Parallele Rechnerarchitektur I

Stefan Lang

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Universität Heidelberg
INF 368, Raum 532
D-69120 Heidelberg
phone: 06221/54-8264
email: Stefan.Lang@iwr.uni-heidelberg.de

WS 12/13

Parallele Rechnerarchitektur I

- Warum Parallelrechnen?
- Von-Neumann Rechner
- Pipelining
- Cache
- RISC und CISC
- Skalierbare Rechnerarchitekturen
- UMA, NUMA
- Protokolle für Cache Kohärenz
- Beispiele

Definition eines Parallelrechners

Was ist ein Parallelrechner?

A collection of processing elements that communicate and cooperate to solve large problems fast
(Almasi und Gottlieb 1989)

Was ist eine parallele Architektur?

It extends the usual concepts of a computer architecture with a communication architecture

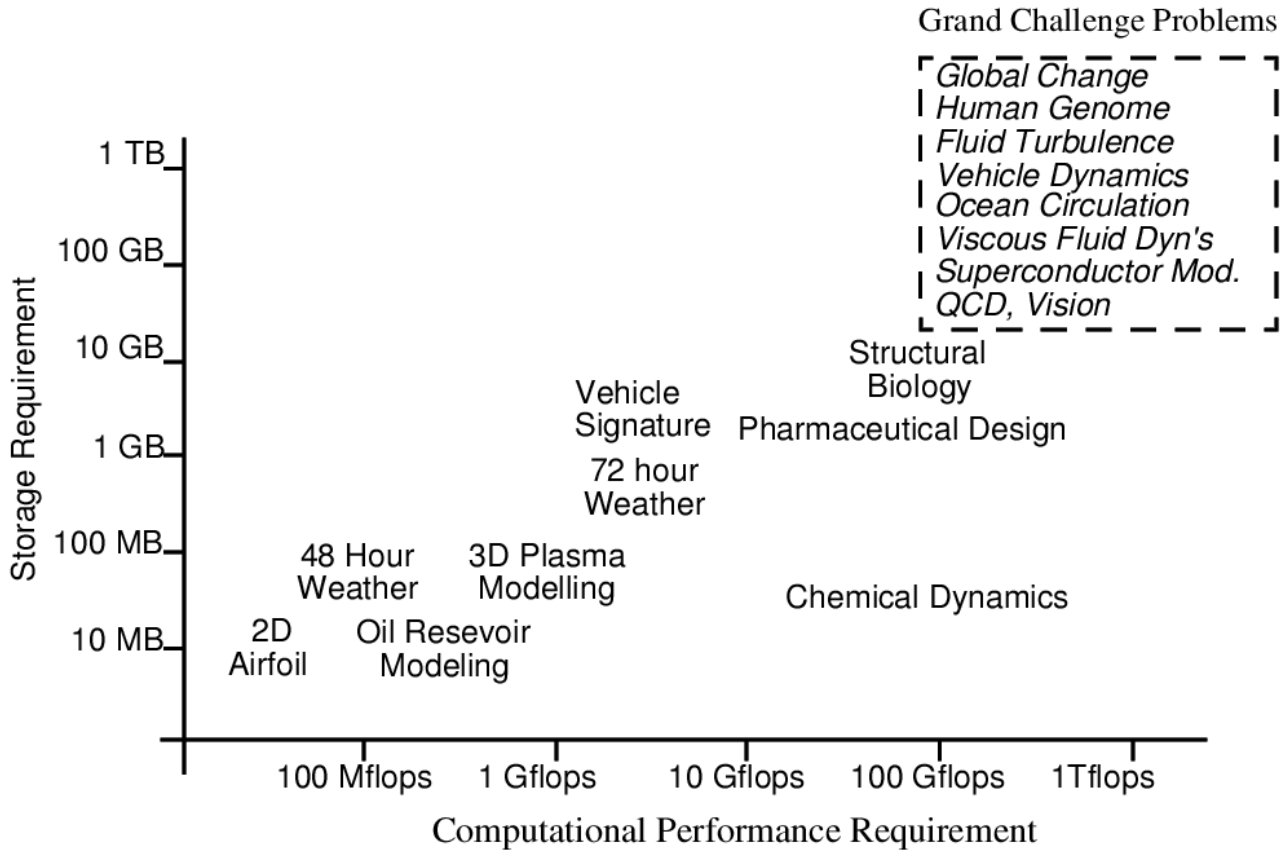
Warum Parallelrechnen?

3 Arten des Parallelrechnens

- Löse ein Problem fester Größe schnell
  Ziel: Minimiere die Lösungszeit (time-to-solution), speedup r&d cycle
- Berechne sehr große Probleme
  Ziel: genaues Ergebnis, komplexe Systeme
- Simuliere sehr große Probleme schnell (bzw. in angemessener Zeit)
  Ziel: Grand Challenges

Einzelprozessorleistung reicht nicht aus
$\rightarrow$ Parallele Architekturen

Was sind Probleme?

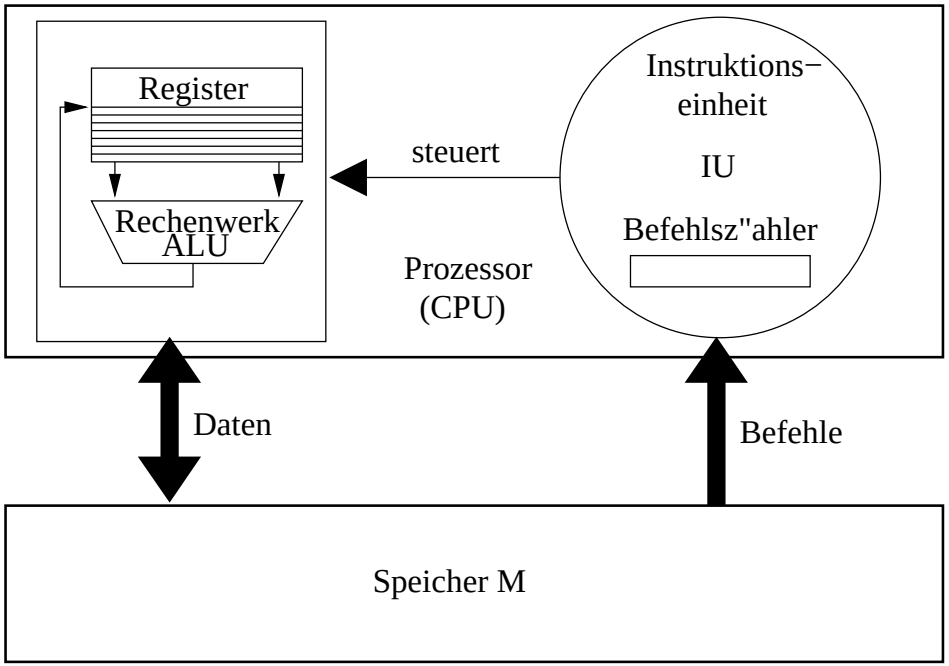


from Culler, Singh, Gupta: Parallel Computer Architecture

- Einordnung der Probleme nach Speicher- und Rechenanforderungen
- Kategorisierung in 3 Typen: speicherlimitierte, rechenzeitlimitierte und ausgewogene Probleme

Von Neumann Rechner

Schematischer Aufbau aus Instruktionseinheit, Rechenwerk und Speicher



Befehlszyklus:

- Instruktion holen
- Instruktion dekodieren
- Instruktion ausführen
- Ergebnisse speichern

- Speicher enthält Programm und Daten
- Datentransfer zwischen Prozessor und Speicher geht über Bus.
- Mehrere Geräte (Prozessoren, E/A-Einheiten, Speicher) am Bus

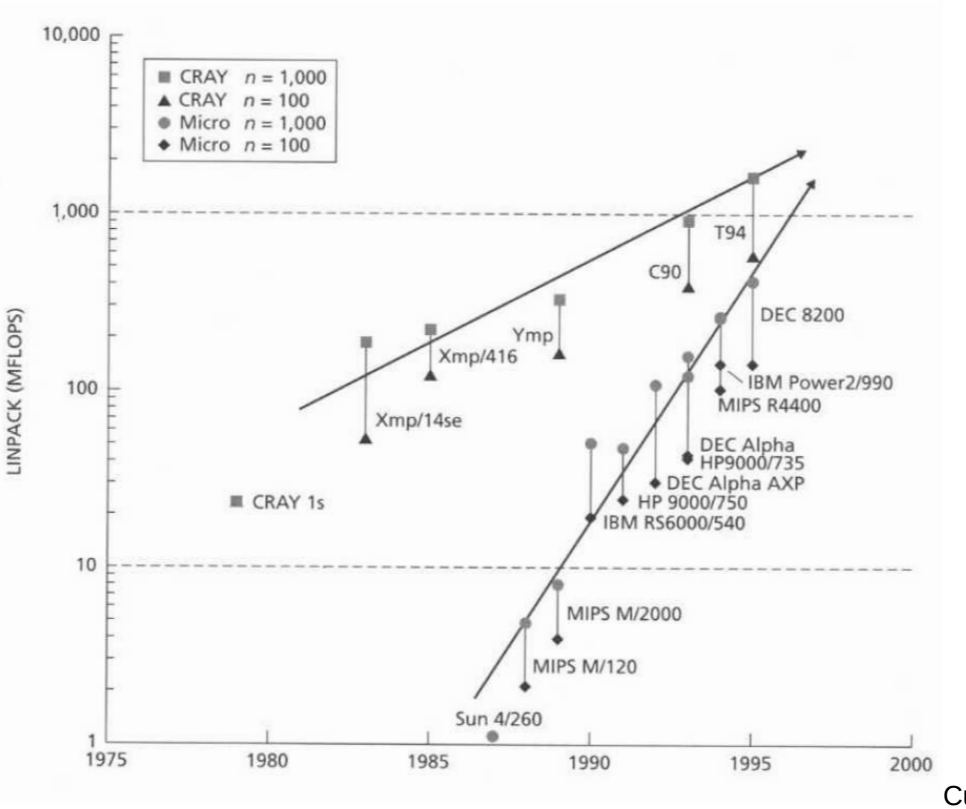
Generationen elektronischer Computer

Einteilung in 5 + 2 Computergenerationen

Generation	Technology and Architecture	Software and Applications	Representative Systems
First (1945-54)	Vacuum tubes and relay memories, CPU driven by PC and accumulator	Machine/assembly languages, single user, no subroutinge programmed I/O using CPU	ENIAC, Princeton IAS, IBM 701
Second (1955-64)	Discrete transistors and core memories, floating-point arithmetic	HLL used with compilers, subroutine libraries, batch processing monitor	IBM 7090, CDC 1604, Univac LARC
Third (1965-74)	Integrated circuits, micro-programming, pipelining cache, lookahead processors	Multiprogramming and time-sharing OS, multiuser applications	IBM 360/370, CDC 6600, TI-ASC, PDP-8
Fourth (1975-90)	LSI/VLSI, semiconductor memory, multiprocessors, vector- and multicomputers	Multiprocessor OS, languages, compilers, enviroments for parallel processing	VAX 9000, Cray X-MP, IBM 3090
Fifth (1991-1997)	ULSI/VHSIC processors, mems and switches, high-density packaging, scalable archs	Massively parallel processing grand challenge applications heterogeneous processing	Fujitsu VPP-500, Cray/MPP, Intel Paragon
Sixth (1997-2003)	commodity-component cluster high speed interconnects	Standardized Parallel Environ-ments and Tools, Metacomputing	Intel ASCI-Red, IBM SP2, SGI Origin
Seventh (2004-present)	Multicore, Powersaving Extending memory hierarchy	Software for Failure Tolerance, Scalable I/O, Grid Computing,	IBM Blue Gene, Cray XT3

nach Hwang (mit Ergänzungen)

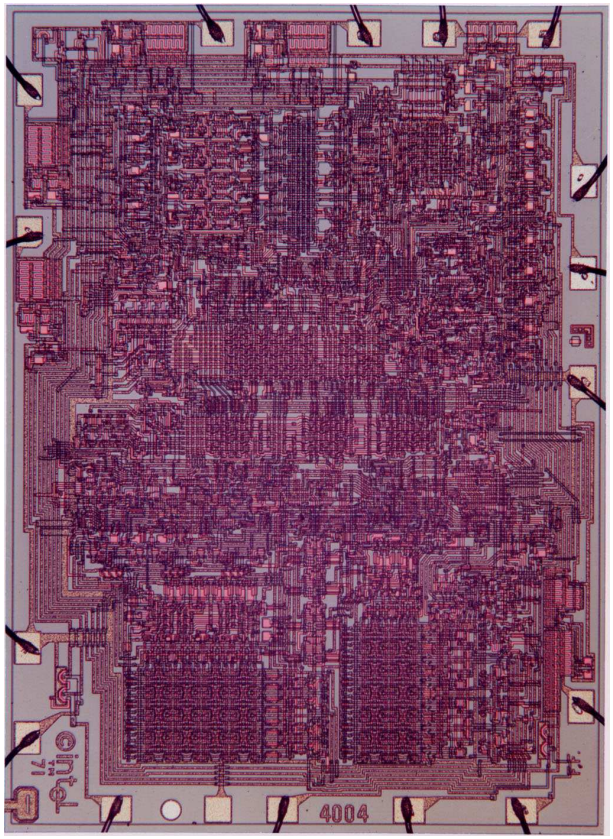
Einzelprozessor Performanz



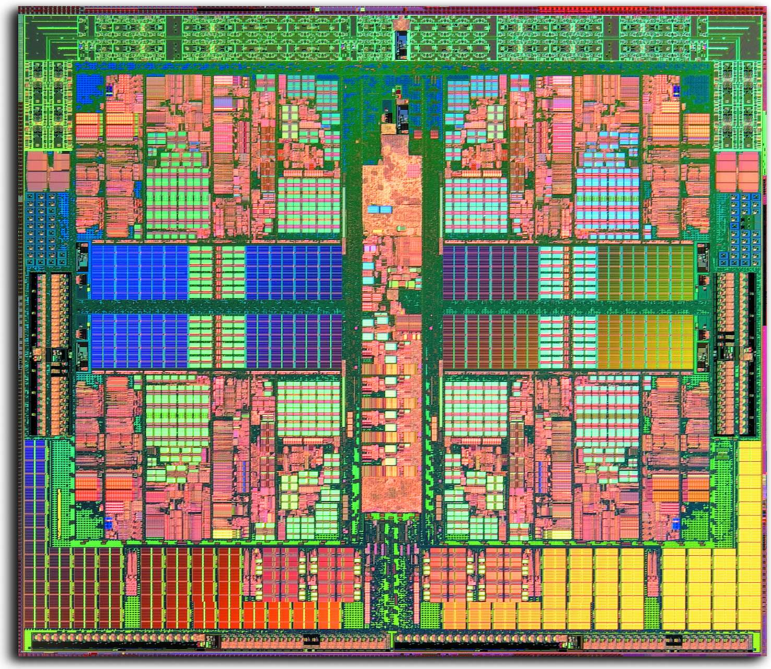
Culler, Singh, Gupta: Parallel Computer Architecture

- Leistungsentwicklung von Vektor- und Superscalar Prozessoren
- früher: viele Hersteller, jetzt: einige Marktführer
- Geschwindigkeitsvorteil des Vektorprozessors schrumpft

Einzelprozessor: zwei Beispiele



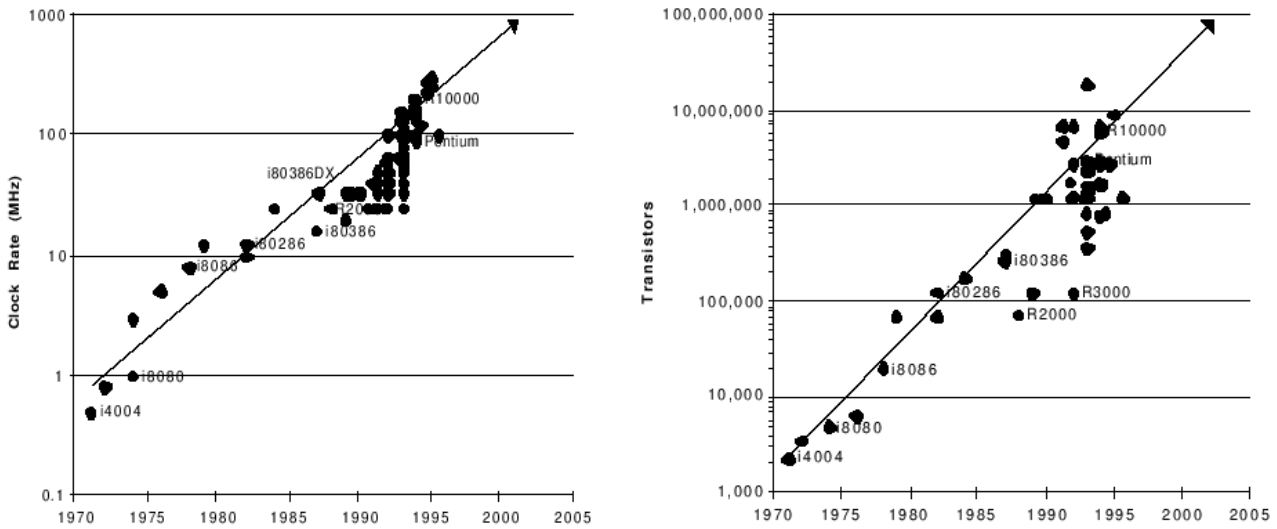
1971: Intel 4004, 2700 Trans.,
4 bit, 100 KHz



2007: AMD Quadcore, 465 Mill. Trans.,
64 bit, 2 GHz

Intel Gründer Andy Grove, Robert Noyce, Gordon Moore in 1978

Integrationsdichte und Taktfrequenz



Culler, Singh, Gupta: Parallel Computer Architecture

- Anstieg gemäß Moore's Gesetz: Verdopplung innerhalb von 18 Monaten
- Moore's Gesetz bezieht sich NICHT auf Leistung sondern Integrationsdichte
- Divergenz von Geschwindigkeit und Kapazität in Speichertechnologien

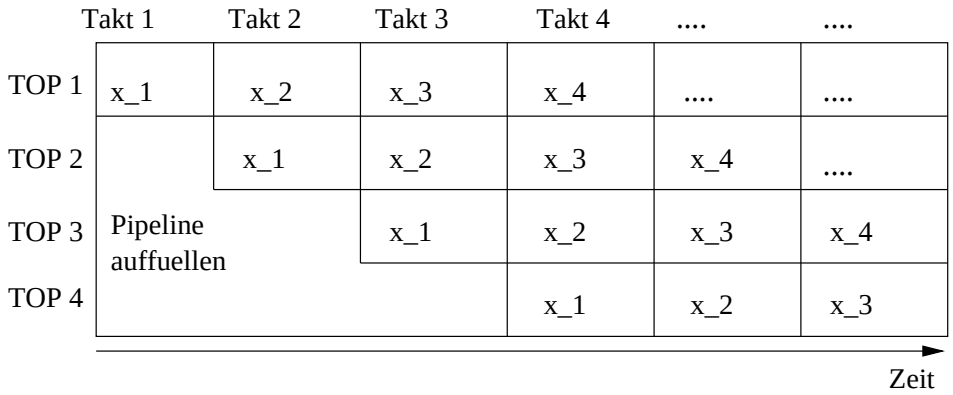
Architektur von Einzelprozessoren

Techniken zur Steigerung der Einzelprozessorleistung

- deep pipelining
- speculative branch prediction
- out-of-order execution
- clock frequency scaling
- superscalar design (instruction level parallelism ILP)
- speculative execution
- thread-level parallelism
- multi-core design

Pipelining I: Prinzip

Gleichzeitige, überlappende Abarbeitung von Operationen
Pipeline mit 4 Stufen:



Voraussetzungen:

- Eine Operation $OP(x)$ muss auf viele Operanden $x_1, x_2, \ldots$ in Folge angewandt werden.
- Die Operation kann in $m > 1$ Teiloperationen (oder auch Stufen) zerlegt werden, die in (möglichst) gleicher Zeit bearbeitet werden können.
- Ein Operand x_i darf nur mit Einschränkungen das Ergebnis einer früheren Operation sein.

Gewinn durch Pipelining: Der Zeitbedarf für die Verarbeitung von N Operanden beträgt

$$T_P(N) = (m + N - 1)\frac{T_{OP}}{m}$$

Pipelining II: Beschleunigung

Die Beschleunigung beträgt somit

$$S(N) = \frac{T_S(N)}{T_P(N)} = \frac{N * T_{OP}}{(m + N - 1)\frac{T_{OP}}{m}} = m\frac{N}{m + N - 1}$$

Für $N \to \infty$ geht die Beschleunigung gegen m.

Anwendungen im Prozessor:

- Instruktionspipelining: fetch, decode, exectue, write back
- Arithmetisches Pipelining: Exponenten angleichen, Mantisse addieren, Mantisse normieren

Weitere Anwendungen:

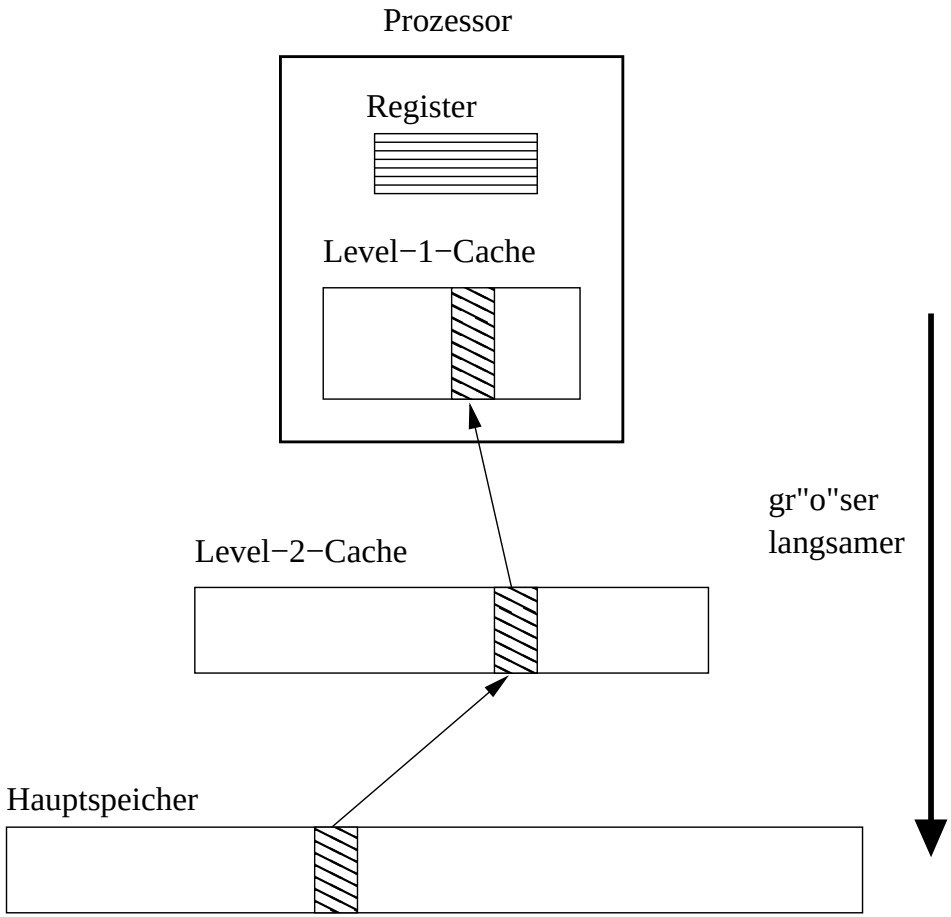
- Verschränkter Speicher (memory interleaving)
- Cut-through Routing
- Wavefront Algorithmen: LU-Zerlegung, Gauß–Seidel
- ...

Cache I: Speicherhierarchie

Speed gap:

- Prozessoren sind schnell: 2-3 GHz Takt, $\geq$ 1 Befehl/Takt durch Pipelining
- Speicher sind langsam: MHz Takt, 7 Takte um 4 Worte zu lesen

Ausweg: Hierarchie von immer langsameren aber größeren Speicher



Cache II: Cache Organisation

Speicher enthält jeweils zuletzt benutzte Daten der nächsten Hierarchieebene

Transfer erfolgt in Blöcken (Cache Lines), typische Größe: 16...128 Bytes

Cache Organisation:

- direkte Zuordnung: Hauptspeicherblock i kann nur in Platz $j = i \mod M$ im Cache geladen werden (M: Größe des Cache).
  Vorteil: einfache Identifikation, Nachteil: Aliasing.
- Assoziativ Cache: Hauptspeicherblock i kann an jede Stelle im Cache geladen werden.
  Vorteil: kein aliasing, Nachteil: aufwendige Identifizierung (M Vergleiche).
- Kombination: k-Wege Assoziativcache.

Ersetzen: LRU (least recently used), random

Zurückschreiben: write through, write back

Cache III: Lokalitätsprinzip

Bisher gingen wir davon aus, dass auf alle Speicherworte gleichschnell zugegriffen werden kann.

Mit Cache kann aber auf die zuletzt geholten Daten schneller zugegriffen werden. Dies hat Auswirkungen auf die Implementierung von Algorithmen.

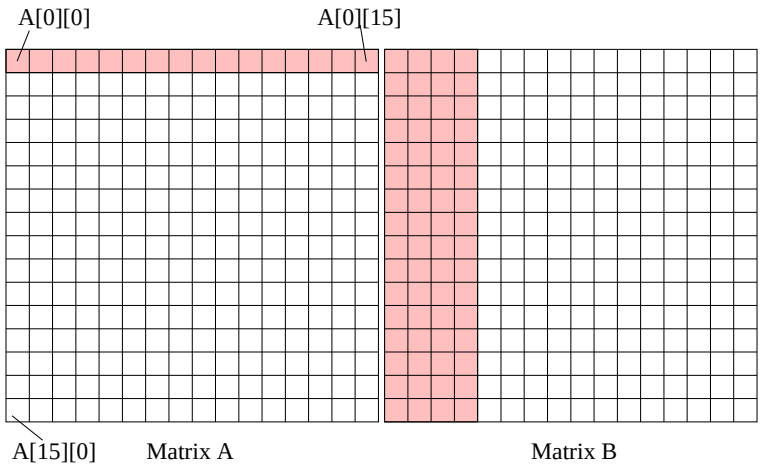
Beispiel: Multiplikation zweier $n \times n$-Matrizen $C = AB$

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Annahme: Cache-Line ist 32 Bytes = 4 Floating Point Zahlen.

Cache III: Lokalitätsprinzip

Nach Berechnung von $C[0][0]$ befinden sich im Cache:



A, B, C voll im Cache: $2n^3$ Rechenoperationen aber nur $3n^2$ Speicherzugriffe

Falls weniger als $5n$ Zahlen in den Cache passen: langsam

Tiling: Bearbeite Matrix in $m \times m$ Blöcken mit $3m^2 \leq M$

```
for (i = 0; i < n; i+=m)
    for (j = 0; j < n; j+=m)
        for (k = 0; k < n; k+=m)
            for (s = 0; s < m; s++)
                for (t = 0; t < m; t++)
                    for (u = 0; u < m; u++)
                        C[i + s][j + t] += A[i + s][k + u] * B[k + u][j + t];
```

RISC und CISC

RISC =„reduced instruction set computer“
CISC=„complex instruction set computer“

Entwicklung von Prozessoren mit immer komplizierteren Befehlssätzen
(z. B. Adressierungsarten): Aufwändige Dekodierung, unterschiedlich lange Befehle

Beginn der 80er Jahre: „Back to the roots“. Einfache Befehle, agressives Nutzen von Pipelining.

Die Idee war nicht neu: Seymor Cray hat immer RISC gebaut (CDC 6600, Cray 1).

Designprinzip von RISC Rechnern:

- Alle Befehle werden in Hardware dekodiert, keine Mikroprogrammierung.
- Agressive Ausnutzung von Instruktionspipelining (Parallelismus auf Instruktionsebene ILP).
- Möglichst ein Befehl/Takt ausführen (oder mehr bei superskalaren Maschinen). Dies erfordert einen möglichst einfachen, homogenen Befehlssatz.
- Speicherzugriffe nur mit speziellen Load/Store–Befehlen, keine komplizierten Adressierungsarten.
- Viele Allzweckregister bereitstellen um Speicherzugriffe zu minimieren. Die gesparte Chipfläche im Instruktionswerk wird für Register oder Caches verwendet.
- Folge dem Designprinzip „Mache den oft auftretenden Fall schnell“.

Heute vorwiegend RISC-Prozessoren. Intel Pentium ist CISC mit RISC-Kern.

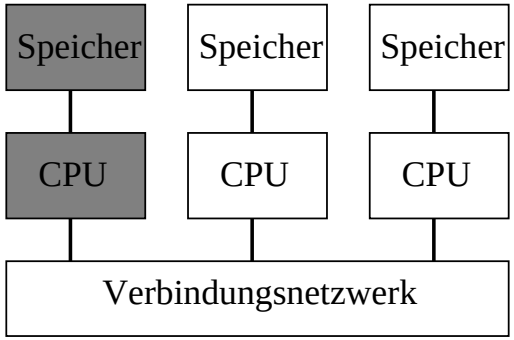
Skalierbare Rechnerarchitekturen I

Klassifikation von Parallelrechnern nach FLYNN (1972)

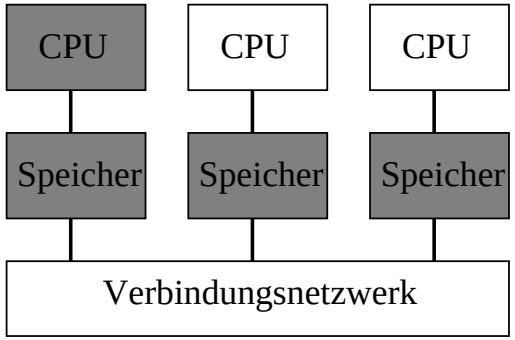
Unterscheidung nach Datenströmen und Kontrollpfaden

- *SISD – single instruction single data*: Der Von Neumann Rechner
- *SIMD – single instruction multiple data*: Diese Rechner, auch Feldrechner genannt, verfügen über ein Instruktionswerk und mehrere unabhängige Rechenwerke von denen jedes mit einem eigenen Speicher verbunden ist. Die Rechenwerke werden taktsynchron vom Instruktionswerk angesteuert und führen die selbe Operation auf unterschiedlichen Daten aus.
- *MISD – multiple instruction single data*: Diese Klasse ist leer.
- *MIMD – multiple instruction multiple data*: Dies entspricht einer Kollektion eigenständiger Rechner, jeder mit einem Instruktions– und einem Rechenwerk ausgestattet.

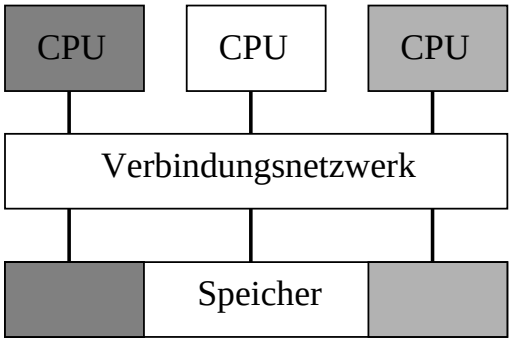
Skalierbare Rechnerarchitekturen II



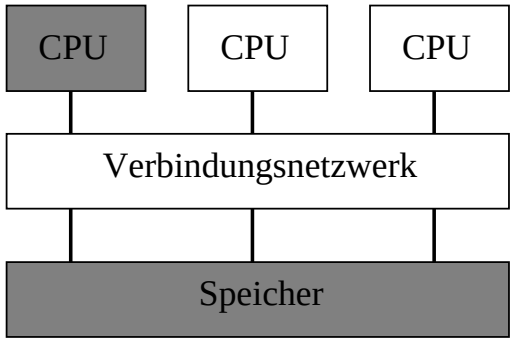
(a) verteilte Speicherorganisation, lokaler Adressraum



(b) verteilte Speicherorganisation, globalerAdressraum



(c) zentrale Speicherorganisation, lokaler Adressraum



(d) zentrale Speicherorganisation, globalerAdressraum

Unterscheidung nach Anordnung von Prozessor, Speicher und Kommunikationsnetzwerk

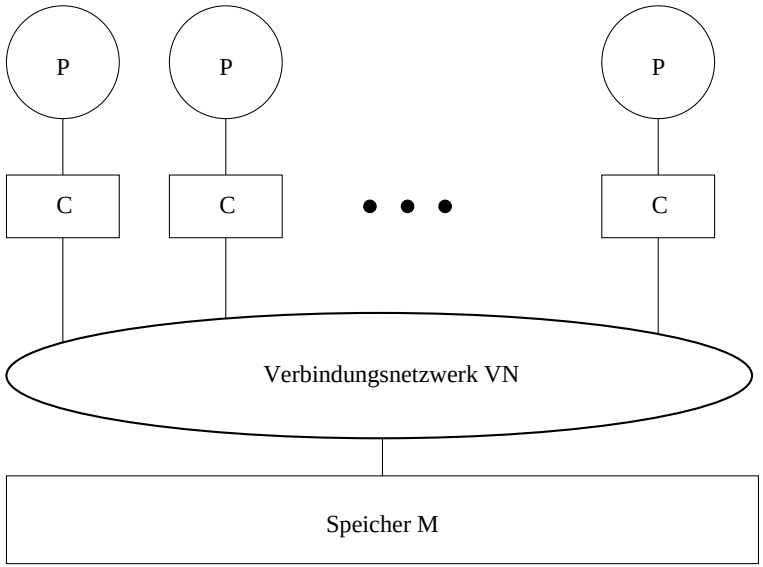
Skalierbare Rechnerarchitekturen III

Klassifikation nach der Art des Datenaustausches:

- Gemeinsamer Speicher (Shared Memory)
  - *UMA – uniform memory access*. Gemeinsamer Speicher mit uniformer Zugriffszeit.
  - *NUMA – nonuniform memory access*. Gemeinsamer Speicher mit *nicht*-uniformer Zugriffszeit, mit Cache-Kohärenz spricht man auch von *ccNUMA*.
- Verteilter Speicher (Distributed Memory)
  - *MP – multiprozessor*. Privater Speicher mit Nachrichtenaustausch.

Wir werden vorwiegend MIMD–Rechner betrachten. Der SIMD-Ansatz existiert noch im datenparallelen Programmiermodell (OpenMP, CUDA/OpenCL).

Shared Memory: UMA



- *Globaler Adressraum*: Jedes Speicherwort hat global eindeutige Nummer und kann von allen Prozessoren gelesen und geschrieben werden.
- Zugriff auf Speicher erfolg über *dynamisches* Verbindungsnetzwerk welches Prozessor und Speicher verbindet (davon später mehr).
- Speicherorganisation: *Low-order interleaving* – aufeinanderfolgende Adressen sind in aufeinanderfolgenden Modulen. *High-order interleaving* – aufeinanderfolgende Adressen sind im selben Modul.

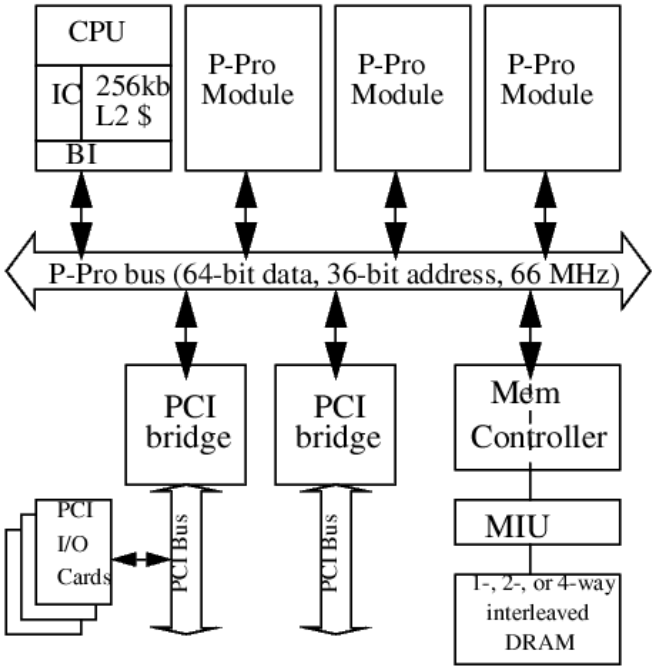
Shared Memory: UMA

Cache ist notwendig um

- Prozessor nicht zu bremsen, und
- Last vom Verbindungsnetzwerk zu nehmen.

- Cache-Kohärenzproblem: Ein Speicherblock kann in mehreren Caches sein. Was passiert, wenn ein Prozessor schreibt?
- Schreibzugriffe auf den selben Block in verschiedenen Caches müssen sequentialisiert werden. Lesezugriffe müssen stets aktuelle Daten liefern.
- UMA erlaubt den Einsatz von bis zu wenigen 10 Prozessoren.

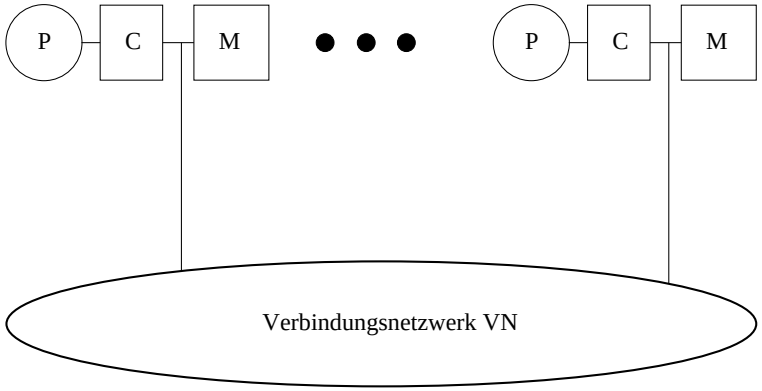
Shared Memory Board: UMA



Quad-prozessor Pentium Pro Motherboard

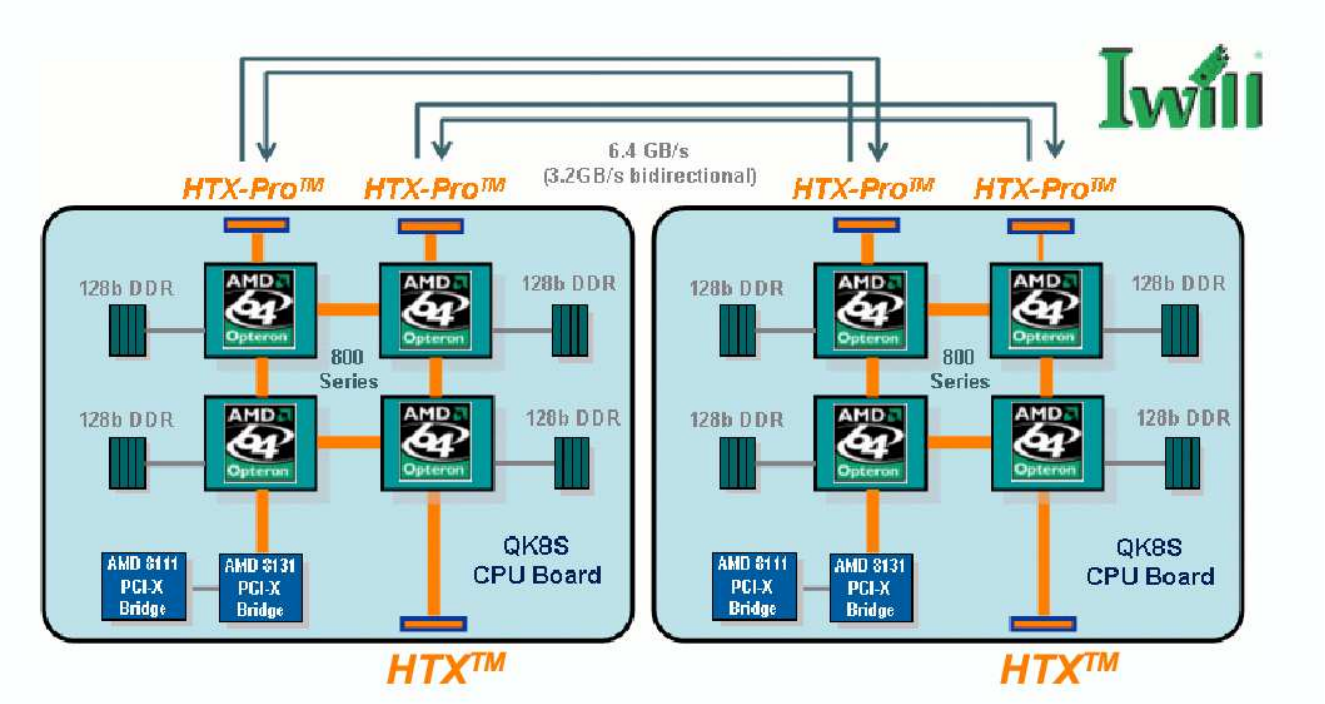
- Symmetric multi processing (SMP)
- Zugriff zu jedem Speicherwort in gleicher Zeit
- Unterstützung von Cache Kohärenz Protokollen (MESI)

Shared Memory: NUMA



- Jeder Baustein besteht aus Prozessor, Speicher und Cache.
- *Globaler Adressraum*: Jedes Speicherwort hat global eindeutige Nummer und kann von allen Prozessoren gelesen und geschrieben werden.
- Zugriff auf lokalen Speicher ist schnell, Zugriff auf andere Speicher ist (wesentlich) langsamer, aber transparent möglich.
- Cache-Kohärenzproblem wie bei UMA
- Extreme Speicherhierarchie: level-1-cache, level-2-cache, local memory, remote memory
- Skaliert bis etwa 1000 Prozessoren (SGI Origin)

Shared Memory Board: NUMA

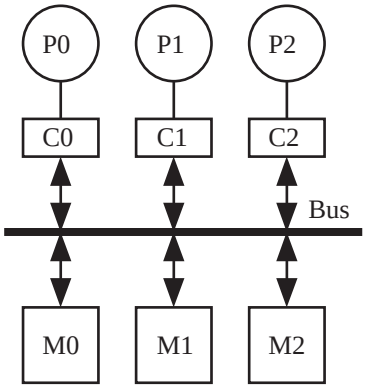


Quad-prozessor Opteron Motherboard

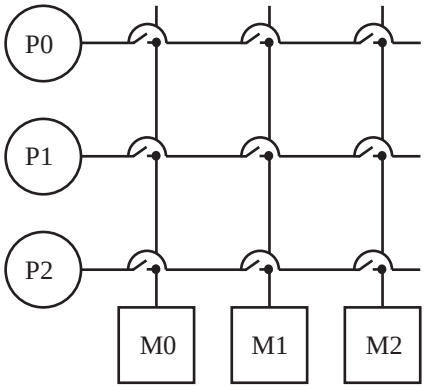
- Verbindung mit Hypertransport HTX-Technologie
- Nicht-uniformer Speicherzugriff (NUMA)

Dynamische Verbindungsnetzwerke

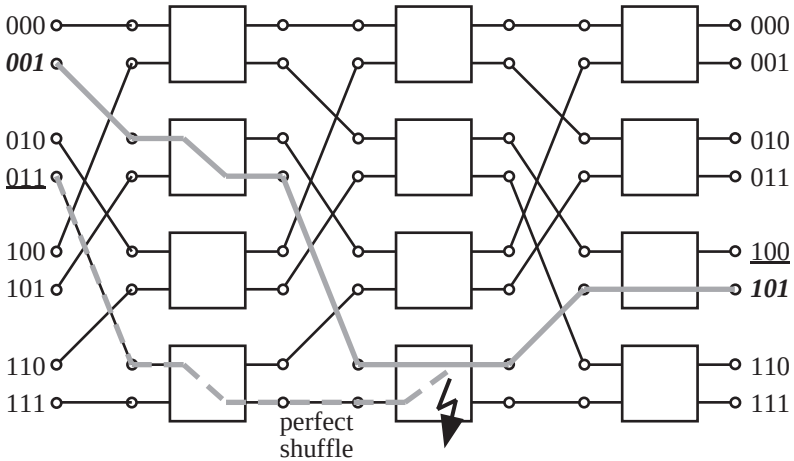
Leitungsvermittelnd: Elektrische Verbindung von Quelle zum Ziel.



(a)



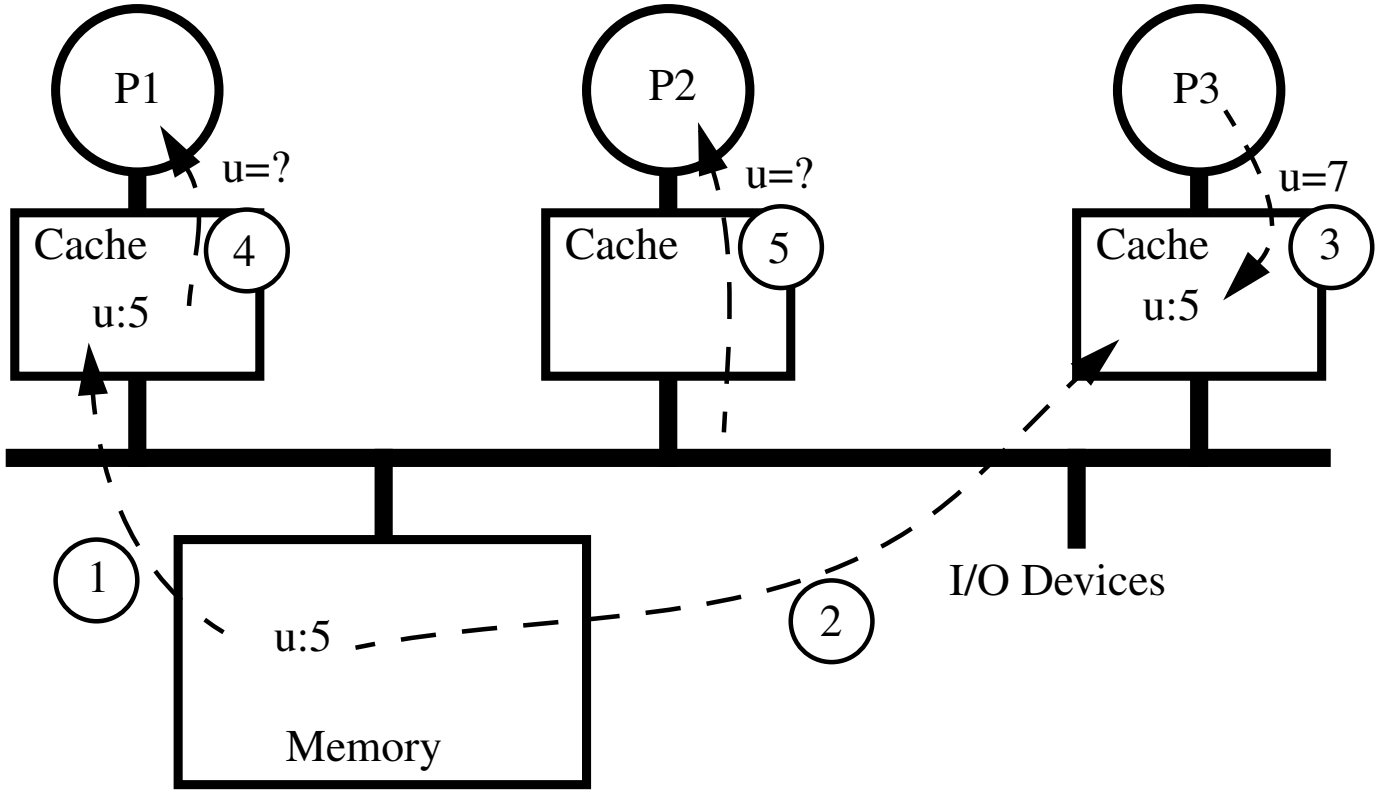
(b)



(c)

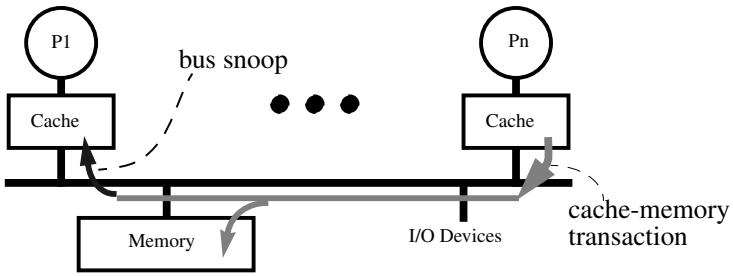
(a) Bus: verbindet nur jeweils zwei Einheiten zu einer Zeit, also nicht skalierbar. Vorteile: billig, Cache-Kohärenz durch Schnüffeln.
(b) Crossbar: Volle Permutation realisierbar, aber: P^2 Schaltelemente.
(c) Ω-Netzwerk: $(P/2)\,\mathrm{ld}\,P$ Schaltelemente, keine volle Permutation möglich, jede Stufe ist *perfect shuffle*, einfaches Routing.

Cache Kohärenz: Ein Beispiel

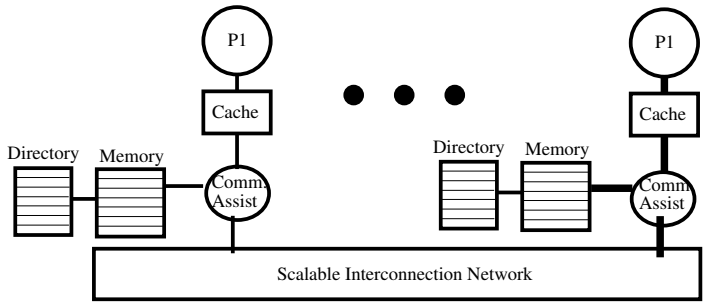


Cache Kohärenz: Protokolltypen

Snooping basierte Protokolle



Verzeichnis basierte Protokolle



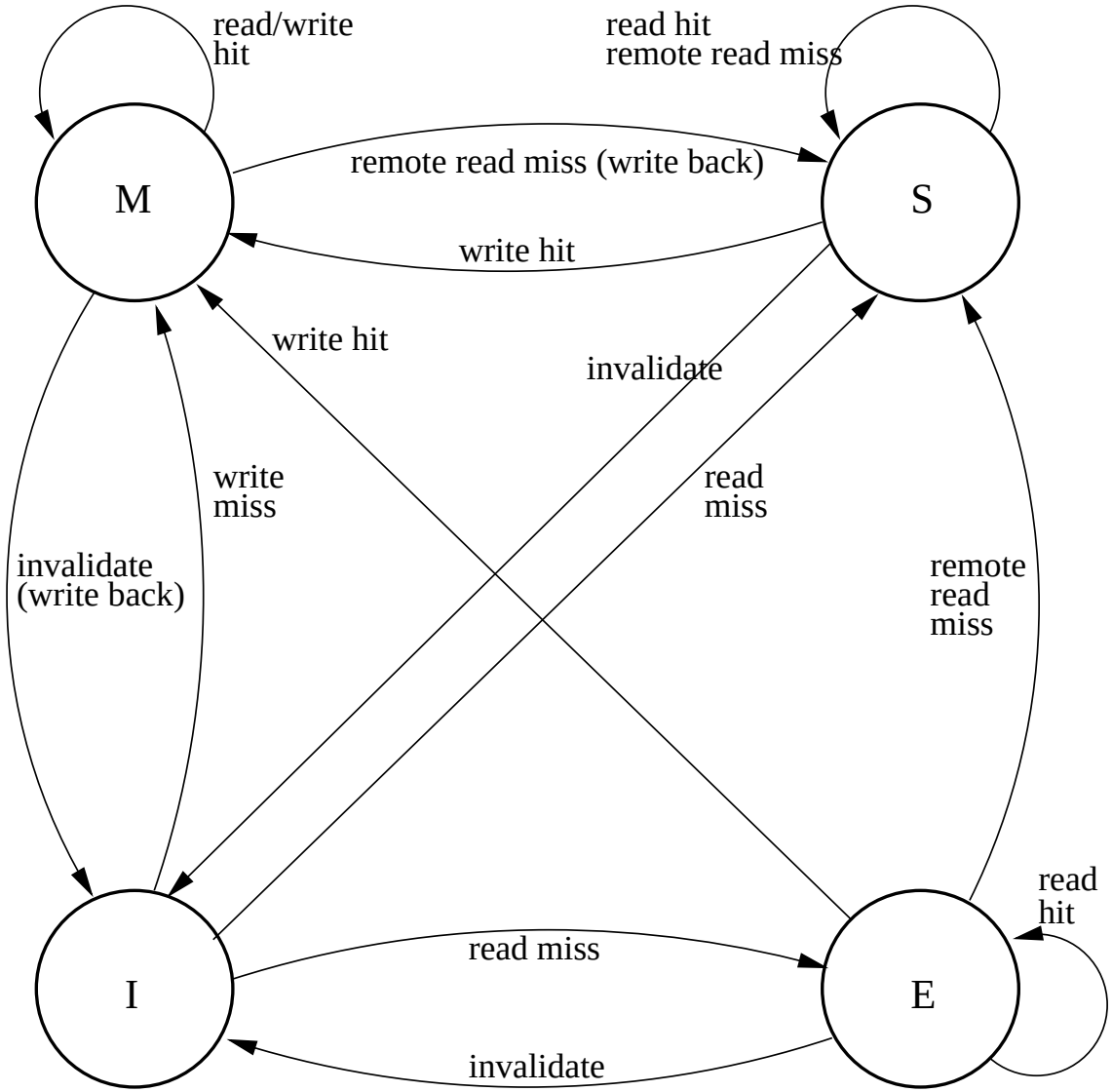
Cache Cohärenz: Bus Snooping, MESI

- Bus erlaubt einfaches, effizientes Protokoll zur Cache-Kohärenz.
- Beispiel MESI: Jeder Cache-Block hat einen der folgenden Zustände:

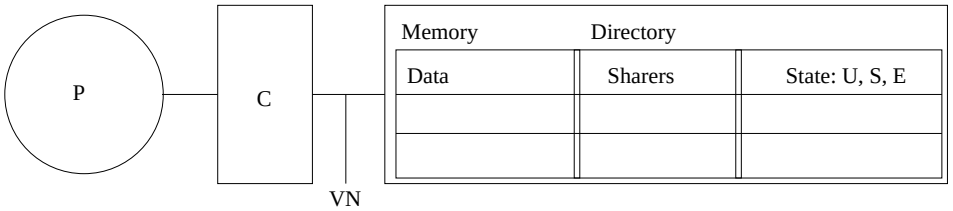
Zustand	Bedeutung
I	Eintrag ist nicht gültig
E	Eintrag gültig, Speicher aktuell, keine Kopien vorhanden
S	Eintrag gültig, Speicher aktuell, weitere Kopien vorhanden
M	Eintrag gültig, Speicher ungültig, keine Kopien vorhanden

- Erweitert write-back Protokoll um Cache-Kohärenz.
- Cache-Controller hört Verkehr auf Bus ab (schnüffelt) und kann folgende Zustandübergänge vornehmen (aus Sicht *eines* controllers):

Cache Cohärenz: Bus Snooping, MESI



Verzeichnisbasierte Cache-Kohärenz I

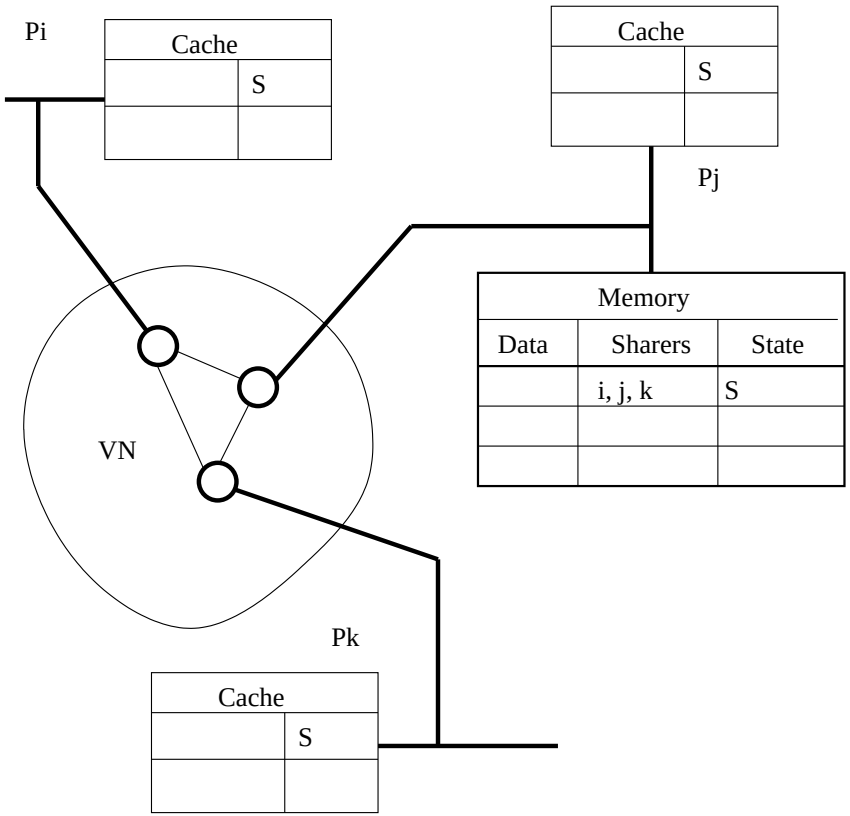


Zustände:

Cache–Block		Hauptspeicherblock	
Zust.	Beschreibung	Zust.	Beschreibung
I	Block ungültig	U	niemand hat den Block
S	$\geq$ 1 Kopien vorhanden, Caches und Speicher sind aktuell	S	wie links
E	genau einer hat den Block geschrieben (entspricht M in MESI)	E	wie links

Verzeichnisbasierte Cache-Kohärenz III: Beispiel

Situation: Drei Prozessoren P_i, P_j, und P_k haben eine Cache Line im Zustand shared. Home Node dieses Speicherblockes ist P_j



Aktionen:

1. Prozessor P_i schreibt in die Cache-Line (write hit): Nachricht an Verzeichnis, dieses benachrichtet Caches von P_j und P_k, Nachfolgezustand ist E in P_i
2. Prozessor P_k liest von diesem Block (read miss): Verzeichnis holt Block von P_i, Verzeichnis schickt Block an P_k

Verzeichnisbasierte Cache-Kohärenz III: Problemfälle

Probleme der ccNUMA-Architekturen:

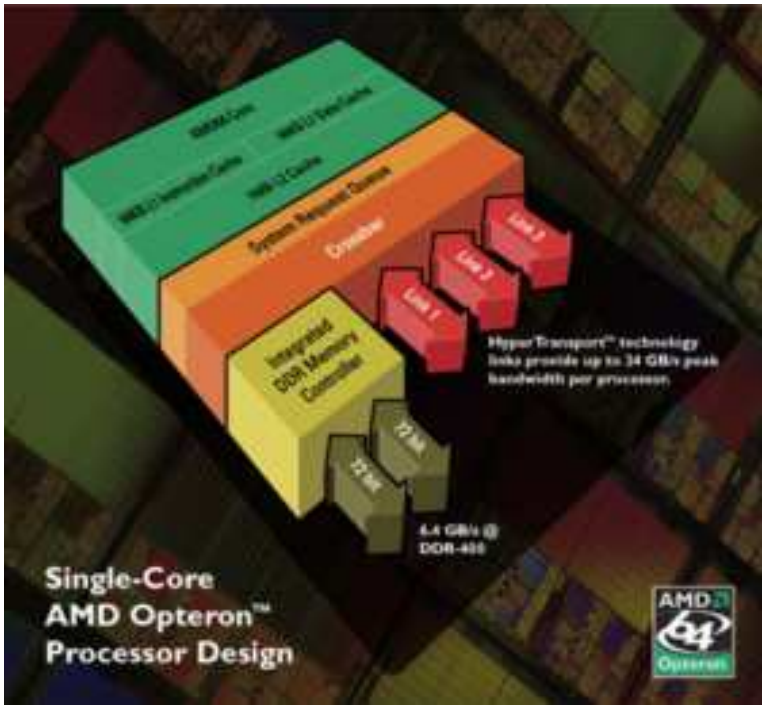
- false sharing: Zwei Prozessoren lesen und schreiben verschiedene Speicherstellen, die sich zufällig im selben Block befinden (Wahrscheinlichkeit steigt mit Blockgröße, Origin: 128 Bytes)
- capacity miss: Datenmenge die ein Prozessor bearbeitet (working set) passt nicht in den Cache und diese Daten sind im Hauptspeicher eines anderen Prozessors

Lösung für das Kapazitätsproblem: Cache Only Memory Architecture (COMA), Software Distributed Shared Memory. Seiten des Hauptspeichers (z. B. 4-16 KB) können automatisch migriert werden, Kombination mit virtuellem Speicher Mechanismus.

Examples I: Intel Xeon MP

- IA32 Architektur (as P4)
- Cache Kohärenz Protockoll MESI
- Hyper-Threading Technik (2 logische CPUs)
- Integrierte 3-schichtige Cache Architektur (-1 MB L2, -8 MB L3)
- Machine Check Architektur (MCA) für externe und interne Busse, Cache, Translation Look-aside Buffer und Instruction Fetch Unit
- Intel NetBurst Mikroarchitektur

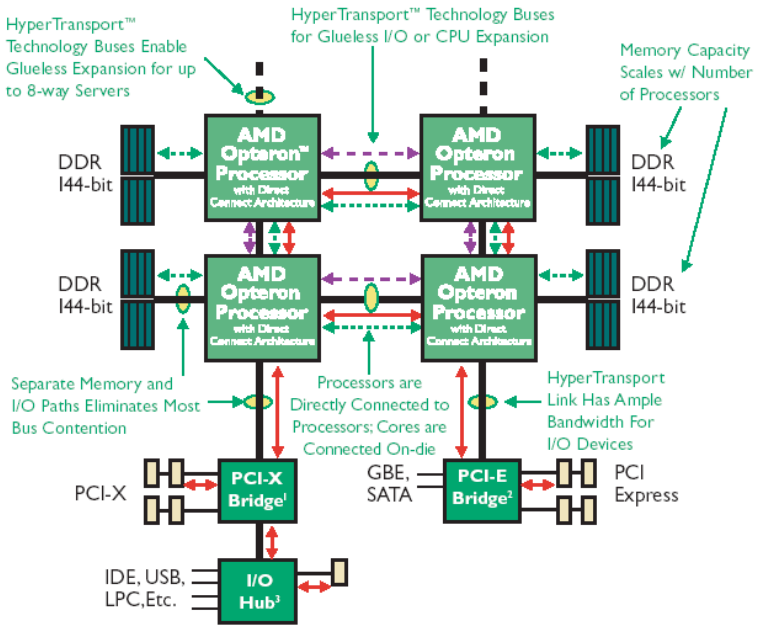
Examples II: AMD Opteron



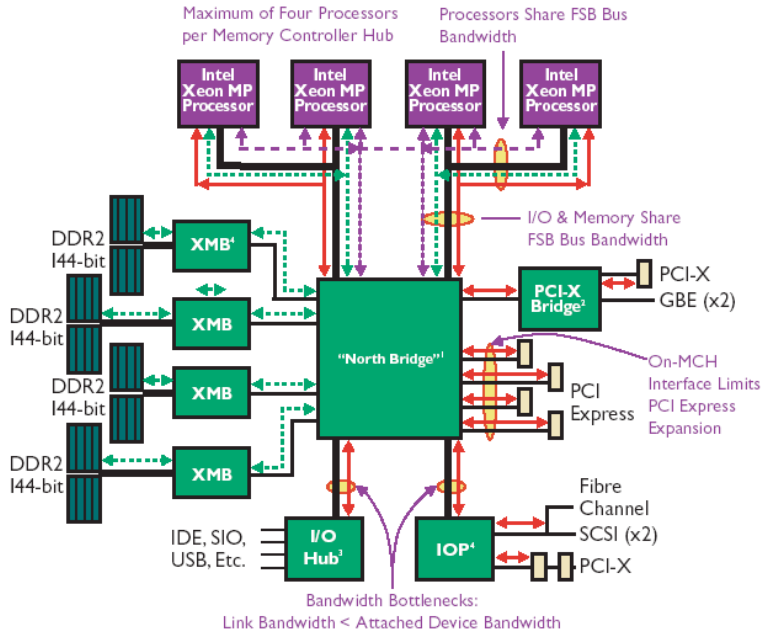
- Direct Connect Architektur
- On-Chip DDR Speicher Kontroller
- HyperTransport Technologie
- Cache Kohärenz Protokoll MOESI
  MESI Zustände + 5-ter Zustand
  direkter Datentransfer zwischen den CPU Caches via Hypertransport
- 64-bit Daten/Adreß Pfad, 48-bit virtueller Adreßraum
- ECC für L1/L2 und DRAM mit hardware scrubbing
- 2 zusätzliche Pipelinestufen
- viele IPCs (instructions per cycle) durch verbesserte Verzweigungsvorhersage (branch prediction)

Examples III: Server Architektur AMD vs INTEL

AMD Opteron™ Processor-based 4P Server



Intel Xeon MP Processor-based 4P Server



www.amd.com/us-en/assets/content_type/DownloadableAssets/AMD_Opteron_Streams_041405_LA.pdf

Examples III: Server Architektur AMD vs INTEL

Server System Comparison	AMD Opteron™	Intel Xeon[1]	Intel Xeon[2]	Intel Xeon MP[3]	Intel Itanium 2[4]
Modular, glueless scalability	Yes	Requires Northbridge	Requires Northbridge	Requires Northbridge	Requires Northbridge
SMP Capabilities	Up to 8-way	Up to 2-way	Up to 2-way	Up to 4-way	Up to 4-way
Direct Connect Architecture	Yes	No	No	No	No
Dual-Core technology	Yes	No	No	No	No
High Performance 32-bit and 64-bit computing	AMD64	No	EM64T	EM64T	No
HyperTransport™ technology	Yes	No	No	No	No
Integrated DDR memory controller	Yes	No	No	No	No
Front Side Bus frequency	1.4 – 2.6GHz†	533MHz	800MHz	667MHz	400MHz
Front Side Bus bandwidth	11.2 – 20.8GB/s†	4.3GB/s	6.4GB/s	10.6GB/s	6.4GB/s
Maximum Inter-processor bandwidth	8.0GB/s	4.3GB/s	6.4GB/s	10.6GB/s	6.4GB/s
Memory support	DDR200/266/333/400	DDR266	DDR333/DDR2-400	DDR266/333/DDR2-400	DDR200
Memory Bandwidth 2P System	12.8GB/s††	4.3GB/s	6.4GB/s	12.8GB/s	6.4GB/s
Memory Bandwidth 4P System	25.6GB/s†††	N/A	N/A	12.8GB/s	6.4GB/s
L1 cache size (max.)	64KB (Data) + 64KB (Instruction) per core	8KB + 12k mop	16KB + 12k mop	16KB + 12k mop	32KB
L2 cache size (max.)	1MB per core	512KB	2MB	1MB	256KB
L3 cache size (max.)	N/A	2MB	N/A	8MB	9MB
Maximum I/O bandwidth 2P System	24.0GB/s††	3.2GB/s	12.3GB/s	14.0GB/s	6.4GB/s
Maximum I/O bandwidth 4P System	32.0GB/s†††	N/A	N/A	14.0GB/s	6.4GB/s
SIMD Instruction Set Support	SSE, SSE2, SSE3	SSE, SSE2	SSE, SSE2, SSE3	SSE, SSE2, SSE3	N/A

Examples IV: Board Level Protokoll

Hypertransport: Low latency chip-to-chip interconnect up to 8 CPUs with I/O aggregate bandwidth 8 GB/s (22.4), link width 16 bit (32), clock 1 GHz (1.4)
Priority request interleaving