Rechnerstrukturen im SS2008

Parallelismus, Parallele Programmierung und Verbindungsstrukturen

Oliver Mattes
mattes@ira.uka.de

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur

05. Juni 2008

Überblick

Parallele Programmierung
Parallelismus – Quantitative Maßzahlen

- Aufgabe 1.1
- Aufgabe 1.2
- Aufgabe 1.3
- Aufgabe 3

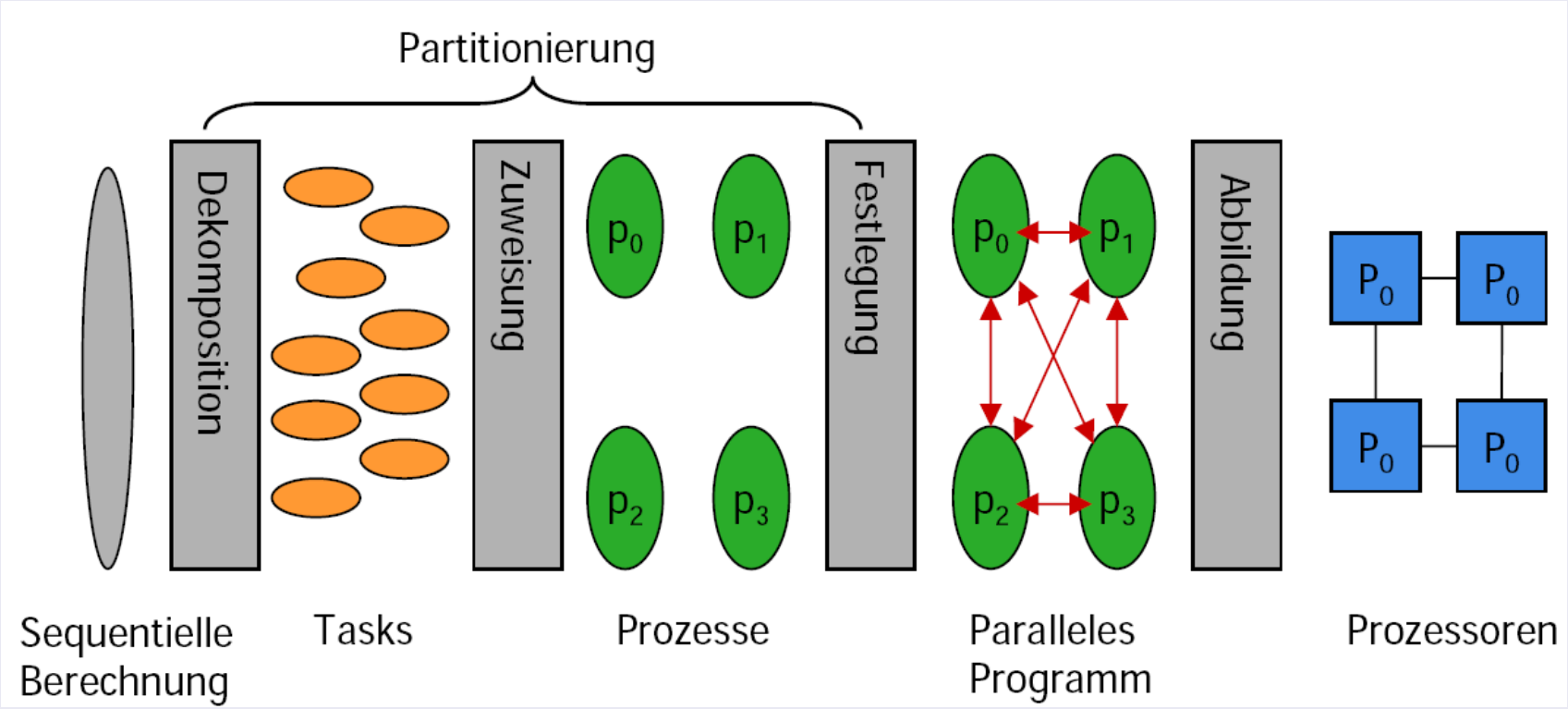
Verbindungsstrukturen

- Statische Verbindungsstrukturen
  Aufgabe 2.1
- Dynamische Verbindungsstrukturen
  Aufgabe 2.2

Verständnisfragen

- Aufgabe 4

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 I

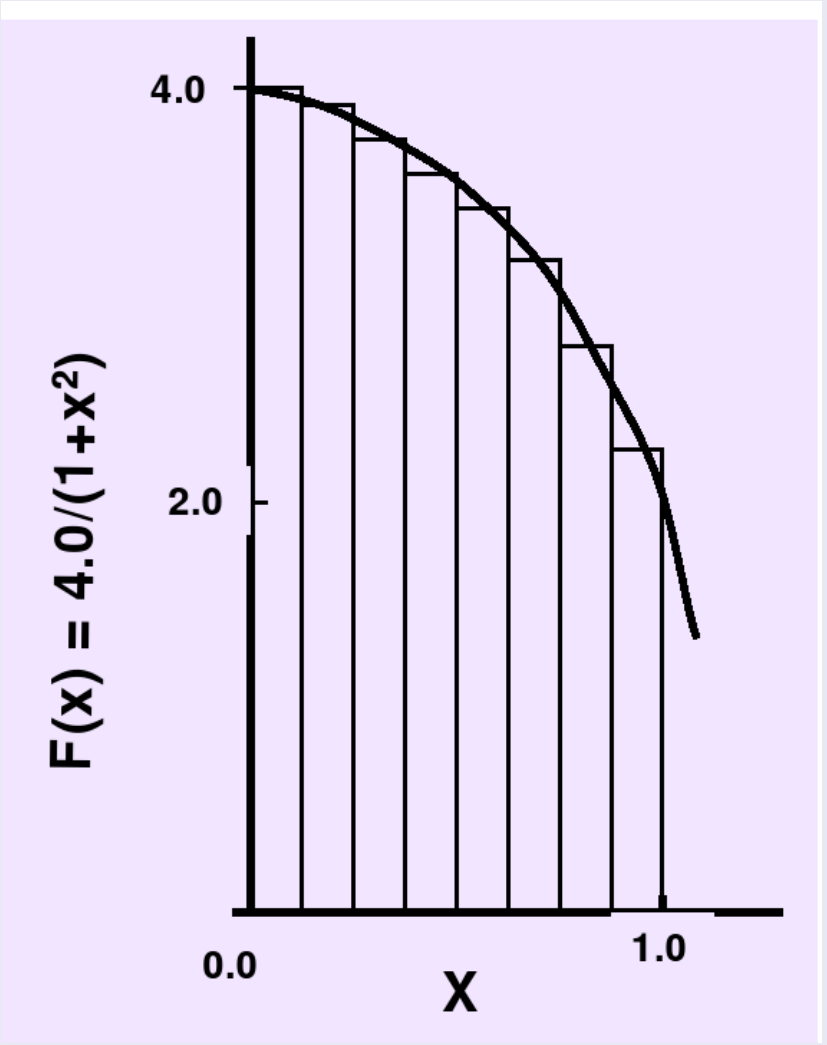
Numerische Integration:

- Wir wissen:

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

- Näherung des Integrals durch

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



Beispiel: PI-Berechnung II

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 III

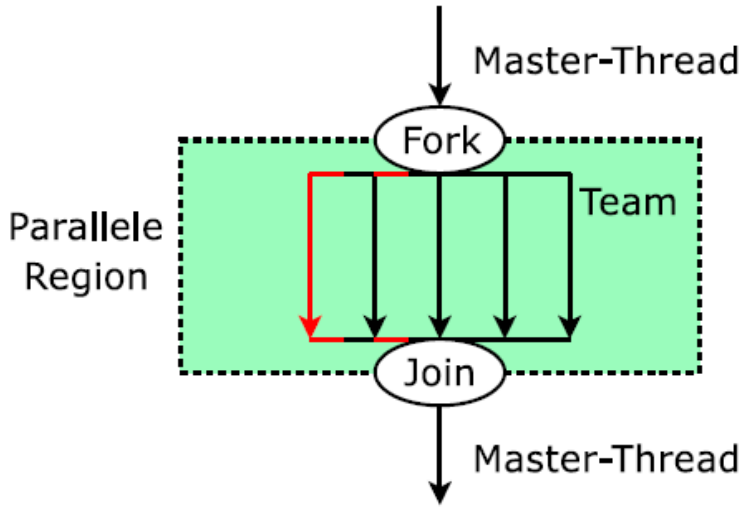
Win32 API:

```
#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 IV

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)

- 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 V

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) ;
}
```

Parallelismus – Quantitative Maßzahlen I

Definitionen:

Definitionen:

- $P(1)$ Anzahl der Einheitsoperationen auf einem Einprozessorsystem
- $P(n)$ Anzahl der Einheitsoperationen auf einem Multiprozessorsystem mit n Prozessoren
- $T(1)$ Ausführungszeit auf einem Einprozessorsystem in Schritten
- $T(n)$ Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten

Vereinfachende Voraussetzungen:

- $T(1) = P(1)$
- $T(n) \leq P(n)$

Parallelismus – Quantitative Maßzahlen II

Beschleunigung (Speed-Up)

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

üblicherweise gilt:

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

Effizienz

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

daraus folgt:

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

Parallelismus – Quantitative Maßzahlen III

Mehraufwand für die Parallelisierung

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

$$1 \leq R(n)$$

Mehraufwand für die Organisation, Synchronisation und Kommunikation der Prozessoren in Multiprozessorsystemen.

Parallelindex

$$I(n) = \frac{P(n)}{T(n)}$$

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

Mittlerer Grad der Parallelität. Anzahl der parallelen Operationen pro Zeiteinheit.

Parallelismus – Quantitative Maßzahlen IV

Amdahls Gesetz (Amdahl's Law)

$$T(n) = \underbrace{\frac{T(1)}{n} * (1-a)}_{1} + \underbrace{T(1) * a}_{2}$$

- a mit $(0 \leq a \leq 1)$ ist der Anteil eines Programms, der nur sequentiell ausgeführt werden kann.

1. Ausführungszeit des parallel ausführbaren Programmteils $(1-a)$ dividiert durch die Anzahl n der parallel arbeitenden Prozessoren.
2. Ausführungszeit des nur sequentiell ausführbaren Programmteils a.

Parallelismus – Quantitative Maßzahlen V

Amdahls Gesetz (Amdahl's Law)

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

Durch Einsetzen in die Formel für die Beschleunigung erhält man:

$$S(n) \quad \leq \quad \frac{1}{a}$$

$\Rightarrow$ Die erreichbare Beschleunigung ist durch die Anzahl der sequentiellen Operationen begrenzt.

Parallelismus – Quantitative Maßzahlen VI

Skalierbarkeit

- Von Skalierbarkeit spricht man, wenn das Hinzufügen von weiteren Verarbeitungselementen zu einer kürzeren Gesamtausführungszeit führt, ohne daß das Programm geändert werden muß.
- Wichtig für die Skalierbarkeit ist eine angemessene Problemgröße

Aufgabe 1.1

Gegeben sei ein Multiprozessorsystem mit 16 Prozessoren. Die Leistungssteigerung gegenüber einem Einprozessorsystem sei $S(16) = 8$. Die Ausführungszeit auf dem Einprozessorsystem sei $T(1) = 80$ und die Anzahl der auszuführenden Einheitsoperationen auf dem Multiprozessorsystem sei $P(16) = 16$.

- Berechnen Sie die Effizienz $E(16)$, die parallele Ausführungszeit $T(16)$ und den Parallelindex $I(16)$.
- Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil der Programme, der nur sequentiell ausführbar ist.

Aufgabe 1.1 (forts.)

- Berechnen Sie die Effizienz $E(16)$, die parallele Ausführungszeit $T(16)$ und den Parallelindex $I(16)$.

$$E(n) = \frac{S(n)}{n} \quad \Rightarrow \quad E(16) = \frac{S(16)}{16} = \frac{8}{16} = 0,5$$

$$S(n) = \frac{T(1)}{T(n)} \quad \Rightarrow \quad T(16) = \frac{T(1)}{S(16)} = \frac{80}{8} = 10$$

$$I(16) = \frac{P(n)}{T(n)} \quad \Rightarrow \quad I(16) = \frac{P(16)}{T(16)} = \frac{16}{10} = 1,6$$

Aufgabe 1.1 (forts.)

Korrektur der Aufgabe:

In der Übung wurde die Frage gestellt, wie diese Aufgabe mit der Gleichung $1 \leq S(n) \leq I(n) \leq n$ zu vereinbaren ist. Diese Formel ist richtig, jedoch wurde bei der Auswahl der in der Aufgabe vorgegebenen Werte mehr auf schöne Rechungen geachtet, als auf sinnvolle Werte.

In der Aufgabenstellung dieser Teilaufgabe wurde fälschlicherweise für $P(16)$ ein äußerst unrealistischer Wert gewählt.

Hier hätte $P(n)$ größer als $P(1)$ sein sollen, da bei parallelen Programmen immer ein gewisser Kommunikations- und Synchronisationsaufwand vorhanden ist.

Wie wir wissen gilt $P(1) = T(1)$ mit laut Aufgabenstellung $T(1) = 80$. Somit wäre $P(16) = 100$ oder $P(16) = 120$ ein besserer Wert gewesen.

Aufgabe 1.1 (forts.)

Korrektur der Aufgabe:

Mit $P(16) = 100$ ergibt sich:

$$I(16) = \frac{P(16)}{T(16)} = \frac{100}{10} = 10$$

und somit wird die Gleichung erfüllt:

$$\begin{aligned} 1 \leq S(n) &\leq I(n) \leq n \\ 1 \leq 8 \quad &\leq 10 \quad \leq 16 \end{aligned}$$

Aufgabe 1.1 (forts.)

- Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil der Programme, der nur sequentiell ausführbar ist.

Ahmdahls Gesetz:

$$T(n) = T(1) * \left(\frac{(1-a)}{n} + a \right) \; = \; T(1) * \frac{1 - a \; + \; na}{n}$$

$$\Rightarrow \quad 10 = 80 * \frac{1 - a \; + \; 16a}{16} = 80 * \frac{1 \; + \; 15a}{16} = 5 * (1 + 15a)$$

$$\Rightarrow \quad 15a = 1 \quad \Rightarrow \quad a = \frac{1}{15} \approx 6,7\,\%$$

$\approx 6,7\,\%$ des Programmcodes sind nur sequentiell ausführbar.

Aufgabe 1.2

Die Ausführungszeit einer sequentiellen Anwendung betrage T Sekunden. Von dieser Anwendung lassen sich 20 % nicht parallelisieren. Die verbleibenden 80 % werden zwischen den Prozessoren fair verteilt. „Fair“ bedeutet, daß jeder Prozessor ungefähr den gleichen Anteil der zu parallelisierenden Aufgabe bearbeitet und jeder gleichviel Zeit benötigt.

Beispielsweise betrage die Ausführungszeit der parallelen Anteile der Anwendung 20 % der sequentiellen Ausführungszeit, wenn vier Prozessoren sie ausführen.

- Setzen Sie voraus, daß die Parallelisierung keinen Aufwand verursacht. Berechnen Sie die Beschleunigung und die Effizienz bei einer unterschiedlichen Anzahl von Prozessoren. Fügen Sie die Ergebnisse in die vorgegebenen Tabelle ein. Evaluieren Sie zusätzlich die Skalierbarkeit der einzelnen Lösungen.

Aufgabe 1.2 (forts.)

Par. Ausführungszeit: $T_{par} = 20\,\% + \frac{80\,\%}{P} * T_{seq}$

Beschleunigung: $S = \frac{T_{seq}}{T_{par}}$

Effizienz: $E = \frac{S}{P}$

Prozessoren	2	4	8	16	32
Beschleunigung	1,67	2,5	3,33	4	4,44
Effizienz	0,83	0,625	0,42	0,25	0,14

Aufgabe 1.2 (forts.)

Par. Ausführungszeit: $T_{par} = 20\,\% + \frac{80\,\%}{P} * T_{seq}$

Beschleunigung: $S = \frac{T_{seq}}{T_{par}}$

Effizienz: $E = \frac{S}{P}$

Prozessoren	2	4	8	16	32
Beschleunigung	1,67	2,5	3,33	4	4,44
Effizienz	0,83	0,625	0,42	0,25	0,14

Die Skalierbarkeit ist sehr schlecht. Grund dafür ist, daß ein großer Anteil vom Code nicht parallelisierbar ist. Die Ausführungszeit von diesem Anteil bestimmt einen zunehmenden Anteil der gesamten Ausführungszeit mit steigender Anzahl von Prozessoren.

Beschleunigung: $S = \frac{T}{\left(20\,\% + \frac{80\,\%}{P}\right) * T} \xrightarrow{\text{P sehr groß}} \frac{1}{20\,\%} = 5$

Die Effizienz strebt damit gegen 0.

Aufgabe 1.2 (forts.)

- Zur Steigerung der Genauigkeit der Berechnung nehmen Sie nun an, daß durch jeden zusätzlichen Prozessor ein Aufwand von 1 % der sequentiellen Ausführungszeit eingefügt wird. Berechnen Sie Beschleunigung und Effizienz für 64 Prozessoren.

Aufgabe 1.2 (forts.)

- Zur Steigerung der Genauigkeit der Berechnung nehmen Sie nun an, daß durch jeden zusätzlichen Prozessor ein Aufwand von 1 % der sequentiellen Ausführungszeit eingefügt wird. Berechnen Sie Beschleunigung und Effizienz für 64 Prozessoren.

Beschleunigung:

$$S(64) = \frac{T_{seq}}{T_{par}} = \frac{T_{seq}}{\left(20\,\% + \frac{80\,\%}{64} \; + \; 1\,\% * 64\right) * T_{seq}} = 1,17$$

Effizienz:

$$E(64) = \frac{S(64)}{64} = \frac{1,17}{64} = 0,018$$

Aufgabe 1.3

Ein Einprozessorsystem soll erweitert werden. Es existieren folgende, in der Anschaffung gleichteure Alternativen:

- Einsetzen eines mathematischen Koprozessors.
  Dieser bietet eine zweifach schnellere Ausführung der Gleitkommaarithmetik als der vorhandene Systemprozessor. Es ist allerdings keine parallele Verarbeitung, d.h. der gleichzeitige Einsatz von Haupt- und Koprozessor, möglich.
- Ausbau zu einem 2-fach SMP-System,
  d.h. die Installation eines zweiten zum vorhandenen identischen Hauptprozessor.

Das zu bearbeitende Problem ist zu 25 % parallelisierbar. Der Anteil der Gleitkommaarithmetik am Gesamtprogramm beträgt 30 %. Bestimmen Sie, welche der beiden Möglichkeiten unter dem Gesichtspunkt der Ausführungszeit zu bevorzugen ist.

Aufgabe 1.3 (forts.)

Betrachten wir zunächst den einfacheren Fall, d.h. die Installation eines zweiten Prozessors. Dies macht das vorhandene System zu einem speichergekoppelten System vom Typ „UMA“, da beide Prozessoren auf den gleichen in dem System installierten Speicher zugreifen.

Aufgrund der Angaben gilt somit $n = 2$ und $a = 1 - 0,25 = 0,75$. Eingesetzt in die Formel für die Beschleunigung $S(n)$ unter Brüchsichtigung von Amdahls Gesetz ergibt sich folglich:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1) * \left(\frac{1-a}{n} + a\right)} = \frac{n}{1 - a + n * a}$$

$$S(n) = \frac{2}{0,25 + 2 * 0,75} = \frac{2}{1,75} \approx 1,14$$

Aufgabe 1.3 (forts.)

Für das zweite – strikt sequentielle – System gilt, daß kein Parallelisierungaufwand $R(n)$ vorliegt, d.h. $R(n) = 1$. Dies ist auch für das optimale parallele System der Fall, für welches $\frac{T(1)}{n} = T_{opt}$ die Laufzeit darstellt.

Sei nun α der Anteil des Programms, der nicht optimiert werden kann (d.h. die Entsprechung von a), so läßt sich die neue Ausführungszeit T_{new} als die von α abhängige gewichtete Summe der optimierten und alten Ausführungszeit beschreiben:

$$T_{new} = (1 - \alpha) * T_{opt} \; + \; \alpha * T_{old}$$

Aufgabe 1.3 (forts.)

Weiterhin gilt aufgrund der Angaben für die optimierbare Programmteile $T_{old} = T_{opt} * 2$ bzw. $T_{opt} = \frac{1}{2} * T_{old}$,

Für die Beschleunigung ergibt sich damit:

$$\begin{aligned} S' &= \frac{T_{old}}{T_{new}} = \frac{T_{old}}{(1-\alpha) * T_{opt} \ + \ \alpha * T_{old}} \\ &= \frac{T_{old}}{\frac{(1-\alpha)}{2} * T_{old} \ + \ \alpha * T_{old}} \\ &= \frac{2}{(1-\alpha) \ + \ 2 * \alpha} \end{aligned}$$

Somit folgt aus $\alpha = 1 - 0,3 = 0,7$ unmittelbar

$$S' = \frac{2}{0,3 \ + \ 2 * 0,7} \approx 1,18 > S(2)$$

Aufgabe 3

Eine Methode aus der Numerischen Mathematik arbeitet auf einem 2D-Torus mit $n * n$ Knoten.

In jeder Iteration werden zwei Schritte durchgeführt:

1. Im ersten Schritt werden die Zustände der Knoten, basierend auf ihrem aktuellen Zustand und den Zuständen ihrer Nachbarknoten, aktualisiert. Dazu müssen erst diese Zustände zuerst aus dem Speicher gelesen werden.
2. Im zweiten Schritt werden die neuen Zustände zurück in den Speicher geschrieben.

Aufgabe 3

Eine Methode aus der Numerischen Mathematik arbeitet auf einem 2D-Torus mit $n * n$ Knoten.

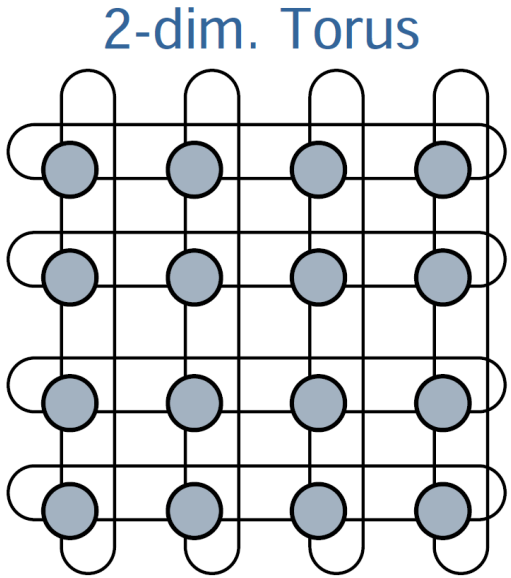
In jeder Iteration werden zwei Schritte durchgeführt:

1. Im ersten Schritt werden die Zustände der Knoten, basierend auf ihrem aktuellen Zustand und den Zuständen ihrer Nachbarknoten, aktualisiert. Dazu müssen erst diese Zustände zuerst aus dem Speicher gelesen werden.
2. Im zweiten Schritt werden die neuen Zustände zurück in den Speicher geschrieben.

Gegeben sei ein SMP-Knoten mit P Prozessoren.

a) Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt hierbei auf?

Aufgabe 3



SMP gemeinsamer Adreßraum, globaler Speicher

$\Rightarrow$ meist **UMA** (Uniform Memory Access), d.h. gleiche Zugriffszeit von allen Knoten

DSM gemeinsamer Adreßraum, physikalisch verteilter Speicher

$\Rightarrow$ meist **NUMA** (Non-Uniform Memory Access), d.h. unterschiedliche Zugriffszeiten

Aufgabe 3 (forts.)

a) Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt hierbei auf?

- Sequentielle Zeit $T(1)$:

$$\underbrace{5n^2}_{\text{Daten aus Speicher holen}} + \underbrace{n^2}_{\text{Berechnung}} + \underbrace{n^2}_{\text{Zurückschreiben}} = 7n^2$$

- Parallele Zeit $T(P)$:

$$\underbrace{5n^2}_{\text{Daten aus Speicher holen}} + \underbrace{\frac{n^2}{P}}_{\text{par. Berechnung}} + \underbrace{n^2}_{\text{Zurückschreiben}} = 6n^2 + \frac{n^2}{P}$$

- Beschleunigung $S(P)$:

$$S(P) = \frac{T(1)}{T(P)} = \frac{7n^2}{6n^2 + \frac{n^2}{P}} = \frac{7}{6 + \frac{1}{P}} < \frac{7}{6} \approx 1,17$$

Das Problem ist hierbei, daß der Speicher als limitierender Faktor wirkt und damit die Beschleunigung stark beschränkt ist.

Aufgabe 3 (forts.)

Um eine Leistungssteigerung zu erhalten, wird der SMP-Knoten durch eine NUMA-Architektur mit P Prozessoren und P Speichern ersetzt.

Beachten Sie dabei folgende vereinfachende Annahme:

- Erfolgt ein Speicherzugriff auf einen entfernten Speicher, so benötigt ein Speichergriff zwei Zeiteinheiten.

b) Welche Beschleunigung läßt sich erzielen, wenn die Zustände der Nachbarknoten immer aus entfernten Speichern abgerufen werden müssen?

Aufgabe 3 (forts.)

b) Welche Beschleunigung läßt sich erzielen, wenn die Zustände der Nachbarknoten immer aus entfernten Speichern abgerufen werden müssen?

- Sequentielle Zeit $T(1)$:

$$\underbrace{5n^2}_{\text{Daten aus Speicher holen}} + \underbrace{n^2}_{\text{Berechnung}} + \underbrace{n^2}_{\text{Zurückschreiben}} = 7n^2$$

- Parallele Zeit $T(P)$:

$$\underbrace{\underbrace{\frac{4*2*n^2}{P}}_{\substack{\text{4 Werte der Nachbarknoten aus entferntem Speicher}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{eigener Wert im lokalen Speicher}}}}_{\text{Daten aus Speicher holen}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{parallele Berechnung}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{Zurückschreiben in lokalen Speicher}}} = \frac{11n^2}{P}$$

Aufgabe 3 (forts.)

b) Welche Beschleunigung läßt sich erzielen, wenn die Zustände der Nachbarknoten immer aus entfernten Speichern abgerufen werden müssen?

- Sequentielle Zeit $T(1)$:

$$T(1) = 5n^2 + n^2 + n^2 = 7n^2$$

- Parallele Zeit $T(P)$:

$$T(P) = \frac{4 * 2 * n^2}{P} + \frac{n^2}{P} + \frac{n^2}{P} + \frac{n^2}{P} = \frac{11n^2}{P}$$

- Beschleunigung $S(P)$:

$$S(P) = \frac{T(1)}{T(P)} = \frac{7n^2}{\frac{11n^2}{P}} = \frac{7}{11}P$$

$$\text{für } P = 2: \quad S(2) = \frac{14}{11} \approx 1,27$$

$$\text{für } P = 3: \quad S(3) = \frac{21}{11} \approx 1,90$$

$\Rightarrow$ Die Beschleunigung skaliert mit $\frac{7}{11}$ der Problemgröße (linear).

Aufgabe 2 und 4

Hinweis:

Aufgabe 2 (Verbindungsstrukturen) und Aufgabe 4 (Verständnisfragen) wurden nicht in der Übung behandelt!

Diese Aufgaben werden u.a. Inhalt der Übung am 19.06.2008 sein.

Fragen?