Zentralübung Rechnerstrukturen im SS2007

Verbindungsstrukturen & Parallele Programmierung

David Kramer

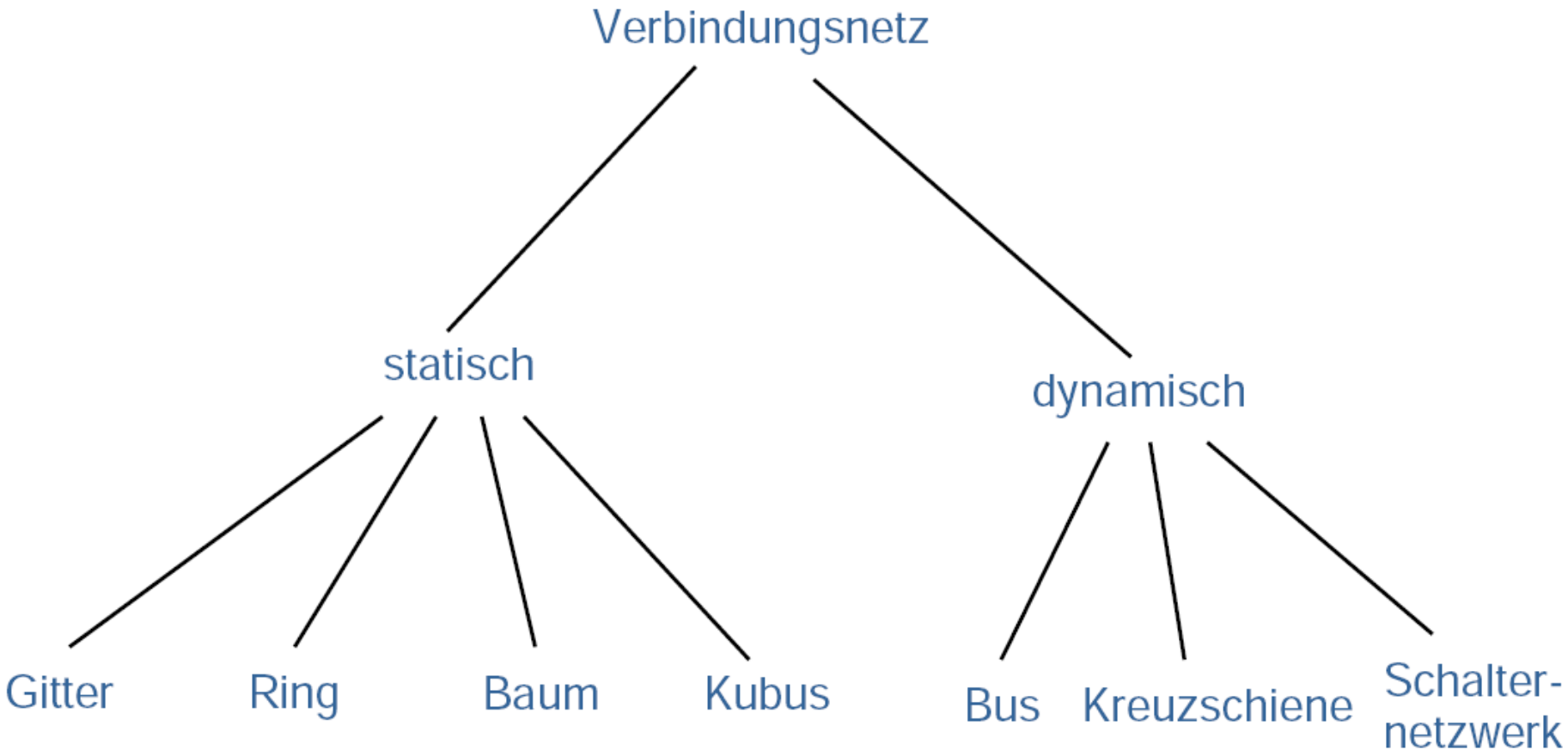
kramer@ira.uka.de

Universität Karlsruhe (TH) - Forschungsuniversität
Institute für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

8. August 2007

Überblick Verbindungsstrukturen

- Topologie: Klassifikation

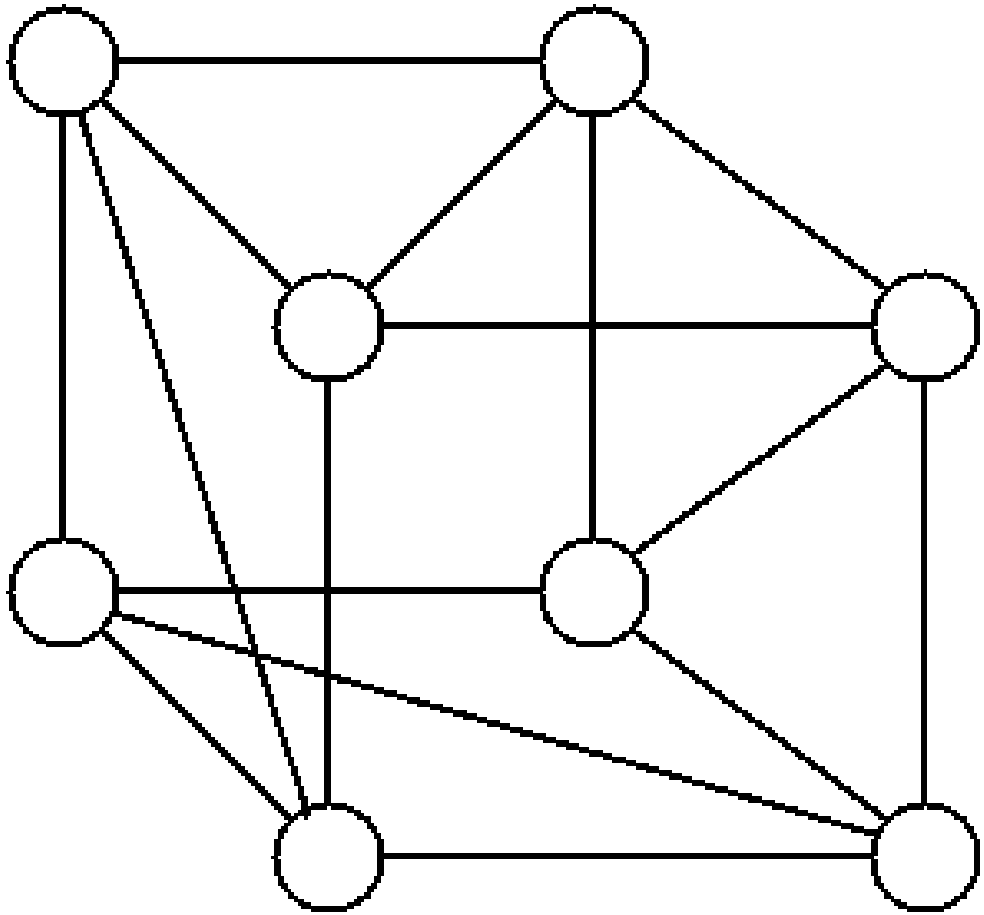


Überblick Verbindungsstrukturen

- *Knotengrad* ist die Anzahl der in einen Knoten mündenden (indizierten) Kanten
- *Verbindungsgrad* ist die Anzahl der Kanten von einem Knoten zu anderen Knoten
- Der *Durchmesser* ist die maximale Distanz zwischen zwei Knoten
- Schneidet man einen Graphen in zwei gleich große in sich zusammenhängende Teile und betrachtet die Menge der Kanten, die diesen Schnitt kreuzen, so bezeichnet man die Kardinalität der kleinsten Kantenmenge (über alle möglichen Schnitte) als *minimale Bisektionsbreite*
- Die *Diskonnektivität* ist definiert als *#Knoten / min. Bisektionsbreite*
- Die *Kosteneffektivität* ist definiert als *Verbindungsgrad * max(Diameter, Diskonnektivität)*

Aufgabe 5a: Statische Verbindungsstrukturen

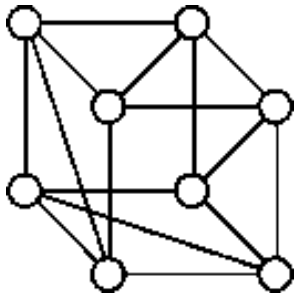
Gegeben sei ein Verbindungsnetzwerk, dessen Topologie in der folgenden Abbildung dargestellt ist:



Aufgabe 5a: Statische Verbindungsstrukturen

a.1) Bestimmen Sie den Verbindungsgrad, den Diameter, die minimale Bisektionsbreite, die Diskonnektivität und die Kosteneffektivität.

Antwort

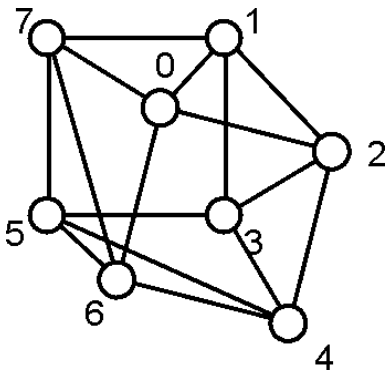


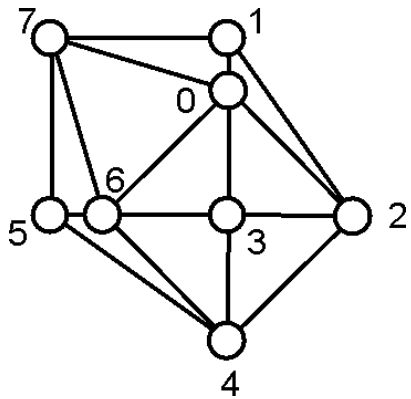
Verbindungsgrad	4
Durchmesser	2
min. Bisektionsbreite	6
Diskonnektivität	8/6 = 1,33
Kosteneffektivität	4 * max (2, 4/3) = 8

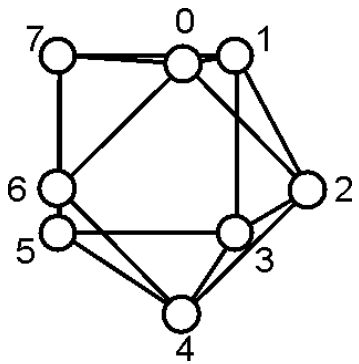
Aufgabe 5a: Statische Verbindungsstrukturen

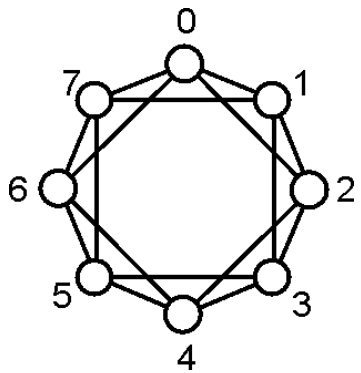
a.2) Um welche Form eines Verbindungsnetzwerkes handelt es sich in diesem Fall?

Antwort Chordaler Ring









Aufgabe 5a: Statische Verbindungsstrukturen

a.3) Liegt Redundanz vor? Falls ja, wieviele Verbindungsleitungen können ausfallen bevor eine Verbindung zwischen zwei beliebigen Knoten nicht mehr geschalten werden kann?

Antwort Es liegt Redundanz vor. Da der Verbindungsgrad jedes Knotens 4 ist und wir bidirektionale Leitungen haben, können bis zu drei Leitungen ausfallen und dennoch jeder Knoten von einem anderen erreicht werden. Allerdings kann beim Ausfall einer Kante der Durchmesser steigen, das heißt es könnten längere Wege notwendig sein.

Aufgabe 5a: Statische Verbindungsstrukturen

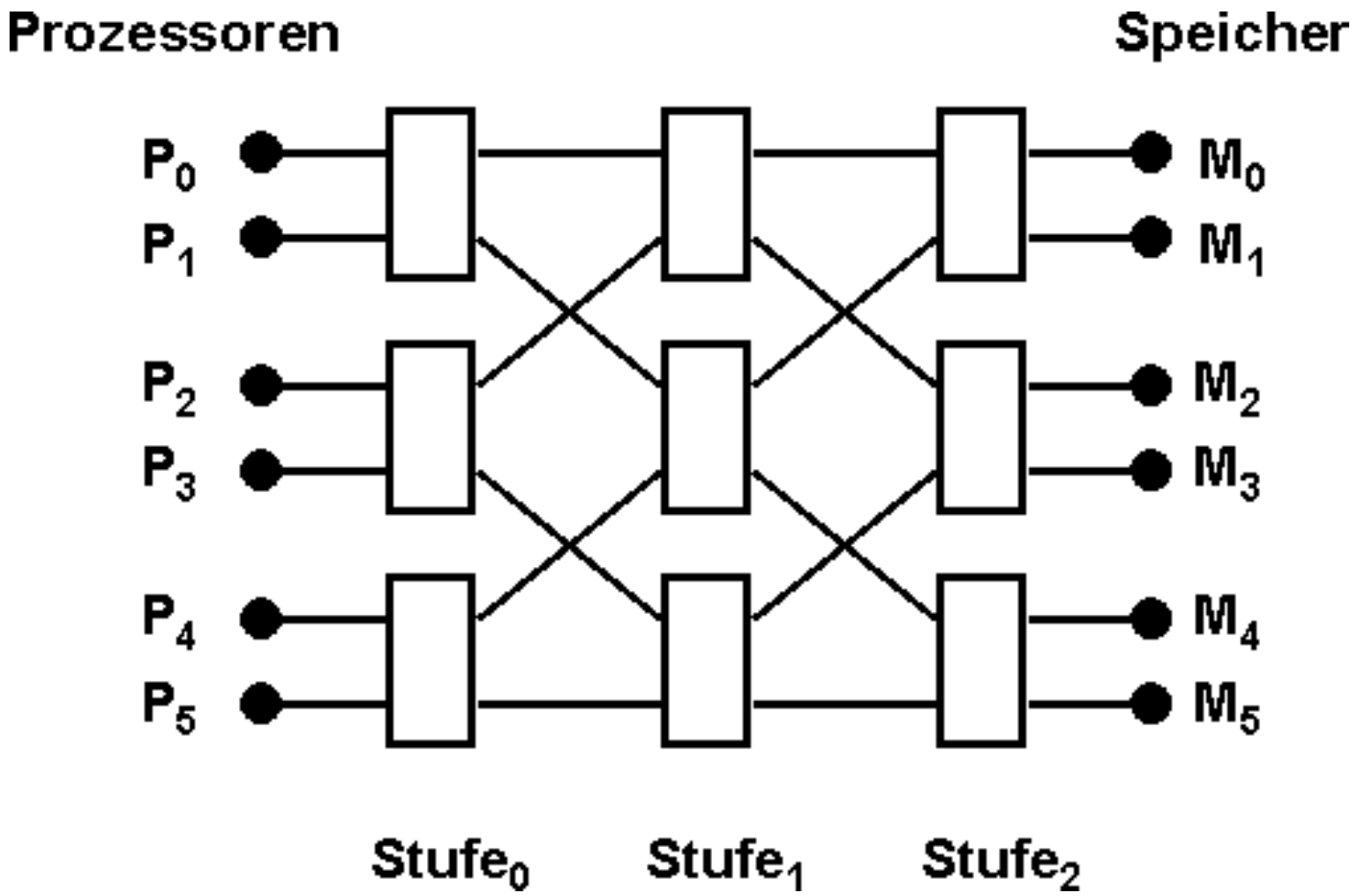
a.4) Vergleichen Sie diese Netzwerktopologie mit den Topologien (unidirektionaler) Ring, 2D-Gitter, (binärer) Baum und Hyperkubus in den Punkten Verbindungsgrad, Durchmesser, min. Bisektionsbreite, Diskonnektivität und Kosteneffektivität.

Antwort N = # Knoten

	Ring	2D-Gitter	Baum	Hyperkubus	Aufgabe
Verbindungsgrad	2	2 - 4	1 - 3	$log_2(N)$	4
Durchmesser	$N-1$	$2*(\sqrt{N}-1)$	$2(log_2 N-1)$	$log_2(N)$	$\sqrt{N}-1$
min. Bisektionsbreite	2	$log_2(N)$	1	N/2	$2*\sqrt{N}$
Diskonnektivität	N/2	$\sqrt{N}$	N	2	$N/(2*\sqrt{N})$
Kosteneffektivität	2N-2	$8*(\sqrt{(N)}-1)$	$6\log_2 N-6$	$\frac{N}{2}*(log_2(N)$	$4*\sqrt{N}-1$

Aufgabe 5b: Dynamische Verbindungstrukturen

Gegeben sei folgendes dynamisches Verbindungsnetzwerk, das 6 Prozessoren mit 6 Speichern verbindet:



Aufgabe 5b: Dynamische Verbindungstrukturen

b.1) Kann zwischen jedem Prozessor-Speicher-Paar eine Verbindung hergestellt werden?

Antwort Ja!

b.2) Kann jede Permutation generiert werden? Begründen Sie Ihre Antwort!

Antwort Nein. Es gibt 6! = 720 Permutationen aber nur 2^9 = 512 Schalterstellungen.

Aufgabe 5b: Dynamische Verbindungstrukturen

b.3) Ist das Netzwerk redundant? Begründen Sie Ihre Antwort!

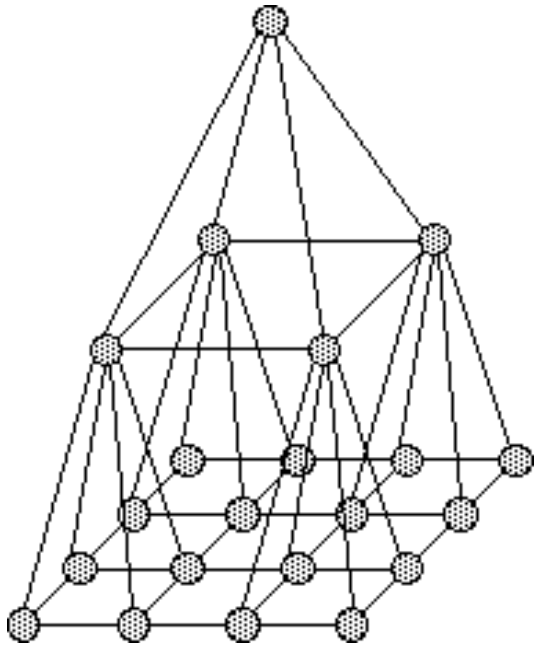
Antwort Ja.
Beispiel: Die Verbindung P0 zu M0 kann auf zwei Arten zustande kommen:
Stufe 0 parallel, Stufe 1 parallel, Stufe 2 parallel
Stufe 0 überkreuz, Stufe 1 parallel, Stufe 2 überkreuz

b.4) Was ist die minimale Verbindungszahl bei der eine Blockierung auftritt? Geben Sie ein Beispiel an.

Antwort Schon bei zwei Verbindungen kann eine Blockierung auftreten: P0-M5 und P1-M4

Aufgabe 5c: Adressierung in Multiprozessorsystemen

Ein Multiprozessorsystem sei mit einer Pyramidenstruktur aus e Ebenen verbunden.



Aufgabe 5c: Adressierung in Multiprozessorsystemen

c.1) Wieviele Knoten hat eine Pyramide mit e Ebenen?

Antwort Definition: Spitze ist die Ebene 0!

$\rightarrow$ Jede Ebene i ist quadratisch und hat eine Kantenlänge von 2^i.

$\rightarrow \#Knoten = \sum_{i=0}^{e-1}(2^i)^2 = \sum_{i=0}^{e-1} 4^i = \frac{1-4^e}{1-4}$

c.2) Wieviele Nachbarn kann ein Knoten maximal haben?

Antwort Ein Knoten im Inneren der Pyramide hat 1 „Vater“, 4 „Brüder“ und 4 „Söhne“, also 9 Nachbarn.

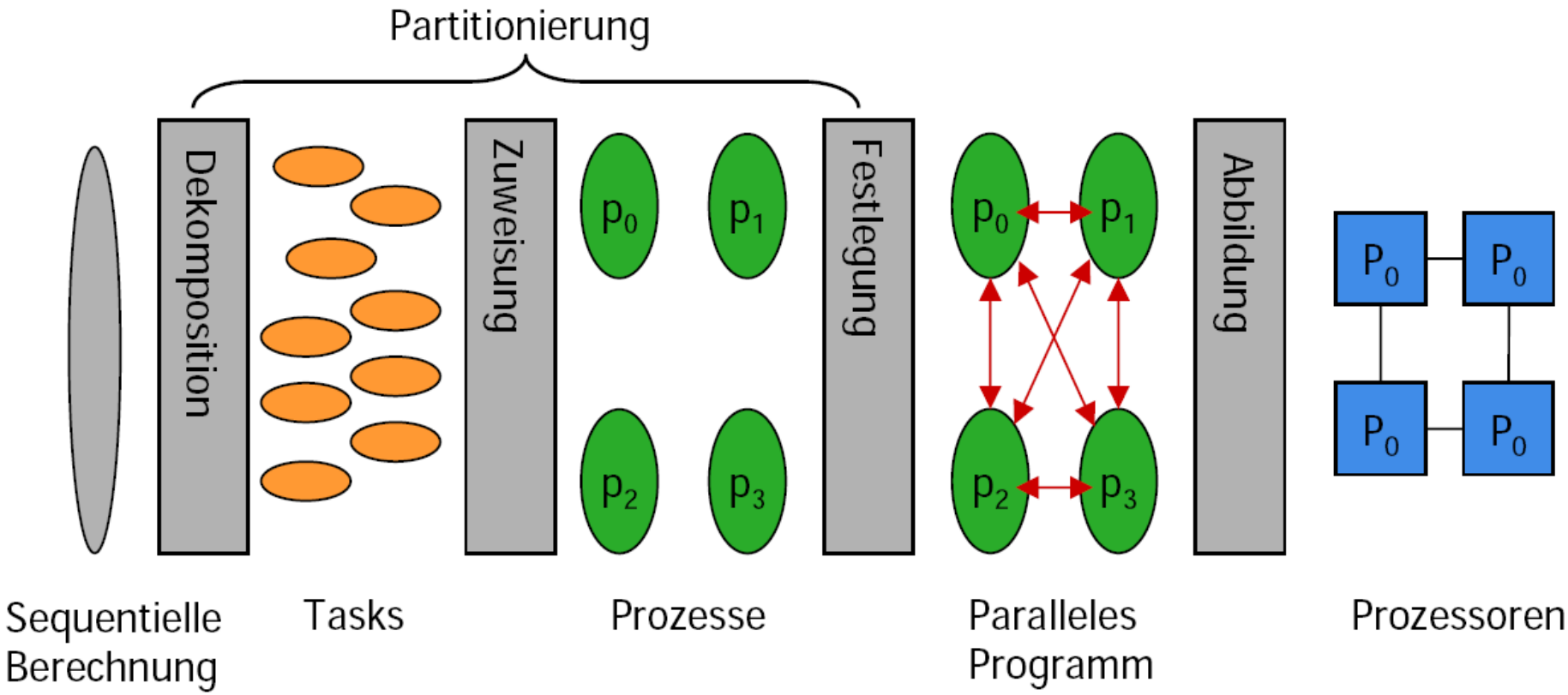
Aufgabe 5c: Adressierung in Multiprozessorsystemen

c.3) Gegeben sei für diese Pyramide folgendes Adressierungsschema:
(Ebene i, Koordinate x, Koordinate y)
(i, x, y größer gleich 0)
Geben sie die Adressen aller Nachbarknoten eines Knotens an, sowie die Bedingungen für die Existenz des Nachbarn.

Antwort Existenzbedingung: $0 \leq i < e, 0 \leq x < 2^i, 0 \leq y < 2^i$

Vater:	(i-1,	x DIV2,	y DIV 2)
Brüder	(i,	x + 1,	y)
	(i,	x - 1,	y)
	(i,	x,	y +1)
	(i,	x,	y - 1)
Söhne	(i+1,	2 x,	2 y)
	(i+1,	2 x + 1,	2 y)
	(i+1,	2 x,	2 y + 1)
	(i+1,	2 x + 1,	2 y + 1)

- Parallelisierungsprozess



Einführung in Parallele Programmierung - Programmiermodelle

- Shared-Memory-Programmiermodell
  - Threadbibliotheken (Win32 API, POSIX Threads)
  - Compilerdirektiven (OpenMP)
- Nachrichten-orientiertes Programmiermodell
  - MPI
- Datenparalleles Programmiermodell
  - High Performance Fortran

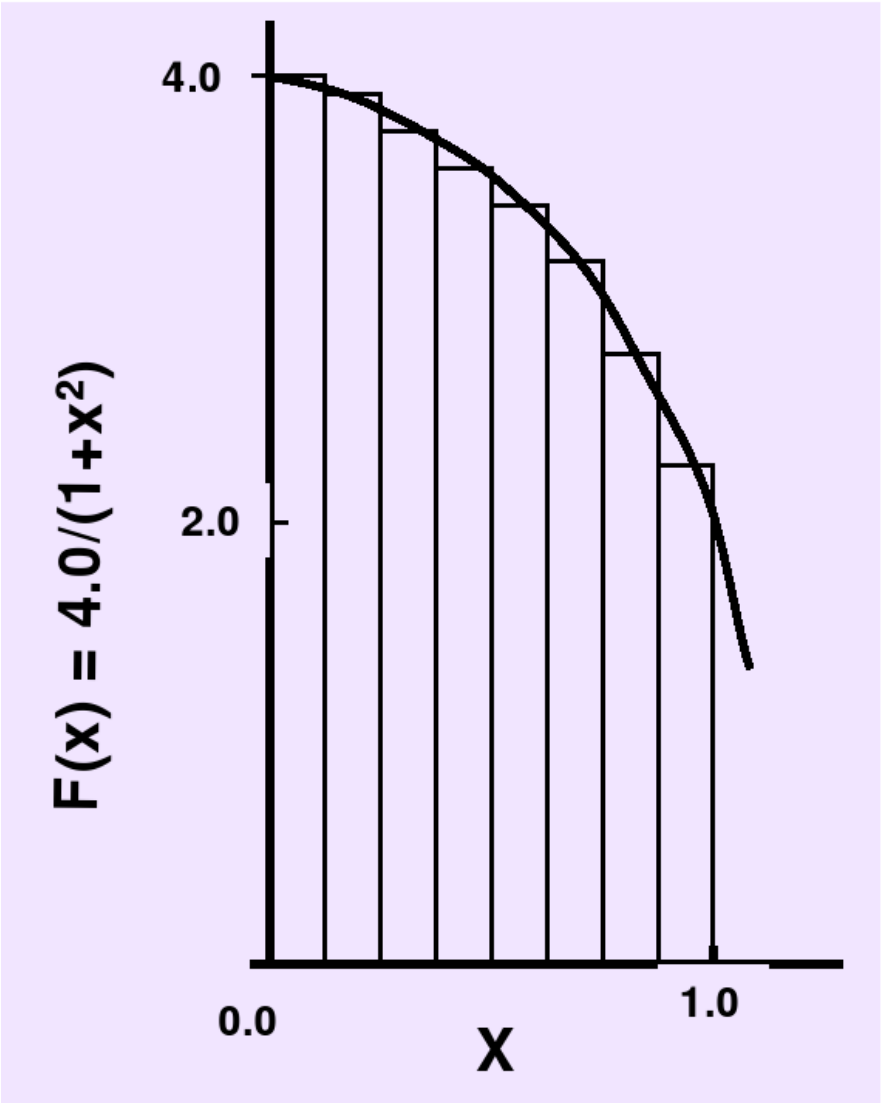
Beispiel: PI-Berechnung (Numerische Integration)

- Wir wissen:

  $\int_0^1 \frac{4.0}{(1+x^2)} dx = \pi$

- Näherung des Integrals durch

  $\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$



Beispiel: PI-Berechnung (Sequentielles Programm)

```
static long num_steps = 100000;
double step;
void main ()
{       int i;   double x, pi, sum = 0.0;

        step = 1.0/(double) num_steps;

        for (i=1;i<= num_steps; i++){
                x = (i-0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step * sum;
}
```

Thread-Programmierung

- Alle Threads eines Prozesses teilen sich Adressraum, Daten, Filehandler, ...
- Parallele Programme für Shared-Memory-Systeme bestehen aus mehreren Threads
- Threads werden meist vom Betriebsystem verwaltet
- Unterstützung vom Betriebsystem notwendig, z.B. für die Erstellung, ...
- Vorteil: durch die Thread-Bibliothek erhält man eine detailierte Kontrolle über die Threads
- Nachteil: die Thread-Bibliothek erzwingt, dass man die Kontrolle über die Threads übernimmt

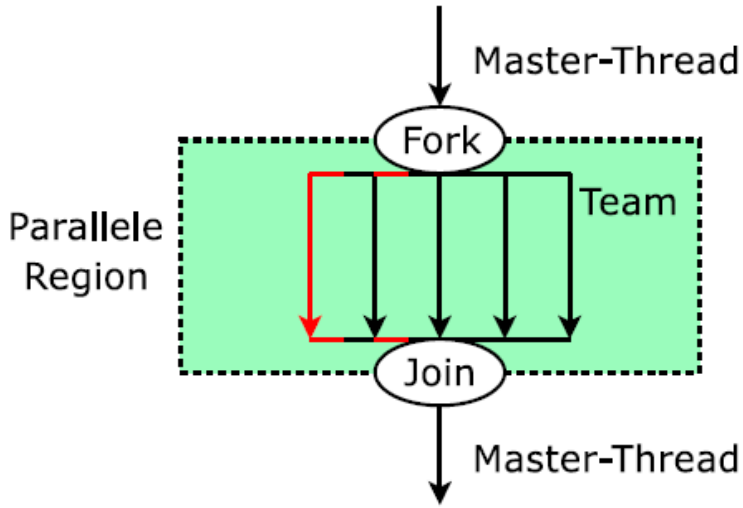
Beispiel: PI-Berechnung (Win32 API)

```
#include <windows.h>
#define NUM_THREADS 2
HANDLE thread_handles[NUM_THREADS];
CRITICAL_SECTION hUpdateMutex;
static long num_steps = 100000;
double step;
double global_sum = 0.0;

void Pi (void *arg)
{
   int i, start;
  double x, sum = 0.0;


  start = *(int *) arg;
  step = 1.0/(double) num_steps;

  for (i=start;i<= num_steps;
i=i+NUM_THREADS){
       x = (i-0.5)*step;
      sum = sum + 4.0/(1.0+x*x);
  }
  EnterCriticalSection(&hUpdateMutex);
  global_sum += sum;
  LeaveCriticalSection(&hUpdateMutex);
}
```

```
void main ()
{
  double pi; int i;
  DWORD threadID;
  int threadArg[NUM_THREADS];

  for(i=0; i<NUM_THREADS; i++)   threadArg[i] = i+1;

  InitializeCriticalSection(&hUpdateMutex);

  for (i=0; i<NUM_THREADS; i++){
          thread_handles[i] = CreateThread(0, 0,
            (LPTHREAD_START_ROUTINE) Pi,
                    &threadArg[i], 0, &threadID);
}

  WaitForMultipleObjects(NUM_THREADS,
                    thread_handles, TRUE,INFINITE);

  pi = global_sum * step;

  printf(" pi is %f \n",pi);
}
```

OpenMP

- OpenMP ist eine offene Spezifikation von Übersetzerdirektiven, Bibliotheken und Umgebungsvariablen, spezifiziert für die Parallelisierung von Programmen auf gemeinsamen Speicher
- Verwendet das Join-Fork-Modell



- Mehrere (auch verschaltelte) parallele Regionen pro Programm sind zugelassen

Beispiel: PI-Berechnung (OpenMP)

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{       int i;   double x, pi, sum = 0.0;
        step = 1.0/(double) num_steps;
        omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction(+:sum) private(x)
        for (i=1;i<= num_steps; i++){
                x = (i-0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step * sum;
}
```

Message Passing interface

- MPI ist ein Standard für die nachrichtenbasierte Kommunikation in einem Multiprozessorsystem
- Nachrichtenbasierter Ansatz gewährleistet eine gute Skalierbarkeit
- Bibliotheksfunktionen koordinieren die Ausführung von mehreren Prozessen, sowie Verteilung von Daten, per Default keine gemeinsamen Daten
- Single Programm Multiple Data (SPMD) Ansatz

Beispiel: PI-Berechnung (MPI)

```
#include <mpi.h>
void main (int argc, char *argv[])
{
        int i, my_id, numprocs;  double x, pi, step, sum = 0.0 ;
        step = 1.0/(double) num_steps ;
        MPI_Init(&argc, &argv) ;
        MPI_Comm_Rank(MPI_COMM_WORLD, &my_id) ;
        MPI_Comm_Size(MPI_COMM_WORLD, &numprocs) ;
        my_steps = num_steps/numprocs ;
        for (i=my_id*my_steps; i<(my_id+1)*my_steps ; i++)
        {
                x = (i+0.5)*step;
                sum += 4.0/(1.0+x*x);
        }
        sum *= step ;
        MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                        MPI_COMM_WORLD) ;
}
```

Aufgabe 5d: Parallele Programmierung

Eine Methode aus der Numerischen Mathematik arbeitet auf einem 2D-Gitter mit $n * n$ Knoten. In jeder Iteration werden zwei Schritte durchgeführt. Im ersten Schritt wird der Zustand eines Knoten basieren auf seinem aktuellen Zustand und den Zuständen seiner Nachbarknoten aktualisiert. Im zweiten Schritt werden die neuen Zustände an die Nachbarknoten weitergereicht. Zur Erfüllung der Aufgabe werden m Iterationen benötigt.
Gegeben ist ein SMP-Knoten mit P Prozessoren.

Aufgabe 5d: Parallele Programmierung

d.1) Berechnen Sie den SpeedUp der Anwendung bei der Ausführung auf einem SMP-Knoten. Welches Problem tritt hier auf?

Antwort Für jede Iteration gilt:

Sequentielle Zeit: $n^2 + n^2 = 2n^2$

Parallele Zeit: $\frac{n^2}{P} + n^2$

Speed-Up: $\frac{2n^2}{\frac{n^2}{P}+n^2} = \frac{2P}{P+1} < 2$

Das Problem ist hier, dass der Speicher als limitierender Faktor wirkt und den Speed-Up auf max. 2 beschränkt.

Aufgabe 5d: Parallele Programmierung

d.2) Um eine Leistungssteigerung zu erhalten, wird der SMP-Knoten durch eine NUMA-Architektur mit P Prozessoren und P Speichern ersetzt. Welche Beschleunigung lässt sich nun erzielen?

Antwort Durch die NUMA-Architektur sind P parallele Speicherzugriffe möglich $\rightarrow$

Sequentielle Zeit: $n^2 + n^2 = 2n^2$

Parallele Zeit: $\frac{n^2}{P} + \frac{n^2}{P}$

Speed-Up: $\frac{2n^2}{\frac{n^2}{P} + \frac{n^2}{P}} = \frac{2n^2}{\frac{2n^2}{P}} = P$

Aufgabe 5d: Parallele Programmierung
Überarbeitete Lösung

In der obigen Musterlösung wurde nicht berücksichtigt, dass die berechneten und in den Hauptspeicher zurückgeschriebenen Werte erst wieder gelesen werden müssen, bevor die nächste Berechnungsphase begonnen werden kann. Demzufolge müssen die zwei Phasen um eine weitere Phase ergänzt werden.
In der erste Phasen werden nun die Daten für die Berechnungsphase (der zweiten Phase) aus dem Hauptspeicher geladen. In der zweiten Phase wird nun ein neuer Wert berechnet, welcher in der dritten Phase zurückgeschrieben wird.
Weiterhin wird hier idealisierter Weise eine perfekte Platzierung der Daten im Hauptspeicher des NUMA-Systems angenommen, die sich in einem realem System nur schwer erreichen läßt.

Aufgabe 5d: Parallele Programmierung
Überarbeitete Lösung

Phasen:

- 1. Phase (Daten holen): 4 Hauptspeicherzugriffe um die 4 Werte der 4 Nachbarknoten zu holen (pro Knoten)
- 2. Phase (Berechnung): Berechnung des neuen Wertes
- 3. Phase (Daten schreiben): 1 Hauptspeicherzugriff un den neu berechneten Werte in den Hauptspeicher zurückzuschreiben (pro Knoten)

Aufgabe 5d: Parallele Programmierung
Überarbeitete Lösung

d.1) Berechnen Sie den SpeedUp der Anwendung bei der Ausführung auf einem SMP-Knoten. Welches Problem tritt hier auf?

Antwort Für jede Iteration gilt:

Sequentielle Zeit: $4 * n^2 + n^2 + n^2 = 6 * n^2$

Parallele Zeit: $4 * n^2 + \frac{n^2}{P} + n^2 = 5 * n^2 + \frac{n^2}{P}$

Speed-Up: $\frac{6*n^2}{\frac{n^2}{P}+5*n^2} = \frac{6P}{5P+1}$

Das Problem, dass der Speicher hier als limitierender Faktor wirkt, wird durch diesen Umstand noch verstärkt und eine Beschleunigung wird nur sehr schwer erreichbar sein.

Aufgabe 5d: Parallele Programmierung
Überarbeitete Lösung

d.2) Um eine Leistungssteigerung zu erhalten, wird der SMP-Knoten durch eine NUMA-Architektur mit P Prozessoren und P Speichern ersetzt. Welche Beschleunigung lässt sich nun erzielen?

Antwort Durch die NUMA-Architektur sind P parallele Speicherzugriffe möglich $\rightarrow$

Sequentielle Zeit: $4 * n^2 + n^2 + n^2 = 6 * n^2$

Parallele Zeit: $\frac{4n^2}{P} + \frac{n^2}{P} + \frac{n^2}{P} = \frac{6*n^2}{P}$

Speed-Up: $\frac{6*n^2}{\frac{6*n^2}{P}} = P$

Aufgabe 5e: Verständnisfragen

(keinen Anspruch auf Vollständigkeit, es kann noch weitere Punkte geben)

e.1) Welche Vorteile bieten Netzwerke auf Basis eines 3D-Torus?

Antwort konstanter Verbindungsgrad, einfache Erweiterbarkeit, einfaches Routing, hohe Fehlertoleranz, ...

e.2) Welche Vorteile bieten Netzwerke auf Basis eines Fat-Trees?

Antwort einfacher Aufbau, einfaches Routing, einfaches Adressierungsschema, ...

Aufgabe 5e: Verständnisfragen

e.3) Was sind die Vor-/Nachteile des Shared-Memory-Programmiermodells?

Antwort

- Vorteile: Niedrige Latenzen, hohe Kommunikationsbandbreite, (implizite Kommunikation)
- Nachteile: Skalierbarkeit

e.4) Was sind die Vor-/Nachteile des Massage-Passing-Programmiermodells?

Antwort

- Vorteile: Skalierbarkeit
- Nachteile: im Vergleich mit Shared-Memory hohe Latenzen, komplexe Programmierung (explizite Kommunikation)

Ausblick

- Nächste Übung: 5. Juli 2007
- Thema: Cache-Kohärenz und -Konsistenz