

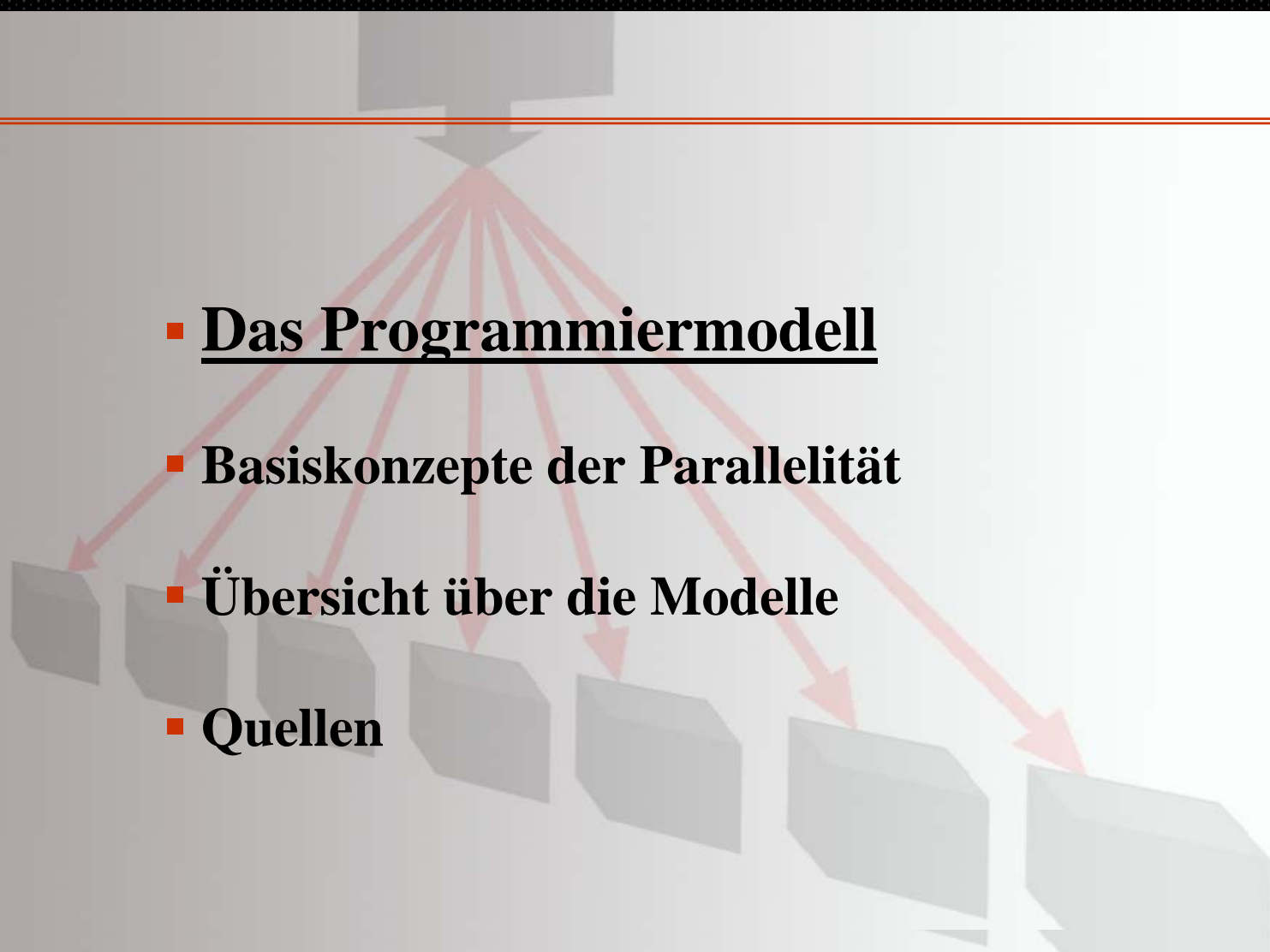


Parallele Modelle

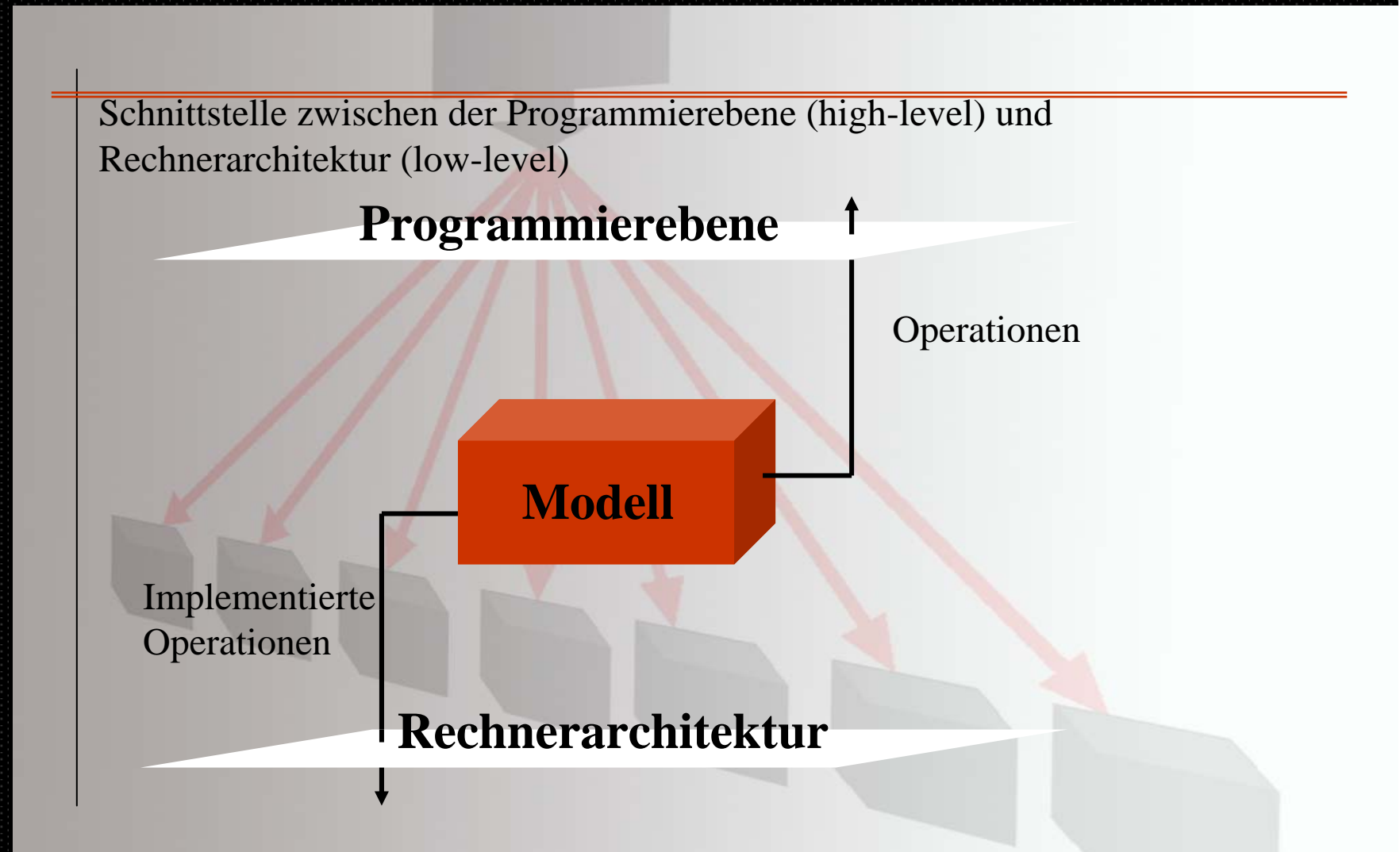
Einführung und Überblick

Universität Marburg
Studienfach Informatik
Spaska Forteva

Überblick

- 
- **Das Programmiermodell**
 - **Basiskonzepte der Parallelität**
 - **Übersicht über die Modelle**
 - **Quellen**

Das Programmiermodell (1)



Das Programmiermodell-Anforderungen (2)

- **Leicht zu programmieren**

Dem Programmierer sollten die meisten Details verborgen werden. Möglichst viel der exakten Struktur des Programms sollte durch den Compiler eingebracht werden.

- **Gutes semantisches Fundament**

Es muss ein gutes semantisches Fundament zur Verfügung sein, damit die Transformationstechniken des Compilers korrekt angewendet werden.

Das Programmiermodell-Anforderungen (3)

- **Architektur-unabhängig**

Die Programme sollten von parallelem Computer zu parallelem Computer portierbar sein, ohne in einer nicht trivialen Weise neu entwickelt werden zu müssen, oder geändert zu werden.

- **Leicht erlernbar**

Ein Modell soll verständlich sein, und die Struktur soll leicht von anderen Menschen akzeptiert werden, damit es möglich ist, vorhandene Software zu studieren und eventuell weiter zu entwickeln.

Das Programmiermodell-Anforderungen (4)

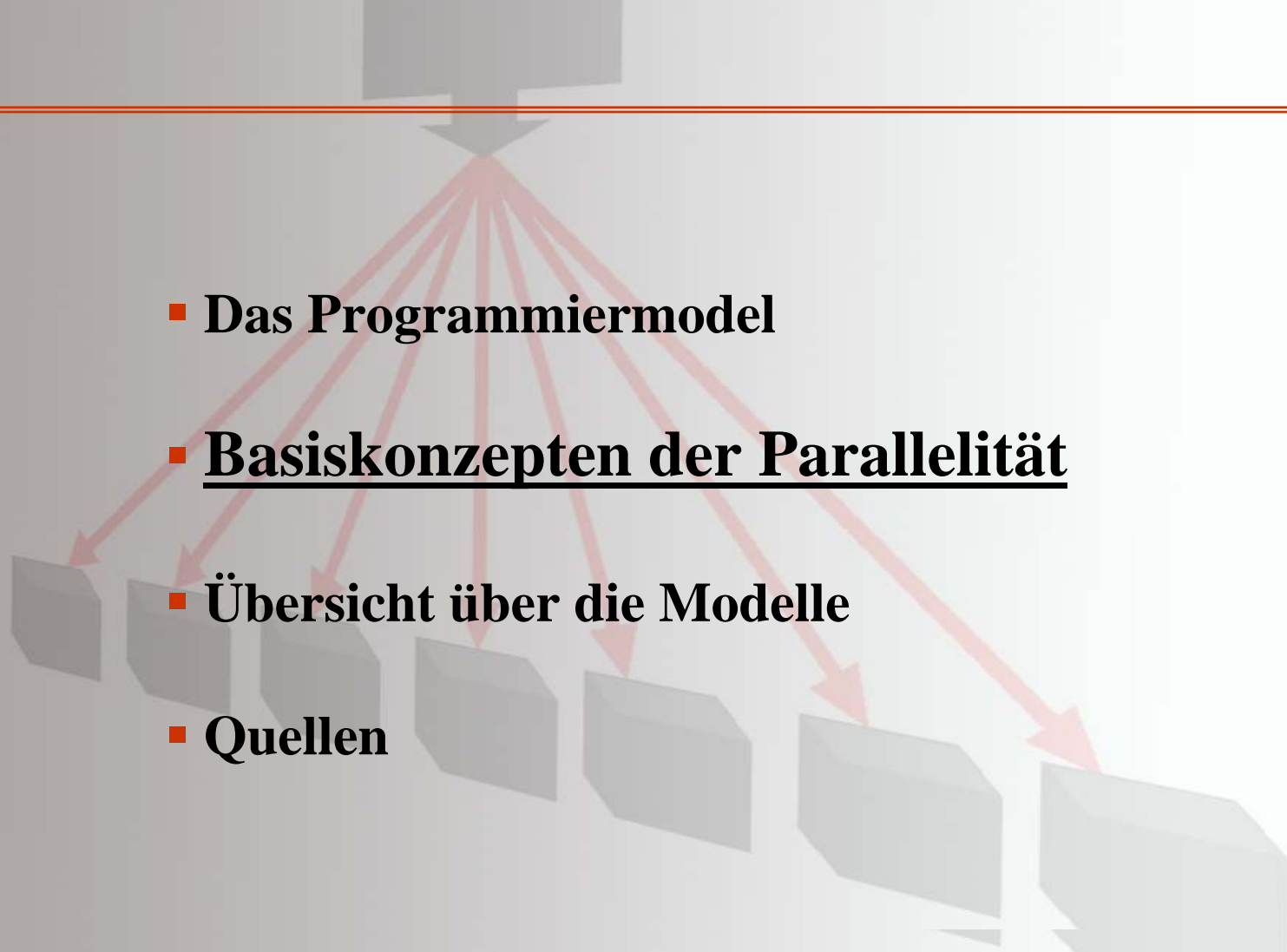
■ Garantierte Performance

Ein Modell sollte eine gute Leistung über Vielzahl von verschiedenen parallelen Architekturen garantieren.

■ Geringe Kosten

Unter den Kosten eines Programms ist zum einen die **Ausführungszeit** des Programms gemeint, zum anderen die **Kosten der Entwicklung**. Die Kostenfrage ist bei den parallelen Software nur schwer einzuschätzen, da schon kleine Änderungen im Programmtext und die Wahl des Rechners sich maßgeblich auf die Kosten eines Programms auswirken.

Überblick

- 
- **Das Programmiermodell**
 - **Basiskonzepten der Parallelität**
 - **Übersicht über die Modelle**
 - **Quellen**

Basiskonzepten der Parallelität (1)

- **Die drei wichtigen Komponente des parallelen Rechners sind:**

- **Prozessoren**

- **Systemkommunikationsnetz**

- Verbindet die Prozessoren und die Speichermodule.

- **Speichereinheit**

- Die Speichereinheiten sind von der Architektur des Rechners abhängig.

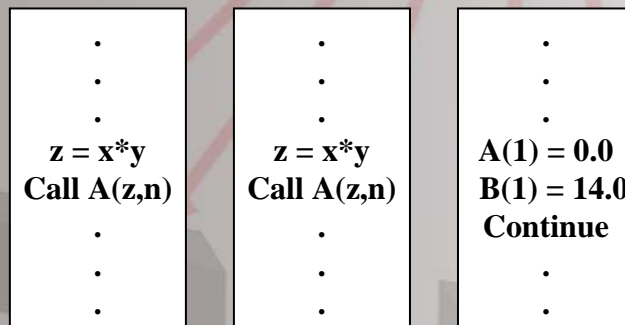
- In dem Seminar werden MIMD-Architekturen betrachtet.

Basiskonzepten der Parallelität (2)

■ Architekturen

■ MIMD(multiple instruction multiple data)

MIMD Models



Prozessor 3

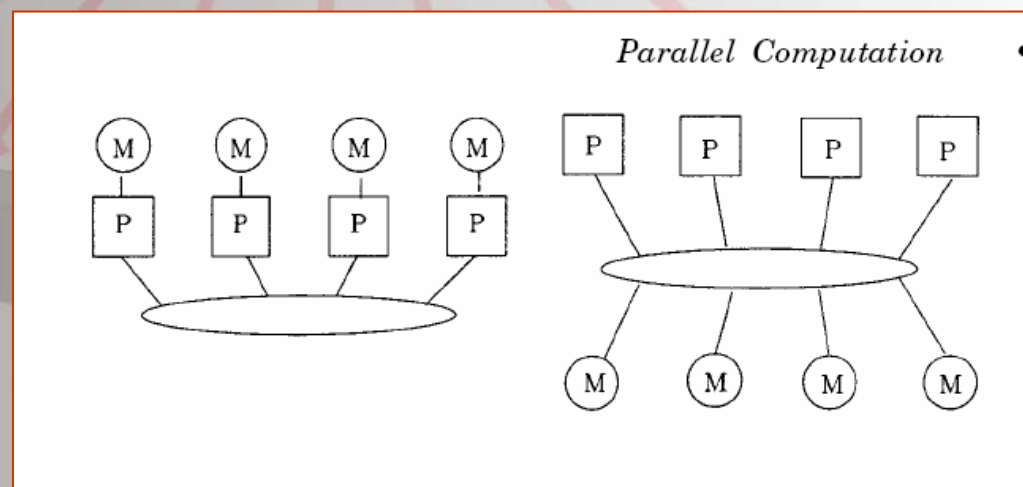
Prozessor 3

Prozessor 3

- Eigener lokaler Speicher (Programmspeicher)
- jeder Prozessor lädt separate Instruktion aus dem Datenspeicher
- Zugriff auf (gemeinsamen oder verteilten) Datenspeicher

Basiskonzepten der Parallelität (3)

- **Memory-Architektur von MIMD**
 - **Distributed-memory (lokaler Speicher)**
 - **Shared-memory (gemeinsamen Speicher)**



Basiskonzepte der Parallelität (4)

■ Kommunikation

■ Message Passing (Distributed-Memory)

Sender und Empfänger treten als Paar auf. Eine Sendeoperation schickt Daten aus dem lokalen Adressraum von P1 zum in der Operation angegebenen P2.

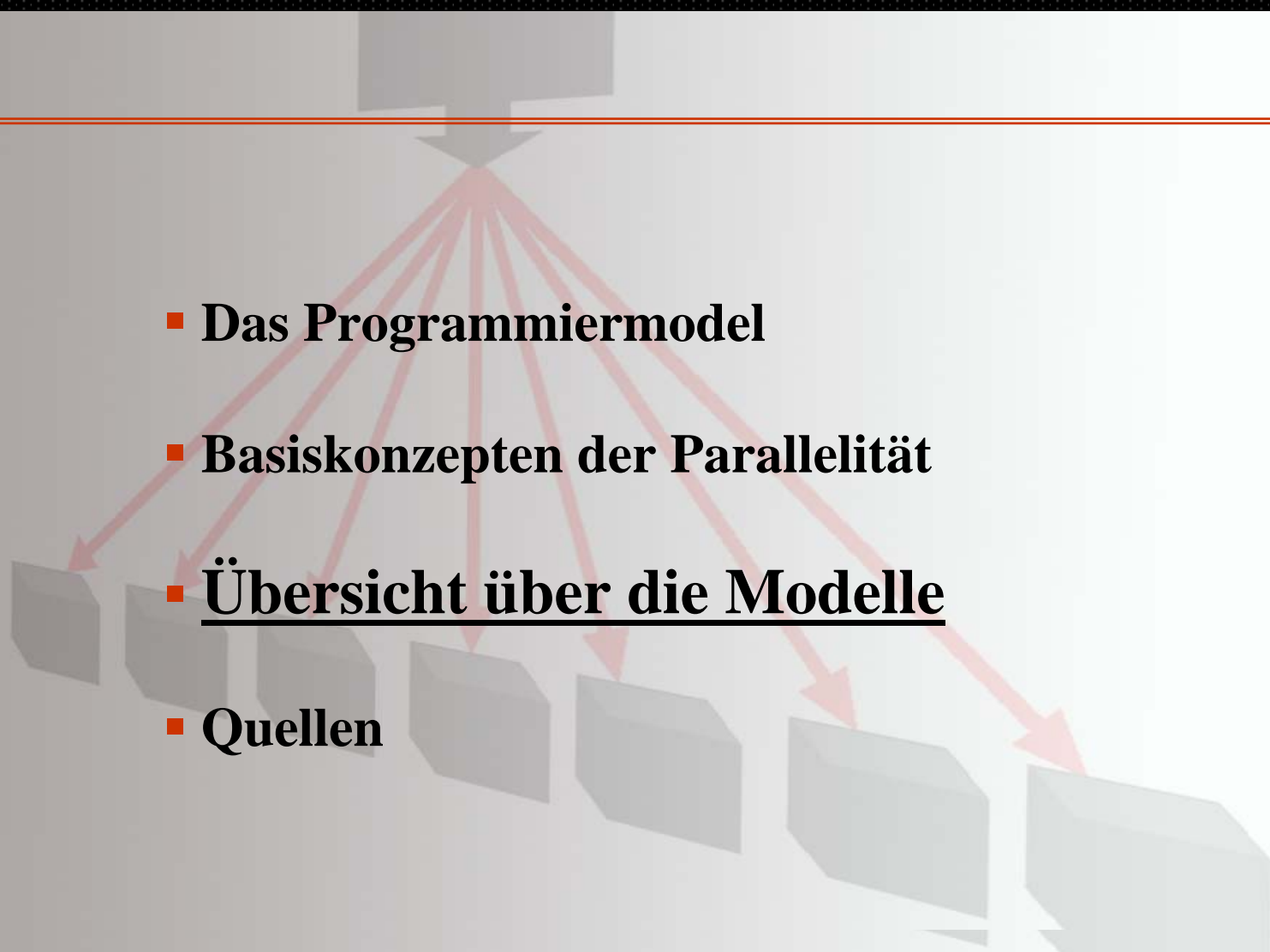
■ Transfer durch Shared Memory

Gemeinsamer Zugriff auf den Datenpool

■ Direct Remote Memory Access

Ein Verarbeitungselement besteht aus zwei Prozessoren:
Berechnungsprozessor, Kommunikationsaustauschprozessor

Überblick

- 
- **Das Programmiermodell**
 - **Basiskonzepten der Parallelität**
 - **Übersicht über die Modelle**
 - **Quellen**

Übersicht über die Modelle

- **Verteilung der Modelle, abhängig von dem Niveaus der Abstraktion (Baum-Repräsentation)**

Stufen der Parallelisierung

- **Dekomposition** Zerlegung des Programms in Stücke (Threads)
- **Mapping** Zuteilung der Threads auf die Prozessoren
- **Kommunikation** zwischen den Threads (Datenaustausch)
- **Synchronisation**: Möglich wenig Potenzial für Deadlocks

Übersicht über die Modelle(1) Nichts Explizit

■ Abstraktionsgrad 1

Programmierer müssen sich nicht um die parallele Abarbeitung kümmern!

Diese Modelle sind einfach zu verstehen!

■ Dynamische Strukturen (Funktionale Programmiersprachen)

Vorteile

■ Graphreduktionstechniken (Funktionen als Bäumen)

■ kein Seiteneffekt -> parallele Auswertung der Argumente der Funktionen.

Z.B. $(exp1 * exp2)$ wo $exp1$ und $exp2$ beliebige Ausdrücke sind, können zwei Prozessoren unabhängig $exp1$ und $exp2$ auswerten.

#parallel mögliche Transformationen = #Prozessoren

Annahme: beliebig viele Prozessoren

Übersicht über die Modelle(2) Nichts Explizit

Beispiel (gut geeignet für parallele Bearbeitung)

Summation von n Zahlen in einem balancierten binären Baum

$\text{sumBt}(\text{Blatt } x) = x$

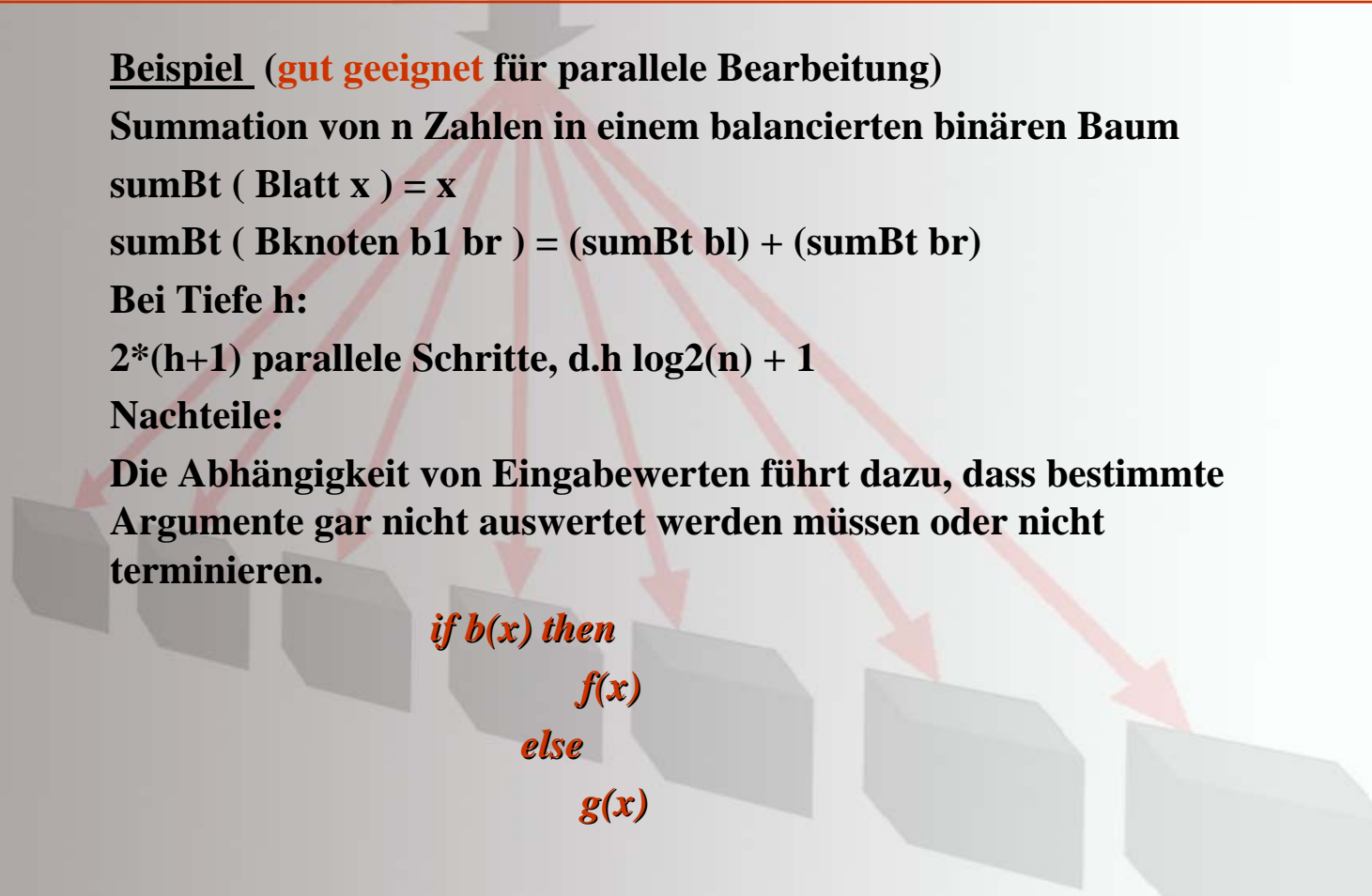
$\text{sumBt}(\text{Bknoten } b_l \text{ } b_r) = (\text{sumBt } b_l) + (\text{sumBt } b_r)$

Bei Tiefe h :

$2^{(h+1)}$ parallele Schritte, d.h. $\log_2(n) + 1$

Nachteile:

Die Abhängigkeit von Eingabewerten führt dazu, dass bestimmte Argumente gar nicht ausgewertet werden müssen oder nicht terminieren.



*if $b(x)$ then
 $f(x)$
else
 $g(x)$*

Übersicht über die Modelle (3) Nichts Explizit

- Statische Strukturen (Skelette)

Die abstrakte Programmen werden aus vorbereiteten Blocks eingebaut.

Algorithmische Skeletten (Higher Order Functions)

Dem Benutzer wird ein Programmrahmen zur Verfügung gestellt. Diese Metafunktionen bekommen als Parameter Benutzerdefinierte Funktionen sowie Datenstrukturen für die Ein-und Ausgabe.

Skelette verkörpern die Programmierparadigmen

Beispiel : Divide&Conquer

`d&c :: (a-> Bool) -> (a->b) -> (a -> [a]) -> ([b] -> b) -> a -> b`
`d&c trivial solve divide conquer p`

if Problem klein genug
then löse das Problem direkt
else

DIVIDE: zerlege das Problem, **CONQUER:** löse jedes Teilproblem rekursiv

COMBINE : berechne aus den Teillösungen die Gesamtlösung

Übersicht über die Modelle (4) Nichts Explizit

Ablauf :

Schritt 1: übergebe das Problem den Wurzel

Schritt 2: falls das Problem zerlegbar → teilen

Schritt 3: wiederhole Schritt 2 rekursiv so lange bis die Teilprobleme direkt gelöst werden können, oder bis der gewünschte Parallelitätsgrad erreicht ist. (**#Prozessoren=#Teillösungen**)

Schritt 4: berechne die Blätter und reiche die Ergebnisse an die Eltern.

Schritt 5: alle inneren Knoten, die Teillösungen von ihren Söhnen empfangen haben, vereinigen diese zu einer neuen Teillösung und geben diese dann an ihren Vater weiter.

Schritt 6: wiederhole Schritt 5 solange, bis die Gesamtlösung die Wurzel erreicht.

Es gibt verschiedene Techniken für die Zuteilung des Berechnungsbaums auf die Prozessoren

Wichtig: Anzahl der Prozessoren hängt von der Anzahl der generierten Subprobleme ab.

Übersicht über die Modelle (4) Nichts Explizit

- Statische Strukturen (Skelette)

- Homomorphe Skelette (Cellular processing Languages- Cellang, Carpet, Cdl, Ceprol)

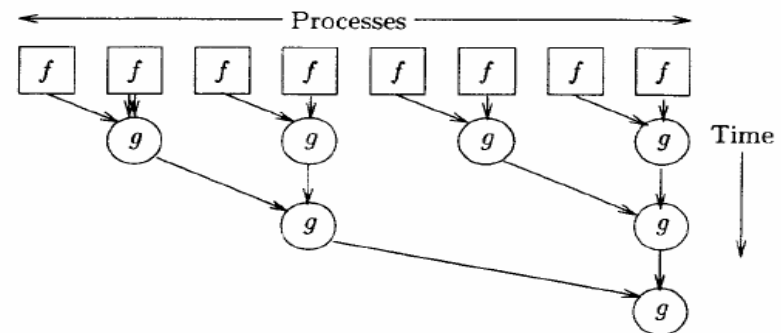
Definition: Funktion h heißt **Homomorphismus**, falls es einen binären Operator \otimes gibt, so dass für alle Listen xs und ys gilt:

$$h(xs ++ ys) = h xs \otimes h ys$$

Beispiel

sum $f = id, g = +$
Maximum $f = id, g = \text{binary max}$
length $f = 1, g = +$
sort $f = id, g = \text{sorted_merge}$

D. B. Skillicorn and D. Talia



Übersicht über die Modelle(5) (Parallelität explizit)

■ Abstraktionsgrad 2

- **Vorteil:** Compiler muss keine komplizierte Datenabhängigkeitsanalyse machen

Dynamische Strukturen (Explizite logische Sprachen)

Interpretation von Hornklauseln

Klausel $\leftarrow C$ ist ein Prozess

Konjunktion $\leftarrow C_1, \dots, C_n$ wird als Prozessvernetzung

Logische Variable zwischen zwei Klauseln ist die Verbindung

Beispiel PARLOG

$$H \leftarrow G_1, G_2, \dots, G_n \mid B_1, B_2, \dots, B_m. \quad N, m \geq 0,$$

Wo H Klauselnkopf ist, die Menge G_i der Guard ist, und B_i der Körper der Klauseln ist. Das Symbol „ \mid “ ist Konjunktion zwischen G_i und B_i

Übersicht über die Modelle(6)_(Parallelität explizit)

Statische Strukturen (Fortran-Varianten, Modula3)

Skelettsverfahren mit Datenstrukturen

Mapping parallel

Beispiel – Fortran

Eine ForALL-Schleife, in der die Iterationen im Loop-Body unabhängig von einander sind, kann parallel ausgeführt werden.

ForALL (I = 1:N, J = 1:M)

*A(I,J) = I * B(J)*

Übersicht über die Modelle(7) (Dekomposition explizit)

■ Abstraktionsgrad 3

Statische Strukturen

Wichtige Vertreter BSP-Modell (Bulk synchronous parallelism)

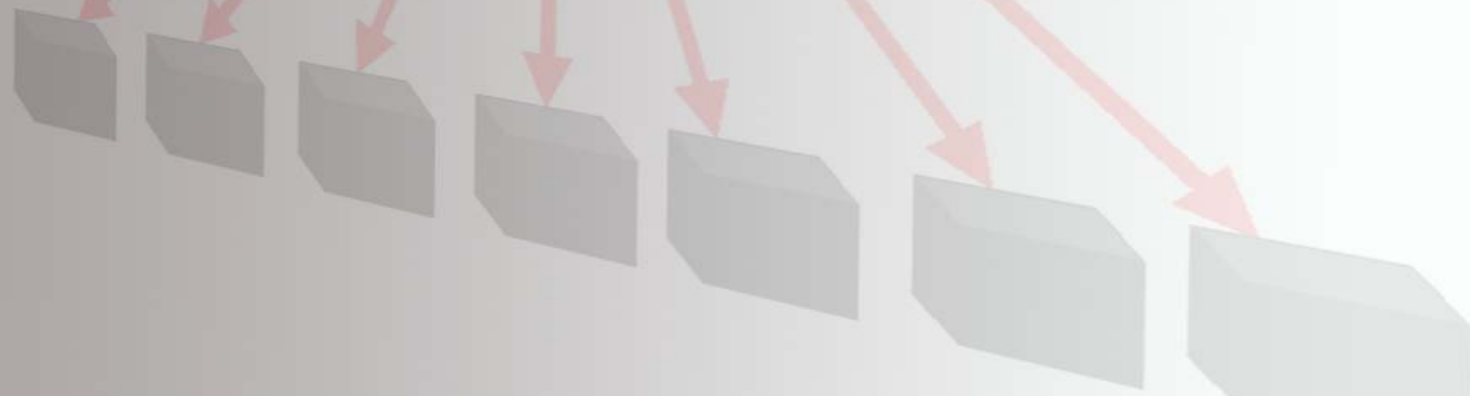
Eine abstrakte BSP-Maschine besteht aus:

Prozessoren mit lokalem Speicher

Netzwerk mit Punkt-zu-Punkt Verbindungen

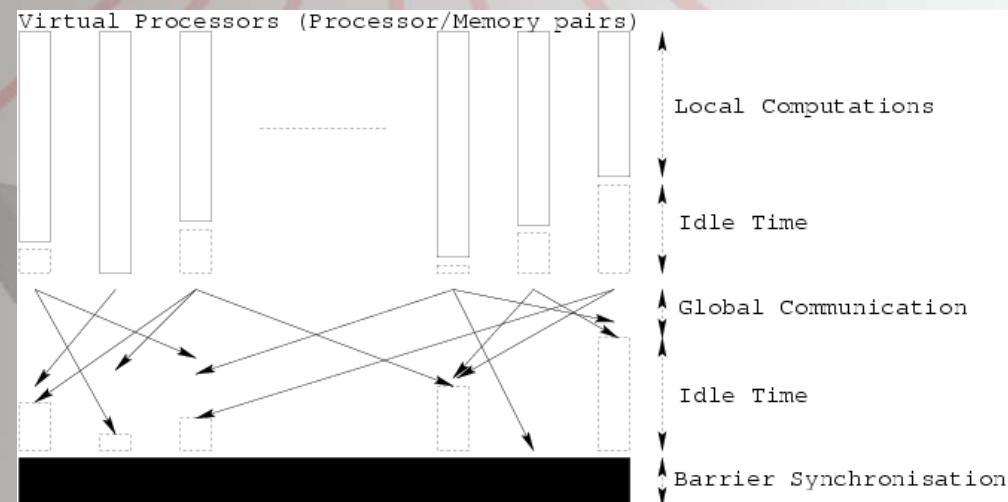
Mechanismus zur Synchronisation der Komponenten

Ein Algorithmus im BSP- Modell besteht aus zeitlichen Abschnitten,



Übersicht über die Modelle(8) (Dekomposition explizit)

- (1) Innerhalb eines **Supersteps** kann jeder Prozessor mit seinen eigenen Daten rechnen oder Nachrichten an andere Prozessoren senden.
- (2) Danach ruft er eine Synchronisationsfunktion auf. Wenn alle Prozessoren ihre Synchronisation gestartet haben, wird gewartet, bis die Nachrichten ihr Ziel erreicht haben.
- (3) Anschließend erfolgt ein neuer Superstep



Übersicht über die Modelle(9)_(Mapping explizi)

■ Abstraktionsgrad 4

Dynamische Strukturen

Coordination Languages-LINDA

ALMS, PCN, Compositional C++

Linda

Linda stellt einen **Datenpool** zur Verfügung, in den Prozesse Daten ablegen oder lesen können. Die drei wichtige Operationen:

- **in**: entferne einen Datenwert
- **rd**: lese einen Datenwert
- **out**: speichere einen Datenwert im Datenpool



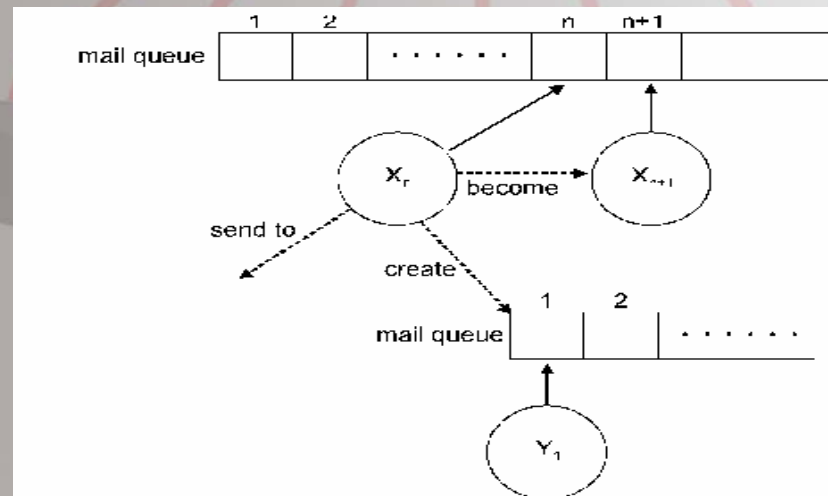
Übersicht über die Modelle (10)_(Alles explizit Synchronisation implizit)

■ Abstraktionsgrad 5

Dynamische Strukturen (Actor-Modell)

In diesem Modell besteht ein paralleles System aus autonomen Objekten, den sogenannten Aktoren. Er besteht aus:

- einer beliebig großen Mail-Queue
- sogenannten actor machine X_n



Übersicht über die Modelle (10)_(Alles explizit Synchronisation implizit)

▪ Parallelität zwischen Sender und Empfänger:

Aufgrund der asynchronen Kommunikation kann der Sender einer Nachricht sofort weitere Operationen ausführen, auch wenn der Empfänger die Nachricht noch nicht vollständig bearbeitet hat oder sogar noch mit anderen Nachrichten beschäftigt ist.

▪ Parallelität der einzelnen Operationen innerhalb einer actor machine:

Da die Operationen, die eine *actor machine* ausführt, im allgemeinen unabhängig voneinander sind, können sie gleichzeitig ausgeführt werden. In X_n wären dies das Verschicken einer Nachricht, das Erzeugen von Y und die Bestimmung des Folgeverhaltens.

▪ Parallelität der einzelnen actor machines:

Mehrere *actor machines* des gleichen Objekts können parallel arbeiten. Dies ist möglich, da sie eigene, unabhängige innere Zustände besitzen. D.h. im Beispiel kann X_{n+1} arbeiten, sobald X_n den Folgezustand festgelegt hat und die Kommunikation $n+1$ eingetroffen ist (diese Nachricht kann natürlich schon längere Zeit in der Mail-Queue stehen), auch wenn X_n noch beschäftigt ist.

Übersicht über die Modelle (11)_(Alles explizit)

■ Abstraktionsgrad 6

Der Programmierer muss sich um alle Details der parallelen Abarbeitung kümmern

Dynamische Strukturen

MPI (Message Passing Interface)

PVM (Parallel Virtual Machine)

Statische Strukturen

PRAM (Parallel Random Access Machine)



Übersicht über die Modelle (12)

- **MPI(Message Passing Interface)**

MPI ist Variante eines Message Passing Protokolls, d.h. Programme schicken sich gegenseitig Nachrichten zu, wenn sie kommunizieren müssen.

MPI stellt Routinen für häufig benötigte Funktionen zur Verfügung, darunter broadcast und reduce

MPI_InitMPI

MPI_Comm_size

MPI_Comm_rank

MPI_Send

MPI_Recv

MPI_FinalizeMPI

Initialisieren

wie viele Prozessoren vorhanden sind

Welcher Prozessor bin ich?

eine Message schicken

eine Message empfangen

terminieren.

Übersicht über die Modelle (13)

- PVM (Parallel Virtual Machine)

Das ist ein Softwarepaket. Es ermöglicht: heterogenes Netz von Computern als einen virtuellen Parallelrechner zu verwenden.

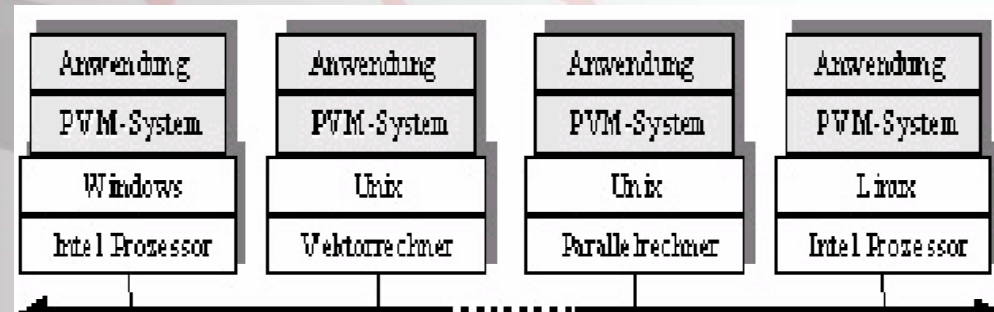
Das **PVM-Paket** besteht aus zwei Bestandteilen:

dem Dämon (pvmd) und der Benutzerbibliothek (libpvm).

Das **PVM-Paket** ist für alle Unix- und Windows-Systeme erhältlich.

PVM unterstützt alle TCP-basierten Rechnernetze wie FDDI, ATM, Ethernet, etc. und enthält C, C++ und Fortran Bibliotheken.

Die Graphik zeigt, dass es möglich ist eine Anwendung auf einem heterogenen System parallel auszuführen.



Übersicht über die Modelle (14)

- **PRAM**(Parallel Random Access maschine) –
Eine Registermaschine, die parallel Befehlsbearbeitung ausführt.

Beispiel (1)

```
for <Bedingung> pardo <Anweisungen>  
for i = 1 to 100 pardo xi := 0  
Wert 0 -> 100 Speicherplätze gleichzeitig  
initialisieren
```

Beispiel (2)

gegeben eine Liste, die unsortiert in
den Speicherzellen x1 bis xn gespeichert
sind. Gesucht MAX-Element

```
for i, j = 0 to n pardo  
  if xi >= xj  
    then bij := 1  
  else bij := 0  
fi  
for i = 0 to n pardo  
  mi := bi1 & bi2 &... & bin
```

Quellen

- **David B. Skilicic & Domenico Talia:**
Models and Languages for parallel Computation“
- **Prof. Dr. Herbert Kuchen**
„Programmierung mit Algorithmischen Skeleten“
- **Prof. Dr. R. Loogen**
„Parallele Programmierung Vorlesung S/2005“
- **Marco Gilbert, Universität Trier**
„Parallel Virtual Machine
- **Andreas Justen : Proseminar**
„Paralleles Rechnen“

Ende

Die Präsentation steht zur Verfügung auf
www.informatik.uni-marburg.de/~chilikov/SeminarParallele ~

Danke