→Grundbegriffe der Leistungsbewertung
→ Leistungsmaße (allgemein und speziell für Parallelrechner)

7.1 Aufgabe und Vorgehensweise

7.2 Betrachtungsebenen für die Leistungsbewertung
globale/lokale Leistungsmaße

7.3 allgemeine Leistungsmaße für Rechnersysteme
HW-Messgrößen, MIPS, MFLOPS, Benchmarks

7.4 Leistungsmaße für Parallelrechner
Laufzeit, Kosten/Overhead, Speedup, Effizienz, Amdahls Gesetz, Skalierung, Gustafsons Gesetz, Karp-Flatt-Metrik

7.5. krit. Bewertung von Leistungsdaten
z.B. Unterschied Rmax-Rpeak, bei akt.TOP500

7.1 Aufgabe und Vorgehensweise (I)

- Häufigstes Ziel beim Einsatz von PR: Leistungssteigerung!
- Problem: Bestimmung der Leistung - WIE?

- *quantitative* Aspekte müssen frühzeitig beim Systementwurf berücksichtigt werden! Dadurch können
  - rechtzeitig Alternativen entwickelt werden
  - Komponenten leistungsgerecht dimensioniert werden

- wie erreicht man hohe Leistung?
  - detaillierte Analyse des dynamischen Ablaufgeschehens
  - Ermittlung und Beachtung typischer Leistungsgrößen

7.1 Aufgabe und Vorgehensweise (II)

- *Aufgabe* der Leistungsbewertung:
  Untersuchung und Optimierung des dynam. Ablaufgeschehens innerhalb und zwischen den Komponenten eines Rechners

- *Ziel* der Leistungsbewertung:
  - Messung und formale Beschreibung realer Abläufe
  - Definition und Ermittlung typischer Leistungsgrößen
  - Bereitstellung von Entscheidungshilfen für den Entwurf der HW, der System-SW und der Anwendungen

HW- und SW-Struktur von Parallelrechnern sind noch komplexer als bei klassischen Rechnern, daher Leistungsbewertung hierfür noch wichtiger!

7.1 Aufgabe und Vorgehensweise (III)

Methodik der Leistungsbewertung

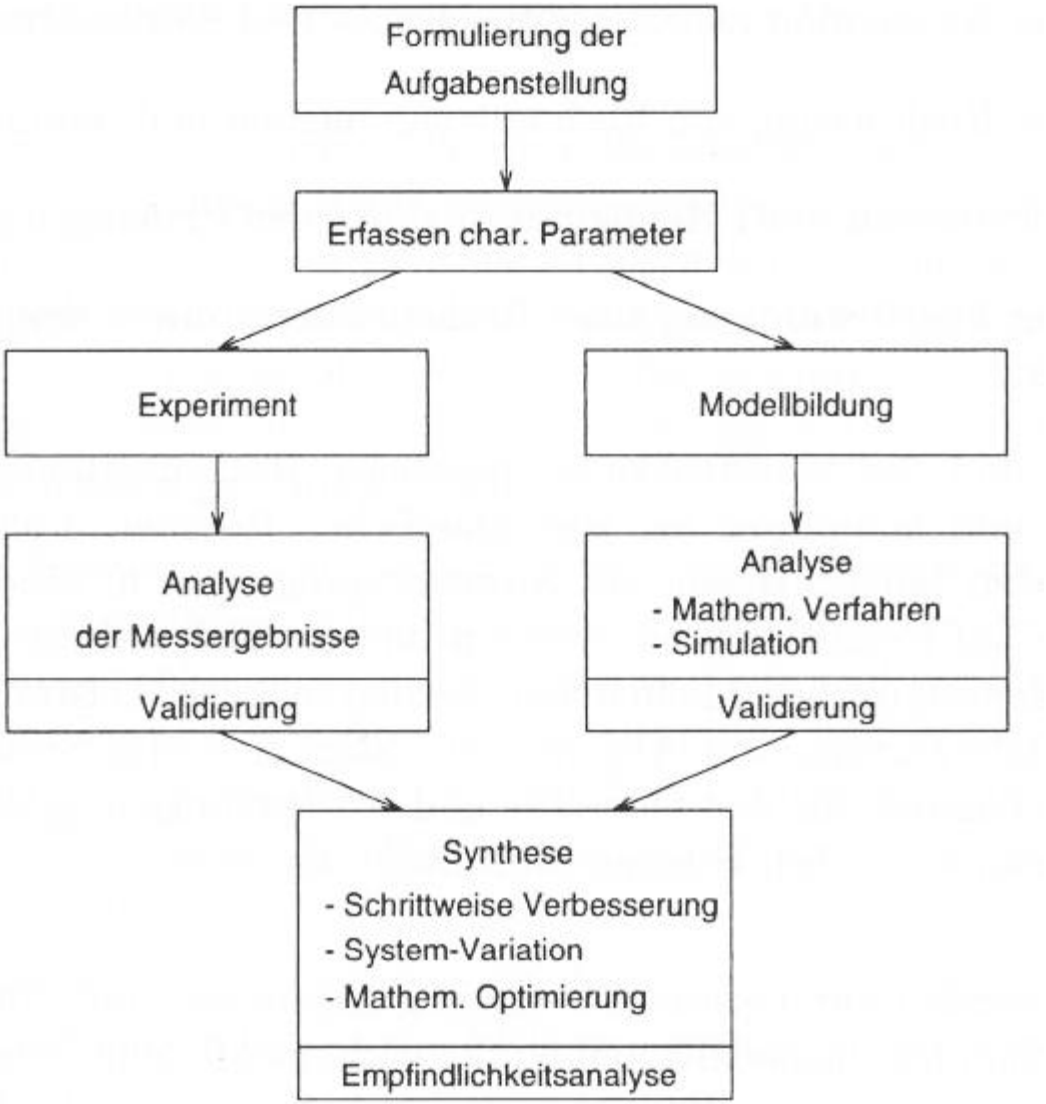


Bild-Quelle: Waldschmidt: Parallelrechner. Teubner 1995

7.2 Betrachtungsebenen für die Leistungsbewertung

7.2.1 Überblick

Geeignete Leistungsmaße sind abhängig von

- konkreter Rechnerarchitektur
- Problemklasse

2 typische Betrachtungsebenen:

- Gesamtverhalten des Systems
  (globale Verkehrsprobleme und Leistungsmaße)
  wichtig für Anwender, Betreiber, Hersteller
- interner Ablauf im Detail
  (lokale Verkehrsprobleme und Leistungsmaße)
  wichtig für Entwickler von HW/SW zur Feinabstimmung der Komponenten und als Basis für eine realistische Modellierung des Gesamtsystems

7.2.2 Globale Verkehrsprobleme und Leistungsmaße

→ Globale Betrachtung des Gesamtsystems bzw. seiner Hauptkomponenten (als Black-Box Analyse)

Typische globale Leistungsmaße:

- Durchsatz des Gesamtsystems (Aufgaben pro Zeit)
- Antwortzeiten (Mittelwerte, statist. Verteilungsfunktion)
- Auslastung der Prozessoren (z.B. via mittl. Verweilzeit, idle time,…)
- Auslastung der Speicher und E/A-Einheiten
- Auslastung einzelner Übertragungsleitungen
- max. Durchsatz des Verbindungsnetzes
- Zuverlässigkeit (z.B. mittlere Lebensdauer von Komponenten, Verfügbarkeit, …)

7.2.3 Lokale Verkehrsprobleme und Leistungsmaße (I)

→ Lokale Betrachtung des Gesamtsystems bzw. seiner Hauptkomponenten (d.h. aus Sicht einzelner Programme, Teilaufgaben oder Befehle)

Typische lokale Leistungsmaße:

- Antwortzeit für individuelle Bedienprogramme
- Zeitanteil für Verwaltungsaufgaben (Overhead), Datentransfers,…
- Grad der Parallelität einzelner Programme (Min./Max.) sowie einzelner Teilaufgaben
- Anzahl und Ausführungszeit einzelner Teilaufgaben
- Anzahl, Ursache und Dauer von Unterbrechungen
- Verluste durch Koordination (Synchronisation) von Teilaufgaben
- Einfluss des Betriebssystems
- relative Häufigkeit von Befehlstypen
- Lebensdauer von Operanden

7.2.3 Lokale Verkehrsprobleme und Leistungsmaße (II)

Spezielle lokale Leistungsmaße für *Multiprozessorsysteme* beziehen sich vor allem auf

- Datentransport zwischen den Komponenten:
  - Anzahl transferierter Datenblöcke, Blocklängen, …
  - Warteschlangenlängen, Verkehrsengpässe
  - Pufferauslastung, Cachetrefferrate, …
- Speicherzugriffe:
  - Anzahl und Abstandsverteilung
  - Anzahl typischer Speicherzugriffskonflikte
    - Prozessor <-> lokaler Speicher
    - Prozessor <-> benachbarter Speicher
    - Prozessor <-> globaler Speicher
  - Leistungsminderung der Prozessoren durch Zugriffskonflikte bei Daten- oder Codesharing

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (I)

Interessante Größen aus HW-Sicht:

- CPU-Geschwindigkeit → wie bei Einzelprozessorsystemen
- E/A-Geschwindigkeit → wie bei Einzelprozessorsystemen
- Leistung des Verbindungsnetzwerkes:
  - Antwortzeit (Umlauflatenz, round trip latency):
    Dauer vom Senden eines Paketes bis zum Eintreffen der Antwort
    abhängig von Vermittlungsart, Routing, Switch, …
  - Bandbreite, z.B. mittels
    - Bisektionsbandbreite,
    - aggregierte Bandbreite (max. Anzahl gleichzeitig übertragbarer Bits)
    - durchschnittl. Bandbreite jeder CPU
    - theoret. Bandbreite ist prakt. nicht annähernd erreichbar
- Bewertung stark anwendungsabhängig
- Mehr Bandbreite kann man „kaufen“, weniger Latenz nicht!

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (II)

Zykluszeit einer CPU:

$$t_c = \frac{1}{\text{Taktrate } r_c \text{ [Hz]}}$$

Rechenzeit für ein Programm A:

$T_{CPU}(A) = n_c(A) * t_c$ mit $n_c(A)$ Anzahl der CPU-Zyklen zur Ausführung aller Maschinenbefehle von Programm A

verschiedene Befehle haben verschiedene Laufzeiten, daher CPI (Clock cycles Per Instruction) als Mittelwert benutzt (mittlere Anzahl von CPU-Zyklen für einen Befehl des Programms A):

$T_{CPU}(A) = n(A) * CPI(A) * t_c$ mit n(A) Anzahl der Befehle für Programm A

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (III)

MIPS-Rate für Programm A (Million Instructions Per Second):

$$MIPS(A) = \frac{n(A)}{T_{CPU}(A) * 10^6} = \frac{r_c}{CPI(A) * 10^6} = \frac{1}{CPI(A) * t_c}$$

Wegen unterschiedl. Befehlsdauer geben standardisierte Befehlsmixe eine *mittlere MIPS-Rate* an:

$$\overline{MIPS}(A) = \frac{1}{\sum_i p_i * n_i * t_c}$$

mit p_i als rel. Häufigkeit für Befehlstyp i und n_i Anzahl der befehle vom Typ i

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (IV)

MIPS-Rate

- abhängig von Befehlssatz der CPU
- abhängig von spez. Beschleunigungsmechanismen
- nur als grober Wert verwendbar

Für numerische, wiss.-techn. Berechnungen sind Gleitkomma-Befehle wichtiger, daher hierfür anderes Maß:

FLOPS-Rate eines Programms (Floating Point Operations Per Second):
meist als MFLOPS (Million FLOPS):

$$MFLOPS = \frac{n_{flop}(A)}{T_{CPU}(A) * 10^6}$$

mit $n_{flop}(A)$ Anzahl der FP-Operationen von A

ohne Beachtung von Unterschieden bei verschiedenen FP-Operationen inzwischen GFLOPS (10^9), TFLOPS (10^{12}), PFLOPS (10^{15}) !

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (V)

MIPS und MFLOPS reichen nicht….

- Parallelität in der HW unberücksichtigt
- Einfluss der SW (BS, Compiler, …) unberücksichtigt
- Lastparameter (Vektorlänge, E/A-Anforderungen,…) unberücksichtigt
- immer noch abhängig von einem konkreten Programm
- oft nur theoretischer Maximalwert angegeben, weicht von praktisch erreichbarem Wert z.T. deutlich ab

→Realistischere (nicht von Einzelbefehlen dominierte) Lasten als Bewertungsbasis nötig:

- idealerweise die künftig auszuführenden Programme ☹
- Ersatz: Benchmarks

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (VI)

Benchmarks

= Testprogramme mit speziellen Eigenschaften

- ihre Ausführungszeit kann auf verschiedenen Computersystemen gemessen werden
- ermöglichen standardisierte Leistungsbewertung
- verschiedene Klassen:

a) Synthetische Benchmarks:

- meist kleinere Menge von Befehlen, die als typisch für größere Programme angesehen werden
- daher meist keine sinnvollen Berechnungen
- Verhalten komplexer Programme nur unvollständig repräsentiert (vor allem Last zwischen CPU und Speicher)
- Gefahr von (unbeabsichtigten) Compileroptimierungen
- Beispiele: Whetstone (in FORTRAN, misst FP-Leistung),
  Dhrystone (in C, misst Integer-Leistung)

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (VII)

b) „Spielzeug-Benchmarks“:

- meist kleinere, vollständige Programme mit rel. Einfachen Algorithmen
- Verhalten komplexer Programme nur unvollständig repräsentiert
- Beispiele: Quicksort
  Sieb des Erathostenes

c) Kernprogramme/Programmkerne

- relevante Teile von realen größeren Programmen, deren Ausführung einen Großteil der Rechenzeit dieser Programme ausmacht, zumeist aber auf wiss.-techn. Fälle beschränkt
- Vorteil: ihr Quelltext ist meist nur ein geringer Teil des Gesamt-Programms
- Beispiele: Livermore Loops (24 Schleifen aus wiss.-techn. Anwend.)
  Linpack (Teil einer FORTRAN-Bibliothek der lin. Algebra)
  Linux-Kernel

7.3 Allgemeine (populäre) Leistungsmaße für Computersysteme (VIII)

d) Benchmarks aus kompletten Programmen:

- meist Sammlung mehrere, für durchschnittliche Benutzer typische Programme
- Vorteil: Berücksichtigung verschiedener Programmeigenschaften
- i.allg. am besten zur Leistungsbewertung geeignet
- am weitesten verbreitete Beispiele:
  - SPEC-Benchmark-Sammlung (SPEC Konsortium, spec.org, System Performance Evaluation Cooperative)
    aktuell: SPEC CPU2006
    vor allem für Integer-Leistung SPECint und FP-Leistung SPECfp von CPUs
    aber auch für zahlreiche andere Leistungsgrößen:
    SPEC HPC, SPEC Mail, SPEC MPI, SPEC OMP, SPEC Power, SPEC Virtualization, …
  - TPC-Benchmark (Transaction Processing Performance Council),Benchmarks zu OLTP und Decision Support Systems

7.4 Spezielle Leistungsmaße für PR (I)

Allgemeine Leistungsmaße (inkl. Benchmarks) erfassen Leistung von PR meist nur grob:

- Verarbeitungsart der Befehle (sequ./parallel, mit/ohne Pipelining) oft nur indirekt erfasst
- Kommunikationsverhalten (CPU – Speicher/Cache, CPU - CPU) inkl. ggf. nötiger Synchronisation praktisch kaum erfasst
- die von der Aufgabenstellung (Problem) abhängige Auslastung des Computers bzw. die Nutzung seiner Komponenten wird nicht betrachtet

→ Daher zahlreiche Versuche für erweiterte Betrachtung dazu, vor allem Basis der Gesamtlaufzeit eines Programms

7.4 Spezielle Leistungsmaße für PR (I)

Gesamtlaufzeit eines Programms der Größe n auf p Prozessoren $T_p(n)$

Laufzeit eines parallelen Programms:
Zeit zwischen dem Start des Programms und dem Ende der Berechnung auf dem letzten Prozessor

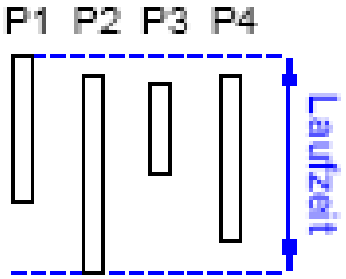


Bild-Quelle: Wissmüller: VO Parallelverarbeitung. Uni Siegen WS2006/07

$T_p(n)$ p = Anzahl der beteiligten Prozessoren
n = Problemgröße (z.B. Datenabhängigkeiten)

besteht aus:

- Rechenzeit T_{CPU} für Berechnungen mit lokalen Daten
- Kommunikationszeit T_{COM} für Datenaustausch zw. CPUs
- Wartezeit T_{WAIT} z.B. wegen ungleicher Last, E/A, Datenabhängigk.
- Synchronisationszeit T_{SYN} für Synchronisation der beteil. Proz.
- Plazierungszeit T_{Place} für dynam. Zuordnung der Tasks zu CPUs
- Startzeit T_{Start} zum Starten der parallelen Tasks auf allen CPUs

7.4 Spezielle Leistungsmaße für PR (II)

Gesamtlaufzeit eines Programms der Größe n auf p Prozessoren $T_p(n)$

$$T_p(n) = T_{CPU} + \underbrace{T_{COM} + T_{WAIT} + T_{SYN}}_{\text{Overheadzeit } T_{CWS}} + \underbrace{T_{Place} + T_{Start}}_{\text{Rüstzeit } T_{Setup}}$$

Laufzeitverringerung vor allem durch Minimierung der Overheadzeit!

Durchführung der Laufzeitmessung: *„Vermesse mit einer Maus einen Elefanten, aber vermesse nie eine Maus mit einem Elefanten!“*

- falls Laufzeit des Messprogramms sehr gering im Vergleich zum zu vermessenden Programm, kann sie vernachlässigt werden
- falls Laufzeit des Messprogramms nicht klein genug, so muss die Problemgröße des zu vermessenden Programms erhöht werden
- falls das technisch nicht geht, dann muss die Laufzeit des Messprogramms ermittelt und abgezogen werden

7.4 Spezielle Leistungsmaße für PR (III)

Kosten eines parallelen Programms $C_p(n)$

$C_p = T_P(n) * p$

$T_p(n)$ Gesamtlaufzeit eines Programms der Größe n
p Anzahl der Prozessoren
→ Maß für die von allen Prozessoren zur Lösung geleistete Arbeit

Wann sind die Kosten optimal?
→ Wenn das parallele Programm zur Lösung genauso viele Operationen durchführt, wie ein sequentielles Programm mit der Laufzeit $T_s(n)$: $C_p(n) = T_s(n)$
d.h. im parallelen Fall wird von den einzelnen parallelen Prozessen die gleiche Anzahl Operationen ausgeführt wie im sequentiellen Fall

7.4 Spezielle Leistungsmaße für PR (IV)

Overhead eines parallelen Programms $H_p(n)$

$$H_p(n) = C_P(n) - T_s(n) = p * T_p(n) - T_s(n)$$

Overhead gibt Differenz an zwischen den Kosten für ein paralleles Programm im Vergleich zum sequentiellen Programm

Für kostenoptimales Programm gilt: $H_p(n) = 0$

d.h. beim Übergang von sequentieller zu paralleler Lösung entstehen keine zusätzlichen Operationen und Redundanzen

7.4 Spezielle Leistungsmaße für PR (V)

Speedup (Beschleunigung, Leistungssteigerung) $S_p(n)$

Absoluter Speedup (praktisch mitunter nicht zu bestimmen):

$$S_p(n) = \frac{T_S}{T_p} = \frac{\text{Laufzeit des schnellsten sequent. Programms/Algor.}}{\text{Laufzeit des parallelen Programms/Algor. auf p CPUs}}$$

Relativer Speedup (praktische Näherung zum absoluten Speedup):

$$S_p(n) = \frac{T_1}{T_p} = \frac{\text{Laufzeit des parallelen Programms/Algor. auf 1 CPU}}{\text{Laufzeit des parallelen Programms/Algor. auf p CPUs}}$$

→ Laufzeitgewinn bei Ausführung eines parallelen Programms auf p Prozessoren im Vergleich zur sequ. Ausführung des Programms (auf 1 Prozessor)

7.4 Spezielle Leistungsmaße für PR (VI)

Speedup

$S_p(n) \leq p$ realer Speedup i.allg. durch Anzahl der CPUs begrenzt

$S_P(n) = p$: linearer Speedup, ideal

$S_P(n) > p$: superlinearer Speedup

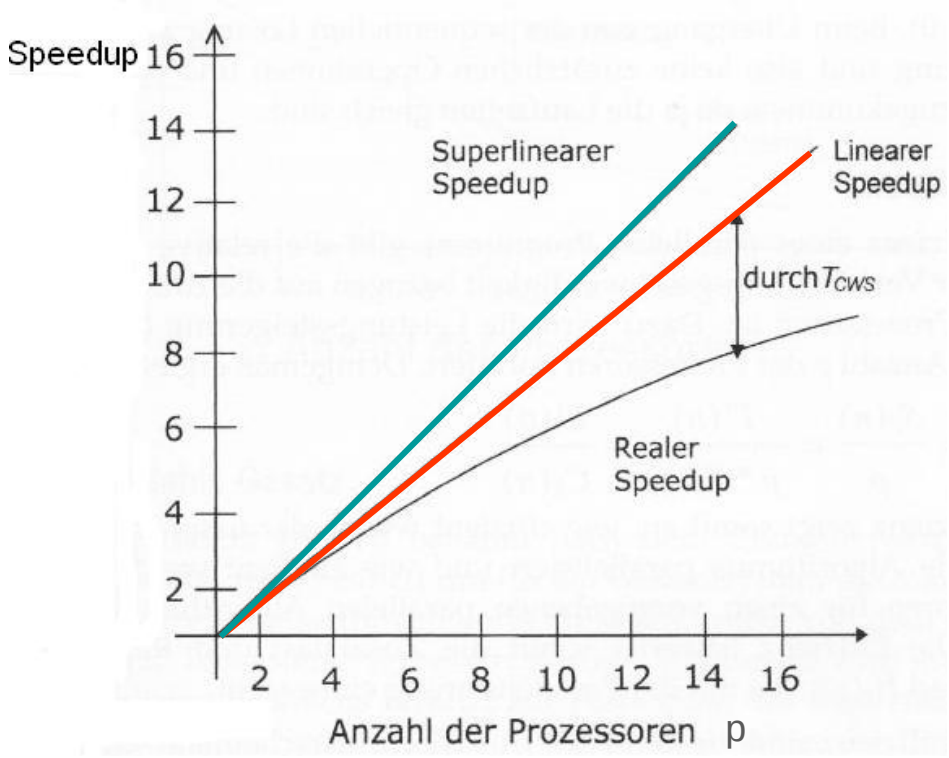


Bild-Quelle: Bengel, u.a.: Masterkurs Parallele und verteilte Systeme. Vieweg/Teubner, 2008

7.4 Spezielle Leistungsmaße für PR (VII)

Speedup

$S_P(n) > p$ eigentlich unmöglich, trotzdem zu beobachten, wegen

- implizite Änderung des Algorithmus, z.B. bei paralleler Baumsuche (begrenzte Breitensuche anstelle von Tiefensuche)
- Caching-Effekte (bei p Prozessoren p-mal soviel Cache wie bei 1 Prozessor,
  → insgesamt höhere Trefferraten)

Oft: bei fester Problemgröße sinkt S_p mit wachsendem p wieder,
Grund: Mehr Kommunikation, weniger Rechenarbeit pro Prozessor.

7.4 Spezielle Leistungsmaße für PR (VIII)

Effizienz $E_p(n)$

$$E_P(n) = \frac{S_p(n)}{p} = \frac{T_s(n)}{p * T_p(n)} = \frac{T_s(n)}{C_p(n)}$$

- gibt relative Verbesserung der Verarbeitungsgeschwindigkeit bezogen auf Anzahl der eingesetzten Prozessoren an
- „Wie effizient wurde der sequentielle Algorithmus parallelisiert?“
- damit Bewertung von Zusatzlast und Redundanz (Overhead) bei der Parallelisierung
- Maß für die Ausnutzung der p Prozessoren des Parallelrechners
- Angabe i.allg. in %
- Bsp.: $E_p(n) = 0.9$:
  90% des theoretisch maximal möglichen Speedups werden erreicht

7.4 Spezielle Leistungsmaße für PR (IX)

Effizienz $E_p(n)$

Typische Fälle:

$E_p(n) < 1$: Algorithmus ist suboptimal bzgl. seiner Kosten praktischer Normalfall

$E_p(n) = 1$: Algorithmus ist kostenoptimal

$E_p(n) > 1$: superlinearer Speedup oder der parallele Algorithmus ist bzgl. seiner Laufzeit optimaler als der vergleichbare sequ. (bzw. als der bzgl. Laufzeit beste bekannte sequ.) Algorith.

7.4 Spezielle Leistungsmaße für PR (X)

Effizienz $E_p(n)$

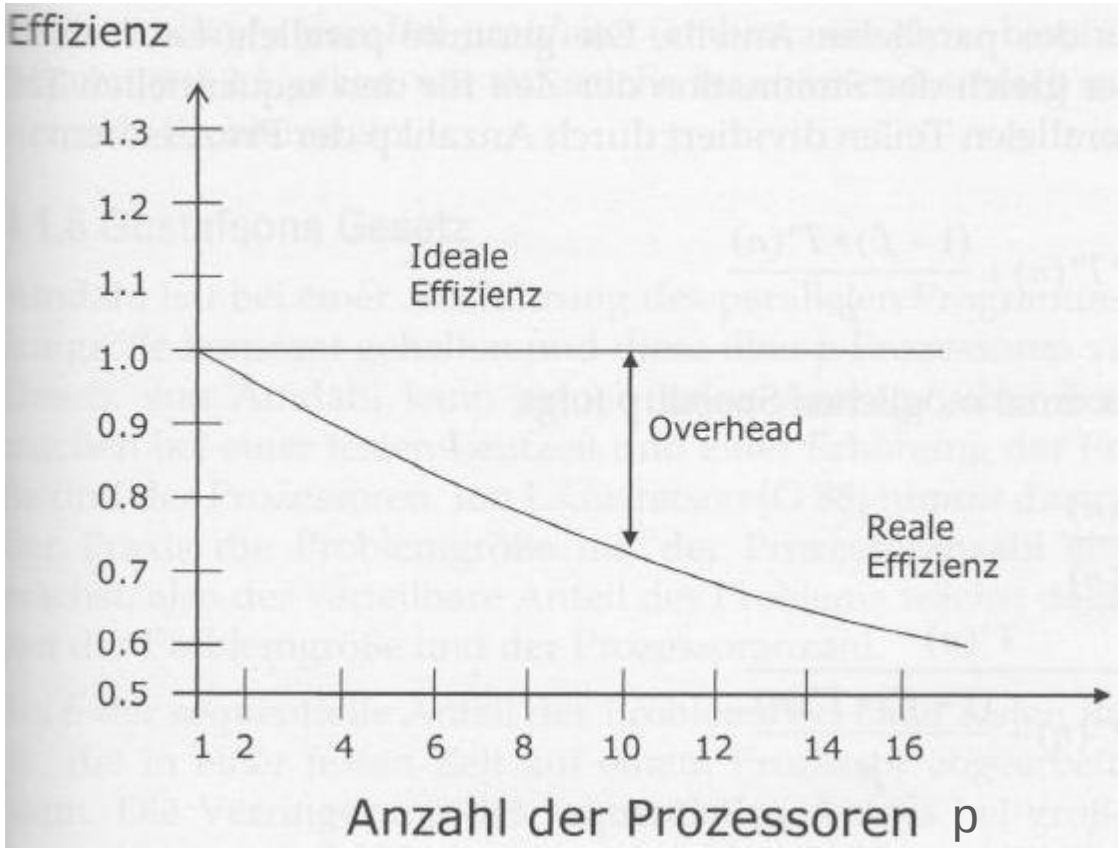


Bild-Quelle: Bengel, u.a.: Masterkurs Parallele und verteilte Systeme. Vieweg/Teubner, 2008

7.4 Spezielle Leistungsmaße für PR (XI)

Amdahls Gesetz (1967)

- Zur Vorhersage maximal zu erwartender Verbesserungen für das Gesamtsystem bei Verbesserung eines/mehrerer Teilsysteme
- Für PR:
  Aussage bzgl. des theoretisch maximalen Speedups bei p Prozessoren und einem nicht parallelisierbaren sequentiellen Anteil f im Programm mit $0 \leq f \leq 1$:
  - f = 0: Algorithmus ist vollständig parallel ausführbar
  - f = 1: kein Teil des Algorithmus ist parallel ausführbar
  - (1-f): parallel ausführbarer Anteil des Algorithmus

- $T_p(n) \geq f * T_s(n) + \frac{(1-f) * T_p(n)}{p}$ Gesamtlaufzeit eines Programms

  Zeit für sequ. Teil | Zeit für parallelen Teil

7.4 Spezielle Leistungsmaße für PR (XII)

Amdahls Gesetz

$$\bar{T}_p(n) \geq f * T_S(n) + \frac{(1-f) * T_S(n)}{p}$$

Für den maximal möglichen Speedup folgt:

$$S_p(n) = \frac{T_S(n)}{T_p(n)}$$

$$= \frac{T_S(n)}{f * T_S(n) + \frac{(1-f) * T_S(n)}{p}}$$

$$S_p(n) = \frac{1}{f + \frac{(1-f)}{p}} \quad (\textit{Amdahlsches Gesetz})$$

$$S_p(n) = \frac{p}{1 + f * (p-1)}$$

Bild-Quelle: Bengel, u.a.: Masterkurs Parallele und verteilte Systeme. Vieweg/Teubner, 2008 (angepasst)

7.4 Spezielle Leistungsmaße für PR (XIII)

$$S_p(n) = \frac{p}{1 + f * (p - 1)}$$

Amdahls Gesetz

→ bereits ein geringer sequ. (nicht parallelisierbarer) Programmanteil begrenzt den max. erreichbaren Speedup erheblich!

Bsp.: bei gegebenem Algorithmus und gegebener Problemgröße

bei sequentiellem Anteil 10% (f=0.1): $S_p(n) = 10$!

→*Problemlösung kann max. um Faktor 10 beschleunigt werden!*

bei sequentiellem Anteil 5% (f=0.05): $S_p(n) = 20$!

bei sequentiellem Anteil 1% (f=0.01): $S_p(n) = 100$!

bei sequentiellem Anteil 0.5% (f=0.005): $S_p(n) = 200$!

konkret bei Zielsystem mit 1000 CPUs
und max. Parallelisierungsgrad von 1000:

sequ. Anteil 10% (f=0.1): $S_{1000}(n) = 10$ und $E_{1000}(n) = 1\%$!

→ *Es werden nur 1% der gesamten CPU-Leistung genutzt!*

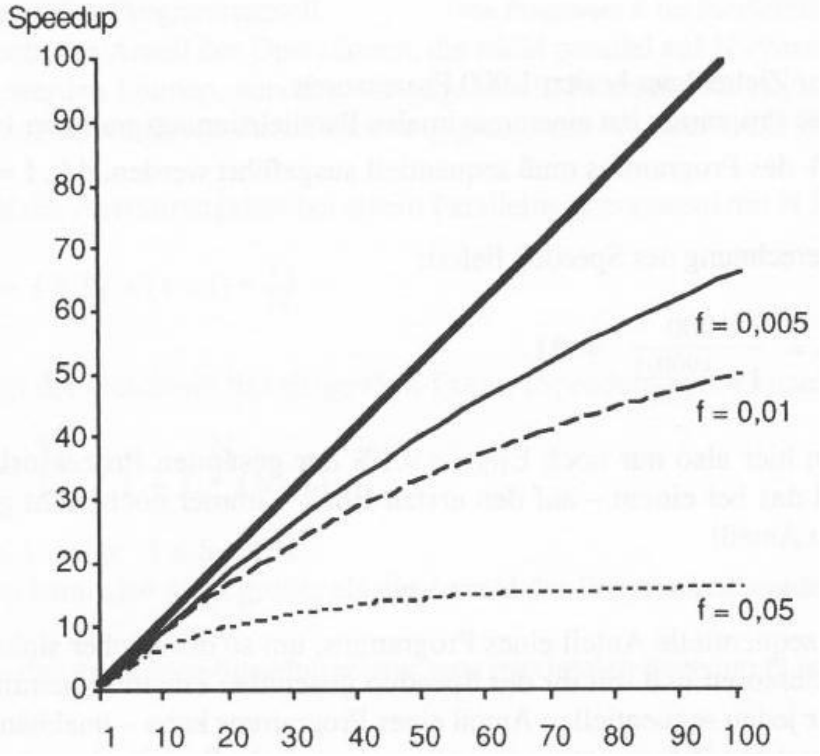
sequ. Anteil 5% (f=0.05): $S_{1000}(n) = 20$ und $E_{1000}(n) = 2\%$!

sequ. Anteil 1% (f=0.01): $S_{1000}(n) = 100$ und $E_{1000}(n) = 10\%$!

sequ. Anteil 0.5% (f=0.005): $S_{1000}(n) = 200$ und $E_{1000}(n) = 20\%$!

7.4 Spezielle Leistungsmaße für PR (XIV)

Amdahls Gesetz



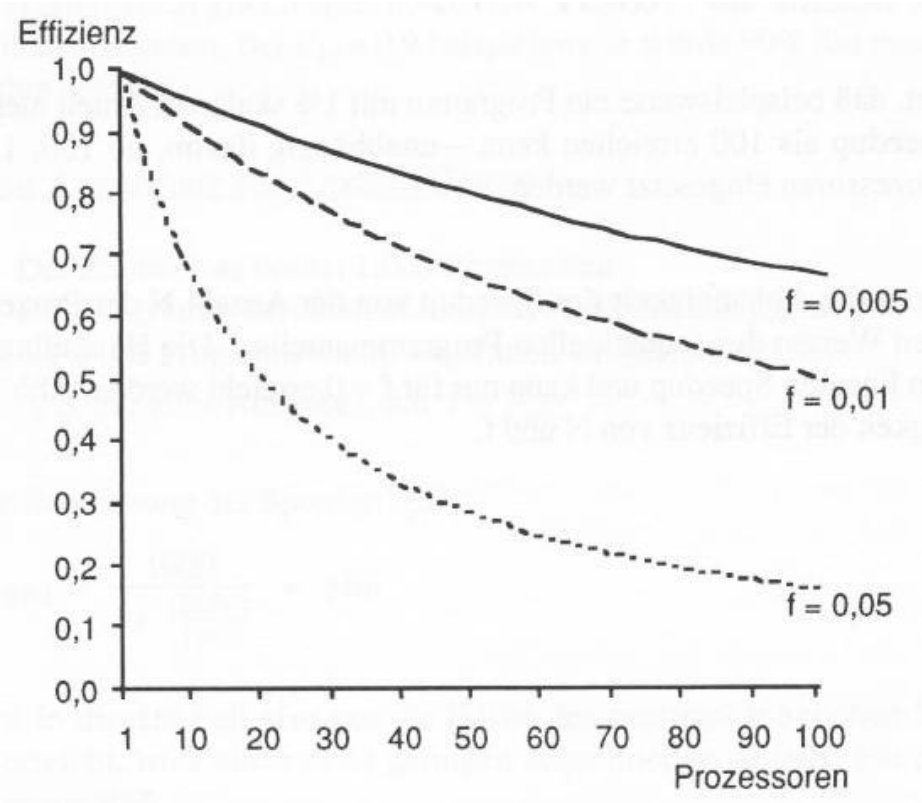


Bild-Quelle: Bräunl: Parallele Programmierung. Vieweg, 1993

7.4 Spezielle Leistungsmaße für PR (XV)

Amdahls Gesetz

Für f>0 und große CPU-Anzahl (p → ∞) folgt aus Amdahls Gesetz näherungsweise

$$S_p(n) \leq \frac{1}{f}$$

Bei 10% sequ. Anteil kann die Lösung nur um max. Faktor 10 beschleunigt werden, egal wie groß n wird

→ Parallele Berechnungen nur für kleine Anzahl von CPUs sinnvoll oder nur mit sehr kleinen Werten von f

Massive Parallelität lohnt sich nur für Probleme, die ohne großen Aufwand parallelisierbar sind und die keinen sequ. Anteil haben
diese Probleme besitzen vorgegebene, inhärente Parallelität
= „embarrasingly parallel problems“
(„beschämend einfach zu parallelisierende Probleme“)
z.B. SETI@home, Monte-Carlo-Simulation

7.4 Spezielle Leistungsmaße für PR (XVI)

Gustafsons Gesetz

Annahme: in der Praxis nimmt mit wachsender CPU-Zahl auch die Problemgröße zu
→ damit wächst der parallelisierbare Programmanteil und der sequ. Anteil des Progr. nimmt ab!

Bsp.:

- 1 Maler braucht für 1 Zimmer 1 Stunde.
- 2 Maler brauchen für das Zimmer nur noch ½ Std. okay?
- …
- 60 Maler brauchen für das Zimmer nur noch 1 Minute. okay???

→Fehler: Sie stehen sich im Weg oder streiten um begrenzte Ressourcen

→Aber: 60 Maler könnten 60 Zimmer parallel in 1 Std. schaffen!

7.4 Spezielle Leistungsmaße für PR (XVII)

Gustafsons Gesetz

$$f = \frac{f_1}{p * (1 - f_1) + f_1}$$

Mit Amdahls Gesetz folgt:

$$S_p(n) = \frac{1}{f + \frac{1 - f}{p}}$$

$$= \frac{1}{\frac{f_1}{p * (1 - f_1) + f_1} + \frac{1 - \frac{f_1}{p * (1 - f_1) + f_1}}{p}}$$

$$= p * (1 - f_1) + f_1 \qquad (\textit{Gustafsons Gesetz})$$

Formel für Verringerung des sequ. Anteils bei größeren Problemen und höherer CPU-Zahl,

mit f1 = sequ. Anteil des Problems bei einer festen Problemgröße, die in einer festen Zeit auf einem Prozessor abgearbeitet werden kann.

Bild-Quelle: Bengel, u.a.: Masterkurs Parallele und verteilte Systeme. Vieweg/Teubner, 2008

7.4 Spezielle Leistungsmaße für PR (XVIII)

Gustafsons Gesetz

Bei fester Laufzeit f_1 und Begrenzung der Problemgröße durch die CPU-Anzahl ergibt sich:

Speedup wächst mit konstantem sequ. Programmanteil linear mit der Prozessoranzahl p

→ Damit ist bei großen Problemgrößen für massiv-parallele Systeme und Cluster mit großer CPU-Anzahl ein Speedup nahe p erreichbar, d.h.:

$$\lim_{n \to \infty} S_p(n) = p$$

7.4 Spezielle Leistungsmaße für PR (XIX)

Skalierbarkeit

Anwendung ist skalierbar, wenn wachsende Problemgröße durch Einsatz von noch mehr Prozessoren so kompensiert werden kann, dass die zur Problemlösung nötige Zeit konstant bleibt.

In diesem Fall bleibt Effizienz eines parallelen Programms konstant, bei gleichzeitigem Ansteigen der Prozessoranzahl und der Problemgröße

Granularität der Programme (Anzahl der Rechenschritte zwischen zwei Kommunikationsschritten) hat Einfluss.

Amdahl oder Gustafson???

- falls Anwendung nicht skalierbar: Amdahls Gesetz
- falls perfekt skalierbar: Gustafsons Gesetz

7.4 Spezielle Leistungsmaße für PR (XX)

Skalierungsgewinn Scaleup …

… für ein Problem der Größe n auf k Prozessoren gegenüber einem kleineren Problem der Größe m (m<n) auf einem Prozessor:

Wenn $T_1(m) = T_k(n)$

d.h. Ausführungszeit für „kleines“ Problem auf einem Prozessor ist gleich Ausführungszeit für „großes“ Problem auf k Prozessoren)

Dann ist der Scaleup $SC_k = \frac{n}{m}$

7.4 Spezielle Leistungsmaße für PR (XXI)

Karp-Flatt-Metrik

Bei Amdahl und Gustafson wird Overhead-Zeit T_{CWS} für die Parallelisierung (d.h. Verlust durch Kommunikation, Lastverteilung und Synchronisation) vernachlässigt.

Alternative:

- Speedup (nach Gustafson) für mehrere Werte von p *experimentell* bestimmen
- daraus *empirischen* sequentiellen Anteil f ermitteln (= Karp-Flatt-Metrik)

7.4 Spezielle Leistungsmaße für PR (XXII)

Karp-Flatt-Metrik

Nach Amdahls Gesetz gilt:

$$\frac{1}{S_p(n)} = f + \frac{1-f}{p}$$

Diese Gleichung lässt sich nach dem sequentiellen Anteil f auflösen:

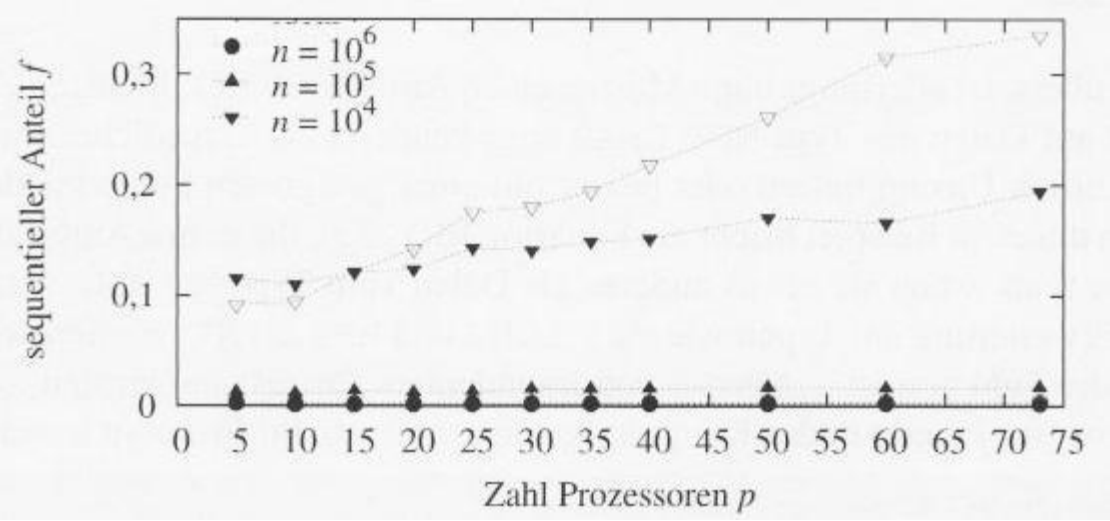
$$f = \frac{1/S_p(n) - 1/p}{1 - 1/p} \qquad (\textit{Karp-Flatt- Metrik})$$

Bild-Quelle: Bengel, u.a.: Masterkurs Parallele und verteilte Systeme. Vieweg/Teubner, 2008

7.4 Spezielle Leistungsmaße für PR (XXIII)

Karp-Flatt-Metrik

- Die Karp-Flatt-Metrik zeigt, dass f abhängig von p ist (entgegen der Annahme bei Amdahl).
- sie zeigt damit, ob die Effizienz eines parallelen Programms durch den inhärenten sequentiellen Anteil dominiert wird oder durch den Overhead



Beispiel einer Karp-Flatt-Metrik (numer. Integration mit n Stützstellen auf einem Cluster mittels MPI)

f wächst mit p: die Kommunikation dominiert also den Overhead.

Bildquelle: Bauke/Mertens: Cluster-Computing. Berlin: Springer, 2006) (angepasst)

7.4 Spezielle Leistungsmaße für PR (XXIV)

Isoeffizienzfunktion

Im Allg. wird die Effizienz eines parallelen Programms

- mit wachsendem p (Prozessorzahl) abnehmen
- aber mit steigender Problemgröße n zunehmen

Isoeffizienzfunktion gibt an, wie stark die Problemgröße wachsen muss, um bei zunehmendem Parallelisierungsgrad die Effizienz konstant zu halten:

$T(n,1) = C\, T_O(n,p)$ mit $T(n,1)$ sequentielle Laufzeit,
und $T_O(n,p)$ Overheadzeit bei Problemgröße n auf p Prozessoren
und $C = E / (1-E)$ (konstant für gegebenes E)

Problemgröße n sollte mind. so schnell wachsen, dass die sequ. Laufzeit genauso schnell wächst, wie die Gesamtzeit des parallelen Overheads

7.5 Kritische Bewertung von Leistungsdaten

- Leistungsdaten wie Speedup eines PR sind immer anwendungsbezogen und somit nur bedingt übertragbar
- Speedup-Werte können insbes. bei SIMD-Systemen verfälscht sein, z.B. wegen Inaktivität nicht benötigter PEs
- Peak-Performance vieler System weicht z.T. erheblich von realen Werten ab (vgl. TOP500)

  Bsp.: Liste 6/2008 #1: IBM Roadrunner:
  Rpeak = 1376 TFLOPS
  Rmax = 1026 TFLOPS
  Differenz ca. 25% !
- Verfügbare höhere Sprachen können mitunter nicht alle Fähigkeiten der PR-Hardware nutzen, daher oft Einbeziehung von maschinenspezifischen Routinen/Bibliotheken
- …