Kapitel 11
Parallele Rechnerstrukturen

Parallele Prozessorarchitekturen
Parallelrechner

11.4 Parallele Rechnerstrukturen

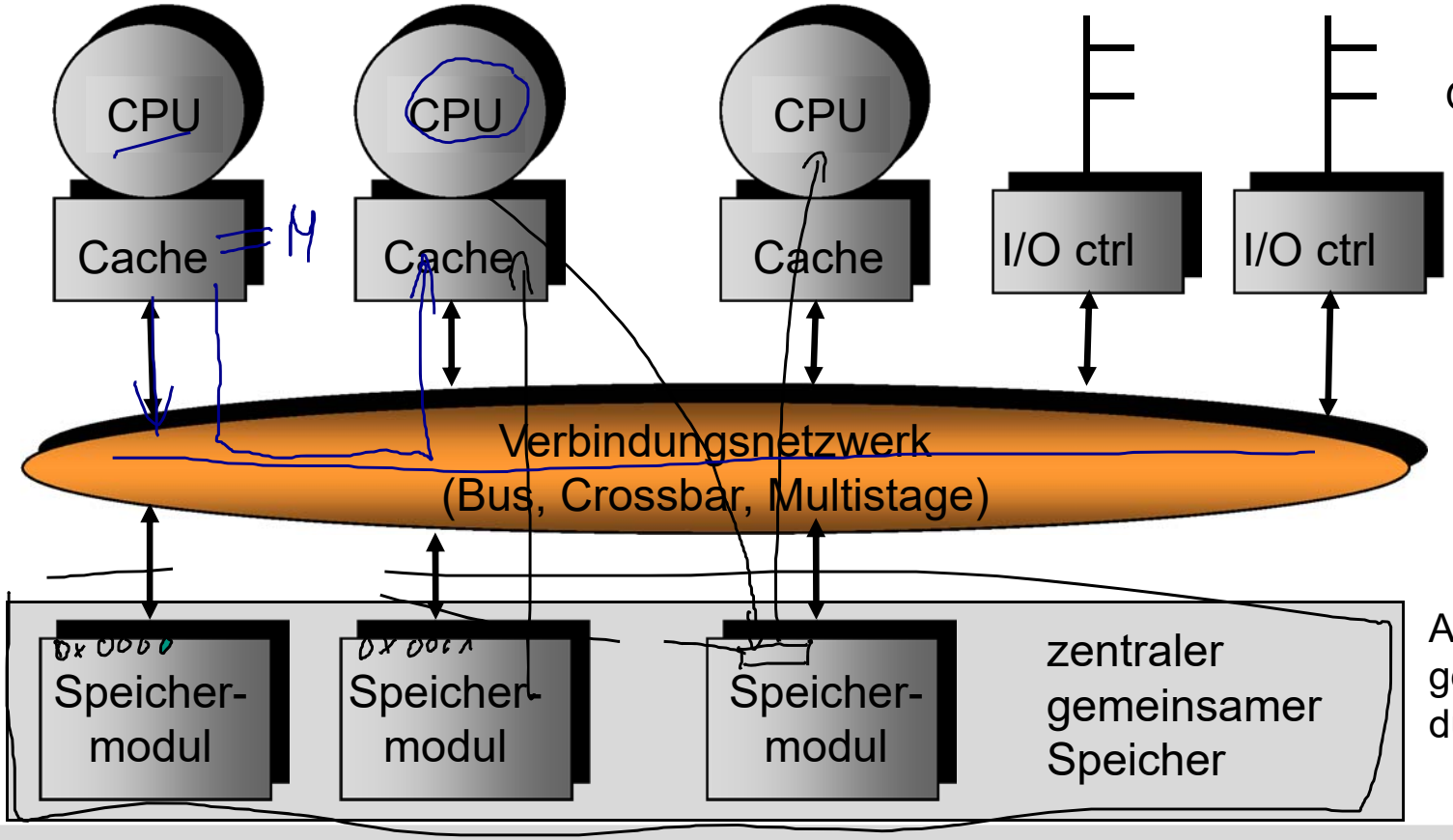
Parallele Architekturen

- Definition Parallelrechner:
  - *„A collection of processing elements that communicate and cooperate to solve large problems“* (Almase and Gottlieb, 1989)
  - Betrachtung einer parallelen Architektur als eine Erweiterung des Konzepts einer konventionellen Rechnerarchitektur um eine Kommunikationsarchitektur

11.4 Parallele Rechnerstrukturen

Multiprozessor mit gemeinsamem Speicher

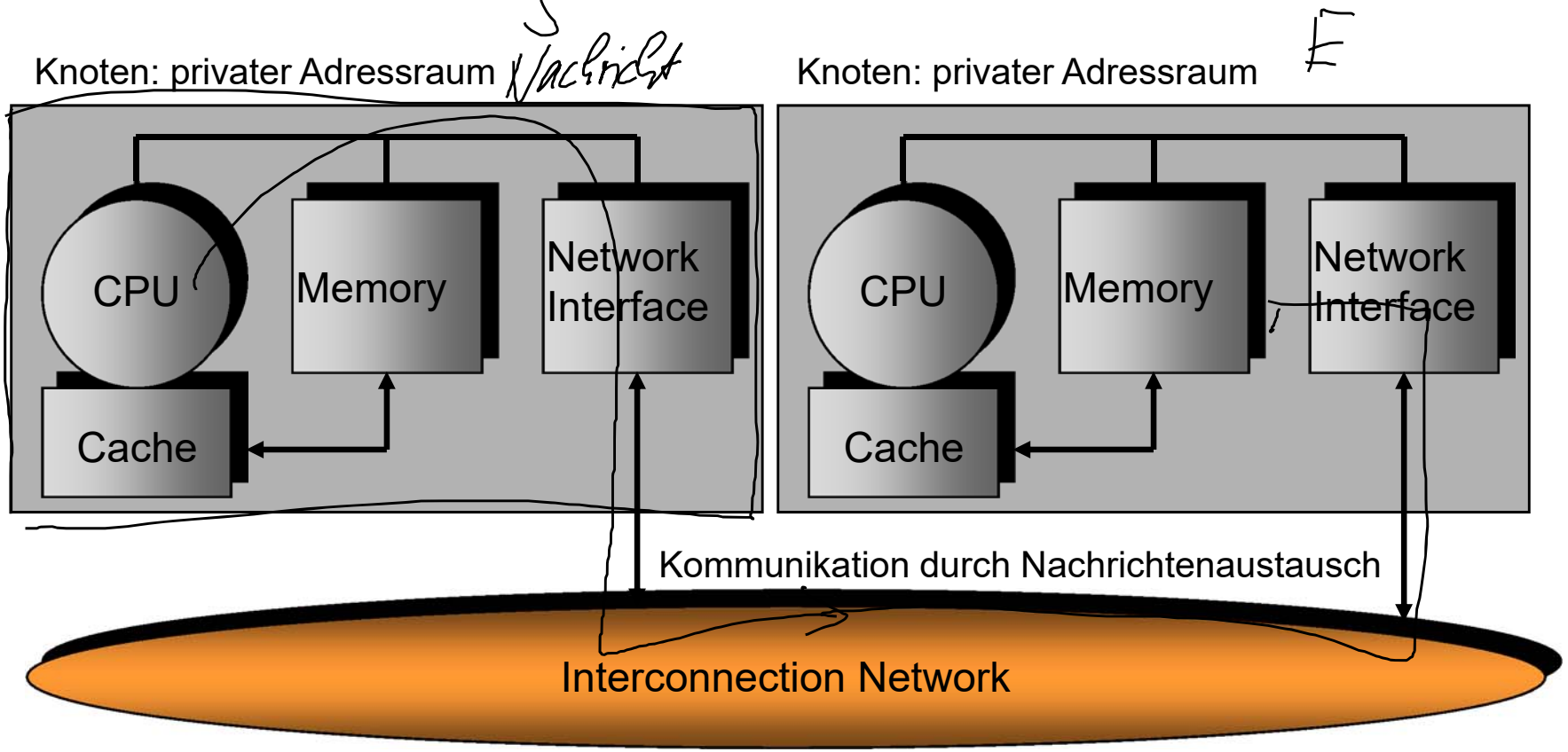
- **UMA**: Uniform Memory Access
- Beispiele: **symmetrischer Multiprozessor (SMP), Multicore-Prozessor**
  - Gleichberechtigter Zugriff der Prozessoren auf die Betriebsmittel



11.4 Parallele Rechnerstrukturen

Multiprozessor mit verteiltem Speicher

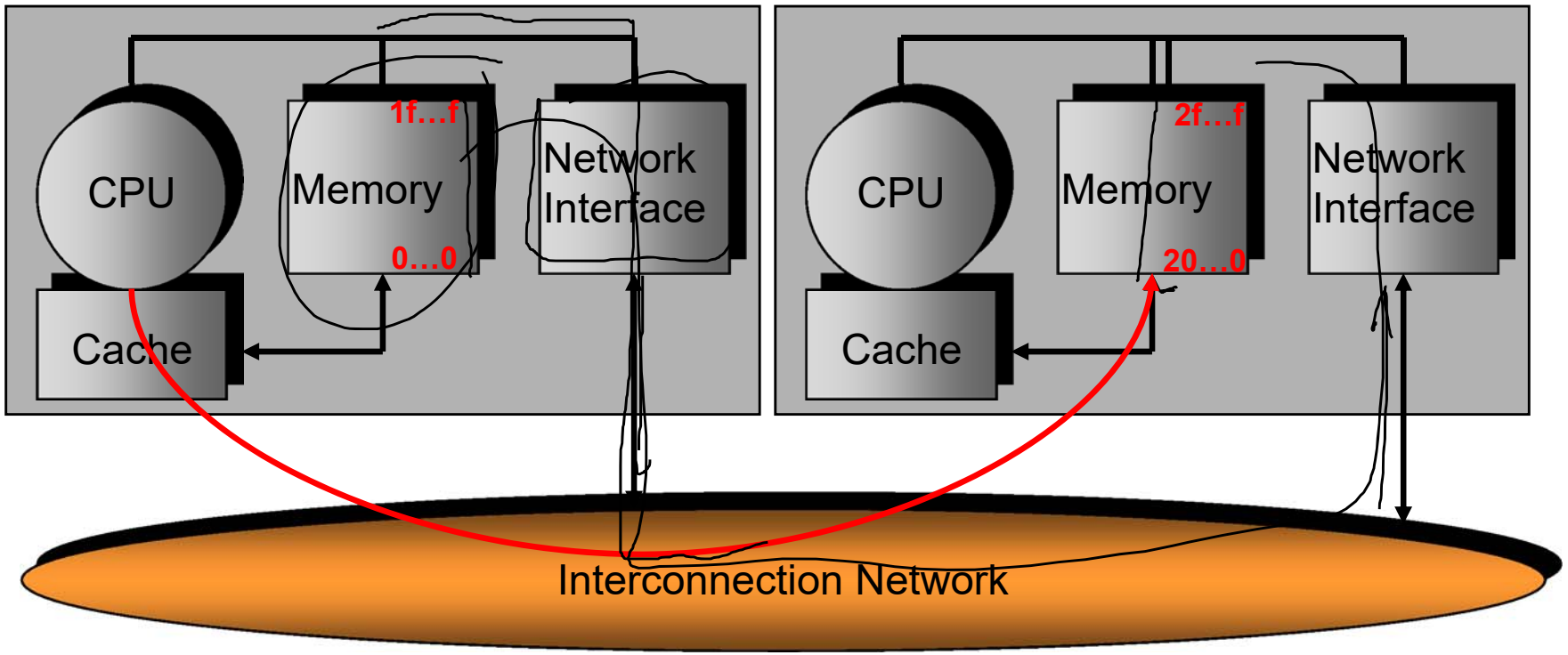
- **NORMA** No Remote Memory Access
- Beispiel: **Cluster**



11.4 Parallele Rechnerstrukturen

Multiprozessor mit verteiltem gemeinsamen Speicher

- **NUMA**: Non-Uniform Memory Access
- **CC-NUMA**: Cache-Coherent Non-Uniform Memory Access
  - Globaler Adressraum: Zugriff auf entfernten Speicher über load / store Operationen



11.4 Parallele Rechnerstrukturen

Parallele Programmiermodelle

- Abstraktion einer parallelen Maschine, auf der der Anwender sein Programm formuliert
- Spezifiziert, wie Teile des Programms parallel abgearbeitet werden, wie Informationen ausgetauscht werden und welche Synchronisationsoperationen verfügbar sind, um die Aktivitäten zu koordinieren
- Üblicherweise implementiert als Erweiterungen einer höheren Programmiersprache wie C/C++ oder Fortran

11.4 Parallele Rechnerstrukturen

Parallele Programmierung

- Formulierung eines parallelen Programms
  - Aufteilung der Arbeit (work partitioning)
    - Identifizieren der Teilaufgaben, die parallel ausgeführt und auf die Prozessoren verteilt werden können
    - Thread oder Prozess:
      - Code , der auf einem Prozessor oder Prozessorkern eines Multiprozessors ausgeführt wird
  - Koordination (coordination)
    - Parallel auf den verschiedenen Prozessoren laufende Threads müssen koordiniert werden, so dass das Ergebnis dasselbe ist wie bei einem entsprechenden sequentiellen Programm
    - Synchronisation der Threads
    - Kommunikation bzw. Austausch von Teilergebnissen zwischen den Threads

11.4 Parallele Rechnerstrukturen

Parallele Programmierung (Aufteilung der Arbeit)

- Ausgehend von einem sequentiellen Programm müssen die Teile identifiziert werden, die parallel ausgeführt werden können
  - Zwei Programmsegmente S1 und S2, die in einem sequentiellen Programm nacheinander ausgeführt werden, können parallel ausgeführt werden, wenn S1 unabhängig von S2 ist
    - Das parallele Programm ist konform zur sequentiellen Semantik

11.4 Parallele Rechnerstrukturen

Parallele Programmierung (Aufteilung der Arbeit)

- Formen des Parallelismus
  - **Datenparallelismus** (data-level parallelism)
    - Berechnungen von verschiedenen Datenelementen sind unabhängig (Feld, Matrix)
      - Beispiel: Matrixmultiplikation
        - Die Berechnung eines Elements der Ergebnismatrix ist unabhängig von der Berechnung der anderen Elemente
    - Single Program Multiple Data (SPMD)
      - Eine Berechnung (eine Funktion) wird auf alle Datenelemente eines Feldes ausgeführt
      - Skalierung der Problemgröße
  - **Funktionsparallelismus** (function-level-parallelism, task-level parallelism)
    - Unabhängige Funktionen werden auf verschiedenen Prozessoren ausgeführt

11.4 Parallele Rechnerstrukturen

Parallele Programmierung (Koordination)

- Synchronisation und Kommunikation
  - Austausch von Informationen über gemeinsamem Speicher oder über explizite Nachrichten
  - Zusätzlicher Zeitaufwand hat Auswirkung auf die Ausführungszeit des parallelen Programms

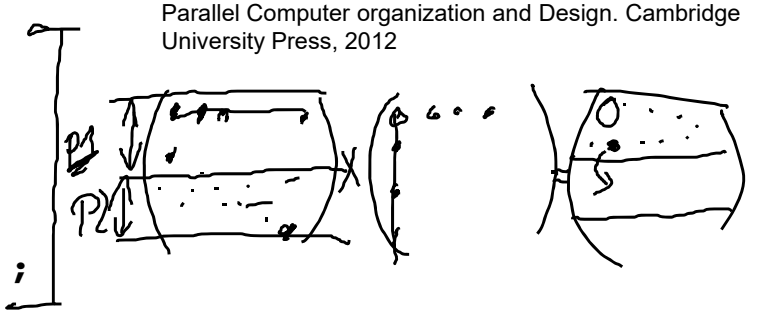
11.4 Parallele Rechnerstrukturen

Parallele Programmierung

- Beispiel: Pseudocode für einen sequentiellen Algorithmus zur Multiplikation zweier Matrizen:

```
1   sum = 0;
2   for (i=0, i<N,i++)
3      for (j=0, j<N, j++){
4         C[i,j] = 0;
5         for (k=0, k<N, k++)
6            C[i,j] = C[i,j] + A[i,k]*B[k,j];
7       sum += C[i,j];
8      }
```

Beispielcode aus: Dubois, M; Annavaram, M.; Stenström, P.: Parallel Computer organization and Design. Cambridge University Press, 2012



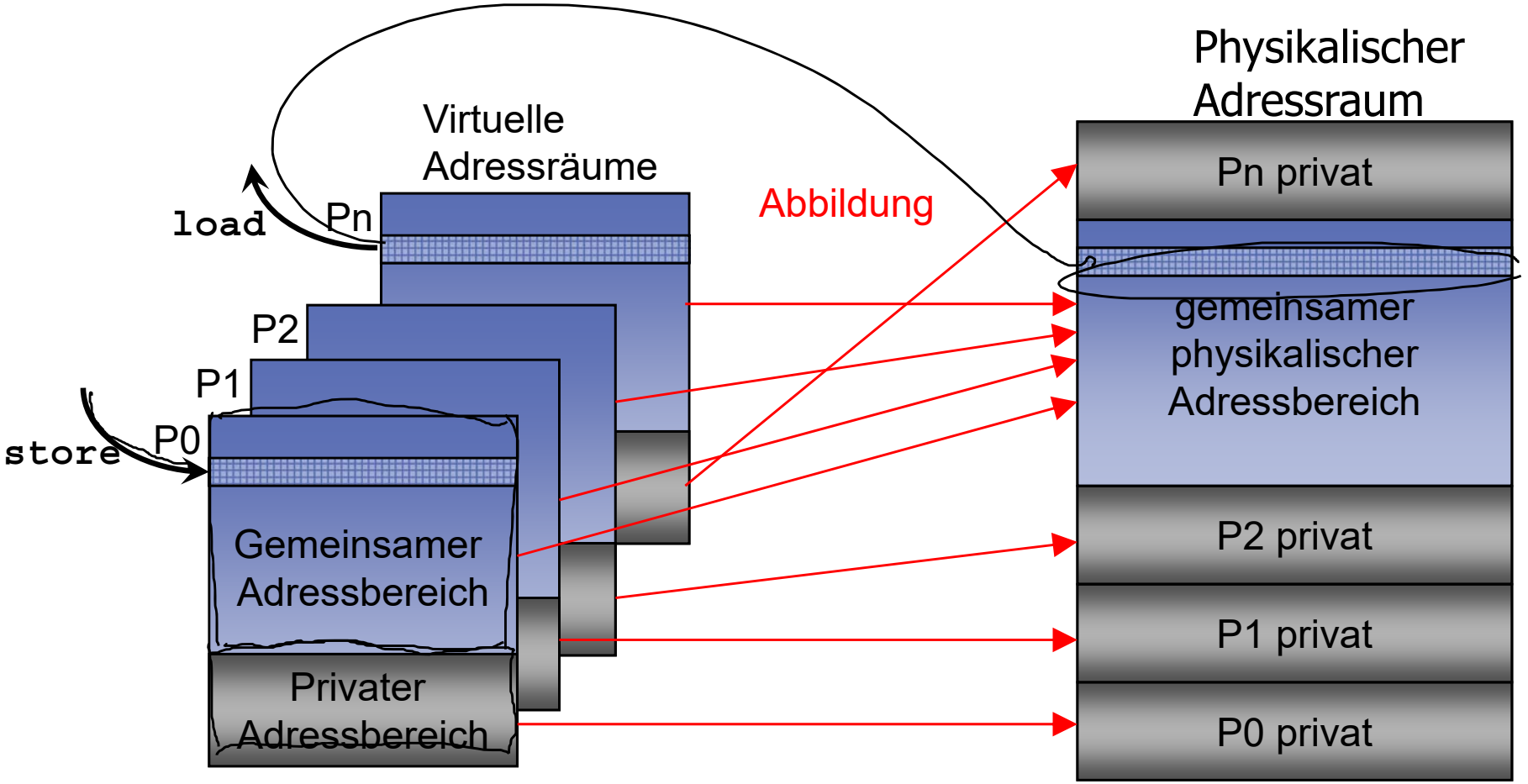
- Datenparallelismus: die Berechnung eines Matrixelements ist unabhängig von der Berechnung der anderen Elemente
- Grundsätzlich können alle Elemente parallel berechnet werden, aber es muss der Aufwand für das Aufsetzen der Threads mit dem Gewinn an Rechenleistung abgewogen werden

11.4 Parallele Rechnerstrukturen

- **Gemeinsamer Speicher (Shared Memory)**
  - Kommunikation und Koordination von Prozessen (Threads) über **gemeinsame Variablen** und Zeiger, die gemeinsame Adressen referenzieren
  - **Kommunikationsarchitektur**
    - Verwendung konventioneller Speicheroperationen für die Kommunikation über gemeinsame Adressen
    - Atomare Synchronisationsoperationen

11.4 Parallele Rechnerstrukturen

Gemeinsamer Speicher (Shared Memory)



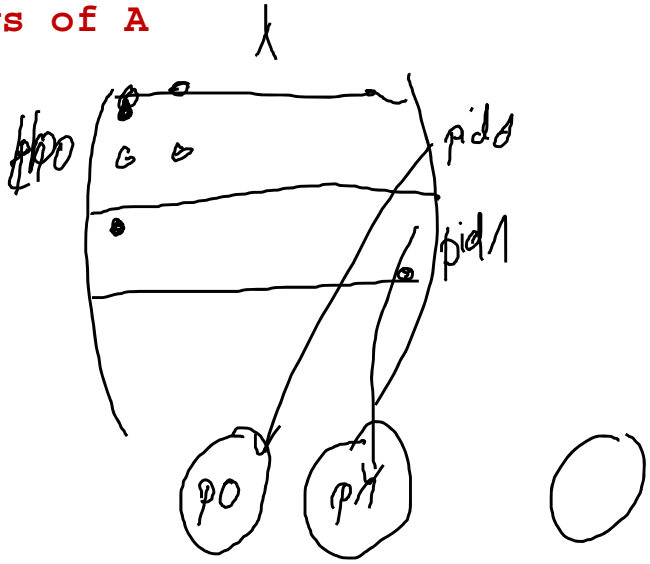
11.4 Parallele Rechnerstrukturen

Parallele Programmierung: Shared Memory

- Beispiel: Pseudocode für einen parallelen Algorithmus:

```
/* A, B, C, BAR, LV and sum are shared
/* All other variables are private
1a low = pid*N/nproc;          /* pid=0…nproc-1
1b hi = low + N/nproc;         /* identifies rows of A
1c mysum = 0; sum = 0;
2   for (i=low, i<hi,i++)
3      for (j=0, j<N, j++){
4         C[i,j] = 0;
5         for (k=0, k<N, k++)
6            C[i,j] = C[i,j] + A[i,k]*B[k,j];
7         mysum += C[i,j];
8   }
9   Barrier (Bar);
10  Lock(LV);
11     sum += mysum;
12  UNLOCK(LV);
```

Kritischer Bereich: nur ein Thread kann den Code in diesem Bereich zu einem Zeitpunkt ausführen



Beispielcode aus: Dubois, M; Annavaram, M.; Stenström, P.: Parallel Computer organization and Design. Cambridge University Press, 2012

11.4 Parallele Rechnerstrukturen

Parallele Programmierung: Shared Memory

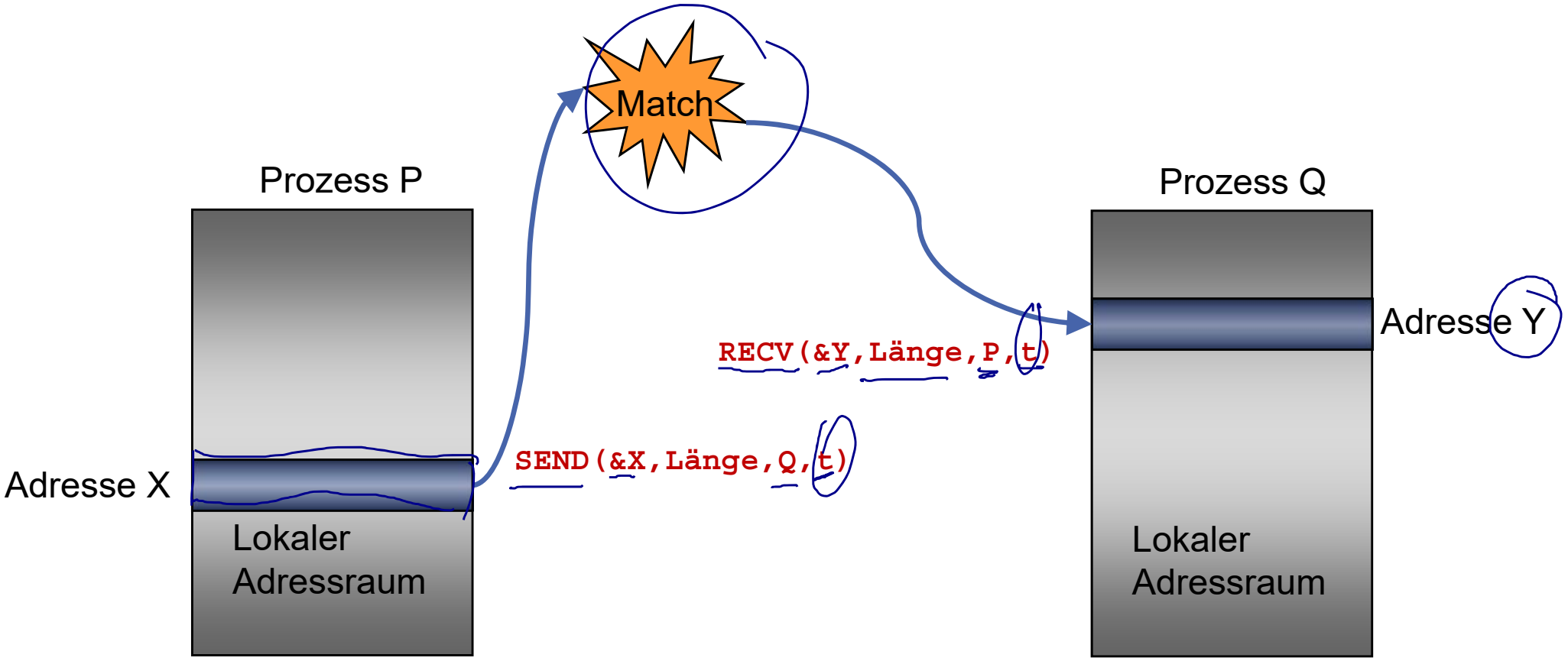
- Beispiel: Pseudocode für einen parallelen Algorithmus (Anmerkungen):
  - Die Arbeit wird auf *nproc* Threads aufgeteilt, wobei $\mathrm{nproc} = \frac{\mathrm{N}}{2^{\mathrm{i}}}$ $\text{für}\ \mathrm{i} = 0{,}1, \ldots, \log_2 \mathrm{N}$ ist
  - Der erste Thread berechnet die Elemente für die ersten N/nproc Reihen, der zweite die nächsten N/nproc Reihen usw.
  - Jedem Thread wird eine eindeutige Nummer zugeordnet (*pid*) im Bereich *[0, nproc-1]*
  - Die Variablen *low, hi* definieren die Indices der aufeinanderfolgenden Reihen
  - Neben dem Matrixprodukt wird die Summe Elemente der Ergebnismatrix berechnet, wofür jeder Thread die private Variable *mysum* verwendet
  - Jeder Thread addiert seine Summe auf die globale Variable *sum* im kritischen Bereich
  - Die Threads werden asynchron abgearbeitet, weshalb ein Thread die Arbeit beenden kann, bevor ein anderer mit seiner Berechnung startet: Synchronisation mit Hilfe Barrier

11.4 Parallele Rechnerstrukturen

- **Nachrichtenorientiertes Programmiermodell (Message Passing)**
  - Kommunikation der Prozesse (Threads) mit Hilfe von **Nachrichten**
    - Kein gemeinsamer Adressbereich
  - **Kommunikationsarchitektur**
    - Verwendung von korrespondierenden Send- und Receive-Operationen
    - **Send**: Spezifikation eines lokalen Datenpuffers und eines Empfangsprozesses (auf einem entfernten Prozessor)
    - **Receive**: Spezifikation des Sende-Prozesses und eines lokalen Datenpuffers, in den die Daten ankommen

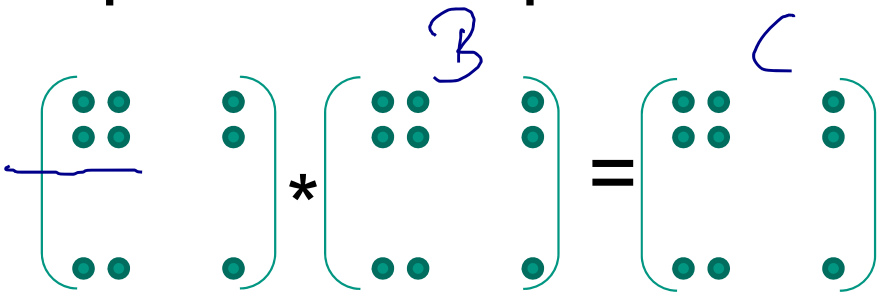
11.4 Parallele Rechnerstrukturen

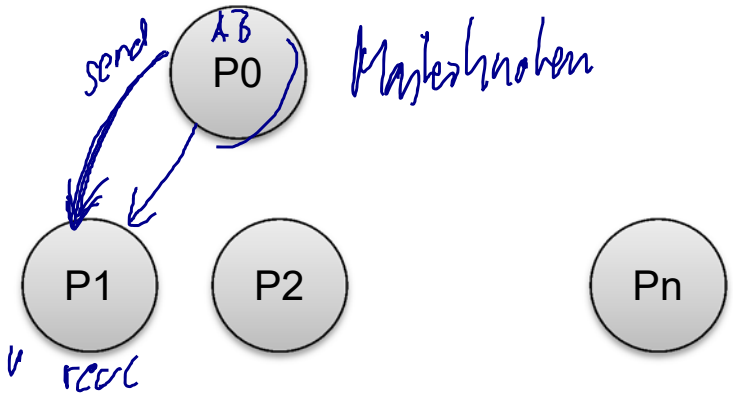
- **Nachrichtenorientiertes Programmiermodell (Message Passing)**



11.4 Parallele Rechnerstrukturen

- **Nachrichtenorientiertes Programmiermodell (Message Passing)**
  - **Beispiel: Matrix-Multiplikation**





Parallele Programmierung: Message Passing

- Beispiel: Pseudocode für einen parallelen Algorithmus (Teil 1):

```
1a myN = N/nproc
1b if(pid == 0)
1c    for(i=1, i<nproc, i++){
1d       k=i*N/nproc;
1e       SEND(&A[k][0],myN*N*sizeof(float),i,IN1);
1f       SEND(&B[0][0],N*N*sizeof(float),i,IN2);
1g    } else {
1h       RECV(&A[0][0],myN*N*sizeof(float),0,IN1);
1i       RECV(&B[0][0],N*N*sizeof(float),0,IN1);
1j    }
1k mysum = 0;
2  for (i=0, i<myN,i++)
3     for (j=0, j<N, j++){
4        C[i,j] = 0;
5        for (k=0, k<N, k++)
6           C[i,j] = C[i,j] + A[i,k]*B[k,j];
7        mysum += C[i,j];
8  }
```

Code für jeden Thread

Beispielcode aus: Dubois, M; Annavaram, M.; Stenström, P.: Parallel Computer organization and Design. Cambridge University Press, 2012

Parallele Programmierung: Message Passing

- Beispiel: Pseudocode für einen parallelen Algorithmus (Teil 2):

```
9   if(pid == 0){
10     sum = mysum;
11     for(i=1, i<nproc, i++){
12        RECV(&mysum,sizeof(float),i,SUM);
13        sum +=mysum;
14     }
15     for(i=1, i<nproc,i++){
16         k = i*N/nproc
17         RECV(&C[k][0],myN*N*sizeof(float),i,RES);
18     }
19   } else{
20        SEND(&mysum,sizeof(float),0,SUM);
21        SEND(&C[0][0],myN*N*sizeof(float),0,RES);
22     }
```

Code für jeden Thread

Beispielcode aus: Dubois, M; Annavaram, M.; Stenström, P.: Parallel Computer organization and Design. Cambridge University Press, 2012

11.4 Parallele Rechnerstrukturen

Parallele Programmierung: Message Passing

- Beispiel: Pseudocode für einen parallelen Algorithmus (Erläuterungen):
  - Die Matrizen A und B werden initial auf einem Knoten (Master Node) gehalten mit *pid=0*
  - Der Master Node teilt die Matrizen auf die anderen Knoten auf und übergibt einen Block von *N/nproc* Reihen von Matrix *A* an jeden anderen Thread (Zeilen 1b-1f)
  - Korrespondierende SEND/RECV Primitive
    - Parameter: Startadresse der lokalen Datenstruktur, Länge der Nachricht, die ID des Empfängers/Senders, Tag zur Unterscheidung von einem anderen Nachrichtenaustausch
  - Während nur ein Teil der Matrix *A* von der Größe *myN* verschickt wird (Zeile 1e), muss die gesamte Matrix *B* an alle Threads geschickt werden (Zeile 1f)
  - Beim Empfänger wird die Partition der Matrix *A* und die gesamte Matrix *B* aus dem Puffer in den lokalen Adressraum (Zeilen 1h und 1i) kopiert
  - Die Berechnung (Zeilen 2-8) ist ähnlich dem Shared-Memory Code mit dem Unterschied, dass jeder Prozess seine ihm zugewiesene Partition berechnet bestehend aus *myN* Reihen
  - Diese lokale Partition der Ergebnismatrix muss an den Master Node zurückkopiert werden
  - Jeder Prozess berechnet die Teilsumme in einer privaten Variablen *mysum*

11.4 Parallele Rechnerstrukturen

Parallele Programmierung: Message Passing

- Beispiel: Pseudocode für einen parallelen Algorithmus (Erläuterungen):
  - Der Masterprozess hat die *pid =0* und ist verantwortlich für die Berechnung der Summe. Alle anderen Prozesse senden ihre jeweilige Teilsumme an den Prozess mit *pid=0* (Zeile 20)
  - Der Masterprozess sammelt diese Teilsummen und addiert die Teilergebnisse auf die Variable *sum* in der for-Schleife von Zeile 11-14
  - Die Partition des Matrixprodukts, die von jedem Prozess berechnet wird, wird mit dem SEND-Kommando in Zeile 21
  - Der Code, der vom Masterprozess ausgeführt wird, um jede Partition in die Ergebnismatrix zu kopieren, ist in Zeile 15-18

  - Synchronisation ist implizit in den SEND/RECV Primitiven
    - Synchrone SEND/RECV Primitive blockieren bis jeder beteiligte Partner den jeweils anderen informiert hat, dass die Nachricht ausgetauscht worden ist.
      - Keine explizite Synchronisation notwendig im obigen Beispiel
    - Asynchrone SEND/RECV Primitive blockieren nicht
      - Überlappung von Berechnung und Kommunikation möglich

11.4 Parallele Rechnerstrukturen

Shared-Memory-Programmiermodell: Primitive

Name	Syntax	Funktion
`CREATE`	`CREATE(p,proc,args)`	Generiere Prozess, der die Ausführung bei der Prozedur **proc** mit den Argumenten **args** startet
`G_MALLOC`	`G_MALLOC(size)`	Allokation eines gemeinsamen Datenbereichs der Größe **size** Bytes
`LOCK`	`LOCK(name)`	Fordere wechselseitigen exklusiven Zugriff an
`UNLOCK`	`UNLOCK(name)`	Freigeben des Locks

11.4 Parallele Rechnerstrukturen

Shared-Memory-Programmiermodell: Primitive

Name	Syntax	Funktion
`BARRIER`	`BARRIER(name,number)`	Globale Synchronisation für `number` Prozesse
`WAIT_FOR_END`	`WAIT_FOR_END(number)`	Warten, bis `number` Prozesse terminieren
`WAIT_FOR_FLAG`	`while (!flag); or WAIT(flag)`	Warte auf gesetztes flag; entweder wiederholte Abfrage (spin) oder blockiere;
`SET FLAG`	`flag=1; or SIGNAL(flag)`	Setze flag; weckt Prozess auf, der `flag` wiederholt abfragt

11.4 Parallele Rechnerstrukturen

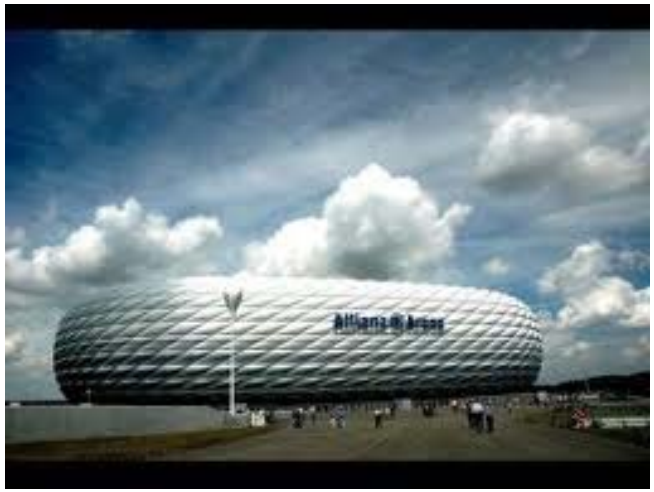
Message Passing: Primitive

Name	Syntax	Funktion
`CREATE`	`CREATE(procedure)`	Erzeuge Prozess, der bei `procedure` startet
`SEND`	`SEND(src_addr,size,dest,tag)`	Sende `size` Bytes von Adresse `src_addr` an `dest` Prozess mit `tag` Identifier
`RECEIVE`	`RECEIVE(buffer_addr, size,src,tag)`	Empfange eine Nachricht mit der Kennung `tag` vom `src`-Prozess und lege `size` Bytes in Puffer bei `buffer_addr` ab
`BARRIER`	`BARRIER(name,number)`	Globale Synchronisation von `number` Prozessen

Anhang
Ausblick

Vorlesung Rechnerstrukturen

Architektur



Vorlesung Rechnerstrukturen

Rechnerarchitektur (Disziplin)

- Allgemeine Strukturlehre mit deren Hilfsmittel
- Ingenieurwissenschaftliche Disziplin, die bestehende und zukünftige Rechenanlagen beschreibt, vergleicht, **beurteilt**, verbessert und **entwirft**.
- Betrachtet den Aufbau und die Eigenschaften des Ganzen (Rechenanlage), seiner Teile (Komponenten) und seiner Verbindungen (Globalstruktur, Infrastruktur)

Rechnerarchitektur (Disziplin)

Entwurf einer Rechenanlage

- Ingenieurmäßige Aufgabe der Kompromissfindung zwischen
  - Zielsetzungen
    - Einsatzgebiet, Anwendungsbereich, Leistung, Verfügbarkeit ...
  - Randbedingungen
    - Technologie, Größe, Geld, Energieverbrauch, Umwelt,...
  - Gestaltungsgrundsätzen
    - Modularität, Sparsamkeit, Fehlertoleranz ...
  - Anforderungen
    - Kompatibilität, Betriebssystemanforderungen, Standards

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Einsatzgebiete, Rechnermärkte
  - Personal Mobile Devices (PMC)

    - Drahtlose Geräte mit Multimedia Schnittestelle
    - Besipiele: Mobiltelephone (Smart phones), Tablets
    - Niedrige Kosten, niedriger Energieverbrauch
    - Hohe Leistung für Multimedia-Anwendungen

  - Desktop Computing

    - PCs bis Workstations
    - Günstiges Preis-/ Leistungsverhältnis
    - Ausgewogene Rechenleistung für ein breites Spektrum von Anwendungen, einschließlich interaktiver Anwendungen (Graphik, Video, Audio) oder WEB-Anwendungen

Rechnerarchitektur (Disziplin)

Zielsetzungen

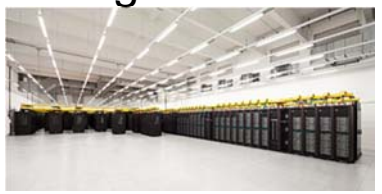
- Einsatzgebiete, Rechnermärkte
  - Server
    - Rechen- und datenintensive Anwendungen, transaktionsorientierte Anwendungen
    - Hohe Anforderungen an die Verfügbarkeit und Zuverlässigkeit, Energieeffizienz
    - Skalierbarkeit
    - Große Datei-Systeme, Ein-/Ausgabesysteme
    - Abgeschlossene Räume, kostenintensiv

Quelle: LRZ: SuperMUC

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Einsatzgebiete, Rechnermärkte
  - Server
    - Höchstleistungsrechner
      - Hohe Rechenleistung bezüglich Gleitkommaverarbeitung
      - Große, kommunikationsintensive Anwendungen
      - Batch-Programme



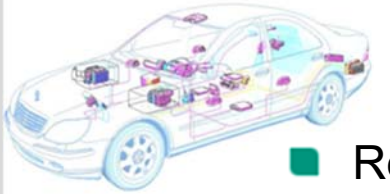
Quelle: LRZ: SuperMUC

    - Server im kommerziellen Bereich, Warehouse-Scale Computers
      - Mainframes, Clusters
      - Durchsatzorientierte Anwendungen
      - Software as a Service (SaaS)

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Einsatzgebiete
  - Eingebettete Systeme (Embedded Systems)
    - Mikroprozessorsysteme, eingebettet in Geräten, daher nicht unbedingt sichtbar

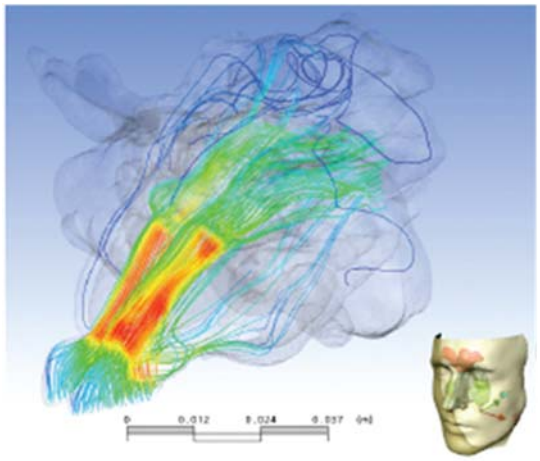


      - Beispiele: Automobil, Unterhaltungselektronik, Telekommunikation, Haushaltsgeräte, …
    - Rechensysteme sind auf spezielle Aufgabe zugeschnitten
      - Hohe Leistungsfähigkeit für spezielle Anwendung
      - Spezialprozessoren, Prozessorkerne mit anwendungsspezifischen Komponenten
    - Breites Preis-/Leistungsspektrum
      - Von einfachen 8-, 16-Bit Microcontrollern bis hin zu komplexen Spezialprozessoren
    - Echtzeitanforderungen
    - Abwägen der Anforderungen an die Rechenleistung, Speicherbedarf, Kosten, Energieverbrauch

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Anwendungsbereiche:
  - Technisch-wissenschaftlichen Bereich:
    - Hohe Anforderungen an die Rechenleistung, insbesondere Gleitkommaverarbeitung
    - Beispiele:
      - Rechnergestützte Simulation
      - Strömungsmechanik
      - Modellierung der globalen klimatischen Veränderungen
      - Struktur von Materialien
      - …
      - Medizintechnik



Rechenzeit auf einer HP XC 4000: (15 TFLOPS): ~4 Tage

Rechenzeit auf einem Rechner im PFLOPS-Bereich: <40 min

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Anwendungsbereiche:
  - Kommerzieller Bereich
    - Datenbankanwendungen
    - WEB, Suchmaschinen
    - Optimierung von Geschäftsprozessen, Unterstützung von Geschäftsentscheidungen (Risikoanalyse)
    - Soziale-Netzwerke, Video-Sharing, On-line-Shopping
  - Eingebettete Systeme
    - Verarbeitung digitaler Medien
    - Automatisierungstechnik
    - Automobil
    - Telekommunikation
    - Medizintechnik
    - …

Rechnerarchitektur (Disziplin)

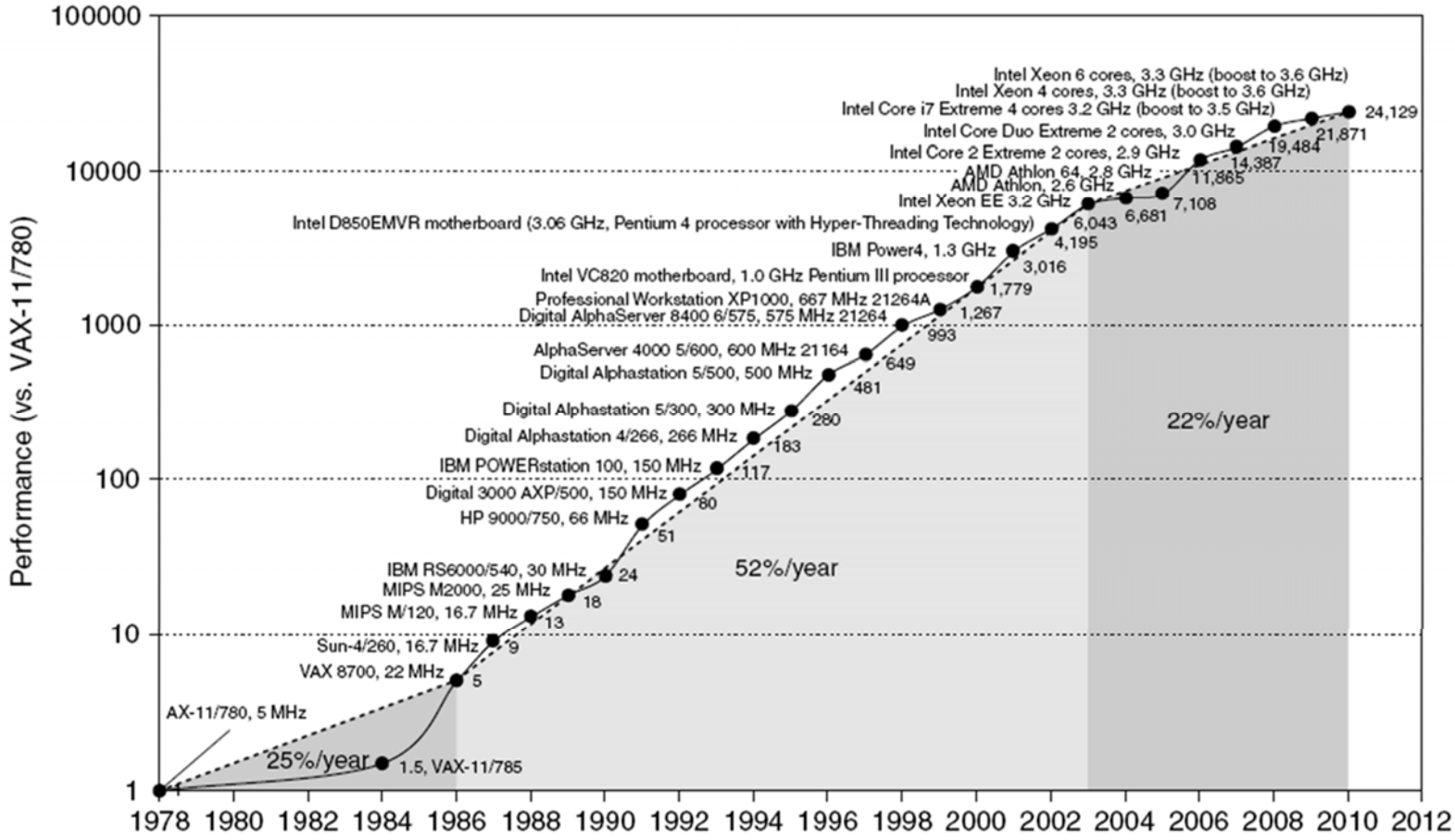
Zielsetzungen

- Rechenleistung
  - Bandbreite, Durchsatz
    - Ausgeführte Arbeit in einem gegebenen Zeitintervall
  - Latenz, Antwortzeit
    - Zeit zwischen dem Start und dem Ende eines Ereignisses
  - Bewertung der Leitungsfähigkeit

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Rechenleistung



Quelle: J. Hennessy, D. Patterson: Computer Architecture – A Quatative Approach, Morgan Kaufmann Publishers, 5th Ed., 2012

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Zuverlässigkeit
  - Bei Ausfällen von Komponenten muss ein betriebsfähiger Kern bereit sein
  - Techniken der Fehlertoleranz
    - Verwendung redundanter Komponenten
    - Wichtig für sicherheitskritische Anwendungen
    - Wichtig im kommerziellen Bereich
  - Bewertung mittels stochastischer Verfahren
    - Fehlerwahrscheinlichkeit
    - Überlebenswahrscheinlichkeit
    - Mittlere Lebensdauer
    - Ausfallrate
- Verfügbarkeit
  - Wahrscheinlichkeit, ein System zu einem beliebigen Zeitpunkt fehlerfrei anzutreffen

Rechnerarchitektur (Disziplin)

Zielsetzungen

- Energieverbrauch, Leistungsaufnahme
  - Mobile Geräte
    - verfügbare Energiemenge durch Batterien und Akkumulatoren begrenzt
    - möglichst lange mit vorhandener Energie auskommen
    - möglichst wenig Energie soll in Wärme umgesetzt werden, um eine Überhitzung zu vermeiden
  - Green IT
    - Rechnerhersteller bieten „green HW“ an:
    - niedriger Energieverbrauch
    - ökologische Produktion
    - einfaches Recycling

Vorlesung Rechnerstrukturen

1. Grundlagen
   - Einführung
   - Allgemeine Grundlagen des Entwurfs von Rechenanlagen
   - Formen des Parallelismus und Klassifizierungen von Rechnerarchitekturen
   - Bewertung von Rechensystemen
   - Zuverlässigkeit, Verfügbarkeit und Fehlertoleranz

2. Prozessortechniken
   - Parallelismus auf Maschinenbefehlsebene
   - Multithreading

Hinweis

- Vom Lehrstuhl angebotene Lehrveranstaltungen:
  - Vorlesungen:
    - Rechnerorganisation
    - Digitaltechnik und Entwurfsverfahren

    } Im Turnus mit Prof. Asfour, Prof. Hanebeck, Prof. Henkel, Prof. Tahoori

    - Rechnerstrukturen
    - Mikroprozessoren I,

    } Regelmäßig im Sommersemester

    - Heterogene parallele Rechnerstrukturen

    } Regelmäßig im Wintersemester

  - Praktika:
    - Basispraktikum Technische Informatik: Hardware-naher Systementwurf
    - Projektorientiertes Software-Praktikum (Parallele Numerik)
    - Projektpraktikum Heterogene Parallele Systeme

  - Seminare
    - Ausgewählte Kapitel der Rechnerarchitektur

Forschungsfragen

Challenges

- Parallelism in all major computing classes
  - Desktop, server, supercomputers
  - Embedded Systems (MPSoC)

- How can programmers be guided on their way?
  - Parallelism on all system levels
  - How to express parallelism?
  - How to exploit the resources most benificially?

- Heterogeneity is growing!
  - How to exploit the potential of coprocessor concepts
  - How to exploit specific hardware capabilities
  - How to exploit adaptable and configurable hardware characteristics to applications

- How to hide the complexity from the user

Forschungsfragen

Approaches

- New parallel programming paradigms, parallel programming concepts
  - E.g. Transactional Memory (TM), PGAS
- Methods and tools supporting programmers to write efficient parallel programs
  - Parallel programming environments
  - Sustained observation of the system‘s behaviour
  - Intelligent data analysis
  - Optimization strategies
  - Visualization
- **Adaptation and Virtualization**
  - **Hiding hardware details while efficiently exploiting the resources**
- **Accuracy variation**
  - Approximate computing
  - Exact arithmetics