

# Parallele Rechnerarchitekturen

Bernd Lutz<sup>1</sup>

20. Juli 2003

# 1. Motivation

Obwohl die Rechenleistung moderner Computer in den letzten Jahren stark angestiegen ist und ein Ende dieses Wachstums in den nächsten Jahren nicht zu erwarten ist, haben Parallelrechner nichts von ihrer Bedeutung und Aktualität verloren. Dies liegt zum einen daran, dass auch modernste sequentielle Rechner viele "große" Probleme wie z.B. die Klimaberechnung nicht zufriedenstellend lösen können, zum anderen erlauben Parallelrechner, durch Verwendung zwar älterer, aber damit auch billigerer Prozessoren, die Rechenleistung pro verwendeter Geldeinheit zu steigern. Außerdem erlaubt eine durch parallele Recheneinheiten deutlich gesteigerte Rechenleistung eine Verfeinerung der verwendeten Berechnungsmodelle, was exaktere Ergebnisse zur Folge hat. Dabei lassen sich viele Algorithmen, die z.B. auf Diskretisierung beruhen, nahezu beliebig verfeinern und der Rechenleistung anpassen.

Wie man also sieht, bleiben Parallelrechner auch auf länger absehbare Zeit aktuell. Grund genug, sich mit möglichen Realisierungen des Konzepts Parallelrechner zu befassen und sich über Einteilung und Leistungsfähigkeit der Hardware Gedanken zu machen. Dies soll im ersten Teil dieser Ausarbeitung geschehen.

Der zweite Teil der Ausarbeitung befaßt sich mit der Bewertung von parallelen Algorithmen auf paralleler Hardware. Dabei wird von einer gegebenen Hardware und einem festen Algorithmus ausgegangen. Interessant ist beispielsweise die Frage nach der *Skalierbarkeit*, wie profitiert der Rechner (in einem solchen Fall) von zusätzlichen Prozessoren?

# 2. Einteilung nach Flynn

Die gebräuchlichste Unterteilung der Rechnerarchitekturen lieferte 1966 Flynn. Er untersuchte existierende Rechner und unterteilte sie nach zwei Eigenschaften: der Anzahl der Befehlsströme (instruction streams) und der Anzahl der Datenströme (data streams).

Unter der Anzahl der Befehlsströme versteht man die Anzahl von verschiedenen Befehlen, die zur selben Zeit auf Daten ausgeführt werden können. Die Anzahl der Datenströme entspricht der größten Anzahl an verschiedenen Daten, auf die diese ausgeführten Befehle gleichzeitig wirken.

Aus dieser Definition ergeben sich die vier Kategorien

- SISD
- SIMD
- MISD
- MIMD

## 2.1. SISD

*SISD* ist die Abkürzung für *Single Instruction stream, Single Data stream*. Es stehen also je ein Befehlsstrom und ein Datenstrom zur Verfügung. Jeder Befehl wird separat ausgeführt und wirkt auf nur ein Datum.

In diese Kategorie fallen alle sequentiellen Rechner wie PCs. Insbesondere zählen alle von-Neumann-Rechner in diesen Bereich. Da sie nicht zu den Parallelrechnern zählen, bleibt diese Kategorie im folgenden unbeachtet.

## 2.2. SIMD

*SIMD* steht für *Single Instruction stream, Multiple Data stream*. Dies bedeutet, dass je ein Befehl separat ausgeführt wird, dieser aber auf mehrere Daten angewandt wird.

Da sich diese Art von Architektur besonders für Vektorarithmetik wie Vektoraddition eignet, nennt man die Rechner dieser Kategorie oft Vektorrechner, allerdings ist auch der Begriff Prozessor Array gebräuchlich. Der später beschriebene Rechner CM-200 gehört zu diesen Vektorrechnern.

## 2.3. MISD

*MISD* bedeutet *Multiple Instruction stream, Single Data stream*. Mehrere Befehle wirken also auf ein einzelnes Datum.

MISD-Rechner spielen in der praktischen Anwendung keine Rolle, da sie nicht gebaut wurden. In der Theorie gibt es Überlegungen, wie eine solche Architektur aussehen könnte, allerdings ist nicht eindeutig geklärt, wie ein MISD-Rechner realisiert werden könnte. Ein Beispiel wäre ein Rechner, bei dem ein Datum nacheinander mehrere Recheneinheiten durchläuft, und dabei jeweils neu bearbeitet wird. Solche Rechner nennt man *systolic arrays*, wobei systolic für eine rhythmische Kontraktion, wie z.B. beim Herzmuskel, steht.

## 2.4. MIMD

Die heute wichtigste Kategorie sind *MIMD*-Rechner, also Rechner mit mehreren Befehlsströmen, die auf mehrere Datenströme wirken. Dies wird meist durch mehrere Prozessoren mit eigenem Speicher realisiert.

In dieses Feld fallen die sogenannten Multicomputersysteme, sowie die Multiprozessorsysteme, die später ausführlich besprochen werden. Als Beispiel dienen der Symetry-Rechner, der TC 2000, der nCUBE 2, der CM-5, sowie der Paragon XP/S, die ebenfalls besprochen werden.

## 2.5. Moderne Parallelrechner

Die meisten modernen Parallelrechner lassen sich nicht einfach in diese Kategorien einordnen. Sie haben hierarchische Strukturen, d.h. sie besitzen Ebenen, auf denen sie sich jeweils unterschiedlich nach außen präsentieren. So besitzt der Earth Simulator, der derzeit schnellste Großrechner, zwei MIMD-Ebenen und eine SIMD-Ebene.

Reine SIMD-Rechner werden aufgrund ihrer starren Strukturen nur noch selten gebaut, während MIMD-Rechner vor allem in Mehrprozessorsystemen und PC-Clustern Verwendung finden.

# 3. Multicomputer

## 3.1. Architektur

Unter dem Begriff Multicomputer faßt man alle Systeme zusammen, in denen mehrere Prozessoren genutzt werden, die jeweils über einen eigenen lokalen Speicher mit separatem Adressraum verfügen. Diese einzelnen Prozessoren werden über ein Netz zusammengeschaltet. Erweiterungen wie Grafikkarten oder Peripheriegeräte werden nicht berücksichtigt, auch können sich mehrere Prozessoren ein Gehäuse (bzw. Schaltschrank) teilen.

Da zwischen den Prozessoren nur die Informationen ausgetauscht werden, die der jeweilige Prozessor zu Berechnung benötigt, nennt man diese Systeme auch *Nachrichtenbasierte Rechner*. Als gemeinsame Softwareschnittstelle dient heute meist das Message-Passing-Interface (*MPI*).

Bei frühen Multicomputern koordinierten die Prozessoren selbst den Nachrichtenaustausch. Dadurch mußten sich auch Prozessoren, die eine Nachricht nicht benötigten, mit allen empfangenen Nachrichten beschäftigen, und diese gegebenenfalls weiterleiten. In neueren Rechnern übernimmt diese Aufgabe ein spezielles Routing-Modul, das die Nachrichten automatisch weiterleitet.

### 3.2. Beispiel: nCUBE 2

Der von der nCube Corporation in Foster City, Kalifornien, gebaute nCUBE2 weist große Ähnlichkeit mit klassischen Prozessor arrays auf, ersetzt jedoch die einzelnen Rechenwerke durch vollständige Prozessoren. Dadurch ergibt sich in Analogie zu Prozessorarrays eine Einteilung in drei Baugruppen:

- ein Front-End Computer
- ein Netz aus Prozessoren
- die Ein-/Ausgabeeinheiten wie Festplatten

Wie bei Prozessorarrays dient der Front-End Computer der Kommunikation mit den Nutzern des Rechners. Jedoch fällt die Kontrolle des Programmflusses der Rechenwerke weg, da diese ihren Programmfluß selbständig steuern. Im Konzept des nCUBE2 zeigt sich jedoch dieser Front-End Computer als Schwachstelle des Rechners. Zum einen fehlt jede Möglichkeit, auf den Rechner zuzugreifen, wenn der Front-End Computer ausfällt, zum anderen kann bei großer Anzahl von Nutzern die Rechenkapazität des Front-End Computer zu gering sein.

Das eigentliche Prozessornetz besteht aus bis zu 8.192 einzelnen Prozessoren, die zu einem Hypercube angeordnet sind. Die Verbindung zwischen zwei Prozessoren kann dabei jeweils bis zu 2,2 Megabyte pro Sekunde übertragen. Als Vermittler der Pakete dienen Hardware-Module, die jeweils über einen DMA-Kanal direkt Daten in den Speicher ihres zugeordneten Prozessors schreiben können. Der Speicher beträgt dabei zwischen 1 und 64 Megabyte pro Prozessor. Zusätzlich verfügt jeder Prozessor über einen DMA-Kanal zur Anbindung von externen Datenträgern.

Jeder Prozessor hat eine maximale Performance von 2,5 Megaflops. Dadurch ergibt sich für einen nCUBE2 mit der Höchstzahl von 8.192 Prozessoren eine Spitzenleistung von etwa 20 gigaflops.

### 3.3. Beispiel: Connection Machine CM-5

Um das Problem des einzigen Front-End Computers zu vermeiden, ging die Thinking Machines Corporation in Cambridge bei der Konstruktion ihrer Connection Machine CM-5 einen anderen Weg, als die Entwickler des oben beschriebenen nCUBE2. Statt einen einzigen Computer mit einem Netz von Prozessoren zu verbinden, integrierten sie mehrere Front-End Computer direkt im Netz. Dadurch ergibt sich für die einzelnen Prozessoren eine Einteilung in drei Rollen:

- Ausführende Prozessoren
- Kontroll-Prozessoren
- Ein-/Ausgabe Kontroll-Prozessoren

Die ausführenden Prozessoren führen die eigentliche Berechnung aus und stellen daher den größten Anteil des Rechners dar.

Die Kontroll-Prozessoren arbeiten als Partitions-Manager, die mehrere ausführende Prozessoren kontrollieren. Außerdem verarbeiten sie die Benutzer-Anfragen und dienen der Administration der CM-5.

Die Ein-/Ausgabe Kontroll Prozessoren dienen der Anbindung der Peripherie wie z.B. Festplattenarrays und Bandlaufwerke.

Die Verteilung der Prozessoren auf diese Kategorien gestaltet sich je nach Realisierung variabel, alle Kategorien bestehen jedoch jeweils aus der identischen Hardware. So werden in allen drei Fällen SPARC Prozessoren verwendet, denen jeweils 4 Vektor-Recheneinheiten für Gleitkommarechnung zur Verfügung stehen. Jeder Prozessor erhält bis zu 32 Megabyte lokalen Speicher. Außerdem sind alle Prozessoren unabhängig von ihrer Rolle auf gleiche Weise im Netz integriert.

Das Netz selbst stellt einen Hyperbaum vom Grad 4 dar, dessen Blätter die Prozessoren sind. Alle inneren Knoten des Hyperbaums dienen lediglich dem Routen der Daten zwischen den Blättern. Dadurch wird eine minimale Netzwerkbandbreite von 5 Megabyte pro Sekunde garantiert.

Eine voll ausgestattete CM-5 besteht aus 16.384 Prozessoren, die über einen gesamten Speicher von 512 Gigabyte verfügen. Da jede der vier Vektor-Einheiten pro Prozessor über eine maximale Rechenleistung von 32 Millionen Gleitkommaoperationen pro Sekunde verfügt, ergibt sich eine theoretische Gesamtrechenleistung von etwa 2 Teraflops.

### 3.4. Beispiel: Paragon XP/S

Um den Engpaß eines einzelnen Frontend-Rechners zu vermeiden, gingen die Entwickler der Paragon XP/S der Intel Corporation in Beaverton, Oregon, noch einen Schritt weiter als die Entwickler der CM-5. Bei der XP/S kann ein Prozessor zur Laufzeit zwischen den Kategorien Service-Knoten und Compute-Knoten wechseln, während bei der CM-5 diese Zuteilung beim Bau des Systems festgelegt wird. Dadurch kann die XP/S je nach Bedarf auch eine große Anzahl an Benutzern bedienen, ohne Rechenleistung einzubüßen, wenn nur wenige Benutzer auf dem Rechner arbeiten. Außerdem kann ein Prozessor auch als Ein-/Ausgabe-Knoten dienen, diese Zuordnung kann jedoch nicht verändert werden.

Die Architektur der Paragon XP/S beruht auf einem 2-dimensionalen Gitter, bei dem die Knoten aus Routing-Modulen bestehen. Jedes Routing-Modul kann mehr als 200 Megabyte Daten pro Sekunden im Gitter übertragen. Außerdem ist jeweils ein Routing Modul mit einem Prozessorknoten verbunden. Dabei wird nicht zwischen Service Knoten, Compute Knoten und Ein-/Ausgabe Knoten unterschieden.

Ein Prozessorknoten besteht aus mehreren einzelnen Baugruppen, die über einen separaten Bus verbunden sind. So übernimmt ein i860 XP Prozessor das Message Passing, also die Übermittlung von Daten an andere Prozessoren. Die eigentliche Programmausführung im Knoten übernehmen ein einzelner oder mehrere Applikations-Prozessoren von gleichem Typ (i860 XP). Ihnen stehen zwischen 16 und 128 Megabyte an Speicher zur Verfügung. Schließlich wird der Knoten noch durch einen Performance Monitor und das Netzwerk-Interface zum Routing Modul ergänzt. Handelt es sich um einen Ein-/Ausgabe Knoten, so ist der Speicher auf 64 Megabyte begrenzt, der Knoten enthält jedoch zusätzlich eine Ein-/Ausgabeschnittstelle.

Die maximale Rechenleistung eines i860 XP/S beträgt 75 Millionen Gleitkommaoperationen pro Sekunde. Bestückt man eine Paragon XP/S mit 1024 Knoten, die jeweils 4 Applikations-Prozessoren enthalten, so ergibt sich eine theoretische maximale Rechenleistung von etwa 300 Gigaflops. Bei 128 Megabyte Speicher pro Knoten, würde die Paragon XP/S damit über 128 Gigabyte Speicher verfügen.

## 4. Multiprozessorsysteme

Im Gegensatz zu Multicomputern teilen sich in Multiprozessorsystemen mehrere Prozessoren einen globalen Adressraum. Dadurch greifen alle Prozessoren bei der selben Speicheradresse auf das identische Datum zu. Durch die Anordnung des Speichers im Bezug zu den Prozessoren ergeben sich zwei Unterteilungen:

- UMA
- NUMA

### 4.1. UMA-Architektur

Uniform-Memory-Access-Architekturen, auch UMA-Architekturen genannt, zeichnen sich durch einen einheitlichen Zugriff auf den Speicher aus, d.h. die Zugriffszeit auf ein bestimmtes Datum ist für alle Prozessoren identisch. Dazu trennt man die Speicherbänke logisch vom Prozessor. Will ein Prozessor auf den Speicher zugreifen, so muß er seine Anfrage über die allen gemeinsame Speicherverwaltung machen.

Aufgrund der hohen Belastung der zwischenliegenden Speicherverwaltung eignet sich UMA nur für eine geringe Anzahl von Prozessoren. Außerdem muß die Speicherverwaltung sicher stellen, dass die Daten der Prozessoren im Cache aktuell bleiben. Dies erschwert die Realisierung der Architektur.

### 4.2. Beispiel: Symetry

Der Symetry, gebaut von Computer Systems Inc. in Beaverton, Oregon, ist ein UMA-Multiprozessor. Als Prozessoren dienen dem Symetry zwischen 2 und 30 Intel 80386 CPUs, ein 32 Bit Prozessor. Um diesem eine Gleitkommaeinheit zur Verfügung zu stellen, erhält jede CPU wahlweise einen Intel 80387 oder einen Weittek WTL116 als Coprozessor. Da der 80386 für Ein-Prozessor-Systeme entwickelt wurde, muß man ihm außerdem einen separaten System link und einen Interrupt Controller zur Verfügung stellen. Jeder Prozessor verfügt zudem über 64 Kbyte Cache, auf dem Daten lokal gespeichert werden.

Die Prozessoren werden über einen Systembus mit 8 bis 240 Mbyte Speicher und der Peripherie verbunden. Der Bus kann 32 Bit-Adressen verarbeiten und 32 oder 64 Bit Datenwerte übertragen. Seine Gesamtübertragungsrate beträgt 53,3 Mbyte pro Sekunde. Der Systembus verfügt über eine Pipeline. Dadurch kann bereits mit anderen Anfragen begonnen werden, während eine Anfrage noch nicht beendet wurde, da sie zum Beispiel noch auf den Speicherzugriff wartet. Außerdem verfügen die Prozessoren über einen separaten kleinen Bus, auf dem z.B. Interrupts und Fehlersignale ausgetauscht werden.

Daß bei allen UMA- und NUMA-Rechnern existierende Cache-Kohärenz-Problem, also das Aktuell halten des Cache auf allen Prozessoren, wurde hier durch eine copy-back-Strategie gelöst. Dies bedeutet, dass ein Prozessor, der Daten im Cache ändert, allen anderen Prozessoren signalisiert, dass dieser Bereich bei ihnen nicht mehr aktuell ist. Er schreibt die Daten jedoch erst zurück in den Speicher (= copy back), wenn ein anderer Prozessor diese anfordert, oder er sie aufgrund eigener anderer Datenzugriffe wieder auf den Speicher auslagern muß.

### 4.3. NUMA-Architektur

Im Gegensatz zu UMA besitzt die NUMA-Architektur keinen einheitlichen Speicherzugriff, also Non-Uniform-Memory-Access. Dies erreicht man, in dem man jedem Prozessor einen Teil des Speicherbereichs eng zuordnet, den vorhandenen Speicherbereich also auf die Prozessoren verteilt. Greift ein Prozessor nun auf Daten zu, die

in seinem Speicherbereich liegen, geschieht dies sehr schnell. Erst wenn er Daten außerhalb dieses Bereichs anfordert, werden die Daten über die Speicherverwaltung geleitet. Dieser Zugriff ist deutlich langsamer.

Auch diese Architektur limitiert die Anzahl der Prozessoren auf eine relativ kleine Zahl. Die Speicherverwaltung muß ebenfalls wieder sicherstellen, dass die Daten in den Caches aktuell bleiben.

#### 4.4. Beispiel: TC200

Die von BBN Systems and Technologies of Cambridge, Massachusetts gebaute TC200 ist ein NUMA-Multiprozessorsystem. Die TC200 vereint bis zu 128 Prozessor-Knoten. Diese bestehen aus einer Motorola 88100 CPU, sowie je 3 Motorola 88200 chips. Diese beinhalten zum einen jeweils einen Instruktions-Cache und einen Daten-Cache, zum anderen stellen sie die Schnittstelle zu 4 bis 16 Mbyte primären Speicher zur Verfügung, auf den jedoch alle Prozessoren zugreifen können. Außerdem stellen sie die Verbindung zum Butterfly-Verbindungsnetz und einen VME-Bus zur Verfügung. Der VME-Bus dient der Anbindung externer Datenträger wie Festplatten, von denen jeweils 16 Mbyte direkt in den Adressraum gemappt werden.

Zur Steuerung des TC200 dient ein weiterer separater Computer. Er überwacht die Prozessoren, steuert den Bootprozess und kann Ergebnisse zusammenfaßen.

Das Cache-Kohärenz-Problem löst die TC200, indem nur Daten, die exklusiv für einen Prozessor bestimmt sind, in den Cache geladen werden dürfen. Dadurch wird das Problem nicht wirklich gelöst, sondern umgangen.

Jeder Prozessor verfügt über eine maximale Performance von 20 megaflops. Für eine voll ausgebaute TC200 ergibt sich somit eine peak-performance von 2,5 gigaflops.

### 5. Prozessor Arrays

Prozessor Arrays sind ein Beispiel für SIMD-Rechner, dies bedeutet, dass sie einen einzigen Befehl parallel auf mehrere Daten anwenden können. Hierzu werden mehrere Rechenwerke ohne eigenen Programmspeicher von einem zentralen sequentiellen Rechner gesteuert.

Der sequentielle Rechner bearbeitet den Programmablauf und führt alle nicht-parallelen Programmanteile aus. Außerdem steuert er über einen separaten Instruktionsbus die einzelnen Rechenwerke.

Die Rechenwerke haben jeweils einen separaten Speicher. Auf diesem führen sie die Befehle aus, die sie vom Rechenwerk erhalten. Dadurch können alle Rechenwerke gleichzeitig denselben Befehl bearbeiten. Außerdem sind die verschiedenen Rechenwerke über ein Netzwerk miteinander verbunden, über das die Speicherinhalte ausgetauscht werden können.

Besonders geeignet sind Prozessor Arrays für Vektorarithmetik. So kann zum Beispiel bei der Addition von zwei Vektoren jeweils ein Rechenwerk einen Vektoreintrag bearbeiten und das Ergebnis in einen Ergebnisvektor schreiben.

Prozessor Arrays gelten heute als veraltet und werden nicht mehr als Parallelrechner gebaut. Dafür besitzen moderne Prozessoren meist eine separate Vektoreinheit, die ähnliche Funktionen realisiert. Man spricht dann oft von Vektorcomputern.

#### 5.1. Beispiel: Connection Machine CM-200

Die Beschreibung der Connection Machine Cm-200 basiert auf der Thinking Machine von 1989. Sie wird von der Thinking Machines Corporation in Cambridge, Massachusetts, gebaut.

Die CM-200 ist ein klassisches Prozessorarray und besteht aus 3 Baugruppen:

- ein Front-End Computer, meist eine Workstation von Sun
- eine Parallel Recheneinheit mit 32-Bit Werten
- ein Ein- und Ausgabesystem

Der Front-End Computer speichert alle sequentiellen Daten und steuert den Programmfluß, wobei sequentielle Anteile am Programm nur von ihm ausgeführt werden. Parallele Befehle gibt er mittels Broadcast-Versand an alle Recheneinheiten weiter. Dabei unterstützt ihn ein Sequencer, der die zu versendenden Befehle in kleinere "Nanoinstruktionen" aufteilt. Diese können vom Rechenwerk direkt ausgeführt werden.

Zum Datenaustausch mit den Rechenwerken gibt es die Möglichkeit, einen Wert an alle Rechenwerke oder nur an ein einzelnes Rechenwerk zu senden. Sollen Daten von Rechenwerken gelesen werden, so gibt es die Möglichkeiten entweder einzeln von jedem Rechenwerk zu lesen oder mehrere Werte gleichzeitig einzulesen und diese dabei mit Operationen wie Summe, Maximum, globales ODER bzw. anderer einfacher Funktionen zu verknüpfen.

Die parallele Recheneinheit besteht aus 2048 bis 65536 einzelnen Rechenwerken. Diese funktionieren seriell, d.h. sie verknüpfen je 2 Bits aus dem Speicher und 1 Bit aus ihren Flags zu einem Bit in den Flags und einem Bit, das gespeichert werden kann. Mit diesen einfachen Verknüpfungen lassen sich alle logischen und arithmetischen Funktionen realisieren, die mit 3 Bits als Eingabe 2 Bits als Ausgabe erzeugen. Um jedoch 32 Bit-Werte zu verarbeiten, muß man entweder 32 Rechenwerke parallel nutzen, oder ein Rechenwerk 32 Rechenschritte Zeit geben und die Bits seriell verknüpfen.

Je 16 einzelne Rechenwerke werden zu einem Chip zusammengefasst. Für die maximale Anzahl an Rechenwerken entspricht dies  $2^{12} = 4096$  Chips. Je zwei dieser Chips teilen sich einen gemeinsamen Speicher und eine Fließkommaeinheit. Verbunden werden die Chips über ein Hypercube-Netzwerk.

Eine voll ausgebaute CM-200 erreicht eine Spitzenleistung von ungefähr 40 Gigaflops. Die schnellsten heutigen Parallelrechner erreichen hier im Vergleich dazu eine Leistung, die um den Faktor  $10^2$  bis  $10^3$  höher liegt.

## 6. Netzwerke

Zur Verbindung von Prozessoren untereinander stehen mehrere Schaltungsmöglichkeiten zur Verfügung. Dabei wird die Verbindung als Punkt-zu-Punkt-Verbindungen zwischen zwei Prozessoren realisiert. Dadurch ergeben sich beliebige Netzwerke, die sich zum Teil erheblich in der Leistungsfähigkeit, der Programmierbarkeit und der Kosten unterscheiden.

Im nächsten Abschnitt werden deshalb zunächst Kriterien zur Beurteilung von Netzen eingeführt und anschließend die gebräuchlichsten Netz vorgestellt. Dabei bedeuten Knoten die Prozessoren und Kanten die Verbindungen zwischen den Prozessoren.

### 6.1. Beurteilung

Zur Beurteilung von Netzen dienen vor allem vier Merkmale:

- Durchmesser des Netzwerks
- bisektions Bandbreite des Netzwerks

- Zahl der Kanten pro Knoten
- Maximale Kantenlänge

Der *Durchmesser* eines Netzwerks gibt die längste Distanz zwischen zwei beliebigen Knoten an. Ein kleiner Durchmesser erleichtert den Datenaustausch zwischen den Prozessoren.

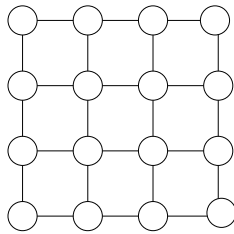
Die *bisectionbreite Bandbreite* eines Netzwerks gibt die minimale Anzahl an Kanten an, die durchtrennt werden müssen, um das Netz in zwei etwa gleich große Teile aufzutrennen. Eine hohe Bisectionsbreite vermindert die Gefahr von Flaschenhälsen im Netz, also Kanten, durch die große Mengen an Daten geführt werden müssen.

Die *Anzahl der Kanten pro Knoten* sollte von der Größe des Netzwerks unabhängig sein. Dadurch skaliert das System besser mit der Knotenzahl.

Die *maximale Kantenlänge* gibt Aufschluss über die physikalische Aufbaugröße des Netzes. Sie sollte möglichst mit wachsendem Netz konstant bleiben.

## 6.2. Gitter oder Torus

Ein lange Zeit verwendetes Netz zur Verbindung von Knoten ist das Gitter. Im Gitter werden die Knoten auf einem n-dimensionalen Netz angeordnet. Dabei darf n auch größer als 3 sein. Jeder Knoten wird mit seinen benachbarten Knoten verbunden. Dadurch ergeben sich für jeden Knoten  $2n$  Verbindungen. Die freien Kanten der Randknoten können dabei mit Knoten auf der gegenüberliegenden Seite oder der versetzten Zeilen und Reihen verbunden werden. Dadurch entsteht ein n-dimensionaler Torus

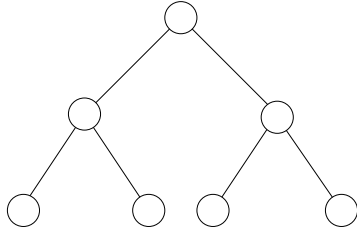


Für Gitter ohne Verbindung an den freien Randknoten ergibt sich für den Durchmesser eines Netzwerks mit  $k^n$  Knoten der Wert  $n(k - 1)$ . Dieser Wert ist im Vergleich zu anderen Netzen relativ hoch und erfordert speziell angepasste Algorithmen.

Die Bisectionsbreite des Gitter liegt bei  $k^{n-1}$ . Dies entspricht im 2-dimensionalen Beispiel dem Durchtrennen von 4 Kanten auf einer beliebigen Spiegelachse. Da jeder innere Knoten mit  $2n$  Knoten verbunden ist, ist auch die Anzahl der Kanten pro Knoten der Wert  $2n$ . Bei 2- und 3-dimensionalen Gittern ist die maximale Kantenlänge konstant, da die Gitter auch real als Gitter bzw. Würfel realisierbar sind.

## 6.3. Binärer Baum

Ein Netzwerk, das sich besonders gut für "Divide and Conquer"-Algorithmen eignet, ist der binäre Baum. Er verbindet bei einer Tiefe von  $k - 1$  insgesamt  $2^k - 1$  Knoten. Dabei hat jeder innere Knoten einen Vater und zwei Kinder. Der oberste Knoten hat nur zwei Kinder, die Blätter nur einen Vater. Dadurch ergibt sich für die Anzahl der Kanten pro Knoten die Zahl 3.

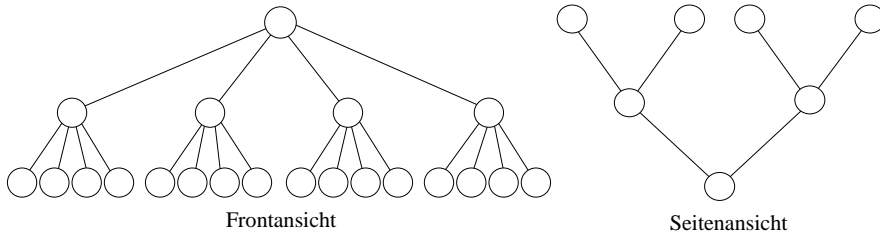


Der binäre Baum hat mit  $2(k - 1)$  einen sehr kleinen Durchmesser. Allerdings ergibt sich eine sehr schlechte Bisectionsbreite von 1, denn es muss nur eine Kante des obersten Knoten (= Wurzel) durchtrennt werden, um den Baum in zwei Bäume aufzuteilen.

Da die Knoten physikalisch eine bestimmte Ausdehnung besitzen, läßt sich der binäre Baum nicht mit einer konstanten maximalen Kantenlänge realisieren.

### 6.4. Hyperbaum

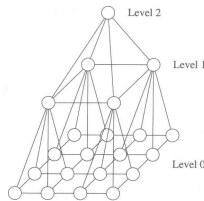
Der Hyperbaum ist eine Weiterentwicklung des binären Baums, der dessen schlechte Bisectionsbreite verbessern soll. Dazu kombiniert man einen normalen abfallenden Baum der Tiefe  $d$  und Grad  $k$ , d.h.  $k$  Kinder pro Knoten, mit einem aufsteigenden binären Baum. In Frontansicht erhält man den abfallenden Baum, während in der Seitenansicht der aufsteigende binäre Baum vorliegt. Dadurch erhält die unterste Ebene des abfallenden Baums eine Reihe von Knoten, die nächst höhere Ebene zwei Reihen und die  $i$ -te Ebene  $2^{(i - 1)}$  Reihen von Knoten für  $i \geq 1$ .



Ein Baum von Grad 4 und Tiefe  $d$  hat damit insgesamt  $2^d(2^{d+1} - 1)$  Knoten. Der Durchmesser beträgt  $2d$ . Die Bisectionsbreite hat sich gegenüber des binären Baums mit  $2^{d+1}$  deutlich verbessert. Die Anzahl der Kanten pro Knoten beträgt für innere Knoten 6 und ist konstant. Genau wie der binäre Baum läßt sich der Hyperbaum ebenfalls nicht mit einer konstanten Kantenlänge realisieren. Sie wächst mit zunehmender Tiefe des Baums.

### 6.5. Pyramide

Die Pyramide kann als eine Kombination von 2-dimensionalen Gittern und Bäumen vom Grad 4 gesehen werden. Sie besteht aus einzelnen 2-dimensionalen Gittern, die Ebenen in der Pyramide bilden. Je ein Knoten in einer Ebene hat 4 Kinder in der darunterliegenden Ebenen. Die einzelnen Ebenen werden von unten beginnend durchnummeriert. Die Größe der Pyramide  $k^2$  gibt die Anzahl der Knoten in der untersten Ebene an. Ihre Höhe beträgt  $\log_2 k$ .

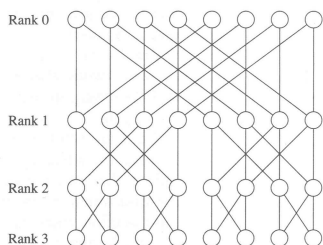


Gegenüber dem 2-dimensionalen Gitter ist der Durchmesser der Pyramide deutlich niedriger. Er beträgt  $2 \log k$ . Die Bisectionsbreite bleibt allerdings etwa konstant und beträgt  $2k$ .

Ein Knoten hat unabhängig von der Größe des Netzes maximal 9 Kanten. Die maximale Kantenlänge steigt allerdings mit steigender Größe des Netzes länger.

## 6.6. Butterfly (Schmetterling)

Im Butterfly-Netzwerk werden die  $(k + 1)2^k$  Knoten auf  $k + 1$  Reihen oder Bänke zu je  $n = 2^k$  Knoten. Diese werden von 0 bis  $k$  nummeriert. Jeder Knoten erhält 4 Verbindungen zu anderen Knoten. 2 Kanten verbinden ihn jeweils mit den beiden gegenüberliegenden Knoten in den benachbarten Bänken.

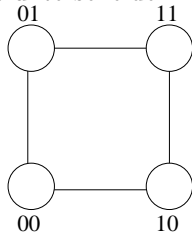


Die restlichen zwei Verbindungen ergeben sich durch folgende Bedingung: Der  $j$ -te Knoten ( $0 \leq j < n$ ) in der  $i$ -ten Bank ( $0 \leq i \leq k$ ) wird mit dem  $m$ -ten Knoten in der  $(i-1)$ -ten Bank verbunden, wobei sich die Zahl  $m$  ergibt, in dem man das Bit  $i$  in der Binärdarstellung von  $j$  invertiert. Diese Verbindungen nennt man Butterfly-Kanten.

Der Durchmesser eines Netzwerks mit  $(k + 1)2^k$  Knoten ist  $2k$ . Die Bisectionsbreite beträgt  $2^{k-1}$ . Die maximale Anzahl der Kanten pro Knoten ist konstant 4, die maximale Kantenlänge wächst mit der Größe des Netzwerks durch das Anwachsen der Kantenlänge der Butterfly-Kanten.

## 6.7. Hypercube

Im Hypercube werden  $2^k$  Knoten zu einem  $k$ -dimensionalen Array zusammengefügt. Dazu verbindet man jeweils die Knoten, deren Binärdarstellung sich in genau einem Bit unterscheiden.

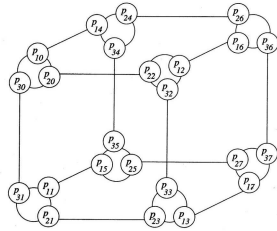


Der Durchmesser dieses Netzes ist  $k$ , da schlimmstenfalls  $k$  Bits ihren Wert ändern müssen. Seine Bisectionsbreite beträgt  $2^{k-1}$ . Damit verbindet der Hypercube einen geringen Durchmesser mit einer hohen Bisectionsbreite. Allerdings ist die Anzahl der Kanten pro Knoten nichtmehr unabhängig von der Netzwerkgröße, sondern hat den Wert  $k$ . Die maximale Kantenlänge steigt mit der Netzwerkgröße.

## 6.8. Cube-Connected Cycles

Das Cube-Connected Cycles Netzwerk besteht aus einem Hypercube, dessen Knoten durch Ringe von  $k$  Knoten ersetzt werden. Jeder Knoten in einem Ring hat

eine Verbindung zu einem Knoten in einem benachbarten Ring, sowie 2 Verbindungen zu Knoten im eigenen Ring. Dadurch werden insgesamt  $k \cdot 2^k$  Knoten im Netz eingebunden.

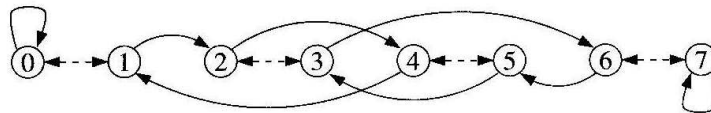


Im Vergleich mit dem Hypercube hat dieses Netz den Vorteil einer konstanten Zahl von Kanten pro Knoten. Dafür steigt der Durchmesser des Netzes von  $k$  auf  $2k$ . Die Bisektionsbreite beträgt  $2^{k-1}$ . Die maximale Kantenlänge steigt ebenfalls mit der Netzgröße.

### 6.9. Shuffle-Exchange

Das Shuffle-Exchange Netzwerk verbindet  $n = 2^k$  Knoten durch zwei verschiedene Arten von Kanten, den Shuffle- und den Exchangekanten.

Die Exchange-Kanten verbinden jeweils 2 Knoten, bei denen sich das niedrigste Bit in der Binärdarstellung ihrer Nummerierung unterscheidet. Die Shuffle-Kanten dagegen verbinden Knoten  $i$  mit dem Knoten, dessen Binärdarstellung sich durch linksrotieren der binären Zahl  $i$  ergibt, d.h. das oberste Bit wird zum niedrigsten Bit, alle anderen Bits wandern eine Position nach oben.



Die Shuffle-Kanten erzeugen eine Ordnung, die man mit dem Mischen eines Kartenstapels vergleichen kann: Teilt man einen Stapel sortierter Karten in zwei gleich große Stapel und nimmt anschließend jeweils eine Karte abwechselnd von den Stapeln, so erhält man die Anordnung des Netzwerks. Für 8 Knoten entspricht das der Anordnung: 0, 4, 1, 5, 2, 6, 3, 7. Dies bedeutet, dass Knoten 4 eine Verbindung mit Knoten 1 hat, Knoten 1 mit Knoten 2, usw.

Ein solches Netz der Größe  $2^k$  hat einen Durchmesser von  $2k - 1$  und eine Bisektionsbreite von  $2^{k-1}/k$ . Die Anzahl der Kanten pro Knoten ist für jeden Knoten 4. Die maximale Kantenlänge ist abhängig von der Größe des Netzes.

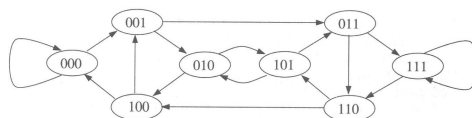
### 6.10. de Bruijn

Ein *de Bruijn*-Netzwerk besteht aus  $n = 2^k$  Knoten, die binär nummeriert werden. Die Verbindungen zwischen den Knoten erhält man, indem man Knoten  $a_{k-1}a_{k-2} \dots a_1a_0$  mit den beiden Knoten

$$a_{k-2}a_{k-3} \dots a_1a_00$$

$$a_{k-2}a_{k-3} \dots a_1a_01$$

verbindet.



Der Durchmesser des Netzwerks ist  $k$ . Dies entspricht etwa der Hälfte eines Shuffle-Exchange Netzwerks. Die Bisectionsbreite beträgt  $2^k/k$ . Die Anzahl der Kanten pro Knoten ist unabhängig von der Netzgröße, während die maximale Kantenlänge mit zunehmender Größe anwächst.

### 6.11. Crossbar Switch

Crossbar Switch sind eine oft verwendete Art, mehrere Rechner miteinander zu vernetzen. Auch bei sehr großen Parallelrechner sind sie heute sehr gebräuchlich.

Prinzipiell handelt es sich bei einem Crossbar Switch um ein separates Gerät, mit dem jeder Rechner verbunden wird. Diese Verbindungen besitzen eine bestimmte maximale Datenrate. Der Switch ist nun in der Lage, zwischen den angeschlossenen Rechnern eine Punkt-zu-Punkt-Verbindung herzustellen. Dazu stellt er einen Teil seiner internen Datenrate für die Verbindung zu Verfügung. Solange diese interne Datenrate nicht überschritten wird, können die verbundenen Rechner mit der vollen Geschwindigkeit ihrer Anbindung kommunizieren.

Die Einordnung des Crossbar-Switch gestaltet sich schwierig, da er eigentlich nicht direkt mit den anderen vorgestellten Netzwerken vergleichbar ist. Es nutzt im Gegensatz zu den anderen Netzwerken eine zusätzliche Verbindungseinheit, das Switch.

### 6.12. Zusammenfassung

Netz	Knoten	Durchmesser	Bisectionsbandbreite
2-D Gitter	$k^2$	$2(k-1)$	$k$
3-D Gitter	$k^3$	$3(k-1)$	$k^2$
Binärer Baum	$2^k - 1$	$2(k-1)$	1
Hyperbaum	$2^k(2^{k+1} - 1)$	$2k$	$2^{k+1}$
Pyramide	$(4k^2 - 1)/3$	$2 \log k$	$2k$
Butterfly	$(k+1)2^k$	$2k$	$2^k$
Hypercube	$2^k$	$k$	$2^{k-1}$
Cube-connected cycle	$k2^k$	$2k$	$2^{k-1}$
Shuffle-exchange	$2^k$	$2k-1$	$\geq 2^{k-1}/k$
de Bruijn	$2^k$	$k$	$2^k/k$
Crossbar Switch	$k$	1	Switch fällt aus

Von allen vorgestellten Netzen läßt sich lediglich das Gitter mit einer konstanten maximalen Kantenlänge realisieren. Damit kann es zum Teil seinen schlechten Durchmesser wieder ausgleichen. Ebenfalls zu erwähnen bleibt der Hypercube, der als einziges keine konstante Anzahl von Knoten in Abhängigkeit der Netzgröße besitzt. Diesen Nachteil kann er allerdings durch seine guten Leistungsdaten für geringe Prozessoranzahlen ausgleichen.

Der Vergleich der verschiedenen Netze mit einem Crossbar Switch fällt schwierig. Es handelt sich bei diesem eigentlich nicht um ein klassisches Netz, sondern um ein zusätzliches Verbindungsgerät. Die Hardwareanforderung bei diesem separaten Gerät steigt bei wachsender Prozessorenanzahl stark an. Damit sind auch die Einträge in der Tabelle nicht direkt vergleichbar, sondern stellen eine Näherung dar.

Insgesamt fällt auf, dass es keine "perfekte" Netzwerke gibt. Prinzipiell erkaufte man sich jeweils einen Vorteil in einem Bereich durch Nachteile in den anderen Kategorien. So erhält man mit dem binären Baum zum Beispiel ein Netz mit niedrigem Durchmesser, aber auch mit einer geringen Bisectionsbreite. Der Hypercube erkaufte sich seine guten Werte bei Durchmesser und Bisectionsbreite durch eine wachsende Anzahl an Kanten pro Knoten. Es gilt also einen Kompromis zu finden, der auf ein gegebenes Problem am Besten passt.

## 7. Abschließendes Beispiel für Hardware: Earth Simulator

Als aktuelles Beispiel für einen Multicomputer dient der derzeit schnellste Großrechner, der Earth Simulator, gebaut von NEC in Japan. Der Earth Simulator ist ein Beispiel für hierarchische Strukturen, d.h. je nach betrachteter Ebene liegen verschiedene Hardwarestrukturen vor. Diese Art von Hardwareorganisation ist bei aktuellen Großrechnern sehr gebräuchlich, jedoch ist sie je nach Rechner unterschiedlich realisiert.

Auf oberster Ebene besteht der Earth Simulator aus einem Verbindungsnetz, das 640 Prozessorknoten mittels eines Crossbar Switch miteinander verbindet. Dabei kann zwischen zwei Knoten eine bidirektionale Verbindung mit einer Datenrate von 12,3 Gigabyte pro Sekunde hergestellt werden. Damit liegt die theoretische Gesamtdatenrate des Netzwerks bei 8 Terabyte pro Sekunde.

Die nächst tiefere Ebene stellen die Prozessorknoten dar. Sie bestehen aus jeweils 8 einzelnen Arithmetik-Prozessoren, die sich 16 Gigabyte gemeinsamen Speicher teilen. Sie bilden also jeweils ein Multiprozessorsystem mit einheitlichem Speicherzugriff (UMA). Die Speicheranbindung besitzt dabei eine Bandbreite von 256 Gigabyte pro Sekunde.

Die unterste Ebene schließlich bilden die Prozessoren. Es handelt sich um mit 500 MHz getaktete Prozessoren von NEC, die intern eine SIMD-Einheit darstellen. Sie kombinieren jeweils eine skalare Einheit mit 8 Vektor-Einheiten, die parallele Befehle ausführen.

Bei diesem Aufbau eines Großrechners werden also insgesamt 5120 Vektor-Prozessoren verwendet. Jeder Prozessor besitzt dabei eine maximale Rechenleistung von 8 Gigaflops. Für einen Prozessorknoten ergibt sich somit eine Rechenleistung von 64 Gigaflops, der gesamte Earth Simulator erreicht eine theoretische Gesamtleistung von 40 Teraflops. Die mit einem Benchmark gemessene Performance liegt jedoch "nur" bei circa 35 Teraflops. Da jeder Prozessorknoten über 16 Gigabyte Speicher verfügt, ergibt sich eine Gesamtspeicherkapazität von 10 Terabyte.

## 8. SpeedUp, Parallelisierbarkeit

Hat man sich für eine bestimmte Rechnerarchitektur entschieden und einen parallelen Algorithmus erstellt, so stellt sich die Frage, wie stark der Algorithmus von der Parallelisierung profitiert. Dazu wurden die Begriffe Speedup (= Beschleunigung) und Scaled Speedup entwickelt.

### 8.1 Speedup

Zur Bestimmung des *Speedup* benötigt man zunächst die Zeit  $t_s$ , die der beste bekannte sequentielle Algorithmus zur Lösung des Problems auf dem Parallelrechner benötigt. Außerdem bestimmt man die Zeit  $t_p$ , die derselbe Parallelrechner mit  $p$  Prozessoren für den parallelen Algorithmus benötigt.

Der Speedup  $S$  berechnet sich nun aus dem Verhältnis der beiden Zeiten:

$$S = \frac{t_s}{t_p}$$

Mit Hilfe des Begriffs Speedup, läßt sich nun der Begriff der *Effizienz* einführen:

$$E = \frac{S}{p}$$

Dabei ist  $S$  der oben berechnete Speedup und  $p$  die Anzahl der Prozessoren im Parallelrechner. Der Wert gibt den Wirkungsgrad eines einzelnen Prozessors im Gesamtsystem an.

Eine andere Leistungsbewertung des Systems ist die *Skalierung* bzw. die *Parallelisierbarkeit*. Um diesen Wert zu bestimmen, wird die Zeit gemessen, die ein Parallelrechner mit einem Prozessor benötigt, um den parallelen Algorithmus zu bearbeiten. Diese setzt man wie beim Speedup ins Verhältnis zur Zeit, die der Parallelrechner für den Algorithmus mit  $p$  Prozessoren benötigt.

## 8.2. Scaled Speedup

Bei der Definition des Speedup ergeben sich zwei Probleme:

- Das zu bearbeitende Problem kann zu groß für einen Prozessor sein.
- Es werden feste Problemgrößen behandelt, obwohl oft die Problemgröße pro festem Zeitabschnitt interessant ist.

Ein Beispiel für den zweiten Fall ist der Wetterbericht für den nächsten Tag. Für die Berechnung ist nicht interessant, ob der Algorithmus eine Woche oder nur 2 Tage benötigt, man hätte das Ergebnis gerne vor dem Tag, für den er bestimmt ist. Dafür würde man das Gitter, das zur Auswertung der aktuellen Wettersituation über die Erde gelegt wird, gerne möglichst eng setzen, ohne dass die zur Verfügung stehende Zeit überschritten wird.

Eine Lösung für diese Probleme ist der *Scaled SpeedUp*, eine Definition, die auf die Problemgröße pro Zeiteinheit aufgebaut wird. Zur Berechnung des Scaled Speedup bestimmt man zunächst die Problemgröße, die ein optimaler sequentieller Algorithmus auf einem Prozessor des Parallelrechners bewältigen kann. Dies ergibt eine Zahl  $n_1$ . Anschließend bestimmt man die Zahl  $n_p$ , die die mögliche Problemgröße auf  $p$  Prozessoren des Parallelrechner angibt. Anschließend bestimmt man die Zeit, die ein Prozessor für die Bearbeitung des größeren Problems benötigt hätte, in dem man das Verhältnis der Problemgrößen bildet.

$$t_1 = \frac{n_p}{n_1}$$

Aus dieser Zeit läßt sich nun der Speedup bilden, in dem man die Zeit  $t_1$  ins Verhältnis zur Zeitbasis  $t$  setzt:

$$S = \frac{t_1}{t}$$

## 8.3. Kann der SpeedUp schneller als linear wachsen?

Nach der Einführung des Begriffs Speedup, stellt sich die Frage, wie stark dieser mit zusätzlicher Prozessoranzahl wachsen kann. Genauer interessiert die Frage, ob der SpeedUp über linear wachsen kann. Dies bedeutet, dass durch Verdopplung der Prozessorenzahl der SpeedUp mehr als doppelt so hoch wird.

In der Theorie gilt dieser Zusammenhang nicht. Das Hauptargument gegen sogenannte Superlineare Speedups ist, dass auf einem Prozessor durch time slicing mehrere Prozessoren emuliert werden können, die das Problem lösen. Dazu teilt man die Rechenzeiten des einzelnen Prozessors in kleine Einheiten auf, in denen jeweils die Aufgaben der simulierten Prozessoren abgearbeitet werden. Dadurch erhält man einen sequentiellen Algorithmus, der durch die Emulation mit großer Wahrscheinlichkeit nicht der schnellste existierende Algorithmus ist. Dadurch ergibt sich rechnerisch sogar, dass der SpeedUp auf jedenfall langsamer als linear wächst.

In der Praxis hingegen gibt es reale Beispiele für Superlineare SpeedUp's. Eines dieser Beispiele sind Algorithmen für Suchprobleme. Während  $p$  Prozessoren

den Suchbereich in  $p$  kleinere Suchbereiche aufteilen können, muß ein einziger Prozessor den kompletten Suchbereich alleine durchsuchen. Dadurch erhöhen die  $p$  Prozessoren gegenüber dem einzelnen Prozessor die Chance auf einen Glückstreffer. Außerdem kann der kleinere Suchbereich auch schneller durchsucht werden.

Ein anderes praktisches Beispiel sind Probleme, deren Größe den Cache eines einzelnen Prozessors übersteigt, jedoch in die Caches von  $p$  Prozessoren paßt. Dadurch verliert ein einzelner Prozessor deutlich mehr Zeit durch Speicherzugriffe außerhalb des Caches als die  $p$  parallelen Prozessoren.

#### 8.4. Amdahls Gesetz

Eine mathematische Untersuchung des SpeedUp führt auf eine mathematische Beziehung, die *Amdahls Gesetz* genannt wird. Sei hierzu  $f$  der Anteil an sequentiell zu bearbeitendem Code in einem Programm.  $f$  liegt hierbei im Bereich  $0 \leq f \leq 1$ . Außerdem ist  $p$  die Anzahl der Prozessoren im Parallelrechner. Dadurch erhält man für den maximalen SpeedUp die Beziehung:

$$S \leq \frac{1}{f + \frac{1-f}{p}}$$

Mit wachsender Problemgröße sinkt meist der sequentielle Anteil im Programm. Dadurch erhält man den sogenannten *Amdahl-Effekt*. Je größer das Problem, desto stärker profitiert man von zusätzlichen Prozessoren, während bei einem festen Anteil von sequentiell Code der SpeedUp ab einer bestimmten Prozessoranzahl kaum mehr wächst.

### 9. Literaturverzeichnis

Dem Vortrag und der Ausarbeitung des Themas lagen folgende Quellen zu Grunde:

- Parallel Computing - Theory and Practice, Michael J. Quinn, Verlag: Mc Graw-Hill, Series in Computer Science.
- Homepage des Earth Simulators: [www.es.jamstec.go.jp](http://www.es.jamstec.go.jp)