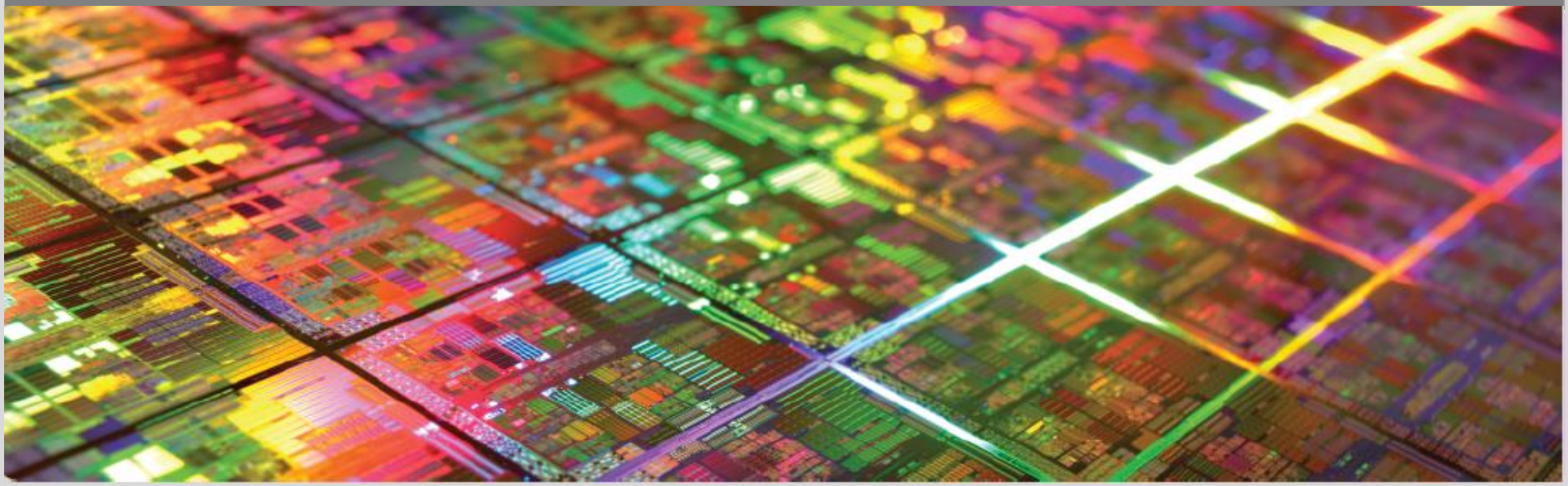
Karlsruher Institut für Technologie

Rechnerstrukturen

Vorlesung im Sommersemester 2014

Prof. Dr. Wolfgang Karl

Fakultät für Informatik – Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



Vorlesung Rechnerstrukturen

Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation

Multiprozessorsysteme

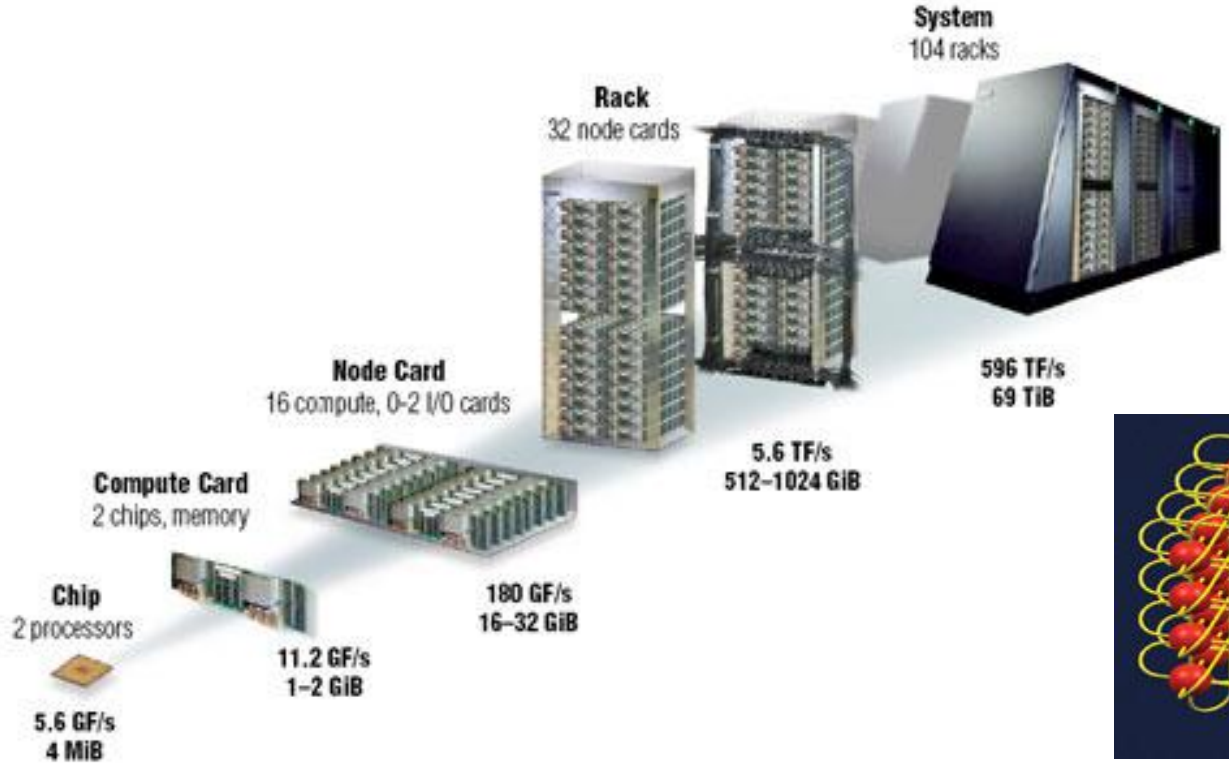
Motivation

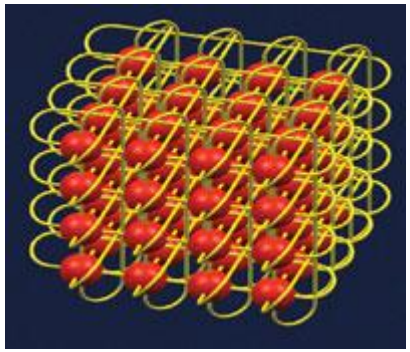
- Allgemeine Grundlagen, parallele Programmierung, Verbindungsstrukturen, Leistungsfähigkeit
- Speichergekoppelte Multiprozessoren: SMP und DSM, Cache-Kohärenz und Speicherkonsistenz, Rechnerbeispiele
- Nachrichtengekoppelte Multiprozessoren, Beispielrechner

Multiprozessorsysteme

Parallelrechner:

- **Beispiel: Multiprozessor mit verteiltem Speicher**





BlueGene/L: Interconnection Network

Quelle: https://asc.llnl.gov/computing_resources/bluegenel/photogallery.html

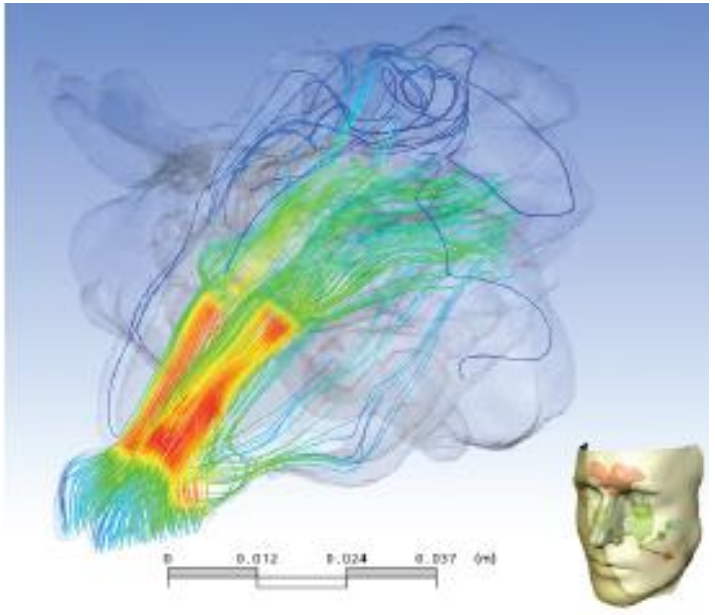
Multiprozessorsysteme

Anwendungsbereiche

- Hohe Anforderungen der Anwendungen an die Rechenleistung
  - Technisch-wissenschaftlicher Bereich
    - 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



Quelle: Persönliche Notiz: V. Heuveline

Vorlesung Rechnerstrukturen

Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen

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

Parallele Architekturen

Rechnerarchitektur

- **Abstraktion**
  - Benutzer-/System-Schnittstelle
  - Hardware-/Software-Schnittstelle
- **Architektur**
  - Spezifiziert die Menge der Operationen an den Schnittstellen und die Datentypen, auf denen diese operieren
- **Organisation**
  - Realisierung der Abstraktionen

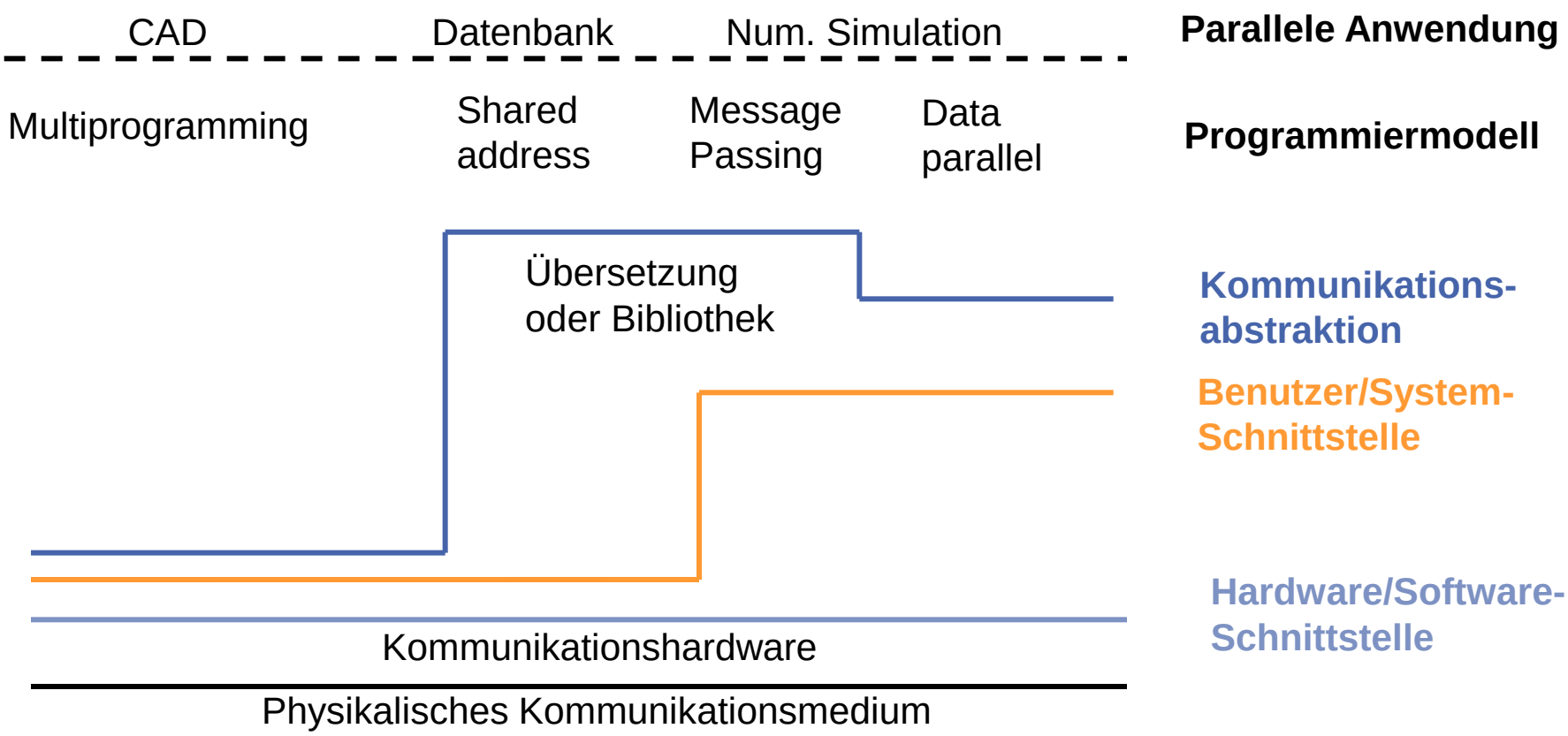
Parallele Architekturen

Kommunikationsarchitektur

- **Abstraktion**
  - Benutzer-/System-Schnittstelle
  - Hardware-/Software-Schnittstelle
- **Architektur**
  - Spezifiziert die Kommunikations- und Synchronisationsoperationen
- **Organisation**
  - Realisierung dieser Operationen

Parallele Architekturen

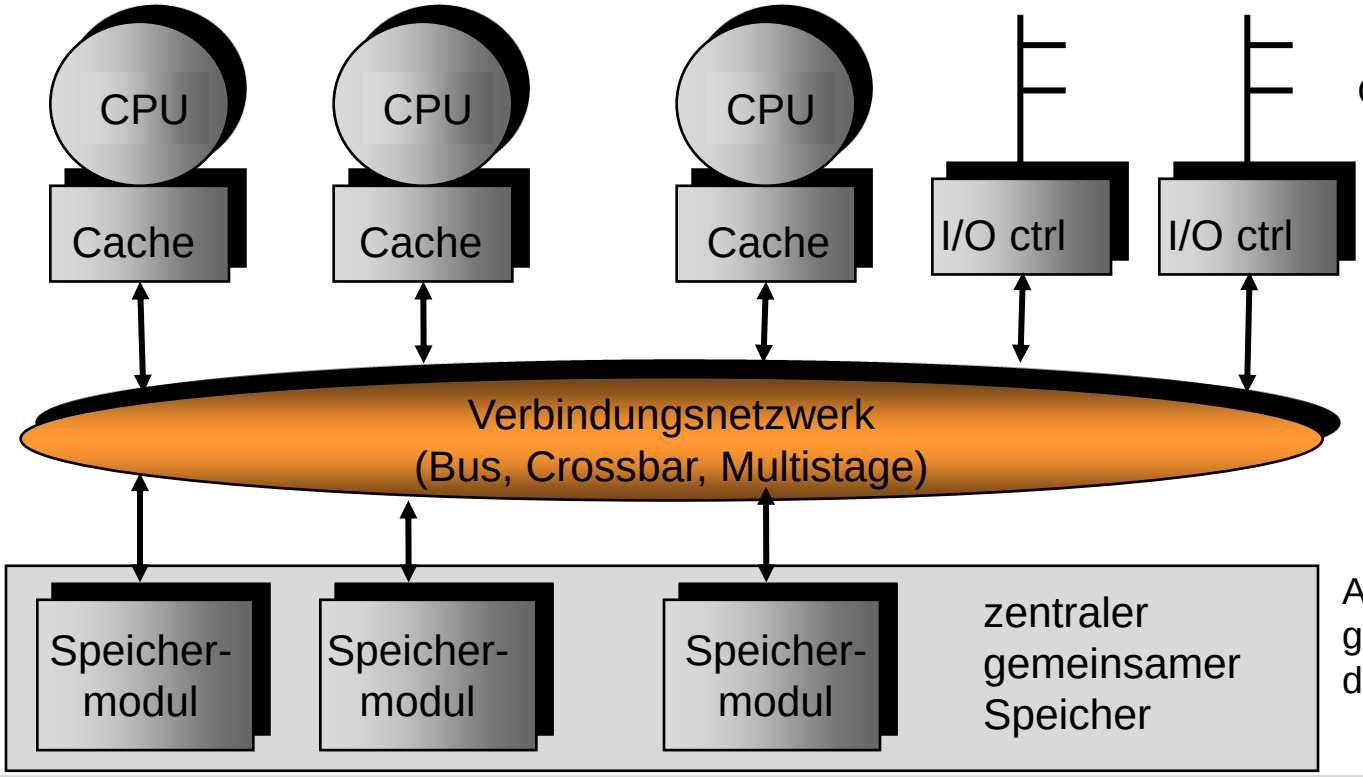
Abstraktion



Parallele Architekturmodelle

Multiprozessor mit gemeinsamem Speicher

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

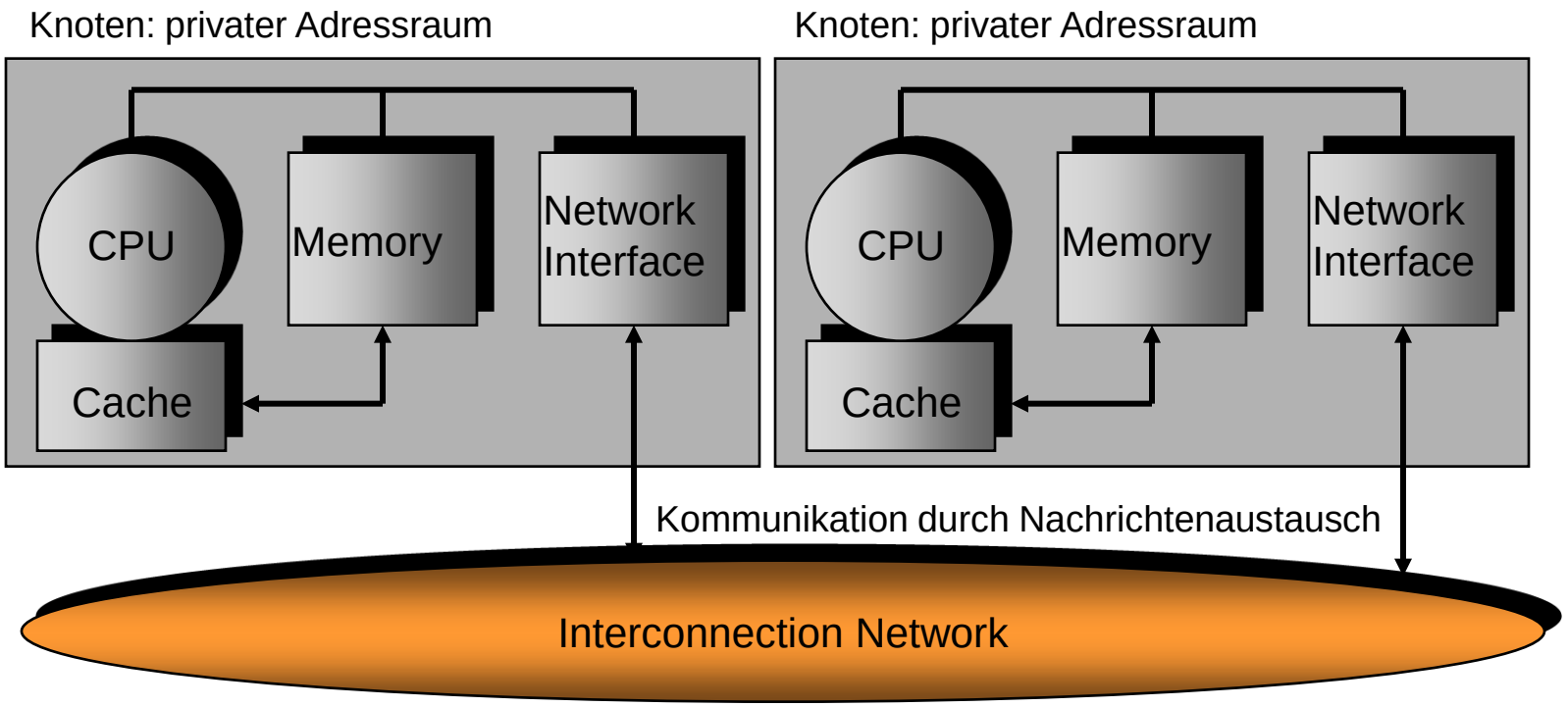


Austausch von Daten über gemeinsamen Speicher durch load /store Op.

Parallele Architekturmodelle

Multiprozessor mit verteiltem Speicher

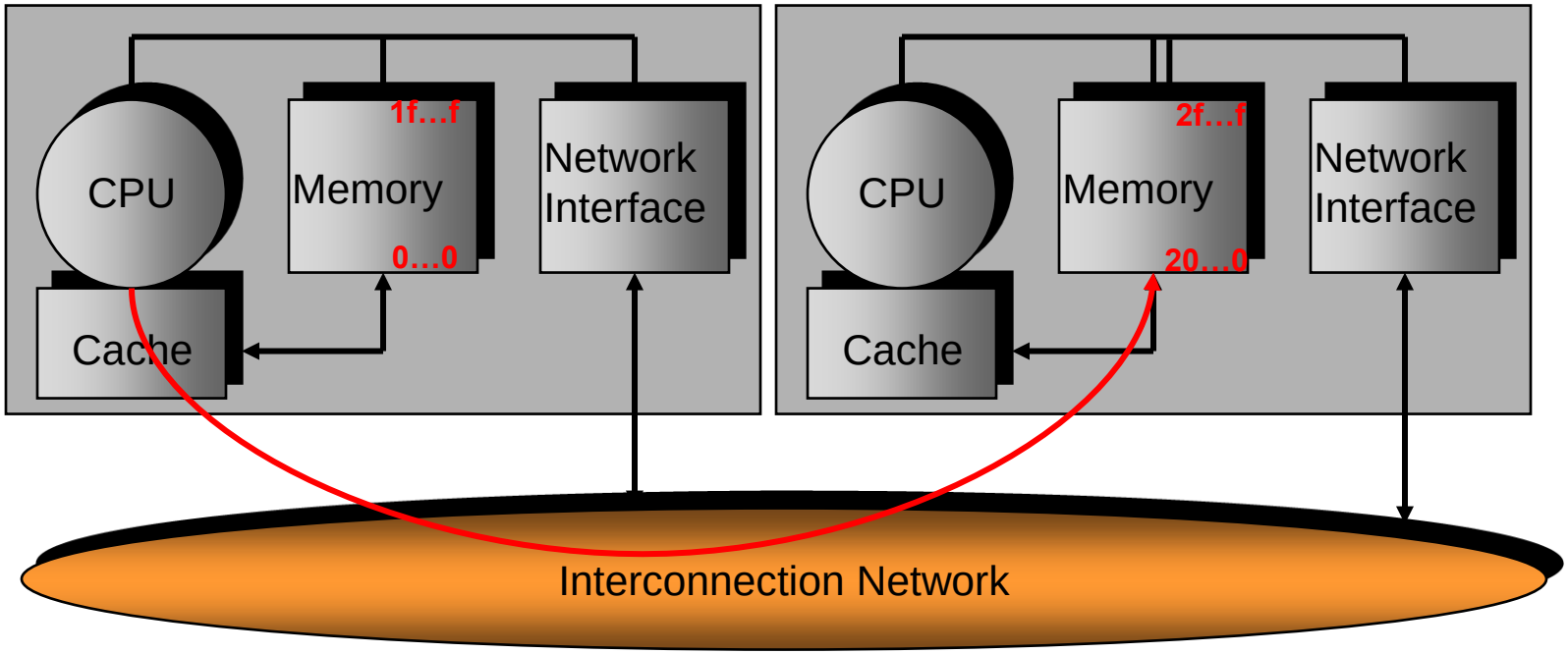
- **NORMA** No Remote Memory Access
- Beispiel: **Cluster**



Parallele Architekturmodelle

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



Parallele Programmiermodelle

Programmiermodell

- 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

Parallele Programmiermodelle

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

Parallele Programmiermodelle

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

Parallele Programmiermodelle

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

Parallele Programmiermodelle

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

Parallele Programmiermodelle

Parallele Programmierung

- Beispiel: Pseudocode für einen sequentiellen Algorithmus zur Addition 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

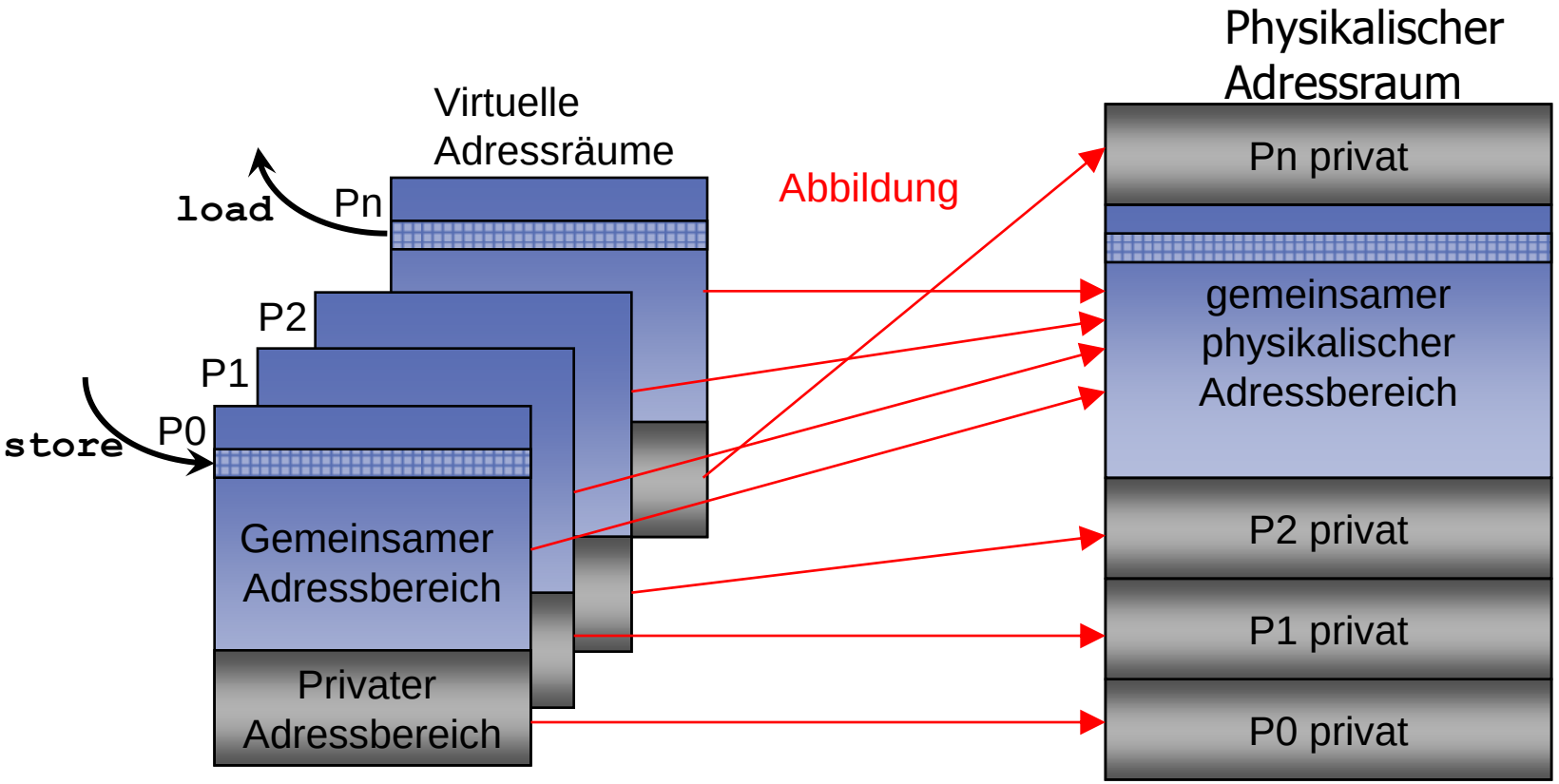
- Datenparallelismus: die Berechnung eines Matrixelements ist unabhängig von der Berechnung der anderen Elemente
- Grundsätzlich können alle Elemente parallel berechnet werden, aber es muss der Aufwand für das Aufsetzen der Threads mit dem Gewinn an Rechenleistung abgewogen werden

Parallele Programmiermodelle

- **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

Parallele Programmiermodelle

- **Gemeinsamer Speicher (Shared Memory)**



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

Parallele Programmierung: Shared Memory

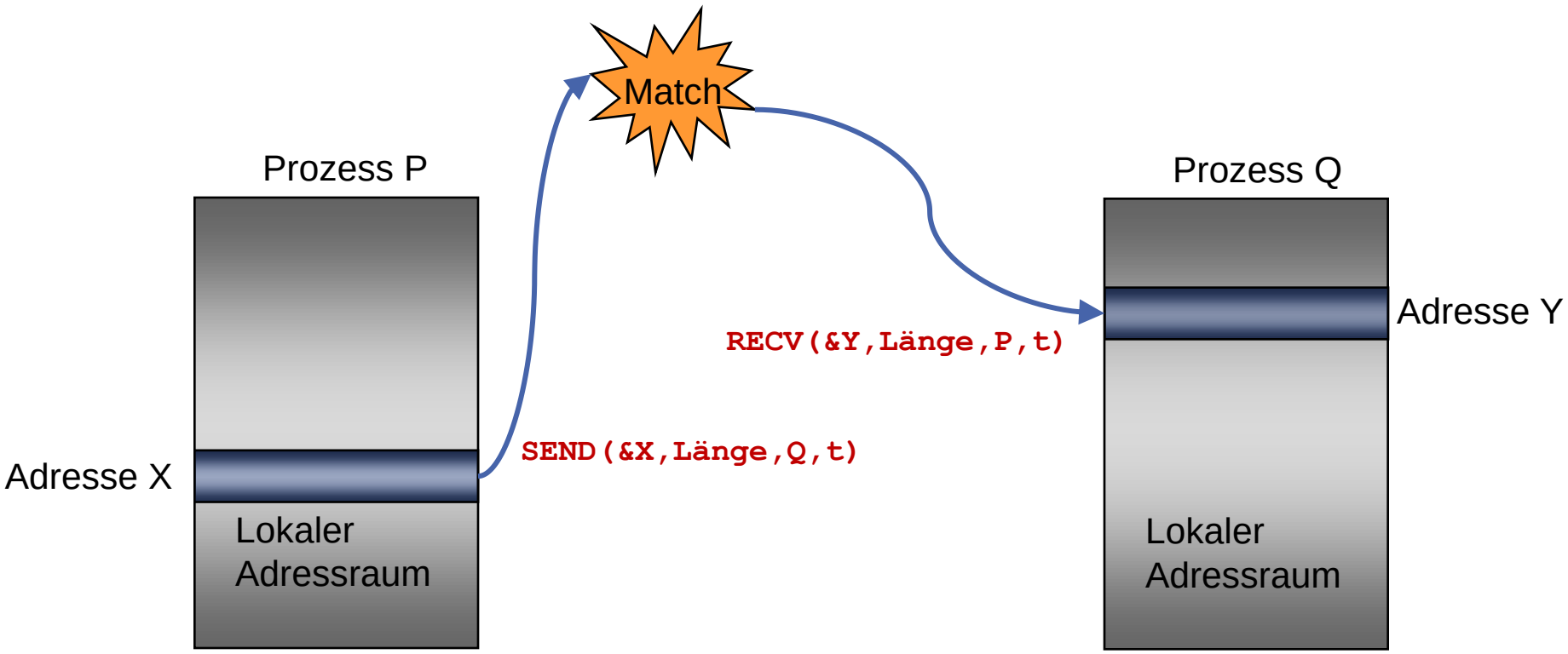
- Beispiel: Pseudocode für einen parallelen Algorithmus (Anmerkungen):
  - Die Arbeit wird auf *nproc* Threads aufgeteilt, wobei $\mathrm{nproc} = \frac{\mathrm{N}}{2^{\mathrm{i}}}$ $\text{für}\ \mathrm{i} = 0,1,\ldots,\log_2 \mathrm{N}$ ist
  - Der erste Thread berechnet die Elemente für die ersten 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

Parallele Programmiermodelle

- **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

Parallele Programmiermodelle

- **Nachrichtenorientiertes Programmiermodell (Message Passing)**



Parallele Programmiermodelle

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 Programmiermodelle

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

Parallele Programmiermodelle

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*

Parallele Programmiermodelle

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

Parallele Programmiermodelle

- **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

Parallele Programmiermodelle

- **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

Parallele Programmiermodelle

- **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

Vorlesung Rechnerstrukturen

Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen
- 3.3 Parallele Programmierung
- 3.4 Quantitative Maßzahlen

Quantitative Maßzahlen

Parallelitätsprofil

- misst die entstehende Parallelität in einem parallelen Programm bzw. bei der Ausführung auf einem Parallelrechner.
- Gibt eine Vorstellung von der inhärenten Parallelität eines Algorithmus/Programms und deren Nutzung auf einem realen oder ideellen Parallelrechner

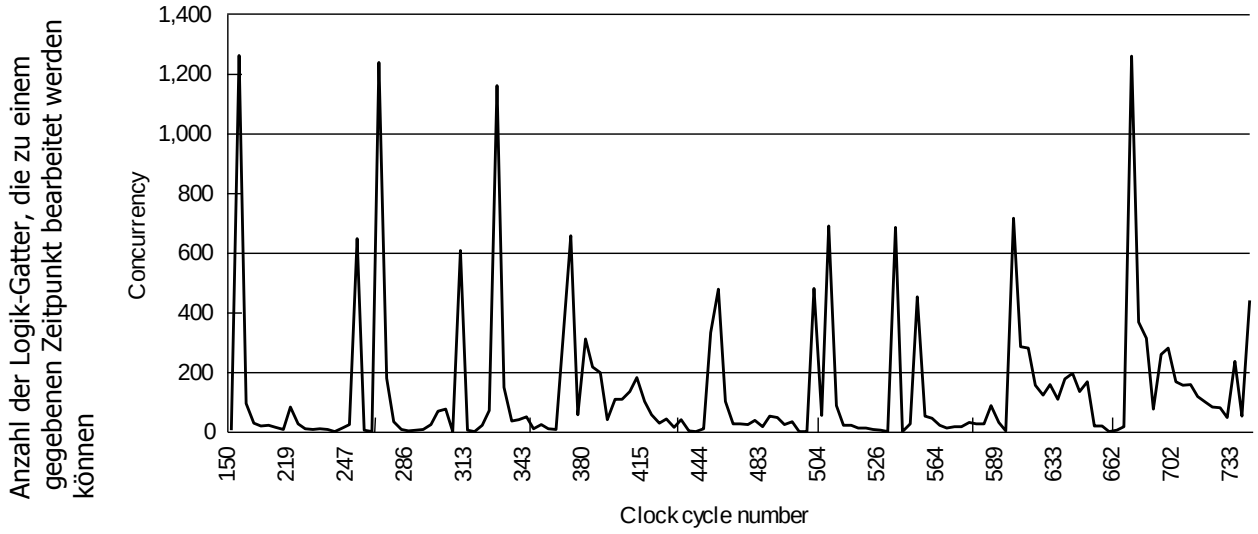
- Grafische Darstellung:
  - Auf der x-Achse wird die Zeit und auf der y-Achse die Anzahl paralleler Aktivitäten angetragen.
  - ➔ Perioden von Berechnungs- Kommunikations- und Untätigkeitszeiten sind erkennbar.

Quantitative Maßzahlen

Parallelitätsprofil

- Zeigt an, wie viele Tasks einer Anwendung zu einem Zeitpunkt parallel ausgeführt werden können
- Parallelitätsgrad PG(t):
  - Anzahl der Tasks, die zu einem Zeitpunkt parallel bearbeitet werden können

Beispiel: Parallele ereignisgesteuerte Simulation der Logik-Synthese



Quelle: D. Culler: Parallel Computer Architecture. Morgan Kaufmann Publishers, 1999, p.87

Quantitative Maßzahlen

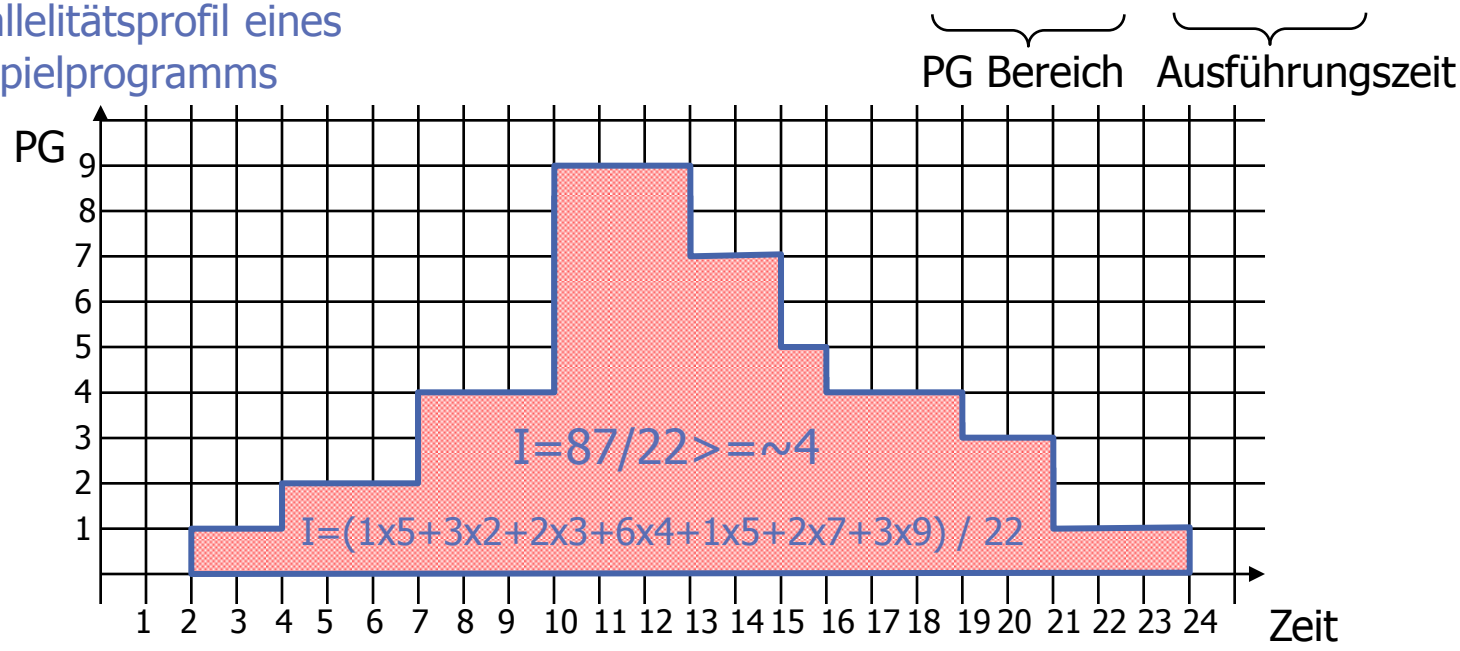
Parallelitätsprofil

- **Parallelindex I** (Mittlerer Grad des Parallelismus):

$$I = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} PG(t)dt$$

$$I = \left(\sum_{i=1}^{m} i * ti \right) / \left(\sum_{i=1}^{m} t_i \right)$$

Parallelitätsprofil eines Beispielprogramms



Quantitative Maßzahlen

Vergleich von Multiprozessorsystemen zu Einprozessorsystemen

- Leistungsangaben zu Multiprozessorsystemen werden mit Leistungsangaben zu Einprozessorsystemen in Beziehung gesetzt
- Notwendig:
  - Programm das auf beiden zu vergleichenden Systemen ablaufen kann

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Definitionen:**
  - P(1): Anzahl der auszuführenden (Einheits-) Operationen (Tasks) des Programms auf einem Einprozessorsystem.
  - P(n): Anzahl der auszuführenden (Einheits-) Operationen (Tasks) des Programms auf einem Multiprozessorsystem mit n Prozessoren.
  - T(1): Ausführungszeit auf einem Einprozessorsystem in Schritten (oder Takten).
  - T(n): Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten (oder Takten).
- **Vereinfachende Voraussetzungen:**
  - T(1) = P(1),
    - da in einem Einprozessorsystem (Annahme: einfacher Prozessor) jede (Einheits-) Operation in genau einem Schritt ausgeführt werden kann.
  - $T(n) \leq P(n)$,
    - da in einem Multiprozessorsystem mit n Prozessoren ($n \geq 2$) in einem Schritt mehr als eine (Einheits-)Operation ausgeführt werden kann.

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung S(n) (Speedup):**

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

  - Gibt die Verbesserung in der Verarbeitungsgeschwindigkeit an
  - Wert bezieht sich auf das jeweils bearbeitete Programm oder kann als Mittelwert eine Menge von Programmen angesehen werden
  - Üblicherweise gilt:

$$1 \leq S(n) \leq n$$

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Effizienz E(n)**

$$E(n) = \frac{S(n)}{n}$$

  - Gibt die relative Verbesserung in der Verarbeitungsgeschwindigkeit an
  - Leistungssteigerung wird mit der Anzahl der Prozessoren n normiert

  - Üblicherweise gilt:

$$\frac{1}{n} \leq E(n) \leq 1$$

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung (Speed-Up), Effizienz:**
  - Algorithmenunabhängige Definition
  - Man setzt den besten bekannten sequentiellen Algorithmus für das Einprozessorsystem in Beziehung zum vergleichbaren parallelen Algorithmus für das Multiprozessorsystem.
    - Absolute Beschleunigung
    - Absolute Effizienz

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung (Speed-Up), Effizienz:**
  - Algorithmenabhängige Definition
  - Man benutzt den parallelen Algorithmus so, als sei er sequentiell, und misst dessen Laufzeit auf einem Einprozessorsystem.
  - Der für die Parallelisierung erforderliche Zusatzaufwand an Kommunikation und Synchronisation kommt „ungerechterweise“ auch für den sequentiellen Algorithmus zum Tragen.
    - Relative Beschleunigung
    - Relative Effizienz

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Mehraufwand R(n) für die Parallelisierung:**

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

  - Beschreibt den bei einem Multiprozessorsystem erforderlichen Mehraufwand für die Organisation, Synchronisation und Kommunikation der Prozessoren
  - Es gilt:

$$1 \leq R(n)$$

    - Anzahl der auszuführenden Operationen eines parallelen Programms größer ist als diejenige des vergleichbaren sequentiellen Programms

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Auslastung U(n):**

$$U(n) = \frac{I(n)}{n} = R(n) \times E(n) = \frac{P(n)}{n \times T(n)}$$

  - Entspricht dem normierten Parallelindex
  - Gibt an, wie viele Operationen (Tasks) jeder Prozessor im Durchschnitt pro Zeiteinheit ausgeführt hat

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Folgerungen
  - Alle definierten Ausdrücke haben für n = 1 den Wert 1.
  - Der Parallelindex gibt eine obere Schranke für die Leistungssteigerung:

$$1 \leq S(n) \leq I(n) \leq n$$

  - Die Auslastung ist eine obere Schranke für die Effizienz:

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

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Zahlenbeispiel:
  - Ein Einprozessorsystem benötige für die Ausführung von 1000 Operationen 1000 Schritte.
  - Ein Multiprozessorsystem mit 4 Prozessoren benötige dafür 1200 Operationen, die aber in 400 Schritten ausgeführt werden können.
  - Damit gilt:

$$P(1) = T(1) = 1000, P(4) = 1200, T(4) = 400$$

  - Daraus ergibt sich:

$$S(4) = 2{,}5 \text{ und } E(4) = 0{,}625$$

  - Die Leistungssteigerung verteilt sich als im Mittel zu 62,5% auf alle Prozessoren

Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Zahlenbeispiel:
  - Parallelindex und Auslastung:

$$I(4) = 3 \text{ und } U(4) = 0{,}75$$

  - Es sind im Mittel drei Prozessoren gleichzeitig tätig, d.h., jeder Prozessor ist nur zu 75% der Zeit aktiv.
  - Mehraufwand:

$$R(4) = 1{,}2$$

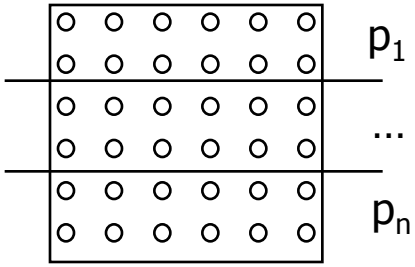
  - Bei Ausführung auf dem Multiprozessorsystem sind 20% mehr Operationen als bei Ausführung auf einem Einprozessorsystem notwendig.

Quantitative Maßzahlen

- **Skalierbarkeit eines Parallelrechners**
  - das Hinzufügen von weiteren Verarbeitungselementen führt zu einer kürzeren Gesamtausführungszeit, ohne dass das Programm geändert werden muss.
  - Insbesondere meint man damit eine lineare Steigerung der Beschleunigung mit einer Effizienz nahe bei Eins.
  - Wichtig für die Skalierbarkeit ist eine angemessene Problemgröße.
  - Bei fester Problemgröße und steigender Prozessorzahl wird ab einer bestimmten Prozessorzahl eine Sättigung eintreten. Die Skalierbarkeit ist in jedem Fall beschränkt.
  - Skaliert man mit der Anzahl der Prozessoren auch die Problemgröße (scaled problem analysis), so tritt dieser Effekt bei gut skalierenden Hardware- oder Software-Systemen nicht auf.

Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Gegeben: ein zweidimensionales *k* x *k*-Gitter
  - 1. Berechnungsphase: Ausführung einer Operation auf allen Gitterpunkten
    - Annahme: keine Abhängigkeiten zwischen den Gitterpunkten
    - Parallele Berechnung auf n Prozessoren



  - 2. Berechnungsphase: Berechnung der Summe der k^2 berechneten Werte der Gittersumme
    - Jeder Prozessor addiert seine k^2/n berechneten Werte zur globalen Summe

Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Problem:
    - Akkumulation der globalen Summe muss serialisiert werden!
    - 2. Phase benötigt k^2 Zeiteinheiten unabhängig von *n*
    - Ausführungszeit des parallelen Programms: $k^2/n + k^2$
    - Ausführungszeit des sequentiellen Programms: $2k^2$
    - Möglicher Speedup S:

$$\frac{2k^2}{\frac{k^2}{n} + k^2} = \frac{2n}{n+1}$$

    - Selbst bei einer hohen Anzahl Prozessoren nicht mehr als 2.

Quantitative Maßzahlen

- **Gesetz von Amdahl**
  - Gesamtausführungszeit T(n)

$$T(n) = T(1) \times \frac{1-a}{n} + T(1) \times a$$

Ausführungszeit des parallel ausführbaren Programmteils 1-a

Ausführungszeit des sequentiell ausführbaren Programmteils a

a: Anteil des Programmteils, der nur sequentiell ausgeführt werden kann

  - Beschleunigung

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1) \times \frac{1-a}{n} + T(1) \times a} = \frac{1}{\frac{1-a}{n} + a}$$

$$\text{Für } n \to \infty: \; S(n) = \frac{1}{a}$$

Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Erhöhung der Parallelität
    - Aufteilung der 2. Berechnungsphase in zwei weiteren Teilphasen:
    - 1. Teilphase: Jeder Prozessor berechnet die Summe seiner berechneten Werte
      - Kann vollständig parallel abgearbeitet werden
    - 2. Teilphase: Akkumulation der Teilsummen
      - Weiterhin seriell!
    - Ausführungszeit $T(n)=k^2/n+k^2/n+n$
    - Beschleunigung $S(n)=n*2k^2/(2k^2+n^2)$
      - Wenn n groß genug, dann nahezu linear!

Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Diskussion
  - Amdahls Gesetz zufolge kann eine kleine Anzahl von sequentiellen Operationen die mit einem Parallelrechner erreichbare Beschleunigung signifikant begrenzen.
  - Beispiel: *a = 1/10* des parallelen Programms kann nur sequenziell ausgeführt werden,
    ➔ das gesamte Programm kann maximal zehnmal schneller als ein vergleichbares, rein sequenzielles Programm sein.
  - Jedoch: viele parallele Programme haben einen sehr geringen sequenziellen Anteil (*a«1*)

Quantitative Maßzahlen

- **Synergetischer Effekt und superlinearer Speedup**
- Theorie : einen „**superlinearen Speedup**“ kann es nicht geben:
  - Jeder parallele Algorithmus lässt sich auf einem Einprozessorsystem simulieren, indem in einer Schleife jeweils der nächste Schritt jedes Prozessors der parallelen Maschine emuliert wird.

- Ein „**superlinearer Speed-up**“ kann real beobachtet werden bei
  - parallelem Backtracking (depth-first search)
  - Beim Programmlauf auf einem Rechner passen die Daten nicht in den Hauptspeicher des Rechners (häufiger Seitenwechsel), aber:
    bei Verteilung auf die Knoten des Multiprozessors können die parallelen Programme vollständig in den Cache- und Hauptspeichern der einzelnen Knoten ablaufen.

Quantitative Maßzahlen

- **Weitere grundsätzliche Probleme bei Multiprozessoren**
  - Verwaltungsaufwand (Overhead)
    - Steigt mit der Zahl der zu verwaltenden Prozessoren
  - Möglichkeit von Systemverklemmungen (deadlocks)
  - Möglichkeit von Sättigungserscheinungen
    - können durch Systemengpässe (bottlenecks) verursacht werden.