kann verteilt implementiert werden, wenn in der ersten Phase die Menge der gewählten Dimensionen pro Paket mit übertragen wird.
In [Vali82] wird gezeigt, daß dieses Verfahren für jede Kommunikationsart mit großer Wahrscheinlichkeit nach $O(\log n)$ Kommunikationsschritten abgeschlossen ist.

5.4 Unicast

Zur Angabe eines Algorithmus speziell für den Datentransfer zwischen zwei Knoten wird für Ein-Kanal Kommunikation das Pipelining-Konzept und im Fall von n-Kanal Kommunikation zusätzlich das Konzept der vielfachen Wege benutzt (Kap. 5.2.3) [Sa&Sc89a].
Nach Kapitel 3.1.5 existieren genau n knotendisjunkte (parallele), also auch kantendisjunkte Wege zwischen zwei Knoten. Ist j der Hamming-Abstand beider Knoten und $j < n$, so haben j der parallelen Wege die Länge j und die übrigen $n - j$ Wege die Länge $j + 2$. Falls $j = n$ ist, existieren nur Wege der Länge n [Sa&Sc85a].
Sei i das Maximum der Längen der obigen n parallelen Wege, d.h.

$$i = \begin{cases} j & \text{,falls } j = n \\ j+2 & \text{,falls } j < n \end{cases}$$

5.4.1 Ein-Kanal Unicast

Eine untere Schranke für die durch Unicast benötigte Zeit mit Ein-Kanal Kommunikation ist

$$(M + j - 1) \cdot t_{comm} + i \cdot t_{start},$$

denn der sendende Knoten muß M Daten verschicken, und das letzte Element muß danach noch einen Weg der Länge $j - 1$ zurücklegen. i ist die minimale Anzahl der Startzeiten, d.h. die minimale Anzahl weiterleitender Knoten.
Da jeder Knoten nur auf einem Kanal lesen und auf einem weiteren schreiben kann, wird zum Transfer eines Pakets der Länge M unter Benutzung von Pipelining nach Kapitel 5.2.3 die Zeit

$$\begin{aligned} T &= \left(\left\lceil \frac{M}{B} \right\rceil + i - 1\right) \cdot (B \cdot t_{comm} + t_{start}) \\ &= (M + (i-1) \cdot B) \cdot t_{comm} + \left(\left\lceil \frac{M}{B} \right\rceil + i - 1\right) \cdot t_{start} \end{aligned}$$

benötigt, falls das Paket in $\lceil \frac{M}{B} \rceil$ etwa gleich große Pakete der Größe B aufgespalten wird und diese über einen der i Wege der Länge i nacheinander übertragen werden. Die optimale Paketgröße beträgt

$$B_{opt} = \sqrt{\frac{M \cdot t_{start}}{(i-1) \cdot t_{comm}}},$$

und man erhält in diesem Fall die minimale Transferzeit

$$T_{min} = \left(\sqrt{M \cdot t_{comm}} + \sqrt{(i-1) \cdot t_{start}}\right)^2.$$

5.4.2 n-Kanal Unicast

Eine untere Schranke für den Zeitbedarf von Unicast mit n-Kanal Kommunikation ist

$$\left(\left\lceil \frac{M}{n} \right\rceil + j - 1\right) \cdot t_{comm} + i \cdot t_{start},$$

denn die Quelle braucht nun nur noch $\lceil \frac{M}{n} \rceil \cdot t_{comm}$ Zeit zum Senden der Daten bei wiederum j Startzeiten.
Ein zu transferierendes Paket der Länge M wird zunächst in n Pakete der Länge $\lceil \frac{M}{n} \rceil$ geteilt. Jedes dieser n Pakete wird nun auch in Pakete der Größe B unterteilt. Die Anzahl der Pakete beträgt nun $\lceil \frac{M}{n \cdot B} \rceil$. Jeweils n Pakete werden schließlich gleichzeitig mittels Pipelining über je einen der n kantendisjunkten Wege zum Ziel geschickt. Dafür benötigt man bei einer optimalen Paketgröße von

$$B_{opt} = \sqrt{\frac{M \cdot t_{start}}{n \cdot (i-1) \cdot t_{comm}}}$$

die Zeit

$$T_{min} = \left(\sqrt{\frac{M}{n} \cdot t_{comm}} + \sqrt{(i-1) \cdot t_{start}}\right)^2.$$

Sinnvoll wird die Aufteilung in n Pakete natürlich erst ab $M \geq n$. Dann jedoch verringert sich die reine Übertragungszeit um dem Faktor n gegenüber der Ein-Kanal Kommunikation. Abbildung 5.2 veranschaulicht den Datentransfer zwischen den Knoten 0000 und 1100 eines vierdimensionalen Hypercube-Rechners. Der Hamming-Abstand beider Knoten beträgt zwei. Dementsprechend existieren zwischen ihnen zwei Wege der Länge zwei und zwei Wege der Länge vier. Letztere Wege sind nicht eindeutig bestimmt (Kap. 3.1.5) und können z.B. durch ein adaptives Routing-Verfahren bestimmt werden.

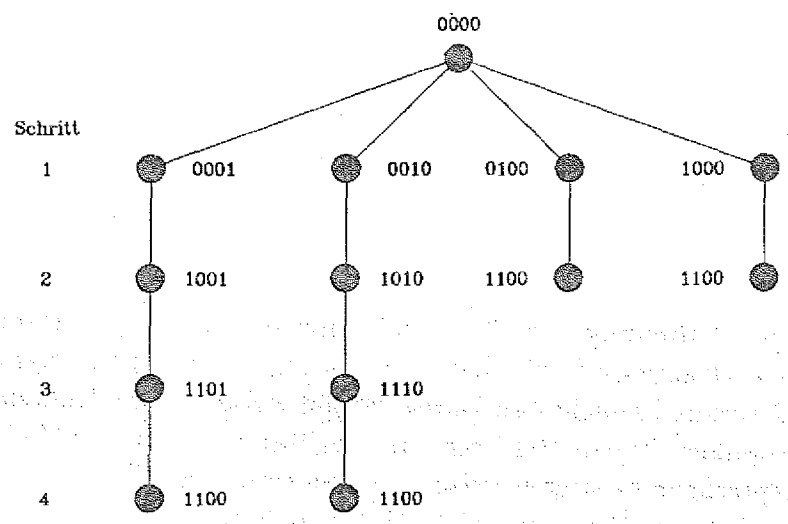


Abb. 5.2: Parallele Wege von 0000 nach 1100

5.5 Multicast

Seien s die Adresse des Senders und $d_1, \ldots, d_k$ die Adressen der Empfänger einer Nachricht in einem n-dimensionalen Hypercube-Rechner. Die Kommunikationsstruktur des Multicast-Problems läßt sich unter Voraussetzung von n-Kanal Kommunikation nach [Lan&&90] durch die Angabe eines Teilbaumes (*Multicast-Baum*) $\mathcal{T} = (\mathcal{V}_T, \mathcal{E}_T)$ eines Hypercube-Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit $\mathcal{N} = 2^n$ (Def. 5.1.3, Seite 63) darstellen. $\mathcal{T}$ muß folgende Eigenschaften besitzen:

1. $\{s, d_1, \ldots, d_k\} \subseteq \mathcal{V}_T$
2. Die Länge eines kürzesten Weges von s nach d_i in $\mathcal{G}$ ist gleich der Länge des Weges von s nach d_i in $\mathcal{T}$ für $1 \leq i \leq k$.

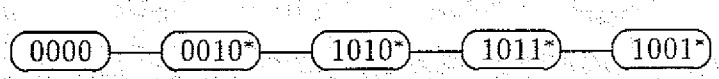
Die zweite Bedingung impliziert, daß der Baum abstandstreu ist, und garantiert die Minimierung der Anzahl der benötigten Routing-Schritte und damit auch der benötigten Zeit, da für deren Messung in [Lan&&90] nur die Anzahl der Routing-Schritte herangezogen wird (Kap. 5.2.2). Aufgrund dessen wird hier weder das Konzept des Pipelining noch die Strategie der vielfachen Wege angewendet.
Ein Baum, der zusätzlich der Bedingung

3. $|\mathcal{E}_T|$ so klein wie möglich

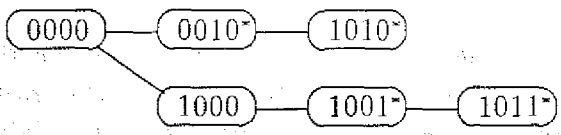
genügt, wird OMT (*Optimal Multicast Tree*) genannt. Ein OMT ist damit derjenige Multicast-Baum, der als Kommunikationsstruktur für Multicast minimales Verkehrsaufkommen erzeugt. Dies wird, ähnlich wie bein Konzept der dynamischen Paketgröße (Kap. 5.2.3) dadurch erreicht, daß die zu übertragenden Daten erst dann vervielfältigt und über verschiedene Wege weitergeleitet werden, wenn ein gemeinsames Wegstück endet.
Daraus folgt jedoch nicht immer, daß das Verkehrsaufkommen eines OMT optimal ist. Dies zeigt folgendes Beispiel:

Beispiel 5.5.1 : $s = 0000$, $(d_1, d_2, d_3, d_4) = (0010, 1001, 1010, 1011)$ (mit * markiert)

Bei vier Empfängern wird ein Verkehr von mindestens vier erzeugt. Dies kann durch einen Baum, bestehend aus dem Weg

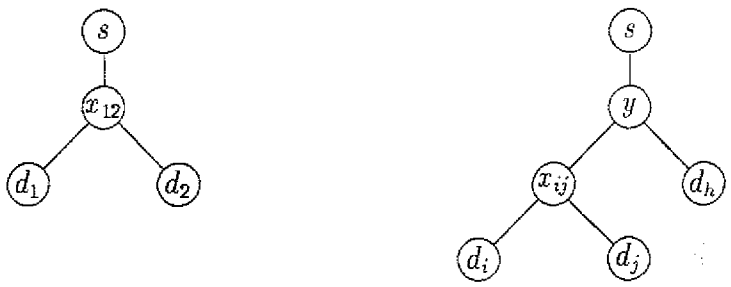


realisiert werden, der aber auch vier Routing-Schritte benötigt. Der OMT mit den zwei Wegen



hingegen hat ein Verkehrsaufkommen von fünf, benötigt aber nur drei Routing-Schritte.

Als problematisch stellt sich bei gegebenem Sender s und Empfängern $d_1, \ldots, d_k$ mit $d_i \neq d_j$ für $i \neq j$ und $s \neq d_i$ $(1 \leq i, j \leq k)$ das Finden eines OMT heraus, denn die Autoren von [Lan&&90] vermuten, daß dieses Problem in der Klasse der *NP-schweren* Probleme liegt. Die Klasse *NP* ist durch alle Probleme charakterisiert, die durch nichtdeterministische Algorithmen mit polynomialem Zeitaufwand gelöst werden können. Ist ein Problem *NP*-schwer, so liegt es in *NP*, und falls ein deterministischer polynomieller Algorithmus zur Lösung des Problems existiert, so existieren Algorithmen gleicher Komplexität für jedes Problem in *NP* [Ho&Ul90].
Daher gibt es vermutlich keinen effizienten Algorithmus zum Finden eines OMT. Statt dessen wird in [Lan&&90] eine Heuristik (*greedy multicast algorithm*, Multicast-Heuristik) angegeben, die einen Multicast-Baum erzeugt. Simulationen zeigen, daß die durch die Multicast-Heuristik gefundenen Multicast-Bäume bezüglich Verkehrsaufkommen sehr nahe an der optimalen Lösung liegen.
Die Schwierigkeiten beim Finden eines OMT liegen darin begründet, daß es aufgrund der verschiedenen Möglichkeiten für die Anzahl und die Lage der Empfänger keine einheitliche Repräsentation der Kommunikationsstruktur durch einen Baum gibt. Lediglich für eine konstante Anzahl von Empfängern $k \in \{1, 2, 3\}$ gibt es einheitliche Muster. Schon für $k = 4$ existieren zwei, für $k = 5$ drei und für $k = 6$ sechs Strukturen, die nicht aufeinander abgebildet werden können.
Abbildung 5.3 zeigt die Kommunikationsstrukturen für Multicast mit zwei und drei Empfängern. Jeder Zwischenknoten ist der eindeutig bestimmte, nächste gemeinsame Vorgänger (*nearest common ancestor*, NCA) seiner beiden Nachfolger. Der NCA zweier Knoten t und u $NCA(t, u)$ bezüglich einer Wurzel s ist als derjenige Knoten mit maximalem Abstand zur Wurzel definiert, der im Hypercube-Graphen $\mathcal{G}$ auf kürzesten Wegen von t und u zur Wurzel liegt. Gegebenenfalls kann der NCA auch t oder u selbst sein. Der NCA kann auf einfache Weise durch eine logische UND-Verknüpfung ($\wedge$) der binären Adressen von t und u berechnet werden, falls $s = 0000$. Ist $s \neq 0000$, so muß für beide Knotenadressen vor der UND-Verknüpfung und mit diesem Ergebnis eine EXKLUSIV-ODER-Verknüpfung ($\oplus$) mit der Adresse von s durchgeführt werden. Beispielsweise gilt



(b) (a)

Abb. 5.3: Allgemeines Kommunikationsmuster für zwei (a) und drei (b) Empfänger

für einen vierdimensionalen Hypercube-Rechner und die Wurzel $s = 0001$:

$$\begin{aligned} NCA(0110, 1100) &= NCA(0110 \ominus s, 1100 \ominus s) \oplus s \\ &= (0111 \wedge 1101) \oplus s \\ &= 0100 \\ NCA(0110, 1001) &= 0001 = s \\ NCA(0111, 1110) &= 0111 = t \end{aligned}$$

Für Abbildung 5.3 (a) können somit je nach Adresse von s, d_1 und d_2 folgende Fälle auftreten, die alle durch den abgebildeten Baum repräsentiert werden.

1. $x_{12} = d_1$
2. $x_{12} = d_2$
3. $x_{12} = s$

Die Kommunikationsstruktur eines Algorithmus für Multicast mit zwei Empfängern kann somit einfach aufgebaut werden durch:

$x_{12} := NCA(d_1, d_2)$
Bilde einen Pfad von s nach x_{12} und sende darüber die Daten
Bilde einen Pfad von x_{12} nach d_1 und sende darüber die Daten
Bilde einen Pfad von x_{12} nach d_2 und sende darüber die Daten

Auch für $k = 3$ existiert ein Baum, der alle dort auftretenden Fälle abdeckt (Abb. 5.3 (b)), und ein Algorithmus für diesen Fall kann auf ähnlich einfache Weise formuliert werden. Ab $k = 4$ jedoch existieren mehrere solcher Bäume (Abb. 5.4), und es muß durch aufwendige Suche der für ein gegebenes Problem passende Baum gesucht werden. Folglich kann nur

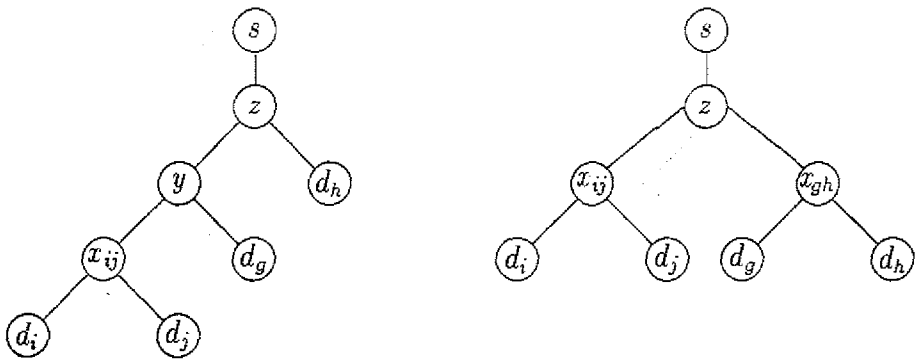


Abb. 5.4: **Allgemeine Kommunikationsmuster für vier Empfänger**

ein Multicast mit $k \leq 3$ Empfängern durch einen effizienten Algorithmus und eine — auf einem OMT basierende — Kommunikationsstruktur realisiert werden.

Die Multicast-Heuristik

Die Multicast-Heuristik wird von jedem Zwischenknoten nach Erhalt eines Datenpakets, bestehend aus den Daten selbst und einer Liste von Empfängern, ausgeführt und beginnt mit der Abarbeitung im Sender s. Die Heuristik sucht nacheinander durch Bestimmung der Wurzel denjenigen Teilbaum aus, in dem die meisten Empfänger liegen und schickt die Daten und die Liste der in diesem Teilbaum liegenden Empfänger an dessen Wurzel. Somit ist ein empfangender Knoten entweder selbst Empfänger der Nachricht oder Zwischenknoten für alle Empfänger in seiner Empfängerliste.
Die Multicast-Heuristik lautet:

1. Vergleiche die eigene Adresse mit den Adressen in der Empfängerliste. Falls Gleichheit mit einem Eintrag besteht, übernehme die Nachricht und lösche den verglichenen Eintrag aus der Liste.

2. Beende den Algorithmus, falls die Empfängerliste leer ist. Andernfalls lege ein Wertungsfeld der Länge n an und initialisiere jeden Eintrag mit null. Durchlaufe die Liste der Empfänger und addiere eins zum i-ten Feldeintrag, falls eine eins an der i-ten Stelle der relativen Adresse von Empfänger und eigener Adresse steht. Dies bedeutet, daß die eigene Adresse und die des betrachteten Empfängers an der Stelle i differieren, d.h. die Dimension i muß auf dem Weg zum Empfänger noch durchlaufen werden.

3. Schicke die empfangene Nachricht über die Verbindung in derjenigen Dimension j, deren Eintrag im Wertungsfeld maximal ist, d.h. die Anzahl der Nachrichten, die über

diese Dimension noch übertragen werden müssen, ist maximal für alle Dimensionen. Schicke mit der Nachricht jeweils eine Teilliste der Empfängerliste, bestehend aus denjenigen Empfängern, deren relative Adressen bezüglich eigener Adresse an der Stelle j eine eins besitzen, d.h. alle Empfänger, die auf kürzestem Weg über die Dimension j erreicht werden können. Falls mehrere Dimensionen mit maximalem Wertungsfeldeintrag existieren, wähle eine beliebige Dimension aus.
Lösche die Teilliste aus der eigenen Empfängerliste und fahre mit Punkt 2 fort.

Zur Veranschaulichung wird obiger Algorithmus für das Senden einer Nachricht in einem vierdimensionalen Hypercube-Rechner von Knoten 1001 zu den Knoten 0111, 1101, 1010 und 1110 durchgeführt. In Tabelle 5.2 sind die wichtigsten Schritte des Algorithmus eingetragen. In den Spalten stehen nacheinander die ausführenden Knoten, deren Empfängerliste, die relativen Adressen der Empfänger, das Wertungsfeld, der Index des Wertungsfelds mit maximalem Eintrag (d.h. die Dimension, über die zuerst übertragen wird) und schließlich die mitzusendende Teilliste der Empfängerliste. Mit † gekennzeichnete Dimensionen wurden zufällig gewählt. Die Ausführung des Algorithmus durch die Knoten, die Blätter des Baumes sind, wurde nicht in die Tabelle eingetragen, denn deren Aktion beschränkt sich auf die Überprüfung der (einelementigen) Empfängerliste, bestehend aus der eigenen Adresse, und damit der Beendigung des Algorithmus.

Knoten	Empfängerliste	relative Adressen der Empfänger	Feld	Dim.	zu sendende Teilliste
1001	0111,1101,1010,1110	1110,0100,0011,0111	1,3,3,2	1†	0111,1010,1110
	1101	0100	0,1,0,0	2	1101
	∅				
1011	0111,1010,1110	1100,0001,0101	1,2,0,2	0†	1010,1110
	0111	1100	1,1,0,0	3†	0111
	∅				
0011	0111	0100	0,1,0,0	2	0111
	∅				
1010	1010,1110				
	1110	0100	0,1,0,0	2	1110
	∅				

Tabelle 5.2: Beispiel eines Ablaufs der Multicast-Heuristik

Die Bäume in Abbildung 5.5 sind die von deren Wurzeln erzeugten Teilbäume des gesamten Multicast-Baumes in Abbildung 5.6. Empfängerknoten sind mit * gekennzeichnet.

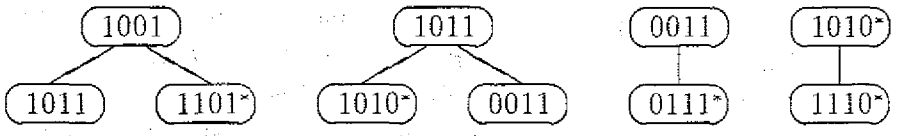


Abb. 5.5: Von Zwischenknoten erzeugte Teilbäume des Multicast-Baumes

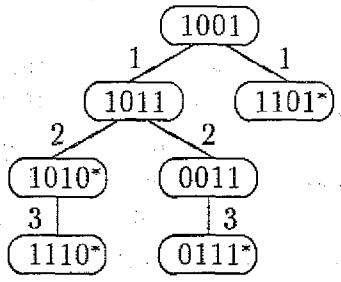


Abb. 5.6: Gesamter Multicast-Baum

Die Ziffern in Abbildung 5.6 geben den Routing-Schritt an, in dem eine Übertragung auf der gekennzeichneten Verbindung stattfindet unter Voraussetzung von n-Kanal Kommunikation. Da die Höhe des Baumes gleich dem Maximum der Hamming-Abstände von der Wurzel zu den Empfängern ist, gibt dieses Maximum die Anzahl der notwendigen Routing-Schritte an. Dies gilt jedoch nur bei der (vorausgesetzten) Vernachlässigung der für die Berechnung der Nachfolger benötigten Zeit. Eine obere Schranke für die Anzahl der benötigten Routing-Schritte ist n.
Der Multicast-Baum aus Abbildung 5.6 ist ein Spezialfall des linken Baumes aus Abbildung 5.4 mit

- $s = 1001, d_i = 1110, d_j = 1010, d_g = 0111, d_h = 1101$
- $x_{ij} = d_j$, denn $x_1 = NCA(d_i, d_j) = 1010$
- $y = 1011$, denn $y = NCA(x_{ij}, d_g) = 1011$
- $z = 1001 = s$, denn $z = NCA(y, d_h) = 1001$.

5.6 Broadcast

Erste einfache Algorithmen für Broadcast in Hypercube-Rechnern werden in [Su&Ba77] angegeben. Die darin zur Anwendung gekommene Idee des Sendens über einen spannenden Binomialbaum wird ebenfalls in späteren Arbeiten [Ho&Jo86, Jo&Ho89] verwendet und dient teilweise als Grundlage für die in den folgenden Kapiteln eingeführten komplexeren Kommunikationsstrukturen. Diese bestehen entweder aus einem (für Broadcast und Scatter) oder aus der Komposition mehrerer Graphen gleichen Typs (für Multibroadcast und Total Exchange), wobei jeder Sender die Position der Wurzel bzw. Anfangsknoten eines Graphen annimmt.
Explizite Beweise der Eigenschaften der Bäume werden der Übersicht halber nicht angegeben, können jedoch ausnahmslos in den angegebenen Quellen nachgeschlagen werden.

5.6.1 Spannender Binomialbaum

Ein spannender Binomialbaum (*spanning binomial tree*, SBT) [Ho&Jo86, Jo&Ho89, Frai90] ist ein Baum der Höhe $n \geq 0$, dessen Name auf die Eigenschaften zurückzuführen ist, daß er als spannender Baum in eine n-dimensionale Hypercube-Struktur eingebettet werden kann und die Knotenzahl auf jeder Ebene i, $0 \leq i \leq n$, $\binom{n}{i}$ beträgt. Allgemein erfolgt die Definition eines SBT rekursiv (Abb. 5.7) durch:

1. Ein SBT der Höhe null besteht aus einem Knoten.
2. Ein SBT der Höhe $n+1$, $n \geq 0$, entsteht aus zwei SBT der Höhe n durch Einfügen einer Kante von der Wurzel des ersten zur Wurzel des zweiten Baumes. Als neue Wurzel wird die als Ausgangspunkt der eingefügten Kante gewählte alte Wurzel genommen.

Ein SBT der Höhe n besteht demnach aus n, zur Wurzel adjazenten Teilbäumen, die die Eigenschaft besitzen, selbst spannende Binomialbäume zu sein, die jeweils die Höhe $0, 1, \ldots, n-1$ besitzen.
Die Einbettung eines SBT in einen Hypercube-Rechner kann verteilt erfolgen, denn jeder Knoten kann unter Verwendung der eigenen Adresse und der Adresse der Wurzel sowohl alle Nachfolger als auch seinen Vorgänger bestimmen. Letztere Möglichkeit ist für die Durchführung der, hier nicht explizit vorgestellten, inversen Kommunikationsarten von Wichtigkeit. Diese können meist einfach durch Benutzung aller Wege in umgekehrter Richtung durchgeführt werden.
Da die auszuführende Kommunikationsart jedem Knoten bekannt sein muß, kann davon ausgegangen werden, daß auch die Wurzeladresse global bekannt ist. Falls dies jedoch nicht der Fall ist, muß sie in jedes zu verschickende Datenpaket integriert werden.
Für den spannenden Binomialbaum eines Hypercube-Graphen ergibt sich die folgende formale Definition:

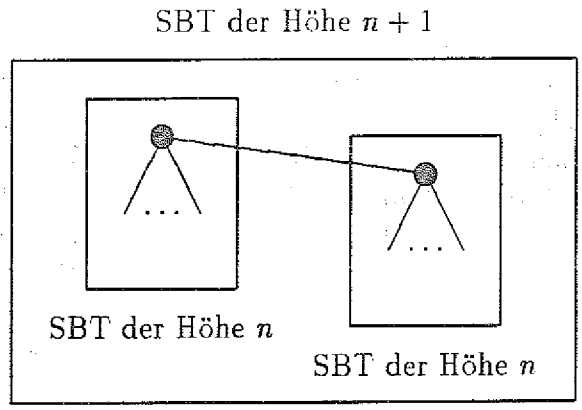


Abb. 5.7: Rekursiver Aufbau eines SBT

Definition 5.6.1 : **(Spannender Binomialbaum, SBT)**

Ein SBT der Höhe n mit der Wurzel $s = (s_{n-1} \ldots s_1 s_0)$ in einem n-dimensionalen Hypercube-Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist gegeben durch die Nachfolgermenge $children_{\text{SBT}}$ und die einelementige Vorgängermenge $parent_{\text{SBT}}$ für alle Knoten $t = (t_{n-1} \ldots t_1 t_0) \in \mathcal{V}$.
$\mathcal{M}_{\text{SBT}}$ ist als Menge der Stellenzahlen (Indizes) der führenden binären Ziffern von $c = t \oplus s = (c_{n-1} \ldots c_1 c_0)$ mit Wert null und p als Stellenzahl der höchstwertigen Ziffer von c mit Wert eins definiert, d.h.

$$c_p = 1 \wedge c_m = 0 \;\; \forall m \in \{p+1, p+2, \ldots, n-1\} = \mathcal{M}_{\text{SBT}}$$

Die Mengen $children_{\text{SBT}}$ und $parent_{\text{SBT}}$ sind nun wie folgt definiert:

$$children_{\text{SBT}}(t,s) = \{(t_{n-1}t_{n-2}\ldots t_{m+1}\overline{t_m}t_{m-1}\ldots t_0)\} \quad \forall m \in \mathcal{M}_{\text{SBT}}$$

$$parent_{\text{SBT}}(t,s) = \begin{cases} \emptyset & \text{falls } t = s \\ \{(t_{n-1}t_{n-2}\ldots t_{p+1}\overline{t_p}t_{p-1}\ldots t_0)\} & \text{falls } t \neq s \end{cases}$$

Ein spannender Binomialbaum kann also durch die Invertierung führender Nullen der Knotenadressen bestimmt werden.

Da die Anzahl der Knoten jeder Ebene i, $0 \leq i \leq n$ eines SBT der Höhe n durch $\binom{n}{i}$ gegeben ist und dieser Term ebenfalls die Anzahl der Knoten im Abstand i von einem gegebenen Knoten einer Hypercube-Struktur bezeichnet (Kap. 3.1.3), ist ein SBT abstandstreu. Somit kann jeder Knoten eines Hypercube-Rechners auf kürzestem Weg erreicht werden.

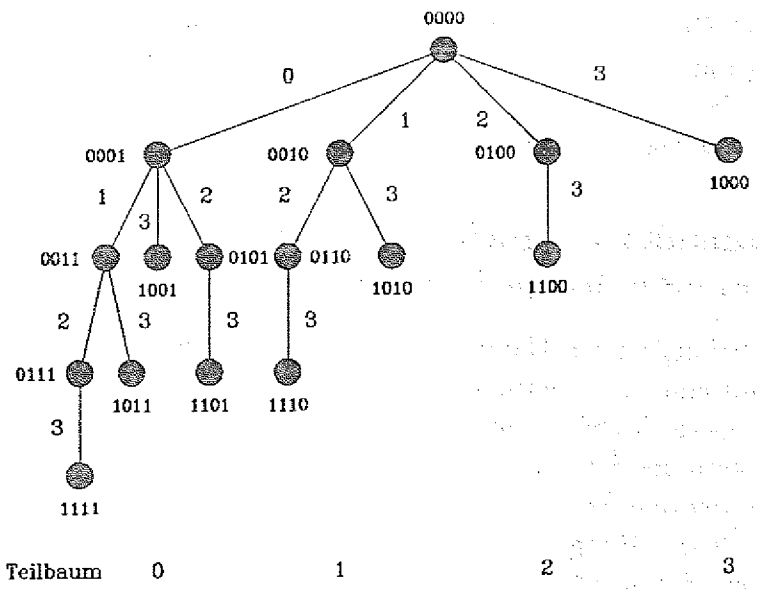


Abb. 5.8: Spannender Binomialbaum

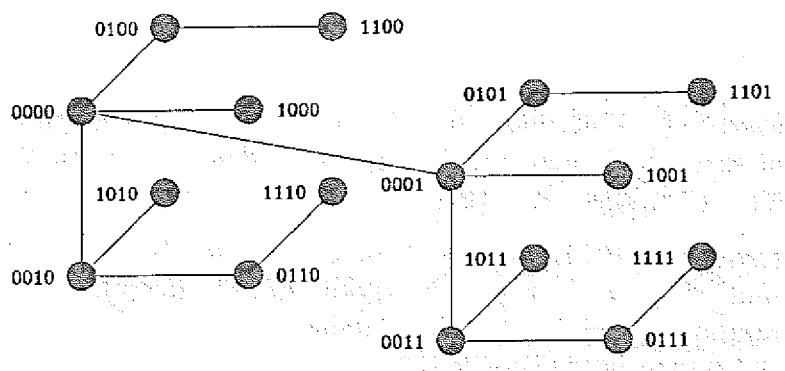


Abb. 5.9: Spannender Binomialbaum

Die Abbildungen 5.8 und 5.9 zeigen einen SBT der Höhe vier mit Wurzel 0 in unterschiedlicher Darstellung. Die Numerierung der Teilbäume in Abbildung 5.8 ist gleich der Anzahl aufeinanderfolgender Nullen am Ende einer Adresse eines Knotens des entsprechenden Teilbaumes. Die Beschriftungen der Kanten des Baumes werden später benötigt.
Ein SBT ist aufgrund der unterschiedlichen Höhen seiner Teilbäume als Kommunikationsstruktur für Ein-Kanal Kommunikation geeignet, wenn die Wurzel die Strategie verfolgt, Daten zuerst an diejenigen Teilbäume mit größter Höhe zu senden.

5.6.2 Spannender balancierter n-Baum, spannender balancierter Graph

Ein spannender balancierter n-Baum (*spanning balanced n-tree*, SBnT) [Jo&Ho89] ist ein spannender Baum eines n-dimensionalen Hypercube-Graphen, der dessen Knotenmenge in n etwa gleich große Mengen aufteilt. Jede dieser Teilmengen bildet einen zur Wurzel adjazenten Teilbaum des SBnT.
Sei $c = t \oplus s$ die relative Adresse zweier Knoten t, s eines Hypercube-Graphen. Dann sei $\mathcal{K}(t,s)$ definiert als die Menge der Zahlen k, die den Wert von c durch k-fache Rotation $Ro^k(c)$ minimieren, d.h. genauer

$$\begin{aligned} \mathcal{K}(t,s) &= \{k_1, \ldots, k_m \mid 0 \leq k_1 < \ldots < k_m < n\} \text{ mit} \\ Ro^{\alpha}(c) &= Ro^{\beta}(c)\ \forall \alpha, \beta \in \mathcal{K}(t,s)\ \wedge \\ Ro^{\alpha}(c) &< Ro^{\gamma}(c)\ \forall \alpha \in \mathcal{K}(t,s), \gamma \notin \mathcal{K}(t,s) \end{aligned}$$

Das Minimum k_1 aus $\mathcal{K}(t,s)$ wird mit $base_{\text{SBnT}}(t,s)$ bezeichnet. Für nicht zyklische Adressen c gilt $|\mathcal{K}(t,s)| = 1$, sonst gilt $|\mathcal{K}(t,s)| = \frac{n}{P(c)}$.

Beispiel 5.6.1 :

1. $base_{\text{SBnT}}(011100, 000000) = 2$, denn $c = 011100 \oplus 000000 = 011100$ ist nicht zyklisch und nur $Ro^2(c)$ ergibt den minimalen Wert 000111, also $\mathcal{K}(011100, 000000) = \{2\}$.
2. $base_{\text{SBnT}}(011010, 001111) = 0$, denn $c = 011010 \oplus 001111 = 010101$ und $P(c) = 2$. c ist also zyklisch und $Ro^0(c) (= Ro^2(c) = Ro^4(c))$ minimiert den Wert von c, daher $\mathcal{K}(011010, 001111) = \{0, 2, 4\}$ und $|\mathcal{K}(011010, 001111)| = \frac{6}{2} = 3$.

Zur Bestimmung eines SBnT in einem Hypercube-Rechner wird — wie im Fall der Bestimmung eines SBT — neben der eigenen Adresse nur die Wurzeladresse benötigt. Damit kann auch hier eine verteilte Bestimmung des Baumes erfolgen.

Definition 5.6.2 : (**Spannender balancierter n-Baum, SBnT**)

Ein SBnT mit der Wurzel s in einem n-dimensionalen Hypercube-Graphen ist gegeben durch den Vorgänger $parent_{\mathrm{SB}n\mathrm{T}}(t,s)$ und die Nachfolgermenge $children_{\mathrm{SB}n\mathrm{T}}(t,s)$ für alle Knoten t des Hypercube-Graphen.
Sei $k = base_{\mathrm{SB}n\mathrm{T}}(t,s)$, $c = t \oplus s$ und l die Stelle des ersten gesetzten Bit zyklisch rechts von der Stelle k in c, falls c mehr als ein gesetztes Bit enthält. In Beispiel 5.6.1 ist im ersten Fall $k = 2$, $c = 011100$ und $l = 4$ und im zweiten Fall $k = 0$, $c = 010101$ und wiederum $l = 4$. Es sei weiterhin

$$p = \begin{cases} -1 & \text{falls } c = 0 \\ k & \text{falls } c_k = 1, c_m = 0 \ \forall m \neq k \\ l & \text{sonst} \end{cases}$$

$$\mathcal{M}_{\mathrm{SB}n\mathrm{T}}(t,s) = \begin{cases} \{p+1,\ldots,n-1\} & \text{falls } k = 0 \\ \{(p+1) \bmod n,\ldots,(k-1) \bmod n\} & \text{falls } k > 0. \end{cases}$$

Es ist $c_m = 0$ für alle $m \in \mathcal{M}_{\mathrm{SB}n\mathrm{T}}(t,s)$, d.h. $\mathcal{M}_{\mathrm{SB}n\mathrm{T}}(t,s)$ ist die Menge direkt aufeinanderfolgender Stellenzahlen nullwertiger Bits zyklisch links von Bit p in c.

$$children_{\mathrm{SB}n\mathrm{T}}(t,s) =$$

$$\begin{cases} \{(t_{n-1}t_{n-2}\ldots t_{m+1}\overline{t_m}t_{m-1}\ldots t_0)\} \\ \qquad \forall m \in \mathcal{M}_{\mathrm{SB}n\mathrm{T}}(t,s) & \text{falls } c = 0 \\ \{q_m = (t_{n-1}t_{n-2}\ldots t_{m+1}\overline{t_m}t_{m-1}\ldots t_0)\} \\ \qquad \forall m \in \mathcal{M}_{\mathrm{SB}n\mathrm{T}}(t,s) \ \wedge \\ \qquad base_{\mathrm{SB}n\mathrm{T}}(q_m,s) = base_{\mathrm{SB}n\mathrm{T}}(t,s) & \text{falls } c \neq 0 \end{cases}$$

$$parent_{\mathrm{SB}n\mathrm{T}}(t,s) = \{(t_{n-1}t_{n-2}\ldots t_{p+1}\overline{t_p}t_{p-1}\ldots t_0)\} \quad \text{falls } c \neq 0$$

Aus der Definition der Menge $parent_{\mathrm{SB}n\mathrm{T}}$ in Definition 5.6.2 folgt direkt, daß ein SBnT abstandstreu ist, denn durch die *parent*-Funktion wird von jedem Knoten aus ein kürzester Weg zur Wurzel definiert. Damit beträgt die Höhe des Baumes n. Weiterhin gilt wegen der Bedingung der Gleichheit der Werte $base_{\mathrm{SB}n\mathrm{T}}(q_m,s)$ und $base_{\mathrm{SB}n\mathrm{T}}(t,s)$, daß ein Teilbaum aus einem anderen Teilbaum durch Rotation der Knotenadressen unter Ausschluß zyklischer Knoten erhalten werden kann. Der Ausschluß zyklischer Knoten erfolgt aufgrund des sonst mehrfachen Auftretens dieser Knoten in verschiedenen Teilbäumen, da durch n Rotationen einer zyklischen Adresse keine n verschiedenen Adressen entstehen und der entstehende Baum somit kein spannender Baum ist.
Eine ausführliche Vorstellung verschiedener Eigenschaften von SBnT-Graphen findet sich in [Ho&Jo89].

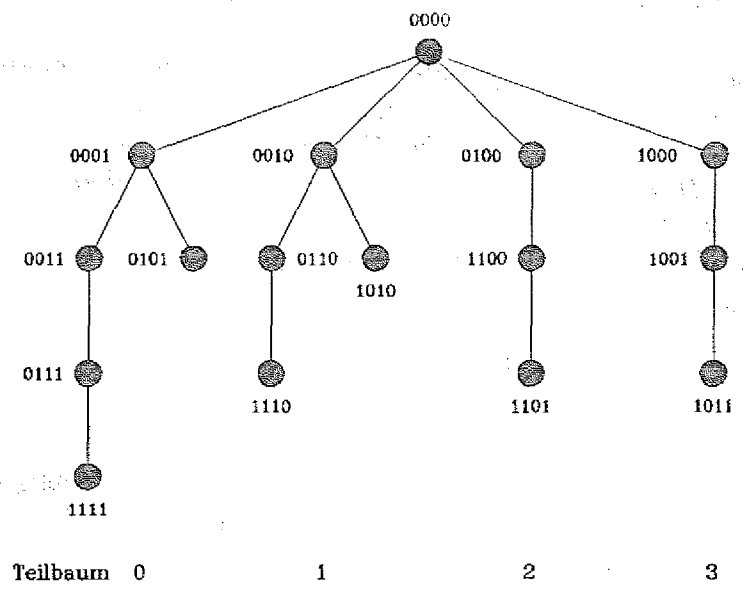


Abb. 5.10: Spannender balancierter 4-Baum

Abbildung 5.10 bzw. 5.11 zeigt einen SB4T bzw. einen SB5T mit Wurzel 0 in einer vier- bzw. fünfdimensionalen Hypercube-Struktur. Die den Teilbäumen zugeordnete Numerierung ist identisch mit dem Wert von $base_{\mathrm{SB}n\mathrm{T}}(t, s)$ für alle ihre Knoten t.
Als Beispiel seien die Berechnungen zu Teilbaum null des SB4T mit Wurzel 0 aus Abbildung 5.10 in Tabelle 5.3 angegeben. In Klammern angegebene Zahlen in den Spalten q_i, $1 \leq i \leq 3$, sind die Werte $base_{\mathrm{SB}n\mathrm{T}}(q_i, s)$.

t	k	l	p	$\mathcal{M}_{\mathrm{SB}n\mathrm{T}}$	q_1	q_2	q_3	$\mathbf{children}_{\mathrm{SB}n\mathrm{T}}$
0000								$\{1000, 0100, 0010, 0001\}$
0001	0		0	$\{1, 2, 3\}$	0011 (0)	0101 (0)	1001 (3)	$\{q_1, q_2\}$
0101	0	2	2	$\{3\}$			1101 (2)	$\emptyset$
0011	0	1	1	$\{2, 3\}$		0111 (0)	1011 (3)	$\{q_2\}$
0111	0	2	2	$\{3\}$			1111 (0)	$\{q_3\}$
1111	0	3	3	$\emptyset$				$\emptyset$

Tabelle 5.3: Berechnungen zu Teilbaum null eines SB4T mit Wurzel 0

Die Differenz der Anzahl von Knoten zweier Teilbäume ist für einen SBnT höchstens eins, falls n Primzahl ist, da in diesem Fall nur die zwei zyklischen Knoten 0 und 1 existieren.

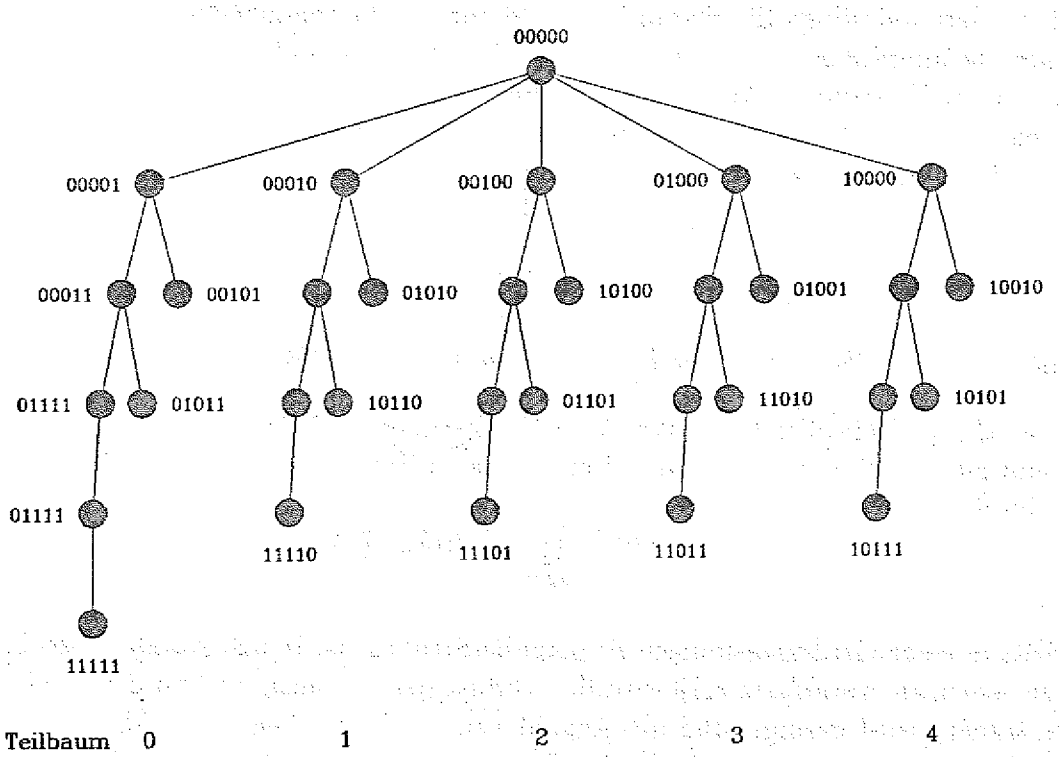


Abb. 5.11: Spannender balancierter 5-Baum

Damit muß bei der Konstruktion des SBnT durch Rotation eines seiner Teilbäume nur der Knoten 1 ausgeschlossen werden. Ist n aber keine Primzahl, so muß eine in Abhängigkeit von n steigende Differenz in Kauf genommen werden. Die entstehende Differenz führt zu einer unterschiedlichen Auslastung der Verbindungen der Wurzel bei Scatter und Multiscatter, denn die Menge der pro Verbindung zu übertragenden Daten hängt für diese Kommunikationsarten von der Anzahl der Knoten in dem durch die Verbindung erreichten Teilbaum ab.
Um die Daten genau gleichmäßig auf die Teilbäume zu verteilen, konstruiert man mehrere kürzeste Wege zu jedem zyklischen Knoten des SBnT. Der auf diese Weise entstehende Graph verliert allerdings die Baumeigenschaft und wird spannender balancierter Graph (*spanning balanced graph*, SBG) genannt [Jo&Ho89]. Er ist definiert durch eine Komposition von n, auf bestimmte Art rotierte, spannende balancierte n-Bäume.
Graphen $\mathcal{G}$, die durch Vereinigung von rotierten Graphen entstehen, werden im folgenden nur für die Wurzel 0 angegeben. Für jede andere Wurzel $s \neq 0$ läßt sich ein Graph des selben Typs durch Translation von $\mathcal{G}$ mit s berechnen:

$$\mathcal{G}(s) = Tr(s, \mathcal{G})$$

Definition 5.6.3 : **(Spannender balancierter Graph, SBG)**

> Sei $\mathcal{G}_{\mathrm{SB}n\mathrm{T}}(0)$ ein SBnT mit Wurzel 0. Ein spannender balanciert Graph $\mathcal{G}_{\mathrm{SBG}}(0)$ mit gleicher Wurzel in einem n-dimensionalen Hypercube-Graphen ist definiert durch:
>
> $$\mathcal{G}_{\mathrm{SBG}}(0) = \bigcup_{d \in \mathcal{D}} Ro^d(\mathcal{G}_{\mathrm{SB}n\mathrm{T}}(0)).$$

Ein SBG in einer vierdimensionalen Hypercube-Struktur ist in den Abbildungen 5.12[1] und 5.13 zu sehen. In Abbildung 5.13 wird der Verlust der Baumeigenschaft des SBG deutlich. In [Ho&Jo89] wird gezeigt, daß die Anzahl verschiedener Wege zu zyklischen Knoten s gleich $\frac{n}{P(s)}$ ist. Spannende balancierte Graphen sind durch die Eigenschaft der Balancierung besonders für n-Kanal Kommunikation geeignet.

5.6.3 n rotierte spannende Binomialbäume

Eine Kommunikationsstruktur zur Realisierung des Konzepts der vielfachen Wege (Kap. 5.2.3) und des Aufspaltens der zu übertragenden Datenmenge ist die Vereinigung von n, jeweils verschieden rotierten, spannenden Binomialbäumen gleicher Wurzel s und gleicher

[1] Es sei darauf hingewiesen, daß die in dieser Abbildung aus Gründen der besseren Anschauung gewählte Darstellung des SBG keinen Graphen darstellt, da einige Knotenbezeichnungen mehrfach verwendet werden. Dies gilt ebenfalls für die Darstellungen anderer Graphen in den Abbildungen 5.15 und 5.18.

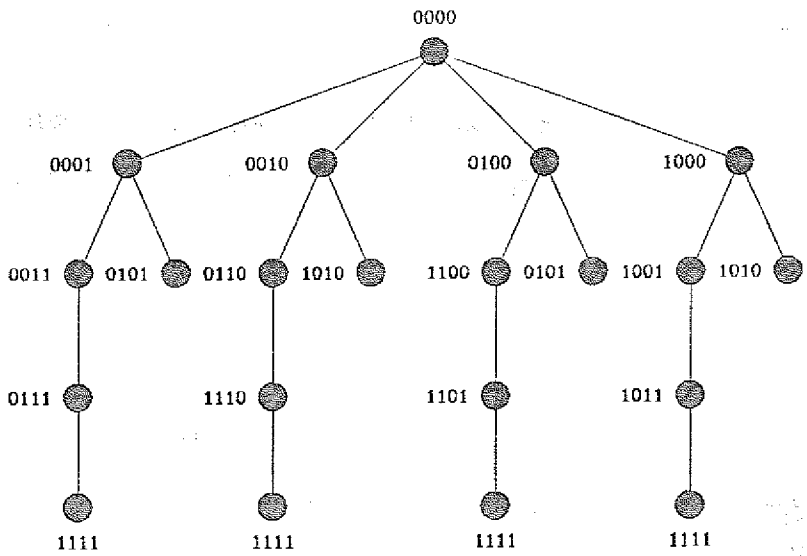


Abb. 5.12: Spannender balancierter Graph

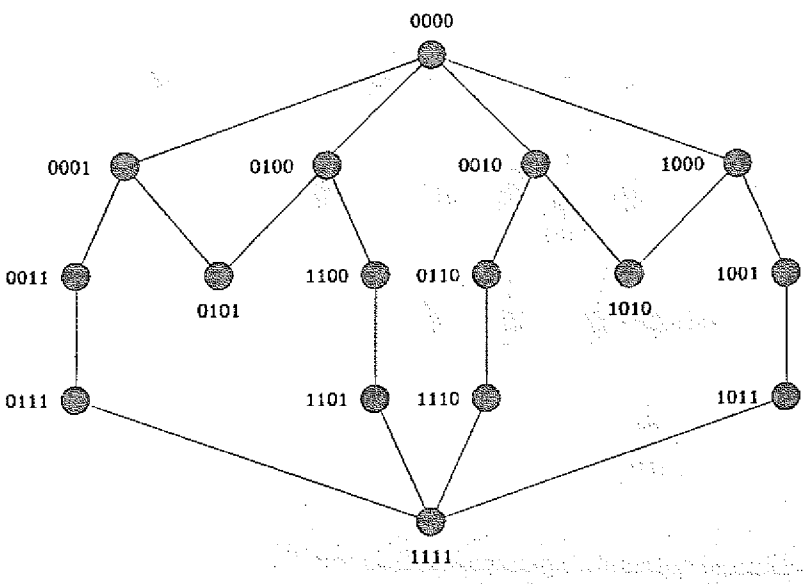


Abb. 5.13: Spannender balancierter Graph

Höhe n (n rotierte spannende Binomialbäume, *n rotated spanning binomial trees*, nRSBT) [Jo&Ho89]. Durch einen nRSBT-Graph werden n verschiedene Wege gleicher Länge von der Wurzel s zu je einem anderen Knoten bereitgestellt. Das Maximum der Weglängen beträgt weiterhin n.

Definition 5.6.4 : **(n rotierte spannende Binomialbäume, nRSBT)**

Sei $\mathcal{G}_{\text{SBT}}(0)$ ein SBT der Höhe n mit Wurzel 0. Ein nRSBT-Graph $\mathcal{G}_{n\text{RSBT}}(0)$ mit Wurzel 0 in einem n-dimensionalen Hypercube-Graphen ist definiert durch

$$\mathcal{G}_{n\text{RSBT}}(0) = \bigcup_{d \in \mathcal{D}} Ro^d(\mathcal{G}_{\text{SBT}}(0))$$

Für die Wurzel $s \neq 0$ gilt:

$$\mathcal{G}_{n\text{RSBT}}(s) = Tr(s, \mathcal{G}_{n\text{RSBT}}(0))$$

Der entscheidende Nachteil des nRSBT ist die fehlende Disjunktion der Kantenmengen der ihn erzeugenden Binomialbäume. Daher können die Verbindungen nicht in beliebiger Reihenfolge benutzt werden, weil sonst Verzögerungen durch Wartezeiten in den Knoten auftreten würden.

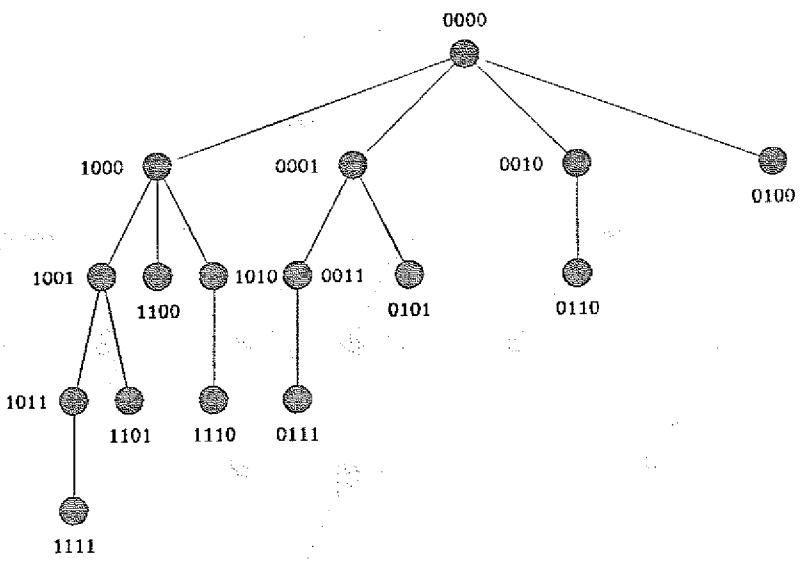


Abb. 5.14: Einmal rotierter, spannender Binomialbaum

Abbildung 5.14 zeigt einen der zur Konstruktion eines nRSBT mit Wurzel 0 benötigten Binomialbäume der Höhe vier. In Abbildung 5.15 ist der gesamte 4RSBT der Höhe vier mit

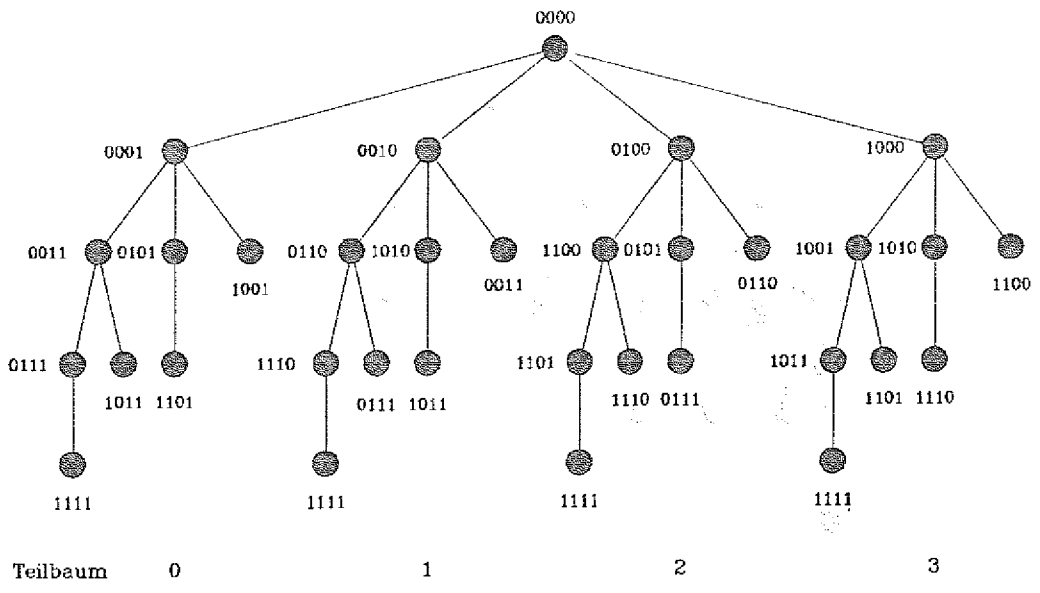


Abb. 5.15: Vier rotierte spannende Binomialbäume (4RSBT)

Wurzel 0 abgebildet, dessen spannende Teilbäume mittels der Anzahl der durchgeführten Rotationen numeriert werden. Mehrfach vorkommende Kanten sind nur einmal eingezeichnet.

5.6.4 n kantendisjunkte spannende Binomialbäume

Ein weiterer Graph, bestehend aus der Vereinigung von n spannenden Teilbäumen der Höhe n, deren Kantenmengen jedoch disjunkt sind, ist der nESBT-Graph (n kantendisjunkte spannende Binomialbäume, *n edge-disjoint spanning binomial trees*, nESBT) [Ho&Jo89]. Die einzelnen spannenden Binomialbäume werden so rotiert, daß die Wurzel s des nESBT der kleinste Teilbaum aller SBT ist. Die Richtung der mit s verbundenen Kante wird umgekehrt, so daß die zu s adjazenten Teilbäume als spannende Binomialbäume angesehen werden können, deren kleinster Teilbaum entfernt wurde (Abb. 5.16).
Es ergibt sich $n+1$ als Höhe des nESBT, welche minimal ist für alle möglichen Vereinigungen von n kantendisjunkten spannenden Binomialbäumen.

Definition 5.6.5 : **(n kantendisjunkte spannende Binomialbäume, nESBT)**

Sei $\mathcal{G}_{\text{SBT}}(0)$ ein SBT der Höhe n mit Wurzel 0. Ein nESBT-Graph $\mathcal{G}_{n\text{ESBT}}(0)$ mit

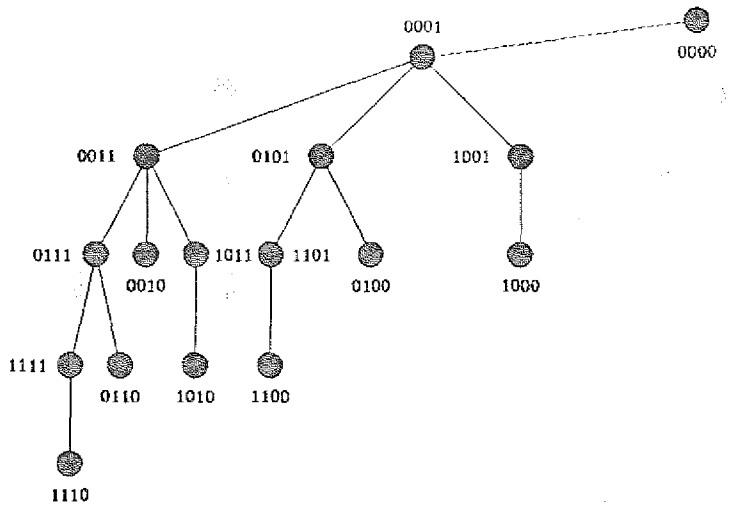


Abb. 5.16: Translierter, rotierter SBT mit Kante zu 0 in umgekehrter Richtung

Wurzel 0 in einem n-dimensionalen Hypercube-Graphen ist durch

$$\mathcal{G}_{n\text{ESBT}}(0) = \bigcup_{d \in \mathcal{D}} Tr(2^d, Ro^{n-d-1}(\mathcal{G}_{\text{SBT}}(0)))$$

definiert. Für Wurzeln $s \neq 0$ gilt:

$$\mathcal{G}_{n\text{ESBT}}(s) = Tr(s, \mathcal{G}_{n\text{ESBT}}(0))$$

Ein mit $0001 = 2^0$ translierter, dreifach rotierter, spannender Binomialbaum $\mathcal{G}_{\text{SBT}}$ ist in Abbildung 5.17 dargestellt ($\mathcal{G}_{\text{SBT}} = Tr(0001, Ro^3(\mathcal{G}(0)))$) und Abbildung 5.18 zeigt einen 4ESBT mit Wurzel 0. Aufgrund der Kantendisjunktion des nESBT müssen alle vier spannenden Teilbäume vollständig eingezeichnet werden. Deren Numerierung ist durch die Dimension $d \in \mathcal{D}$ aus obiger Definition gegeben. Die eingezeichnete Numerierung einiger Kanten wird später benötigt.

5.6.5 Ein-Kanal Broadcast

Bei der Durchführung von Ein-Kanal Broadcast müssen die M zu übertragenden Daten von der Wurzel gesendet und mindestens ein Datum nacheinander von $n-1$ Zwischenknoten weitergeleitet werden. Dafür sind mindestens n aufeinanderfolgende Routing-Schritte nötig. Daher ist

$$(M + n - 1) \cdot t_{comm} + n \cdot t_{start} \tag{5.1}$$

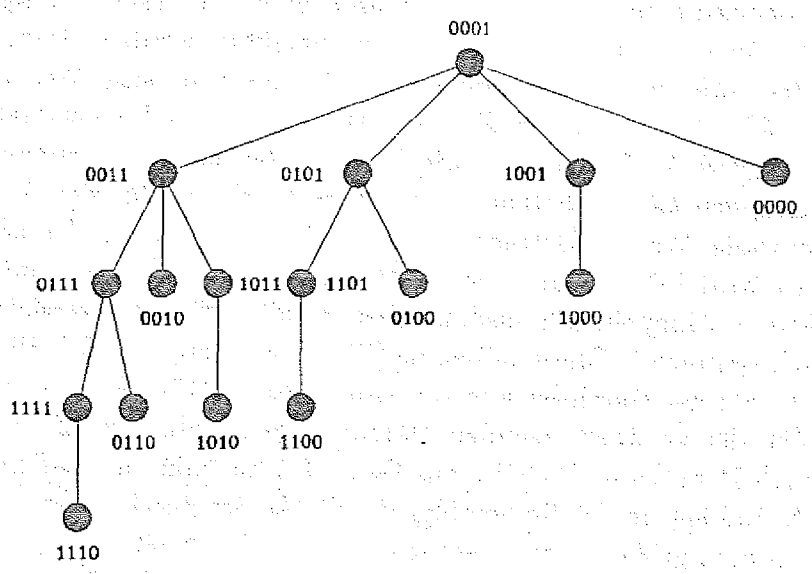


Abb. 5.17: Translierter, rotierter, spannender Binomialbaum

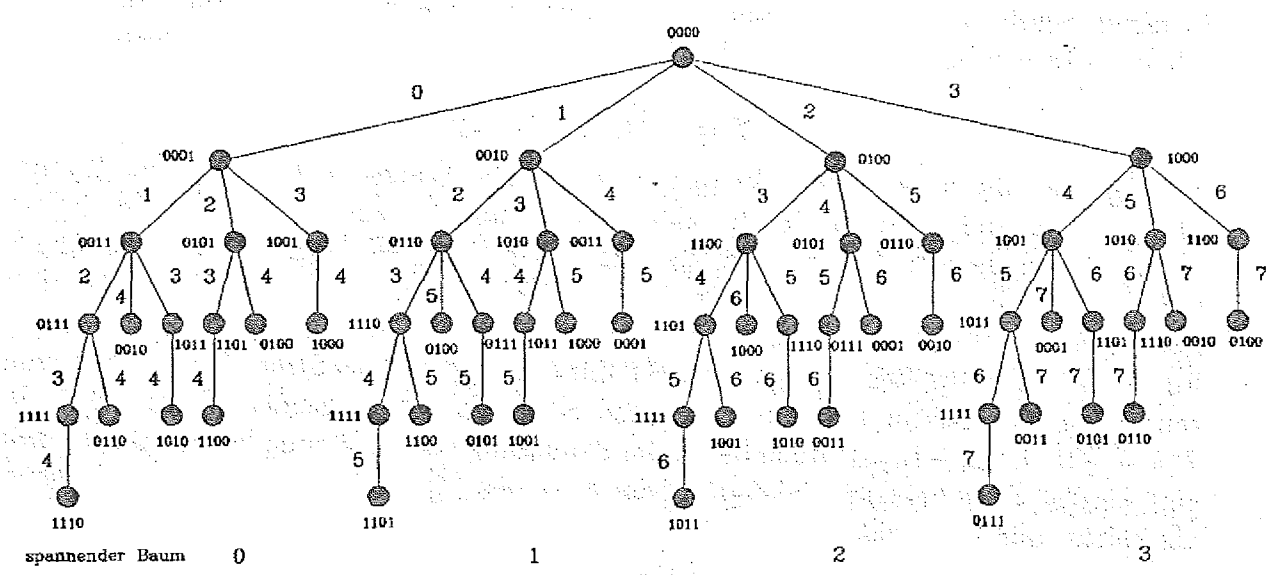


Abb. 5.18: Vier kantendisjunkte spannende Binomialbäume (4ESBT)

eine untere Schranke für den Zeitbedarf von Ein-Kanal Broadcast [Frai89].
Ein-Kanal Broadcast kann mit Hilfe jedes spannenden Baumes oder Teilgraphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit $\mathcal{V} = \mathcal{N}$ eines Hypercube-Graphen durchgeführt werden. Dazu gehört beispielsweise auch der einfachste spannende Baum, der Hamiltonsche Weg (*hamiltonian path*) [Tv&Na89, Jo&Ho89], der durch die Einbettung eines eindimensionalen Gitters mit N Knoten ohne äußere Verbindungen aufgebaut werden kann. Als spannender Baum eines Hypercube-Graphen kann ebenfalls der *balancierte 3-quasistar* mit N Knoten verwendet werden [Tv&Na89]. Weitere Betrachtungen hierzu finden sich in [Hara&&88]. Der geringste Zeitbedarf wird jedoch nur erzielt, falls sich die Anzahl der sendenden Knoten pro Routing-Schritt verdoppelt, d.h. nach n Schritten alle 2^n Knoten erreicht werden. Dies läßt sich nur durch spannende Binomialbäume (SBT) realisieren [Ho&Jo86].
Die effektivste Art der Durchführung von Broadcast mit Ein-Kanal Kommunikation hängt von der Größe der zu übertragenden Datenmenge M ab. Für $M = 1$ kann die untere Schranke durch Benutzung des SBT aus Def. 5.6.1 als Kommunikationsstruktur und Senden an alle Nachfolger in der Reihenfolge der Größe der durch sie erreichbaren Teilbäume, beginnend mit dem größten Baum, erreicht werden [Jo&Ho89]. Der größte, zu einem Knoten adjazente Teilbaum wird über Verbindung null, der zweitgrößte über Verbindung eins und der kleinste schließlich über Verbindung $n-1$ erreicht. Diese Strategie wird im folgenden als *SBT-Strategie* bezeichnet.
Aufgrund des rekursiven Aufbaus eines SBT ist die SBT-Strategie in allen Knoten durchzuführen, und nach n Schritten hat jeder Knoten das Datum genau einmal empfangen. In jedem Schritt i, $0 \leq i < n$, senden 2^i Prozessoren gleichzeitig über die Dimension i, d.h. die n Dimensionen des Hypercube-Rechners werden nacheinander abgearbeitet. Damit wird für $M = 1$ die Zeit

$$T = n \cdot (t_{comm} + t_{start})$$

benötigt und die untere Schranke (5.1) erreicht. Die Wurzel sendet in jedem Routing-Schritt, jeder andere Knoten nach Erhalt des Datums bis zum $(n-1)$-ten Schritt. Die Anzahl der sendenden Knoten verdoppelt sich somit nach jedem Schritt. In Abbildung 5.8 auf Seite 83 sind die Kanten des abgebildeten SBT in der Reihenfolge ihrer Benutzung numeriert.
Für $M > 1$ führt die Benutzung eines SBT mit der angegebenen Strategie zu dem, asymptotisch um den Faktor n über der unteren Schranke (5.1) liegenden, hohen Zeitbedarf $T = n \cdot (M \cdot t_{comm} + t_{start})$. Auch durch die Benutzung von Pipelining kann T nicht verringert werden, denn bei einer Paketgröße von B werden $\lceil \frac{M}{B} \rceil$ Pakete nach der SBT-Strategie verschickt, und man erhält

$$\begin{aligned} T &= n \cdot \left\lceil \frac{M}{B} \right\rceil \cdot (B \cdot t_{comm} + t_{start}) \\ &= n \cdot \left(M \cdot t_{comm} + \left\lceil \frac{M}{B} \right\rceil \cdot t_{start} \right). \end{aligned}$$

Dieser Wert wird aufgrund der minimal notwendigen Anzahl von n Routing-Schritten für $B = B_{opt} = M$ minimiert und man erhält auch hier $T_{min} = n \cdot M \cdot t_{comm} + n \cdot t_{start}$.
Die Benutzung eines nESBT statt eines SBT führt für $M > 1$ zu einer besseren Ausnutzung der Gesamtbandbreite des Hypercube-Rechners. Es werden die Konzepte der vielfachen Wege und Pipelining benutzt. Die zu übertragenden Daten werden in $i = \min\{n, M\}$ disjunkte, etwa gleich große Teilmengen aufgespalten. Für $M \geq n$ werden also n, sonst M Teilmengen gebildet. Im letzten Fall werden $n - M$ Teilbäume nicht benutzt. Je eine Teilmenge wird über genau einen spannenden Teilbaum durch Pipelining mit einer Paketgröße von B und zyklisches Senden der Wurzel des nESBT an die Wurzeln der beteiligten Teilbäume übertragen. Innerhalb der Teilbäume wird wiederum zuerst an die Teilbäume mit größter Knotenzahl gesendet. Durch diese Strategie werden $2 \cdot i - 1$ Routing-Schritte benötigt, um den letzten Knoten des i-ten Teilbaumes zu erreichen. Abbildung 5.18 auf Seite 93 zeigt die entsprechende Numerierung der Kanten des nESBT.
Es ergeben sich somit für jeden der i Teilbäume $\lceil \frac{M}{i \cdot B} \rceil$ Durchgänge, in denen durch die Wurzel jeweils B Daten übertragen werden. Das zuletzt verschickte Datenpaket benötigt nach Übertragung durch die Wurzel noch n Routing-Schritte zum Erreichen des letzten Knotens, da die Höhe des nESBT $n + 1$ beträgt. Damit benötigt man nun die Zeit

$$\begin{aligned} T &= i \cdot \left\lceil \frac{M}{i \cdot B} \right\rceil \cdot (B \cdot t_{comm} + t_{start}) + n \cdot (B \cdot t_{comm} + t_{start}) \\ &= M \cdot t_{comm} + \left\lceil \frac{M}{B} \right\rceil \cdot t_{start} + n \cdot B \cdot t_{comm} + n \cdot t_{start} \\ &= (M + n \cdot B) \cdot t_{comm} + \left(\left\lceil \frac{M}{B} \right\rceil + n \right) \cdot t_{start}. \end{aligned}$$

Man erhält, unabhängig von i, die minimale Zeit

$$T_{min} = \left(\sqrt{M \cdot t_{comm}} + \sqrt{n \cdot t_{start}} \right)^2$$

durch Wahl der optimalen Paketgröße

$$B_{opt} = \sqrt{\frac{M \cdot t_{start}}{n \cdot t_{comm}}},$$

die durch Nullsetzen der Ableitung nach B berechnet werden kann.
Für große Datenmengen M wird die angegebene untere Schranke (5.1) damit asymptotisch erreicht. Dies kann anschaulich durch die — im Vergleich zum Senden über einen SBT — bessere Ausnutzung der Gesamtbandbreite (Kap. 3.1.8) des Hypercube-Rechners erklärt werden.

5.6.6 n-Kanal Broadcast

Eine untere Schranke für n-Kanal Broadcast ist

$$\left(\left\lceil\frac{M}{n}\right\rceil + n - 1\right) \cdot t_{comm} + n \cdot t_{start}, \tag{5.2}$$

da die Wurzel nur noch die reine Übertragungszeit $\lceil\frac{M}{n}\rceil \cdot t_{comm}$ zum Senden der M Daten über n Verbindungen benötigt. Ansonsten ändert sich hier im Vergleich zur unteren Schranke von Ein-Kanal Broadcast nichts [Frai89].
Die Auswahl der geeigneten Bäume oder Graphen, die für die Durchführung von n-Kanal Broadcast mit minimaler Anzahl von Routing-Schritten herangezogen werden können, beschränkt sich im Gegensatz zu Ein-Kanal Broadcast nicht auf Binomialbäume. Vielmehr kann der Transfer über jeden spannenden Baum der Höhe n nach n Schritten beendet werden [Ho&Jo86, Jo&Ho89].
Ähnlich wie für den Ein-Kanal Fall wird auch hier für kleine Datenmengen bei der Wahl des Baumes auf die Eigenschaft der minimalen Höhe Wert gelegt, während für große Werte von M die Bandbreite des Hypercube-Rechners möglichst gut genutzt werden soll. Zusätzlich ist eine effektive Unterstützung der Kommunikation auf allen Verbindungen eines Prozessors wichtig.
Ein nRSBT wird für den Fall $M \leq n$ benutzt. Die Datenmenge wird in M einelementige Mengen geteilt, und diese werden von der Wurzel gleichzeitig über je einen spannenden Baum übertragen. Auch hier werden $n - M$ Teilbäume nicht genutzt. Innerhalb der spannenden Bäume wird wiederum die bekannte SBT-Strategie angewendet, also das Senden an die Nachfolger in der Reihenfolge der Größe der durch sie erreichten Teilbäume. Pipelining wird aufgrund der fehlenden Disjunktion der Kantenmengen der spannenden Teilbäume nicht benutzt. Damit werden n Schritte mit der Übertragung je eines Datums benötigt, also

$$T = n \cdot (t_{comm} + t_{start}).$$

Die untere Schranke (5.2) wird also genau erreicht.
Die Benutzung der beschriebenen Strategie und eines nRSBT führt für $M > n$ zu einem um den Faktor n gegenüber der unteren Schranke (5.2) erhöhten Zeitbedarf. Bei einer Paketgröße von B, $\lceil\frac{M}{n \cdot B}\rceil$ von der Wurzel auszuführenden Routing-Schritten und $n - 1$ weiteren Schritten zum Erreichen des letzten Knotens, wird die Zeit

$$T = n \cdot \left\lceil\frac{M}{n \cdot B}\right\rceil \cdot (B \cdot t_{comm} + t_{start})$$

benötigt. Die optimale Paketgröße B_{opt} beträgt $\lceil\frac{M}{n}\rceil$, denn es gilt $1 \leq B \leq M$, und die minimale Zeit T_{min} beträgt damit

$$T_{min} = M \cdot t_{comm} + n \cdot t_{start}.$$

Für $M > n$ kann die untere Schranke durch Benutzung eines nESBT asymptotisch erreicht werden. Auch hier teilt man die Datenmenge auf und erhält n Pakete. Da die Kantenmengen der spannenden Teilbäume disjunkt sind, wird Pipelining mit einer Paketgröße B angewendet. Die Wurzel sendet auf allen n Verbindungen gleichzeitig, und die spannenden Binomialbäume werden wieder sequentiell nach der SBT-Strategie durchlaufen [Jo&Ho89]. Man erhält wiederum $\lceil \frac{M}{n \cdot B} \rceil$ Pakete pro spannenden Teilbaum des nESBT, d.h. $\lceil \frac{M}{n \cdot B} \rceil$ Routing-Schritte müssen von der Wurzel ausgeführt werden. Die im letzten Routing-Schritt versendeten Daten haben noch einen Weg der Länge n zurückzulegen, da die Höhe des nESBT $n+1$ beträgt. Damit ergibt sich die Zeit

$$\begin{aligned} T &= \left\lceil \frac{M}{n \cdot B} \right\rceil \cdot (B \cdot t_{comm} + t_{start}) + n \cdot (B \cdot t_{comm} + t_{start}) \\ &= \left(\frac{M}{n} + n \cdot B \right) \cdot t_{comm} + \left(\left\lceil \frac{M}{n \cdot B} \right\rceil + n \right) \cdot t_{start}. \end{aligned}$$

Q. F. Stout und B. Wagar haben in [St&Wa90] eine geringfügige Modifikation dieser Strategie vorgeschlagen, die mit einem Routing-Schritt weniger auskommt (siehe auch [Frai89]). Die Änderung beruht darauf, daß der in Ebene $n+1$ des nESBT liegende Knoten eines spannenden Teilbaumes $j, 0 \leq j < n$, auch in Ebene $n-1$ des Teilbaumes $(j+1) \bmod n$ liegt. Das nochmalige Senden der Pakete des letzten Routing-Schritts für Teilbaum j an Teilbaum $(j+1) \bmod n$, $0 \leq j < n$, führt dazu, daß der Algorithmus nach Erreichen der Ebene n durch die ursprünglich zuletzt gesendeten Daten bzw. nach Erreichen der Ebene $n-1$ durch die zusätzlich gesendeten Daten beendet werden kann.
Abbildung 5.19 zeigt die Teilbäume eins und zwei des 4ESBT aus Abbildung 5.18 auf Seite 93. Die Numerierung der Kanten zeigt die Reihenfolge der Benutzung der Verbindungen zum Transfer des letzten Pakets für den linken Teilbaum. Die Wurzel sendet das Paket zunächst an den vorgesehenen Teilbaum $j = 1$, und im nächsten Routing-Schritt an Teilbaum $(j+1) \bmod n = 2$. Nach Erreichen des Knotens 1111 des linken Teilbaumes bzw. des Knotens 1101 des rechten Teilbaumes kann der Algorithmus im Vergleich mit der ursprünglichen Strategie einen Schritt früher beendet werden.
Zwar spart man durch diese Vorgehensweise einen Routing-Schritt, jedoch wird das Verkehrsaufkommen durch das doppelte Senden der letzten Pakete gesteigert und $n-1$ Knoten jedes Teilbaumes erhalten sie doppelt. Durch diese Modifikation wird nun nur noch die Zeit

$$\begin{aligned} T &= \left\lceil \frac{M}{n \cdot B} \right\rceil \cdot (B \cdot t_{comm} + t_{start}) + (n-1) \cdot (B \cdot t_{comm} + t_{start}) \\ &= \left(\frac{M}{n} + (n-1) \cdot B \right) \cdot t_{comm} + \left(\left\lceil \frac{M}{n \cdot B} \right\rceil + n - 1 \right) \cdot t_{start} \end{aligned}$$

benötigt. Mit Hilfe der Nullstellen der Ableitung nach B erhält man die optimale Pa-

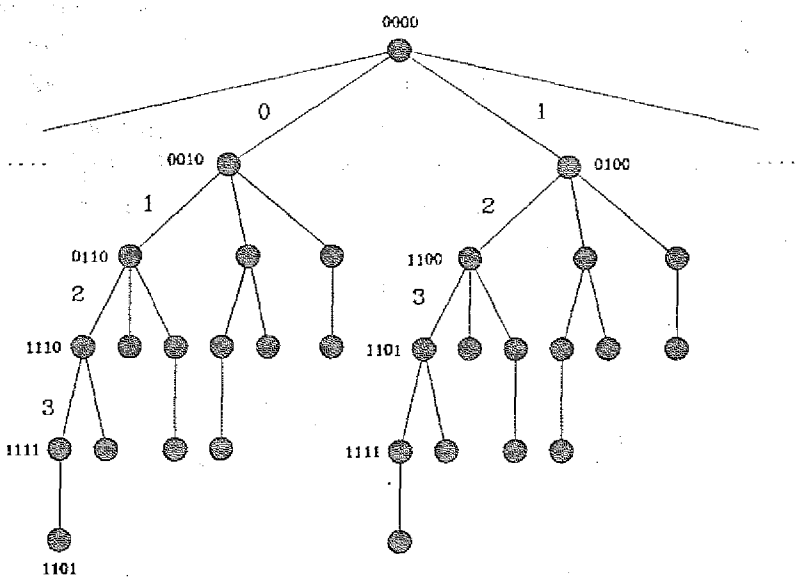


Abb. 5.19: **Doppeltes Senden des letzten Pakets über zwei Teilbäume**

ketgröße

$$B_{opt} = \sqrt{\frac{M \cdot t_{start}}{n \cdot (n-1) \cdot t_{comm}}}$$

und schließlich den minimalen Zeitbedarf

$$T_{min} = \left(\sqrt{\frac{M}{n} \cdot t_{comm}} + \sqrt{(n-1) \cdot t_{start}} \right)^2 .$$

5.7 Multibroadcast

5.7.1 Ein-Kanal Multibroadcast

Bei der Durchführung von Ein-Kanal Multibroadcast sendet jeder Knoten M Daten an jeden anderen Knoten, d.h. jeder Knoten muß $(N-1)\cdot M$ Daten empfangen. Die benötigte minimale reine Übertragungszeit beträgt also

$$(N-1)\cdot M\cdot t_{comm}.$$

n Routing-Schritte sind nötig, um alle Knoten zu erreichen. Da jedoch die von verschiedenen Knoten stammenden Pakete i.a. verschieden sind und zum Transfer von Zwischenknoten zusammengefaßt werden können, kann hier bei hinreichend großem M der Fall auftreten, daß das Senden eines großen Pakets gleichzeitig zum Senden von kleineren Paketen geschieht. Dann würde der Zeitbedarf zum Senden der kleineren Pakete nicht in die Gesamttransferzeit eingehen und Startzeiten könnten gespart werden. Ein Beispiel zur Verdeutlichung dieser Problematik findet sich in Kapitel 5.8.2.
In der in dieser Arbeit untersuchten Literatur ist zwar kein Algorithmus für Ein-Kanal Multibroadcast zu finden, der die minimale reine Übertragungszeit benötigt und in dessen Gesamttransferzeit weniger als n Startzeiten eingehen, jedoch fehlt ein überzeugendes Argument, welches die Berechnung einer unteren Schranke mittels Addition der minimalen reinen Übertragungszeit und der für die n Routing-Schritte benötigten n Startzeiten erlauben würde. Somit kann nur

$$\max\{(N-1)\cdot M\cdot t_{comm}, n\cdot t_{start}\} \tag{5.3}$$

als eine untere Schranke für die benötigte Zeit angegeben werden [Frai89]. In Abhängigkeit von t_{comm} und t_{start} werden für große Werte von M die reine Übertragungszeit und für kleine Werte von M die Startzeiten dominieren.
Die zugrunde liegende Kommunikationsstruktur muß einen Weg von jedem Knoten zu jedem anderen Knoten enthalten. Dies kann durch die Vereinigung von N geeigneten Graphen (spannende Bäume oder andere Teilgraphen eines Hypercube-Graphen, die alle Knoten enthalten) mit je einer Wurzel eines Graphen oder Baumes pro Knoten realisiert werden. Die Übertragung geschieht gleichzeitig über die N Bäume mit jeweils gleicher Strategie und der Benutzung des Konzepts der dynamische Paketgröße (Kap. 5.2.3).
Der geringste Zeitbedarf wird durch die Benutzung von spannenden Binomialbäumen mit der gleichen Strategie wie bei Ein-Kanal Broadcast (SBT-Strategie) erzielt, d.h. nach n Schritten wird die Kommunikation beendet. Da nun — im Gegensatz zu Broadcast mit nur einem Sender — genau N verschiedene Pakete nacheinander über die n Dimensionen übertragen werden, werden pro Schritt p Pakete zu einem zusammengefaßt. p kann aufgrund der vorausgesetzten Ein-Kanal Kommunikation höchstens gleich zwei sein. Die entsprechenden

Pakete sind die in allen vorherigen Schritten empfangenen Pakete sowie das an die anderen Knoten zu verbreitende eigene Paket [Jo&Ho89].
Zuerst (nullter Schritt) schickt also jeder Knoten sein eigenes Paket über Dimension null, d.h. jeder Knoten empfängt ein Paket der Länge M, und damit ist eines der $N-1$ zu empfangenden Pakete bereits angekommen. Dieses wird mit den eigenen Daten zu einem Paket der Länge $2 \cdot M$ zusammengefaßt, welches im nächsten Schritt über Dimension eins übertragen wird. In diesem Schritt empfängt jeder Knoten somit ein Paket, welches sich aus zwei Datenmengen der Länge M zusammensetzt, kopiert diese in den eigenen Speicherbereich und erstellt ein neues Paket der Größe $4 \cdot M$, welches sich aus den soeben empfangenen (Größe $2 \cdot M$), den eigenen (Größe M) und allen vorher empfangenen Daten (Größe M) zusammensetzt. Nach dessen Übertragung besteht das daraufhin erstellte Paket wiederum aus allen empfangenen sowie den eigenen Daten (Größe $(4+2+1+1) \cdot M = 8 \cdot M$). Nach jedem Schritt verdoppelt sich die Größe der zu übertragenden Pakete. Durch die Ausführung von Schritt i, $0 \leq i < n$, werden in jedem Knoten die Daten von $\sum_{j=0}^{i} 2^j = 2^{i+1} - 1$ verschiedenen Prozessoren über Dimension i empfangen. Nach n Schritten sind damit alle $\sum_{i=0}^{n-1} 2^i = 2^n - 1 = N - 1$ verschiedenen Pakete eingetroffen.

Routing-Schritt	**Adressen der Ursprungsknoten**
0	0001
1	0010 0011
2	0100 0101 0110 0111
3	1000 1001 1010 1011 1100 1101 1110 1111

Tabelle 5.4: Von Knoten 0 empfangene Datenpakete in den Routing-Schritten null bis drei

Tabelle 5.4 zeigt die Adressen derjenigen Knoten eines vierdimensionalen Hypercube-Rechners, deren Datenmengen Knoten 0 in den Routing-Schritten null bis drei empfängt. In Routing-Schritt zwei beispielsweise empfängt Knoten 0 über Dimension zwei von Knoten 0100. Dieser hat in den vorherigen Schritten über Dimension eins bzw. null Daten von Knoten 0110 bzw. 0101 erhalten. Knoten 0110 wiederum hat in Schritt null von Knoten 0111 empfangen.
Abbildung 5.20 zeigt die zurückgelegten Wege der von Knoten 0 gesendeten Daten. Die Reihenfolge der Benutzung der Kanten ist die gleiche wie bei der SBT-Strategie. Die Größe der von den Zwischenknoten zu übertragenden Datenmengen sind in Vielfachen von M neben den benutzten Kanten eingetragen. Man beachte, daß nach dem nullten Schritt auch Knoten 0 Daten anderer Knoten weiterleitet, d.h. Zwischenknoten ist.
Die zur Durchführung von Multibroadcast in einem dreidimensionalen Hypercube-Rechner notwendigen Kommunikationsschritte sind in Abbildung 5.21 zu sehen. In jedem Routing-

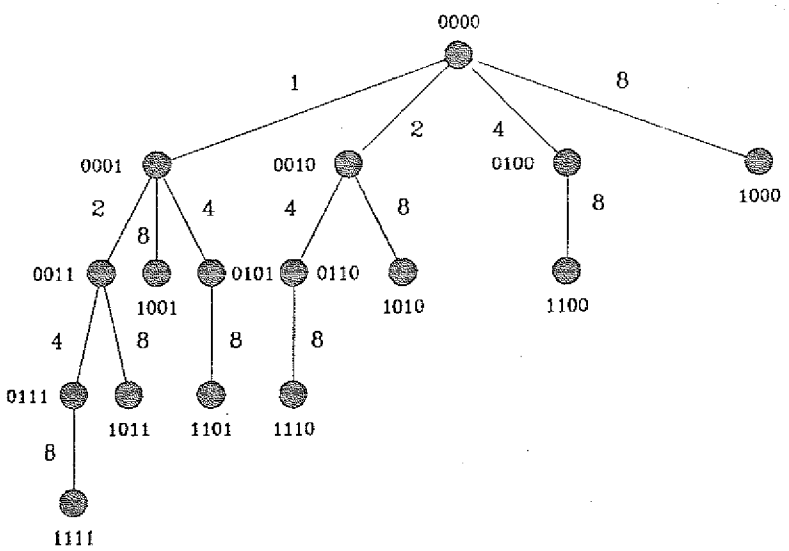


Abb. 5.20: Spannender Binomialbaum mit Paketgrößen für Ein-Kanal Multibroadcast

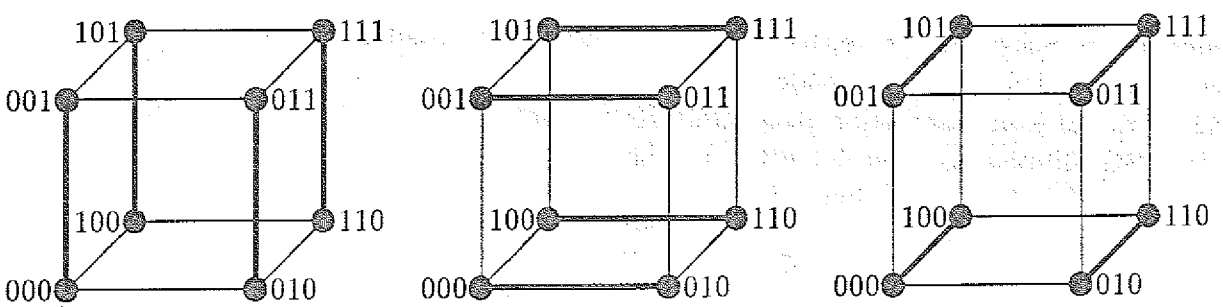


Abb. 5.21: Ein-Kanal Multibroadcast in einem dreidimensionalen Hypercube-Rechner

Schritt werden die dick eingezeichneten Verbindungen in beiden Richtungen benutzt.
Da alle Knoten in jedem Routing-Schritt über die gleiche Dimension senden und empfangen, sind die Voraussetzungen für Ein-Kanal Kommunikation erfüllt, und jeder Knoten sendet und empfängt Daten von Anfang bis Ende des Algorithmus.
Die für die n Schritte benötigte Zeit T beträgt bei einer sich von Schritt zu Schritt verdoppelnden Paketgröße

$$B(i) = 1 + \sum_{j=0}^{i-1} 2^j = 2^i$$

beginnend mit M und endend mit $(1 + \sum_{i=0}^{n-2} 2^i) \cdot M = \frac{N \cdot M}{2}$ Daten

$$\begin{aligned} T &= \sum_{i=0}^{n-1} (2^i \cdot M \cdot t_{comm} + t_{start}) \\ &= (N-1) \cdot M \cdot t_{comm} + n \cdot t_{start} \end{aligned}$$

und überschreitet somit zwar die untere Schranke (5.3), jedoch wird sowohl die minimale Anzahl von Routing-Schritten als auch die minimale reine Übertragungszeit benötigt.

5.7.2 n-Kanal Multibroadcast

Nur die von der Wurzel benötigte reine Übertragungszeit reduziert sich bei n-Kanal Kommunikation um den Faktor n gegenüber Ein-Kanal Kommunikation. Damit ist

$$\max \left\{ \left\lceil \frac{(N-1) \cdot M}{n} \right\rceil \cdot t_{comm}, n \cdot t_{start} \right\} \tag{5.4}$$

eine untere Schranke des Zeitbedarfs für n-Kanal Multibroadcast [Frai89].
Ähnlich wie bei n-Kanal Broadcast wird auch hier statt spannender Binomialbäume eine für n-Kanal Kommunikation geeignetere Kommunikationsstruktur benutzt. Dies sind hier der SBG-Graph $\mathcal{G}_{\text{SBG}}$ oder der nRSBT-Graph $\mathcal{G}_{n\text{RSBT}}$.
Sei $\mathcal{G}^*$ im folgenden einer der beiden Graphen

$$\begin{aligned} \mathcal{G}^* &= \bigcup_{i \in \mathcal{N}} \mathcal{G}_{\text{SBG}}(i) \qquad \text{oder} \\ \mathcal{G}^* &= \bigcup_{i \in \mathcal{N}} \mathcal{G}_{n\text{RSBT}}(i), \end{aligned}$$

d.h. $\mathcal{G}^*$ besteht aus der Vereinigung von N Graphen, die jeder die gesamte Knotenmenge des Hypercube-Rechners umfassen und die Höhe bzw. den Durchmesser n besitzen.
Jeder Knoten sendet seine Daten über den von ihm ausgehenden Teilgraphen von $\mathcal{G}^*$ nach der Strategie, die in n disjunkte Teilmengen aufgeteilten Daten der Länge $\lceil \frac{M}{n} \rceil$ gleichzeitig

über alle spannenden Teilbäume zu übertragen. Alle Zwischenknoten senden gleichzeitig an ihre Nachfolger im jeweiligen Teilgraphen. Damit werden mindestens n Schritte benötigt [Jo&Ho89].
Auch hier werden Datenmengen, die über die gleiche Verbindung eines Knotens übertragen werden sollen, in einem Paket zusammengefaßt. Damit läßt sich die minimale Anzahl von Routing-Schritten erreichen. Die maximale Anzahl der zusammengefaßten Pakete $e(i)$ und damit die maximale Paketgröße B_{max}, die während eines Routing-Schritts i, $0 \leq i < n$, auftritt, ist für dessen Ausführungszeit von Bedeutung. Die Summe über den Zeitbedarf aller Routing-Schritte ergibt den gesamten Zeitbedarf. Es gilt

$$\begin{aligned} T &= \sum_{i=0}^{n-1} \left(\left\lceil \frac{M}{n \cdot B} \right\rceil (B \cdot t_{comm} + t_{start}) \cdot e(i) \right) \\ &= \sum_{i=0}^{n-1} \left(\frac{M}{n} \cdot t_{comm} \cdot e(i) + \left\lceil \frac{M}{n \cdot B} \right\rceil \cdot t_{start} \cdot e(i) \right). \end{aligned}$$

Mit $B = B(i) = e(i) \cdot \frac{M}{n}$ ergibt sich

$$T = \sum_{i=0}^{n-1} e(i) \cdot \frac{M}{n} \cdot t_{comm} + n \cdot t_{start}.$$

Es ist sowohl für die Vereinigung von SBG- als auch für nRSBT-Graphen, $e(i) = \binom{n}{i+1}$ [Jo&Ho89] und weiter

$$\begin{aligned} T &= \sum_{i=1}^{n} \binom{n}{i} \cdot \frac{M}{n} \cdot t_{comm} + n \cdot t_{start} \\ &= \frac{(2^n - 1) \cdot M}{n} \cdot t_{comm} + n \cdot t_{start} \\ &= \frac{(N-1) \cdot M}{n} \cdot t_{comm} + n \cdot t_{start}. \end{aligned}$$

Auch für n-Kanal Multibroadcast wird also die angegebene untere Schranke (5.4) nicht erreicht. Allerdings wird weder mehr als die minimale reine Übertragungszeit noch mehr als die minimale Anzahl von Routing-Schritten benötigt.
Mit der Stirlingschen Formel

$$n! \approx \left(\frac{n}{e}\right)^n \cdot \sqrt{2 \cdot \pi \cdot n}, \quad n \in \{0, 1, 2, \ldots\} \tag{5.5}$$

ergibt sich die maximal benötigte Paketgröße B_{max} zu

$$B_{max} = \max\{e(i) \mid 0 \leq i < n\} \cdot \frac{M}{n} = \binom{n}{\lceil \frac{n}{2} \rceil} \cdot \frac{M}{n}$$

$$
\begin{aligned}
&= \frac{n!}{(\frac{n}{2})! \cdot (\frac{n}{2})!} \cdot \frac{M}{n} \\
&\approx \frac{\left(\frac{n}{e}\right)^n \cdot \sqrt{2 \cdot \pi \cdot n}}{\left(\frac{n}{2 \cdot e}\right)^{\frac{n}{2}} \cdot \sqrt{2 \cdot \pi \cdot \frac{n}{2}} \cdot \left(\frac{n}{2 \cdot e}\right)^{\frac{n}{2}} \cdot \sqrt{2 \cdot \pi \cdot \frac{n}{2}}} \cdot \frac{M}{n} \\
&= \sqrt{\frac{2}{\pi}} \cdot \frac{N \cdot M}{n^{3/2}}
\end{aligned}
$$

Mit der Benutzung des nESBT- statt der SBG- oder nRSBT-Graphen ergibt sich eine Übertragung, die aufgrund der Höhe $n+1$ des nESBT eine Startzeit mehr, ansonsten jedoch die gleichen Zeit benötigt.

untere Schranke	Graph	M	Paketgröße	Zeitdedarf
Ein-Kanal Broadcast				
$(M+n-1) \cdot t_{comm} + n \cdot t_{start}$	SBT	1	1	$n \cdot (t_{comm} + t_{start})$
	nESBT	> 1	$\sqrt{\frac{M \cdot t_{start}}{n \cdot t_{comm}}}$	$\left(\sqrt{M \cdot t_{comm}} + \sqrt{n \cdot t_{start}}\right)^2$
n-Kanal Broadcast				
$(\frac{M}{n} + n - 1) \cdot t_{comm} + n \cdot t_{start}$	nRSBT	$\leq n$	1	$n \cdot (t_{comm} + t_{start})$
	nESBT	$> n$	$\sqrt{\frac{M \cdot t_{start}}{n \cdot (n-1) \cdot t_{comm}}}$	$\left(\sqrt{\frac{M}{n} \cdot t_{comm}} + \sqrt{(n-1) \cdot t_{start}}\right)^2$
Ein-Kanal Multibroadcast				
$\max\{(N-1) \cdot M \cdot t_{comm}, n \cdot t_{start}\}$	SBT	≥ 1	$\leq \frac{N \cdot M}{2}$	$(N-1) \cdot M \cdot t_{comm} + n \cdot t_{start}$
n-Kanal Multibroadcast				
$\max\{\frac{(N-1) \cdot M}{n} \cdot t_{comm}, n \cdot t_{start}\}$	SBG	≥ 1	$\leq \sqrt{\frac{2}{\pi}} \cdot \frac{N \cdot M}{n^{3/2}}$	$\frac{(N-1) \cdot M}{n} \cdot t_{comm} + n \cdot t_{start}$
	nRSBT	≥ 1	$\leq \sqrt{\frac{2}{\pi}} \cdot \frac{N \cdot M}{n^{3/2}}$	$\frac{(N-1) \cdot M}{n} \cdot t_{comm} + n \cdot t_{start}$

Tabelle 5.5: Zeitbedarf von Broadcast/Multibroadcast

Abschließend werden in Tabelle 5.5 die wichtigsten Ergebnisse zu Broadcast und Multibroadcast zusammengefaßt. In der Spalte „Paketgröße" sind für Broadcast die optimalen und für Multibroadcast die maximalen Paketgrößen eingetragen.
Für Ein- und n-Kanal Multibroadcast sowie für einelementige Datenmengen bei Ein-Kanal Broadcast und für Datenmengen kleinerer Größe als n bei n-Kanal Broadcast werden nur die entsprechenden unteren Schranken der reinen Übertragungszeit benötigt. In den restlichen Fällen wird diese Zeit nur asymptotisch bei großen zu übertragenden Datenmengen erreicht.
Der Zeitbedarf der vorgestellten Algorithmen ist minimal unter allen in der angegebenen Literatur beschriebenen Algorithmen.

5.8 Scatter

Algorithmen für *Scatter* werden in [Jo&Ho89, Ho&Jo86, Frai89, Sa&Sc89a, Bert&&89, St&Wa90] beschrieben. In ihnen kommen die bekannten Konzepte des Pipelining, der vielfachen Wege und der dynamischen Paketgröße (Kap. 5.2.3) zur Anwendung. Der für den Zeitbedarf bedeutsamste Unterschied zu den vorgestellten Algorithmen für Broadcast besteht in der wurzelorientierten Kommunikation. Darunter wird verstanden, daß die Wurzel alle Pakete für alle Empfänger selbst senden muß, denn eine Vervielfältigung von Nachrichten, die sich bei Broadcast als sinnvoll erweist, entfällt hier aufgrund der Unterschiedlichkeit der zu versendenden Pakete. Damit ist insgesamt ein höherer Zeitbedarf als für Broadcast zu erwarten.

5.8.1 Ein-Kanal Scatter

Bei Ein-Kanal Scatter sendet die Quelle mindestens $N-1$ Pakete der Größe M und benötigt dafür die minimale reine Übertragungszeit

$$(N-1) \cdot M \cdot t_{comm}.$$

Der längste zurückzulegende Weg hat die Länge n, und daher werden mindestens n Routing-Schritte benötigt. Aufgrund einer analogen Überlegung wie bei Multibroadcast fehlt ein überzeugendes Argument für eine Addition der minimalen reinen Übertragungszeit und des Zeitbedarfs für die n Routing-Schritte zu einer unteren Schranke. Daher wird in [Jo&Ho89]

$$\max\{(N-1) \cdot M \cdot t_{comm}, n \cdot t_{start}\} \tag{5.6}$$

als eine untere Schranke für den Zeitbedarf von Ein-Kanal Scatter angegeben.
Die minimale reine Übertragungszeit kann nur erreicht werden, falls mit der Beendigung des Sendens der Wurzel auch der Algorithmus beendet wird. Dies wiederum ist, wie bei Ein-Kanal Multibroadcast, nur durch Senden über einen spannenden Binomialbaum nach der SBT-Strategie unter Anwendung des Konzepts der dynamischen Paketgröße zu erreichen. Dies bedeutet, daß n Routing-Schritte benötigt werden und in jedem Schritt nur Übertragungen über Verbindungen genau einer Dimension stattfinden. Im Unterschied zu Broadcast wird hier jedoch kein Datenpaket vervielfältigt, denn je zwei Knoten erhalten verschiedene Pakete. Alle Daten für die Knoten in Teilbaum i, $0 \leq i < n$, werden während des i-ten Routing-Schritts in einem Paket verschickt. Da im i-ten Teilbaum eines SBT der Höhe n insgesamt 2^{n-i-1} Knoten zu finden sind, hat das im i-ten Routing-Schritt, $0 \leq i < n$, von der Wurzel verschickte Paket die Größe

$$B = B(i) = 2^{n-i-1} \cdot M.$$

Die Zwischenknoten spalten das erhaltene Paket auf und senden alle für einen Teilbaum bestimmten Daten in einem Paket an dessen Wurzel. Hierbei wird aufgrund des rekursiven Aufbaus des SBT die Reihenfolge des Sendens ebenfalls nach der SBT-Strategie festgelegt. Alle auf diese Weise im i-ten Routing-Schritt von 2^i Knoten gesendeten Datenmengen haben die gleiche Größe. Damit ist der Datentransfer beendet, wenn die Wurzel das letzte Paket (Größe M) geschickt hat. Der Zeitbedarf T beträgt somit

$$\begin{aligned} T &= \sum_{i=0}^{n-1} \left(2^{n-i-1} \cdot M \cdot t_{comm} + t_{start}\right) \\ &= (N-1) \cdot M \cdot t_{comm} + n \cdot t_{start} \end{aligned}$$

und die untere Schranke (5.6) wird aufgrund der n aufeinanderfolgenden Routing-Schritte überschritten. Die minimale reine Übertragungszeit wird jedoch erreicht.
Wie in allen vorherigen Datentransferalgorithmen — mit Ausnahme von n-Kanal Broadcast — wird auch hier jedes Paket auf kürzestem Weg übertragen und genau einmal von den Empfängern erhalten.

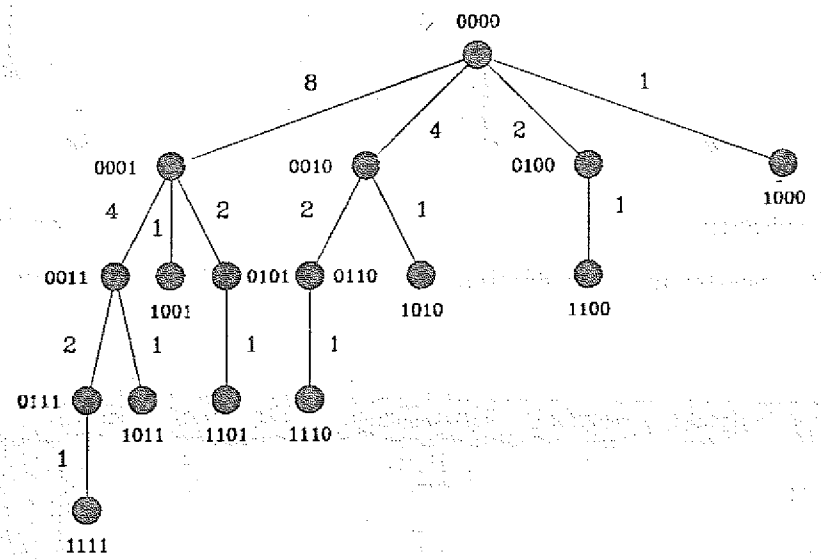


Abb. 5.22: Paketgröße bei Ein-Kanal Scatter in einem Hypercube-Rechner

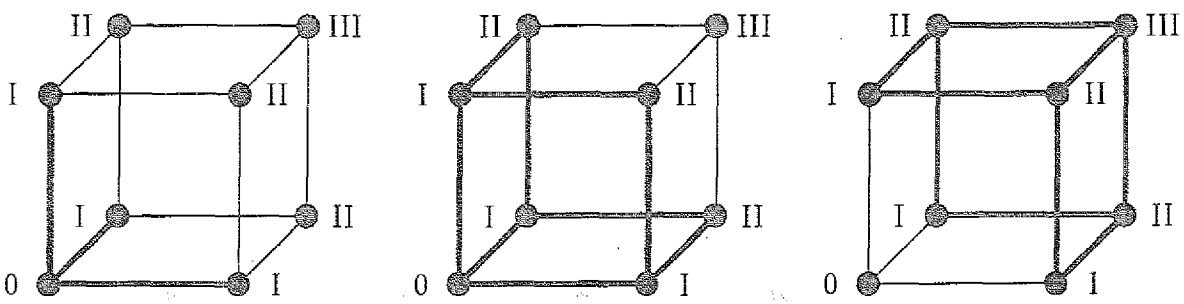
Abbildung 5.22 zeigt einen SBT der Höhe vier mit Wurzel 0 in einem vierdimensionalen Hypercube-Rechner. Die Kanten sind mit den Größen der über sie gesendeten Pakete in Vielfachen von M beschriftet. Pakete gleicher Größe werden im gleichen Routing-Schritt gesendet. Begonnen wird mit der Übertragung zu Knoten 0001.

5.8.2 n-Kanal Scatter

Zum Senden von $N-1$ Paketen der Größe M mit n-Kanal Kommunikation benötigt die Wurzel mindestens die Zeit

$$\left\lceil \frac{(N-1)\cdot M}{n} \right\rceil \cdot t_{comm}.$$

Im Vergleich zu Ein-Kanal Scatter ändert sich sonst bezüglich der Berechnung einer unteren Schranke nichts. Die minimale Anzahl von Startzeiten beträgt auch hier n. Hier läßt sich jedoch ein Beispiel angeben, welches zeigt, daß der minimale Zeitbedarf für die reine Übertragung und das Minimum der Startzeiten nicht zu einer unteren Schranke addiert werden dürfen [St&Wa90].



nullter Schritt erster Schritt zweiter Schritt

Abb. 5.23: Scatter in einem dreidimensionalen Hypercube-Rechner mit zwei Startzeiten

Routing-Schritt	Sender	Empfänger	Datenmengen	Zeitbedarf
0	0	II, III	4	$4/3 \cdot M \cdot t_{comm} + t_{start}$
1	0	I	3	$M \cdot t_{comm} + t_{start}$
	I	III	1	$1/6 \cdot M \cdot t_{comm} + t_{start}$
2	I	II	3	$1/2 \cdot M \cdot t_{comm} + t_{start}$
	II	III	1	$1/3 \cdot M \cdot t_{comm} + t_{start}$

Tabelle 5.6: Routing-Schritte bei n-Kanal Scatter und deren Zeitbedarf

Betrachtet wird eine Scatter-Operation in einem dreidimensionalen Hypercube-Rechner. Diese benötigt zwar aufgrund des Durchmessers drei der Verbindungsstruktur auf jeden

Fall drei Routing-Schritte, jedoch können ab einer gewissen Größe der zu übertragenden Datenmenge M unter Erhalt der minimalen reinen Übertragungszeit zwei Routing-Schritte gleichzeitig ausgeführt werden [St&Wa90]. Damit wird einmal die Startzeit t_{start} gespart. Abbildung 5.23 zeigt die während der Routing-Schritte aktiven Verbindungen. Die Knoten sind in drei Gruppen I, II und III jeweils gleichen Abstands eins, zwei und drei von der Wurzel 0 eingeteilt. Tabelle 5.6 zeigt den genauen Ablauf des Algorithmus. Die Spalte „Empfänger" enthält diejenigen Empfängeradressen, für die die gesendeten Pakete des jeweiligen Routing-Schritts bestimmt sind.
In Schritt null sendet Knoten 0 die insgesamt vier Datenmengen für die Gruppen II und III über die drei Verbindungen in drei gleich großen Paketen. Schritt eins besteht aus dem Senden der Daten für Gruppe I durch Knoten 0 über drei Verbindungen und gleichzeitigem Senden der Daten für Gruppe III durch alle Knoten der Gruppe I über die sechs zur Verfügung stehenden Verbindungen. Das Senden des Knotens 0 benötigt bei hinreichend großer Datenmenge M mehr Zeit als das Senden durch Gruppe I. Dadurch kann der zweite Schritt, bestehend aus dem Senden von Gruppe I an Gruppe II und von Gruppe II an Gruppe III schon angefangen und aufgrund der großen Bandbreite zwischen den Knoten der Gruppen I und II sogar beendet werden, bevor Knoten 0 den ersten Routing-Schritt beendet hat. Der zweite Routing-Schritt mit geringem Zeitaufwand, insbesondere die dafür benötigte Startzeit, wird also durch den ersten Routing-Schritt mit großem Zeitaufwand „verdeckt".
Damit geht nur der Zeitbedarf des nullten und des ersten Routing-Schritts in die insgesamt benötigte Zeit T ein und man erhält

$$\begin{aligned} T &= \frac{4}{3} \cdot M \cdot t_{comm} + t_{start} + M \cdot t_{comm} + t_{start} \\ &= \frac{7}{3} \cdot M \cdot t_{comm} + 2 \cdot t_{start} \end{aligned}$$

Damit treten nur zwei Startzeiten in der Berechnung von T auf. Die reine Übertragungszeit kann jedoch nicht weiter verringert werden.
In [St&Wa90] wird ein Algorithmus vorgestellt, in dessen Zeitbedarf in einem n-dimensionalen Hypercube-Rechner nur ca. $\frac{n}{\log_2 n}$ Startzeiten eingehen, also

$$T \approx \frac{(N-1) \cdot M}{n} \cdot t_{comm} + \frac{n}{\log_2 n} \cdot t_{start}.$$

Der zu erzielende Zeitgewinn wird jedoch erst ab einer — von der Startzeit t_{start} abhängenden — Größe von M erreicht, ist dann jedoch von M unabhängig, d.h. falls ein Zeitgewinn auftritt, ist er für alle Werte von M konstant.
Als untere Schranke für den Zeitbedarf von Scatter kann somit nicht die Addition der minimalen reinen Übertragungszeit und der minimalen Anzahl von Startzeiten herangezogen

werden. Daher wird in [Jo&Ho89] das Maximum beider Werte

$$\max\left\{\left\lceil\frac{(N-1)\cdot M}{n}\right\rceil\cdot t_{comm}, n\cdot t_{start}\right\} \tag{5.7}$$

als untere Schranke angegeben.
Diese wird sowohl in dem hier vorgestellten Algorithmus als auch in dem Algorithmus aus [St&Wa90] überschritten. In beiden Fällen wird lediglich die gleiche minimale reine Übertragungszeit $\lceil\frac{(N-1)\cdot M}{n}\rceil\cdot t_{comm}$ erreicht.
Der hier vorgestellte Algorithmus basiert entweder auf dem SBG- oder dem nRSBT-Graphen. Die für beide Kommunikationsstrukturen benutzte Strategie ist die *reverse-breadth-first* Strategie [Jo&Ho89]. Sie stellt sicher, daß die Daten für alle Knoten der $(n-i)$-ten Ebene, $0 \le i < n$, im i-ten Routing-Schritt von der Wurzel gesendet werden. Damit benötigt man n Routing-Schritte. Werden in jedem Routing-Schritt nach der reverse-breadth-first Strategie auf jeder Verbindung der Wurzel genau gleich viele Daten übertragen, d.h. die Anzahl der erreichbaren Knoten muß für alle Verbindungen gleich sein, so wird für das Senden von B Daten in einem Routing-Schritt die Zeit $\lceil\frac{B}{n}\rceil\cdot t_{comm}+t_{start}$ benötigt. Mit der Abstandstreue beider Graphen, d.h. jeder Knoten ist auf kürzestem Weg erreichbar, und der Anzahl $\binom{n}{i}$ der Knoten, die sich im Abstand i zur Wurzel befinden [Jo&Ho89], ergibt sich der Zeitbedarf

$$\begin{aligned} T &= \sum_{i=0}^{n-1}\left(\binom{n}{n-(i+1)}\cdot\frac{M}{n}\cdot t_{comm}+t_{start}\right) \\ &= \sum_{i=0}^{n-1}\left(\binom{n}{i+1}\cdot\frac{M}{n}\cdot t_{comm}\right)+n\cdot t_{start} \\ &= \frac{(N-1)\cdot M}{n}\cdot t_{comm}+n\cdot t_{start} \end{aligned}$$

bei einer Paketgröße von

$$B = B(i) = \binom{n}{i+1}\cdot\frac{M}{n}$$

im i-ten, $0 \le i < n$, Routing-Schritt. Die maximale Paketgröße B_{max} wird im Routing-Schritt $\lceil\frac{n}{2}\rceil$ bzw. $\lfloor\frac{n}{2}\rfloor$ erreicht und läßt sich mit Hilfe der Stirlingschen Formel (5.5) berechnen zu

$$\begin{aligned} B_{max} &= \binom{n}{\lceil\frac{n}{2}\rceil}\cdot\frac{M}{n} \\ &\approx \sqrt{\frac{2}{\pi}}\cdot\frac{N\cdot M}{n^{3/2}}. \end{aligned}$$

Die Bedingung der gleichmäßigen Verteilung der in einem Routing-Schritt zu versendenden Daten auf die n Verbindungen der Wurzel wird im Fall des nRSBT-Graphen durch das Konzept der vielfachen Wege erreicht. Jedes einzelne der $\binom{n}{i+1}$ in Schritt i zu verschickenden Datenpakete wird in $j = \min\{n, M\}$ disjunkte Teilmengen aufgespalten. $\binom{n}{i+1}$ Teilmengen aus jeweils verschiedenen Datenpaketen werden zusammengefaßt und über eine Verbindung übertragen. Falls $M < n$, d.h. $j = M$, werden nur j Teilbäume des nRSBT benutzt.
Für SBG-Graphen gilt, daß nur zyklische Knoten auf verschiedenen Wegen erreichbar sind. Die Anzahl der erreichbaren zyklischen Knoten einer Ebene ist für alle Wurzelverbindungen gleich. Somit kann die Anzahl der pro Verbindung und Routing-Schritt zu übertragenden Daten genau balanciert werden, d.h. über jede Verbindung wird pro Routing-Schritt die gleiche Datenmenge übertragen [Jo&Ho89]. Damit ergibt sich für die Benutzung des SBG der gleiche Zeitbedarf wir für den nRSBT.

5.9 Total Exchange

5.9.1 Ein-Kanal Total Exchange

Ein-Kanal Total Exchange besteht aus N-maligem Ein-Kanal Scatter von M Daten mit N verschiedenen Wurzeln. Zur Bestimmung einer unteren Schranke für den Zeitbedarf wird die benötigte Bandbreite (*bandwidth*) [Jo&Ho89] herangezogen. Sie beträgt für eine Scatter-Operation von einem Knoten aus sowohl für Ein- als auch für n-Kanal Kommunikation

$$\sum_{i=1}^{n} i \cdot \binom{n}{i} \cdot M = \frac{n \cdot N \cdot M}{2},$$

denn genau $\binom{n}{i}$, $0 < i \leq n$, Pakete der Länge M müssen von der Wurzel aus einen Weg der Länge i zurücklegen. Insgesamt wird für Total Exchange also die Bandbreite

$$\frac{n \cdot N^2 \cdot M}{2} \tag{5.8}$$

benötigt. Aufgrund der Annahme der Ein-Kanal Kommunikation kann pro Routing-Schritt auf maximal N Verbindungen gleichzeitig gesendet werden. Damit beträgt die minimale reine Übertragungszeit

$$\frac{n \cdot N \cdot M}{2} \cdot t_{comm}.$$

Auch hier trifft die bei der Vorstellung eines Algorithmus für Ein-Kanal Multibroadcast (Kap. 5.7.1) formulierte Überlegung zu, daß verschiedene Routing-Schritte derart überlagert werden können, daß weniger als n Startzeiten in die Gesamttransferzeit eingehen. In keiner der angegebenen Quellen ist jedoch ein Algorithmus für Ein-Kanal Total Exchange zu finden, der die minimale reine Übertragungszeit und weniger als n Routing-Schritte benötigt. Somit wird

$$\max\left\{\frac{n \cdot N \cdot M}{2} \cdot t_{comm}, n \cdot t_{start}\right\} \tag{5.9}$$

als eine untere Schranke für den Zeitbedarf von Ein-Kanal Total Exchange angegeben [Jo&Ho89].

Die Benutzung der Vereinigung von N SBT-Graphen nach der SBT-Strategie, d.h. jeder Knoten sendet nach der SBT-Strategie über den von ihm ausgehenden Binomialbaum, führt zu einer Transferzeit, die der unteren Schranke (5.9) am nächsten kommt [Jo&Ho89]. Pakete verschiedener Sender an einen Empfänger, die über die gleiche Verbindung eines Knotens übertragen werden sollen, werden in einem Paket zusammengefaßt (dynamische Paketgröße, Kap. 5.2.3). Jeder Knoten schickt dabei im i-ten Routing-Schritt alle Datenpakete sowohl die bis dahin erhaltenen, die nicht für ihn selbst bestimmt sind, als auch die

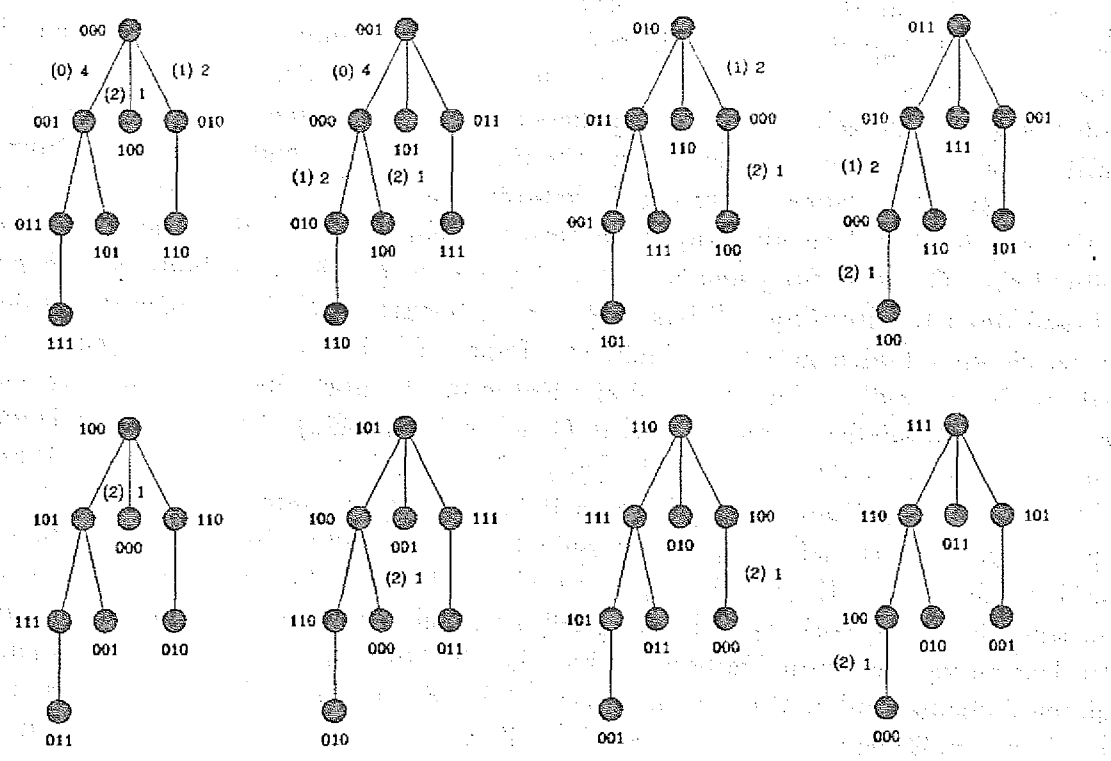


Abb. 5.24: Kommunikationsstruktur für Ein-Kanal Total Exchange in einem dreidimensionalen Hypercube-Rechner

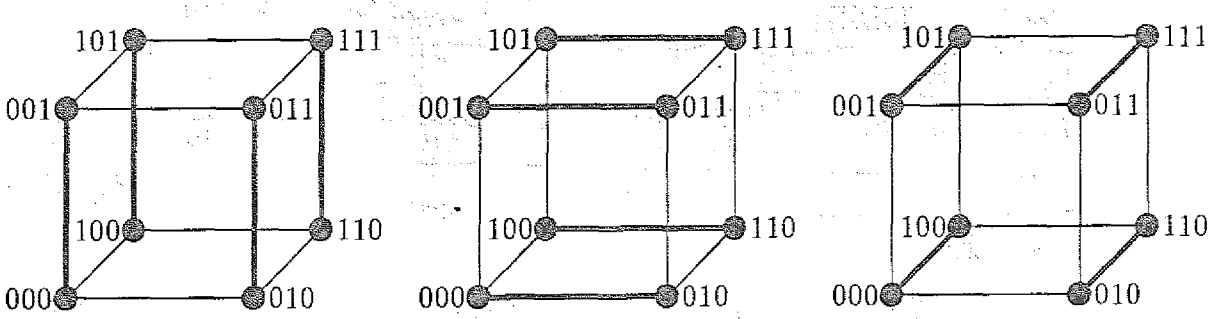


Abb. 5.25: Ein-Kanal Total Exchange in einem dreidimensionalen Hypercube-Rechner

von ihm selbst zu verschickenden Pakete in einem Gesamtpaket über die Verbindung in Dimension i, d.h. an die Wurzel des höchsten von ihm aus erreichbaren Teilbaumes.
Abbildung 5.24 zeigt die acht verwendeten spannenden Binomialbäume in einem dreidimensionalen Hypercube-Rechner. Die mit Knoten 0 verbundenen Kanten sind mit den Größen des über sie gesendeten Datenpakets in dem in Klammern angegebenen Routing-Schritt beschriftet. Es werden jeweils nur die für den entsprechenden einzelnen Baum, d.h. für eine Scatter-Operation relevanten Paketgrößen angegeben. Da die in einem Routing-Schritt von Knoten 0 zu übertragenden Daten in einem Paket zusammengefaßt werden, bleibt dessen Größe B für jeden Schritt i, $0 \leq i < n$, gleich. In Abbildung 5.25 sind die während des i-ten Routing-Schritts, $0 \leq i < 2$, benutzten Verbindungen durch die dick eingezeichneten Linien gekennzeichnet. Die Paketgröße ist $B = 4 \cdot M$ in jedem Schritt, denn im Schritt null sendet Knoten 0 alle Daten für die über Dimension null erreichbaren Knoten im Binomialbaum mit Wurzel 0. Gleichzeitig empfängt Knoten 0 über Dimension null ein Datenpaket der Größe $4 \cdot M$ (siehe Binomialbaum mit Wurzel 001). Dieses besteht aus einem für Knoten 0 bestimmten Paket und drei weiteren Paketen für die Knoten 010, 110 und 100. Im nächsten Schritt sendet Knoten 0 über Dimension eins die eigenen Daten an die beiden Knoten 010 und 110 in dem Binomialbaum mit Wurzel 0 sowie die von Knoten 001 im vorherigen Routing-Schritt erhaltenen Daten für die Knoten 010 und 110. Das zu verschickende Datenpaket hat also die Größe $4 \cdot M$. In Schritt eins empfängt Knoten 0 ebenfalls über Dimension eins ein Paket der Länge $4 \cdot M$, bestehend aus je zwei Einzelpaketen für den Knoten 100 und sich selbst, von den Knoten 011 und 010. Im letzten Schritt schließlich sendet Knoten 0 über Dimension zwei an Knoten 100 vier Einzelpakete, die von den Knoten 001, 010, 011 und von sich selbst stammen. Empfangen werden Pakete der Knoten 110, 111, 100 und 101.

Routing-Schritt	gesendete Pakete	empfangene Pakete
0	001 011 111 101	001
1	010 110	010 011
2	100	110 111 100 101

Tabelle 5.7: von Knoten 0 gesendete und empfangene Datenpakete in den Routing-Schritten null bis zwei

Tabelle 5.7 listet die Adressen derjenigen Knoten auf, für die die in Routing-Schritt i von Knoten 0 gesendeten Datenpakete bestimmt sind, bzw. von denen Knoten 0 in Routing-Schritt i das für ihn bestimmte Paket empfängt.
Da jeder Knoten genau einmal als Wurzel eines SBT auftritt, gilt analog auch für die Knoten 001, ..., 111, daß die Größe der zu übertragenden Pakete konstant gleich $4 \cdot M$ ist.

Bei Durchführung des Algorithmus wird also in n Routing-Schritten über jeweils eine Verbindung jedes Knotens ein Datenpaket konstanter Größe empfangen und ein weiteres gleichzeitig gesendet. Die Möglichkeiten der Ein-Kanal Kommunikation werden also voll ausgeschöpft, und es werden nacheinander alle n Dimensionen des Hypercube-Rechners genutzt.
Die Paketgröße B beträgt in jedem Schritt

$$B = \frac{N \cdot M}{2}.$$

Der Zeitbedarf T errechnet sich damit auf einfache Weise zu

$$\begin{aligned} T &= n \cdot (B \cdot t_{comm} + t_{start}) \\ &= \frac{n \cdot N \cdot M}{2} \cdot t_{comm} + n \cdot t_{start}, \end{aligned}$$

womit zwar wiederum die untere Schranke (5.9) nicht erreicht, aber nur die minimale reine Übertragungszeit und n Routing-Schritte benötigt werden.

5.9.2 n-Kanal Total Exchange

Die untere Schranke

$$\max\left\{\frac{N \cdot M}{2} \cdot t_{comm}, n \cdot t_{start}\right\} \tag{5.10}$$

für den Zeitbedarf von n-Kanal Total Exchange erhält man durch die gleiche Überlegung bezüglich der benötigten Bandbreite , der pro Routing-Schritt maximal benutzbaren Verbindungen und möglicher Überlagerungen von Routing-Schritten wie für Ein-Kanal Total Exchange in Kapitel 5.9.1 [Jo&Ho89].
Da sich gegenüber Ein-Kanal Total Exchange nur die Annahmen bezüglich der Kommunikationmöglichkeiten jedes einzelnen Knotens verändert haben, bleibt die benötigte Bandbreite die gleiche wie in Formel (5.8). Es können nun jedoch alle Verbindungen des Hypercube-Rechners gleichzeitig genutzt werden. Damit ergibt sich die für große Datenmengen M um den Faktor n geringere untere Schranke (5.10).
Durch Benutzung eines für n-Kanal Kommunikation besonders geeigneten Graphen kann die untere Schranke (5.10) nur bezüglich reiner Übertragungszeit erreicht werden. Wie im Fall von n-Kanal Multibroadcast (Kap. 5.7.2) kann sowohl der SBG-Graph als auch der nRSBT-Graph als Basis für die Kommunikationsstruktur benutzt werden. Diese setzt sich dann entweder aus der Vereinigung von N SBG-Graphen oder aus N nRSBT-Graphen zusammen. Jeder der N Knoten des Hypercube-Rechners ist Wurzel genau eines der N Teilgraphen. Die durchzuführende Strategie des Sendens ist die reverse-breadth-first Strategie (Kap. 5.8.2), d.h. alle Daten für die Knoten einer Ebene jedes der N Teilgraphen

werden von dessen Wurzel gleichzeitig gesendet. Begonnen wird mit der am weitesten entfernten Ebene, und nach n Schritten wird die Ebene — bestehend aus den zur Wurzel adjazenten Knoten — abgearbeitet.
Jeder Knoten des Hypercube-Rechners ist in jeder Knotenmenge der N SBG- bzw. nRSBT-Graphen enthalten und sendet die in jedem dieser Graphen erhaltenen Datenpakete an die entsprechenden Empfänger. Pakete, die über die gleiche Verbindung übertragen werden sollen, werden auch hier in einem Paket zusammengefaßt.
Wie bei n-Kanal Multibroadcast errechnet sich der gesamte Zeitbedarf T mit Hilfe der durch die Anwendung des Konzepts der dynamischen Paketgröße in Routing-Schritt i, $0 \leq i < n$, entstehenden Paketgröße B [Jo&Ho89]

$$B = B(i) = \frac{M}{n} \cdot \sum_{j=n-i}^{n} \binom{n}{j} = \frac{M}{n} \cdot \sum_{j=i+1}^{n} \binom{n}{j}.$$

Es ist

$$\begin{aligned} T &= \sum_{i=0}^{n-1} (B(i) \cdot t_{comm} + t_{start}) \\ &= \sum_{i=0}^{n-1} \frac{M}{n} \cdot \sum_{j=i+1}^{n} \binom{n}{j} \cdot t_{comm} + n \cdot t_{start} \\ &= \frac{M}{n} \cdot t_{comm} \cdot \sum_{i=1}^{n} i \cdot \binom{n}{i} + n \cdot t_{start} \\ &= \frac{M}{n} \cdot t_{comm} \cdot n \cdot 2^{n-1} + n \cdot t_{start} \\ &= \frac{N \cdot M}{2} \cdot t_{comm} + n \cdot t_{start}, \end{aligned}$$

und die maximale Paketgröße B_{max} wird im ersten Routing-Schritt benötigt und beträgt

$$\begin{aligned} B_{max} &= \frac{M}{n} \sum_{i=1}^{n} \binom{n}{i} \\ &= \frac{(N-1) \cdot M}{n}. \end{aligned}$$

Analog zu n-Kanal Multibroadcast wird auch hier zur Erreichung einer gleichmäßigen Verteilung der zu übertragenden Daten auf die n pro Wurzel zur Verfügung stehenden Verbindungen für die Vereinigung von nRSBT-Graphen das Konzept der vielfachen Wege (Kap. 5.2.3) herangezogen, d.h. die Datenmengen der Länge M werden in n Teilmengen der Länge $\lceil \frac{M}{n} \rceil$ geteilt und jede über einen anderen der spannenden n Teilbäume übertragen.

Falls $M < n$, werden nur M Teilbäume zur Übertragung der, in diesem Fall einelementigen, Mengen genutzt.
Für den SBG-Graphen gilt aufgrund der gleichen Überlegung wie in Kapitel 5.7.2, daß sich die zu übertragenden Daten genau auf die n Verbindungen der Wurzel aufteilen lassen [Jo&Ho89].
Als Abschluß der Vorstellung von Scatter-Algorithmen werden die wichtigsten diesbezüglichen Ergebnisse in Tabelle 5.8 zusammengefaßt. In der Spalte „Paketgröße" sind für diejenigen Algorithmen, deren Paketgrößen sich von Routing-Schritt zu Routing-Schritt ändern (Ein-Kanal- und n-Kanal Scatter sowie n-Kanal Total Exchange) die maximalen und für Algorithmen mit konstanter Paketgröße (Ein-Kanal Total Exchange) die optimalen Paketgrößen eingetragen.
Bei allen Scatteralgorithmen werden die angegebenen unteren Schranken für den Zeitbedarf der reinen Übertragunszeit für alle Größen der zu übertragenden Datenmenge erreicht. Bis auf n-Kanal Scatter besitzen die vorgestellten Algorithmen den geringsten Zeitbedarf aller in der angegebenen Literatur zu findenden Algorithmen.

untere Schranke	Graph	M	Paketgröße	Zeitdedarf
Ein-Kanal Scatter				
$\max\{(N-1)\cdot M\cdot t_{comm}, n\cdot t_{start}\}$	SBT	≥ 1	$\leq \frac{N\cdot M}{2}$	$(N-1)\cdot M\cdot t_{comm}+n\cdot t_{start}$
n-Kanal Scatter				
$\max\{\frac{(N-1)\cdot M}{n}\cdot t_{comm}, n\cdot t_{start}\}$	nRSBT	≥ 1	$\leq \sqrt{\frac{2}{\pi}}\cdot\frac{N\cdot M}{n^{3/2}}$	$\frac{(N-1)\cdot M}{n}\cdot t_{comm}+n\cdot t_{start}$
	SBG	≥ 1	$\leq \sqrt{\frac{2}{\pi}}\cdot\frac{N\cdot M}{n^{3/2}}$	$\frac{(N-1)\cdot M}{n}\cdot t_{comm}+n\cdot t_{start}$
Ein-Kanal Total Exchange				
$\max\{\frac{n\cdot N\cdot M}{2}\cdot t_{comm}, n\cdot t_{start}\}$	SBT	≥ 1	$\frac{N\cdot M}{2}$	$\frac{n\cdot N\cdot M}{2}\cdot t_{comm}+n\cdot t_{start}$
n-Kanal Total Exchange				
$\max\{\frac{N\cdot M}{2}\cdot t_{comm}, n\cdot t_{start}\}$	SBG	≥ 1	$\leq \frac{(N-1)\cdot M}{n}$	$\frac{N\cdot M}{2}\cdot t_{comm}+n\cdot t_{start}$
	nRSBT	≥ 1	$\leq \frac{(N-1)\cdot M}{n}$	$\frac{N\cdot M}{2}\cdot t_{comm}+n\cdot t_{start}$

Tabelle 5.8: **Zeitbedarf von Scatter/Total Exchange**

5.10 Vergleich der Transferzeiten für parallele Architekturen

Der Zeitbedarf von Datentransferalgorithmen in Hypercube-Rechnern soll abschließend mit den Laufzeiten von entsprechenden Algorithmen für die folgenden parallelen Architekturen verglichen werden:

Broadcast Bus : In diesem Modell seien N Prozessoren durch einen Bus miteinander verbunden, der in der Lage ist, Daten von einem zu allen anderen Prozessoren genau so schnell wie von einem zu nur einem anderen Prozessor zu übertragen. Alle Prozessoren können gleichzeitig über den Bus kommunizieren.

gemeinsamer Speicher : Hier sind N Prozessoren durch einem gemeinsamem Speicher verbunden, von dem alle gleichzeitig lesen oder auf den alle gleichzeitig schreiben können.

Ring : N Prozessoren sind nach Art eines Ringes verbunden und jeder Prozessor kann an beide Nachbarn gleichzeitig senden bzw. an einen Nachbarn senden und vom anderen Nachbarn empfangen.

zweidimensionale Gitter : Es wird eine zweidimensionale Gitterstruktur mit $\sqrt{N}$ Prozessoren pro Dimension als Verbindungsstruktur der Prozessoren angenommen. Jeder Prozessor kann gleichzeitig mit allen seinen Nachbarn kommunizieren.

geschalteter Rechner : In diesem Modell kann jeder der N Prozessoren mittels eines zentralen Schaltnetzes mit genau einem Prozessor verbunden werden. Nach dem Aufbau der Verbindungen können alle Prozessoren gleichzeitig mit ihrem direkt verbundenen Prozessor kommunizieren.

Die Laufzeiten für diese Algorithmen in Tabelle 5.9 und die Rechnermodelle stammen aus [Sa&Sc89b, Sa&Sc86, St&Wa90]. Es sei darauf hingewiesen, daß in diesen beiden Arbeiten keinerlei Anspruch auf Zeitoptimalität erhoben wird, also möglicherweise schnellere Algorithmen für einige oder gar alle Architekturen, ausgenommen der Hypercube-Struktur, existieren. Die angegebenen Laufzeiten für die Hypercube-Struktur hingegen sind die schnellsten der in den angegebenen Quellen beschriebenen Algorithmen bei n-Kanal Kommunikation.
Das benutzte Kommunikationsmodell entspricht im wesentlichen dem aus Kapitel 5.2.2. Für den Rechner mit gemeinsamem Speicher kann $1/t_{comm}$ als Bandbreite eines Prozessors angesehen werden. Die zum Transfer eines Pakets der Größe M von einem Prozessor zum gemeinsamen Speicher oder umgekehrt benötigte Zeit beträgt $M \cdot t_{comm} + t_{start}$. Für das Modell des geschalteten Rechners entspricht die Zeit t_{start} der Zeit, die das Schaltnetz

	Broadcast Bus	gemeinsamer Speicher
Unicast	$M \cdot t_{comm} + t_{start}$	$2 \cdot (M \cdot t_{comm} + t_{start})$
Broadcast	$M \cdot t_{comm} + t_{start}$	$2 \cdot (M \cdot t_{comm} + t_{start})$
Multi-broadcast	$N \cdot M \cdot t_{comm} + N \cdot t_{start}$	$N \cdot M \cdot t_{comm} + 2 \cdot t_{start}$
Scatter	$N \cdot M \cdot t_{comm} + N \cdot t_{start}$	$N \cdot M \cdot t_{comm} + t_{start}$
Total Exchange	$N \cdot M \cdot t_{comm} + N \cdot t_{start}$	$2 \cdot (N \cdot M \cdot t_{comm} + t_{start})$

	Ring	zweidimensionales Gitter
Unicast	$\left(\sqrt{\frac{M}{2} \cdot t_{comm}} + \sqrt{(i-1) \cdot t_{start}}\right)^2$	$\left(\sqrt{\frac{M}{4} \cdot t_{comm}} + \sqrt{(i-1) \cdot t_{start}}\right)^2$
Broadcast	$\left(\sqrt{M \cdot t_{comm}} + \sqrt{(\lfloor \frac{N}{2} \rfloor - 1) \cdot t_{start}}\right)^2$	$\left(\sqrt{\frac{M}{2} \cdot t_{comm}} + \sqrt{(\lceil \sqrt{N} \rceil - 1) \cdot t_{start}}\right)^2$
Multi-broadcast	$(N-1) \cdot M \cdot t_{comm} + (N-1) \cdot t_{start}$	$N \cdot M \cdot t_{comm} + 2 \cdot \sqrt{N} \cdot t_{start}$
Scatter	$\frac{N \cdot M}{2} \cdot t_{comm} + \lceil \frac{N}{2} \rceil \cdot t_{start}$	$N \cdot M \cdot t_{comm} + 2 \cdot \sqrt{N} \cdot t_{start}$
Total Exchange	$\frac{N \cdot M}{2} \cdot t_{comm} + N \cdot t_{start}$	$2 \cdot N \cdot M \cdot t_{comm} + 2 \cdot \sqrt{N} \cdot t_{start}$

	geschalteter Rechner	Hypercube-Rechner
Unicast	$M \cdot t_{comm} + t_{start}$	$\left(\sqrt{\frac{M}{n} \cdot t_{comm}} + \sqrt{(i-1) \cdot t_{start}}\right)^2$
Broadcast	$2 \cdot \left(\sqrt{M \cdot t_{comm}} + \sqrt{(\log_2 N - 2) \cdot t_{start}}\right)^2$	$\left(\sqrt{\frac{M}{n} \cdot t_{comm}} + \sqrt{(n-1) \cdot t_{start}}\right)^2$
Multi-broadcast	$N \cdot M \cdot t_{comm} + \log_2 N \cdot t_{start}$	$\frac{(N-1) \cdot M}{n} \cdot t_{comm} + n \cdot t_{start}$
Scatter	$N \cdot M \cdot t_{comm} + \log_2 N \cdot t_{start}$	$\frac{(N-1) \cdot M}{n} \cdot t_{comm} + \frac{n}{\log_2 n} \cdot t_{start}$
Total Exchange	$\log_2 N \cdot (M \cdot t_{comm} + 2 \cdot t_{start})$	$\frac{N \cdot M}{2} \cdot t_{comm} + n \cdot t_{start}$

Tabelle 5.9: Vergleich von Transferzeiten für verschiedene parallele Architekturen

zum Aufbau der N Verbindungen benötigt. Der Vergleichbarkeit halber werden für alle Architekturen die gleichen Werte für t_{start} und t_{comm} angenommen. Alle Ausdrücke sind für große zu übertragende Datenmengen M angegeben.

5.11 Kommunikation in fehlerhaften Hypercube-Rechnern

Bei der Untersuchung der Datentransferalgorithmen in den Kapiteln 5.2–5.9 wurde immer ein fehlerfreier Betrieb aller Verbindungsleitungen und Knoten des betrachteten Hypercube-Rechners vorausgesetzt. Es muß jedoch in der Praxis mit zunehmender Systemgröße auch mit einer steigenden Wahrscheinlichkeit für den Ausfall einer oder mehrerer Hardware-Komponenten gerechnet werden. In diesem Fall können Daten beim Transfer unter Beteiligung einer fehlerhaften Komponente verfälscht werden oder sogar ganz verloren gehen. Jedoch kann durch eine geringe Anzahl fehlerhafter Komponenten nicht das ganze System in Mitleidenschaft gezogen werden, da die einzelnen Knoten nur über spezielle Verbindungen kommunizieren können und, im Gegensatz zu Parallerechnern mit gemeinsamem Speicher, keinen direkten Zugriff auf Speicher haben, die von anderen Knoten benutzt werden.
Um laufende Berechnungen erfolgreich beenden zu können oder um eventuell sogar den weiteren Betrieb des Rechners bis zum Austausch der fehlerhaften Komponente zu gewährleisten, sollten Kommunikationsalgorithmen bereitgestellt werden, die im Fehlerfall nach der Identifikation der fehlerhaften Einheit für deren Umgehung und somit für den fehlerfreien Betrieb des Rechners sorgen [Go&St88b].
Falls die Aufrechterhaltung des Betriebs erfolgreich durchgeführt werden kann, muß dennoch für die Dauer des Defekts mit einem erheblichen Verlust an Leistungsfähigkeit gerechnet werden. Dies liegt an der meist speziell auf die Verbindungsstruktur und auf Gleichmäßigkeit ausgerichteten Lastverteilung, die durch die Umgehung fehlerhafter Komponenten sehr unausgeglichen werden kann. Zum einen muß sowohl bei Ausfall von Verbindungsleitungen als auch bei Ausfall eines Knotens mit längeren Wegen beim Datentransfer für diejenigen Wege gerechnet werden, die im Normalfall die nun fehlerhaften Komponenten benutzen. Zum anderen muß bei Ausfall eines Knotens die für ihn anfallende Arbeit von anderen, fehlerfreien Knoten übernommen werden. Letzteres erweist sich als schwierig, falls angefangene Berechnungen noch erfolgreich beendet werden sollen, denn es bleibt offen, wie die benötigten Daten, die im fehlerhaften Knoten gespeichert sind, dem fehlerfreien Restsystem zur Verfügung gestellt werden können.
Eine einfache Möglichkeit zur weiteren Nutzung des Rechners — allerdings unter Verzicht auf die Beendigung bereits angefangener Berechnungen — besteht im Ausweichen auf kleinere, fehlerfreie Konfigurationen des Hypercube-Rechners, d.h. Beschränkung auf die Nutzung einer Hypercube-Struktur kleinerer Dimension (Unterwürfel, Kap. 3.1.1) als die Dimension der gesamten Struktur. In diesem Fall muß im allgemeinen weder die Lastverteilung geändert noch ein größerer Overhead durch längere Kommunikationswege in Kauf genommen werden. Jedoch wird höchstens die Hälfte aller zur Verfügung stehenden Prozessoren genutzt, und es muß ein Algorithmus zur Anforderung und ggf. Freigabe

fehlerfreier Unterwürfel (siehe z.B. [Ka&Sh88]) implementiert werden.

Weitere Maßnahmen zum durchgehenden Betrieb des Rechners im Fall von Knotenausfällen sind algorithmenbasierte Fehlererkennungsmethoden (*algorithm-based fault detection*, [Bane&&90]). Hierbei können während der Berechnung durch algorithmenspezifische Maßnahmen, z.B. geeignete Kodierungen, Knotenfehler erkannt und der Algorithmus entsprechend umgestellt werden, so daß die Benutzung des fehlerhaften Knotens vermieden wird. In [Bane&&90] werden entsprechende Methoden für die Matrixmultiplikation, die Gaußsche Elimination und die schnelle Fourier-Transformation vorgestellt. Literaturhinweise auf weitere Anwendungen und theoretische Betrachtungen zu algorithmenbasierter Fehlererkennung finden sich in [Ba&St88].

Der Ausfall eines Knotens muß aufgrund oben genannter Gründe als ein schwerwiegender Fehler angesehen werden, während bei Ausfall einer Verbindungsleitung noch recht gute Aussichten für die erfolgreiche Beendigung laufender Berechnungen und für den weiteren Betrieb bestehen. Dies wird anschaulich durch die mögliche Betrachtung eines Knotenausfalls als einen Ausfall aller mit diesem Knoten verbundenen Verbindungsleitungen unterstützt. Weiterhin müssen schlimmstenfalls n Verbindungsleitungen ausfallen (Fehlertoleranz, Kap. 3.1.6) — nämlich gerade alle mit einem bestimmten Knoten verbundenen — um dessen Isolierung, d.h. Ausfall und damit auch den Defekt aller ihn enthaltenden Unterwürfel nach sich zu ziehen. Der Ausfall eines Knotens hingegen schließt dessen weitere Benutzung von vorneherein aus. Weitergehende Betrachtungen über den Zusammenhang zwischen der Anzahl fehlerhafter Komponenten eines Hypercube-Rechners und der Existenz fehlerfreier Unterwürfel finden sich in [Be&Si88].

H.P. Katseff beschäftigt sich in [Kats86, Kats88] mit unvollständigen Hypercube-Rechnern (*incomplete hypercubes*). Ein unvollständiger Hypercube-Rechner besitzt eine Knotenzahl, die nicht einer Zweierpotenz entspricht, wobei die Knoten mit den höchsten Adressen fehlen. Somit kann diese Klasse von Rechnern als Spezialfall fehlerhafter Hypercube-Rechner aufgefaßt werden.

Abbildung 5.26 zeigt einen unvollständigen Hypercube-Rechner mit sieben Knoten, der einem Hypercube-Rechner entspricht, dessen Knoten mit Adresse 111 ausgefallen ist.

Laut [Kats88] sollen unvollständige Hypercube-Rechner die schrittweise Erweiterung eines bestehenden Systems erleichtern, denn statt einer Verdoppelung der Knotenzahl kann nun mit beliebiger Knotenzahl erweitert werden. Zur Unterstützung dieser Behauptung werden Algorithmen für Routing und Broadcast angegeben, die ebenso einfach sind wie Algorithmen für herkömmliche Hypercube-Rechner. Aufgrund der i.a. entstehenden Asymmetrie der Verbindungsstruktur muß jedoch davon ausgegangen werden, daß eine gleichmäßige Lastverteilung für weitverbreitete Probleme mit symmetrischer Struktur (z.B. Matrixalgorithmen) unter Benutzung aller Knoten nur sehr schwierig zu erreichen ist. Dies muß als schwerwiegender Nachteil angesehen werden, der durch den Vorteil einer leichteren Erweiterbarkeit des Rechners kaum ausgeglichen werden kann.

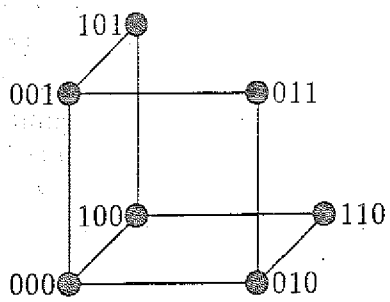


Abb. 5.26: Unvollständiger Hypercube-Rechner mit sieben Knoten

In den nun folgenden Abschnitten werden die in der angegebenen Literatur zu findenden Methoden zum fehlertoleranten Datentransfer in Hypercube-Rechnern überblicksweise angegeben, aufgrund der Vielfalt der Algorithmen jedoch nur teilweise näher erläutert. Das Hauptaugenmerk liegt in allen Artikeln auf dem Problem des Findens eines möglichst kurzen Weges zwischen zwei fehlerfreien Knoten bzw. auf dem Problem der Feststellung, daß kein derartiger Weg existiert. Auf das Problem des möglichst schnellen Datentransfers durch Benutzung von Konzepten wie vielfache Wege, Pipelining oder dynamische Paketgröße (Kap. 5.2.3) sowie auf eingehendere Untersuchungen der benötigten Transferzeit, z.B. durch Unterscheidung von Startzeit t_{start} und reiner Kommunikationszeit t_{comm} (Kap. 5.2.2) wird in keiner der angegebenen Arbeiten eingegangen. Als Maß für die Leistungsfähigkeit eines Algorithmus wird vielmehr die zu erwartende Länge eines durch ihn entstehenden Weges herangezogen, d.h. es wird ein Modell wie in [Lan&&90, Bert&&89] benutzt, welches die Länge einer Nachricht unberücksichtigt läßt.

5.11.1 Fehlertoleranz durch zusätzliche Hardware

Eine Möglichkeit, den Betrieb des Rechners im Fehlerfall aufrechtzuerhalten, besteht in der Verwendung von zusätzlicher Hardware, die im Fehlerfall die ausgefallene Komponente vollständig ersetzt. Dies läßt sich für den Fall eines Knotenausfalls durch die Bereitstellung eines oder mehrerer Ersatzknoten (*spare nodes*) für je einen Unterwürfel konstanter Größe realisieren. In [Al&Me89] werden sogenannte fehlertolerante Unterwürfel (*fault tolerant basic block*, FTBB) aufgebaut, die für je einen dreidimensionalen Unterwürfel des Hypercube-Rechners vier Ersatzknoten vorsehen, die in einer zweidimensionalen Hypercube-Struktur miteinander verbunden sind. Die Knoten sind mit den Ersatzknoten so verbunden, daß je vier Knoten mit einem Ersatzknoten und je zwei Ersatzknoten mit einem Knoten verbunden sind. Anschaulich kann dies durch die Verbindung jedes der vier Ersatzknoten mit allen

Knoten jeweils verschiedener Seiten eines Würfels dargestellt werden (Abb. 5.27). Durchgezogene Linien repräsentieren die Verbindungen des Unterwürfels, gestrichelte Linien die zusätzlichen Verbindungen von Knoten zu Ersatzknoten und gepunktete Linien die Verbindungen der zweidimensionalen Hypercube-Struktur der Ersatzknoten. Ein Ersatzknoten kann alle Knoten ersetzen, deren Adressen durch Ersetzen der „X" in der Bezeichnung des Ersatzknotens entstehen.

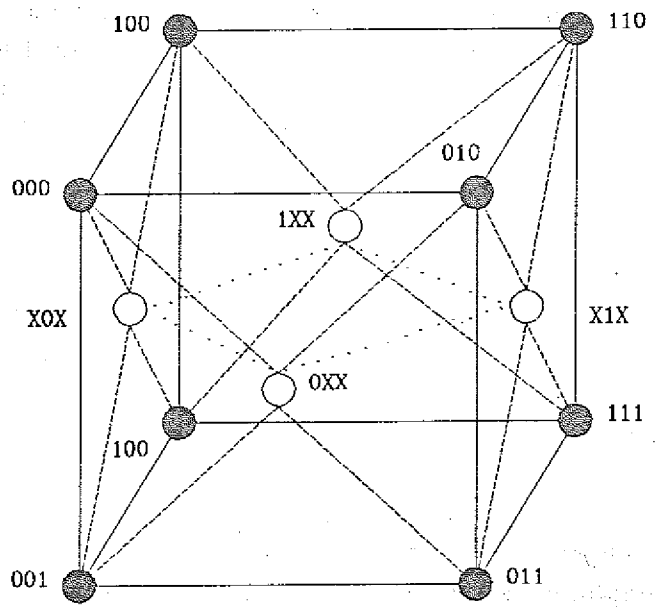


Abb. 5.27: Fehlertoleranter Unterwürfel (FTBB)

Fällt ein Knoten eines FTBB aus, so nimmt ein freier Ersatzknoten dessen Adresse an und führt dessen Berechnungen weiter, sofern dies ohne Kenntniss der durch den Fehler verlorengegangenen Daten möglich ist. Sind beide Ersatzknoten des ausgefallenen Knoten schon beschäftigt, so führt dies zu einem fehlerhaften FTBB.
Neben der Bestimmung, welcher der beiden Ersatzknoten die Funktionen eines fehlerhaften Knotens übernimmt, muß noch ein Routing-Algorithmus bereitgestellt werden, der alle für den fehlerhaften Knoten bestimmte Nachrichten an dessen Ersatzknoten weiterleitet.
Der Betrieb eines Hypercube-Rechners kann hierdurch theoretisch durch eine konstante Anzahl aktiver Prozessoren ohne Leistungsverluste aufrechterhalten werden. Dies ist jedoch nur bis zu einer bestimmten Maximalzahl von Ausfällen möglich, und es bleibt neben dem Problem der Wiederherstellung verlorener Daten das Problem eines aufwendigeren Routings. Eventuell müssen auch hier längere Wege in Kauf genommen werden und nicht zuletzt muß zusätzliche Hardware im Umfang der halben Knotenzahl des ursprünglichen Rechners beschafft werden, die nur im Fehlerfall genutzt wird.

5.11.2 Fehlertoleranz ohne zusätzlichen Hardware-Aufwand

Anstatt zusätzliche Hardware zu beschaffen, die nur im Fehlerfall aktiviert wird und daher im allgemeinen nur eine geringe Auslastung aufweist, läßt sich fehlertolerantes Routing auch durch geeignete Maßnahmen im Software-Bereich erzielen.
Ein solches Konzept ist das vielfache Senden von Nachrichten auf kantendisjunkten Wegen (*sending multiple copies through disjoint paths*) [Ra&Sh88]. Kopien der zu übertragenden Nachricht können hierbei auf bis zu n kantendisjunkten Wegen gleichzeitig übertragen werden (Kap. 3.1.5). Falls der Rechner fehlerfrei arbeitet, erhält der Empfänger schließlich alle gesendeten, identischen Nachrichten. Anderenfalls erhält der Empfänger möglicherweise keine oder nur wenige der gesendeten Nachrichten. Je nach zugrundegelegtem Fehlermodell muß der Empfänger nun geeignete Maßnahmen ergreifen [Ra&Sh88, Lamp&&82]:

einfache Fehler : (*simple omission faults*)
Falls man davon ausgehen kann, daß nur einfache Fehler auftreten können, d.h. daß ein Knoten entweder eine empfangene Nachricht korrekt weiterleitet oder diese gar nicht sendet, ist die Entscheidung für den Empfänger der Nachricht einfach. Falls er überhaupt eine Nachricht erhält, ist sie in genau dieser Form auch von der Quelle gesendet worden, und alle weiteren eintreffenden Kopien dieser Nachricht sind identisch.

schwere Fehler : (*Byzantine-, non-Byzantine faults*)
Unter der Annahme, daß durch auftretende Fehler Nachrichten verändert, umgeleitet oder ausgelassen werden können, muß damit gerechnet werden, daß der Empfänger Nachrichten verschiedenen Inhalts erhält. Der Entscheidungsmechanismus, welche der erhaltenen Nachrichten als die richtige zu akzeptieren ist, ist in diesem Fall nicht einfach [Lamp&&82]. Eine Möglichkeit besteht darin, diejenige Nachricht, die am häufigsten empfangen wurde, als die richtige zu akzeptieren (Mehrheitsentscheid, *majority voting*), wobei eine bestimmte Mindestzahl identischer Kopien eintreffen muß, um mit hoher Wahrscheinlichkeit entscheiden zu können, daß diese die unverfälschte Nachricht ist.

Ein Vorteil dieser Methode liegt darin, daß keine Identifikation von fehlerhaften Komponenten notwendig ist und somit, zumindest hierfür, kein Overhead anfällt. Als Nachteil hingegen muß jedoch sowohl das stark ansteigende Verkehrsaufkommen als auch ein aufwendiges Akzeptanzverfahren angeführt werden, welches im wesentlichem aus dem Vergleich aller eintreffenden Kopien aller Nachrichten besteht.
Zur Vermeidung dieser Nachteile bietet sich an, daß jeder Knoten Informationen über den Zustand von Teilen des Netzwerks sammelt und mit deren Hilfe versucht, die Benutzung fehlerhafter Knoten oder Verbindungen zu vermeiden. Es ist dabei ausreichend zu wissen,

ob eine Verbindung oder ein Knoten fehlerhaft ist oder nicht. Vorschläge über die Menge der zu sammelnden Netzwerkinformationen reichen vom Sammeln von Informationen nur über die unmittelbaren Nachbarn [Le&Ha88, Ch&Sh88, Ch&Sh90, Go&St88a, Go&St88b, Blou&&89] bis hin zur vollständigen Information jedes Knotens über den gesamten Rechner [Le&Ha88]. Letzteres ist nur mit großem Aufwand realisierbar, kann jedoch ohne weiteres zum Finden des kürzesten noch verbleibenden Weges zwischen zwei fehlerfreien Knoten genutzt werden. Mit abnehmender Informationsmenge über das Netzwerk wird es für jeden Knoten natürlich zunehmend schwieriger oder gar unmöglich zu entscheiden, welches die beste Routing-Entscheidung ist.

Für den Fall des Sammelns von Informationen über die Nachbarn wird häufig von der Methode des *Sidetracking* Gebrauch gemacht [Go&St88b, Blou&&89, Ch&Sh88, Ch&Sh90]. Es wird hierbei zunächst versucht, eine fehlerfreie Verbindung zu einem fehlerfreien Knoten zu benutzen, so daß die Entfernung zum Empfänger verringert wird. Die Auswahl einer solchen Verbindung kann sowohl durch zufällige Entscheidung als auch z.B. durch Wahl der kleinsten oder größten in Frage kommenden Dimension erfolgen. Falls nun keine derartige Verbindung existiert, benutzt man nacheinander alle anderen Verbindungen, über die noch nicht besuchte Knoten erreicht werden können, d.h. es wird ein Umweg in Kauf genommen. Falls auch keine derartigen Verbindungen mehr existieren, wird die Nachricht wieder an denjenigen Knoten zurückgesendet, von dem sie zuerst empfangen wurde (Backtracking). Ist der ausführende Knoten, der die Nachricht zurückschickt, der ursprüngliche Sender, so sind Sender und Empfänger nicht verbunden. Zur Bestimmung, welcher Knoten bereits besucht wurde und welcher nicht, muß mit jeder Nachricht zusätzlich zur Adresse des Senders auch eine Liste der Adressen der bereits besuchten Knoten oder der bis jetzt benutzten Dimensionen mitgeschickt werden.

Das Struktogramm aus Abbildung 5.28 verdeutlicht genauer diese mit Tiefensuche (*depth-first search*) bezeichnete Vorgehensweise in einem n-dimensionalen Hypercube-Rechner [Ch&Sh90]. e_i sei gleich der binären Zahl mit Wert 2^i, d.h. diejenige binäre Zahl mit einer eins an der Stelle i und null sonst. $\oplus$ bezeichne die bitweise EXKLUSIV-ODER Operation (Kap. 5.1). Der Algorithmus beginnt in jedem Knoten nach Empfang aller gesendeten Daten.

Betrachtet man beispielsweise die Darstellung eines fehlerhaften Hypercube-Rechners in Abbildung 5.29 auf Seite 129 und wendet obigen Algorithmus auf das Problem des Routings zwischen den Knoten 0001 und 1101 an, so werden die Zwischenknoten 1001, 1011 und 1111 durch Wahl der kleinsten Dimension, deren Benutzung die Entfernung zum Empfänger verkürzt, in den Knoten 0001 und 1011 gewählt. In Knoten 1001 hingegen muß ein Umweg in Kauf genommen werden, da die einzige Verbindung, die einen kürzesten Weg garantiert, fehlerhaft ist. Die Reihenfolge der Benutzung der Verbindungen ist mit arabischen Ziffern gekennzeichnet. Es ist ersichtlich, daß Backtracking in diesem Fall nicht benötigt wird. Dies wäre jedoch der Fall, falls auch die Verbindung von 1001 nach 1011 fehlerhaft wäre. Der in

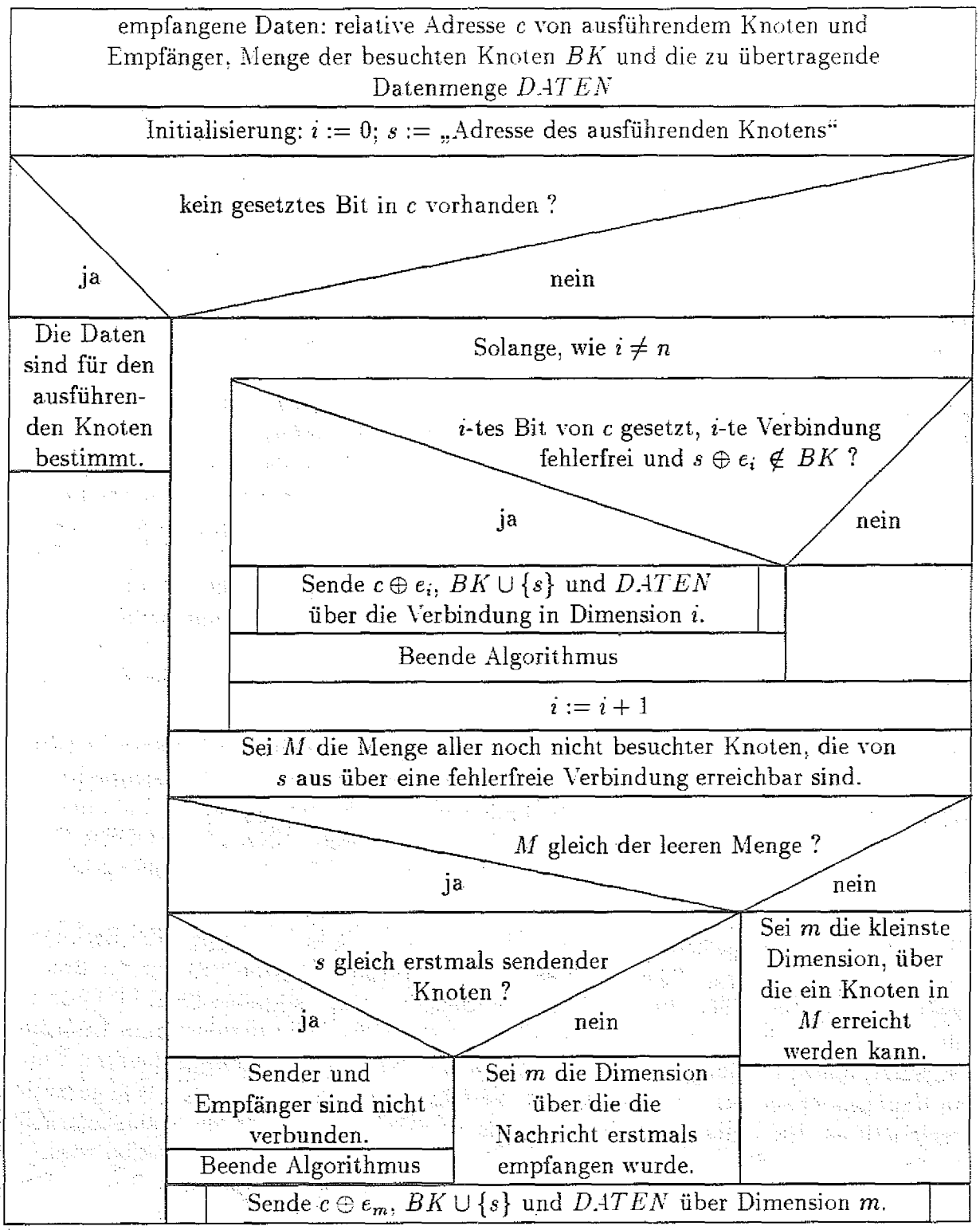


Abb. 5.28: Routing-Algorithmus mit Tiefensuche für einen fehlerhaften Hypercube-Rechner

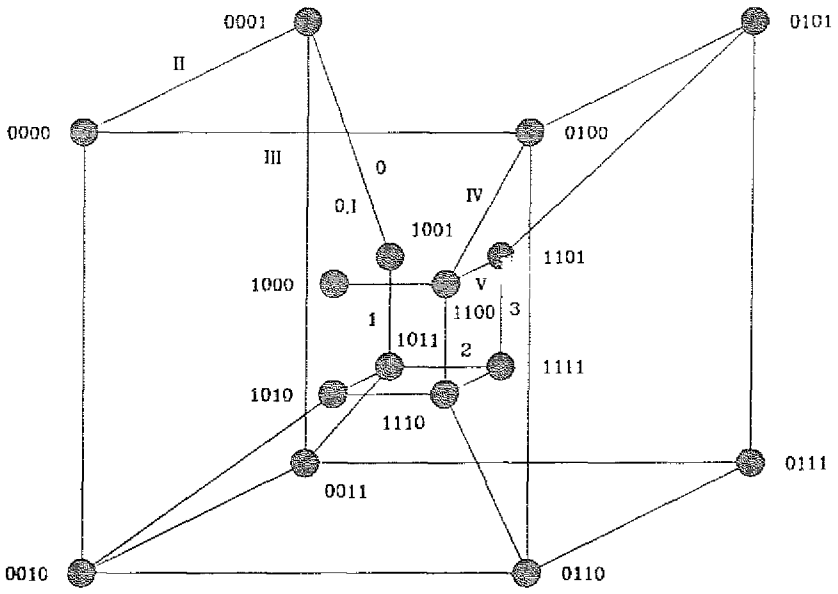


Abb. 5.29: Vierdimensionaler, fehlerhafter Hypercube-Rechner

diesem Fall resultierende Weg ist der mit römischen Ziffern gekennzeichnete. Man erkennt nun, daß bei Knoten 1001 aufgrund fehlender fehlerfreier Verbindungen ein Backtracking erforderlich wird.
Der beschriebene Algorithmus findet in jedem Fall einen Weg zwischen zwei fehlerfreien Knoten, falls ein solcher existiert, denn durch Backtracking werden alle in Frage kommenden Wege getestet, bis man das Ziel erreicht oder feststellt, daß kein Weg dorthin existiert. Der zurückgelegte Weg kann also wesentlich länger werden als der Hamming-Abstand zwischen Sender und Empfänger. Falls die Anzahl fehlerhafter Knoten gegenüber der Gesamtzahl relativ gering ist, findet der Algorithmus jedoch mit großer Wahrscheinlichkeit einen kürzesten Weg [Ch&Sh90].
Neben fehlertolerantem Routing besteht eine weitere Einsatzmöglichkeit von Sidetracking-Algorithmen in der Umgehung vielbeschäftigter Knoten [Go&St88a]. Denkbar wäre dabei die bevorzugte Benutzung von Verbindungen, die zu wenig beschäftigten Knoten führen. Eventuell auftretende längere Wege führen in diesem Fall aufgrund kürzerer Wartezeiten nicht unbedingt zu einer längerern Transferzeit.

6 Zusammenfassung und Ausblick

Ziel dieser Arbeit war die Auswertung der verfügbaren Literatur über Hypercube-Rechner mit Schwerpunkten auf den Eigenschaften der Hypercube-Struktur, dem Aufbau von Hypercube-Rechnern und der Interprozessorkommunikation. Es hat sich gezeigt, daß die Hypercube-Struktur eine Vielzahl gut untersuchter Eigenschaften besitzt, die sowohl die einfache Abbildung vieler Problemstrukturen als auch effizienten Datentransfer zwischen den Knoten des Rechners erlaubt. Zusätzlich können die Verbindungsstrukturen der meisten zur Zeit angebotenen Multicomputer auf einfache Weise ohne zusätzlichen Kommunikationsaufwand simuliert werden.
Zwei verschiedene Entwicklungsrichtungen zeigen sich beim Aufbau von Hypercube-Rechnern. Zum einen werden zunehmend leistungsfähigere Knoten verwendet (iPSC/860) und in Systemen mittlerer Knotenzahl benutzt, die die Lösung von Problemen großer Granularität auf dem Gesamtsystem erlauben. Zum anderen wird versucht, durch Bereitstellung einfachster Prozessoren in großer Zahl (Connection Machine) die Lösung von Problemen feiner Granularität effizient zu unterstützen. In beiden Fällen wird die Bearbeitung zunehmend größerer Probleme in angemessener Zeit ermöglicht.
Da der auf jedem Multiprozessorsystem erzielbare Speedup wesentlich von der Effizienz der Interprozessorkommunikation abhängt [Ch&Sh88]), wurde in dieser Arbeit besonderes Gewicht auf die Vorstellung und Untersuchung von Datentransferalgorithmen gelegt. Die behandelten Kommunikationsarten können auf Hypercube-Rechnern verteilt implementiert werden, wobei in den meisten Fällen die angegebenen unteren Schranken für den Zeitbedarf erreicht werden. Dies läßt auf eine sehr gute Nutzung der leistungsfähigen Verbindungsstruktur des Hypercube-Rechners schließen.
Die Unterstützung von Kommunikationsarten durch existierende Betriebssysteme für Hypercube-Rechner geht jedoch nur in wenigen Fällen über die Bereitstellung von Routinen für Unicast hinaus (siehe einige Beispiele in Tabelle 6.1). Aus dem Rahmen fallen in diesem Zusammenhang das Betriebssystem CrOS III (*Crystalline Operating System*), welches neben Routinen für Broadcast, Scatter und Unicast noch weitere speziellere Routinen zur Verfügung stellt [Angu&&90], und *Topologies*, eine Betriebssystemerweiterung, die dem Benutzer die Definition eigener Kommunikationsgraphen ermöglicht [Sc&Bo88]. Auf Hardware-Ebene wird lediglich durch die Intel Hypercube-Rechner iPSC/2 und iPSC/860 Unicast unterstützt (DCM, Kap. 4.3.1).
In Zusammenhang mit Datentransferalgorithmen in Hypercube-Rechnern wäre auch eine — in dieser Arbeit unberücksichtigt gebliebene — Untersuchung der Vielzahl von Metho-

CrOS III	NX/2	SIMPLEX	MOOSE	Trillium
Unicast Broadcast Scatter verschiedenes	Unicast	Unicast	Unicast	Unicast Broadcast
Hypercube allg.	iPSC/2	NCUBE	iPSC/1 NCUBE	FPS - T-Serie
[Angu&&90]	[Pier88]	[Krum88]	[Salm&&88]	[Burn&&88]

Tabelle 6.1: **Hypercube-Betriebssysteme**

den zur Lastverteilung interessant, denn durch sie werden im wesentlichen die benötigten Kommunikationsarten vorgegeben. Literaturhinweise hierzu finden sich in Anhang A.4.

Ohne Zweifel besteht heute und in Zukunft im wissenschaftlich-technischen Bereich ein großer Bedarf an Rechnern, deren Leistungsfähigkeit um ein Vielfaches höher liegt als die heutiger Supercomputer. Leistungssteigerungen um einige Größenordnungen sind jedoch vermutlich nur durch den Einsatz massiv paralleler Systeme mit großer Knotenzahl zu erzielen. Allerdings muß betont werden, daß es noch viele Schwierigkeiten bei der Nutzung derartiger Systeme gibt. Beispiele hierfür sind die unzureichende Software-Ausstattung sowohl im Anwendungs- als auch im Entwicklungsbereich. Weiterhin ist die Portierung der bestehenden Programme auf MIMD-Rechner keine einfache Aufgabe. Das automatische Erkennen und Ausnutzen vorhandener Parallelität in Programmen und deren effizienteste Ausnutzung wirft weitere Schwierigkeiten auf, für die momentan noch keine befriedigenden Lösungen gefunden wurden. Die benötigte mathematische Theorie zur Nutzung massiv paralleler Rechner ist insgesamt alles andere als gut erschlossen [Tv&Na89]. Auch hat der Mensch selbst meist Probleme, sich in parallelen Denkweisen zurechtzufinden.

Das noch nicht abzusehende Ende in der Entwicklung auf dem Gebiet der Mikroprozessoren hingegen schafft die technischen Vorraussetzungen für den Bau zukünftiger massiv paralleler Rechner mit sehr hohen Spitzenleistungen. Die dringend notwendige Entwicklung von Anwendungs- und Entwicklungsprogrammen ist Gegenstand intensiver Forschungen in vielen verschiedenen Projekten. Eine Auswahl hierzu findet sich z.B. in [Heat85a, Heat86, Fox88, Angu&&90]. Vom Erfolg dieser Projekte wird es abhängen, ob die Ausnutzung der Besonderheiten massiv paralleller Rechner in Zukunft, ähnlich wie dies für heutige Vektorrechner der Fall ist, automatisiert und vereinfacht werden kann. Damit würde in vielen Fällen überhaupt erst eine sinnvolle Nutzung mit vertretbarem Arbeitsaufwand möglich.

Neben großer Spitzenleistung ist das günstige Preis-Leistungsverhältnis, welches in der

Hauptsache durch Verwendung preisgünstiger Mikroelektronik erreicht wird, ein weiterer Vorteil massiv paralleler Systeme. Dieses Verhältnis liegt z.B. bei Hypercube-Rechnern um etwa eine Größenordnung niedriger als bei heutigen Vektor-Supercomputern, d.h. die Kosten pro erzielbaren MFLOPS liegen für Hypercube-Rechner bei einem Zehntel der entsprechenden Kosten für einen Vektor-Supercomputer. Dadurch werden Hypercube-Rechner und auch sonstige massiv parallele Systeme mit ähnlich günstigen Kosten für kleinere Firmen und Institutionen finanzierbar. Bereits Mitte 1989 waren von 1288 weltweit installierten Supercomputern 288 parallele Systeme mit mindestens 64 Prozessoren [Arch90], wovon wiederum der größte Anteil auf Hypercube-Rechner entfiel. Jedoch kann heute noch niemand absehen, ob die Hypercube-Struktur als Verbindungsstruktur massiv paralleler Systeme weiterhin diese dominierende Rolle einnimmt oder ob sie von einer anderen aus der Vielzahl möglicher Verbindungsstrukturen verdrängt wird.

Literaturverzeichnis

[Alma85] Almasi G.S.
Overview of Parallel Processing
Parallel Computing **2**, 191-203, 1985

[Al&Me89] Alam M.S., Melhem R.
Fault Tolerance and Reliable Routing in Augmented Hypercube Architectures
Proceedings of the 8th Annual International Phoenix Conference on Computers and Communications, 19-23, March 22-24, 1989, Scottsdale/AZ, USA

[Amda67] Amdahl G.M.
Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities
AFIPS Conference Proceedings **30**, 483-485, 1967

[Angu&&90] Angus I.G., Fox G.C., Kim J.S., Walker D.W.
Solving Problems on Concurrent Processors Vol II:
Software for Concurrent Processors
Prentice Hall, Englewood Cliffs, 1990

[Arch90] Architecture Technology Corporation, Minneapolis/MN, USA
Supercomputers
Elsevier Advanced Technology, Oxford/UK, January 1990

[Bane&&90] Banerjee P., Rahmeh J.T., Stunkel C., Nair V.S., Roy K., Balasubramanian V., Abraham J.A.
Algorithm-Based Fault Tolerance on a Hypercube Multiprocessor
IEEE Transactions on Computers **C-39**, 1132-1145, September 1990

[Ba&St88] Banerjee P., Stunkel C.B.
A Novel Approach to System-Level Fault Tolerance in Hypercube Multiprocessors
Proceedings of the 3rd Conference on Hypercube Concurrent Computers

and Applications, 307-311, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Be&Si88] Becker B., Simon H.U.
How Robust is the n-Cube?
Information and Computation **77**, 161-178, 1988

[Bern89] Berntsen J.
Communication Efficient Matrix Multiplication on Hypercubes
Parallel Computing **12**, 335-342, 1989

[Berr&&90] Berrendorf R., Docter J., Gutbrod F.
Performance-Messungen auf dem iPSC/860
Gemeinsames HLRZ-ZAM Seminar, Forschungszentrum Jülich, 6.10.1990

[Bert&&89] Bertsekas D.P., Ozveren C., Stamoulis G.D., Tseng P., Tsitsiklis J.N.
Optimal Communication Algorithms for Hypercubes
Report LIDS-P-1847, Laboratory for Information and Decision Systems, M.I.T., Cambridge/MA, USA, January 1989

[Bhat&&86] Bhatt S.N., Chung F.R.K., Leighton F.T., Rosenberg A.L.
Optimal Simulations of Tree Machines
Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 274-282, October 27-29, 1986, Toronto/Ontario, Canada

[Bh&Ip85] Bhatt S.N., Ipsen I.C.F.
How to Embed Trees in Hypercubes
Research Report YALEU/DCS/RR-443, Yale University, Department of Computer Science, December 1985

[Bh&Ag84] Bhuyan L.N., Agrawal D.P.
Generalized Hypercube and Hyperbus Structures for a Computer Network
IEEE Transactions on Computers **C-33**, 323-333, 1984

[Bi&Lo89] Bier T., Loe K.F.
Embedding of Binary Trees into Hypercubes
Journal of Parallel and Distributed Computing **6**, 679-691, 1989

[Blou&&89] Blough D.M., Bagherzadeh N., Sehgal R.
A New Fault-Tolerant Routing Algorithm for Hypercube Systems
Proceedings of the 3rd Annual Parallel Processing Symposium, 130-137, March 29-31, 1989, California State University, edited by L.H. Carter

[Bo&Ro89] Bomans L., Roose D.
Benchmarking the iPSC/2 Hypercube Multiprocessor
Concurrency: Practice and Experience **1**, 3-18, September 1989

[Bouk&&72] Bouknight W.J., Denenberg S.A., McIntyre D.E., Randall J.M., Sameh A.H., Slotnick D.L.
The Illiac IV System
Proceedings of the IEEE **60**, 369-379, April 1972

[Burn&&88] Burns G.D., Pfiffer A.K., Fielding D.L., Brown A.A.
Trillium Operating System
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 374-376, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ch&Sh88] Chen M.S., Shin K.G.
Message Routing in an Injured Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 312-317, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ch&Sh90] Chen M.S., Shin K.G.
Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers
IEEE Transactions on Parallel and Distributed Systems **1**, 152-159, April 1990

[Choi&&87] Choi Y., Esfahanian A.H., Ni L.M.
One-to-k Communication in Distributed-Memory Multiprocessors
Proceedings of the 25th Annual Allerton Conference on Communication, Control and Computing, 268-269, September 30 - October 2, 1987, Monticello/IL, USA

[Clos88] Close P.
The iPSC/2 Node Architecture
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 43-50, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Dall88] Dally W.J.
Finite-Grain Message Passing Concurrent Computers
Proceedings of the 3rd Conference on Hypercube Concurrent Computers

and Applications, 2-12, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Da&Se87] Dally W.J., Seitz C.L.
Deadlock-Free Message Routing in Multiprocessor Interconnection Networks
IEEE Transactions on Computers **C-36**, 547-553, May 1987

[Do&Du89] Dongarra J.J, Duff I.S.
Advanced Architecture Computers
University of Texas, Computer Science Department, Knoxville/TN, USA, TN 37996-1301, November 1989

[Do&Ja87] Dowd P.W., Jabbour K.
Performance Evaluation of Spanning Multiaccess Channel Hypercube Interconnection Network
IEE Proceedings **134**, 295-302, 1987

[Dunc90] Duncan R.
A Survey of Parallel Computer Architectures
Computer **23**, 5-16, February 1990

[Duni86] Dunigan T.H.
Hypercube Performance
Proceedings of the 2nd Conference on Hypercube Multiprocessors, 178-192, September 29 - October 1, 1986, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1987

[Efe&&88] Efe K., Blackwell P., Shiau T., Slough W.
A Reduced Diameter Interconnection Network
Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation, 471-474, October 10-12, 1988, Fairfax/VA, USA

[El&La90] El-Amawy A., Latifi S.
Bridged Hypercube Networks
Journal of Parallel and Distributed Computing **10**, 90-95, 1990

[Ensl77] Enslow P.H.
Multiprocessor Organisation - A Survey
Computing Surveys **9**, 102-129, March 1977

[Feng81] Feng T.Y.
A Survey of Interconnection Networks
Computer **14**, 12-27, December 1981

[Flyn66] Flynn M.J.
Very High-Speed Computing Systems
Proceedings of the IEEE **54**, 1901-1909, December 1966

[Flyn72] Flynn M.J.
Some Computer Organizations and Their Effectiveness
IEEE Transactions on Computers **C-21**, 948-960, September 1972

[Fox88] Fox G.C. (editor)
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications I and II, January 19-20, 1988, Pasadena/CA, USA, ACM 1988

[Fox&&87] Fox G.C., Otto S.W., Hey A.J.G.
Matrix Algorithms on a Hypercube I: Matrix Multiplication
Parallel Computing **4**, 17-31, 1987

[Fox&&88] Fox G.C., Johnson M., Lyzenga G., Otto S., Salman J., Walker D.
Solving Problems on Concurrent Processors Vol I:
General Techniques and Regular Problems
Prentice Hall, Englewood Cliffs, 1988

[Fo&Fu86] Fox G.C., Furmanski W.
Communication Algorithms for Regular Convolution and Matrix Problems on the Hypercube
Proceedings of the 2nd Conference on Hypercube Multiprocessors, 223-238, September 29 - October 1, 1986, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1987

[Fo&Fu88a] Fox G.C., Furmanski W.
Optimal Communication Algorithms for Regular Decompositions on the Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 648-713, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Fo&Fu88b] Fox G.C., Furmanski W.
Hypercube Algorithms for Neural Network Simulations

Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 714-724, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Frai89] Fraigniaud P.
Performance Analysis of Broadcasting in Hypercubes
Proceedings of the 1st European Workshop on Hypercube and Distributed Computers, 311-327, October 4-6, 1989, Rennes/France, edited by F. André and J.P. Verjus, North-Holland 1989

[Frai90] Fraigniaud P.
Complexity Analysis of Broadcasting in Hypercubes with Restricted Communication Capabilities
Rapport LIP-IMAG 90-16, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Mai 1990

[Fren86] Frenkel K.A.
Evaluating Two Massively Parallel Machines
Communications of the ACM **29**, 752-758, August 1986

[Gehr&&88] Gehringer E.F., Abullarade J., Gulyn M.H.
A Survey of Commercial Parallel Processors
Computer Architecture News **16**, 75-107, September 1988

[Gilb58] Gilbert E.N.
Gray Codes and Paths on the n-Cube
The Bell System Technical Journal **37**, May 1958

[Go&Mr89] Gonauser M., Mrva M.
Multiprozessorsysteme
Springer–Verlag Berlin Heidelberg, 1989

[Go&Se81] Goodman J.R., Sequin C.H.
Hypertree: A Multiprocessor Interconnection Topology
IEEE Transactions on Computers **C-30**, 923-933, December 1981

[Go&St88a] Gordon J.M., Stout Q.F.
Hypercube Message Routing in the Presence of Faults
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 318-327, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Go&St88b] Gordon J.M., Stout Q.F.
Fault Tolerant Message Routing on Large Parallel Systems
Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation, 155-157, October 10-12, 1988, Fairfax/VA, USA

[Gr&Re86] Grunwald D.C., Reed D.A.
Benchmarking Hypercube Hardware and Software
Report No. UIUCDCS-R-86-1303, Department of Computer Science, University of Illinois at Urbana-Champaign, November 1986

[Gr&Re89] Grunwald D.C., Reed D.A.
Analysis of Backtracking Routing in Binary Hypercube Computers
Report No. UIUCDCS-R-89-1486, Department of Computer Science, University of Illinois at Urbana-Champaign, February 1989

[Gust&&86] Gustafson J.L., Hawkinson S., Scott K.
The Architecture of a Homogeneous Vector Supercomputer
Journal of Parallel and Distributed Computing **3**, 297-304, 1986

[Hara&&88] Harary F., Hayes J.P., Wu H.J.
A Survey of the Theory of Hypercube Graphs
Computers and Mathematics with Applications **15**, 277-289, 1988

[Hayn&&82] Haynes L.S., Lau R.L., Siewiorek D.P., Mizell D.W.
A Survey of Highly Parallel Computing
Computer **15**, 9-24, January 1982

[Heat85a] Heath M.T. (editor)
Hypercube Multiprocessors
Proceedings of the 1st Conference on Hypercube Multiprocessors, August 26-27, 1985, Knoxville/TN, USA, SIAM 1986

[Heat85b] Heath M.T.
The Hypercube: A Tutorial Overview
Proceedings of the 1st Conference on Hypercube Multiprocessors, 7-10, August 26-27, 1985, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1986

[Heat86] Heath M.T. (editor)
Hypercube Multiprocessors
Proceedings of the 2nd Conference on Hypercube Multiprocessors, September 29 - October 1, 1986, Knoxville/TN, USA, SIAM 1987

[Heat&&90] Heath M.T., Geist G.A., Drake J.B.
Early Experiences with the Intel iPSC/860 at Oak Ridge National Laboratory
Report Oak Ridge National Laboratory, July 24, 1990

[Ho&&88a] Ho A., Fox G.C., Walker D.W., Synder S., Chang D., Chen S., Breaden M., Cole T.
PC-CUBE, A Personal Computer Based Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 92-97, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ho&&88b] Ho A., Fox G.C., Walker D.W., Breaden M., Chen S., Knutson A., Kuwamoto S., Cole T.
MAC-CUBE, The Macintosh-Based Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 98-103, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ho&Jo86] Ho C.T., Johnsson S.L.
Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes
Proceedings of the International Conference on Parallel Processing, 640-648, August 26-27, 1986, St. Charles/IL, USA, edited by K. Hwang, S.M. Jacobs and E.E. Swartzlander

[Ho&Jo88] Ho C.T., Johnsson L.
Optimal Algorithms for Stable Dimension Permutations on Boolean Cubes
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 725-736, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ho&Jo89] Ho C.T., Johnsson S.L.
Spanning Balanced Trees in Boolean Cubes
SIAM Journal on Scientific and Statistic Computing **10**, 607-630, July 1989

[Hock85] Hockney R.W.
MIMD Computing in the USA - 1984
Parallel Computing **2**, 119-136, 1985

[Hock87] Hockney R.W.
Classification and Evaluation of Parallel Computer Systems
Springer-Verlag Lecture Notes in Computer Science **295**, 13-25, 1987

[Ho&Je88] Hockney R.W., Jesshope C.R.
Parallel Computers 2,
Adam Hilger, Bristol, 1988

[Hong&&85] Hong J.P., Tomlinson R.D., Patel N., Pollard L.H.
A Hypercube Project and a Simulator for a Hypercube of Computers
Proceedings of the 1st Conference on Hypercube Multiprocessors, 11-18, August 26-27, 1985, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1986

[Ho&Ul90] Hopcroft J.E., Ullman J.D.
Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie
Addison-Wesley, 1990

[Hw&Gh87] Hwang K., Ghosh J.
Hypernet: A Communication-Efficient Architecture for Constructing Massively Parallel Computers
IEEE Transactions on Computers **C-36**, 1450-1466, December 1987

[INTEL88] The Intel iPSC/2 System: The Concurrent Supercomputer for Production Applications
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 843-846, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[INTEL90] Intel-Kundenschriften 1990

[John88] Johnsson S.L.
Ensemble Architectures and Their Algorithms: An Overview
The IMA Volumes in Mathematics and Its Applications **13**, 109-144, 1988, edited by Martin H. Schultz

[Jo&Ho88] Johnsson S.L., Ho C.T.
Algorithms for Matrix Transposition on Boolean N-Cube Configured Ensemble Architectures
SIAM Journal on Matrix Analysis Applications **9**, 419-454, July 1988

[Jo&Ho89] Johnsson S.L., Ho C.T.
Optimum Broadcast and Personalized Communication in Hypercubes
IEEE Transactions on Computers **C-38**, 1249-1268, September 1989

[Ka&Sh88] Kandlur D.D., Shin K.G.
Hypercube Management in the Presence of Node Failures
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 328-336, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Kats86] Katseff H.P.
Incomplete Hypercubes
Proceedings of the 2nd Conference on Hypercube Multiprocessors, 258-264, September 29 - October 1, 1986, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1987

[Kats88] Katseff H.P.
Incomplete Hypercubes
IEEE Transactions on Computers **C-37**, 604-608, May 1988

[Ki&Re88] Kim C.K., Reed D.A.
Adaptive Packet Routing in a Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 625-630, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Knut75] Knuth D.E.
Big Omicron and Big Omega and Big Theta
SIGACT News, 18-23, 1976

[Kris89] Krishnamurthy E.V.
Parallel Processing: Principles and Practice
Addison-Wesley, 1989

[Krum88] Krumme D.W.
The Simplex Operating System
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 381-383, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Krum&&85] Krumme D.W., Venkataraman K.N., Cybenko G.
Hypercube Embedding is NP-Complete

Proceedings of the 1st Conference on Hypercube Multiprocessors, 148-157, August 26-27, 1985, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1986

[Lamp&&82] Lamport L., Shostak R., Pease M.
The Byzantine Generals Problem
ACM Transactions on Programming Languages and Systems 4, 382-401, July 1982

[Lan&&88] Lan Y., Esfahanian A.H., Ni L.M.
Multicast in Hypercube Multiprocessors
Proceedings of the 7th Annual Internaional Phoenix Conference on Computers and Communications, 26-30, March 16-18, 1988, Scottsdale/AZ, USA

[Lan&&90] Lan Y., Esfahanian A.H., Ni L.M.
Multicast in Hypercube Multiprocessors
Journal of Parallel and Distributed Computing 8, 30-41, 1990

[La&Dh88] Lakshmivarahan S., Dhall S.K.
A New Hierarchy of Hypercube Interconnection Schemes for Parallel Computers
The Journal of Supercomputing 2, 81-108, 1988

[La&El89] Latifi S., El-Amawy A.
On Folded Hypercubes
Proceedings of the International Conference on Parallel Processing, 180-187, 1989, St. Charles/IL, USA

[Lawr75] Lawrie D.H.
Access and Alignment of Data in an Array Processor
IEEE Transactions on Computers **C-24**, 1145-1155, December 1975

[Le&Ha88] Lee T.C., Hayes J.P.
Routing and Broadcasting in Faulty Hypercube Computers
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 346-354, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Leng82] Lengauer T.
Cube-Connected-Cycles – Shuffle-Exchange-Graph
Informatik Spektrum 5, 192-194, 1982

[Levi84] Levitan S.P.
Parallel Algorithms and Architectures: A Programmer's Perspective
Dissertation, University of Massachusetts, 1984

[Losz88] Loszarek J.L.
Hardware Support for Distributed Objects in a Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 26-32, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Marg90] Margulis N.
Cray on a Chip
c't, 38-45, Februar 1990

[McBr88] McBryan O.A.
New Architectures: Performance Highlights and New Algorithms
Parallel Computing **7**, 477-499, 1988

[McBr89] McBryan O.A.
The Connection Machine: PDE Solution on 65536 Processors
Parallel Computing **9**, 1-24, 1988/89

[Mill75] Millard W.
Hyperdimensional μP Collection Seen Functioning as Mainframe
Digital Design, 20-31, November 1975

[Nose&&86] Nosenchuck D.M., Littman M.G., Flannery W.
Two-Dimensional Nonsteady Viscous Flow Simulation on the Navier-Stokes Computer MiniNode
Journal of Scientific Computing **1**, 53-73, 1986

[Nuge88] Nugent S.F.
The iPSC/2 Direct-Connect Communication Technology
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 51-60, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Palm85] Palmer J.F.
A VLSI Parallel Supercomputer
Proceedings of the 1st Conference on Hypercube Multiprocessors, 19-26, August 26-27, 1985, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1986

[Palm88] Palmer J.F.
The NCUBE Family of High-Performance Parallel Computer Systems
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 847-851, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Pa&Ja88] Pan W.M., Jackson V.
A Concurrent Debugger for iPSC/2 Programmers
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 823-831, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Peas77] Pease M.C.
The Indirect Binary n-Cube Microprocessor Array
IEEE Transactions on Computers **C-26**, 458-473, May 1977

[Pier88] Pierce P.
The NX/2 Operating System
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 384-390, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Poll&&86] Pollard H., Cyrus T., Therman R., Hong J., Kellner R., Tomlinson B.
UNMHC: The Los Alamos and University of New Mexico Hypercube
Proceedings of the 2nd Conference on Hypercube Multiprocessors, 271-275, September 29 - October 1, 1986, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1987

[Pr&Vu81] Preparata F.P., Vuillemin J.V.
The Cube Connected Cycles: A Versatile Network for Parallel Computation
Communications of the ACM **24**, 300-309, May 1981

[Quin88] Quinn M.J.
Algorithmenbau und Parallelcomputer
McGraw-Hill, New York, Hamburg, 1988

[Raab&&88] Raabe U., Lobjinski M., Horn M.
Verbindungsstrukturen für Multiprozessoren
Informatik Spektrum **11**, 195-206, 1988

[Ragh&&88] Raghupathy S., Leuze M.R., Schach S.R.
Message Routing Schemes in a Hypercube Machine
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 640-646, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Ra&Sh88] Ramanathan P., K.G. Shin
Reliable Broadcast in Hypercube Multiprocessors
IEEE Transactions on Computers **C-37**, 1654-1657, October 1988

[Ratt85] Rattner J.
Concurrent Processing: A New Direction in Scientific Computing
AFIPS Conference Proceedings **54**, 157-166, July 15-18, 1985, edited by A.S. Wojcik

[Re&Fu87] Reed D.A., Fujimoto R.M.
Multicomputer Networks: Message-Based Parallel Processing
M.I.T. Press Cambridge/MA, USA, 1987

[Re&Ru89] Reed D.A., Rudolph D.C.
Experiences with Hypercube Operating System Instrumentation
International Journal of High Speed Computing **1**, 517-542, 1989

[Sa&Sc85a] Saad Y., Schultz M.H.
Topological Properties of Hypercubes
Research Report YALEU/DCS/RR-389, Yale University, Department of Computer Science, June 1985

[Sa&Sc85b] Saad Y., Schultz M.H.
Data Communication in Hypercubes
Research Report YALEU/DCS/RR-428, Yale University, Department of Computer Science, October 1985

[Sa&Sc86] Saad Y., Schultz M.H.
Data Communication in Parallel Architectures
Research Report YALEU/DCS/RR-461, Yale University, Department of Computer Science, March 1986

[Sa&Sc88] Saad Y., Schultz M.H.
Topological Properties of Hypercubes
IEEE Transactions on Computers **C-37**, 867-872, July 1988

[Sa&Sc89a] Saad Y., Schultz M.H.
Data Communication in Hypercubes
Journal of Parallel and Distributed Computing **6** 115-135, 1989

[Sa&Sc89b] Saad Y., Schultz M.H.
Data Communication in Parallel Architectures
Parallel Computing **11**, 131-150, 1989

[Salm&&88] Salmon J., Callahan S., Flower J., Kolawa A.
MOOSE: A Multi-Tasking Operating System for Hypercubes
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 391-396, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Sc&Bo88] Schwan K., Bo W.
Topologies – Computational Messaging for Multicomputers
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 580-592, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Seit85] Seitz C.L.
The Cosmic Cube
Communications of the ACM **28**, 22-33, 1985

[Sh&Fi89] Shih Y., Fier J.
Hypercube Systems and Key Applications
in: Parallel Processing for Supercomputers & Artificial Intelligence
McGraw-Hill, 1989, edited by K. Hwang and D. DeGroot

[Sieg&&87a] Siegel H.J., Hsu W.T.Y., Jeng M., Nation W.G.
Communication Techniques in Parallel Processing
Springer–Verlag Lecture Notes in Computer Science **295**, 35-60, 1987

[Sieg&&87b] Siegel H.J., Tsun-Yukhsu W., Jeng M.
An Introduction to the Multistage Cube Family of Interconnection Networks
The Journal of Supercomputing **1**, 13-42, 1987

[Sq&Pa63] Squire J.S., Palais S.M.
Programming and Design Consideration for a Highly Parallel Computer
AFIPS Conference Proceedings **23**, 395-400, 1963

[Ston71] Stone H.S.
Parallel Processing with the Perfect Shuffle
IEEE Transactions on Computers **C-20**, 153-161, February 1971

[St&Wa86] Stout Q.F., Wagar B.
Passing Messages in Link-Bound Hypercubes
Proceedings of the 2nd Conference on Hypercube Multiprocessors, 251-257, September 29 - October 1, 1986, Knoxville/TN, USA, edited by M.T. Heath, SIAM 1987

[St&Wa90] Stout Q.F., Wagar B.
Intensive Hypercube Communication
Journal of Parallel and Distributed Computing **10**, 167-181, 1990

[Stou90] Stout Q.F.
Special Issue on Algorithms for Hypercube Computers
Journal of Parallel and Distributed Computing **8**, 301-302, 1990

[Su&Ba77] Sullivan H., Bashkour T.R.
A Large Scale, Homogeneous Fully Distributed Parallel Machine, I
Proceedings of the 4th Annual Symposium on Computer Architecture, 105-117, March 23-25, 1977, Silver Spring/MD, USA

[Trel88] Treleaven P.C.
Parallel Architecture Overview
Parallel Computing **8**, 59-70, 1988

[Tuaz&&88] Tuazon J., Peterson J., Pniel M.
Mark IIIfp Hypercube Concurrent Processor Architecture
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, 71-80, January 19-20, 1988, Pasadena/CA, USA, edited by G.C. Fox

[Tv&Na89] Tvrdik P., Nakamura Y.
Embedding Paths and Circuits into the Hypercube
Journal of the Faculty of Engeneering **65**, Shinshu University, February 1989

[Vali80] Valiant L.G.
Experiments with a Parallel Communication Scheme
Proceedings of the 18th Allerton Conference on Communication, Control and Computing, 802-811, University of Illinois, October 8-10, 1980

[Vali82] Valiant L.G.
A Scheme for Fast Parallel Communication
SIAM Journal of Computing **11**, 350-361, May 1982

[Vali88] Valiant L.G.
Optimally Universal Parallel Computers
Philosophical Transactions of the Royal Society London A **326**, 373-376, 1988

[Va&Br81] Valiant L.G., Brebner G.J.
Universal Schemes for Parallel Communication
Proceedings of the 13th ACM Symposium on Theory of Computation, 263-277, 1981

[Wagn89] Wagner A.
Embedding Arbitrary Binary Trees in a Hypercube
Journal of Parallel and Distributed Computing **7**, 503-520, December 1989

[We&Pa85] Welty L.L., Patton P.C.
Hypercube Architecture
AFIPS Conference Proceedings **54**, 259-265, July 15-18, 1985, edited by A.S. Wojcik

[Wile87] Wiley P.
A Parallel Architecture Comes of Age at Last
IEEE Spectrum **24**, 46-50, June 1987

[Wu85] Wu A.Y.
Embedding of Tree Networks into Hypercubes
Journal of Parallel and Distributed Computing **2**, 238-249, 1985

[Wu&Fe80] Wu C.L., Feng T.Y.
On a Class of Multistage Interconnection Networks
IEEE Transactions on Computers **C-29**, 694-702, August 1980

[Yang&&88] Yang C.S., Wang J.F., Lee J.Y., Boesch F.T.
Graph Theoretic Reliability Analysis for the Boolean n-Cube Networks
IEEE Transactions on Circuits and Systems **35**, 1175-1179, September 1988

[Yo&Na90] Youssef A.S., Narahari B.
The Banyan-Hypercube Networks

IEEE Transactions on Parallel and Distributed Systems **1**, 160-169, April 1990

A Weitere Literatur

Zum Themenbereich Hypercube-Rechner existiert eine Vielzahl von Literatur, die meist in Form von Artikeln in regelmäßig erscheinenden Fachzeitschriften und Tagungsbänden von Konferenzen, aber auch als Berichte von Forschungseinrichtungen und Universitäten erscheinen.
Nachstehende, gesichtete und nach Stichworten geordnete Literaturangaben aus diesem Bereich wurden in der vorliegenden Arbeit nicht verwendet, entweder weil sie nicht das Thema der Arbeit behandeln oder weil deren Berücksichtigung den Umfang der Arbeit gesprengt hätte.
Insbesondere die im Literaturverzeichnis auf den Seiten 139 und 141 aufgeführten Tagungsbände dreier Konferenzen über Hypercube-Rechner und deren Anwendung [Heat85a, Heat86, Fox88] beinhalten eine Vielzahl von Artikeln über, hier nicht oder nur zum Teil berücksichtigte, Aspekte von Hypercube-Rechnern.

A.1 Überblick, Allgemeines

André F., Verjus J.P. (editors)
Hypercube and Distributed Computers
Proceedings of the 1st European Workshop on Hypercube and Distributed Computers, October 4-6, 1989, Rennes/France

Dilger E., Maehle E.
Systemarchitektur und Fehlertoleranz
Informatik-Spektrum **9**, 110-118, 1986

Enslow P.H.
What is a „Distributed" Processing System?
Computer **11**, 13-21, January 1978

Leuze M.R., Dowdy L.W., Park K.H.
Multiprogramming a Distributed-Memory Multiprocessor
Concurrency: Practice and Experience **1**, 19-33, September 1989

Prior D.M.N., Norman M.G., Radcliffe N.J., Clarke L.J.
What Price Regularity?
Concurrency: Practice and Expierience **2**, 55-78, March 1990

Stone H.S.
High Performance Computer Architecture
Addison–Wesley Series in Electrical and Computer Engeneering, 1987

A.2 Spezielle Algorithmen

Ackland B., Ahuja S., DeBenedictis E., London T., Lucco S., Romero D.
MOS Timing Simulation on a Message Based Multiprocessor
IEEE International Conference on Computer Design, VLSI in Computers, 446-450, October 6-9, 1986, Port Chester/NY, USA

Agerwala T., Lint B.
Communications in Parallel Algorithms for Boolean Matrix Multiplication
Proceedings of the International Conference on Parallel Processing, 146-153, August 22-25, 1978, Bellaire/MI, USA

Barszcz E., Chan T.F., Jespersen D.C., Tuminaro R.S.
FLO52 on Hypercubes: Performance and Modelling of a Multigrid Euler Code
International Journal of High Speed Computing **1**, 481-503, 1989

Bastani F.B., Leu D.R.
Abstract Data Types for SIMD Hypercube Machines
Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation, 609-616, October 10-12, 1988, Fairfax/VA, USA

Brochard L.
Efficiency of some Parallel Numerical Algorithms on Distributed Systems
Parallel Computing **12**, 21-44, 1989

Chamberlain R.M.
Gray Codes, Fast Fourier Transforms and Hypercubes
Parallel Computing **6**, 225-234, February 1988

Chan T.F., Saad Y.
Multigrid Algorithms on the Hypercube Multiprocessor
IEEE Transactions on Computers **C-35**, 969-977, November 1986

Chen M.S., Shin K.G.
Processor Allocation in an N-Cube Multiprocessor using Gray-Codes
IEEE Transactions on Computers **C-36**, 1396-1407, December 1987

Choi Y., Esfahanian A.H., Ni L.M.
Ont-to-k Communication in Distributed-Memory Multiprocessors
Proceedings of the 25th Annual Allerton Conference on Communication, Control and Computing, 268-269, September 30 - October 2, 1987, Monticello/IL, USA

Chu E., George A.
QR Factorization of a Dense Matrix on a Hypercube Multiprocessor
SIAM Journal on Scientific and Statistical Computing **11**, September 1990

Cosnard M., Fraigniaud P.
A Performance Analysis of Network Topologies in Finding the Roots of a Polynomial
Proceedings of the Joint International Conference on Vector and Parallel Processing, 875-886, September 10-13, 1990, Zürich/Switzerland, edited by H. Burkhart

Cosnard M., Fraigniaud P.
Finding the Roots of a Polynomial on an MIMD Multicomputer
Parallel Computing **15**, 75-86, 1990

Das S.K., Deo N., Prasad S.
Parallel Graph Algorithms for Hypercube Computers
Parallel Computing **13**, 143-158, 1990

Das S.K., Deo N., Prasad S.
Two Minimum Spanning Forest Algorithms on Fixed-Size Hypercube Computers
Parallel Computing **15**, 179-188, 1990

Deák I.
Uniform Random Number Generators for Parallel Computers
Parallel Computing **15**, 155-164, 1990

Dehne F., Ferreira A.G., Rau-Chaplin A.
Parallel Branch and Bound on Fine-Grained Hypercube Multiprocessors
Parallel Computing **15**, 201-210, 1990

Dekel E., Nassimi D., Sahni S.
Parallel Matrix and Graph Algorithms
SIAM Journal on Computing **10**, 657-675, 1981

Fox G.C.
A Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube
The IMA Volumes in Mathematics and Its Applications **13**, 37-62, 1988, edited by Martin H. Schultz

García I., Merelo J.J., Bruguera J.D., Zapata E.L.
Parallel Quadrant Interlocking Factorization on Hypercube Computers

Parallel Computing **15**, 87-100, 1990

Gustafson J.L., Mordry G.R., Benner R.E.
Developement of Parallel Methods for a 1024-Processor Hypercube
SIAM Journal on Scientific and Statistic Computing **9**, 609-638, July 1988

Hedetniemi S.M., Hedetniemi S.T., Liestman A.L.
A Survey of Gossiping and Broadcasting in Communication Networks
Networks **18**, 319-349, 1988

Hillis W.D., Steele G.L. Jr.
Data Parallel Algorithms
Communications of the ACM **29**, 1170-1183, December 1986

Hoßfeld F.
Parallele Algorithmen
Informatik-Fachberichte Band 64, Springer–Verlag, Heidelberg, 1983

Joe K., Mori Y., Miyake S.
Construction of a Large-Scale Neural Network: Simulation of Handwritten Japanese Character Recognition on NCUBE
Concurrency: Practice and Experience **2**, 79-107, June 1990

Johnsson S.L.
Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures
Journal of Parallel and Distributed Computing 4, 133-172, April 1987

Lee J., Shragowitz E., Sahni S.
A Hypercube Algorithm for the 0/1 Knapsack Problem
Journal of Parallel and Distributed Computing **5**, 438, August 1988

McBryan O.A., Van de Velde E.F.
Hypercube Algorithms and Implementation
SIAM Journal on Scientific and Statistic Computing **8**, s227-s287, March 1987

McBryan O.A., Van de Velde E.F.
Matrix and Vector Operations on Hypercube Parallel Processors
Parallel Computing **5**, 117-126, July 1987

Mudge T.N., Abdel-Rahman T.S.
Vision Algorithms for Hypercube Machines

Journal of Parallel and Distributed Computing 4, 79-94, February 1987

Ni L.M., King C.T., Prins P.
Parallel Algorithm Design Considerations for Hypercube Multiprocessors
Proceedings of the International Conference on Parallel Processing, 717-720, August 17-21, 1987, University Park/PA, USA, edited by S.K. Sahni

Olukotun O.A., Mudge T.N.
A Preliminary Investigation into Parallel Routing on a Hypercube Computer
Proceedings of the 24th ACM/IEEE Conference on Design Automation, 814-820, June 28 - July 1, 1987, Miami Beach/FL, USA

Olukotun O.A., Mudge T.N.
Hierarchical Gate-Array Routing on a Hypercube Multiprocessor
Journal of Parallel and Distributed Computing **8**, 313-324, 1990

Ranka S., Sahni S.
Hypercube Algorithms with Applications to Image Processing and Pattern Recognition
Springer–Verlag New York Berlin Heidelberg, 1990

Ranka S., Sahni S.
Image Template Matching on MIMD Hypercube Multiprocessors
Journal of Parallel and Distributed Computing **10**, 79-84, 1990

Sheu J.P., Kuo N.L., Chen G.H.
Graph Search Algorithms and Maximum Bipartite Matching Algorithm on the Hypercube Network Model
Parallel Computing **13**, 245-251, 1990

Woo J., Sahni S.
Hypercube Computing: Connected Components
The Journal of Supercomputing **3**, 209-234, 1989

A.3 Netzwerk-Struktur

Abraham S., Paddmanabhan K.
Performance of the Direct Binary n-Cube Network for Multiprocessors
IEEE Transactions on Computers **C-38**, 1000-1011, July 1989

Banerjee P.
The Cubical Ring Connected Cycles: A Fault Tolerant Parallel Computation Network
IEEE Transactions on Computers **C-37**, 632-636, May 1988

Djokovic D.Z.
Distance-Preserving Subgraphs of Hypercubes
Journal of Combinatorical Theory (B) **14**, 263-267, 1973

Erdös P., Spencer J.
Evolution of the n-Cube
Computers and Mathematics with Applications **5**, 33-39, 1979

Foldes S.
A Characterization of Hypercubes
Discrete Mathematics **17**, 155-159, 1977

Garey M.R., Graham R.L.
On Cubical Graphs
Jouurnal of Combinatorical Theory (B) **18**, 84-95, 1975

Hart S.
A Note on the Edges of the n-Cube
Discrete Mathematics **14**, 157-163, 1976

Jeng J.F., Sahni S.
All Pairs Shortest Paths on a Hypercube Multiprocessor
Proceedings of the International Conference on Parallel Processing, 713-716, August 17-21, 1987, University Park/PA, USA, edited by S.K. Sahni

Kuo S.Y., Fuchs W.K.
Reconfigurable Cube-Connected Cycles Architecture
Journal of Parallel and Distributed Computing **9**, 1-10, May 1990

Mulder M.
$(0, \lambda)$-Graphs and n-Cubes

Discrete Mathematics **28**, 179-188, 1979

Ringel G.
Über drei kombinatorische Probleme am n-dimensionalen Würfel und Würfelgitter
Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **20**, 10-19, 1955

Wittie L.D.
Communication Structures for Large Networks of Microcomputers
IEEE Transactions on Computers **C-30**, 264-273, April 1981

A.4 Mapping-Strategien, Einbettungen

Chan M.Y., Chin F.Y.L.
On Embedding Rectangular Grids in Hypercubes
IEEE Transactions on Computers **C-37**, 1285-1288, October 1988

Chen G.H., Ho H.F., Lin S.H., Sheu J.P.
Data Mapping of Linear Programming on Fixed-Size Hypercubes
Parallel Computing **13**, 235-243, 1990

Dehne F., Gastaldo M.
A Note on the Load Balancing Problem for Coarse Grained Hypercube Dictionary Machines
Proceedings of the Joint International Conference on Vector and Parallel Processing, 417-422, September 10-13, 1990, Zürich/Switzerland, edited H. Burkhart

Deshpande S.R., Jenevin R.M.
Scalability of a Binary Tree on a Hypercube
Proceedings of the International Conference on Parallel Processing, 661-668, August 19-22, 1986, St. Charles/IL, USA, edited by K. Hwang, S.M. Jacobs and E.E. Swartzlander

Fox G.C.
A Review of Automatic Load Balancing and Decomposition Methods for the Hypercube
The IMA Volumes in Mathematics and Its Applications **13**, 63-76, 1988, edited by Martin H. Schultz

Gulati S. Iyengar S.S., Bahren J.
The Pebble-Crunching Model for Fault-Tolerant Load Balancing in Hypercube Ensembles
The Computer Journal **33**, June 1990

Ho C.T., Johnsson S.L.
Embedding Meshes in Boolean Cubes by Graph Decomposition
Journal of Parallel and Distributed Computing **8**, 325-339, 1990

Krämer O., Mühlenbein H.
Mapping Strategies in Message Based Multiprocessor Systems
Parallel Computing **9**, 213-225, 1989

Lai T.H., White W.
Mapping Pyramid Algorithms into Hypercubes
Journal of Parallel and Distributed Computing **9**, 42-54, 1990

Matic S.
Emulation of Hypercube Architecture on Nearest-Neighbour Mesh-Connected Processing Elements
IEEE Transactions on Computers **C-39**, 698-700, May 1990

Poplawski D.A.
Mapping Rings and Grids onto the FPS T-Series Hypercube
Parallel Computing **7**, 1-10, April 1988

Scott D.S., Brandenburg J.
Minimal Mesh Embeddings in Binary Hypercubes
IEEE Transactions on Computers **C-37**, 1284-1285, October 1988

Sadayappan P., Ercal F., Ramanujam J.
Cluster Partitioning Approaches to Mapping Parallel Programs onto a Hypercube
Parallel Computing **13**, 1-16, 1990

A.5 Fehlertoleranz

Armstrong J.R., Gray F.G.
Fault Diagnosis in a Boolean n-Cube Array of Multiprocessors
IEEE Transactions on Computers **C-30**, 587-590, August 1981

Esfahanian A.H.
Generalized Measures of Fault Tolerance with Application to N-Cube Networks
IEEE Transactions on Computers **C-38**, 1586-1591, November 1989

Ghafoor A., Sole P.
Performance of Fault-Tolerant Diagnostics in the Hypercube Systems
IEEE Transactions on Computers **38**, 1164-1172, August 1989

Rennels D.A.
On Implementing Fault-Tolerance in Binary Hypercubes
Proceedings of the 16th Annual International Symposium on Fault Tolerant Computing Systems, 344-349, July 1-4, 1986, Vienna/Austria

A.6 Spezielle Rechner

Davis IV N.J., Siegel H.J.
The PASM Prototype Interconnection Network Design
AFIPS Conference Proceedings **54**, 183-190, July 15-18, 1985, edited by A.S. Wojcik

Hillis W.D.
The Connection Machine
ACM Distinguished Dissertations, M.I.T. Press, Cambridge/MA, USA, 1985

Messina P., Baillie C., Felten E., Hipes P., Williams R., Alagar A., Kamrath A., Leary R., Pfeiffer W., Rogers J.
Benchmarking Advanced Architecture Computers
Concurrency: Practice and Experience **2**, September 1990

A.7 Sonstige Konzepte für Hypercube-Rechner

Bomans L., Roose D., Hempel R.
The Argonne/GMD Macros in FORTRAN for Portable Parallel Programming and Their Implementation on the Intel iPSC/2
Parallel Computing **15**, 119-132, 1990

Brooks III E.D.
The Shared Memory Hypercube
Parallel Computing **6**, 235-245, 1988

Chen M.S., Shin K.G.
Subecube Allocation and Task Migration in Hypercube Multiprocessors
IEEE Transactions on Computers **39**, 1146-1155, September 1990

Chow E., Madan H., Peterson J. Grunwald D., Reed D.
Hyperswitch Network for the Hypercube Computer
Proceedings of the 15th Annual IEEE Symposium on Computer Architecture, 90-99, May 30 - June 2, 1988 , Honolulu/HI, USA

Ercal F., Ramanujam J., Sadayappan P.
Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning
Journal of Parallel and Distributed Computing **10**, 35-44, 1990

Reed D.A., Rudolph D.C.
Experiences with Hypercube Operating System Instrumentation
International Journal of High Speed Computing **1**, 517-542, 1989

Reddy A.L.N., Banerjee P.
Design, Analysis and Simulation of I/O Architectures for Hypercube Multiprocessors
IEEE Transactions on Parallel and Distributed Systems **1**, 140-151, April 1990

Wu K.L., Fuchs W.K.
Recoverable Distributed Shared Virtual Memory
IEEE Transactions on Computers **C-39**, 460-469, April 1990

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

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich anfertigen zu können und Frau Inge Gutheil für die Betreuung dieser Arbeit.

FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Jül-2428
Januar 1991
ISSN 0366-0885