Überblick

- Einleitung
  - Lit., Motivation, Geschichte, v.Neumann-Modell, VHDL
- Befehlsschnittstelle
- Mikroarchitektur
- Speicherarchitektur
- Ein-/Ausgabe
- Multiprozessorsysteme, ...

Kap. 6
Multiprozessorsysteme

6.1 Einleitung/Klassifikation
6.2 Verbindungsstrukturen
6.3 Beispiele
6.4 Leistungsbewertung
6.5 Cache-Kohärenz

Kap. 6.1
Einleitung/Klassifikation

Einsatz der zusätzlich verfügbaren Chipfläche

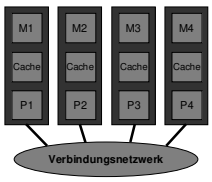
- Parallelität auf Bitebene: bis etwa 1985
  - **Kombinatorische Addierer** und **Multiplizierer**, etc.
  - → *wachsende Wortbreite auf 64 Bit*
- Parallelität auf Instruktionsebene: 1985 bis heute
  - **Pipelining** der Instruktionsverarbeitung
  - **Mehrere Funktionseinheiten** (→ *superskalare Prozessoren*)
- Integration von Caches und Hauptspeicher auf die Chipfläche: 1990 bis heute
  - Zur Verringerung der mittleren Zugriffszeiten
  - → *DEC Alpha 21164: 77% der Fläche für Caches*
- Parallelität auf Prozessorebene

Performanz von Rechnern läßt sich durch Parallelverarbeitung steigern.

- **Verteilten Systemen**
  Diese bestehen aus mehreren Prozessoren ohne gemeinsamen Speicher. Jeder Prozessor hat einen eigenen Speicher (local memory). Die Prozessoren tauschen Daten durch Nachrichten aus.
  Nachteil: Evtl. hoher Aufwand für Nachrichtenaustausch.
- **Parallelrechnern**
  Diese bestehen ebenfalls aus mehreren Prozessoren und haben einen gemeinsamen Speicher (shared memory). Die Kommunikation erfolgt über den gemeinsamen Speicher.
  Nachteil: Evtl. Performanz-Probleme, wenn viele Prozessoren gleichzeitig auf den Speicher zugreifen wollen. Konsistenzprobleme beim Einsatz von lokalen Caches

Architektur von verteilten/parallelen Systemen

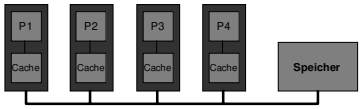
- Systeme mit Verbindungsnetzwerk und lokalem Speicher



- Zugriff auf fremden lokalen Speicher nicht möglich
- Kommunikation durch Austausch von Nachrichten
- gute physikalische Skalierbarkeit

Architektur von verteilten/parallelen Systemen ff

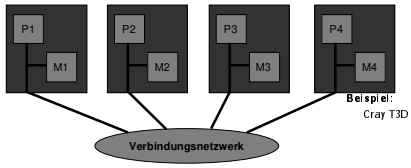
- Symmetrische Multiprozessoren (SMP = shared memory processing)
  - gemeinsamer nichtverteilter Speicher



  - meist busbasiert mit physikalisch gemeinsamem Speicher
  - Reduktion der Speicherzugriffslatenz durch lokale Caches
  - UMA (Uniform-Memory-Access-Modell)
  - **Problem**: Cache-Kohärenz
  - **Beispiele**: SGI Challenge, Sun Enterprise

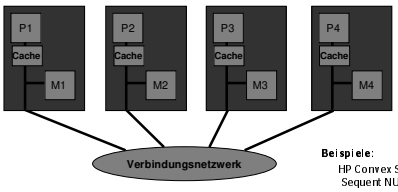
Architektur von verteilten/parallelen Systemen ff

- Verteilter gemeinsamer Speicher (DSM=distributed shared memory)
  - lokale Speichermodule
  - gemeinsamer Adreßraum
  - NUMA (Non-Uniform-Memory-Access-Modell)



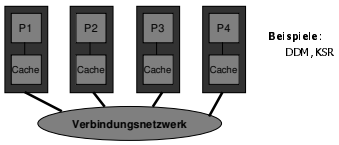
Architektur von verteilten/parallelen Systemen ff

- Verteilter gemeinsamer Speicher mit lokalem Cache
  - lokale Speichermodule
  - gemeinsamer Adreßraum
  - Zugriff der Prozessoren über prozessoreigenen Cache
  - cc-NUMA (cache coherent NUMA)



Architektur von verteilten/parallelen Systemen ff

- Cache-Only-Memory-Access-Modell (COMA) Maschinen
  - es gibt keinen Hauptspeicher, sondern nur Cachespeicher



  - **Problem**: Cache-Kohärenz, Auffinden von Daten in fremden Caches

Unterteilung von paralleln/verteilten Systemen

nach dem verwendeten Speichermodell:
- verteilter Speicher, getrennte Adressräume
- verteilter Speicher, gemeinsamer Adressraum
- Nichtverteilter Speicher, gemeinsamer Adressraum

nach der Homogenität der Prozessoren
- **homogene Parallelrechner**: alle Prozessoren sind gleich
- **heterogene Parallelrechner**: Prozessoren dürfen sich hardwaremäßig unterscheiden

Nach der Hierarchie zwischen den Prozessoren
- **symmetrische Parallelrechner**: die Prozessoren sind bzgl. ihrer Rolle im System untereinander austauschbar
- **nichtsymmetrische Parallelrechner**: es gibt Masters und Slaves

Nach der Eigenständigkeit der Prozessoren
- **lose gekoppelte Parallelrechner**: Netzwerk von eigenständigen Rechnern
- **eng gekoppelte Parallelrechner**: physikalisch ein Rechner

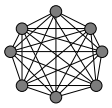
Bsp.: Flynn's Schema ´66

- **Einteilung von Rechnern in Klassen:**
  - **SISD** (**S**ingle **I**nstruction stream, **S**ingle **D**ata stream)
    → Personal Computer
  - **SIMD** (**S**ingle **I**nstruction stream, **M**ultiple **D**ata streams)
    → Vektorrechner z.B. Cray-1 (1976), Cray-2 (1985)
    ein Befehl wird auf ein Feld (Vektor) von Daten angewendet
    (Vektorpipeline, komplexe Simulationen →viele FP-Berechnungen)
    → Feldrechner z.B. Illiac IV (1972, 64 PEs), CM-2 (1987, 65536 PEs)
    Die gleiche Instruktion wird parallel auf einen Feld von Verarbeitungseinheiten angewendet.
  - **MISD** (**M**ultiple **I**nstruction streams, **S**ingle **D**ata stream)
    Diese Klasse ist leer!
  - **MIMD** (**M**ultiple **I**nstruction streams, **M**ultiple **D**ata streams)
    → INMOS Transputer, CONVEX SPP, CRAY T3D/T3E, IBM SP2

Kap. 6.2
Verbindungsstrukturen

Verbindungsstrukturen

- Der **Kommunikationsaufwand** zwischen den Prozessoren ist einer der Hauptpunkte für die Leistung des parallelen/verteilten Systems.
  ⇒ nicht jedes Problem ist für Parallelisierung geeignet.
- Verschiedene Kommunikationsstrukturen unterscheiden sich hinsichtlich ihrer Kosten und ihrer Leistung.



Beispiel:
Vollständiger
Verbindungsgraph

Modellierung von Verbindungen

- Die Topologie eines Parallelrechners/Netzwerkes wird durch einen abstrakten Graphen G=(V,E) dargestellt, mit
  - V = { 1, ... , n } Menge der Knoten, d.h. der Prozessoren bzw. Schaltelemente
  - E $\subseteq$ { {a,b}; a,b $\in$ V }, die Menge der Kanten, d.h. der Verbindungen

Charakteristika einer Verbindungsstruktur

- **Komplexität** oder **Kosten:**
  Hardware- Aufwand für das Verbindungsnetz gemessen in der Anzahl und der Art der Schaltelemente und Verbindungsleitungen.
- **Verbindungsgrad**
  eines Knotens ist definiert als die Anzahl der Verbindungen, die von dem Knoten zu anderen Knoten bestehen.
- **Diameter** oder **Durchmesser**:
  maximale Distanz für die Kommunikation zweier Prozessoren, also die Anzahl der Verbindungen, die durchlaufen werden müssen. Man spricht auch von der maximalen Pfadlänge zwischen zwei Knoten.

Charakteristika einer Verbindungsstruktur

- **Regelmäßigkeit** des Verbindungsmusters:
  Ein regelmäßiges Verbindungsmuster lässt sich meist besser implementieren.
- Notwendige **Leitungslängen**:
  Kurze Leitungslängen für alle Verbindungen eines Verbindungsnetzes sind vorteilhaft.
- **Blockierung:**
  Ein Verbindungsnetz heißt **blockierungsfrei,** falls jede gewünschte Verbindung zwischen den Prozessoren oder zwischen den Prozessoren und Speichern unabhängig von schon bestehenden Verbindungen hergestellt werden kann.

Charakteristika einer Verbindungsstruktur

- **Erweiterbarkeit** :
  Multiprozessoren können begrenzt, stufenlos oder nur durch Verdopplung der Anzahl der Prozessoren erweiterbar sein.
- **Skalierbarkeit** :
  Fähigkeit, die wesentlichen Eigenschaften des Verbindungsnetzes auch bei beliebiger Erhöhung der Knotenzahl beizubehalten.

Charakteristika einer Verbindungsstruktur

- **Ausfallstoleranz** oder **Redundanz** :
  Verbindungen zwischen Knoten sind selbst dann noch zu schalten, wenn einzelne Elemente des Netzes (Schaltelemente, Leitungen) ausfallen. Ein fehlertolerantes Netz muss also zwischen jedem Paar von Knoten mindestens einen zweiten, redundanten Weg bereitstellen. Die Eigenschaft eines Systems, bei Ausfall einzelner Komponenten unter deren Umgehung funktionstüchtig zu bleiben, wenn auch mit verminderter Leistung, wird als *Graceful degradation* bezeichnet.

Charakteristika einer Verbindungsstruktur

- **Durchsatz** oder **Übertragungsbandbreite**:
  Die maximale Übertragungsleistung des Verbindungsnetzes oder einzelner Verbindungen, meist in Megabits pro Sekunde (MBit/ s).
- **Komplexität der Pfadberechnung** oder **Wegefindung**: die Art, wie der Weg einer Nachricht vom Sender- zum Zielknoten berechnet wird. Die Wegefindung sollte einfach sein, um mittels eines schnellen Hardware- Algorithmus in jedem Verbindungselement implementierbar zu sein. Zu einer Verbindungsstruktur kann es mehrere Wegefindungsalgorithmen geben.
  Unterscheidungsmerkmale

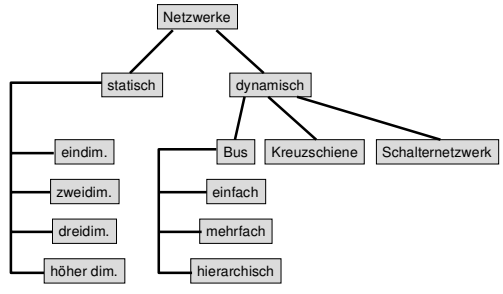
Charakteristika einer Verbindungsstruktur

- Bei **statischen Netzen** existieren fest installierte Verbindungen zwischen Paaren von Netzknoten.
- **Dynamische Netze** enthalten eine Komponente „Schaltnetz", an die alle Knoten über Ein- und Ausgänge angeschlossen sind. Direkte fest installierte Verbindungen zwischen den Knoten existieren nicht.

Charakteristika einer Verbindungsstruktur

- Durchmesser(G)
  $= \max_{v,w \in V} \{ \text{Länge des kürzesten Pfades von v nach w} \}$
- Grad(G) $= \max_{v \in V} | \{ w \in V;\ \{v,w\} \in E \} |$
- Anzahl der physikalischen Verbindungen(G) = | E |
- minimale Bisektionsbreitee:
  Teilt man das Netz in zwei (annähernd) gleich große Hälften A und B, so daß die Anzahl der Kanten e zwischen A und B minimal ist, so wird e als minimale Bisektionsbreitebezeichnet
- Diskonnektivität d = n/e
  n = Anzahl der Prozessorknoten, e = minimale Bisektionsbreite
  Schlimster Fall: alle Knoten in A senden eine Nachricht an Knoten in B und umgekehrt
- Kosteneffektivität K(G) =
  Grad(G) * max(Durchmesser(G),Diskonnektivität(G))

Klassifikation von Verbindungsstrukturen



Statische Verbindungsstrukturen, 1-dim

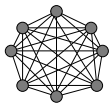
- **Kette**

Großer Durchmesser sehr fehleranfällig

- Durchmesser = n-1
- Grad = 2
- |Verbindungen| = n-1
- e = 1
- d = n
- K = 2n

Statische Verbindungsstrukturen, 2-dim

- **Vollständiger Verbindungsgraph**

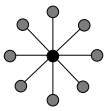


Zu teuer wegen dem großen Fanout

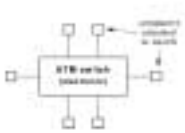
- Durchmesser = 1
- Grad = n-1
- |Verbindungen| = n(n-1)/2, keine „Kollisionen“
- e = $(n/2)^2$
- d = 4/n
- K = (n-1)*max(1,4/n)=n-1 n>4

Statische Verbindungsstrukturen, 2-dim

- **Stem**



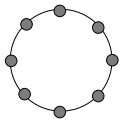
Zentraler Knoten ist Flaschenhals



- Durchmesser = 2
- Grad = n-1
- |Verbindungen| = n-1
- e = n/2
- d = 2
- K = (n-1)*max(2,2)=2*(n-1)

Statische Verbindungsstrukturen, 2-dim

- **Ring**



Beispiel für Ring: Token-Ring
Es kreist ein sogenanntes **Token** (spezielles Paket).
Ein Rechner darf nur dann senden, wenn er das Token besitzt.

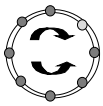
- Durchmesser = n/2
- Grad = 2
- |Verbindungen| = n
- e = 2
- d = n/2
- K = 2*max(n/2,n/2)=n

Beispiel für Ringtopologie : CDDI/FDDI-Ring

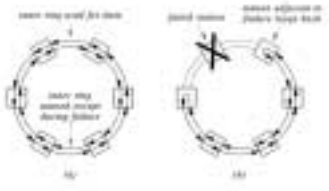
[Copper / Fiber Distributed Data Interconnect]

Charakteristika

- Ring-Topologie
- Besteht aus zwei gegenläufigen Ringen
  → **Fehlertolerantes Netz**
- ... ansonsten wie beim Token-Ring

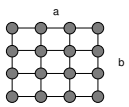


CDDI/FDDI-Ring ist fehlertolerant!



Statische Verbindungsstrukturen, 2-dim

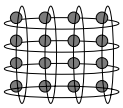
- **2D Gitter**



- Durchmesser = a+b-2
- Grad = 4
- |Verbindungen| = a(b-1)+b(a-1) = 2ab - a - b
- e = min(a,b)
- d = ab/min(a,b)
- K = 4*(a+b-2) a>1, b>1

Statische Verbindungsstrukturen, 2-dim

- **MESH** (torusähnliches Gitter)



Typischer Vertreter war das Transputer-Netz von INMOS

- Durchmesser = $n^{1/2}$
- Grad = 4
- |Verbindungen| = 2n
- e = $2n^{1/2}$
- d = $n^{1/2}/2$
- K = $4*n^{1/2}$

Statische Verbindungsstrukturen, 3-dim

- **Hypercube** (d-dimensionaler Würfel)

- Durchmesser = log n
- Grad = log n
- |Verbindungen| = (n log n) / 2
- e = n/2
- d = 2
- K = $(\log n)^2$

e-Cube-Routing

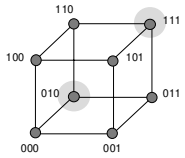
- Die Knotennummern werden als Binärzahlen geschrieben, dadurch unterscheiden sich benachbarte Knoten in genau einer Stelle, die zudem die Richtung der Verbindung angeben kann.
- Eine einfache Wegewahl:
  die Bits in Start- und Zieladresse werden mittels einer XOR- Verbindung verknüpft und das Resultat bestimmt die möglichen Wege.

e-Cube-Routing

- „higher dimensions of channels until the destination is reached"
  Dimension eines Kanals =
  Bitposition von (Knoten# XOR Knoten#)
- Beispiel: A = (010) und B =(111)

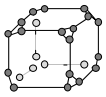
010 xor 111 = 101

Y-Achse X-Achse



Statische Verbindungsstrukturen, 3-dim

- **Cube Connected Cycle** (CCC)

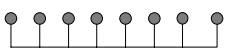


- d = Dimension
- r = Anzahl Knoten in Ringen
- häufig: r=d
- |Knoten| = $2^d*r = n$

- Durchmesser = (r/2)*d
- Grad = 3
- |Verbindungen| = $(d*d^d)/2 + 2^d*r$
- e = $2^d/2$
- d = 2r
- K = 3*(r/2)*d für d > 4, 6r für d < 4

Dynamische Verbindungsstrukturen

- **Bus**



Topologie versagt bei den heutigen Technologien bei grossem Datentransfer zwischen den Prozessoren

- Ein Bus läßt sich als Stern modellieren, wobei der zentrale Knoten aber kein Prozessor ist, sondern der zentrale Bus.
- Gleiches „Flaschenhalsproblem" wie bei Stern.

Beispiel für Bustopologie: Ethernet

Charakteristika

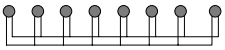
- Bus-Topologie
- 10 - 100 Mbit / Sekunde
- Paket-Versand mit Paketgrößen von *64-1518 Bytes*
- Nicht abhörsicher: *alle hören mit*!
- (Lokales) Rechnernetz über eine **Bridge** mit der Aussenwelt verbunden, die die Nachrichten filtert und verstärkt

Übertragungsvorgang

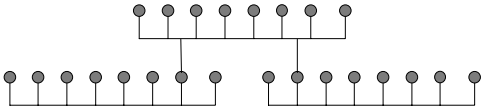
- Nachrichten werden in Pakete fester Länge zerteilt. Jedes Paket enthält **Headerinformation** mit *Zieladresse* und *Sequenznummer*
- Jeder Rechner horcht am Bus und empfängt die Pakete, die seine Adresse tragen
- Kollisionen von mehreren Sendern werden erkannt.
- Falls Kollision, dann später erneuter Sendeversuch

Alternative Busstrukturen

- Mehrfachbus



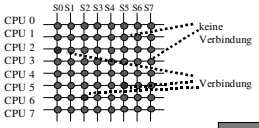
- Hierarchischer Bus



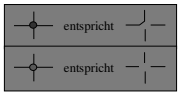
Dynamische Verbindungsstrukturen ff

- **Crossbar Switch**

meist verwendete Struktur bei Parallelrechnern mit gemeinsamen "nichtverteilten" Speicher



Nachteil: n·m Crosspoints



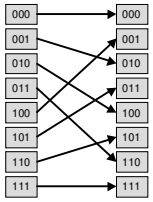
Permutationsnetzwerke

- Vermeidung des hohen Aufwandes von Kreuzschienen
- Aufgebaut aus Zweierschaltern (2 Zustände = 1 Bit):

- Permutation: p Eingänge werden parallel auf p Ausgänge geschaltet (Eingänge werden permutiert)
- Einstufige, mehrstufige oder rückgekoppelte Netze
- Reguläre/irreguläre Permutationsnetzwerke

Permutationsnetzwerke

- Mischpermutation (perfect shuffle)

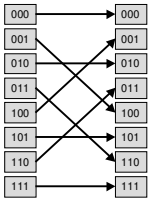


Perfect shuffle:
PS(n) = 2 * n for n < N/2
PS(n) = 2 * n - N + 1 for n >= N/2

$M(a_n,\ldots,a_1) = a_{n-1}, \ldots, a_1, a_n$

Permutationsnetzwerke

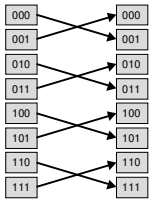
- Kreuzpermutation



$K(a_n,\ldots,a_1) = a_1, a_{n-1}, \ldots, a_2, a_n$

Permutationsnetzwerke

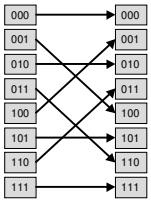
- Tauschpermutation



$T(a_n,...,a_1) = a_n, a_{n-1}, ..., a_2, \overline{a_1}$

Permutationsnetzwerke

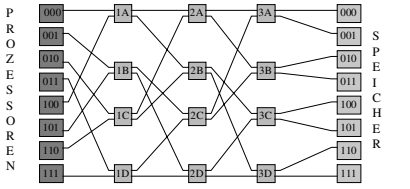
- Umkehrpermutation (Butterfly)



$U(a_n,...,a_1) = a_1, a_2, ..., a_{n-1}, a_n$

Dynamische Verbindungsstrukturen ff

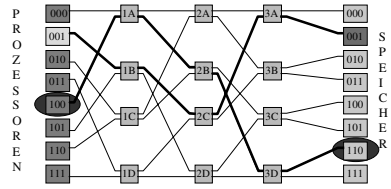
- **Omega Netzwerk** (log n Stufen von Mischpermutationen)



... besitzt nur n/2 log n Schalter

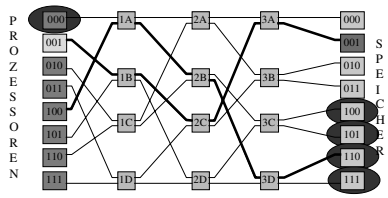
Dynamische Verbindungsstrukturen ff

- **Omega Netzwerk**
  - der obere Ausgang eines Schalters ist der 0-Ausgang
  - der untere Ausgang eines Schalters ist der 1-Ausgang
  - ein Schalter der Stufe i schaltet gemäß dem iten Bit der Zieladresse

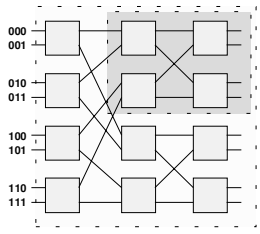


Dynamische Verbindungsstrukturen ff

- **Omega Netzwerk**
  - nicht jede Kommunikation ist gleichzeitig möglich, auch wenn alle Zieladressen paarweise verschieden sind (**blocking network**)

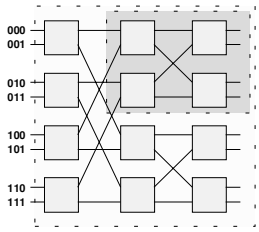


Delta-Netzwerk



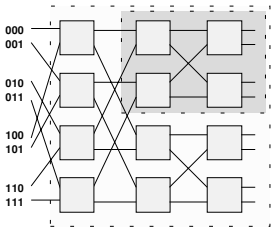
- Log n Stufen
- Mischpermutation zwischen den Stufen
- Zwischen den Stufen i und i+1 wird die Permutation nur auf $2^{(\log n - i+1)}$ Ein-/Ausgänge angewandt
- Eingänge werden ohne Permutation mit der ersten Stufe verbunden

Butterfly-Netzwerk



- Log n Stufen
- Umkehrpermutation zwischen den Stufen
- Zwischen den Stufen i und i+1 wird die Permutation nur auf $2^{(\log n - i+1)}$ Ein-/Ausgänge angewandt
- Eingänge werden ohne Permutation mit der ersten Stufe verbunden

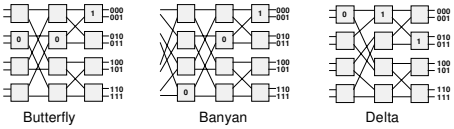
Banyan-Netzwerk



- log n Stufen
- Umkehrpermutation zwischen den Stufen
- Zwischen den Stufen i und i+1 wird die Permutation nur auf $2^{(\log n - i+1)}$ Ein-/Ausgänge angewandt
- Eingänge werden per Mischpermutation mit der ersten Stufe verbunden

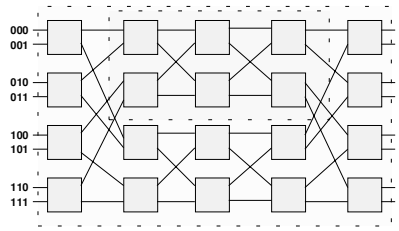
Self-Routing Property

- Zieladresse liefert Wegeinformation
- jede Stufe betrachtet das korrespondiesernde Bit der Zieladresse
  - '0' bedeutet, Nachricht an den oberen Ausgang
  - '1' bedeutet, Nachricht an den unteren Ausgang



Beneš Netzwerk

- 2 log n -1 Stufen
- Zusammengesetzt aus zwei gespiegelte Butterfly-Netzwerken



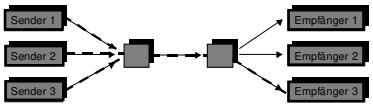
Eigenschaften des Benes Netzwerkes

- Vorteil
  - Jede Permutation kann konfliktfrei geschaltet werden
  - Achtung: zwei Butterfly-Netzwerke in Serie haben nicht diese Eigenschaft!
- Nachteil:
  - die Pfadbestimmung ist sehr komplex und muß deshalb off-line durchgeführt werden

Vermittlungsarten

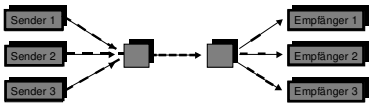
- **Paket-vermittelnde Vermittlung**
  - Nachricht wird in ein oder mehrere Pakete verpackt
  - Jedes der Pakete enthält die Adresse des Empfängers
  - Es wird kein Pfad vom Sender zum Empfänger freigeschaltet
  - Das Paket wird in Abhängigkeit der Empfängeradresse immer nur zu einem direkten Nachbarn geschickt.
- **Leitungsvermittelnde Vermittlung**
  - Es wird im Netzwerk ein Pfad vom Sender zum Empfänger geschaltet, über den alle Nachrichten geschickt werden (Bsp: Telefonverbindung)

Leitungsvermittelnde Vermittlung



- Schnelle Übertragung grosser Datenmengen
- Geschalteter Pfad blockiert andere Verbindungen

Paket-vermittelnde Vermittlung



- Keine Verbindung muss lange warten
- Unterbrechungen während einer Übertragung möglich

Adressierungsarten

- **zielbasiert** (*destination-based routing*):
  Kopfteil eines Pakets (oder einer Nachricht) wird mit einer systemweit eindeutigen Empfängeradresse versehen, die bei der Wegefindung von jedem Zielknoten zur Auswahl eines Übertragungskanals genutzt wird
- **quellenbasiert** (*source-based routing*):
  Paket wird mit allen Informationen versehen, um über die Zwischenknoten zum Empfänger zu gelangen. Für jeden Zwischenknoten wird im voraus die Abzweigung bestimmt, die das Paket nehmen muss.

Wegewahl

- **Deterministische Wegewahl:**
  Nachrichten zwischen zwei Knoten müssen immer den gleichen Weg gehen
  - Vorteil: einfachen Pfadberechnung
  - Nachteile: erhöhten Möglichkeit zu Blockierungen, Mangel an Fehlertoleranz.
- **Adaptive Wegewah**l:
  Möglichkeit auch andere Wege zu nehmen,
  - Nachteil: Etwas höheren Hardware-Aufwand.
  - Vorteil: Blockierte Strecken und ausgefallene Knoten oder Schaltelemente können umgangen werden.

Kommunikationskontrolle

- deterministische Kontrolle
  der Weg eines jeden Pakets ist reproduzierbar
- randomisierte Kontrolle
  an bestimmten Stellen des Algorithmus werden zufällige Entscheidungen getroffen
  - Beispiel: Valiant-Paradigma
    - Route zu einer zufälligen Zwischenadresse
    - Route dann erst zum Ziel

Phit und Flusskontrolle

- Eine Nachricht selbst wird in eine Anzahl von **Übertragungseinheiten** (*phits* – *physical transfer units* – oder auch *flits* – *flow control digits* – genannt) zerlegt.
- Ein ***Phit*** ist dabei die Datenportion, die zu einem Zeitpunkt zwischen zwei Knoten übertragen werden kann.
- Bei der Nachrichtenübertragung zwischen nicht benachbarten Sender- und Empfängerknoten sind Puffer nötig.

Phit und Flusskontrolle

- Bei der Zwischenspeicherung muss der Knoten dafür sorgen, dass sein Puffer nicht überläuft.
- Dies geschieht durch eine zusätzliche **Fluss-Steuerung** (*flow control*) z.B. über
  - spezielle Leitungen (z. B. *ack* - oder *strobe*-Leitung) vom empfangenden (Zwischen-)Knoten zum sendenden (Zwischen-) Knoten

Übertragungsmodi

- **Store-and-forward-Modus**:
  Nachricht wird von jedem Zwischenknoten in Empfang genommen, vollständig zwischengespeichert und dann weiter übertragen

Übertragungsmodi

- **Virtual-cut-through-Modus**:
  - Nachricht wird als Kette von Phits transportiert.
  - Der Kopfteil der Nachricht enthält die Empfängeradresse und bestimmt den einzuschlagenden Weg.
  - Bei Ankunft des ersten Phits an einem Zwischenknoten wird diese sofort an den nächsten Knoten weitergeleitet, falls der Pfad dorthin frei ist.
  - Alle anderen folgen ähnlich wie bei einer Pipeline- Verarbeitung.
  - ankommende Daten werden *nur im Konfliktfall* im Knoten vollständig zwischengespeichert
  - In jedem Knoten werden Puffer bereit gehalten, die auch ein maximal großes Nachrichtenpaket zwischenspeichern können

Übertragungsmodi

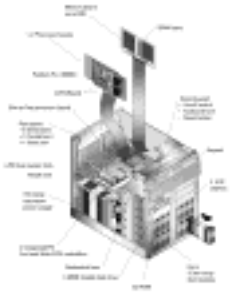
- **Wormhole-routing-Modus**:
  - solange keine Übertragungskanäle blockiert sind, mit den Virtual-cut-through-Modus identisch.
  - Falls der Kopfteil der Nachricht auf einen Kanal trifft, der gerade belegt ist, wird er abgeblockt.
  - Alle nachfolgenden Übertragungseinheiten der Nachricht verharren dann ebenfalls an ihrer augenblicklichen Position, bis die Blockierung aufgehoben ist.
  - Durch das Verharren werden die Puffer nachfolgender Kanäle auch für weitere Nachrichten blockiert.

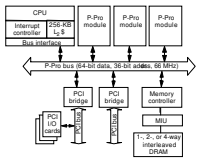
Übertragungsmodi

- **Buffered wormhole routing:**
  - Kompromisslösung zwischen Virtual-cut-through- und Wormhole-routing-Modus
  - begrenzter Puffer zur Aufnahme kleinerer Pakete vorhanden
  - größere Pakete werden im Blockierungsfall ähnlich dem Wormhole-routing-Modus in den Puffern mehrerer Knoten zwischengespeichert.

Kap. 6.3
Multiprozessorsysteme: Beispiele

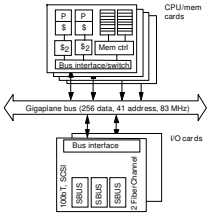
Intel Pentium Pro Quad





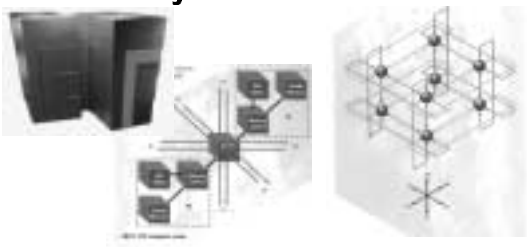
- Bis zu 4 Prozessoren
- 66 MHz Busverbindung mit bis zu 528 MB/sek
- Koherenz- und Multiprozessorlogik im Prozessor integriert

SUN Enterprise



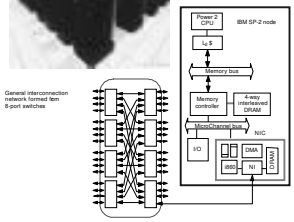
- Bis zu 16 Einsteckkarten: (bis zu 4) Prozessor + Speicher oder I/O-Karte
- 100 MHz Bus mit bis zu 2GB/sek

Cray T3E



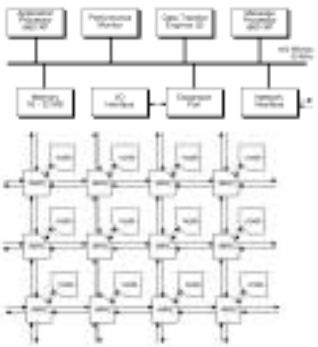
- Bis zu 2048 Prozessoren, 480MB/s Verbindungen
- DSM: Memory controller generiert Kommunikationsanforderungen für nichtlokale Speicherzugriffe
- Spitzen-Bisektionsbandbreite: 166GB/sec (512 Prozessoren)

IBM SP-2



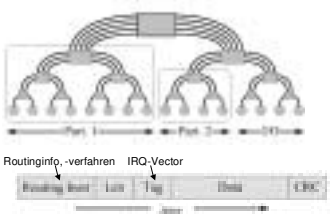
- 2 bis 512 Knoten
- Jeder Knoten ist eine vollständige RS6000 Workstation
- Verbindungsnetz:
  - 4x4 Crossbars
  - 8 Crossbars = 16x16 Verbindungsnetz

Intel Paragon



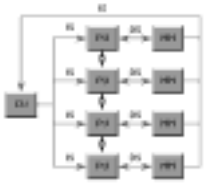
- 2D Gitternetzwerk
- Wormhole routing
- größtes System mit 1984 Knoten

Connection Machine: CM5



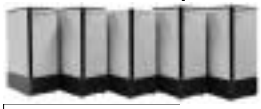
- 32 - 16384 Prozessoren
- message-passing, distributed memory
- Control-, Data- und Diagnostic-Network
- MIMD, SIMD möglich durch Control-Network

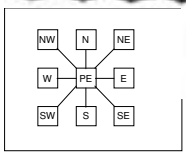
ILLIAC 4 (SIMD)

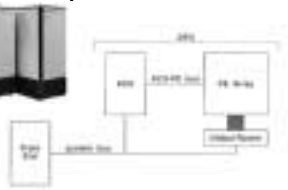


- 1 "control unit" (CU) mit Zugriff auf den Speicher steuert 64 Prozessoreinheiten (PUs)
- jede PE hat lokalen Speicher
- jede PE teilt Speicher mit benachbarten PUs
- 2D Gitterstruktur

MasPar MP2 (SIMD)







- Bis zu 16.384 Prozessorelemente
- Jeder Knoten ist mit seinen 8 Nachbarn verbunden

Kap. 6.4 Leistungsbewertung

Leistungsbewertung

- Leistung eines Rechners läßt sich unterschiedlich definieren.
  - Antwortzeit
    - Zeit für ein Programm.
  - Durchsatz
    - Zahl der gleichzeitig verarbeiteten Programme.
- Sehr einfaches Leistungsmaß sind MIPS.
- Andere Form der Leistungsmessung ist der Leistungsvergleich
  - Betrachtung eines konkreten Programms
  - Benchmarks

Leistungsbewertung durch MIPS

- Vorteil
  - sehr einfaches und leicht verständliches Maß.
- Nachteil
  - Schnellere Rechner können niedrigere MIPS-Zahl haben.
  - Hängt vom Befehlssatz des Rechners ab.
  - Speicher wird nicht berücksichtigt.
  - Betriebssystem wird nicht berücksichtigt.

Leistungsbewertung durch Benchmarks

- Es werden reale Programme auf den Rechnern ausgeführt.
- Dadurch werden Speicher, I/O, Betriebssystem, Compiler, etc. berücksichtigt.
- Um aussagekräftige Vergleiche erhalten zu können, müssen typische Anwendungen verglichen werden.
- Solche typische Anwendungen werden in Benchmarks zusammengefaßt:
  - SPEC-Benchmarks (www.specbench.org)
  - TPC-Benchmarks
  - DIN-Benchmarks
  - usw.

Vergleich einiger Architekturen

	DEC Alpha21262	MIPS R10000	IBM PPC750	SUN UltraSparc	PII Klamath	PII Xeon	AMD K6	AMD K6-3D
Taktfrequenz	667	250	400	333	300	400	300	350
SPECint95	44	14.7	17.6	14.2	11.9	16.5	?	?
SPECfp95	66	24.5	12.2	16.9	8.6	13.7	?	?
Transistoren (Mio)	15.2	6.8	6.35	5.4	7.5	7.5	8.8	9.3
Leistungsverbrauch (W)	72	>30	5.7	<30	?	23.3	?	?

Quelle: Rosenstiel, 1999

Vergleich einiger Architekturen

Hersteller Prozessor	Intel P4	AMD Athlon	Sun UltraSparc III
Taktfrequenz in GHz SPECint2000 SPECfp2000	2,53 922 901	1,733 749 660	0,9 533 713

Quelle: www.specbench.org

Leistungsbewertung von Parallelen Programmen

- MIPS: gibt die Leistung des Rechners an
- Benchmarks: Leistung einer Programmsuit auf einem Rechner/Speicher/Betriebssystem
- Wie mißt man den Gewinn durch den Einsatz von Parallelrechnern?

Speed-Up

- Definitionen: Gegeben ein Programm
  - P(1) = Zahl der auszuführenden Operationen auf Monoprozessorsystem
  - P(n) = Zahl der auszuführenden Operationen auf Multiprozessorsystem
  - T(1) = Ausführungszeit auf Monoprozessorsystem in Schritten
  - T(n) = Ausführungszeit auf Multiprozessorsystem in Schritten
- Speedup nach Lee (Leistungssteigerung):

$$S(n) = \frac{T(1)}{T(n)}$$

nach Lee gilt: P(1) = T(1)

Was ist eine gute Beschleunigung?

- Hoffentlich: S(n) > 1
- Linear speedup:
  - S(n) = n
  - Programm skaliert perfekt mit der Zahl der Prozessoren
- Superlinear speedup:
  - S(n) > n
  - Ist das möglich?
- Realität (Lee'sches Prinzip): $1 \leq S(n) \leq n$

Alternative Definitionen

- Schnellster bekannter Algorithmus der auf einer Einprozessormaschine ausgeführt wird

$$S(n) = \frac{T_s}{T_p}$$

- Schnellster bekannter Algorithmus der auf einer einem Prozessor der Multiprozessormaschine ausgeführt wird(Gustafson)

$$S(n) = \frac{T^*(1)}{T^*(n)}$$

Kann Speedup superlinear sein?

- Antwort: Ja und Nein, es hängt von der Definition ab
- 1. Antwort: NEIN, Speedup kann nicht superlinear sein!
  - Sei M eine parallele Maschine mit n Prozessoren
  - Sei T(X) die Zeit um das Problem mit x Prozessoren auf M zu lösen
  - Speedup definition: $S(n) = \frac{T(1)}{T(n)}$
  - Nimm an, daß ein paralleler Algorithmus A eine Instanz des Problems in t Zeitschritten löst
  - Dann kann A das gleiche Problem in nt Zeitschritten durch Zeitmultiplexing lösen
  - Die beste serielle Zeit kann nicht gößer sein als nt
  - Also kann der Speedup nicht größer sein als n

$$S(n) = \frac{T(1)}{T(n)} \leq \frac{nt}{t} = n$$

Kann Speedup superlinear sein?

- Antwort 2: Speedup kann superlinear sein!
  - Sei M eine parallele Maschine mit n Prozessoren
  - Sei T(X) die Zeit um das Problem mit x Prozessoren auf M zu lösen
  - Speedup definition: $S(n) = \frac{T_s}{T(n)}$
  - Die serielle Version kann mehr Overhead erzeugen als die parallele Version
    - Z.B. A=B+C auf SIMD Maschine mit A,B,C Matritzen, versus Schleifenoverhead einer seriellen Maschine
    - unterschiedliche Algorithmen, z.B. Tiefensuche bei serieller Maschine, Breitensuche bei Parallelermaschine
  - Es werden unterschiedliche Algorithmen verglichen

Grenzen des Speedups

- Software Overhead
  z.B. zusätzliche Indexberechnungen um den Code auf mehrere Prozessoren zu verteilen
- Load Balancing
  Kann die Ausgabe gleichmäßig auf mehrere Prozessoren verteilt werden?
- Communication Overhead
  Wird Kommunikation vom Prozessor ausgeführt?
  Wie groß sind die Latenzzeiten?

Grenzen des Speedup

- f = Anteil des Programms, der nicht parallelisiert werden kann
  - z.B. Filezugriffe auf eine Platte
- Amdahls Gesetz:

$$T(n) = fT(1) + (1-f)\frac{T(1)}{n}$$

$$S(n) = \frac{T(1)}{fT(1) + \frac{(1-f)T(1)}{n}} = \frac{n}{nf + 1 - f} = \frac{1}{\frac{(n-1)f + 1}{n}}$$

$$\lim_{n \to \infty} = \frac{1}{f}$$

Beispielanwendung von Amdahls Gesetz

- Algorithmus hat 4% seriellen Code, wie groß ist der maximal erreichbare Speedup mit 16 Prozessoren?
  - 16/(1 + (16 – 1)*.04) = 10
- Wie groß ist der maximale Speedup?
  - 1/0.04 = 25

Andere Leistungskennzahlen

- Effizienz = relative Verbesserung in der Verarbeitungsgeschwindigkeit

$$E(n) = \frac{S(n)}{n} \qquad \frac{1}{n} \leq E(n) \leq 1$$

- Redundanz = Mehraufwand (im Code) durch Parallelverarbeitung

$$R(n) = \frac{P(n)}{P(1)} \qquad 1 \leq R(n)$$

- Parallelindex = Anzahl der parallelen Operationen pro Zeiteinheit

$$I(n) = \frac{P(n)}{T(n)}$$

Andere Leistungskennzahlen

- Auslastung = Operationen pro Zeiteinheit in jedem Prozessor $U(n) = \frac{I(n)}{n}$
- Qualität = qualitativ erzielte Leistungssteigerung $Q(n) = S(n)\frac{E(n)}{R(n)}$
- Es gilt

$$1 \le Q(n) \le S(n) \le I(n) \le n$$

$$\frac{1}{n} \le E(n) \le U(n) \le 1$$

Kap. 6.5 Cache-Kohärenzprotokolle

Prinzipien
Standard-Protokolle
MESI-Protokoll

Multiprozessor-Cache-Kohärenz

- Zur Sicherstellung der Kohärenz von Cache und Hauptspeicher gibt es sogenannte cache coherency protocols
- zwei Prinzipien:
  - directory based: Information über einen Block befindet sich nur an einer Stelle
    - Nachteil: Kohärenzinformation im Prinzip proportional zur Anzahl Blöcke im Hauptspeicher
  - snooping: Information über Blöcke befindet sich beim jeweiligen Block im jeweiligen Cache. Jeder Cache-Controller betreibt "Snooping" am Bus, um zu überwachen, was mit Block passiert
    - erfordert gemeinsamen Bus
    - üblich und wird im folgenden diskutiert

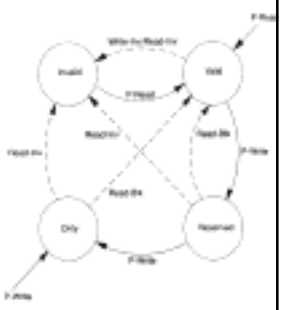
Snooping-Protokolle

- zwei Arten, die sich bei Write unterscheiden:
  - write invalidate: wenn ein Prozessor einen Block verändern will, macht er vorher alle anderen Kopien dieses Blocks ungültig
  - write broadcast: nach dem Ändern wird der geänderte Block an alle anderen Prozessoren geschickt, die ihre Kopie dann ggf. aktualisieren können
  - beide Protokolle haben ihre Berechtigung und werden auch verwendet

„Write-Once“ Cache Protokoll

Prinzip:
Cache-Block wird beim Ersetzen in den Speicher zurückgeschrieben, falls er, nachdem er in den Cache geschrieben wurde, verändert wurde (=Zustand "Dirty", s.u.)

4 Zustände
Invalid: inkonsistente Cache-Kopie
Valid: Cache-Kopie mit Speicher konsistent
Reserved: Cache-Kopie mit Speicher konsistent und einzige Kopie und Daten zum ersten Mal geschrieben
Dirty: einzige Kopie und Daten mehr als einmal verändert



...Erläuterung zum "write once" (Standard-) Protokoll

- Übergänge durch
  - Lesen und Schreiben durch jeweiligen Prozessor (P-Read, P-Write)
  - (normale) Speicher-Lese/Schreib-Operationen (Read-Blk, Write-Blk)
  - Konsistenz-Operationen, die nicht durch den Prozessor, sondern durch den gemeinsamen Bus ausgelöst werden (gestrichelte Übergänge):
    - Write-Invalidate (Write-Inv): macht alle anderen Cache-Kopien eines Blocks ungültig (invalid)
    - Read-Invalidate (Read-Inv): liest einen Block und macht alle anderen Cache-Kopien ungültig (invalid)

...Erläuterung zum "write once" (Standard-) Protokoll

- Treffer beim Lesen:
  - kann lokal im Cache abgewickelt werden
  - keine Zustandsübergänge erforderlich
- Miss beim Lesen:
  - keine Kopie im Dirty-Zustand:
    - Block wird aus Speicher gelesen
    - Zustand = Valid
  - Kopie im Dirty-Zustand:
    - Cache mit Dirty-Kopie verhindert, daß Speicher Kopie schickt, und schickt selbst seine Kopie zum anfordernden Cache
    - Speicher wird aktualisiert, Zustände beider Kopien = Valid

...Erläuterung zum "write once" (Standard-) Protokoll

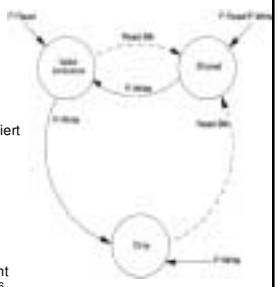
- Treffer beim Schreiben:
  - Kopie im Dirty- oder Reserved-Zustand:
    - Schreiben kann lokal erfolgen
    - neuer Zustand = Dirty
  - Kopie im Valid-Zustand:
    - Write-Invalidate wird ausgelöst
    - Speicher-Kopie wird aktualisiert
    - neuer Zustand = Reserved

...Erläuterung zum "write once" (Standard-) Protokoll

- Miss beim Schreiben:
  - Kopie kommt aus Speicher oder von anderem Cache mit Kopie im Dirty-Zustand
  - wird durch Read-Invalidate ausgelöst
  - Speicher wird aktualisiert
  - neuer Zustand = Dirty
- Ersetzen eines Blocks:
  - Zurückschreiben in den Speicher falls im Dirty-Zustand
  - sonst keine Aktion erforderlich

Firefly

Prinzip:
Firefly-Protokoll benutzt "copy-back" für private Blöcke und "write-through" für gemeinsame Blöcke, wobei über privat und gemeinsam erst zur Laufzeit entschieden wird
zur Implementierung wird zusätzliche Busleitung "shared" benutzt, die während des Snooping-Mechanismus Schreiber über tatsächliche Existenz weiterer Kopien informiert



3 Zustände
Valid-exclusive: einzige Cache-Kopie mit Speicher konsistent
Shared: Cache-Kopie mit Speicher konsistent und es existieren weitere konsistente Kopien
Dirty: einzige Kopie mit Speicher inkonsistent

...Erläuterung zum "Firefly" Write-Update-Protokoll

- Übergänge durch
  - Lesen und Schreiben durch jeweiligen Prozessor (P-Read, P-Write)
  - (normale) Speicher-Lese/Schreib-Operationen (Read-Blk, Write-Blk)
  - Konsistenz-Operation Write-Update:
    - alle anderen Cache-Kopien einschließlich des Speichers werden aktualisiert
    - falls Shared-Zustand bei anderen Kopien nicht mehr vorliegt, ist Shared-Busleitung nicht aktiv
      - wird z.B. ausgenutzt, um beim Schreiber Shared-Zustand durch Valid-exclusive-Zustand zu ersetzen

...Erläuterung zum "Firefly" Write-Update-Protokoll

- Treffer beim Lesen:
  - kann lokal im Cache abgewickelt werden
  - keine Zustandsübergänge erforderlich
- Miss beim Lesen:
  - keine Cache-Kopie existiert:
    - Block wird aus Speicher gelesen
    - Zustand = Valid-exclusive
  - eine Cache-Kopie im Dirty-Zustand:
    - Cache mit Dirty-Kopie verhindert, daß Speicher Kopie schickt, und schickt selbst seine Kopie zum anfordernden Cache
    - Speicher wird aktualisiert, Zustände beider Kopien = Shared

...Erläuterung zum "Firefly" Write-Update-Protokoll

- wenn mehrere Kopien im Shared-Zustand existieren:
  - diese Caches synchronisieren Übertragung und schicken die Kopie direkt zum anfordernden Cache
  - Zustand = Shared

...Erläuterung zum "Firefly" Write-Update-Protokoll

- Treffer beim Schreiben:
  - Cache-Kopie im Dirty- oder Valid-exclusive-Zustand:
    - Schreiben kann lokal erfolgen
    - neuer Zustand = Dirty
  - Kopie im Shared-Zustand:
    - Write-Update aktualisiert alle Kopien einschließlich Speicher
    - Zustand = Shared
    - Ausnahme: während Write-Update ist Shared-Busleitung nicht aktiv, der Shared-Zustand ist also bei den anderen Kopien zwischenzeitlich beendet worden
      - dann gilt: Zustand = Valid-exclusive

...Erläuterung zum "Firefly" Write-Update-Protokoll

- Miss beim Schreiben:
  - Kopie kommt aus Speicher oder von anderem Cache
    - falls Kopie vom Speicher kommt: neuer Zustand = Dirty
    - falls Kopie von anderem Cache kommt: Write-Update aktualisiert alle Kopien einschließlich Speicher: neuer Zustand = Shared
- Ersetzen eines Blocks:
  - Zurückschreiben in den Speicher falls im Dirty-Zustand
  - sonst keine Aktion erforderlich

"Dragon"-Protokoll

- Dragon-Protokoll für Dragon-Multiprozessor Workstation von Xerox vorgeschlagen
- verbessert Effizienz der Cache-Cache-Transfers dadurch, daß auf Speicher-Aktualisierung verzichtet wird
- Speicher wird erst beim Ersetzen eines Blocks aktualisiert

Pentium-MESI-Protokoll

- ist i.w. modifiziertes Write- Once-Protokoll
- Modified: einzige Cache-Kopie, nicht konsistent mit Haupspeicher, lesen und schreiben lokal
- Exclusive: einzige Cache-Kopie, nicht modifiziert und daher konsistent mit Hauptspeicher, lesen und schreiben lokal, schreiben führt zu Übergang in Modified-Zustand
- Shared: Kopie in anderen Caches, lesen lokal, schreiben führt zu write-through und invalidate der anderen Kopien
- Invalid: Cache-Block nicht verfügbar, Lesen/Schreiben führt u.U. zum Nachladen und zwar in Abänderung des Write-Once Prot.:

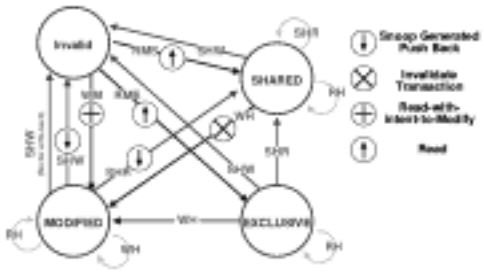
...Pentium-MESI-Protokoll

- Read Miss: Mögliche Folgezustände:
  - S: falls Block Read-Only
  - S: falls CPU-Write-Through Attribut aktiv
  - S: falls Kopie in anderem Cache existiert
  - M: falls modifizierte Daten direkt von anderem Cache kommen, Speicher wird nicht aktualisiert
  - E: sonst
- Write Miss: 2 Möglichkeiten direktes Schreiben in Speicher mit evtl. Invalidate anderer Kopie
  - ohne Nachladen des Cache
  - mit (folgendem) Nachladen des Cache:WriteMiss with Allocation

...Pentium-MESI-Protokoll

- Ergänzung um explizite Befehle für Invalidate-Zustand:
  - INVD: setzt interne und externe Caches in Invalidate-Zustand
    - i.w. für Testzwecke oder ähnliches geeignet
    - modifizierte Cache-Blöcke werden nicht in Speicher zurückgeschrieben
  - WBINVD: schreibt modifizierte Cache-Blöcke zurück bzw. veranlaßt Zurückschreiben und setzt Blöcke in Invalidate-Zustand

Zustandsübergänge beim MESI-Protokoll



Zustände des (Pentium-)MESI-Protokolls

Zustand Cache-Block	M modified	E exclusive	S shared	I Invalid
Cache-Block	gültig	gültig	gültig	ungültig
Speicher-Block	ungültig	gültig	gültig	–
weitere Cache-Kopien	nein	nein	möglich	möglich
Treffer beim Lesen	lokal	lokal	lokal	abh. von Cache Contr.
Treffer beim Schreiben	lokal	lokal	Update-Zyklus	geht zum Speicher

Cache-Aktualisierung beim Pentium

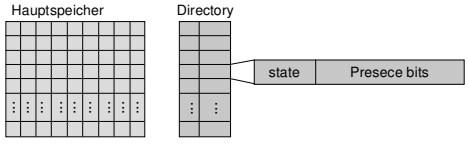
- Pentium ist bzgl. Aktualisierungsstrategie programmierbar
- Cache mit und ohne Hauptspeicherkonsistenz abschaltbar
- im Betrieb mit Cache Write-Back oder Write-Through wählbar:
  - Standard-Protokoll ist Write-Back, um gerade bei Multiprocessing Anzahl der Speicherzugriffe zu reduzieren
  - Write-Through kann manchmal erforderlich sein, z.B. für Frame Buffer, damit immer aktuelles Bild auf Schirm
  - seitenweise durch Software (PWT-Bit in Seitentabelle) oder Hardware (WB/WT#) kontrollierbar
- Besonderheit: I-Cache nicht beschreibbar

Directory-basierte Protokolle

- Snoopy-Protokolle
  - benötigen ein Broadcast Kommunikationsmedium (Bus)
  - beschränkte Zahl von Komponenten (bis zu 30)
- Directory-based Protokolle
  - werden in DSM-Systemen eingesetzt
  - anwendbar für große Systeme mit vielen Prozessoren

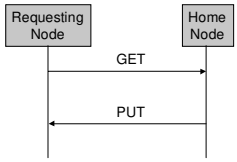
Bit-Vector-Protocol

- Für jede Cache-line existiert ein Eintrag in einer Tabelle (directory)
- Eintrag hät Information über
  - den Zustand der Cache-line (shared, clean, dirty)
  - die Knoten, die eine Kopie der Cache-line halten
    - als Bitvector, jedem Bit ist ein Knoten zugeordnet (presence bits)

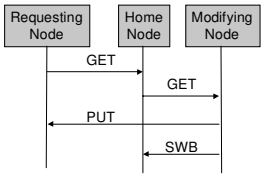


Bit-Vector-Protocol

- Read-Miss
  - Anfordern der Daten beim „Home Node" per GET
  - Cache-Line im Zustand *clean* oder *shared*
  - Home-Node
    - schickt Daten übers Netz (PUT) und
    - setzt das Presence-Bit des Anfragenden Knotens
    - neuer Zusand der Cache-line ist shared



Bit-Vector-Protocol



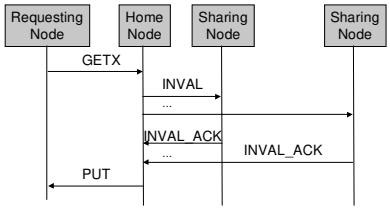
- Read-Miss
  - Anfordern der Daten beim „Home Node" per GET
  - Cache-Line im Zustand *dirty*
  - Home-Node
    - leitet Anfrage an Modifying-Node weiter
  - Modifying-Node
    - schickt die Daten an den Requesting-Node
    - und ein SWB (shared writeback) packet mit der aktuellen Cache-line an den Homenode
  - Home-Node
    - setzt den Zustand auf shared
    - aktualisiert das Presence-Bit
    - schreibt Cache-line in den Speicher

Bit-Vector-Protocol

- Write-Miss
  - GETX-Anfrage an den Home-Node
  - Cache-Line in keinem Cache:
    - Home-Node sendet ein PUTX mit der Cache-line
    - aktualisiert den Presence-Bit-Vector
    - setzt Cache-line in den Zustand *dirty*

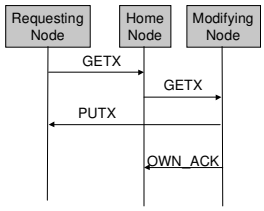
Bit-Vector-Protocol

- Write-Miss
  - Cache-Line in mehreren Caches vorhanden:



Bit-Vector-Protocol

- Write-Miss
  - Cache-Line ist *dirty*:



- Home-Node
  - leitet Anfrage weiter
- Modifying-Node
  - sendet Daten an Requesting-Node
  - sendet *ownership transfer* an den Home-Node
- Home-Node
  - beläßt den Zustand Dirty
  - aktualisiert den Presence-Bit-Vector

Bit-Vector-Protocol

- Pro Cache-Line im Hauptspeicher
  - 2 Bits für Zusand
  - 1 Bit pro Prozessorknoten im System

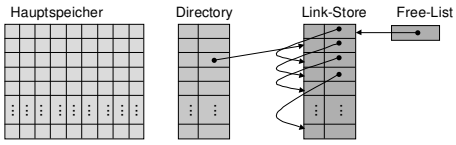
→ großer Speicherbedarf

→ Protokoll nur für kleine und mittelgroße Systeme

- Coarse-Vector
  - jedes Bit repräsentiert ein Cluster von Prozessoren
  - Cluster kann selbst z.B. ein SMP mit snoopy-Protokoll sein

Dynamic-Pointer-Allocation

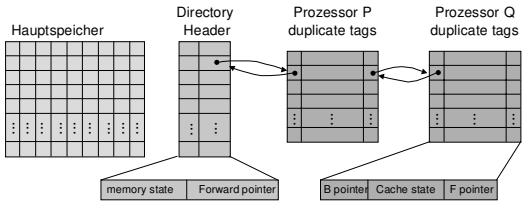
- Prozessoren die eine Cache-Line teilen sind als verkettete Liste abgelegt



- Durch dynamische Allokierung auch für große Systeme mit vielen Prozessoren

Scalable Coherent Interface

- SCI ist standardisiert IEEE 1596-1992
- Die Verkettete Liste wird über die Knoten verteilt gespeichert (doppelverzeigert)



Software-basierte Cache-Coherency Verfahren

- Hardwareunterstützung gerade bei Netzwerk-Architekturen sehr teuer
- Softwarelösung durch Compiler und Betriebssystem interessante Alternative:
  - Cachebility Marking
  - Cache Coherency Enforcement
    - Indiscrimininate Invalidation
    - Fast Selective Invalidation
    - Timestamp

Verschiedene Softwareverfahren

- Cachebility Marking:
  - Einteilung sogenannter "computational units" (z.B. Schleife etc.) in Art der Zugriffe
  - Abhängig von Zugriff kann Variable in Cache oder nicht
  - Beim Übergang zwischen computational units erfolgt Cache-Invalidierung bzw. Speicher-Update
  - Compiler kann/muß:
    - computational unit identifizieren
    - Typ des Zugriffs aufgrund einer Datenflußanalyse ermitteln

Einteilung bzgl. Speicherung in Cache

- Arten von Zugriffen (Ann.: Prozesse auf versch. Prozessoren)
  - Read-Only für beliebig viele Prozesse (cacheable)
  - Read-Only für beliebig viele Prozesse und Read-Write für genau einen Prozeß (höchstens Read-Write Prozeß darf Variable in Cache halten, muß aber dann Konsistenz mit Hauptspeicher sicherstellen z.B. durch write through)
  - Read-Write für genau einen Prozeß (cacheable und copy back)
  - Read-Write für eine beliebige Zahl von Prozessen (non cachable, gilt z.B. typischerweise für Semaphorvariable und andere Synchronisationspunkte)

Cache Coherency Enforcement

- indiscriminate invalidation: zwischen den computation units wird Cache ungültig und es erfolgt Hauptspeicher-Update, alle Variablen eines bestimmten Typs werden hierbei gleich behandelt (leicht zu implementieren aber zu pessimistisch)
- selective invalidation: nur diejenigen Cache-Variablen, die tatsächlich Inkonsistenz erzeugen können werden ungültig
- timestamp: verhindert, daß Cache-Inhalt ungültig wird, wenn gleicher Prozessor wiederholt (aber in verschiedenen computation units) auf dieselbe Variable zugreift