Kapitel 3
Leistungsmaße für das parallele Rechnen

Dietmar Fey

Zusammenfassung Das folgende Kapitel stellt Leistungsmaße vor, welche die Qualität von Algorithmen im Hinblick auf ihre Eignung für eine parallele Berechnung bewerten. Dadurch ist es möglich, Aussagen darüber zu treffen, wie gut sich bestimmte Algorithmen für eine Parallelisierung in parallelen Rechnerumgebungen wie Cluster-Rechnern und Grids eignen.

3.1 Speed-Up und Effizienz

Eine wichtige Größe in diesem Zusammenhang ist der sog. Speed-Up $S(n,p)$, der die Beschleunigung eines Programms durch Parallelverarbeitung ausdrückt (3.1). Der Speed-Up ist definiert als das Verhältnis der Laufzeit im sequenziellen Fall, $T_s(n)$, zur Laufzeit im parallelen Fall, $T_p(n,p)$. Dabei beschreibt n die Problemgröße, z.B. Vektoren der Dimension n, auf denen die Operationen ausgeführt werden. Die Größe p entspricht der bei der Parallelverarbeitung eingesetzten Anzahl an Prozessoren.

$$S(n,p) = \frac{T_s(n)}{T_p(n,p)} \tag{3.1}$$

Im theoretischen Idealfall ergibt sich für *S(n,p)* ein Speed-Up von p. In der Regel wird die Laufzeit in einer parallelen Umgebung jedoch nicht einfach der sequenziellen Laufzeit geteilt durch die Anzahl der Prozessoren entsprechen. Dies liegt daran, dass Teile der Berechnung eventuell nur sequenziell ausgeführt werden können. Ferner kann bei den parallelen Berechnungen ein zusätzlicher Aufwand (engl.: *Overhead*), bedingt durch eine notwendige Kommunikation und Synchronisation zwischen den Prozessoren, entstehen. Dies führt dazu, dass der Speed-Up nicht optimal mit der Anzahl der Prozessoren skaliert.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik, Lehrstuhl für Rechnerarchitektur
E-mail: dietmar.fey@informatik.uni-erlangen.de

D. Fey (ed.), *Grid-Computing*, eXamen.press, DOI 10.1007/978-3-540-79747-0_3,

Um auch dafür ein Maß zu erhalten, wird der Begriff der Effizienz eingeführt. Die Effizienz eines parallelen Programms ergibt sich aus dem Verhältnis des erzielten Speed-Ups *S(n,p)* und der dafür eingesetzten Anzahl an Prozessoren p (3.2).

$$E(n,p) = \frac{S(n,p)}{p} \tag{3.2}$$

3.2 Das Amdahlsche Gesetz

Dieses Gesetz wurde 1967 von Gene Amdahl publiziert [1]. Es lässt sich wie folgt herleiten. Sei f, $(0 \leq f \leq 1)$, der Anteil an der Berechnung, der sequenziell ausgeführt werden muss, und sei $T_s(n)$ die Gesamtlaufzeit im seriellen Fall. Dann erhält man – unter Berücksichtigung dieses seriellen Anteils an der Gesamtberechnung – für die Gesamtlaufzeit im parallelen Fall, $T_p(n,p)$, die Summe aus dem seriellen Anteil an der ursprünglichen Gesamtlaufzeit plus die Laufzeit, die sich durch den parallelisierbaren Restanteil $(1-f)$ bestimmt und aufgrund der Parallelisierung maximal um den Faktor p verbessert wird (3.3).

$$T_p(n,p) = T_s(n) \cdot \ f + \frac{T_s(n) \cdot (1-f)}{p} \tag{3.3}$$

Setzt man $T_p(n,p)$ in (3.1) ein, so ergibt sich das in (3.4) gezeigte Amdahlsche Gesetz für den Speed-Up $S(n,p)$ beim Einsatz von p Prozessoren.

$$S(n,p) \leq \frac{1}{f + \frac{1-f}{p}} \tag{3.4}$$

Das Gesetz von Amdahl setzt eine Grenze für die Parallelisierbarkeit von Problemen. Selbst beim Einsatz von beliebig vielen Prozessoren wird der Speed-Up immer unter der Grenze $1/f$ fallen. Wie man in Abb. 3.1 sehen kann, geht auch bei einem recht hohen Anteil an Parallelität von 90% die zugehörige Speed-Up-Kurve relativ bald in Sättigung. Selbst bei einem seriellen Anteil von nur 1% wird der Speed-Up nie über den Faktor 100 steigen, womit der Nutzen von noch mehr Prozessoren schnell schwindet. Dies bedeutetet, dass die Effizienz beim Einsatz von immer mehr Prozessoren deutlich abnehmen kann.

Im Gesetz von Amdahl sind noch nicht die Kosten für Kommunikation und Synchronisation berücksichtigt, die durch die parallele Berechnung entstehen. Diese skalieren mit der Anzahl der Prozessoren, so dass der Speed-Up ab einer bestimmten Anzahl von Prozessoren nicht nur gegen eine Asymptote konvergiert, sondern sogar wieder abnimmt (sog. *Ellbogeneffekt*).

Das folgende aus [2] entnommene Beispiel stellt diese Aussage anschaulich dar. Gegeben sei ein Programm, das für die Ein-/Ausgabe einer Datenmenge der Größe n eine Zeit von $(18.000+n)\ \mu s$ benötigt. Dieser Aufwand entspricht zugleich dem seriellen Programm-Anteil $T_s(n)$. Der Anteil des Programms, der parallel ausführbar

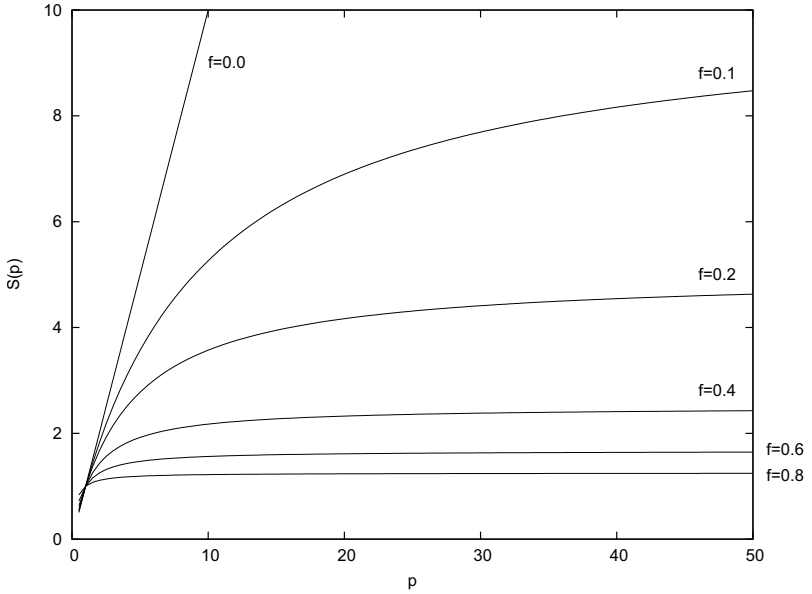

Abb. 3.1 Funktionsverlauf Amdahlsches Gesetz

ist, besitze die von n abhängige Ausführungszeit $(n^2/100)\ \mu s$. Wie hoch ist in diesem Falle der durch eine Parallelisierung maximal erreichbare Speed-Up bei einer Problemgröße von $n = 10.000$? Dazu muss zunächst der serielle Anteil bestimmt werden. Diesen erhält man laut (3.1) aus dem Quotienten des Zeitaufwands für den seriellen Teil dividiert durch den benötigten Zeitaufwand für die gesamte Berechnung, der sich wiederum aus der Summe des seriellen und des parallelen Anteils ergibt. Somit errechnet sich der serielle Anteil f für das Beispiel $n = 10.000$ gemäß (3.5).

$$f = \frac{(18.000 + 10.000)\ \mu sec}{(18.000 + 10.000)\ \mu sec + 10.000^2/100\ \mu sec} = 0.0272 \tag{3.5}$$

Eingesetzt in (3.4) ergibt dies (3.6).

$$S(10.000, p) = \frac{1}{0.0272 + (1 - 0.0272)/p} \tag{3.6}$$

Abb. 3.2 zeigt den Kurvenverlauf des Speed-Ups für dieses Beispiel. Man sieht wie der Kurvenverlauf (Speed-Up ohne Kommunikation) zunächst steil ansteigt und dann abflacht. Im Folgenden wird dieses Beispiel um die Berücksichtigung des Aufwands für die Kommunikation erweitert. Angenommen es existieren in der Anwendung insgesamt $\lceil ldn \rceil$ Kommunikationspunkte, von denen jeder einen Aufwand von $(10.000\lceil ldp \rceil + n/10)\ \mu sec$ verursacht. Somit drückt sich der allgemein zu leistende Zusatzaufwand $T_o(n, p)$ durch (3.7) aus.

$$T_o(n, p) = \lceil ldn \rceil \times (10.000 \times \lceil ldp \rceil + n/10)\ \mu sec \tag{3.7}$$

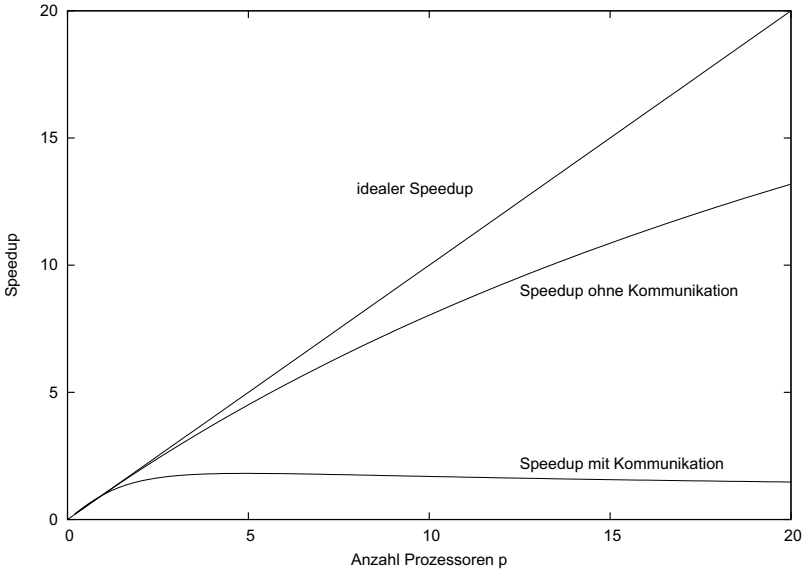

Abb. 3.2 Graphische Darstellung des Ellbogen-Effekts bei Berücksichtigung der Kommunikation. Bei der Kurve für den Fall ohne Berücksichtigung der Kommunikation wurde auf die Darstellung der ganzzahligen Supremum-Funktion aus Gründen der besseren Darstellung verzichtet, d.h. statt $\lceil ldp \rceil$ wurde ldp verwendet

Für die angenommene Datenmenge von $n = 10.000$ ergibt sich damit $(140.000 \times \lceil ldp \rceil + 14.000)\ \mu sec$. Stellt man diesen Aufwand in Relation zum Gesamtberechnungsaufwand ohne Kommunikation, d.h.

$$\frac{T_o(10.000, p)}{((10.000 + 18.000) + 10.000^2/100)} = \frac{(140.000 \times \lceil ldp \rceil + 14.000)\ \mu sec}{1.028.000\ \mu sec}$$
$$= 0.1362 \times \lceil ldp \rceil + 0.0136,$$

erhält man (3.8) als Ausdruck für den von p abhängigen Verlauf des Speed-Ups. Man sieht in Abb. 3.2 (Speed-Up mit Kommunikation) klar, wie in diesem Falle der Speed-Up mit zunehmender Prozessorzahl p einknickt.

$$S(10.000, p) = \frac{1}{0.0272 + (1 - 0.0272)/p + 0.1362\lceil ldp \rceil + 0.0136} \tag{3.8}$$

Die Schlussfolgerungen dieser Zusammenhänge führten dazu, dass die Parallelrechnung für 21 Jahre in eine Nische gedrängt wurde. Erst durch die Arbeiten von Gustafson, Montry und Brenner [3], [4] änderte sich diese Ansicht. Sie zeigten 1988, dass man mit einem System von 1024 Prozessoren durchaus einen Speed-Up von 1000 erreichen kann, wenn das Problem nur groß genug und der serielle Anteil

damit sehr klein ist. Dieser Zusammenhang wird im Gesetz von Gustafson-Barsis formuliert. Heiko Bauke und Stephan Mertens geben dafür in [5] ein anschauliches Beispiel: Wenn ein Maler zum Streichen eines Zimmers eine gewisse Zeit benötigt, so werden 60 Maler nicht ein Sechzigstel dieser Zeit benötigen, da sie sich gegenseitig behindern. Dies entspricht dem Gesetz von Amdahl.

3.3 Der Amdahl-Effekt

Diesen Gedanken kann man wie folgt fortsetzen: Wenn man möchte, dass die 60 Maler dennoch effizient arbeiten, dann gibt man ihnen einfach 60 Zimmer zum Streichen. Sie beenden dann ihre Arbeit in der gleichen Zeit, die ein Maler für eine Wand braucht. Auf die mathematische Analyse bezogen bedeutet dies Folgendes: Dadurch, dass man die Arbeitslast erhöht, zögert man das Abflachen des Kurvenverlaufs der Speed-Up-Kurven gegenüber der Anzahl eingesetzter Prozessoren hinaus und man erhält wieder befriedigende Speed-Up-Werte und damit eine bessere Effizienz. Dieses Hinausschieben der abnehmenden Steigerung des Speed-Ups durch die Erhöhung der Problemlast n bezeichnet man in der Literatur als den *Amdahl-Effekt* [6].

Abb. 3.3 zeigt den Verlauf der Speed-Up-Kurven für das Zahlenbeispiel aus 3.8 erweitert um höhere Problemlasten $n = 20.000$ bzw. $n = 30.000$. Man erkennt deutlich, wie die Kurven für $S(20.000, p)$ und $S(30.000, p)$ ansteigen und der bei $n = 10.000$ bereits frühzeitig einsetzende Ellbogen-Effekt sich für die Situation einer Last von $n = 20.000$ erst ab etwa 25 Prozessoren bemerkbar macht und für eine Last von $n = 30.000$ bis 30 Prozessoren noch gar nicht auswirkt.

Voraussetzung für das Greifen dieses Effektes ist, dass der parallele Anteil über n eine höhere Komplexität aufweist als der serielle Anteil. Diese Aussage lässt sich auf den praktischen Fall verallgemeinern, indem man den Zusatzaufwand für Kommunikation und Synchronisation berücksichtigt. Wenn das zu lösende Problem derart gestaltet ist, dass die parallele Berechnung von höherer Komplexität als der Aufwand für die Kommunikation und die Synchronisation ist, wird bei zunehmend großer Problemlast der Einsatz von vielen Prozessoren, oder allgemein ausgedrückt von zusätzlicher Parallelität, wieder lohnenswert sein. Dies gilt auch im Hinblick auf eine Steigerung der Effizienz. Voraussetzung dafür ist natürlich, dass die Erhöhung der Problemlast Sinn macht.

Auch wenn in der Praxis des täglichen Lebens nicht allzu häufig 60 Wände gestrichen werden müssen, um den Einsatz von 60 Malern zu rechtfertigen, gilt dieser Sachverhalt übertragen auf Anwendungen in Computational Science glücklicherweise nicht. Bisher wünschten sich Anwender aus Computational Science Disziplinen in der Vergangenheit stets mehr Rechenleistung, um beispielsweise ihre Simulationsergebnisse noch genauer berechnen zu können, z.B. durch eine Erhöhung der Anzahl Gleichungen und unbekannter Variablen oder durch feinere Gitterauflösungen in bezüglich Raum und Zeit diskretisierten Differentialgleichungssystemen. Zusätzlich gilt, dass, bedingt durch die Natur der Problemräume, der Aufwand für die Berechnung gegenüber n quadratisch zunimmt, während die zusätzliche

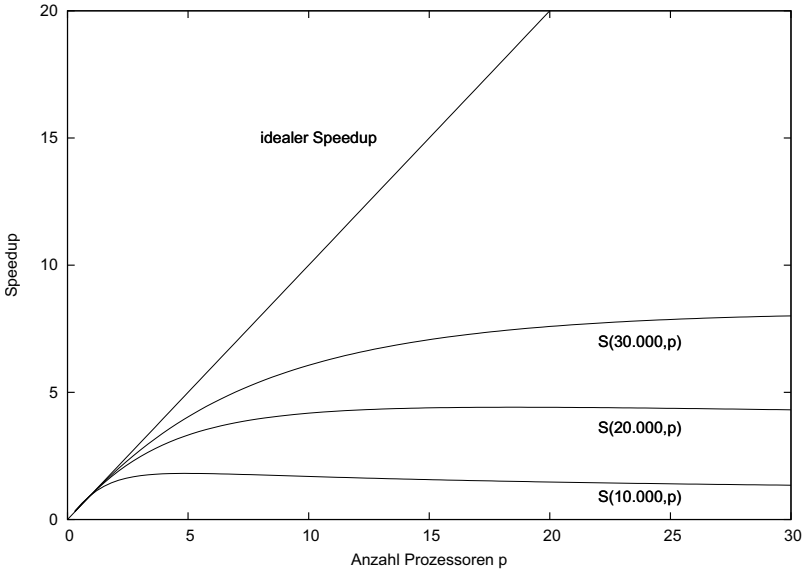

Abb. 3.3 Graphische Darstellung des Amdahl-Effekts

Belastung für die Kommunikation nur linear steigt. Somit gibt es wieder mehr zu rechnen und die Effizienz nimmt zu. Das heißt, dass auch in Grid-Strukturen, die von vornherein einen höheren Kommunikationsaufwand aufweisen, paralleles Rechnen im Sinne des High-Performance-Computing sich lohnen kann, sofern die Problemlast nur hoch genug ist und das Erhöhen der Problemlast hinsichtlich der erwarteten Ergebnisse auch Sinn macht.

3.4 Das Gesetz von Gustafson-Barsis

Im Kern geht es bei diesem Gesetz um eine andere Sichtweise als beim Gesetz von Amdahl. Es wird nicht mehr betrachtet, wie viel schneller ein Programm bei einer größeren Anzahl von Prozessoren ist, sondern wie viel mehr Zeit die Ausführung eines parallelen Programms auf einem Rechner mit nur einer CPU benötigen würde. Anders ausgedrückt, Gustafson-Barsis gehen bei ihrer Analyse nicht wie Amdahl von einem gegebenen seriellen Programm aus, das parallelisiert werden muss, sondern von einem parallelen Programm, bei dem die Last des parallelen Anteils erhöht wird und sich damit auch ein erhöhter Speed-Up beim Einsatz von mehr Prozessoren ergibt.

Die sequenzielle Laufzeit T_1 ergibt sich, wenn man von einem parallelen Programm startet, aus der parallelen Laufzeit T_p nach (3.9). Diese Formel lässt sich wie folgt motivieren: Der serielle Zeitaufwand des parallelen Programms $f_G \times T_p$ muss selbstverständlich auch auf einem seriellen Rechner erbracht werden, wobei f_G dem seriellen Anteil im parallelen Programm entspricht. Hinzu kommt noch der parallele Zeitaufwand $(1 - f_G) \times T_p$, der auf einem seriellen Rechner p-mal so viel Zeit in Anspruch nimmt.

$$T_1 = f_G \cdot T_p + p \cdot (1 - f_G) \cdot T_p \tag{3.9}$$

Daraus ergibt sich mit $S(p) = T_1/T_p$ der Speed-Up nach Gustafson-Barsis (3.10).

$$S(p) = f_G + (1 - f_G) \cdot p = p - (p - 1) \cdot f_G \tag{3.10}$$

Vergleicht man beide Definitionen des Speed-Ups, so sieht man, dass der Speed-Up bei Amdahl durch den seriellen Anteil begrenzt, bei Gustafson-Barsis aber im Prinzip beliebig hoch (bei unbegrenzter Anzahl von CPUs) sein kann. Dies ergibt sich aus der veränderten Betrachtungsweise. Der serielle Anteil bei Amdahl ist der serielle Anteil an der Laufzeit T_1, während der serielle Anteil bei Gustafson der serielle Anteil der Laufzeit T_p ist. Der Speed-Up bei Gustafson-Barsis wird deshalb auch skalierter Speed-Up genannt, da die Problemgröße mit der Anzahl der Prozessoren skalieren sollte.

3.5 Die Karp-Flatt-Metrik

Gemein ist sowohl dem Ansatz von Amdahl als auch dem nach Gustafson-Barsis, dass die Eignung eines Programms zur Ausführung in einer massiv-parallelen Umgebung stark vom Anteil der sequenziell auszuführenden Berechnung abhängt. Beide Ansätze vernachlässigen jedoch den Zusatzaufwand für Kommunikation und Synchronisation. Die Vernachlässigung dieses Zusatzaufwands hat zur Folge, dass die ermittelten Werte für den Speed-Up nach Amdahl bzw. für den Speed-Up nach Gustafson-Barsis leicht überschätzt werden. Karp und Flatt schlugen daher eine andere Metrik vor, die den Zusatzaufwand explizit berücksichtigt [7]. In ihrer Metrik stellen sie den Zusatzaufwand in ein Verhältnis zu dem parallelen Anteil eines Programms.

Der tatsächliche serielle Anteil, in diesem Falle exakt der nicht über die p Prozessoren parallelisierbare Anteil, besteht aus der Summe des inhärent seriellen Anteils des Programms, $\sigma(n)$, und den Zusatzkosten, $\kappa(n,p)$, in Relation zu der Laufzeit des seriellen Programms, die sich aus der Summe des seriellen Anteils, $\sigma(n)$, und des parallelen Anteils, $\phi(n)$, ergibt (3.11). In diesem Falle werden bei $\sigma(n)$ und $\phi(n)$ deren Abhängigkeiten von der Größe der Problemlast n berücksichtigt, bei $\kappa(n,p)$ kommt zusätzlich noch eine Abhängigkeit von der Anzahl der Prozessoren hinzu, die den Zusatzaufwand für Kommunikation und Synchronisation mitbestimmt.

$$e = \frac{\sigma(n) + \kappa(n,p)}{\sigma(n) + \phi(n)} \tag{3.11}$$

Der Wert e muss jedoch nicht ausschließlich über (3.11) bestimmt werden. Viel einfacher ist es, e dadurch zu bestimmen, dass man das Gesetz von Amdahl einfach nach dem sequentiellen Anteil auflöst. In diesem Fall erhält man die Karp-Flatt-Metrik e nach (3.12).

$$
\begin{aligned}
1/S(p) = f + \frac{1-f}{p} = \frac{f \times (p-1) + 1}{p} \Rightarrow \\
\frac{p}{S(p)} - 1 = f \times (p-1) \Rightarrow \\
f = \frac{\frac{p}{S(p)} - 1}{p-1} \Rightarrow \\
f = \frac{\frac{1}{S(p)} - \frac{1}{p}}{1 - \frac{1}{p}} = e
\end{aligned}
\tag{3.12}
$$

Dies erlaubt den sequentiellen Anteil einschließlich des Zusatzaufwands empirisch zu bestimmen. Dies geschieht über Laufzeit-Messungen an einem vorhandenen parallelen Programm. Man spricht in diesem Zusammenhang auch von dem *experimentell bestimmten seriellen Anteil e*. Um den experimentellen seriellen Anteil zu bestimmen, wird der Speed-Up des Programms für mehrere Werte von p, also für eine verschiedene Anzahl von Prozessoren ermittelt. Setzt man die verschiedenen gemessenen Speed-Up-Werte und die zugehörige Anzahl an Prozessoren in die Formel (3.12) ein, erhält man den sequentiellen Anteil e.

Durch die Bestimmung von e für mehrere Werte von p, lässt sich untersuchen, ob e von p unabhängig ist oder nicht. Ist e unabhängig von p, so spielen Kommunikation und Synchronisation keine Rolle bei der Ausführungszeit des Programms. Das heißt, ist man mit dem Verlauf des Speed-Ups beim Einsatz mehrerer Prozessoren nicht zufrieden, liegt es in diesem Falle nicht an den durch die Parallelisierung entstandenen Zusatzkosten. Vielmehr ist dann die Situation gegeben, dass quasi die gesamte Algorithmen-inhärente Parallelität herausgezogen und ein weiterer Speed-Up daher nicht mehr möglich ist. Mit anderen Worten ausgedrückt, der eigentliche serielle Anteil des Algorithmus ist zu hoch.

Wächst e allerdings mit der Anzahl der Prozessoren, so gibt es einen parallelen Overhead und der Speed-Up ist geringer, als er sich nach dem Gesetz von Amdahl oder Gustafson-Barsis ergeben würde. In diesem Falle ist für die Stagnation oder für den Rückgang des Speed-Ups wirklich der durch die Parallelisierung bedingte Zusatzaufwand verantwortlich. Hier muss bei einer Optimierung des parallelen Algorithmus angesetzt werden. Beispielsweise sollte überprüft werden, ob sich Berechnung und Kommunikation zeitgleich ausführen lassen.

Wir werden im Verlauf der Ausführungen in Kapitel 21.3 über die Parallelisierung der Simulation von Zellulären Automaten auf das Karp-Flatt-Maß wieder zurückgreifen, um Aussagen zu treffen, ob sich der Übergang von einer Berechnung in einem Cluster-Rechner zu einem z.B. aus Multi-Clustern bestehenden Grid lohnt.

3.6 Die Isoeffizienz-Funktion

Mit steigender Anzahl der verwendeten Prozessoren wird durch den sequenziellen Anteil f die Effizienz der parallelen Verarbeitung eines Programms abnehmen (s. Abb. 3.2), wenn die Problemgröße konstant bleibt. Steigt hingegen die Problemgröße n bei konstanter Anzahl von Prozessoren, so wird im Regelfall der sequentielle Anteil an der Gesamtrechenlast kleiner und die Effizienz steigt. Dies ist die Grundlage des skalierten Speed-Ups. Die Isoeffizienz-Funktion [8] gibt an, wie stark die Problemgröße wachsen muss, um bei einem Mehr an Parallelisierung, d.h. einer steigenden Anzahl von Prozessoren, die Effizienz konstant zu halten. Dabei geht man wie beim Ansatz von Gustafson-Barsis von einem parallelen Programm aus. Von Interesse ist also die Laufzeit eines parallelen Programms $T(n,p)$ in Abhängigkeit von der Problemgröße n und der Anzahl der Prozessoren p. Wie bereits erwähnt, bestimmt sich die Laufzeit $T(n,p)$ aus der Zeit, die für den sequentiellen Teil der Berechnung benötigt wird, $T_s(n,p)$, der Zeit für den parallelen Anteil, $T_p(n)$, und dem Zusatzaufwand, $T_o(n,p)$, der durch Kommunikation und Synchronisation entsteht (3.13).

$$T(n,p) = T_s(n,p) + \frac{T_p(n)}{p} + T_o(n,p) \tag{3.13}$$

Unter Verwendung von $T(n,p)$ nach (3.13) lässt sich die Effizienz gemäß (3.14) ausdrücken.

$$E(n,p) = \frac{T(n,1)}{p \cdot T(n,p)} \tag{3.14}$$

Setzt man den Ausdruck (3.13) in (3.14) ein, lässt sich die Effizienz gemäß (3.15) formulieren.

$$E(n,p) = \frac{T_{seq}(n) + T_{par}(n)}{p \cdot T_s(n,p) + T_{par}(n) + p \cdot T_{cost}(n,p)} \tag{3.15}$$

Weiterhin gilt (3.16) für $T(n,1)$, entsprechend der Laufzeit auf einem Prozessor.

$$T(n,1) = T_{seq}(n) + T_{par}(n) \tag{3.16}$$

Daraus ergibt sich (3.17). Dabei entspricht $T_{ges}(n,p)$ dem gesamten im parallelen Algorithmus zusätzlich zum rein seriellen Anteil zu leistenden Aufwand.

Dieser ergibt sich aus den um den Faktor $p-1$ größeren seriellen Anteil $T_s(n,p)$, der in den p Prozessoren ausgeführt werden muss, plus dem im parallelen Fall zu leistenden Aufwand für Kommunikation und Synchronisation $T_o(n,p)$.

$$\begin{aligned} E(n,p) = \frac{T(n,1)}{T(n,1)+(p-1)\cdot T_s(n,p)+T_o(n,p)} \Rightarrow \\ T(n,1) = E(n,p)\cdot(T(n,1)+T_{ges}(n,p)) \Rightarrow \\ T(n,1)\cdot(1-E(n,p)) = E(n,p)\cdot T_{ges}(n,p) \Rightarrow \\ T(n,1) = \frac{E(n,p)}{1-E(n,p)}\cdot T_{ges}(n,p) \end{aligned} \tag{3.17}$$

Wenn man somit eine konstante Effizienz wünscht, muss der Ausdruck $\frac{E(n,p)}{1-E(n,p)}$ konstant bleiben und somit lässt sich (3.17) zu (3.18) vereinfachen.

$$T(n,1) \geq C \cdot T_{ges}(n,p) \tag{3.18}$$

Diese Gleichung wird als Isoeffizienz-Funktion bezeichnet. Der konstante Faktor C ist spezifisch für eine bestimmte Problemklasse. Damit die Effizienz beim Einsatz von mehr Prozessoren mindestens gleich bleibt, muss die Last n soweit erhöht werden, dass der dabei entstehende sequentielle Zeitaufwand $T(n,1)$ höchstens um den gleichen Faktor zunimmt wie der bei einer auf p Prozessoren erweiterten Parallelisierung zusätzlich zum sequentiellen Rechenaufwand zu leistende gesamte Restaufwand $T_{ges}(n,p)$.

3.7 Weiterführende Literatur

Weiterführende Literatur insbesondere zu dem Thema Isoeffizienz findet sich in [2] und [9]. Dort werden auch für verschiedene Anwendungsklassen Asymptoten für die Konstante C gegenüber einer steigenden Prozessorzahl p aufgezeigt. Eine interessante Neubewertung des Gesetzes von Amdahl unter Berücksichtigung von Multikern-Prozessorarchitekturen zeigt [10].

Literaturverzeichnis

1. G.M. Amdahl, „Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities“ in *Proc. Am. Federation of Information Processing Societies Conf.*, AFIPS Press, 1967, pp. 483-485.
2. M.J. Quinn, *Parallel Programming in C with MPI and OpenMP*, (McGraw-Hill, New York, 2003).
3. J.L. Gustafson, „Reevaluating Amdahl‘s Law“, Comm. ACM, May 1988, pp. 532-533.
4. J.L. Gustafson, G.R. Montry, and R.E. Brenner, „Development of Parallel Methods for a 1024-Processor Hypercube“, *SIAM J. Scientific and Statistical Computing*, Vol. 9, No. 4, July 1988, pp. 532- 533.
5. H. Bauke, S. Mertens, *Cluster Computing*, (Springer, Berlin, Heidelberg, 2006)
6. S.E. Goodman, S.T. Hedetniemi, *Introduction to the Design and Analysis of Algorithms*, (McGraw-Hill, 1977).
7. A.H. Karp, H.P. Flatt, „Measuring parallel processor performance“, Communications of the ACM **33(5)** May 1990, pp. 539-543.
8. A.Y. Grama, A. Gupta, V. Kumar „Isoefficiency Function: A Scalibility Metric for Parallel Algorithms and Architectures“, IEEE Parallel and Distributed Technology, **1(3):12-21**, August 1993.
9. A. Grama, A. Gupta, G. Karypis, V. Kumar, 2nd edn. *Introduction to Parallel Computing* (Addision-Wesley, Harlow, England, 2003).
10. M.D. Hill, M.R. Marty, „Amdahl‘s Law in the Multicore Era“, *Computer*, pp. 33-38, July 2008.