Karlsruher Institut für Technologie

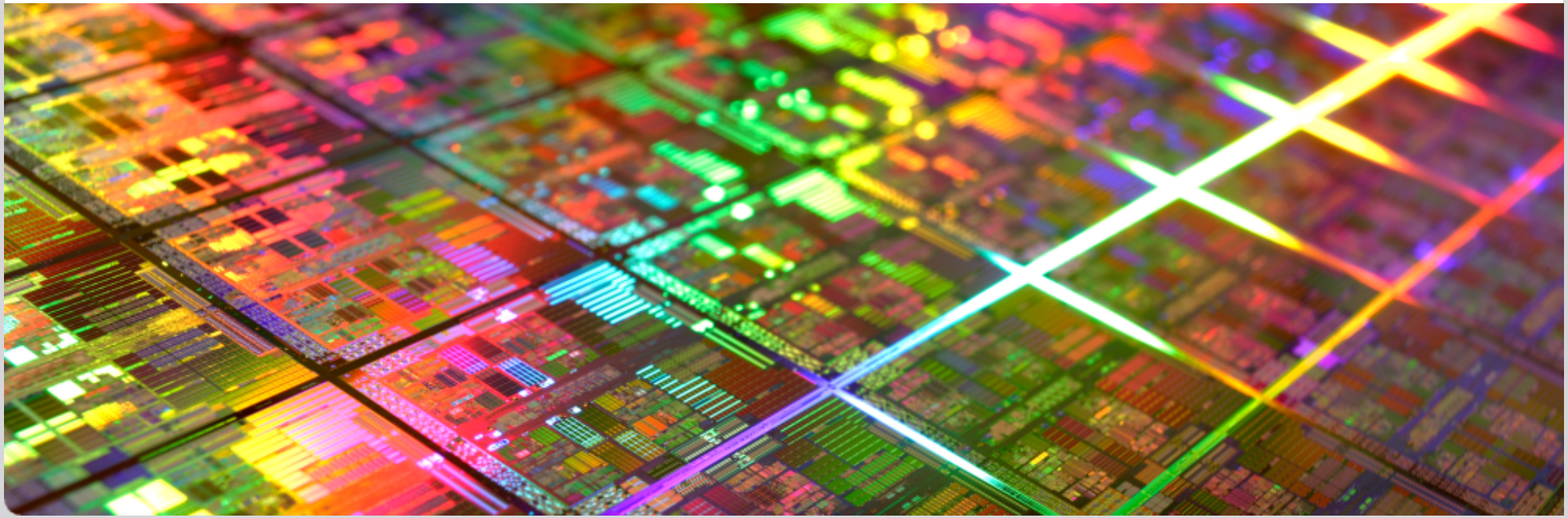
# Zentralübung Rechnerstrukturen im SS 2015

## Parallelismus und Parallele Programmierung

Mario Kicherer, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

10. Juni 2015



# Parallelismus: Parallelrechner

Karlsruher Institut für Technologie

## Klassifikation nach Flynn

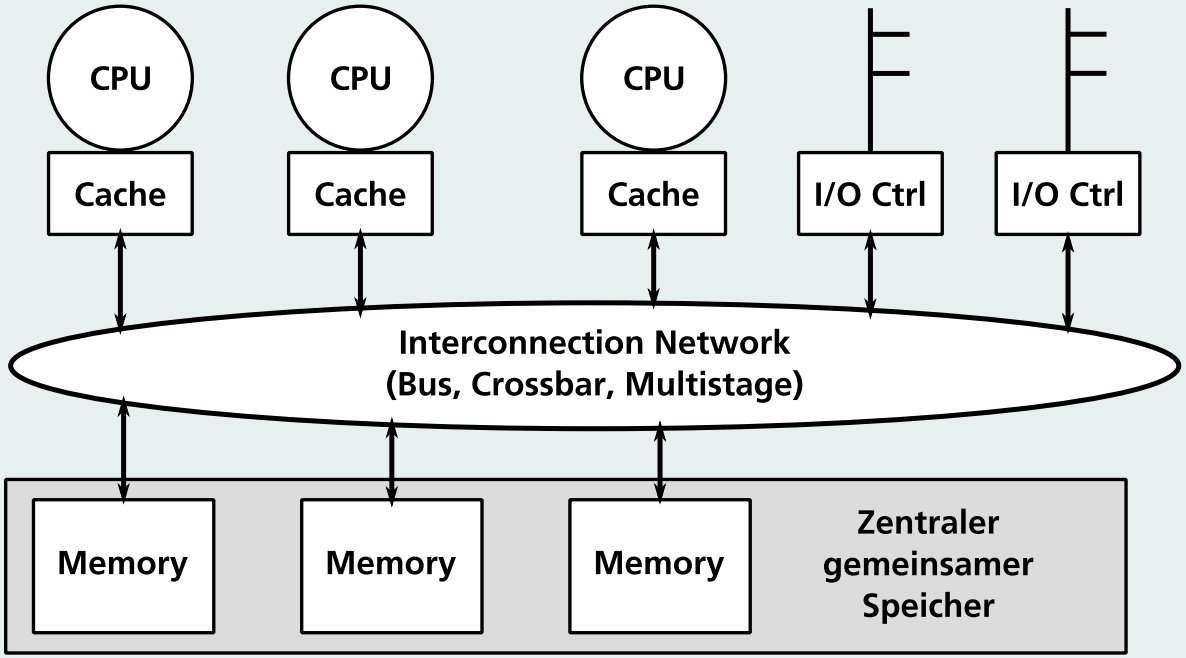
Vier Klassen von Rechnerarchitekturen:

- **SISD** Single Instruction, Single Data
  - Uniprozessor
- **SIMD** Single Instruction, Multiple Data
  - Vektorrechner, Feldrechner
- **MISD** Multiple Instructions, Single Data
  - ?
- **MIMD** **Multiple Instructions, Multiple Data**
  - Multiprozessor

# Parallelismus: Parallele Architekturen

## Multiprozessoren mit gemeinsamem Speicher

- **Globaler Speicher $\Leftrightarrow$ Gemeinsamer Adressraum**
- Beispiel: Symmetrischer Multiprozessor (SMP)

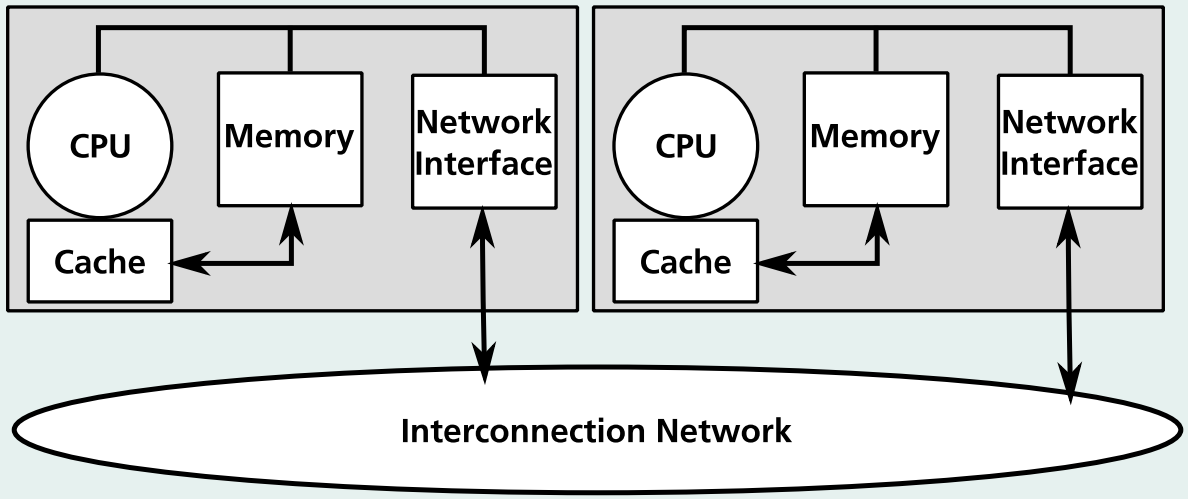



- UMA: Uniform Memory Access

# Parallelismus: Parallele Architekturen

## Multiprozessoren mit verteiltem Speicher

- **Verteilter Speicher $\Leftrightarrow$ Verteilter Adressraum**
- Beispiel: Cluster
- Nachrichtengekoppelter (Shared-nothing-) Multiprozessor

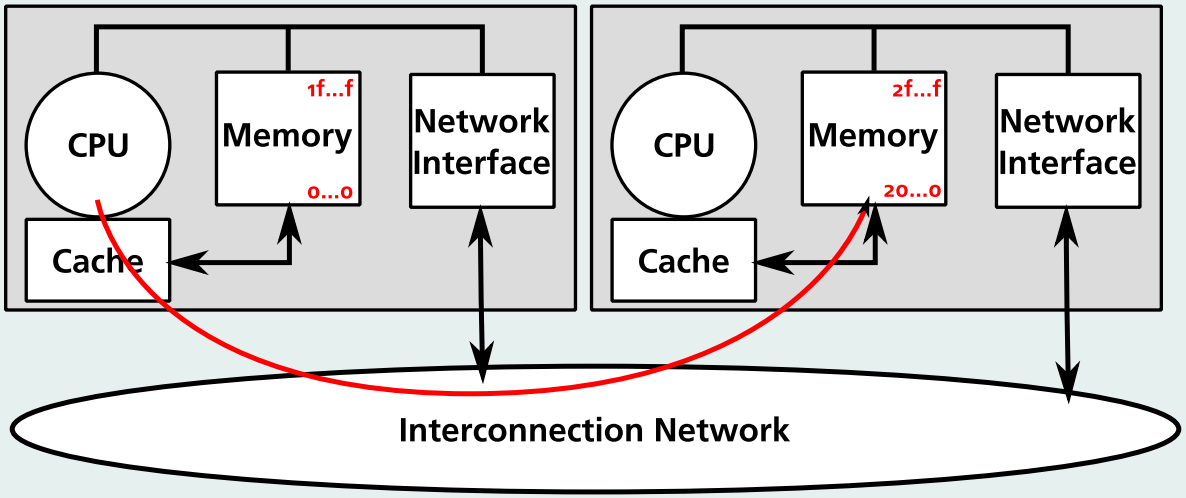



- NORMA: No Remote Memory Access

# Parallelismus: Parallele Architekturen

## Multiprozessoren mit verteiltem gemeinsamen Speicher

- **Verteilter Speicher $\Leftrightarrow$ Gemeinsamer Adressraum**
- Beispiel: Distributed-shared-memory Multiprozessor (DSM)




- NUMA: Non-Uniform Memory Access
- CC-NUMA: Cache-Coherent Non-Uniform Memory Access

# Parallelismus: Parallele Architekturen

Karlsruher Institut für Technologie

## Konfigurationen von Multiprozessoren

| | Globaler Speicher | Physikalisch verteilter Speicher |
|---|---|---|
| gemeinsamer Adreßraum | SMP Symmetrischer Multiprozessor | DSM Distributed-shared-memory Multiprozessor |
| | UMA: Uniform Memory Access | NUMA: Non-Uniform Memory Access |
| verteilte Adreßräume | leer | Nachrichtengekoppelter (Shared-nothing) Multiprozessor / Cluster |
| | | NORMA: No Remote Memory Access |

# Parallelismus: Parallele Architekturen

Karlsruher Institut für Technologie

## Programmiermodelle

- Shared-Memory-Programmiermodell
- Nachrichten-orientiertes Programmiermodell
- Datenparalleles Programmiermodell

$\Rightarrow$ Einführung erfolgt später in der Übung!

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## 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

Verbesserung der Verarbeitungsgeschwindigkeit

## 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

Verbesserung der Verarbeitungsgeschwindigkeit

## 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

Verbesserung der Verarbeitungsgeschwindigkeit

## 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:

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

**Algorithmenunabhängige vs. Algorithmenabhängige Definition**

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## Algorithmenunabhängige Definition

- **Absolute Beschleunigung** und **absolute Effizienz** erhält man, indem der beste sequentielle Algorithmus mit dem besten parallelen Algorithmus verglichen wird.

## Algorithmenabhängige Definition

- **Relative Beschleunigung** und **relative Effizienz** erhält man, wenn ein paralleler Algorithmus auf einem Einprozessorsystem ausgeführt wird.
- Zusatzaufwand für Parallelisierung durch Kommunikation und Synchronisation „verfälscht“ bei der sequentiellen Ausführung

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## 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

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

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

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## 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

Karlsruher Institut für Technologie

## Auslastung (Utilization)

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

Gibt an wieviel Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausführt. Entspricht dem normierten Parallelindex.

## Folgerungen

- 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:

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

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## Auslastung (Utilization)

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

Gibt an wieviel Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausführt. Entspricht dem normierten Parallelindex.

## Folgerungen

- 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:

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

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## Superlinearer Speed-Up

$$S(n) > n \qquad T(n) < \frac{T(1)}{n}$$

Beispiele:

- **Paralleles Backtracking (depth-first search)**
  $\Rightarrow$ Verwendung verschiedener Algorithmen
- **Cache- und Hauptspeicherausnutzung**
  Nach der Aufteilung eines Problems auf verschiedene Teilsysteme, können eventuell jeweils die Teilprogramme und -daten komplett im Hauptspeicher bzw. Cache der einzelnen Knoten gehalten werden.
  $\Rightarrow$ kein Seitenwechsel mehr nötig

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## Skalierbarkeit eines Parallelrechners

- Von Skalierbarkeit spricht man, wenn das Hinzufügen von weiteren Verarbeitungselementen zu einer kürzeren Gesamtausführungszeit führt, ohne dass das Programm geändert werden muß.
- lineare Steigerung der Beschleunigung, Effizienz nahe 1
- Wichtig für die Skalierbarkeit ist eine angemessene Problemgröße, da sonst ab einer bestimmten Prozessorenzahl eine Sättigung auftritt

- **Achtung:** Nicht verwechseln mit der Skalierbarkeit eines Verbindungsnetzes!

# Parallelismus: Quantitative Maßzahlen

## Amdahls Gesetz (Amdahl's Law)

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

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

1. Ausführungszeit des parallel ausführbaren Programmteils $(1 - a)$
2. Ausführungszeit des nur sequentiell ausführbaren Programmteils $a$.

# Parallelismus: Quantitative Maßzahlen

## 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:

$$\begin{aligned} S(n) &= \frac{T(1)}{T(n)} = \frac{T(1)}{\frac{T(1)}{n} * (1 - a) + T(1) * a} \\ &= \frac{1}{(1 - a) * \frac{1}{n} + a} \\ S(n) &\leq \frac{1}{a} \end{aligned}$$

$\Rightarrow$ Die erreichbare Beschleunigung wird durch den sequentiellen Programmteil begrenzt.

# Parallelismus: Quantitative Maßzahlen

Karlsruher Institut für Technologie

## Achtung!

**Alle auch hier nicht wiederholten Inhalte der Vorlesung sind für die Klausur relevant!**

# Aufgabe 1.1 – Leistungsbewertung

Karlsruher Institut für Technologie

Gegeben sei ein Multiprozessorsystem mit 16 Prozessoren. Die Leistungssteigerung gegenüber einem Einprozessorsystem sei $S(16) = 8$. Die Ausführungszeit auf dem Einprozessorsystem sei $T(1) = 800$ und die Anzahl der auszuführenden Einheitsoperationen auf dem Multiprozessorsystem sei $P(16) = 1200$.

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

b) Interpretieren Sie den berechneten Parallelindex.

c) Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil des Programms, der nur sequentiell ausführbar ist.

# Aufgabe 1.1 – Leistungsbewertung

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

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

$$S(n) = \frac{T(1)}{T(n)} \quad \Rightarrow \quad T(16) = \frac{T(1)}{S(16)} = \frac{800}{8} = 100$$

$$I(16) = \frac{P(n)}{T(n)} \quad \Rightarrow \quad I(16) = \frac{P(16)}{T(16)} = \frac{1200}{100} = 12$$

# Aufgabe 1.1 – Leistungsbewertung

Karlsruher Institut für Technologie

## b) Interpretieren Sie den berechneten Parallelindex

- Der Parallelindex gibt den mittleren Grad der Parallelität an, d.h. die Anzahl der parallelen Operationen pro Zeiteinheit.
- Der berechnete Wert ist kleiner als die Anzahl der Prozessoren. Das bedeutet, dass zeitweise einige Prozessoren keinen Befehl ausführen.
- Besser verdeutlicht wird dies durch Berechnung der Auslastung $U$, die angibt wieviel Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausführt.

$$U(n) = \frac{I(n)}{n} \quad \Rightarrow \quad U(16) = \frac{12}{16} = \frac{3}{4} = 75\,\%$$

# Aufgabe 1.1 – Leistungsbewertung

Karlsruher Institut für Technologie

## b) Interpretieren Sie den berechneten Parallelindex

- Der Parallelindex gibt den mittleren Grad der Parallelität an, d.h. die Anzahl der parallelen Operationen pro Zeiteinheit.
- Der berechnete Wert ist kleiner als die Anzahl der Prozessoren. Das bedeutet, dass zeitweise einige Prozessoren keinen Befehl ausführen.
- Besser verdeutlicht wird dies durch Berechnung der Auslastung $U$, die angibt wieviel Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausführt.

$$U(n) = \frac{I(n)}{n} \quad \Rightarrow \quad U(16) = \frac{12}{16} = \frac{3}{4} = 75\,\%$$

# Aufgabe 1.1 – Leistungsbewertung

c) **Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil des Programms, der nur sequentiell ausführbar ist.**

Ahmdahls Gesetz:

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

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

# Aufgabe 1.1 – Leistungsbewertung

Karlsruher Institut für Technologie

c) **Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil des Programms, der nur sequentiell ausführbar ist.**

Ahmdahls Gesetz:

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

$$\Rightarrow \quad 100 = 800 * \frac{1 - a + 16a}{16} = 800 * \frac{1 + 15a}{16}$$
$$= 50 * (1 + 15a)$$

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

# Aufgabe 1.1 – Leistungsbewertung

Karlsruher Institut für Technologie

c) **Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil des Programms, der nur sequentiell ausführbar ist.**

Ahmdahls Gesetz:

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

$$\Rightarrow \quad 100 = 800 * \frac{1 - a + 16a}{16} = 800 * \frac{1 + 15a}{16}$$

$$= 50 * (1 + 15a)$$

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

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

# Aufgabe 1.2 – Leistungsbewertung

Karlsruher Institut für Technologie

Die Ausführungszeit einer sequentiellen Anwendung betrage $T_{seq}$. Von dieser Anwendung lassen sich 20 % nicht parallelisieren. Die verbleibenden 80 % werden zwischen den Prozessoren gleichmäßig verteilt. Das 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.

a) 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 vorgegebene Tabelle ein.

# Aufgabe 1.2 – Leistungsbewertung

Karlsruher Institut für Technologie

a) **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 vorgegebene Tabelle ein.**

Par. Ausführungszeit: $T(n) = \frac{80\,\%}{n} * T_{seq} + 20\,\% * T_{seq}$

Beschleunigung: $S(n) = \frac{T_{seq}}{T(n)}$

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

| Prozessoren | n=2 | n=4 | n=8 | n=16 | n=32 |
|---|---|---|---|---|---|
| Beschleunigung | 5/3=1,67 | 5/2=2,50 | 10/3=3,33 | 4 | 40/9=4,44 |
| Effizienz | 5/6=0,83 | 5/8=0,625 | 10/24=0,42 | 1/4=0,25 | 5/36=0,14 |

**Achtung:** In der Klausur bei Berechnungen wenn möglich Brüche statt gerundete Zahlen als Ergebnisse angeben!

# Aufgabe 1.2 – Leistungsbewertung

Karlsruher Institut für Technologie

a) **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 vorgegebene Tabelle ein.**

Par. Ausführungszeit: $T(n) = \frac{80\,\%}{n} * T_{seq} + 20\,\% * T_{seq}$

Beschleunigung: $S(n) = \frac{T_{seq}}{T(n)}$

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

| Prozessoren | n=2 | n=4 | n=8 | n=16 | n=32 |
|---|---|---|---|---|---|
| Beschleunigung | 5/3=1,67 | 5/2=2,50 | 10/3=3,33 | 4 | 40/9=4,44 |
| Effizienz | 5/6=0,83 | 5/8=0,625 | 10/24=0,42 | 1/4=0,25 | 5/36=0,14 |

**Achtung:** In der Klausur bei Berechnungen wenn möglich Brüche statt gerundete Zahlen als Ergebnisse angeben!

# Aufgabe 1.2 – Leistungsbewertung

Karlsruher Institut für Technologie

| Prozessoren | n=2 | n=4 | n=8 | n=16 | n=32 |
|---|---|---|---|---|---|
| Beschleunigung | 5/3=1,67 | 5/2=2,50 | 10/3=3,33 | 4 | 40/9=4,44 |
| Effizienz | 5/6=0,83 | 5/8=0,625 | 10/24=0,42 | 1/4=0,25 | 5/36=0,14 |

b) **Evaluieren Sie zusätzlich die Skalierbarkeit der einzelnen Lösungen**

Die Skalierbarkeit ist sehr schlecht. Grund dafür ist, dass ein großer Anteil des Programms nicht parallelisierbar ist. Die Ausführungszeit dieses Anteils bestimmt mit steigender Anzahl von Prozessoren einen zunehmenden Anteil der gesamten Ausführungszeit.

Beschleunigung: $S(n) = \frac{T_{seq}}{(20\,\% + \frac{80\,\%}{n}) * T_{seq}} \xrightarrow{n \text{ sehr groß}} \frac{1}{20\,\%} = 5$

Die Effizienz strebt damit für große $n$ gegen 0.

# Aufgabe 1.2 – Leistungsbewertung

| Prozessoren | n=2 | n=4 | n=8 | n=16 | n=32 |
|---|---|---|---|---|---|
| Beschleunigung | 5/3=1,67 | 5/2=2,50 | 10/3=3,33 | 4 | 40/9=4,44 |
| Effizienz | 5/6=0,83 | 5/8=0,625 | 10/24=0,42 | 1/4=0,25 | 5/36=0,14 |

b) **Evaluieren Sie zusätzlich die Skalierbarkeit der einzelnen Lösungen**

Die Skalierbarkeit ist sehr schlecht. Grund dafür ist, dass ein großer Anteil des Programms nicht parallelisierbar ist. Die Ausführungszeit dieses Anteils bestimmt mit steigender Anzahl von Prozessoren einen zunehmenden Anteil der gesamten Ausführungszeit.

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

Die Effizienz strebt damit für große $n$ gegen 0.

# Aufgabe 1.2 – Leistungsbewertung

Karlsruher Institut für Technologie

c) **Zur Steigerung der Genauigkeit der Berechnung nehmen Sie nun an, dass durch jeden verwendeten Prozessor eine zusätzliche Bearbeitungszeit von 1 % der ursprünglichen sequentiellen Ausführungszeit benötigt wird. Berechnen Sie Beschleunigung und Effizienz für 64 Prozessoren.**

Beschleunigung:

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

$$\Rightarrow \quad S(64) \approx 1,17$$

Effizienz:

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

# Aufgabe 1.2 – Leistungsbewertung

c) **Zur Steigerung der Genauigkeit der Berechnung nehmen Sie nun an, dass durch jeden verwendeten Prozessor eine zusätzliche Bearbeitungszeit von 1 % der ursprünglichen sequentiellen Ausführungszeit benötigt wird. Berechnen Sie Beschleunigung und Effizienz für 64 Prozessoren.**

Beschleunigung:

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

$$\Rightarrow \quad S(64) \approx \quad 1,17$$

Effizienz:

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

# Aufgabe 1.3 – Leistungsbewertung

Karlsruher Institut für Technologie

Ein Einprozessorsystem soll erweitert werden. Dabei existieren folgende, in der Anschaffung gleich teure Alternativen:

- Ausbau zu einem 2-fach SMP-System
  - Installation eines zweiten identischen Hauptprozessors
  - Zugriff auf einen gemeinsamen Hauptspeicher
  - Synchronisationsoverhead: Ausführung des Programms wird um 2 % der unparallelisierten Ausführungszeit erhöht.
- Einsetzen eines mathematischen Koprozessors
  - 10x schnellere Ausführung der Gleitkommaarithmetik
  - Keine parallele Verarbeitung, d.h. der gleichzeitige Einsatz von Haupt- und Koprozessor, möglich.

Das zu bearbeitende Problem ist zu 74 % parallelisierbar. Der Anteil der Gleitkommaarithmetik am Gesamtprogramm beträgt 40 %. Bestimmen Sie, welche der beiden Möglichkeiten unter dem Gesichtspunkt der Ausführungszeit zu bevorzugen ist.

# Aufgabe 1.3 – Leistungsbewertung

Karlsruher Institut für Technologie

Die Ausführungszeiten lassen sich in beiden Fällen wiederum mit Hilfe von Ahmdahls Gesetz ausrechnen und damit auch die erreichte Beschleunigung vergleichen. $T_{seq}$ ist hierbei die Zeit ohne Parallelisierung oder Koprozessorunterstützung.

- SMP-System:

$$T_{SMP} = 74\,\% * \frac{1}{2} * T_{seq} + 26\,\% * T_{seq} + 2\,\% * T_{seq} = 65\,\% * T_{seq}$$

$$S_{SMP} = \frac{T_{seq}}{T_{SMP}} = \frac{1}{65\,\%} \approx 1,5385$$

- System mit Koprozessor:

$$T_{CP} = 40\,\% * \frac{1}{10} * T_{seq} + 60\,\% * T_{seq} = 64\,\% * T_{seq}$$

$$S_{CP} = \frac{T_{seq}}{T_{CP}} = \frac{1}{64\,\%} \approx 1,5625$$

# Aufgabe 1.3 – Leistungsbewertung

Karlsruher Institut für Technologie

Die Ausführungszeiten lassen sich in beiden Fällen wiederum mit Hilfe von Ahmdahls Gesetz ausrechnen und damit auch die erreichte Beschleunigung vergleichen. $T_{seq}$ ist hierbei die Zeit ohne Parallelisierung oder Koprozessorunterstützung.

- SMP-System:

$$T_{SMP} = 74\,\% * \frac{1}{2} * T_{seq} + 26\,\% * T_{seq} + 2\,\% * T_{seq} = 65\,\% * T_{seq}$$

$$S_{SMP} = \frac{T_{seq}}{T_{SMP}} = \frac{1}{65\,\%} \approx 1,5385$$

- System mit Koprozessor:

$$T_{CP} = 40\,\% * \frac{1}{10} * T_{seq} + 60\,\% * T_{seq} = 64\,\% * T_{seq}$$

$$S_{CP} = \frac{T_{seq}}{T_{CP}} = \frac{1}{64\,\%} \approx 1,5625$$

# Aufgabe 1.3 – Leistungsbewertung

Karlsruher Institut für Technologie

Die Ausführungszeiten lassen sich in beiden Fällen wiederum mit Hilfe von Ahmdahls Gesetz ausrechnen und damit auch die erreichte Beschleunigung vergleichen. $T_{seq}$ ist hierbei die Zeit ohne Parallelisierung oder Koprozessorunterstützung.

- SMP-System:

$$T_{SMP} = 74\,\% * \frac{1}{2} * T_{seq} + 26\,\% * T_{seq} + 2\,\% * T_{seq} = 65\,\% * T_{seq}$$

$$S_{SMP} = \frac{T_{seq}}{T_{SMP}} = \frac{1}{65\,\%} \approx 1,5385$$

- System mit Koprozessor:

$$T_{CP} = 40\,\% * \frac{1}{10} * T_{seq} + 60\,\% * T_{seq} = 64\,\% * T_{seq}$$

$$S_{CP} = \frac{T_{seq}}{T_{CP}} = \frac{1}{64\,\%} \approx 1,5625$$

# Aufgabe 1.3 – Leistungsbewertung

- SMP-System:

$$S_{SMP} \approx 1,5385$$

- System mit Koprozessor:

$$S_{CP} \approx 1,5625$$

$\Rightarrow$ Die Koprozessorlösung ist für diese Anwendung deshalb dem SMP-System vorzuziehen.

$\Rightarrow$ Ohne Beachtung der benötigten Synchronisationszeit würde die Berechnung ein fehlerhaftes Ergebnis liefern.

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Parallelisierungsprozess

- **Ziel: Schnellere Lösung der parallelen Version gegenüber der sequentiellen Version**

- Festlegen der Aufgaben, die parallel ausgeführt werden können
- Aufteilen der Aufgaben und der Daten auf Verarbeitungsknoten
  - Berechnung
  - Datenzugriff
  - Ein-/Ausgabe
- Verwalten des Datenzugriffs, der Kommunikation und Synchronisation

- Programmierer oder Automatische Parallelisierung

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Parallelisierungsprozess – Definitionen

- **Tasks**
  - Kleinste Parallelisierungseinheit
  - Beispiel: Berechnung eines Gitterpunkts (oder einer Teilmenge von Punkten)
  - grobkörnig vs. feinkörnig
- **Prozess** oder Thread
  - Paralleles Programm setzt sich aus mehreren kooperierenden Prozessen zusammen, von denen jeder eine Teilmenge der Tasks ausführt
  - Kommunikation und Synchronisation der Prozesse untereinander
  - Virtualisierung von einem Multiprozessor, Abstraktion

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Parallelisierungsprozess – Definitionen

- **Tasks**
  - Kleinste Parallelisierungseinheit
  - Beispiel: Berechnung eines Gitterpunkts (oder einer Teilmenge von Punkten)
  - grobkörnig vs. feinkörnig
- **Prozess** oder Thread
  - Paralleles Programm setzt sich aus mehreren kooperierenden Prozessen zusammen, von denen jeder eine Teilmenge der Tasks ausführt
  - Kommunikation und Synchronisation der Prozesse untereinander
  - Virtualisierung von einem Multiprozessor, Abstraktion

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Parallelisierungsprozess – Definitionen

- **Prozessor**
  - Ausführung eines oder mehrerer Prozesse
  - Physikalische Ressource

- **Unterscheidung zwischen Prozess und Prozessor!**

$\Rightarrow$ Anzahl der Prozesse muss nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Parallelisierungsprozess – Definitionen

- **Prozessor**
  - Ausführung eines oder mehrerer Prozesse
  - Physikalische Ressource

- **Unterscheidung zwischen Prozess und Prozessor!**

$\Rightarrow$ Anzahl der Prozesse muss nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Programmiermodelle

- Definition einer abstrakten parallelen Maschine
- Spezifikationen
  - Parallele Abarbeitung von Teilen des Programms
  - Informationsaustausch
  - Synchronisationsoperationen zur Koordination
- Anwendungen werden auf der Grundlage eines parallelen Programmiermodells formuliert

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Multiprogramming

- Menge von unabhängigen sequentiellen Programmen
- Keine Kommunikation oder Koordination
- **Aber: Mögliche gegenseitige Beeinflussung beim gleichzeitigen Zugriff auf den Speicher!**

**Verwendung:**

- Auf allen Rechnern, die „gleichzeitig“ verschiedene Programme ausführen können (multitasking-fähiges Betriebssystem)

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Gemeinsamer Speicher (Shared Memory)

- Kommunikation und Koordination von Prozessen über gemeinsame Variablen
- Atomare Synchronisationsoperationen
- Semaphoren, Mutex, Monitore, Transactional Memory,...

**Verwendung:**

- Symmetrischer Multiprozessor (SMP)
- Distributed-shared-memory Multiprozessor (DSM)

**Speicherzugriff:**

- Uniform Memory Access (UMA)
- Non-Uniform Memory Access (NUMA), CC-NUMA

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Message Passing

- Nachrichtenorientiertes Programmiermodell
- Kein gemeinsamer Adressraum
- Kommunikation der Prozesse mit Hilfe von Nachrichten
- Verwendung von korrespondierenden Send- und Receive-Operationen

**Verwendung:**

- Cluster
- Nachrichtengekoppelter (shared-nothing-) Multiprozessor

**Speicherzugriff:**

- No Remote Memory Access (NORMA)

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Shared Memory und Message Passing

- Mischung der Programmiermodelle
- **Cluster mit SMP-/DSM-Knoten** (Multicore-CPUs oder Multiprozessor-System mit gemeinsamem Speicher)
- Shared-Memory-Programmiermodell innerhalb eines Knotens
- Nachrichtenorientiertes Programmiermodell zwischen den Knoten

# Parall. Programm. – Programmiermodelle

Karlsruher Institut für Technologie

## Datenparallelismus

- Gleichzeitige Ausführung von Operationen auf getrennten Elementen einer Datenmenge (Feld, Vektor, Matrix)

**Verwendung:**

- Typischerweise in Vektorrechnern

$\Rightarrow$ Übung 8

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

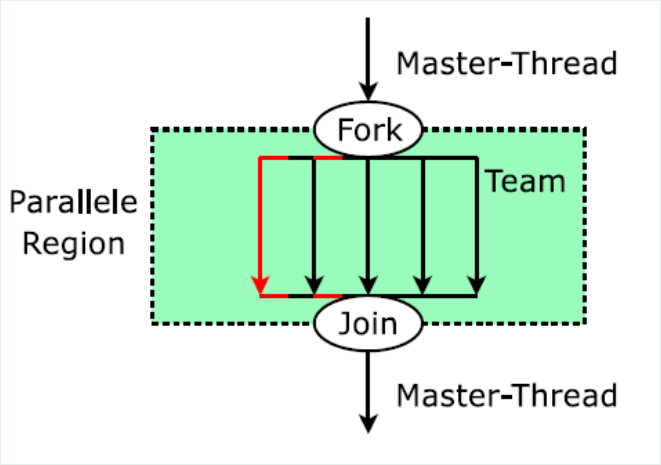
## Thread-Programmierung

- Parallele Programme für Shared-Memory-Systeme bestehen aus mehreren Threads (i.d.R. 1 Thread pro Kern)
- Alle Threads eines Prozesses teilen sich Adressraum, Daten, Dateideskriptoren,. . .
- Threads werden meist vom Betriebsystem verwaltet
- Manuelle Synchronisation
- Direkte Thread-Programmierung z.B. mittels pthreads vergleichsweise aufwändig

# Parallele Programmierung – Einführung

## OpenMP

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



# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## OpenMP – Beispiele

- Bibliothek und Funktionen:
  - `#include <omp.h>`
  - `omp_set_num_threads(NUM_THREADS)`
  - `omp_get_num_threads()`
  - `omp_get_max_threads()`
- Pragmas:
  - `#pragma omp parallel`
  - `#pragma omp parallel for`
  - `#pragma omp parallel for private(x) shared(delta_x) reduction(+:sum)`
  - `#pragma omp critical`
  - `#pragma omp atomic`

# Parallele Programmierung – $\pi$-Berechnung

Karlsruher Institut für Technologie

## 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;
}
```

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## 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 Program Multiple Data (SPMD) Ansatz

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Message Passing Interface (MPI)

- Bibliothek und Funktionen:
  - `#include <mpi.h>`
  - Initialisierung

    `MPI_Init(&argc, &argv)`
  - `MPI_COMM_WORLD`

    beinhaltet alle am Programm beteiligten Prozesse
  - Prozessnummer abfragen:

    `MPI_Comm_rank(MPI_Comm comm, int *rank)`

    `MPI_Comm_rank(MPI_COMM_WORLD, &rank)`
  - Anzahl an Prozesse:

    `MPI_Comm_size(MPI_Comm comm, int *size)`

    `MPI_Comm_size(MPI_COMM_WORLD, &size)`
  - `MPI_Finalize()`

## Message Passing Interface (MPI)

- Kommunikation und Synchronisation:
  - Senden:
    `MPI_Send`
  - Senden (nicht blockierend):
    `MPI_Isend`
  - Gepuffert:
    `MPI_Bsend, MPI_Ibsend`
  - ready/synchron:
    `MPI_Rsend, MPI_Ssend, MPI_Irsend, MPI_Issend`
  - Empfangen:
    `MPI_Recv, MPI_Irecv`
  - `MPI_Sendrecv`

# Parallele Programmierung – Einführung

Karlsruher Institut für Technologie

## Message Passing Interface (MPI)

- Kommunikation und Synchronisation:
  - Warten auf alle Prozesse in `comm`:
    `MPI_Barrier(comm)`
  - Verteilen von Daten:
    `MPI_Bcast, MPI_Scatter`
  - Sammeln von Daten:
    `MPI_Gather, MPI_Reduce`
  - Empfangstests einer Sende- oder Empfangsoperation:
    `MPI_Probe, MPI_Test, MPI_Wait`

# Parallele Programmierung – $\pi$-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 2 – Parallelisierung

Karlsruher Institut für Technologie

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

In jeder Iteration werden drei Schritte durchgeführt:

1. Zustände aus dem Speicher lesen
2. Berechnung der neuen Zustände der Knoten, basierend auf ihrem aktuellen Zustand und den Zuständen ihrer Nachbarknoten
3. Zurückschreiben der neuen Zustände in den Speicher

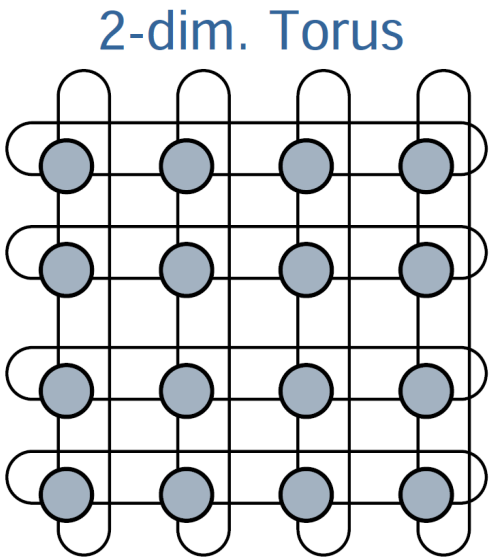
# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

Gegeben sei ein SMP-Knoten mit $P$ Prozessoren.

a) Beachten Sie, dass in diesem System nur immer ein Lese- oder Schreibzugriff ausgeführt werden kann. Setzen Sie zusätzlich voraus, dass sich die Lese- und Schreibzugriffe nicht mit den Berechnungen überlappen (z.B. wegen einer vorherigen Synchronisation).
**Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt in dem SMP-System auf?**

# Aufgabe 2 – Parallelisierung




SMP gemeinsamer Adressraum, globaler Speicher

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

DSM gemeinsamer Adressraum, physikalisch verteilter Speicher

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

# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

a) **Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt in dem SMP-System 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 $S(P)$:

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

# Aufgabe 2 – Parallelisierung

a) **Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt in dem SMP-System 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 $S(P)$:

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

# Aufgabe 2 – Parallelisierung

a) **Berechnen Sie die Beschleunigung der Anwendung bei der Ausführung auf dem SMP-Knoten. Welches Problem tritt in dem SMP-System 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 $S(P)$:

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

# Aufgabe 2 – Parallelisierung

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

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

In der Realität benötigten Synchronisationen jedoch in SMP-Systemen ebenfalls Speicherzugriffe von allen Prozessoren und damit sehr viel Zeit! Die erreichbare Beschleunigung wäre deshalb noch geringer.

# Aufgabe 2 – Parallelisierung

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

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

In der Realität benötigten Synchronisationen jedoch in SMP-Systemen ebenfalls Speicherzugriffe von allen Prozessoren und damit sehr viel Zeit! Die erreichbare Beschleunigung wäre deshalb noch geringer.

# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

b) **Was ändert sich, wenn auf eine Synchronisation vor der Berechnung verzichtet wird?**

Durch ein Verzicht auf die Synchronisation kann die Berechnung auf den Prozessoren mit den Datenzugriffen überlappend erfolgen.

Bei der parallelen Ausführungszeit spart man sich dadurch die Zeit für die Berechnung $\frac{n^2}{P}$ ein. Dies gilt aber nur, wenn die Berechnungszeit kürzer ist, als die Zeit für die Datenzugriffe.

# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

b) **Was ändert sich, wenn auf eine Synchronisation vor der Berechnung verzichtet wird?**

Es gilt dann:

$$T(P) = 5n^2 + n^2 = 6n^2$$

$$S(P) = \frac{T(1)}{T(P)} = \frac{7n^2}{6n^2} = \frac{7}{6} = 1,1\overline{6}$$

Insgesamt überwiegt auch hier die Zeit für die nicht parallelisierbaren Lese- und Schreibzugriffe. Der gemeinsame Speicher des SMP-Systems ist der limitierende Faktor.

# Aufgabe 2 – Parallelisierung

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 drei Zeiteinheiten.
- Wird nur ein Prozessor verwendet, so befinden sich alle zur Berechnung notwendigen Daten im lokal zum Prozessor gehörenden Speicher.

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

# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

c) **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 $T(P)$:

$$\underbrace{\underbrace{\frac{4 * 3 * n^2}{P}}_{\substack{\text{4 Werte der Nachbarknoten aus entferntem Speicher}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{eigener Wert im lokalen Speicher}}}}_{\text{Daten aus Speicher holen}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{parallele Berechnung}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{Zurückschreiben in lokalen Speicher}}} = \frac{15n^2}{P}$$

# Aufgabe 2 – Parallelisierung

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

- Sequentielle Zeit $T(1)$:

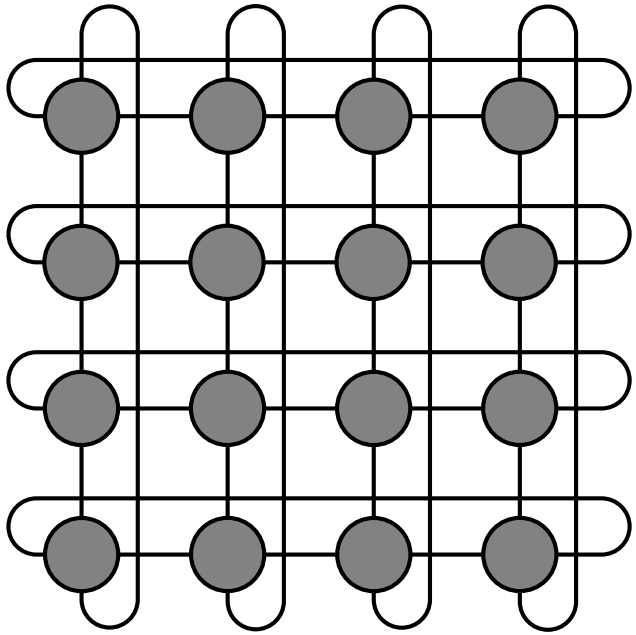
$$\underbrace{5n^2}_{\text{Daten aus Speicher holen}} + \underbrace{n^2}_{\text{Berechnung}} + \underbrace{n^2}_{\text{Zurückschreiben}} = 7n^2$$

- Parallele Zeit $T(P)$:

$$\underbrace{\underbrace{\frac{4 * 3 * n^2}{P}}_{\substack{\text{4 Werte der Nachbarknoten aus entferntem Speicher}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{eigener Wert im lokalen Speicher}}}}_{\text{Daten aus Speicher holen}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{parallele Berechnung}}} + \underbrace{\frac{n^2}{P}}_{\substack{\text{Zurückschreiben in lokalen Speicher}}} = \frac{15n^2}{P}$$

# Aufgabe 2 – Parallelisierung

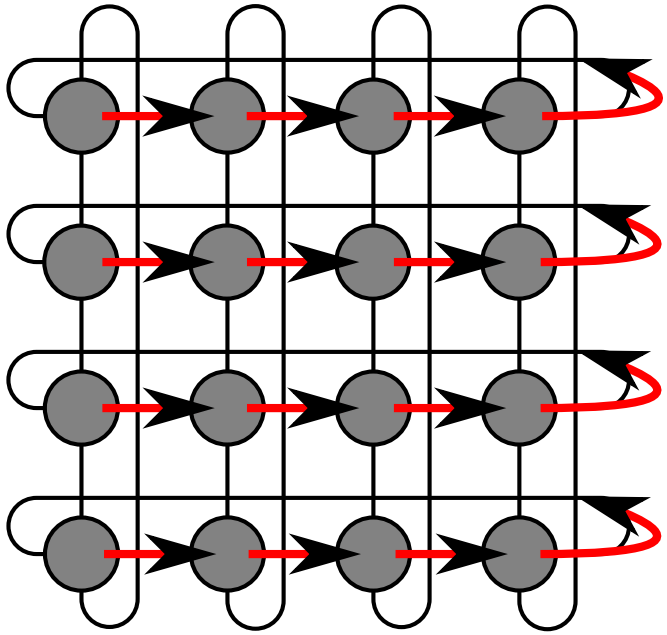
c) **Welche Beschleunigung läßt sich erzielen, wenn die Zustände der Nachbarknoten immer aus entfernten Speichern abgerufen werden müssen?**



- Parallele Zeit $T(P)$ für $n^2 = P$:

# Aufgabe 2 – Parallelisierung

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

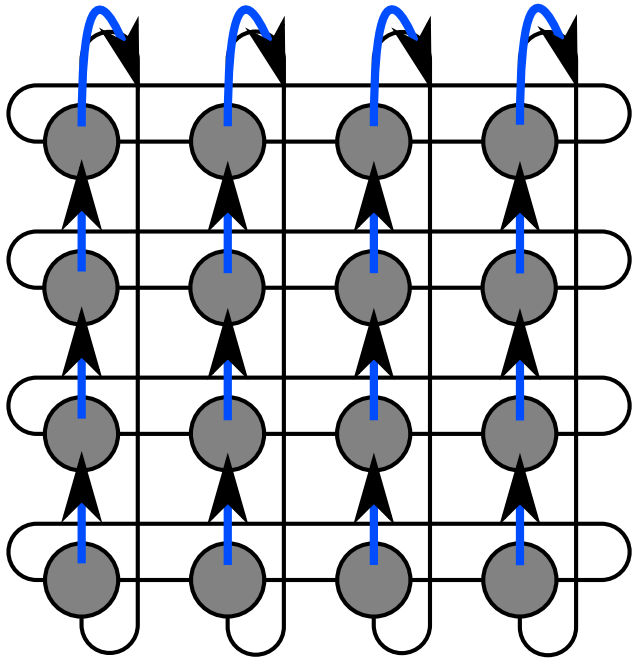


- Parallele Zeit $T(P)$ für $n^2 = P$:

$$3$$

# Aufgabe 2 – Parallelisierung

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

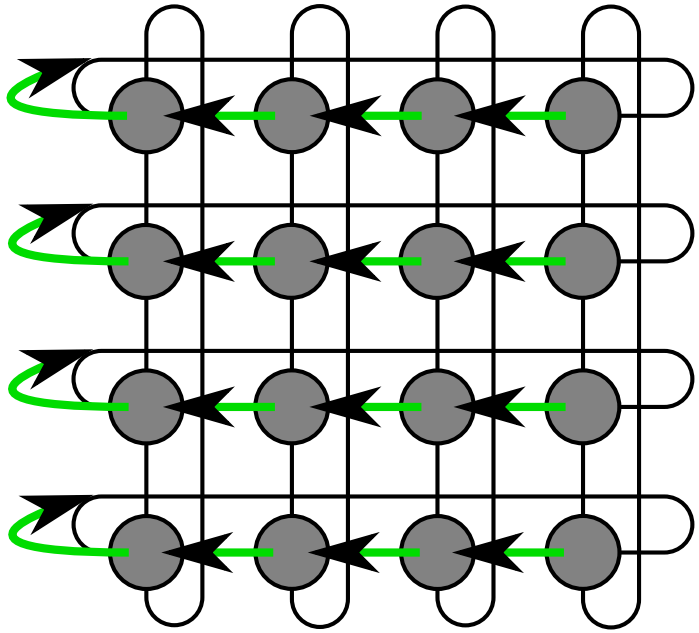


- Parallele Zeit $T(P)$ für $n^2 = P$:

$$3 + 3$$

# Aufgabe 2 – Parallelisierung

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

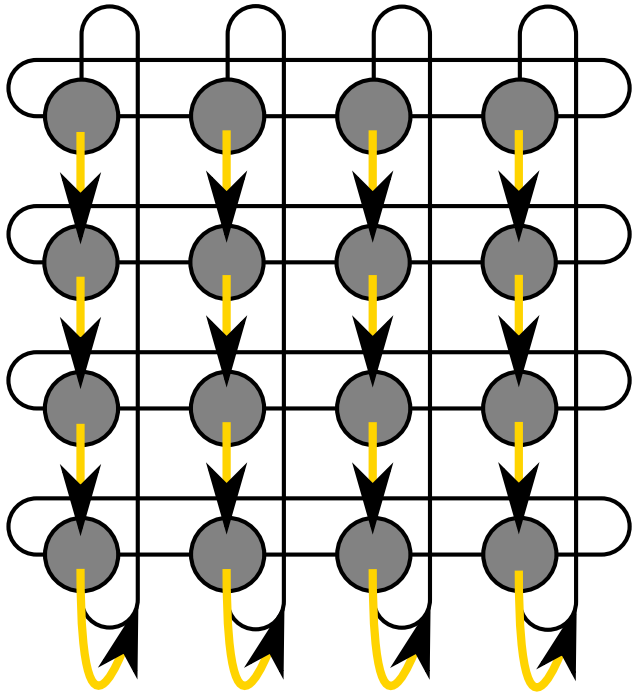


- Parallele Zeit $T(P)$ für $n^2 = P$:

$$3 + 3 + 3$$

# Aufgabe 2 – Parallelisierung

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

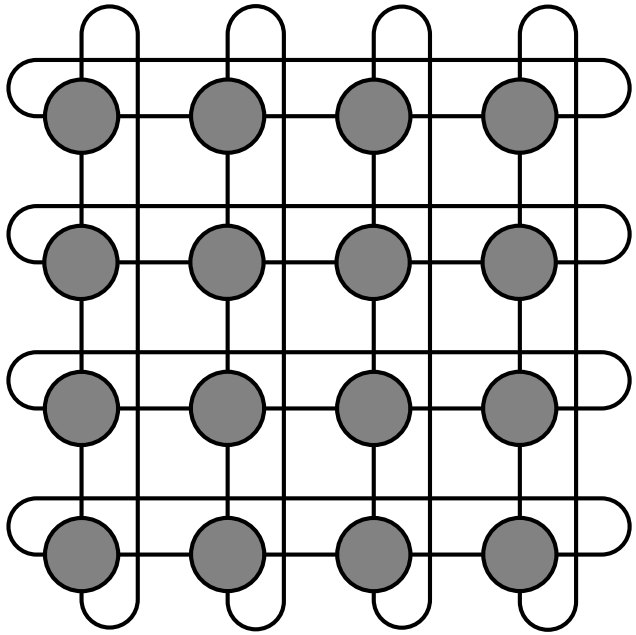


- Parallele Zeit $T(P)$ für $n^2 = P$:

$$3 + 3 + 3 + 3$$

# Aufgabe 2 – Parallelisierung

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

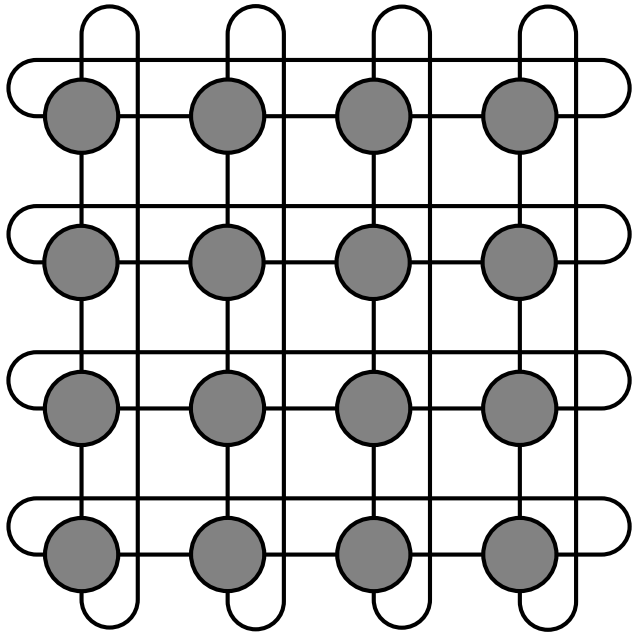


- Parallele Zeit $T(P)$ für $n^2 = P$:

$$3 + 3 + 3 + 3 = 4 * 3$$

# Aufgabe 2 – Parallelisierung

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



- Parallele Zeit $T(P)$ für $n^2 = P$:

$$T(P) = 4 * 3 + 1 + 1 + 1 = 15$$

# Aufgabe 2 – Parallelisierung

Karlsruher Institut für Technologie

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

- Sequentielle Zeit $T(1) = 7n^2$
- Parallele Zeit $T(P) = \frac{15n^2}{P}$
- Beschleunigung $S(P)$:

$$S(P) = \frac{T(1)}{T(P)} = \frac{7n^2}{\frac{15n^2}{P}} = \frac{7}{15}P$$

$$\text{für } P = 2: \quad S(2) = \frac{7}{15} * 2 = \frac{14}{15} \approx 0,93$$

$$\text{für } P = 3: \quad S(3) = \frac{7}{15} * 3 = \frac{21}{15} \approx 1,40$$

$$\text{für } P = 4: \quad S(4) = \frac{7}{15} * 4 = \frac{28}{15} \approx 1,87$$

$\Rightarrow$ Die Beschleunigung skaliert mit $\frac{7}{15}$ der Prozessorenzahl (linear)

Karlsruher Institut für Technologie

# Zentralübung Rechnerstrukturen im SS 2015

## Parallelismus und Parallele Programmierung

Mario Kicherer, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

10. Juni 2015