

1 Parallelrechner

1.1 Einführung

1.2 Gründe für Parallelität

1.3 Einwände gegen Parallelität

1.4 Cache Kohärenz

1.5 Summationsproblem

1.6 Rechnerklassifizierung

Bild eines Rechnerkonglomerats:

Parallelrechner, Multirechner oder Rechnernetz?

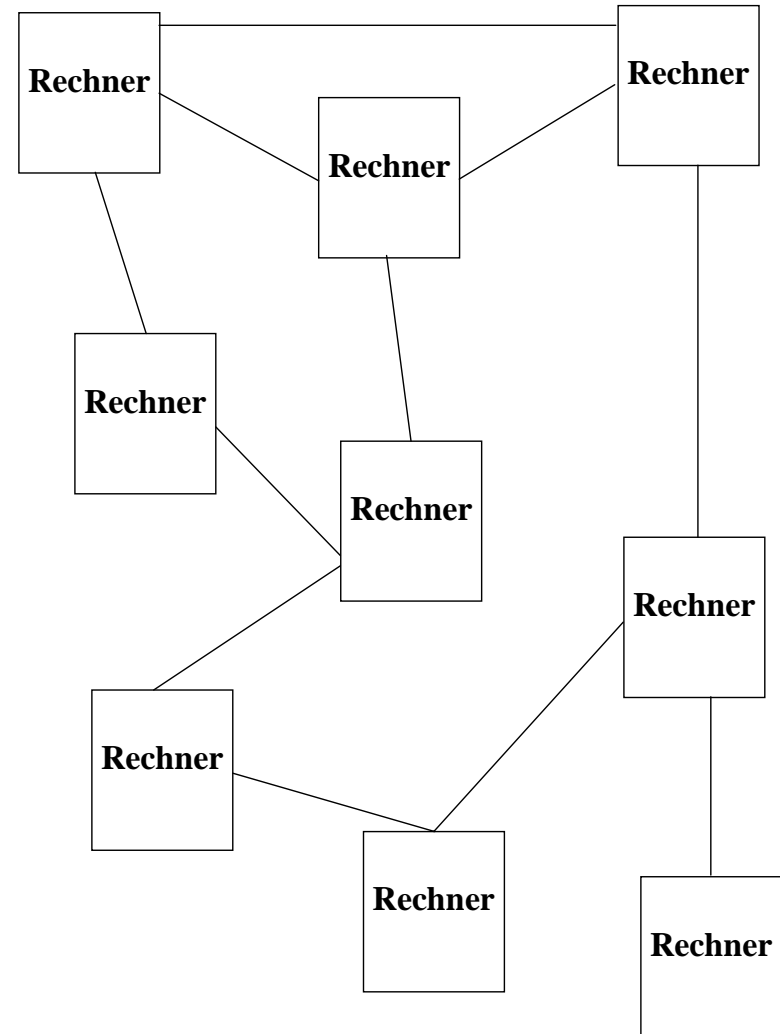


Bild eines einfachen lokalen Rechnernetzes:

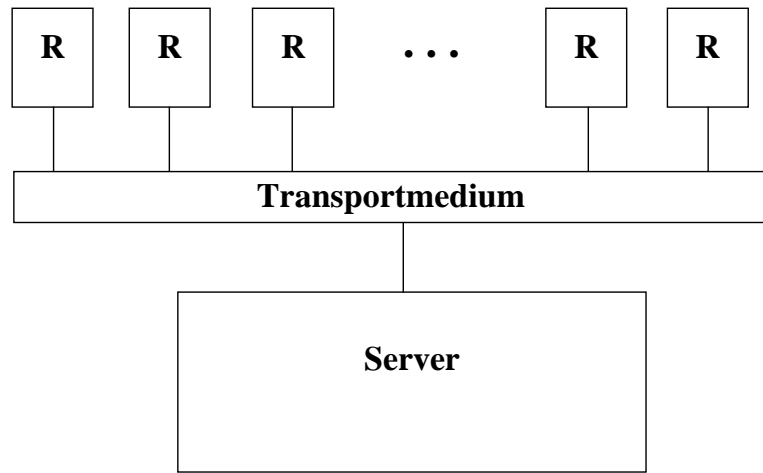
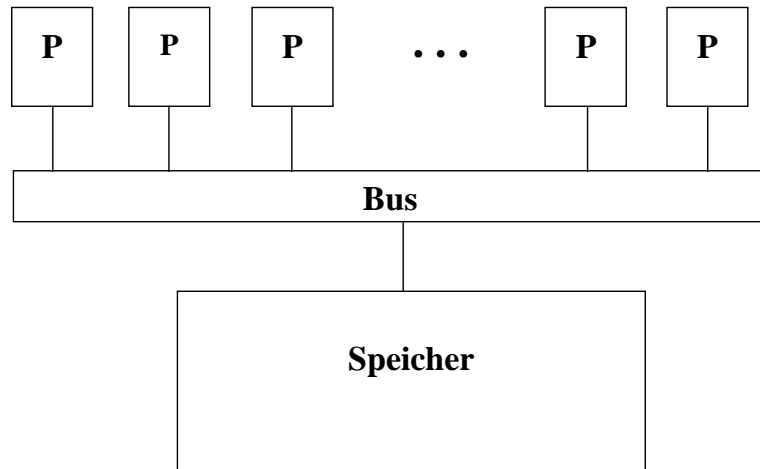


Bild eines speichergekoppelten Mehrprozessorsystems ("Symmetric Multiprocessor"):



Auszug aus Top-500 Liste vom November 2005:

Rang	Rechner	Leistung (GFLOPS) (a) Linpackl. (b) Maximall.	Zahl der Prozes- soren	Jahr
1	BlueGene/L - eServer BGS IBM	(a) 280.600 (b) 367.000	131.072	2005
2	BGW - eServer BGS, IBM	(a) 91.290 (b) 114.688	40.960	2005
3	ASC Purple IBM	(a) 63.390 (b) 77.824	10.220	2005
4	Columbia SGI Altix 1.5 GHz	(a) 51.870 (b) 60.960	10.160	2004
6	Red Storm Cray XT3	(a) 36.190 (b) 43.540	10.880	2005
7	Earth Simulator NEC	(a) 35.860 (b) 40.960	5.120	2002
47	ASCI White SP Power3, 375 MHz, IBM	(a) 7.304 (b) 12.288	8.192	2000
276	ASCI Red Intel	(a) 2.379 (b) 3.207	9.632	1999
494	SR8000-F1/168 Hitachi	(a) 1.653 (b) 2.016	168	2002

Bemerkung: 249 der Top-500 Systeme nutzen Gigabit-Ethernet als Verbindungsnetz.

Struktur des Rechners BlueGene/L:

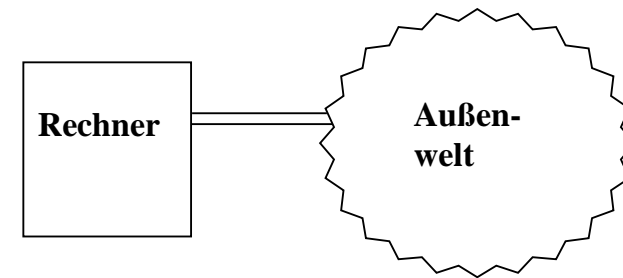
- Chip:** Zwei PowerPC 440 Prozessoren mit integrierten Fließkommaeinheiten, 4 MiB gemeinsamen L3 Cache
- Karte:** 2 Chips mit 512 MiB
- Board:** 16 Karten
- Schrank:** 2 * 16 Boards
- System:** 64 Schränke, insgesamt 2^{17} Prozessoren
- zusätzlich:** 1024 E/A-Knoten

Fünf Verbindungsnetze:

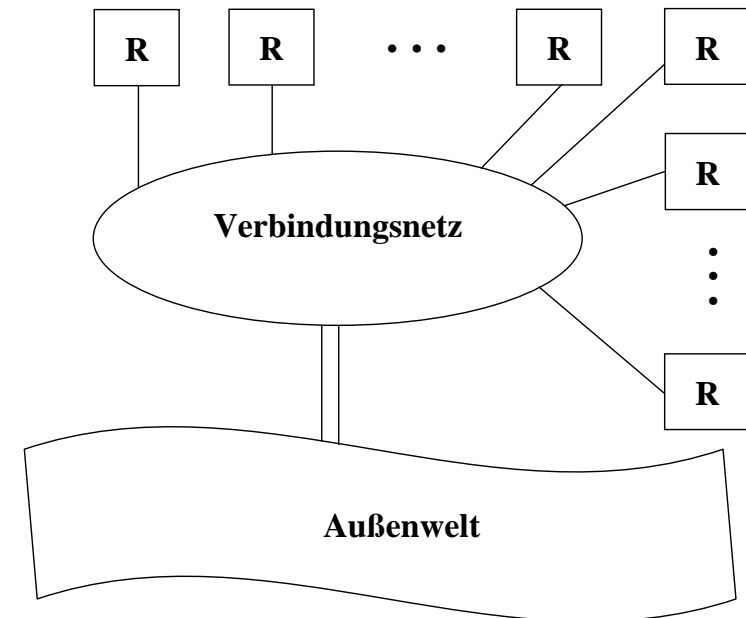
Test- und Service Netz,
Netz für Ein- und Ausgabe,
Netz für Synchronisationsaufgaben,
baumstrukturiertes Reduktionsnetz,
Nachrichtennetz in Form eines dreidimensionalen
Torus (64 x 32 x 32)

MFTB: 6,16 Tage

Monorechner:



Multirechner:



Gründe für Parallelität:

1. Erhöhung der Systemleistung
 - (a) Höherer Durchsatz
 - (b) Kürzere Reaktionszeit
2. Vereinfachung der Programmierung
3. Erhöhung der Verfügbarkeit
4. Erhöhung der Verlässlichkeit
5. Graduelle Leistungsanpassung
6. Minderung der Kosten

Sieger des Gordon Bell Wettbewerbs 1993:

Berechnungen von Turbulenzen in Gasen mittels Boltzmann Gleichungen.

Rechner: CM-5 mit 1024 Knoten;
je Sparc-Knoten
4 Vektorprozessoren a 32 MFLOPS

Leistungsgrenze: $2^{10} * 4 * 2^5$ MFLOPS
 ≈ 131 GFLOPS

Erreichte Dauerleistung: 60 GFLOPS

Programmiersprache: Fortran

Leistung des simplen Fortran-Codes: 40 GFLOPS

Leistung nach Handoptimierung: 60 GFLOPS

Quelle: Computer, Jan. 1994, Vol. 27, 1, pp. 69-75

Minskys Vermutung:

Ein Lösungsmodell für allgemeine Probleme besteht darin, daß man das allgemeine Problem in mehrere Teilprobleme zerlegt, die Teilprobleme löst und die Teillösungen zu einer Lösung des Urproblems zusammenfügt. Ist das ursprüngliche Problem ein großes, dann sind die Zerlegungs- und Zusammenfüggungsschritte mehrfach rekursiv zu wiederholen. Hieraus folgerte Minsky, für die Lösung allgemeiner Probleme gilt:

Nutzbare Leistung eines Mehrprozessorsystems
= Konstante * Logarithmus (Prozessorzahl)

Schaubild zu Minskys Vermutung

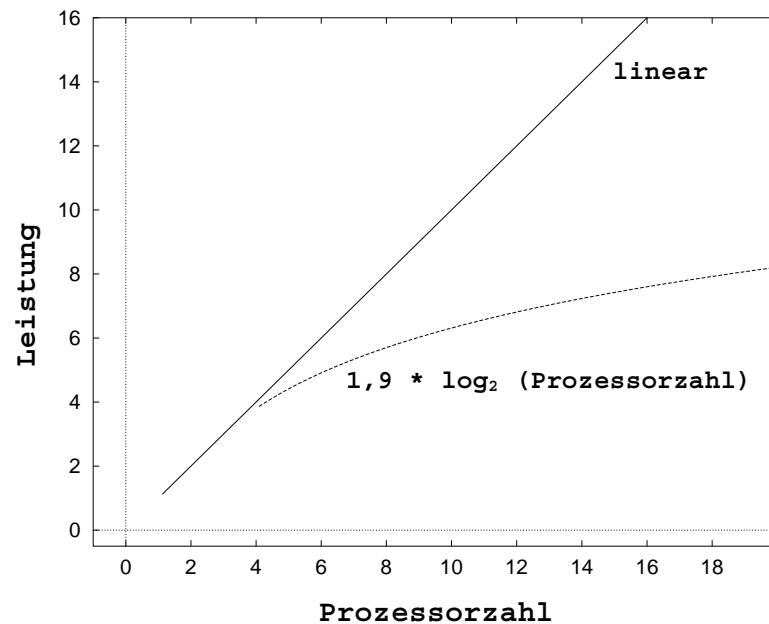
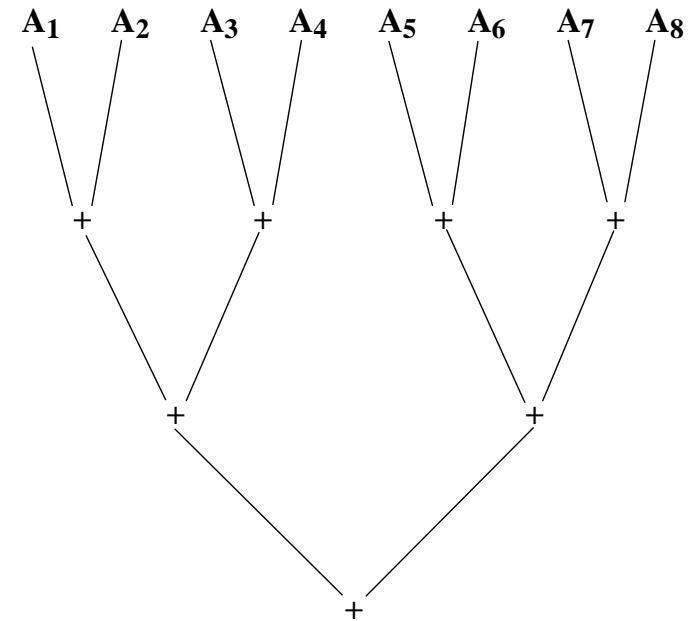


Illustration zu Minskys Vermutung:

Berechne $\sum_{i=1}^n A_i$

Sei $n = 8$



Hier: Zahl der Zeitschritte = $3 = \log_2 8$

Bemerkung: Das obige Beispiel ist nur paradigmatisch zu verstehen. Viele Probleme sind inhärent hierarchisch strukturiert, daher unterliegen ihre Lösungen einem logarithmischen Zusammenfüggungsgesetz.

Amdahls Gesetz:

Im April 1967 hielt G. Amdahl, der Architekt der Rechnerreihe IBM/360, einen Vortrag etwa folgenden Inhalts.

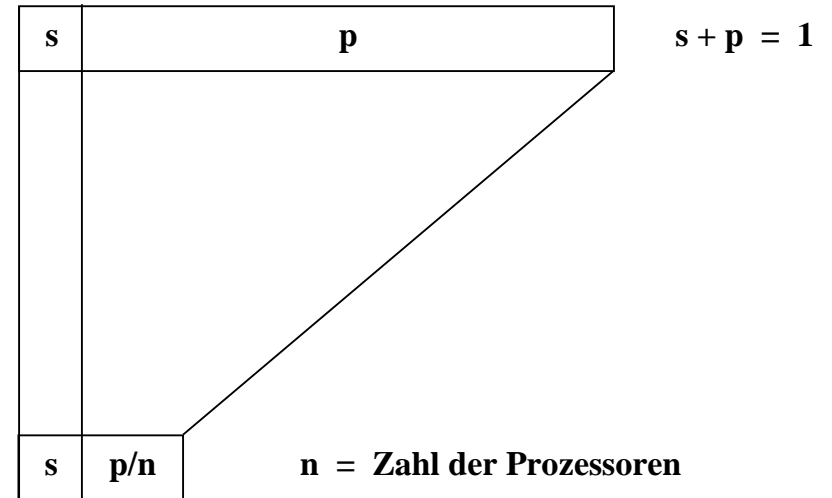
1. Seit 10 Jahren wird behauptet, daß der Monorechner seine "natürlichen" Grenzen erreicht hat, wirklicher Fortschritt ist nur im Bereich der Multirechner zu erwarten.
2. Es ist schwer einzusehen, daß man mit einem Parallelrechner im allgemeinen eine Geschwindigkeitssteigerung um mehr als einen Faktor 7 gegenüber einem Monorechner erreichen kann.

Ein Beispiel: Laufzeiten eines Programms auf verschiedenen Architekturen:

80386 mit 80387	1.181 sec.
80386 mit Weitek (2 MFLOPS)	515 sec.
80386 mit 4 T800 Transputern (4 * 1,5 MFLOPS)	
<direkte Übertragung>	1.652 sec.
<Überlappung E/A und Rechnung>	770 sec.
<Maschinencode für Kernschleife>	254 sec.

Daten aus Jon Flower & al.: Finite-Element Analysis on a PC, IEEE Software, Sept. 91, p.50-57

Jede Aufgabe besteht aus einem seriellen Teil s und einem parallelisierbaren Teil p .



Geschwindigkeitssteigerung:

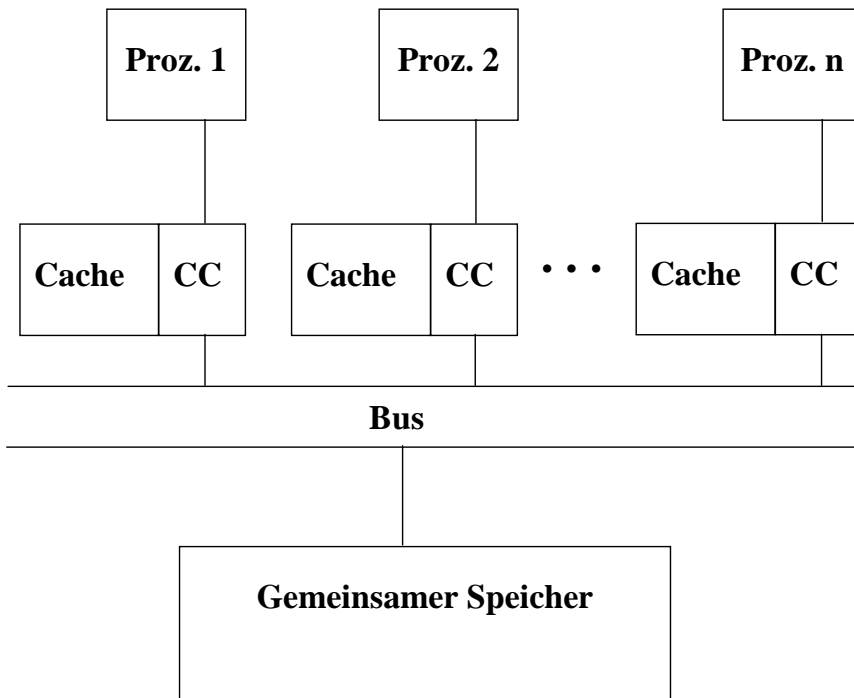
$$b(s,n) = \frac{s + p}{s + \frac{p}{n}} = \frac{1}{s + \frac{p}{n}}$$

$$b^*(s) = \lim_{n \rightarrow \infty} b(s,n) = \frac{1}{s}$$

Beispiele:

$b^*(0,2)$	=	5
$b^*(0,1)$	=	10
$b^*(0,01)$	=	100
$b^*(0,001)$	=	1000

Cache Kohärenz:



CC = Cache Controller

Bemerkungen:

- (i) **Der Bus stellt ein Broadcast-Medium dar. Die Cache Controller beobachten alle Vorgänge auf dem Bus.**
- (ii) **Auf dem Bus werden nur atomare Transaktionen abgewickelt. Es gibt keine verschachtelten Bustransaktionen.**

Die Begriffspaare Write-Invalidate – Write-Update und busbeobachtend – verzeichnisbasiert bestimmen die Protokollklasse für Cache-Kohärenz.

Beispiel zu Write-Update:

Aktion	Cache 1	Cache 2	Hauptspeicher
	-	-	A
P1: Lese A	A	-	A
P2: Lese A	A	A	A
P1: Schreibe A	A	A (veraltet)	A (veraltet)
Lege A auf Bus	A	A	A
P2: Lese A	keine Busaktivitäten		

Beispiel zu Write-Invalidate:

Aktion	Cache 1	Cache 2	Hauptspeicher
	-	-	A
P1: Lese A	A	-	A
P2: Lese A	A	A	A
P1: Schreibe A	A	A (veraltet)	A (veraltet)
Invalidiere A	A	-	A (veraltet)
P2: Lese A	Buslesen wird von CC 1 abgebrochen		
CC1: A auf Bus	A	A (aktuell)	A (aktuell)

Bemerkung: Im Allgemeinen verursacht ein Write-Invalidate Protokoll eine geringere Buslast als ein Write-Update Protokoll. Warum?

Write-Once Protokoll von Goodman (1983):

Aktionen des Prozessors:

**P-Read,
P-Write.**

Busaktionen des Cache Controllers:

**Blockread,
Blockwrite,
Write-Invalidate (Einen Cacheblock neu schreiben
und alle sonstigen inhaltsgleichen invalidieren.),
Read-Invalidate (Einen Cacheblock zwecks
überschreiben einlesen.).**

Zustände eines Cache-Blocks:

**Invalid,
Valid,
Reserved,
Dirty.**

Kurzbeschreibung der Zustände:

Invalid: Der Cache-Block ist nicht vorhanden oder ungültig.

Valid: Der Cache-Block stimmt überein mit dem Hauptspeichereintrag.

Reserved: Ein gültiger Cache-Block wurde genau einmal überschrieben, im Hauptspeicher existiert eine exakte Kopie.

Dirty: Der einzige gültige Cache-Block im System.

Beschreibung der Zustandsübergänge und Aktionssequenzen:

Lesetreffer bei P-Read: Keine Zustandsänderung.

Lese-Nichttreffer bei P-Read: Der Cache Controller plaziert einen Blockread auf den Bus. Falls eine Dirty-Kopie des angeforderten Blocks existiert, unterbricht der zugehörige Cache Controller den intendierten Lesevorgang aus dem Hauptspeicher und sendet über den Bus seine Kopie, hierbei wird auch der Hauptspeicher aktualisiert. Beide Cacheblöcke wechseln dann in den Zustand Valid.

Schreibtreffer bei P-Write: Falls der Cacheblock sich in einem der Zustände Reserved oder Dirty befindet, wird der Cacheblock nur lokal aktualisiert. Der Endzustand ist dann Dirty. Falls der Cacheblock sich im Zustand Valid befindet, wird der Cacheblock lokal aktualisiert, mit Write-Invalidate werden Kopien in anderen Cachespeichern invalidiert. Die Hauptspeicherkopie wird aktualisiert. Der Zustand wechselt nach Reserved.

Schreib-Nichttreffer bei P-Write: Die gelesene Kopie stammt aus dem Hauptspeicher oder von einer Dirty-Kopie aus einem Cache. Eingelesen wird mittels des Befehls Read-Invalidate. Alle möglichen Kopien werden invalidiert. Lokal wird die Kopie aktualisiert und in den Zustand Dirty überführt.

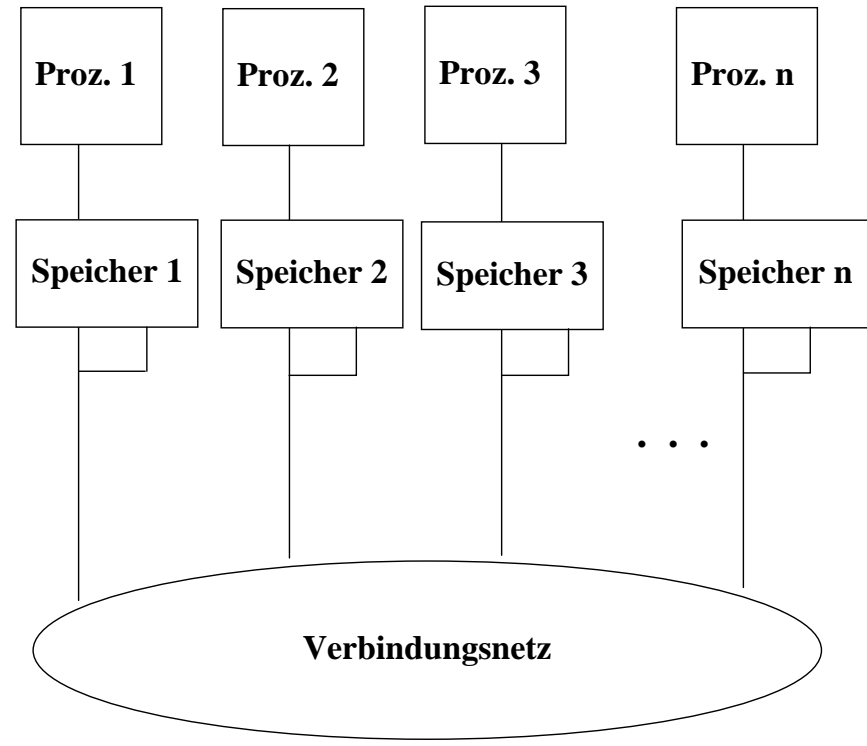
Bemerkung: Beim ersten Schreiben folgt der Cache einer Write-Through Disziplin, ansonsten einer Write-Back Disziplin.

Beispiel:

Die Variablen A und B mögen im gleichen Cacheblock liegen. Zu Beginn sei der Zustand jeweils Invalid.

Aktion	C1-Zustand, Wert	C2-Zustand, Wert	Hauptspeicher
P1: P-Write A = 10, Read-Invalidate	Invalid	Invalid	A = 5, B = 5
P1: P-Read A	Dirty	A = 10	A = 5, B = 5
P2: P-Read A, Blockread	Dirty	A = 10	A = 5, B = 5
P2: P-Write A = 20	Valid	A = 10	A = 10, B = 5
Write-Invalidate	Invalid	A = 20	A = 20, B = 5
P2: P-Write B = 30, P1: P-Read A	Invalid	A=20,B=30	A = 20, B = 5
Blockread	Valid	A=20,B=30	A = 20, B = 30

Bild eines Parallel-Rechners:



Zu lösen sind zwei Aufgaben:

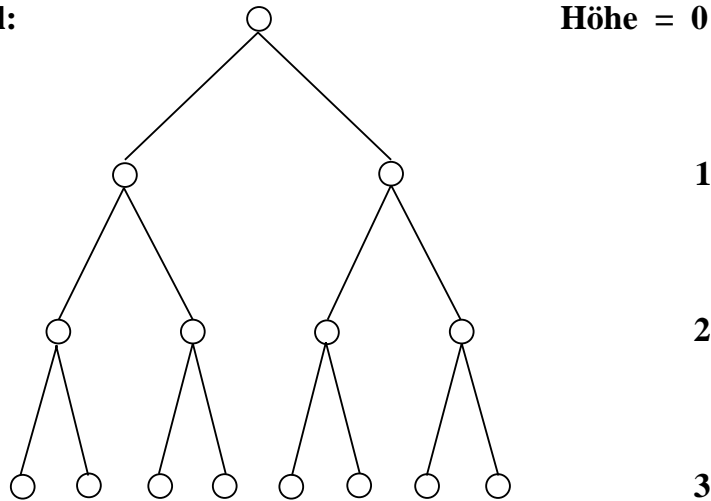
- (i) Interprozessorkommunikation
- (ii) Synchronisation

Zitat aus Almasi und Gotlieb (1989): A parallel computer is a "collection of processing elements that communicate and cooperate to solve large problems fast."

Summationsbaum:

Problem: Berechne $A = \sum_{i=1}^N a[i]$

Beispiel:



Sei N = Zahl der Summanden,
 P = Zahl der Prozessoren.

Bestimmung von

Zeitbedarf $T(N, P)$,
Geschwindigkeitssteigerung ("speed up") $S(N, P)$,
Effizienz $E(N, P)$.

Fall 1: $N = P$

$$T(N, P) = \Theta(\log P) = \Theta(\log N)$$

$$S(N, P) = T(N, 1) / T(N, P) = \Theta(P / \log P)$$

$$E(N, P) = S(N, P) / P = \Theta(1 / \log P)$$

Fall 2: $N = L * P$, P konstant

$$T(N, P) = T(L * P, P) = \Theta(L) + \Theta(\log P)$$

$$S(N, P) = \Theta(L * P) / \Theta(L + \log P) = \Theta(P)$$

$$E(N, P) = \Theta(P) / P = \Theta(1)$$

Bemerkung: Der Kommunikationsaufwand ist vernachlässigbar, der Geschwindigkeitsgewinn begrenzt.

Fall 3: Problemgröße und Prozessorzahl wachsen in ausgewogenem Verhältnis.

Sei $N = P * \log P$

$$T(N, P) = T(P * \log P, P)$$

$$= \Theta(\log P) + \Theta(\log P)$$

\swarrow Phase der unabhängigen Summationen \uparrow Kombinationsphase

$$= \Theta(\log P)$$

$$S(N, P) = \Theta(P * \log P) / \Theta(\log P) = \Theta(P)$$

$$E(N, P) = \Theta(P) / P = \Theta(1)$$

Frage: Müssen Parallelarchitekturen dem Problem angepaßt werden?

Eine Rechnerklassifikation:

SISD	Monoprozessor
SIMD	Vektorrechner Feldrechner
MISD	
MIMD	Gemeinsamer Speicher (enge Kopplung) SMP NUMA COMA Verteilter Speicher (lose Kopplung) MPP (Grid, Hypercube) "Cluster"

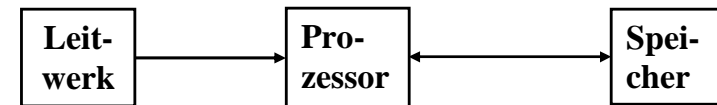
Abkürzungen:

COMA	=	Cache Only Memory Architecture
MIMD	=	Multiple Instruction Streams, Multiple Data Streams
MISD	=	Multiple Instruction Streams, Single Data Stream
SIMD	=	Single Instruction Stream, Multiple Data Streams
SISD	=	Single Instruction Stream, Single Data Stream
SMP	=	Symmetric Multiprocessor
MPP	=	Massively Parallel Processing
NUMA	=	Nonuniform Memory Access

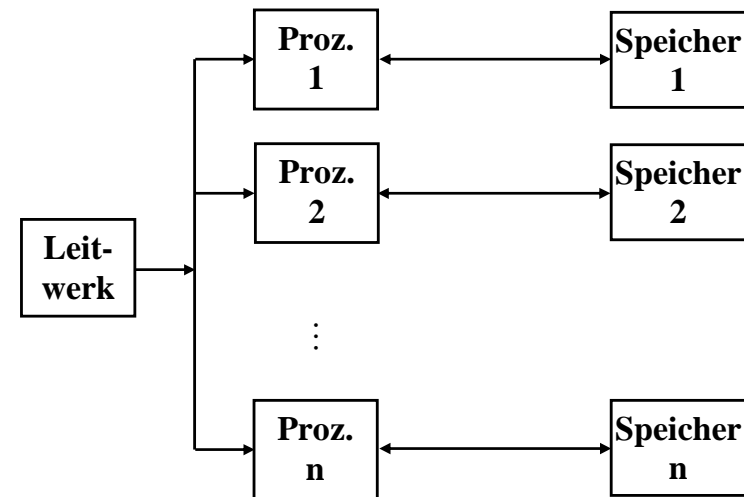
Bemerkung: Zu unterscheiden von der Hardware-Struktur ist das Programmiermodell, es gibt das Botschaften- und das Wandtafelmodell.

Beispiele zu Rechnerarten:

(i) SISD

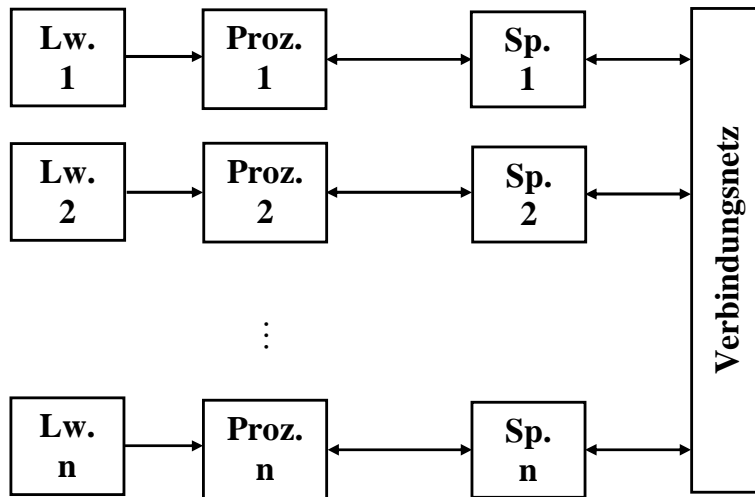


(ii) SIMD



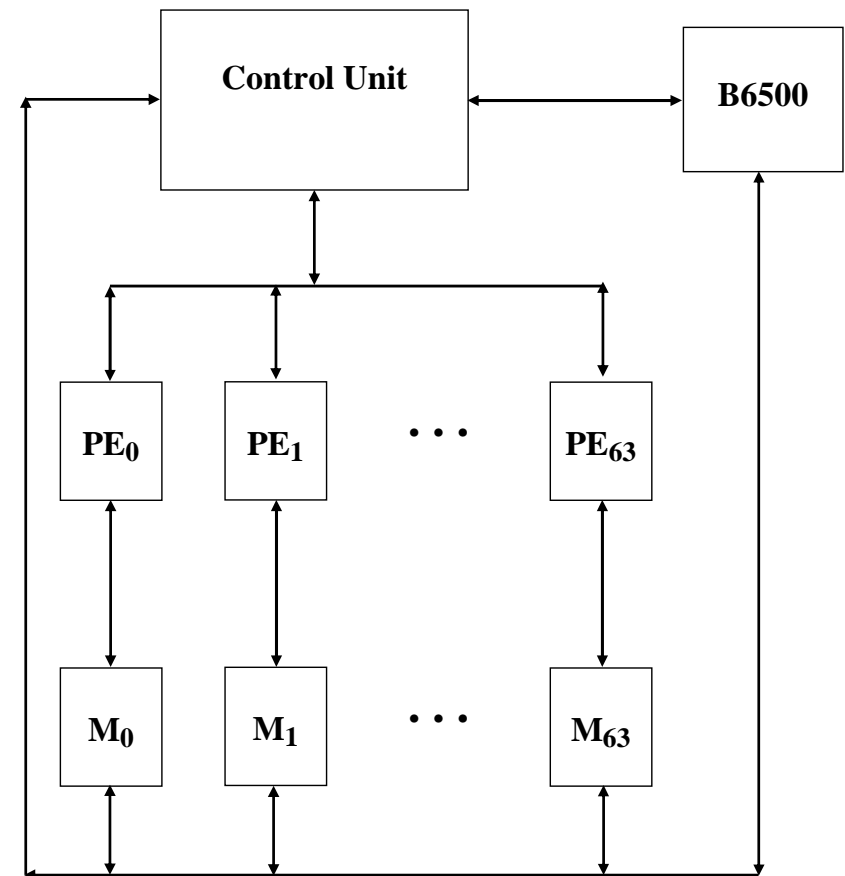
Bemerkung: Im allgemeinen sind die einzelnen Speicher eines SIMD-Rechners durch ein Transportnetz verbunden.

(iii) MIMD



Beispiele für COMA-Rechner sind KSR1 und KSR2 von Kendall Square Research. In diesem Fall wird der gesamte verteilte Hauptspeicher virtualisiert und in Seiten verwaltet. Es existiert keine starre Bindung zwischen Seiten und den Orten ihrer Speicherung, die Zuordnung von Seiten zu lokalen Speichern wird in Verzeichnissen geführt. Bei Anforderung eines Datums, das nicht im lokalen Speicher residiert, wandert die zugehörige Seite oder eine Kopie in den lokalen Speicher. Ein MESI-ähnliches Protokoll mit den Zuständen exclusive, copy, invalid und non-exclusive sorgt für Datenkonsistenz.

Blockbild der ILLIAC IV (= Illinois Array Computer IV):



Bemerkung: Die Fortran Schleife
DO 10 I = 1, 64
10 A(I) = B(I) + C(I)
wird in drei Assemblerbefehle übersetzt:
LDA ALPHA + 2
ADRN ALPHA + 1
STA ALPHA