Rechnerstrukturen

Vorlesung im Sommersemester 2006

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

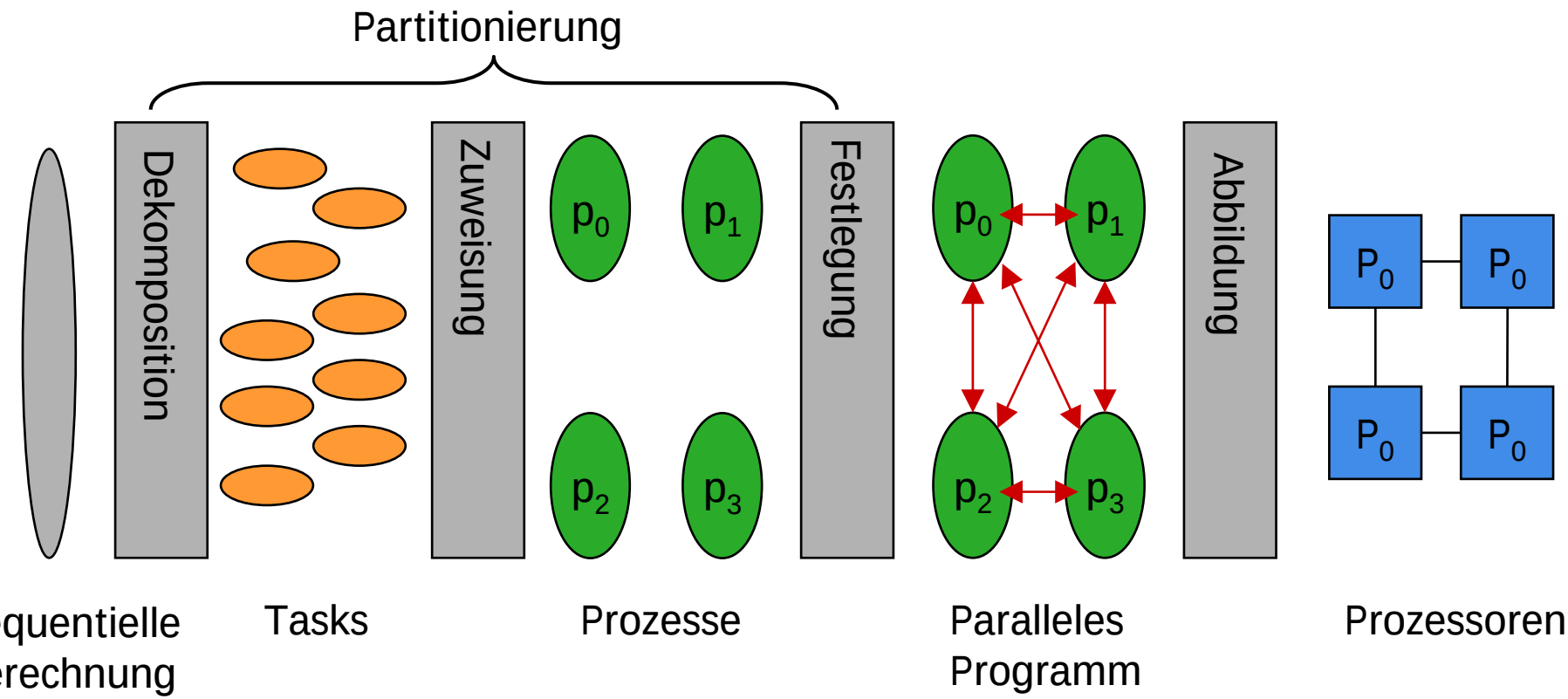
Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.2: Parallele Programmierung

- Parallelisierungsprozess
  - Schritte bei der Parallelisierung und die Beziehung zwischen Tasks, Prozessen und Prozessoren



- Parallelisierungsprozess
  - Zuweisung
    - Spezifikation des Mechanismus, mit dessen Hilfe die Tasks auf Prozesse aufgeteilt werden
    - Ziel:
      - ausgewogene Lastverteilung: Lastbalanzierung
      - Reduzierung der Interprozess-Kommunikation
      - Reduzierung des Aufwands zur Laufzeit
    - Statische oder dynamische Zuweisung

- Parallelisierungsprozess
  - Festlegung
    - Architektur und Programmiermodell sowie die Programmiersprache spielen eine Rolle

      - Um die zugewiesenen Tasks ausführen zu können, benötigen die Prozesse Mechanismen
        - » für den Zugriff auf die Daten,
        - » für die Kommunikation (Austausch von Daten)
        - » für die Synchronisation untereinander
      - Fragen
        - » Organisation der Datenstrukturen
        - » Ablauf der Tasks
        - » Explizite oder implizite Kommunikation
      - Ziel:
        - » Reduzierung des Kommunikations- und Synchronisationsaufwandes (aus der Sicht des Prozessors)
        - » Erhalten der Lokalität der Datenzugriffe, soweit möglich
        - » Reduzierung des Parallelisierungsaufwandes
      - Rechnerarchitekt: bereitstellen effizienter Mechanismen, die die Festlegung vereinfachen

- Parallelisierungsprozess
  - Festlegung
    - Programmiermodelle:
      - Shared-Memory-Programmiermodell
      - Nachrichten-orientiertes Programmiermodell
      - Datenparalleles Programmiermodell

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(p,proc,args)**	Generiere Prozess, der die Ausführung bei der Prozedur **proc** mit den Argumenten **args** startet
G_MALLOC	**G_MALLOC(size)**	Allokation eines gemeinsamen Datenbereichs der Größe **size** Bytes
LOCK	**LOCK(name)**	Fordere wechselseitigen exklusiven Zugriff an
UNLOCK	**UNLOCK(name)**	Freigeben des Locks

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
BARRIER	**BARRIER(name,number)**	Globale Synchronisation für **number** Prozesse
WAIT_FOR_END	**WAIT_FOR_END(number)**	Warten, bis **number** Prozesse terminieren
wait for flag	**while (!flag); or WAIT(flag)**	Warte auf gesetztes flag; entweder wiederholte Abfrage (spin) oder blockiere;
set flag	**flag=1; or SIGNAL(flag)**	Setze flag; weckt Prozess auf, der **flag** wiederholt abfragt

- Parallelisierungsprozess
  - Festlegung
    - Message Passing
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(procedure)**	Erzeuge Prozess, der bei **procedure** startet
SEND	**SEND(src_addr,size, dest,tag)**	Sende **size** Bytes von Adresse **src_addr** an **dest** Prozess mit **tag** Identifier
RECEIVE	**RECEIVE(buffer_addr ,size,src,tag)**	Empfange eine Nachricht mit der Kennung **tag** vom **src**-Prozess und lege **size** Bytes in Puffer bei **buffer_addr** ab
BARRIER	**BARRIER(name,number)**	Globale Synchronisation von **number** Prozessen

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.3: Quantitative Maßzahlen

- Ausführungszeit
  - Die (Gesamt-)Ausführungszeit T (Execution Time) eines parallelen Programms ist die Zeit zwischen dem Starten der Programmausführung auf einem der Prozessoren bis zu dem Zeitpunkt, an dem der letzte Prozessor die Arbeit an dem Programm beendet hat.
  - Zu beachten:
    - Prozessorzustand:
      - Während der Programmausführung sind alle Prozessoren in einem der drei Zustände:
        - » rechnend
        - » kommunizierend
        - » untätig

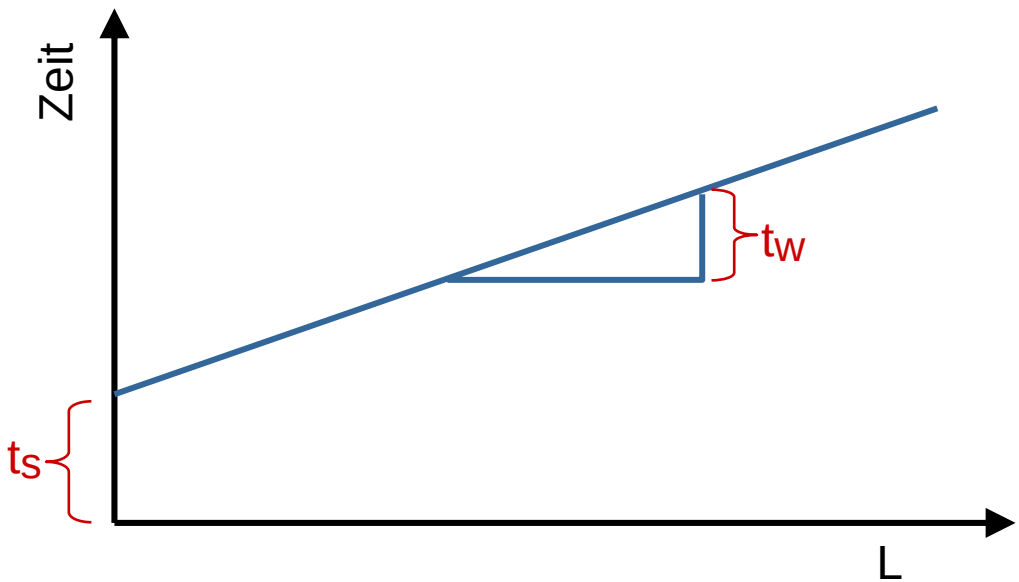
- Ausführungszeit
  - Die Ausführungszeit eines parallelen Programms auf einem dediziert zugeordneten Parallelrechner setzt sich zusammen aus:
    - Berechnungszeit T_{comp} (Computation Time)
      - Die Zeit, die für die Rechenoperationen verwendet wird
    - Kommunikationszeit T_{comm} (Communication Time)
      - Die Zeit, die für Sende- und Empfangsoperationen verwendet wird
    - Untätigkeitszeit T_{idle} (Idle Time)
      - Die Zeit, die mit Warten (auf zu empfangene Nachrichten oder und zu versendende) verbraucht wird
  - Es gilt:
    - $T = T_{comp} + T_{comm} + T_{idle}$

- Ausführungszeit
  - Übertragungszeit einer Nachricht T_{msg}
    - die Zeit, die für das Verschicken einer Nachricht von einer bestimmten Länge zwischen zwei Prozessoren benötigt wird
    - Die Übertragungszeit setzt sich zusammen aus:
      - der Startzeit t_s (Message Startup Time):
        - » Die Zeit, die benötigt wird, um die Kommunikation zu initiieren
      - Transferzeit t_w pro übertragenem Datenwort:
        - » hängt von der physikalischen Bandbreite des Kommunikationsmediums ab.
      - Voraussetzung:
        - » Verbindungsnetz ist konfliktfrei, da sonst die Übertragungszeit nicht fest berechnet werden kann

- Ausführungszeit
  - Übertragungszeit einer Nachricht T_{msg}
    - Formel:
      - $T_{msg} = t_s + t_w * L$,

mit L: Anzahl der Datenwörter

- Graphische Darstellung:

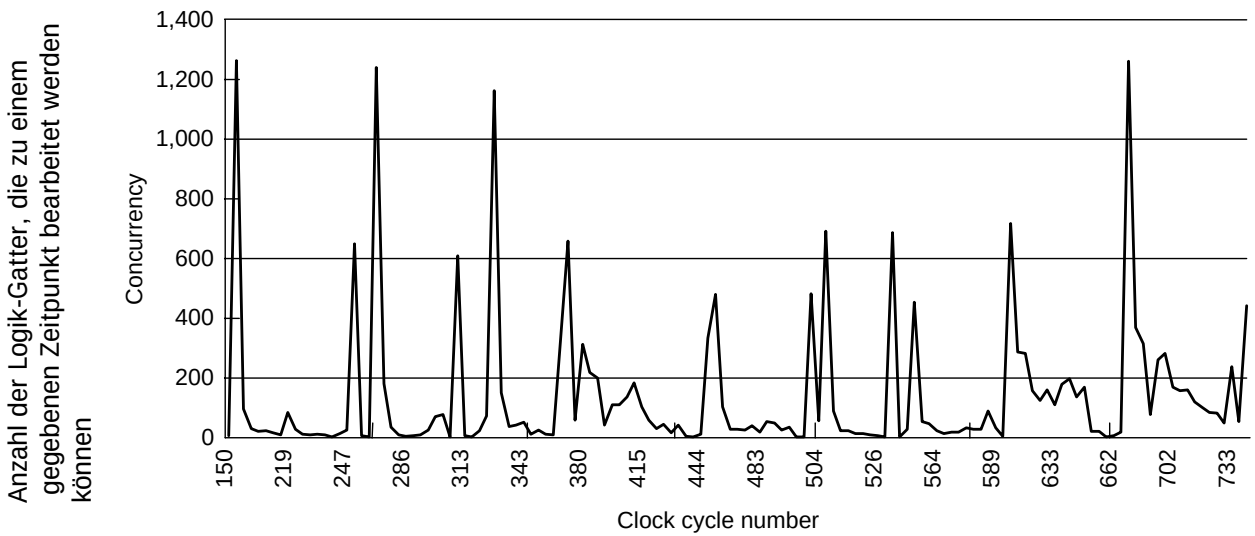


- Parallelitätsprofil
  - misst die entstehende Parallelität in einem parallelen Programm bzw. bei der Ausführung auf einem Parallelrechner.
  - Gibt eine Vorstellung von der inhärenten Parallelität eines Algorithmus/Programms und deren Nutzung auf einem realen oder ideellen Parallelrechner
  - Grafische Darstellung:
    - Auf der x-Achse wird die Zeit und auf der y-Achse die Anzahl paralleler Aktivitäten angetragen.

    ➔Perioden von Berechnungs- Kommunikations- und Untätigkeitszeiten sind erkennbar.

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

Beispiel: Parallele ereignisgesteuerte Simulation der Logik-Synthese



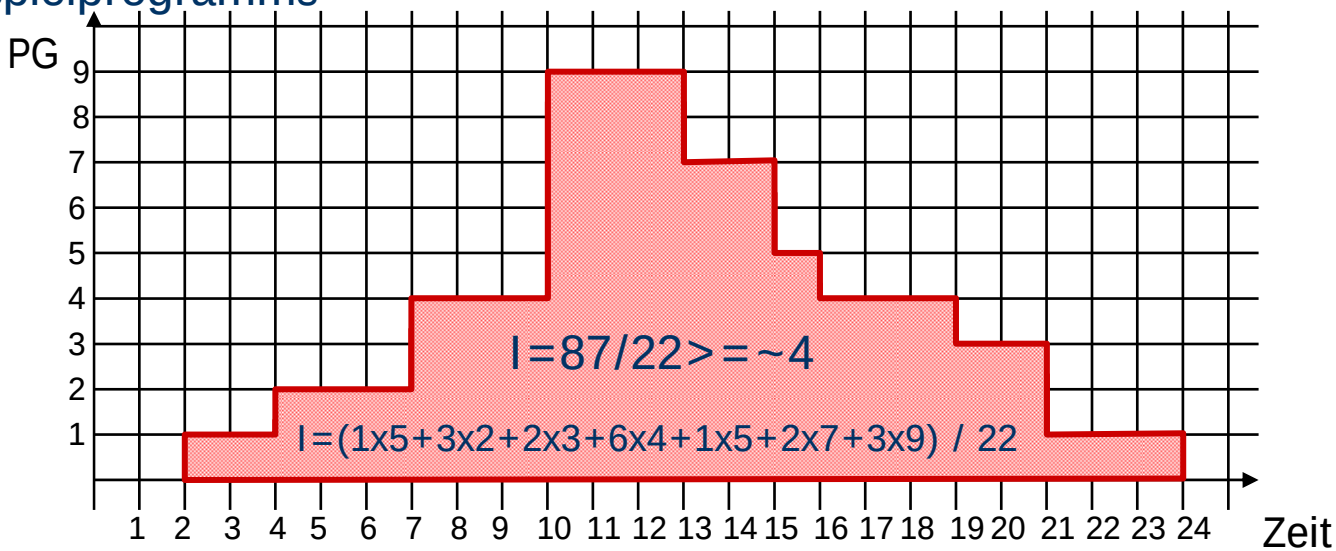
Quelle: D. Culler: Parallel Computer Architecture. Morgan Kaufmann Publishers, 1999, p.87

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

$$I = \frac{1}{t_2 - t_1} \int_{t1}^{t2} PG(t)dt \qquad I = \left(\sum_{i=1}^{m} i * t_i \right) \Big/ \left(\sum_{i=1}^{m} t_i \right)$$

PG Bereich Ausführungszeit

Parallelitätsprofil eines Beispielprogramms



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

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

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

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

    - Gibt die Verbesserung in der Verarbeitungsgeschwindigkeit an
    - Wert bezieht sich auf das jeweils bearbeitete Programm oder kann als Mittelwert eine Menge von Programmen angesehen werden
    - Üblicherweise gilt: $1 \leq S(n) \leq n$

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

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

  - Gibt die relative Verbesserung in der Verarbeitungsgeschwindigkeit an
  - Leistungssteigerung wird mit der Anzahl der Prozessoren *n* normiert
  - Üblicherweise gilt: $\frac{1}{n} \leq E(n) \leq 1$

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

      ➔Absolute Beschleunigung

      ➔Absolute Effizienz

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

      ➔Absolute Beschleunigung

      ➔Absolute Effizienz

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

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

  - Beschreibt den bei einem Multiprozessorsystem erforderlichen Mehraufwand für die Organisation, Synchronisation und Kommunikation der Prozessoren
  - Es gilt: $1 \leq R(n)$
    - Anzahl der auszuführenden Operationen eines parallelen Programms größer ist als diejenige des vergleichbaren sequentiellen Programms

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

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

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

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

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

    - Die Auslastung ist eine obere Schranke für die Effizienz:

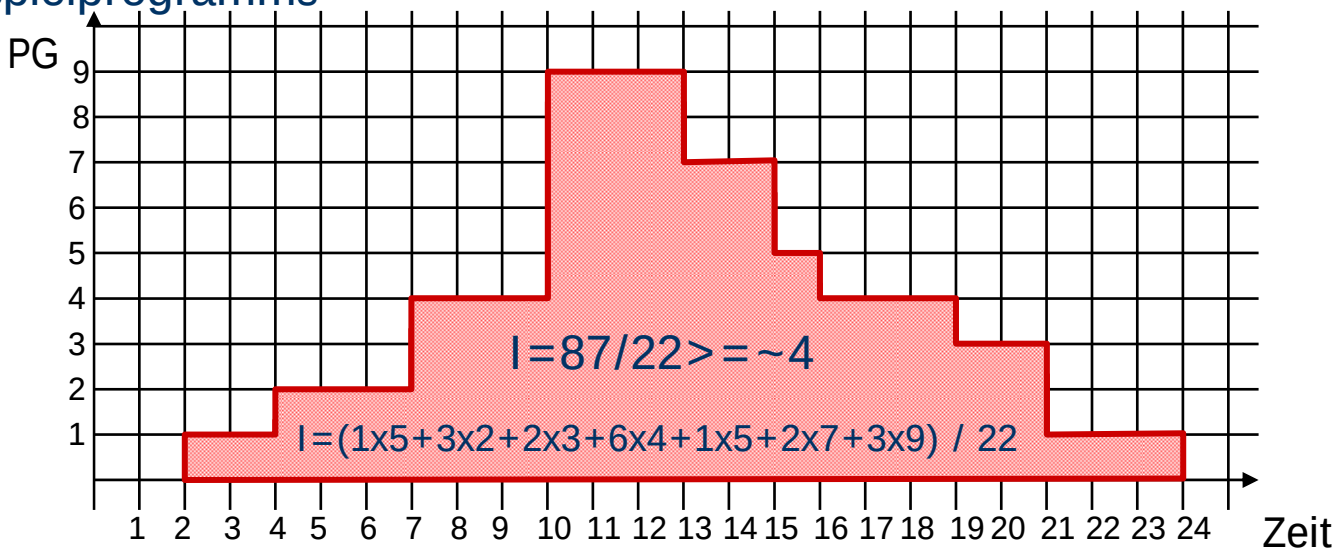
$$\frac{1}{n} \leq E(n) \leq U(n) \leq 1$$

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

$$I = \frac{1}{t_2 - t_1} \int_{t1}^{t2} PG(t)dt \qquad I = \left(\sum_{i=1}^{m} i * t_i \right) \Big/ \left(\sum_{i=1}^{m} t_i \right)$$

PG Bereich Ausführungszeit

Parallelitätsprofil eines Beispielprogramms



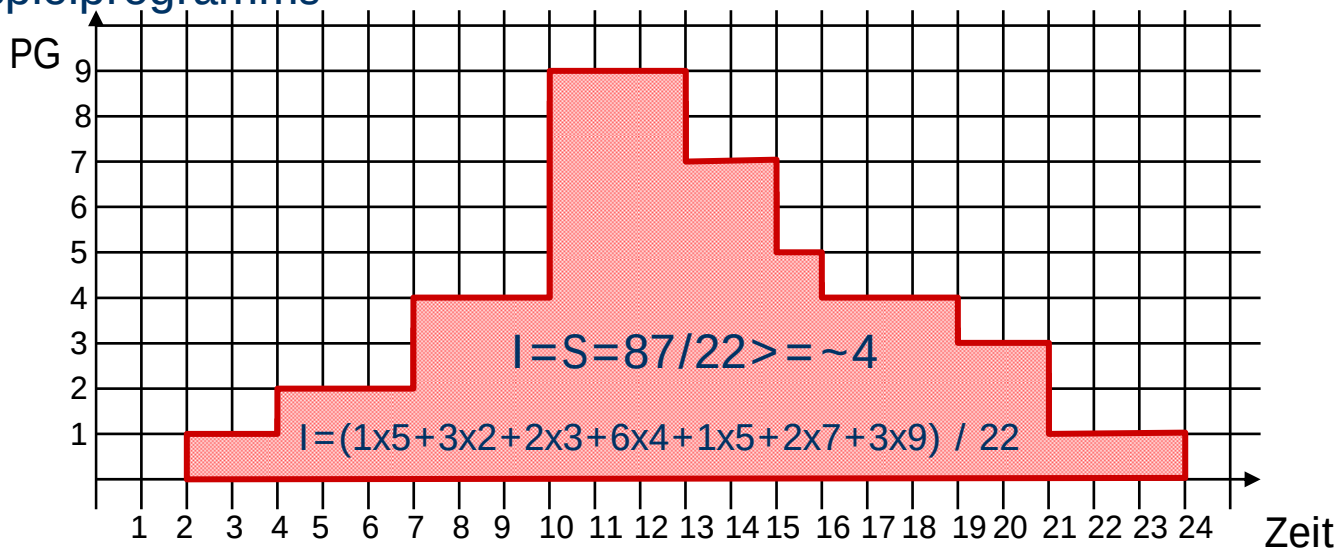
- Vergleich von Multiprozessorsystemen zu Einprozessorsystemen
  - Parallelindex I(n):
    - Unter den vereinfachenden Voraussetzungen (siehe Folie 12-19) gilt:

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

    - und damit: I(n)=S(n)

- Vergleich von Multiprozessorsystemen zu Einprozessorsystemen
  - Beziehung Parallelindex und Speedup
    - Unter den vereinfachenden Voraussetzungen (Folie 12-19)

Parallelitätsprofil eines Beispielprogramms



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

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

      - Daraus ergibt sich:

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

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

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

      $I(4) = 3$ und $U(4) = 0,75$

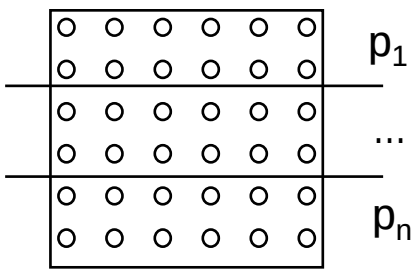
      - Es sind im Mittel drei Prozessoren gleichzeitig tätig, d.h., jeder Prozessor ist nur zu 75% der Zeit aktiv.
    - Mehraufwand:

      $R(4) = 1,2$

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

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

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



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

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

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

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

- Gesetz von Amdahl
  - Gesamtausführungszeit T(n)

$$T(n) = \underbrace{T(1) \cdot \frac{1-a}{n}}_{\text{Ausführungszeit des sequentiell ausführbaren Programmteils a}} + \underbrace{T(1) \cdot a}_{\text{Ausführungszeit des parallel ausführbaren Programmteils 1-a}}$$

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

  - Beschleunigung

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

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

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

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

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

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

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