Bewertung paralleler Algorithmen

Stefan Lang

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Universität Heidelberg
INF 368, Raum 532
D-69120 Heidelberg
phone: 06221/54-8264
email: Stefan.Lang@iwr.uni-heidelberg.de

WS 11/12

Themen

Bewertung paralleler Algorithmen

- Speedup, Effizienz
- Parallelitätsgrad, Kosten
- Isoeffizienz
- Gesetz von Amdahl
- Gustafson-Skalierung
- Skalierbarkeit

Bewertung von parallelen Algorithmen I

Wie analysiert man die Eigenschaften eines parallelen Algorithmus zur Lösung eines Problems $\Pi(N)$?

- Problemgröße N ist frei wählbar
- Lösung des Problems mit sequentiellem bzw. parallelem Algorithmus
- Hardwarevoraussetzungen:
  - MIMD-Parallelrechner mit P identischen Rechenknoten
  - Kommunikationsnetzwerk skaliert mit der Anzahl der Rechenknoten
  - Aufsetzzeit, Bandbreite und Knotenleistung sind bekannt
- Ausführung des sequentiellen Programms auf einem Knoten
- Paralleler Algorithmus + Parallele Implementierung + Parallele Hardware = Paralleles System
- Der Begriff der Skalierbarkeit charakterisiert die Fähigkeit eines Parallelen Systems wachsende Resourcen in Form von Prozessoren P oder der Problemgröße N zu nutzen.

Ziel: Analyse der Skalierbarkeitseigenschaften eines Parallelen Systems

Bewertung von parallelen Algorithmen II

Maße für parallele Algorithmen

- Laufzeit
- Speedup und Effizienz
- Kosten
- Parallelitätsgrad

Bewertung von parallelen Algorithmen III

Definition verschiedener Ausführungszeiten:

- Die **sequentielle Ausführungszeit** $T_S(N)$ bezeichnet die Laufzeit eines sequentiellen Algorithmus zur Lösung des Problems Π bei Eingabegröße N.
- Die **optimale Ausführungszeit** $T_{best}(N)$ charakterisiert die Laufzeit des *besten* (existierenden) sequentiellen Algorithmus zur Lösung des Problems Π bei Eingabegröße N. Dieser hat für fast alle Größen von N den geringsten Zeitbedarf hat.
- Die **parallele Laufzeit** $T_P(N,P)$ beschreibt die Laufzeit des zu untersuchenden parallelen Systems zur Lösung von Π in Abhängigkeit der Eingabegröße N *und* der Prozessorzahl P.

Bewertung paralleler Algorithmen IV

Die Messung dieser Laufzeiten ermöglicht die Definition weiterer Größen:

- **Speedup**

$$S(N,P) = \frac{T_{best}(N)}{T_P(N,P)}. \tag{1}$$

Für alle N und P gilt $S(N,P) \leq P$.
Angenommen es sei $S(N,P) > P$, dann existiert ein sequentielles Programm, welches das parallele Programm simuliert (im Zeitscheibenbetrieb abarbeitet). Dieses hypothetische Programm hätte dann die Laufzeit $PT_P(N,P)$ und es wäre

$$PT_P(N,P) = P\frac{T_{best}(N)}{S(N,P)} < T_{best}(N), \tag{2}$$

was offensichtlich ein Widerspruch ist.

- **Effizienz**

$$E(N,P) = \frac{T_{best}(N)}{PT_P(N,P)}. \tag{3}$$

Es gilt $E(N,P) \leq 1$. Die Effizienz gibt den Anteil des maximal erreichten Speedups an. Man kann auch sagen, daß $E \cdot P$ Prozessoren wirklich an der Lösung von Π arbeiten und der Rest $(1-E)P$ nicht effektiv zur Problemlösung beiträgt.

Bewertung paralleler Algorithmen V

- **Kosten**
  Als Kosten C definiert man das Produkt

$$C(N,P) = PT_P(N,P), \tag{4}$$

  da man diese Rechenzeit im Rechenzentrum bezahlen müßte.
  Man bezeichnet einen Algorithmus als *kostenoptimal*, falls $C(N,P) = const T_{best}(N)$.
  Offensichtlich gilt dann

$$E(N,P) = \frac{T_{best}(N)}{C(N,P)} = 1/const, \tag{5}$$

  die Effizienz bleibt also konstant.

Bewertung paralleler Algorithmen VI

- **Parallelitätsgrad**
  Mit $\Gamma(N)$ bezeichnen wir den *Parallelitätsgrad*. Das ist die maximale Zahl gleichzeitig ausführbarer Operationen im besten sequentiellen Algorithmus.
  - Offensichtlich könnten prinzipiell umso mehr Operationen parallel ausgeführt werden, je mehr Operationen überhaupt auszuführen sind, je größer also N ist. Der Parallelitätsgrad ist also von N abhängig.
  - Andererseits kann der Parallelitätsgrad nicht größer sein als die Zahl der insgesamt auszuführenden Operationen. Da diese Zahl proportional zu $T_S(N)$ ist, können wir sagen, dass

$$\Gamma(N) \leq O(T_S(N)) \tag{6}$$

    gilt.

Bewertung paralleler Algorithmen: Speedup

Wesentlich ist das Verhalten des Speedups $S(N,P)$ eines parallelen Systems in Abhängigkeit von P.

Mit dem zweiten Parameter N hat man die Wahl verschiedener Szenarien.

1. Feste sequentielle Ausführungszeit

- Wir bestimmen N aus der Beziehung

$$T_{best}(N) \stackrel{!}{=} T_{fix} \quad \rightarrow \quad N = N_A \tag{7}$$

  wobei T_{fix} ein Parameter ist. Der skalierte Speedup ist dann

$$S_A(P) = S(N_A, P), \tag{8}$$

  dabei steht A für den Namen *Amdahl*.
- Wie verhält sich der skalierte Speedup?
  Annahme: das parallele Programm entsteht aus dem besten sequentiellen Programm mit sequentiellem Anteil $0 < q < 1$ und einem vollständig parallelisiertem Rest $(1-q)$. Die parallele Laufzeit (für das feste N_A!) ist also

$$T_P = qT_{fix} + (1-q)T_{fix}/P. \tag{9}$$

Bewertung paralleler Algorithmen: Amdahl

Für den Speedup gilt dann

$$S(P) = \frac{T_{fix}}{qT_{fix} + (1-q)T_{fix}/P} = \frac{1}{q + \frac{1-q}{P}} \tag{10}$$

Somit gilt das Gesetz von Amdahl

$$\lim_{P \to \infty} S(P) = 1/q. \tag{11}$$

Konsequenzen:

- Der maximal erreichbare Speedup wird also rein durch den sequentiellen Anteil bestimmt.
- Die Effizienz sinkt stark ab wenn man nahe an den maximalen Speedup herankommen will.
- Diese Erkenntnis führte Ende der 60er Jahre zu einer sehr pessimistischen Einschätzung der Möglichkeiten des parallelen Rechnens.
- Dies hat sich erst geändert als man erkannt hat, dass für die meisten parallelen Algorithmen der sequentielle Anteil q mit steigendem N *abnimmt*.

Der Ausweg aus dem Dilemma besteht also darin mit mehr Prozessoren immer größere Probleme zu lösen!
Wir stellen nun drei Ansätze vor wie man N mit P wachsen lassen kann.

Bewertung paralleler Algorithmen: Gustafson

2. Feste parallele Ausführungszeit

Wir bestimmen N aus der Gleichung

$$T_P(N,P) \stackrel{!}{=} T_{fix} \quad \rightarrow \quad N = N_G(P) \tag{12}$$

für gegebenes T_{fix} und betrachten dann den Speedup

$$S_G(P) = S(N_G(P),P). \tag{13}$$

- Diese Art der Skalierung wird auch „Gustafson Skalierung“ genannt.
- Motivation sind bspw. Anwendungen im Bereich der Wettervorhersage. Hier hat man eine bestimmte Zeit T_{fix} zur Verfügung in der man ein möglichst großes Problem lösen will.

Bewertung paralleler Algorithmen: Speicherlimitierung

3. Fester Speicherbedarf pro Prozessor

Viele Simulationsanwendungen sind speicherbeschränkt, der Speicherbedarf wächst als Funktion $M(N)$. Je nach Speicherkomplexität bestimmt nicht die Rechenzeit sondern der Speicherbedarf welche Probleme man auf einer Maschine noch berechnen kann.

Annahme: Nehmen wir an, dass der Parallelrecher aus P identischen Prozessoren besteht, die jeweils einen Speicher der Grösse M_0 besitzen, so bietet sich die Skalierung

$$M(N) \stackrel{!}{=} PM_0 \quad \rightarrow \quad N = N_M(P) \tag{14}$$

an und wir betrachten

$$S_M(P) = S(N_M(P), P). \tag{15}$$

als skalierten Speedup.

Bewertung paralleler Algorithmen: Effizienzlimitierung

4. Konstante Effizienz

Wir wählen N so, dass die parallele Effizienz konstant bleibt.

Wir fordern

$$E(N,P) \stackrel{!}{=} E_0 \quad \rightarrow \quad N = N_I(P). \tag{16}$$

Dies bezeichnet man als *isoeffiziente Skalierung*. Offensichtlich ist $E(N_I(P),P) = E_0$ also

$$S_I(P) = S(N_I(P),P) = PE_0. \tag{17}$$

Eine isoeffiziente Skalierung ist nicht für alle parallelen Systeme möglich. Man findet nicht unbedingt eine Funktion $N_I(P)$, die (16) identisch erfüllt. Somit kann man andererseits vereinbaren, dass ein System skalierbar ist genau dann wenn eine solche Funktion gefunden werden kann.

Bewertung paralleler Algorithmen: Beispiel I

Zur Vertiefung der Begriffe betrachten wir ein **Beispiel**

- Es sollen N Zahlen auf einem Hypercube mit P Prozessoren addiert werden. Wir gehen folgendermassen vor:
  - Jeder hat N/P Zahlen, die er im ersten Schritt addiert.
  - Diese P Zwischenergebnisse werden dann im Baum addiert.
- Wir erhalten somit die sequentielle Rechenzeit

$$T_{best}(N) = (N-1)t_a \tag{18}$$

- die parallele Rechenzeit beträgt

$$T_P(N,P) = (N/P-1)t_a + \mathrm{ld}\, P t_m, \tag{19}$$

  wobei t_a die Zeit für die Addition zweier Zahlen und t_m die Zeit für den Nachrichtenaustausch ist (wir nehmen an, dass $t_m \gg t_a$).

Bewertung paralleler Algorithmen: Beispiel II

1. Feste sequentielle Ausführungszeit (Amdahl)

Setzen wir $T_{best}(N) = T_{fix}$ so erhalten wir, falls $T_{fix} \gg t_a$, in guter Näherung

$$N_A = T_{fix}/t_a.$$

Für sinnvolle Prozessorzahlen P gilt: $P \leq N_A$.
Für den Speedup erhalten wir im Fall von $N_A/P \gg 1$

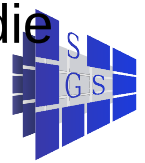
$$S_A(P) = \frac{T_{fix}}{T_{fix}/P + \mathrm{ld}\, Pt_m} = \frac{P}{1 + P\,\mathrm{ld}\, P\frac{t_m}{T_{fix}}}. \tag{20}$$

2. Feste parallele Ausführungszeit (Gustafson)

Hier erhält man

$$\left(\frac{N}{P} - 1\right) t_a + \mathrm{ld}\, Pt_m = T_{fix} \Longrightarrow N_G = P\left(1 + \frac{T_{fix} - \mathrm{ld}\, Pt_m}{t_a}\right). \tag{21}$$

Die maximal nutzbare Prozessorzahl ist wieder beschränkt: $2^{T_{fix}/t_m}$. Ist $\mathrm{ld}\, Pt_m = T_{fix}$, so wird bei ein Einsatz von mehr Prozessoren in jedem Fall die maximal zulässige Rechenzeit überschritten.
Immerhin können wir annehmen, dass $2^{T_{fix}/t_m} \gg T_{fix}/t_a$ gilt.



Bewertung paralleler Algorithmen: Beispiel III

Der skalierte Speedup S_G ist unter der Annahme $N_G(P)/P \gg 1$:

$$S_G(P) = \frac{N_G(P)t_a}{N_G(P)t_a/P + \mathrm{ld}\, P t_m} = \frac{P}{1 + \mathrm{ld}\, P \frac{t_m}{T_{fix}}}. \tag{22}$$

Es gilt $N_G(P) \approx PT_{fix}/t_a$.
Für gleiche Prozessorzahlen ist also S_G größer als S_A.

3. Fester Speicher pro Prozessor (Speicherlimitierung)

Der Speicherbedarf ist $M(N) = N$, damit gilt für $M(N) = M_0 P$ die Skalierung

$$N_M(P) = M_0 P.$$

Wir können nun unbegrenzt viele Prozessoren einsetzen, im Gegenzug steigt die parallele Rechenzeit auch unbegrenzt an. Für den skalierten Speedup bekommen wir:

$$S_M(P) = \frac{N_M(P)t_a}{N_M(P)t_a/P + \mathrm{ld}\, P t_m} = \frac{P}{1 + \mathrm{ld}\, P \frac{t_m}{M_0 t_a}}. \tag{23}$$

Für die Wahl $T_{fix} = M_0 t_a$ ist dies die selbe Formel wie S_G. In beiden Fällen sehen wir, dass die Effizienz mit P fällt.

Bewertung paralleler Algorithmen: Beispiel IV

4. Isoeffiziente Skalierung

Wir wählen N so, dass die Effizienz konstant bleibt, bzw. der Speedup linear wächst, d.h.:

$$S = \frac{P}{1 + \frac{P \operatorname{ld} P}{N} \frac{t_m}{t_a}} \stackrel{!}{=} \frac{P}{1+K} \Longrightarrow N_I(P) = P \operatorname{ld} P \frac{t_m}{K t_a},$$

für ein frei wählbares $K > 0$. Da $N_I(P)$ existiert wird man den Algorithmus als skalierbar bezeichnen. Für den Speedup gilt $S_I = P/(1+K)$.

Isoeffizienzanalyse I

Weitere Formalisierung des Prinzips der isoeffizienten Skalierung

- Ziel Beantwortung von Fragestellungen:
  „Ist dieser Algorithmus zur Matrixmultiplikation auf dem Hypercube besser skalierbar als jener zur schnellen Fouriertransformation auf einem Feld“
- Problemgröße: Parameter N war bisher willkürlich gewählt.
- N bei der Matrixmultiplikation kann sowohl Anzahl der Matrixelemente als auch Elementanzahl pro Zeile bezeichnen.
- In diesem Fall würde sich im ersten Fall $2N^{3/2}t_f$ und im zweiten Fall $2N^3t_f$ als sequentielle Laufzeit ergeben.
- Vergleichbarkeit von Algorithmen erfordert Invarianz des Aufwandsmaß bezüglich der Wahl des Problemgrößenparameters.
- Wir wählen als Maß für den Aufwand W eines (sequentiellen) Algorithmus dessen Ausführungszeit, wir setzen also

$$W = T_{best}(N) \tag{24}$$

  selbst. Diese Ausführungszeit ist wiederum proportional zur Zahl der auszuführenden Operationen in dem Algorithmus.
- Für den Parallelitätsgrad Γ erhält man:

$$\Gamma(W) \leq O(W),$$

  denn es können nicht mehr Operationen parallel ausgeführt werden als überhaupt auszuführen sind.
- Mittels $N = T_{best}^{-1}(W)$ können wir schreiben

$$\tilde{T}_P(W, P) = T_P(T_{best}^{-1}(W), P),$$

  wobei wir jedoch die Tilde im folgenden gleich wieder weglassen werden.

Isoeffizienzanalyse II

Wir definieren den *Overhead* als

$$T_o(W,P) = PT_P(W,P) - W \geq 0. \tag{25}$$

$PT_P(W,P)$ ist die Zeit, die eine Simulation des sequentiellen Programms auf einem Prozessor benötigen würde. Diese ist in jedem Fall nicht kleiner als die beste sequentielle Ausführungszeit W. Der Overhead beinhaltet zusätzliche Rechenzeit aufgrund von Kommunikation, Lastungleichheit und „überflüssigen“ Berechnungen.

Isoeffizienzfunktion Aus dem Overhead erhalten wir

$$T_P(W,P) = \frac{W + T_o(W,P)}{P}.$$

Also erhalten wir für den Speedup

$$S(W,P) = \frac{W}{T_P(W,P)} = P\frac{1}{1+\frac{T_o(W,P)}{W}},$$

bzw. für die Effizienz

$$E(W,P) = \frac{1}{1+\frac{T_o(W,P)}{W}}.$$

Im Sinne einer isoeffizienten Skalierung fragen wir nun: Wie muss W als Funktion von P wachsen damit die Effizienz konstant bleibt. Wegen obiger Formel ist dies dann der Fall wenn $T_o(W,P)/W = K$, mit einer beliebigen Konstanten $K \geq 0$. Die Effizienz ist dann $1/(1+K)$.

Isoeffizienzanalyse III

- Eine Funktion $W_K(P)$ heißt *Isoeffizienzfunktion* falls sie die Gleichung

$$T_o(W_K(P), P) = KW_K(P)$$

  identisch erfüllt.
- Ein paralleles System heißt skalierbar genau dann wenn es eine Isoeffizienzfunktion besitzt.
- Das asymptotische Wachstum von W mit P ist ein Maß für die Skalierbarkeit eines Systems:
  Besitzt etwa ein System S_1 eine Isoeffizienzfunktion $W = O(P^{3/2})$ und ein System S_2 eine Isoeffizienzfunktion $W = O(P^2)$ so ist S_2 *schlechter* skalierbar als S_1.

Isoeffizienzanalyse IV

Wann gibt es eine Isoeffizienzfunktion?

- Wir gehen aus von der Effizienz

$$E(W,P) = \frac{1}{1 + \frac{T_o(W,P)}{W}}.$$

- Effizienz bei *festem* W und wachsendem P. Es gilt für jedes parallele System, dass

$$\lim_{P \to \infty} E(W,P) = 0$$

wie man aus folgender Überlegung sieht: Da W fest ist, ist auch der Parallelitätsgrad fest und damit gibt es eine untere Schranke für die parallele Rechenzeit: $T_P(W,P) \geq T_{min}(W)$, d.h. die Berechnung kann nicht schneller als T_{min} sein, egal wieviele Prozessoren eingesetzt werden. Damit gilt jedoch asymptotisch

$$\frac{T_o(W,P)}{W} \geq \frac{PT_{min}(W) - W}{W} = O(P)$$

und somit geht die Effizienz gegen 0.

Isoeffizienzanalyse V

Betrachten wir nun die Effizienz bei *festem P* und wachsender Arbeit W, so gilt für viele (nicht alle!) parallele Systeme, dass

$$\lim_{W \to \infty} E(W, P) = 1.$$

Offensichtlich bedeutet dies im Hinblick auf die Effizienzformel, dass

$$T_o(W, P)|_{P=const} < O(W) \tag{26}$$

bei festem P wächst also der Overhead weniger als linear mit W. In diesem Fall kann für jedes P ein W gefunden werden so dass eine gewünschte Effizienz erreicht wird. Gleichung (26) sichert also die Existenz einer Isoeffizienzfunktion. Als Beispiel für ein nicht skalierbares System sei die Matrixtransposition genannt. Wir werden später ableiten, dass der Overhead in diesem Fall $T_o(W, P) = O(W, \mathrm{ld}\, P)$ beträgt. Somit existiert keine Isoeffizienzfunktion.

Optimal parallelisierbare Systeme
Wir wollen nun der Frage nachgehen wie Isoeffizienzfunktionen mindestens wachsen müssen. Dazu bemerken wir zunächst, dass

$$T_P(W, P) \geq \frac{W}{\Gamma(W)},$$

denn $\Gamma(W)$ (dimensionslos) ist ja die maximale Zahl gleichzeitig ausführbarer Operationen im sequentiellen Algorithmus bei Aufwand W. Somit ist $W/\Gamma(W)$ eine untere Schranke für die parallele Rechenzeit.

Isoeffizienzanalyse VI

Nun können sicher nicht mehr Operationen parallel ausgeführt werden als überhaupt auszuführen sind, d.h. es gilt $\Gamma(W) \leq O(W)$. Wir wollen ein System als *optimal parallelisierbar* bezeichnen falls

$$\Gamma(W) = cW$$

gilt mit einer Konstanten $c > 0$. Nun gilt

$$T_P(W,P) \geq \frac{W}{\Gamma(W)} = \frac{1}{c},$$

die minimale parallele Rechenzeit bleibt konstant. Für den Overhead erhalten wir dann in diesem Fall

$$T_o(W,P) = PT_P(W,P) - W = P/c - W$$

und somit für die Isoeffizienzfunktion

$$T_o(W,P) = P/c - W \stackrel{!}{=} KW \iff W = \frac{P}{(K+1)c} = \Theta(P).$$

Optimal parallelisierbare Systeme haben somit eine Isoeffizienzfunktion $W = \Theta(P)$. Wir merken uns also, dass Isoeffizienzfunktionen mindestens linear in P wachsen.

Wir werden in den folgenden Kapiteln die Isoeffizienzfunktionen für eine Reihe von Algorithmen bestimmen, deswegen sei hier auf ein ausführliches Beispiel verzichtet.