Rechnerarchitektur SS 2014

Parallele Rechnersysteme

Michael Engel

TU Dortmund, Fakultät für Informatik

Teilweise basierend auf Material von Gernot A. Fink und R. Yahyapour

3. Juni 2014

Mehrprozessorsysteme

"The turning away from the conventional organization came in the middle 1960s, when the law of diminishing returns began to take effect in the effort to increase the operational speed of a computer ... Electronic circuits are ultimately limited in their speed of operation by the speed of light ... and many ... were already operating in the nanosecond range."

W. Jack Bouknight et al.: The Illiac IV System (1972)

"... sequential computers are approaching a fundamental physical limit on their potential power. Such a limit is the speed of light ..."

Angel L. DeCegama: The Technology of Parallel Processing (1989)

Mehrprozessorsysteme: Einführung

Beobachtung: Einschätzung, dass Fortschritte bei Uni-Prozessor-Architekturen sich ihrem Ende nähern, wurde verschiedentlich und zu verschiedenen Zeiten vertreten.

Allerdings: Allerdings: 1985-2000 höchste Leistungssteigerung bei Uni-Prozessoren zu beobachten (seit d. Einführung transistorbasierter Rechner Ende der 1950er)

$\Rightarrow$ Triebkraft: Verwendung/Technologie von Mikroprozessoren

Trotzdem: Bedeutung paralleler Prozessoren wächst und wird *mit Sicherheit* in der Zukunft weiter steigen

Mehrprozessorsysteme: Einführung II

Gründe für wachsende Bedeutung von MPs

- Mikroprozessoren werden vorherrschende Technologie für Uni-Prozessoren bleiben
  $\Rightarrow$ Weg zur Leistungssteigerung: Kombination mehrerer Prozessoren (seit einiger Zeit auch on-chip, z.B. Dual/Quad-Core-Prozessoren, MPSoCs)
  Annahme: Kosteneffektiver als Spezialprozessoren
- Fraglich, ob Fortschritt bei Ausnützung von ILP (seit ca. 20 Jahren Basis für Innovationen und Leistungssteigerungen bei Uni-Prozessoren) gehalten werden kann
  Beachte: Komplexität von z.B. multiple-issue Prozessoren
- Stetiger Fortschritt bei größter Hürde für MP-Einsatz: *Software*
  - Groß bei Server-Anwendungen (haben häufig "natürlichen" Parallelismus)
  - Problematischer bei Desktop-Systemen/Anwendungen
  - Ex. erste weiter verbreitete Programmiersysteme (z.B. OpenMP)

Mehrprozessorsysteme: Einführung III

Motivation der Themenbehandlung

- Bereits jetzt Abschwächung der Leitstungssteigerung bei Uni-Prozessoren bemerkbar!

  Trotzdem: Steigerung der Integrationsdichte folgt (vorerst?) ungebrochen dem Moore'schen Gesetz ... betrifft aber mehr und mehr Intergration *weiterer Komponenten* on-chip!

- Vorstoss von MPs in breitere Märkte vollzogen (im Desktop/Server-Bereich: Multi-Core-Prozessoren)
- Parallelisierung *immer schon* wichtigste algorithmische Triebkraft bei der Leistungssteigerung:
  MPs wenden Prinzip nur auf anderer Granularitätsstufe an (i. Vgl. zu Pipelining)

$\Rightarrow$ Mehrprozessorsysteme werden in Zukunft weiter an Attraktivität gewinnen!

Mehrprozessorsysteme: Einführung IV

Skopus der Themenbehandlung

Problem: MP-Architektur ist großes, sehr heterogenes Gebiet, das *immer noch* relativ am Anfang seiner Entwicklung steht

- Kommen und Gehen vieler Ideen / Methoden / Systeme
- Mehr erfolglose als erfolgreiche Rechner / Architekturen

Wie viele der aktuellen Verfahren werden überdauern?

Fokus auf "*main stream*" MP-Systemen

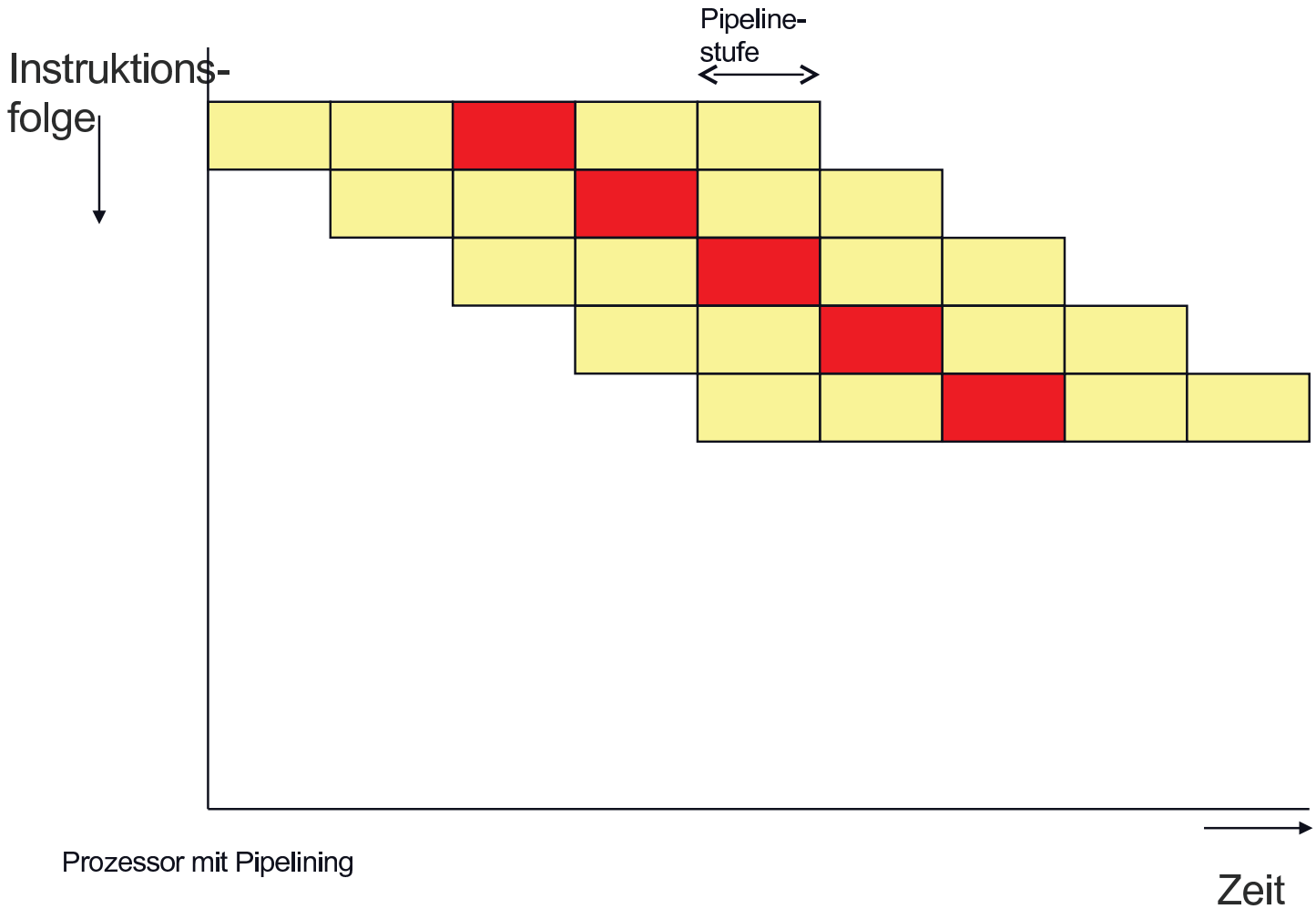
- Kleine bis mittlere Anzahl von Prozessoren (bis ca. 128)
- Keine Behandlung von sehr speziellen Architekturen mit extrem vielen Prozessoren ($\gg 128$)

Aber: Cluster als Hybride Lösung (= spezielle, vernetzte Uni-Prozessoren / "Standardrechner")

Welche Arten der Parallelität kennen wir?

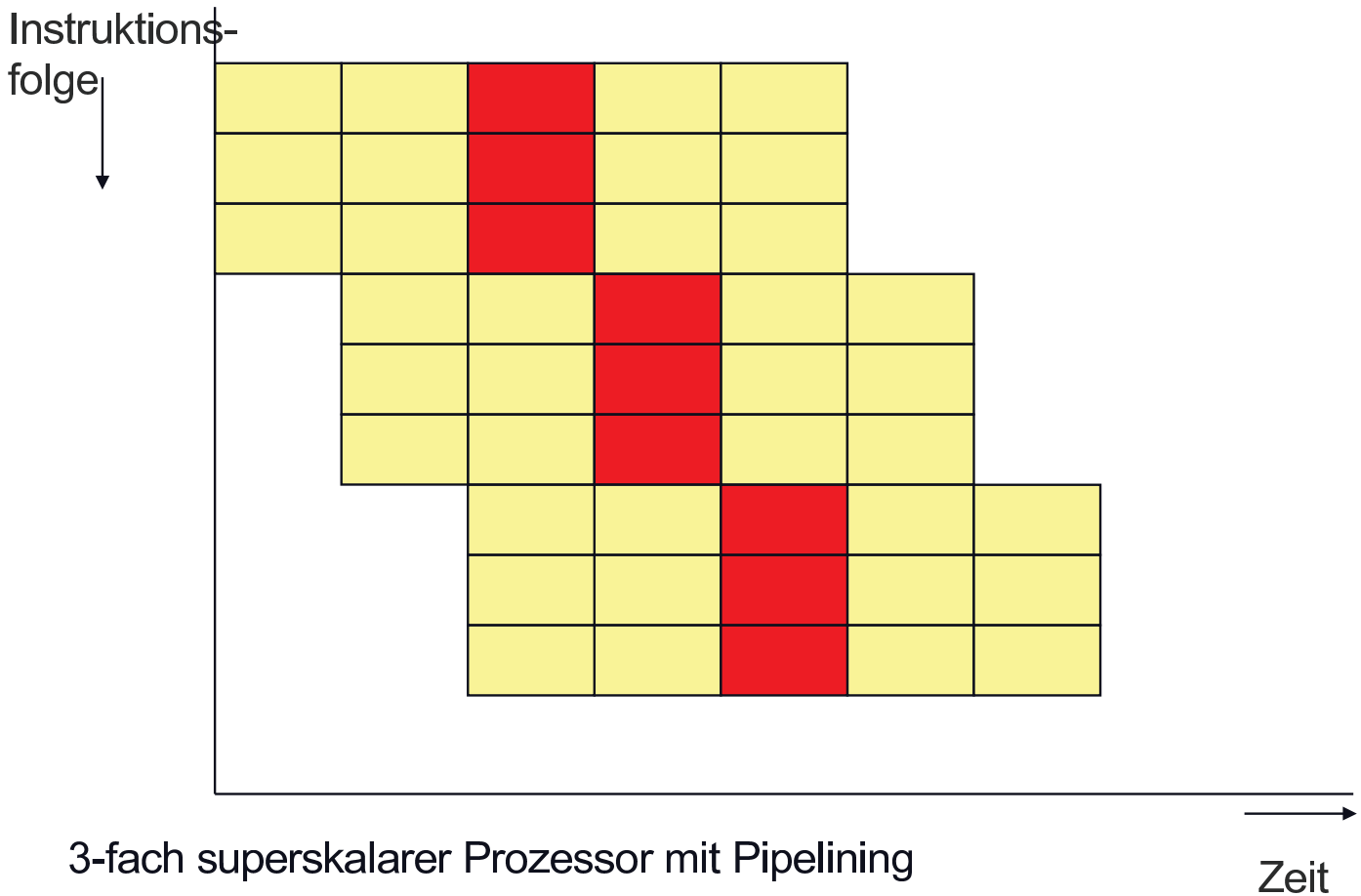
Parallelität: Pipelining

Quelle: R. Yahyapour, ehem. TU Dortmund



Parallelität: Superskalarität

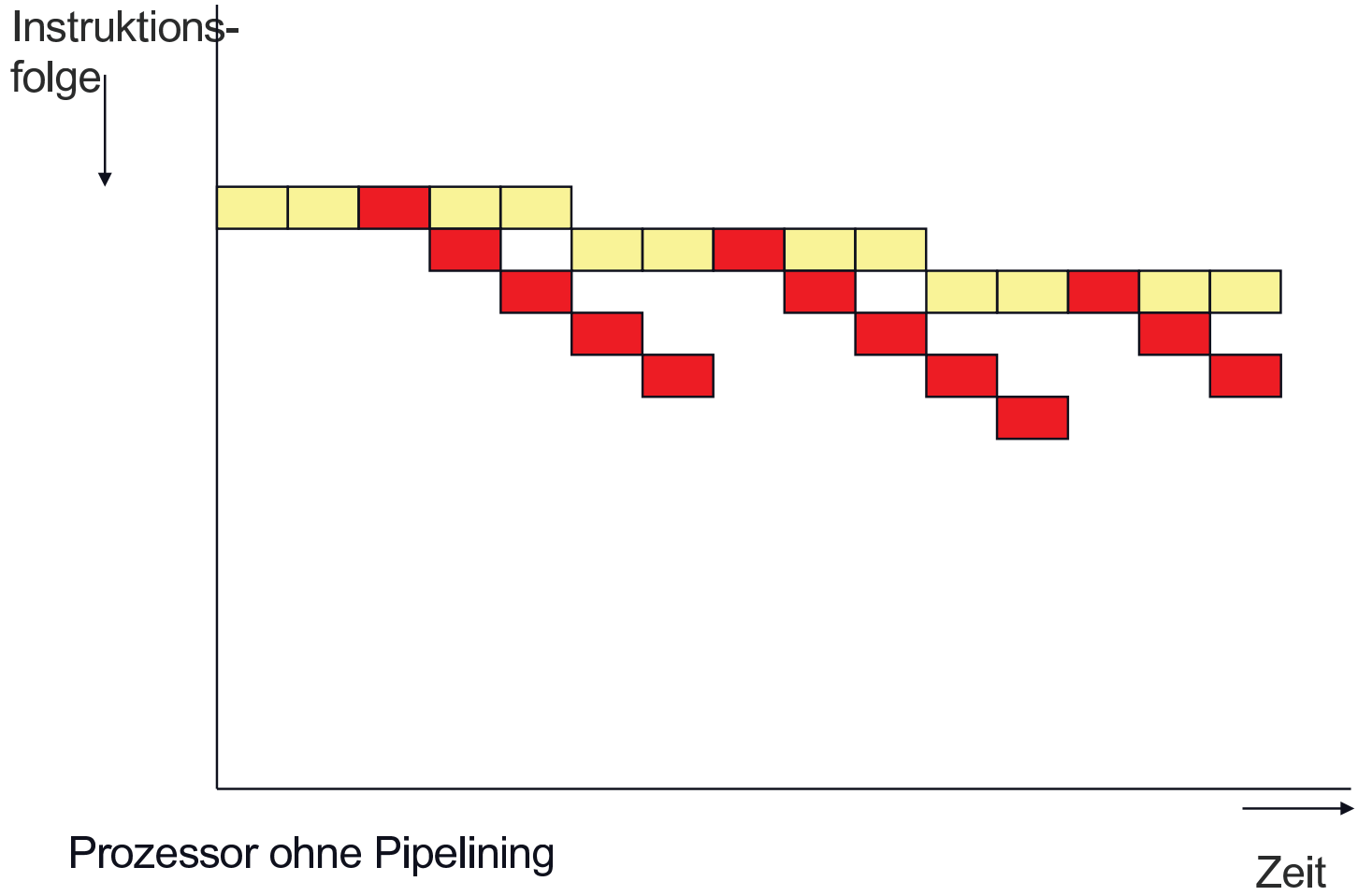
Quelle: R. Yahyapour, ehem. TU Dortmund



3-fach superskalarer Prozessor mit Pipelining

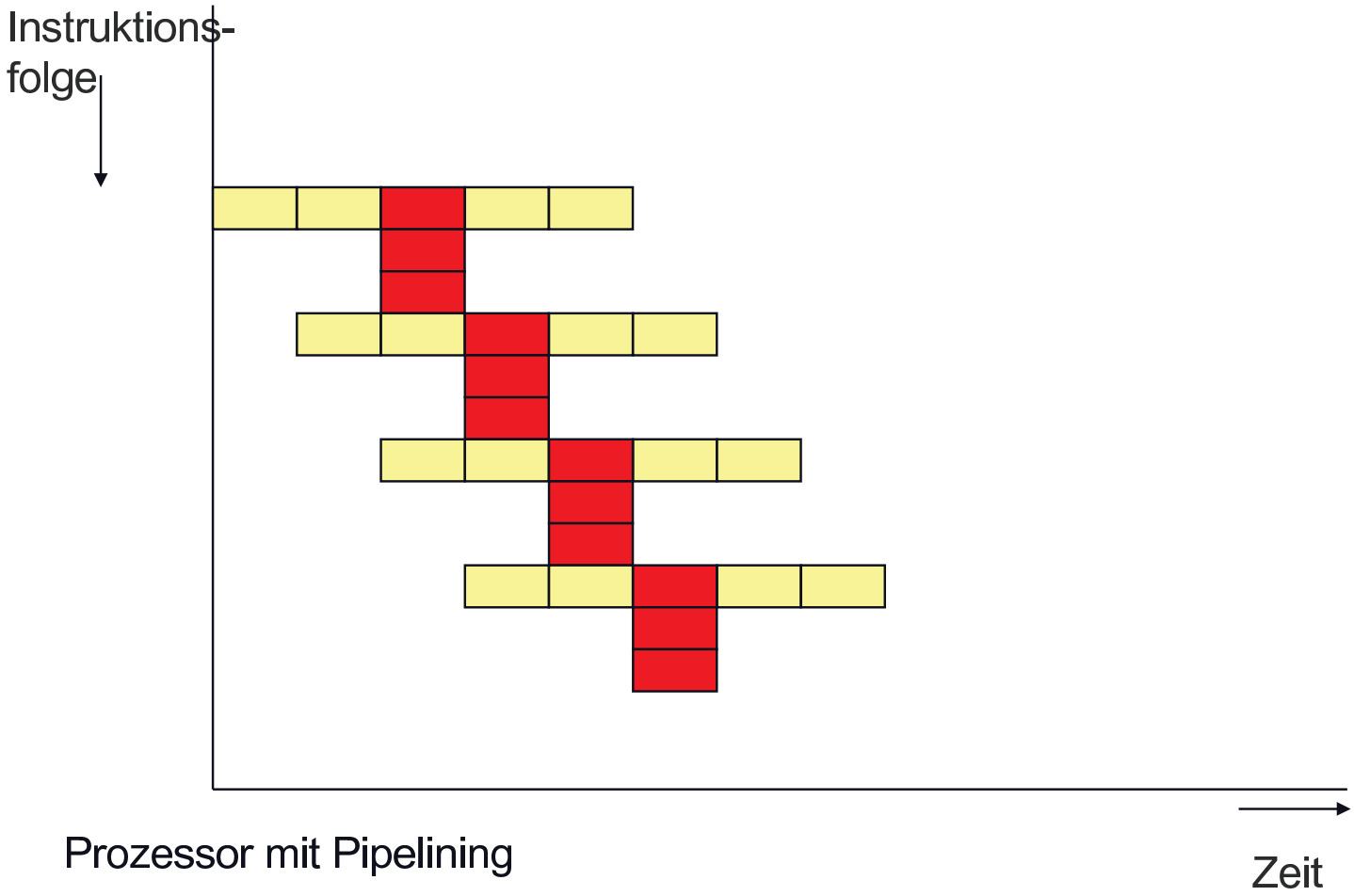
Parallelität: Vektorprozessoren

Quelle: R. Yahyapour, ehem. TU Dortmund



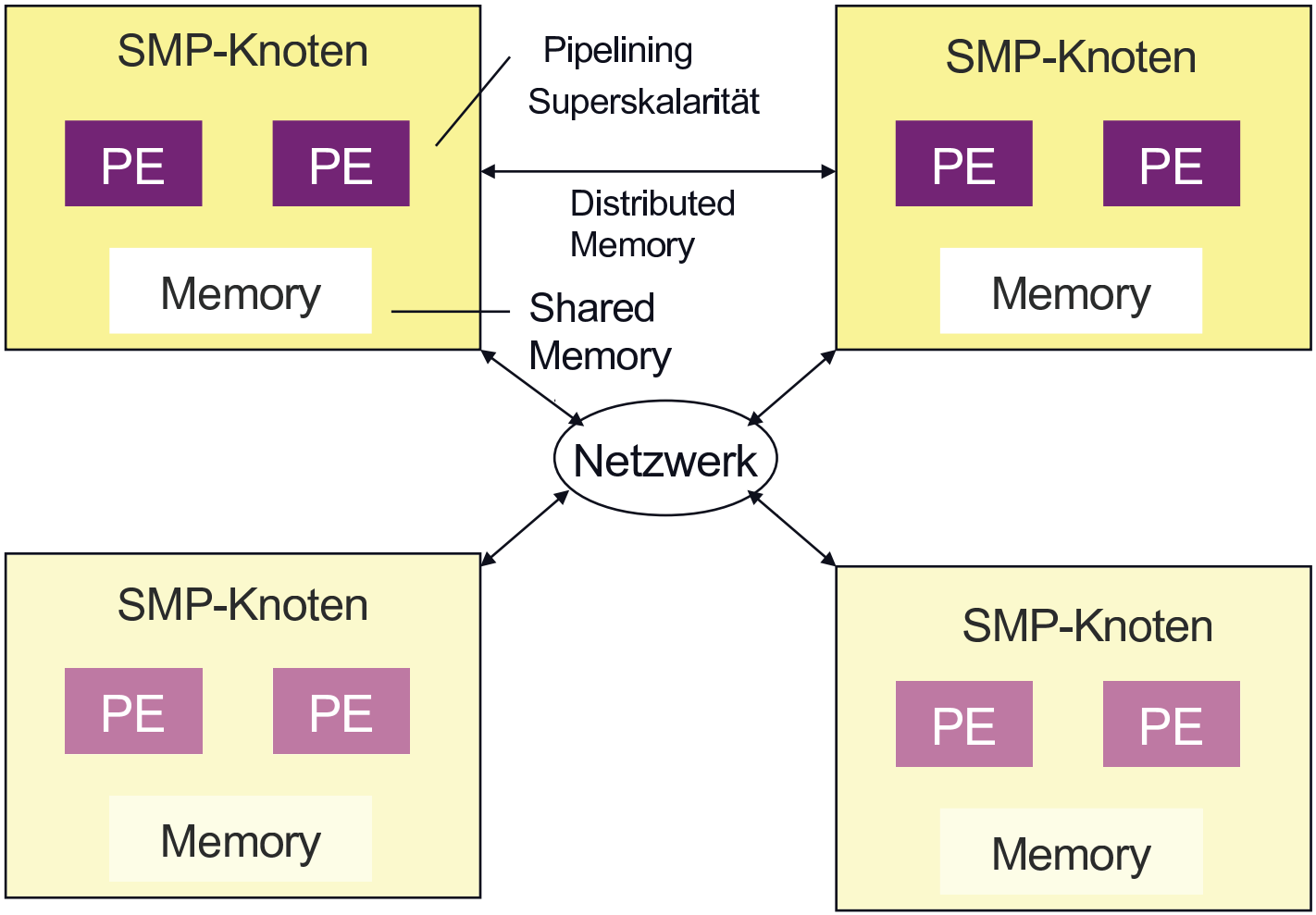
Parallelität: Very Long Instruction Word (VLIW)

Quelle: R. Yahyapour, ehem. TU Dortmund



MP-Beispiel: IBM Scalable Parallel

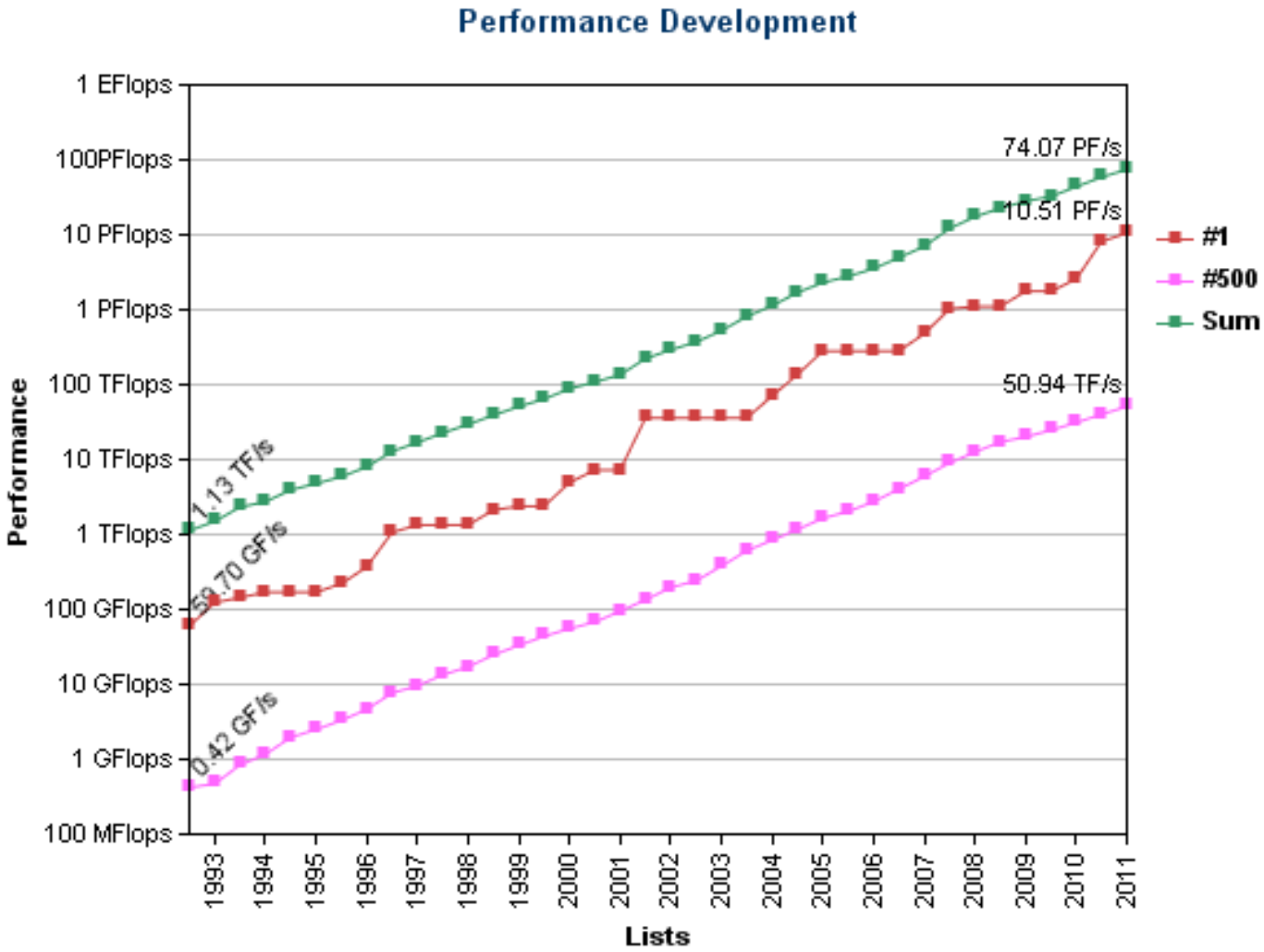
Quelle: R. Yahyapour, ehem. TU Dortmund



MP-Beispiel: JUGENE Cluster

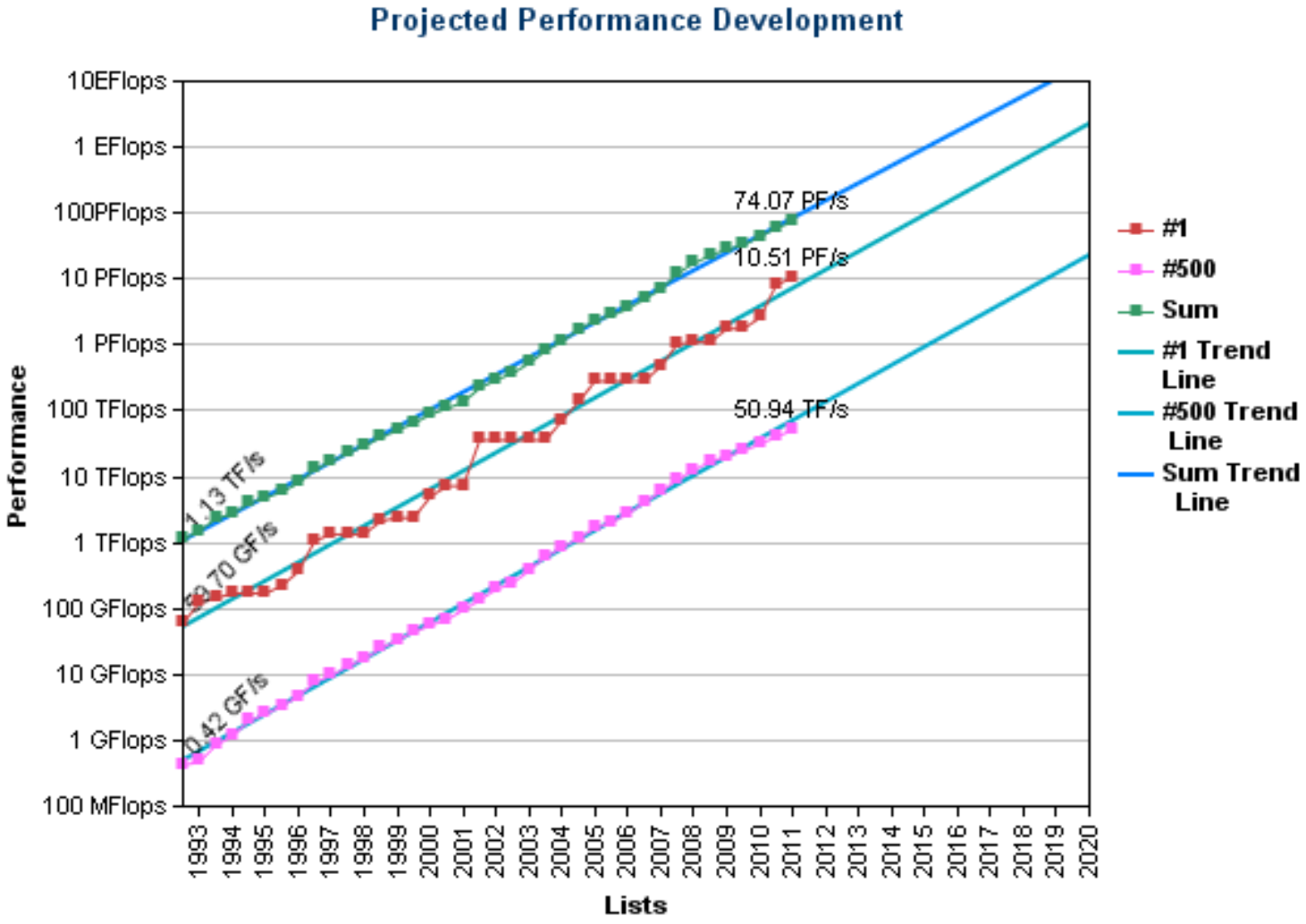
Quelle: www.physikblog.eu

MP-Leistungsentwicklung: Die Top500 Rechner



Quelle: www.top500.org

Ausblick: Exaflop-Rechner in 2017/18



Quelle: www.top500.org

Taxonomie von Mehrprozessorsystemen

Idee der Leistungssteigerung durch Paralleles Rechnen bereits i.d. Anfängen der Rechnerentwicklung verfolgt

Taxonomie möglicher Architekturen nach Flynn (1966)
(betrachtet Parallelismus im Befehls- und Datenstrom)

Single instruction, single data (SISD)

- Entspricht "klassischem/normalem" Uni-Prozessor-System

Single instruction, multiple data (SIMD)

- Gleicher Befehl(-sstrom) wird von mehreren Prozessoren auf unterschiedlichen Daten(strömen) ausgeführt
- Eigene Datenspeicher pro Prozessor, aber gemeinsamer Befehlsspeicher und Kontrolleinheit (zum Holen/Ausgeben der Befehle)

Beispiel: Vektorrechner
Vergleichbar: Multimedia-Befehle $\hat{=}$ eingeschränkte Variante

Taxonomie von Mehrprozessorsystemen II

Single instruction, single data (SISD) ...

Single instruction, multiple data (SIMD) ...

Multiple instruction, single data (MISD)

- Ex. Mehrere Verarbeitungsknoten, in Matrix angeordnet, bearbeiten verschiedene Befehlsströme
- Ein Datenstrom durchläuft Berechnungsmatrix
- Bezeichnet als systolisches Array

Multiple instruction, multiple data (MIMD)

- Jeder Prozessor bearbeitet eigenen Befehls- und eigenen Datenstrom
- Verwendung "normaler" Mikroprozessoren möglich!
- Verbreitetstes Architektur-Schema paralleler Rechner

Taxonomie von Mehrprozessorsystemen: Diskussion

- Grobe Kategorisierung $\Rightarrow$ Ex. "hybrideFormen
- SIMD und MISD nur für spezielle Berechnungen geeignet ($\Rightarrow$ keine kommerziellen MISD-Systeme)
- Viele frühe MPs nach SIMD-Prinzip, seit 1990er praktisch verschwunden (Ausnahme: Vektorrechner)
- MIMD ist vorherrschende Architektur für "*general purpose*" Mehrprozessorsysteme
  - Große Flexibilität (gesamtes MP für eine(n) Anwendung/Benutzer bis parallele Bearbeitung vieler unterschiedlicher Tasks/Prozesse)
  - Können von Kosten-Leistungs-Vorteilen gängiger Mikroprozessoren profitieren
    - $\Rightarrow$ praktisch alle heutigen MIMD-Systeme mit gleichen Mikroprozessoren wie Server/Desktops realisiert!

$\Rightarrow$ *Betrachten im Folgenden nur MIMD-Rechner!*

Klassen von MIMD-Rechnern

Unterscheidung nach Speicherorganisation und Verbindungsstrategie

(beachte: wesentlich beeinflußt durch Anzahl der Prozessoren)

1. *Architekturen mit zentralem Speicher*

- Heute ca. < 100 Prozessoren

$\Rightarrow$ Möglichkeit besteht, dass ...

  - alle Prozessoren gemeinsamen Speicher verwenden und ...
  - Prozessoren u. Speicher über Bussystem verbunden sind.

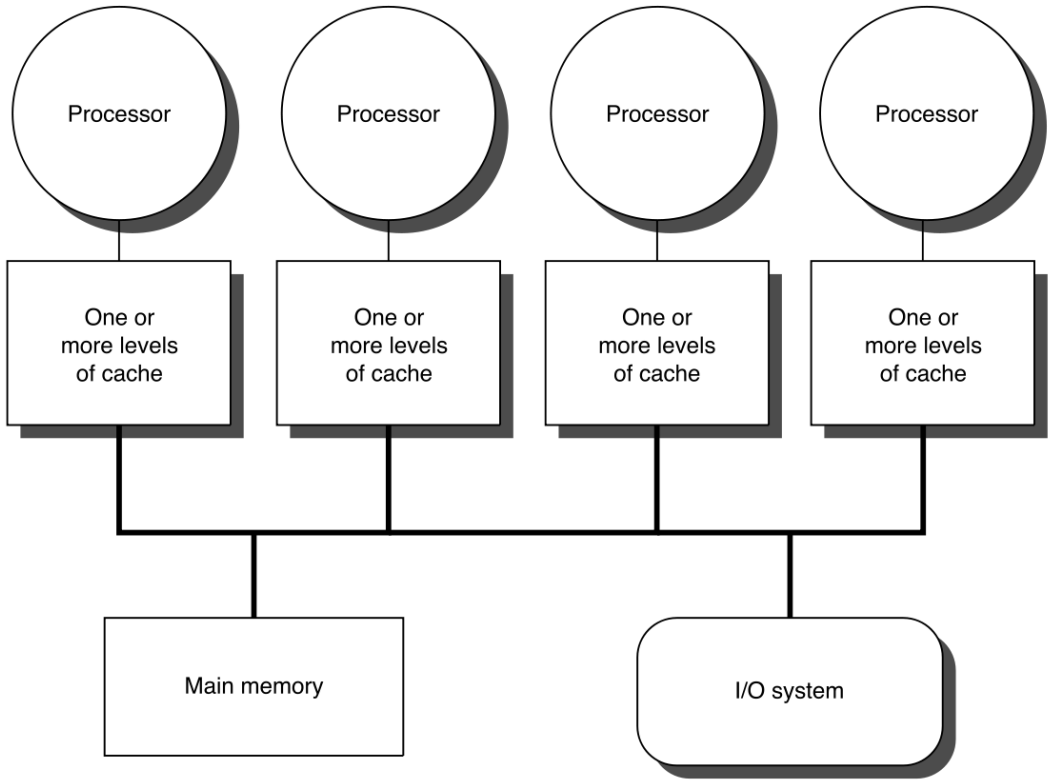
- Mit großen Caches kann Bus und zentraler Speicher Anforderungen weniger(!) Prozessoren befriedigen

Beachte: Zentraler Bus kann durch mehrere Busse oder *Switch* ersetzt werden

$\Rightarrow$ Skaliert ggf. bis wenige Dutzend Prozessoren

Aber: Nicht attraktiv bei weiterer Steigerung der Anzahl von Prozessoren!

Klassen von MIMD-Rechnern II



- Da zentraler Hauptspeicher symmetrisches Verhalten zu allen Prozessoren und uniforme Zugriffszeit ermöglicht:

⇒ *symmetric shared-memory multi processors* (SMP)

⇒ Speicherzugriff: *uniform memory access* (UMA)

✓ Verbreitetste Architektur für Mehrprozessorsysteme

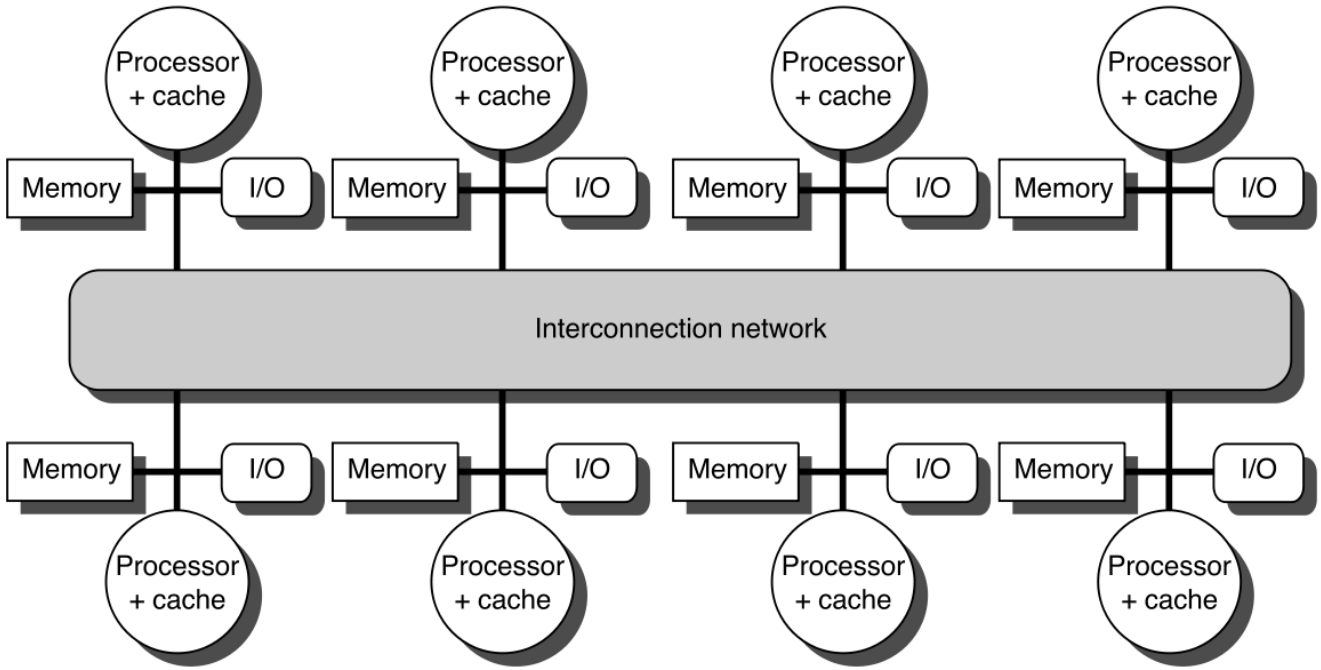
Klassen von MIMD-Rechnern III

2. *Architekturen mit verteiltem Speicher*

   Speicher ist physikalisch verteilt
   ⇒ kann größere Anzahl Prozessoren unterstützen
   (hohe Anforderungen an Speicherbandbreite nicht durch zentralen Speicher erfüllbar)
   Aber: Erfordert Verbindungsstruktur/-netz mit (relativ) großer Bandbreite zwischen Prozessorknoten (Bandbreite aber deutlich kleiner möglich, als bei Speicher!)
   Möglichkeiten der Verbindung

   - Direkte Verbindungsstrukturen ("Switches")
   - Indirekte Verbindung (via beteiligter Knoten; z.B. mehrdimensionale Gitter [später mehr])

Klassen von MIMD-Rechnern IV



- ✓ Kosteneffektive Methode, Bandbreite für lokale Zugriffe mit Prozessoranzahl skalierbar zu machen
- ✓ Auch Latenz lokaler Zugriffe reduziert
- ↯ Datenaustausch zwischen Prozessoren deutlich komplexer!
- ↯ Wesenlich größere Latenz “entfernter” Zugriffe (ab 100×)!

Modelle für Kommunikation und Speicherarchitektur

Problem: Wie Daten zwischen Prozessoren mit physikalisch verteiltem Speicher austauschen?

⇒ 2 grundlegende Methoden

1. **Gemeinsamer Adressraum** (shared address space)
   - Physikalisch getrennte Speicher werden zu einem logischen Adressraum zusammengefaßt
     ⇒ Jede Speicherstelle (lokal oder "entfernt") kann von jedem Prozessor mit passender Adresse angesprochen werden
   - Bezeichnet als *distributed shared-memory architecture* (DSM)
     Beachte: shared memory hier nur für Adressraum!
   - Im Gegensatz zu UMA-Architekturen (d.h. SMPs) hängt Zugriffszeit auf Speicher von konkretem Ort der physikalischen Speicherstelle ab
     ⇒ *non-uniform memory access* (NUMA) Architektur
   - Datenaustausch erfolgt implizit durch Speichertransferbefehle

Modelle für Kommunikation und Speicherarchitektur

2. **Mehrere private Adressräume**
   - Speicher sind logisch disjunkt, können von "entfernten" Prozessoren nicht angesprochen werden
   - Gleiche physikliasche Adresse bezieht sich auf unterschiedliche Speicherstellen in jedem Prozessor

   $\Rightarrow$ Prozessorknoten entsprechen *de facto* separaten Rechnern
   - Auch bezeichnet als *multi computer* [vs. multi processor] (bzw. Mehr-Rechner-System vs. Multi-Prozessor-System)

   Extremfall: multi computer aus separaten Rechnersystemen verbunden über lokales (ggf. spezialisiertes) Netzwerk

   $\Rightarrow$ Cluster (kosteneffektive MPs, falls wenig Kommunikation zwischen Rechenknoten erforderlich)

Herausforderung der Parallelverarbeitung

Zwei wesentliche Herausforderungen ex. beim Versuch, Parallelverarbeitung zur Leistungssteigerung einzusetzen:

1. Leistungssteigerung durch zusätzliche Prozessoren
2. Kosten der notwendigen Kommunikation

Fragen:

- *Welchen Leistungsgewinn erzielt ein N-Prozessor-System gegenüber einem Uni-Prozessor?*

  ... im allgemeinen?? ... für eine best. Aufgabe?

- Wieviel Overhead entsteht durch Datenaustausch?

  ... wieso eigentlich Datenaustausch?

Gesetze von Amdahl und Gustafson machen darüber Aussagen!

Das Amdahl'sche Gesetz

Welchen Leistungsgewinn erzielt ein N-Prozessor-System gegenüber einem Uni-Prozessor?

Speedup $$S_{\text{tot}} = \frac{\text{Laufzeit } \textit{ohne} \text{ Parallelisierung}}{\text{Laufzeit } \textit{mit} \text{ Parallelisierung}}$$

Nach Amdahl'schem Gesetz abhängig von:

- Anteil P des parallelisierbaren Codes des betreffenden Programms / der Anwendung
- Speedup S, der durch paralleles Rechnen erreicht wird
  (vereinfachende Annahme: Parallelisierung auf diesem Code [Anteil P] perfekt $\Rightarrow$ Speedup $S = N =$ Anz. der Prozessoren)

$$S_{\text{tot}} = \frac{1}{(1-P) + \frac{P}{S}} = \frac{1}{(1-P) + \frac{P}{N}}$$

Das Amdahl'sche Gesetz II

$$S_{\text{tot}} = \frac{1}{(1-P) + \frac{P}{S}} = \frac{1}{(1-P) + \frac{P}{N}}$$

- Betrachten Beispiel (jeweils $N = 64$ Prozessoren):
  - 80% parallelisierbarer Code $\Rightarrow$ Speedup ca. 4,7
  - 50% parallelisierbarer Code $\Rightarrow$ Speedup ca. 2
  - 20% parallelisierbarer Code $\Rightarrow$ Speedup ca. 1,25
- Maximaler Speedup bei beliebig vielen Prozessoren:

$$\lim_{N \to \infty} S_{\text{tot}} = \frac{1}{(1-P)}$$

Das Amdahl'sche Gesetz II

$$S_{\text{tot}} = \frac{1}{(1-P) + \frac{P}{S}} = \frac{1}{(1-P) + \frac{P}{N}}$$

- Betrachten Beispiel (jeweils $N = 64$ Prozessoren):
  - 80% parallelisierbarer Code $\Rightarrow$ Speedup ca. 4,7
  - 50% parallelisierbarer Code $\Rightarrow$ Speedup ca. 2
  - 20% parallelisierbarer Code $\Rightarrow$ Speedup ca. 1,25
- Maximaler Speedup bei beliebig vielen Prozessoren:

$$\lim_{N \to \infty} S_{\text{tot}} = \frac{1}{(1-P)}$$

Allein vom sequentiellen Programmanteil abhängig!

Das Amdahl'sche Gesetz II

$$S_{\text{tot}} = \frac{1}{(1-P) + \frac{P}{S}} = \frac{1}{(1-P) + \frac{P}{N}}$$

- Betrachten Beispiel (jeweils $N = 64$ Prozessoren):
  - 80% parallelisierbarer Code $\Rightarrow$ Speedup ca. 4,7
  - 50% parallelisierbarer Code $\Rightarrow$ Speedup ca. 2
  - 20% parallelisierbarer Code $\Rightarrow$ Speedup ca. 1,25
- Maximaler Speedup bei beliebig vielen Prozessoren:

$$\lim_{N \to \infty} S_{\text{tot}} = \frac{1}{(1-P)}$$

Allein vom sequentiellen Programmanteil abhängig!

Max. Speedup bei 80% parallel. Code: 5!
(d.h. 128 statt 64 Prozessoren kaum Gewinn [4,8 statt 4,7])

Das Gesetz von Gustafson (& Barsis)

... stellt quasi Gegenstandpunkt zum Amdahl'schen Gesetz dar:

$$S'_{\text{tot}} = N - (1 - P) \cdot (N - 1)$$

- Betrachten Beispiel (wieder jeweils $N = 64$ Prozessoren):
  - 80% parallelisierbarer Code $\Rightarrow$ Speedup ca. 51,4
  - 50% parallelisierbarer Code $\Rightarrow$ Speedup ca. 32,5
  - 20% parallelisierbarer Code $\Rightarrow$ Speedup ca. 13,6

Wieso dieser (scheinbare) Widerspruch zum Amdahl'schen Gesetz?

Das Gesetz von Gustafson (& Barsis)

... stellt quasi Gegenstandpunkt zum Amdahl'schen Gesetz dar:

$$S'_{\text{tot}} = N - (1 - P) \cdot (N - 1)$$

- Betrachten Beispiel (wieder jeweils $N = 64$ Prozessoren):
  - 80% parallelisierbarer Code $\Rightarrow$ Speedup ca. 51,4
  - 50% parallelisierbarer Code $\Rightarrow$ Speedup ca. 32,5
  - 20% parallelisierbarer Code $\Rightarrow$ Speedup ca. 13,6

Wieso dieser (scheinbare) Widerspruch zum Amdahl'schen Gesetz?

Annahme: Problemgröße wächst linear mit der Anzahl d. Prozessoren!

$$S'_{\text{tot}} = \frac{(1 - P) + P \cdot N}{(1 - P) + P}$$

Amdahl vs. Gustafson

Eine automobilistische Metapher: (Quelle: Wikipedia)

Amdahl's Law approximately suggests:

"Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph. No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city. Since it has already taken you 1 hour and you only have a distance of 60 miles total; going infinitely fast you would only achieve 60 mph."

Gustafson's Law approximately states:

"Suppose a car has already been traveling for some time at less than 90mph. Given enough time and distance to travel, the car's average speed can always eventually reach 90mph, no matter how long or how slowly it has already traveled. For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour, and so on."

Leistungsbewertung

Problem:

- Ermittlung des Speedup erfordert immer die Betrachtung eines Programms/einer Anwendung
- Die theoretische "Peak Performance" ist häufig weit entfernt von der realistischen Leistung eines Systems

Lösungsansatz: Benchmarks für die Bewertung der Rechenleistung

Beispiele:

- *NAS Parallel Benchmarks* (http://www.nas.nasa.gov/)
  "... set of programs designed to help evaluate the performance of parallel supercomputers. ... derived from computational fluid dynamics (CFD) applications and consist of five kernels ..."
- LINPACK (http://www.netlib.org/benchmark/hpl/)
  Ursprünglich numerische Programmbibliothek zum Lösen von lin. Gleichungssystemen

Achtung: Einfluss von Programmierung, Compiler, ...