Seminarvortrag
Amdahlsches und Gustafsonsches Gesetz

Volker Grabsch Yves Radunz

26. Mai 2008

Veröffentlicht unter

`http://www.profv.de/uni/`

Lizenziert unter

Inhaltsverzeichnis

1 Einleitung

Es gibt viele theoretische Untersuchungen zur parallelen Programmierung. Doch zwei von ihnen haben besonderes Aufsehen erregt: das Gesetz von Amdahl [Amd67] und das Gesetz von Gustafson [Gus88].

Während das Amdahlsche Gesetz gern als Totschlagargument gegen massive Parallelisierung angeführt wurde, sorgte das Gustafsonsche Gesetz wieder für mehr Optimismus. Obwohl es sehr einfache Theorien sind, die eng miteinander zusammenhängen, ranken sich um sie viele Missverständnisse und hitzige Diskussionen. Es gab sogar Versuche zur „Schlichtung“. [Shi96]

Wir werden uns in diesem Vortrag beide Modelle und ihre Folgerungen genauer ansehen. Dabei versuchen wir, ein wenig Klarheit in dieses verwirrende und heiß diskutierte Thema hinein zu bringen.

2 Das Amdahlsche Gesetz

Das Amdahlsche Gesetz [Amd67] beschäftigt sich mit der Frage, wie stark man ein bestimmtes Programm durch Parallelisierung beschleunigen kann. In diesem Abschnitt werden wir uns das Gesetz an sich ansehen, einfache Folgerungen ziehen und zum Schluss die Grenzen des Modells beleuchten.

2.1 Amdahlsches Gesetz

Das Amdahlsche Gesetz geht von einem sehr einfachen Modell des zu parallelsierenden Programmcodes aus. Es nimmt an, dass jedes Programm zu einem gewissen Teil beliebig stark parallelisierbar ist, und zu einem gewissen Teil sequenziell ablaufen muss, d.h. überhaupt nicht parallelisiert werden kann.

Wir starten unser Programm hypothetisch auf einem System mit nur einem Prozessor, und normieren seine Laufzeit auf 1. Dann schauen wir, wieviel Zeit das Programm im parallelisierbaren Teil verbringt, und bezeichnen diesen Anteil mit P. Der sequenzielle Teil hat dann die Laufzeit $(1 - P)$.

Wie schnell läuft das Programm auf einem System mit N Prozessoren? Die Laufzeit des sequenziellen Teils ändert sich nicht. Der parallelisierbare Teil jedoch wird optimal auf alle Prozessoren verteilt und läuft daher N-mal so schnell. Als neue Laufzeit ergibt sich:

$$\underbrace{(1-P)}_{\text{sequenziell}} \quad + \quad \underbrace{\frac{P}{N}}_{\text{parallel}}$$

Wieviel schneller ist das Programm nun geworden? Ganz einfach:

$$\text{Zeitgewinn} = \frac{\text{ursprüngliche Laufzeit}}{\text{neue Laufzeit}} = \frac{1}{(1-P) + \frac{P}{N}}$$

Dies ist das *Amdahlsche Gesetz*. Dabei ist N die Anzahl der Prozessoren und P der Laufzeit-Anteil des parallelisierbaren Programmcodes.

Ein Zeitgewinn von 2 bedeutet, dass das Programm nun doppelt so schnell ist wie vorher. In der Literatur findet man an Stelle des Begriffs „Zeitgewinn“ auch „speedup“ oder „Geschwindigkeitsgewinn“.

2.2 Höchstgeschwindigkeit

Kommen wir nun zur ersten und wichtigsten Konsequenz des Amdahlschen Gesetzes: Ein Programm kann durch Parallelisierung nur bis zu einer bestimmtem Grenze hin beschleunigt werden. Das ist ein harter Schlag! Es ist die Hauptursache für den Pessimismus gegenüber massiver Parallelisierung. Genauer gesagt gilt folgende Ungleichung:

$$\text{Zeitgewinn} < \frac{1}{1-P}$$

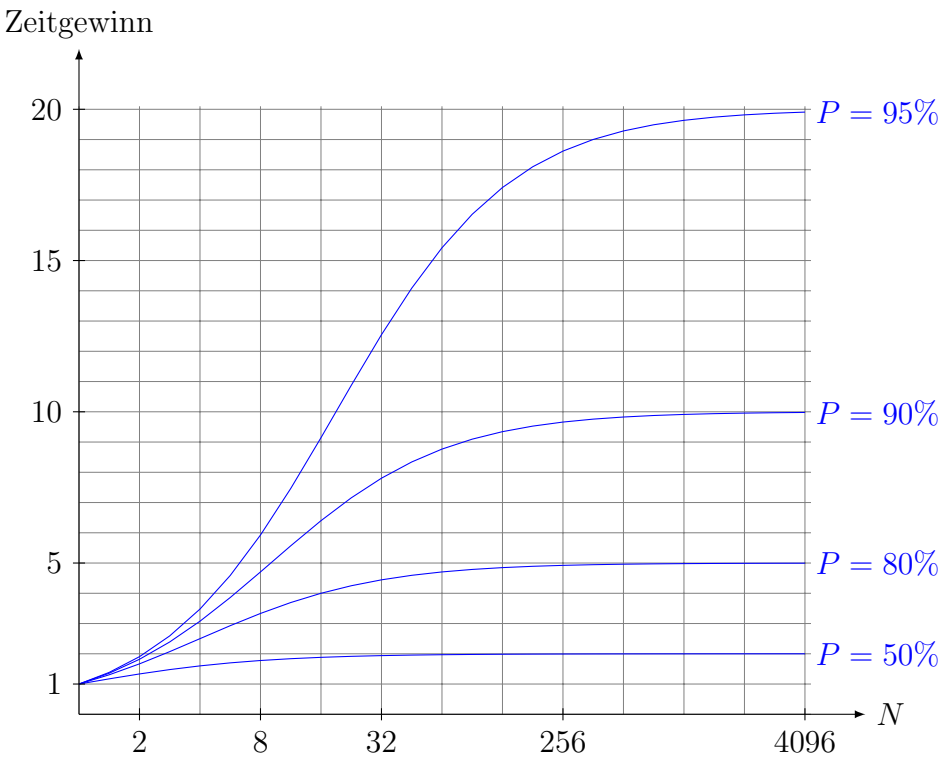
Diese Ungleichung folgt direkt aus dem Modell, denn $(1-P)$ ist per Definition genau der Laufzeit-Anteil des Programms, der bei Parallelisierung nicht verschwindet.

Wir können die Grenze auch dadurch herleiten, dass wir im Amdahlschen Gesetz die Anzahl N der Prozessoren gegen ∞ laufen lassen:

$$\lim_{N\to\infty} \text{Zeitgewinn} = \lim_{N\to\infty} \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{1-P}$$

Das heißt, selbst mit tausenden Prozessoren können wir ein zu 95% parallelisierbares Programm höchstens um den Faktor 20 beschleunigen. ($\frac{1}{1-P} = \frac{1}{1-0{,}95} = 20$)

Das folgende Diagramm veranschaulicht dies für verschieden gut parallelisierbare Programme. Dabei ist die N-Achse logarithmisch eingeteilt, d.h. die Anzahl der Prozessoren wird immer verdoppelt:



Weitere Anwendungen des Amdahlschen Gesetzes besprechen wir in Kapitel 3.

2.3 Grenzen des Amdahlschen Gesetzes

So schön und einfach das Amdahlsche Gesetz auch ist, es hat natürlich seine Grenzen. Folgende Aspekte der Parallelisierung werden vom ihm nicht berücksichtigt:

1. Je mehr Prozessoren an einem Problem arbeiten, umso aufwändiger wird die **Kommunikation** zwischen ihnen. Dieser zusätzliche Verwaltungsaufwand senkt die zu erwartenden Zeitgewinne durch Parallelisierung nochmals.

   Wir werden auf diesen Aspekt nicht weiter eingehen, da er den ohnehin schon vorhandenen Pessimismus nur noch schlimmer machen würde.

2. Neben der Anzahl der Prozessoren wachsen auch **weitere Resourcen** mit der Parallelisierung. So bringt jeder zusätzliche Prozessor normalerweise auch Cache mit, sodass insgesamt für die Berechnungen viel mehr Prozessor-Cache zu Verfügung steht.

   In einem Rechencluster kommt mit jedem neuen Prozessor sogar ein kompletter Computer hinzu, sodass mehr RAM, mehr Festplattenspeicher, u.s.w. zu Verfügung steht.

   Wir werden in Kapitel 5.3 hierauf zurück kommen.

3. Oft wollen wir unsere Ergebnisse gar nicht schneller haben, sondern lieber **größere Probleme** in der bisherigen Zeit berechnen.

   Diese Idee führt uns direkt zum Gustafsonschen Gesetz, welches wir in Kapitel 4 besprechen werden.

3 Ausweitungen des Amdahlschen Gesetzes

Bisher hatten wir uns nur mit dem Effekt der Parallelisierung eines Programmabschnitts auf die Gesamtlaufzeit befasst. An dieser Stelle sollen daher einige Bemerkungen zu der Anwendung des Amdalschen Gesetzes in anderen Zusammenhängen folgen. [Wik08]

3.1 Übertragung auf sequenzielle Programme

Betrachten wir einmal ein sequenzielles Programm, welches sich aus mehreren Abschnitten zusammensetzt. Jetzt stellt sich die Frage, wie stark die Gesamtlaufzeit des Programms verkürzt würde, wenn wir die Ausführung eines der Abschnitte durch Optimierung um einen bestimmten Faktor beschleunigten.

Hierzu wählen wir P als den ursprünglichen (d.h. vor der Optimierung) Anteil des optimierten Programmteils an der Gesamtlaufzeit. Für N können wir nun an Stelle der Anzahl der verwendeten Prozessoren den Faktor einsetzen, um welchen die Ausführung dieses Programmfragments beschleunigt wird.

Damit ergibt sich eine neue Laufzeit von $(1 - P) + \frac{P}{N}$, da der nicht optimierte Abschnitt weiterhin mit der einfachen Geschwindigkeit ausgeführt wird und damit einen Zeitaufwand von $1-P$ hat. Der optimierte Abschnitt wird nun N-mal schneller ausgeführt und benötigt somit nur noch eine Zeit von $\frac{P}{N}$.

Der Geschwindigkeitsgewinn in der Ausführung des gesamten Programm ergibt sich also wieder durch den bereits bekannten Quotienten aus altem Zeitaufwand und neuem Zeitaufwand:

$$\frac{1}{(1-P)+\frac{P}{N}}$$

Oftmals stellt sich das Problem, welcher Teil des Programms eher zu optimieren ist, da mit der Optimierung schließlich auch ein Mehraufwand bei der Implementierung und/oder Übersetzung verbunden ist. Mit Hilfe des Amdahlschen Gesetzes lässt sich der erwartete Zeitgewinn für das gesamte Programm nun recht leicht abschätzen. Man benötigt dafür wie bereits erwähnt nur den Anteil des ggf. zu optimierenden Programmteils an der gesamten Laufzeit des Programms und den erwarteten Geschwindigkeitsgewinn innerhalb dieses Programmteils nach der Optimierung.

Beispielsweise wählen wir ein Programm, welches aus zwei Teilen besteht. Nennen wir diese einmal A und B. Programmteil A habe im unoptimierten Zustand einen Anteil von 75% an der Gesamtlaufzeit des Programms, wohingegen B nur für die übrigen 25% verantwortlich ist. Nun mag es mit einem vertretbaren Aufwand möglich sein, höchstens einen der beiden Abschnitte zu verbessern. Insbesondere sei für den ersten Abschnitt nur ein Verfahren bekannt, welches die Ausführungszeit lediglich halbiert. Für den zweiten Abschnitt existiere hingegen eine Möglichkeit, die Ausführung auf das Zehnfache zu beschleunigen.

Obwohl der Abschnitt B viel besser optimiert werden kann, sieht man schnell, dass eine Optimierung von A einen viel größeren Geschwindigkeitsvorteil bringt:

Dies lässt sich auch rechnerisch nachvollziehen:

1. Optimierung von Teil A:

   In diesem Fall gilt $P = \frac{3}{4}$ und $N = 2$, womit sich eine Beschleunigung des gesamten Programmablaufs ergibt von:

   $$\frac{1}{(1-\frac{3}{4})+\frac{\frac{3}{4}}{2}} = \frac{1}{\frac{1}{4}+\frac{3}{8}} = \frac{1}{\frac{5}{8}} = \frac{8}{5} = 1.6$$

2. Optimierung von Teil B:

   Hier haben wir $P = \frac{1}{4}$ und $N = 10$. Der erwartete Zeitgewinn liegt in diesem Fall jedoch nur bei

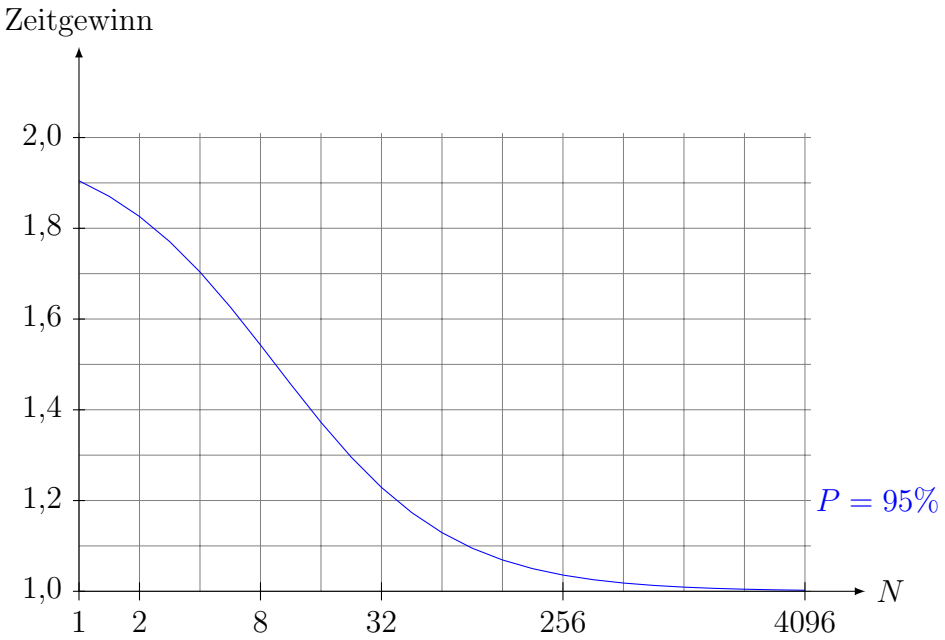
   $$\frac{1}{(1-\frac{1}{4})+\frac{\frac{1}{4}}{10}} = \frac{1}{\frac{3}{4}+\frac{1}{40}} = \frac{1}{\frac{31}{40}} = \frac{40}{31} < 1.3$$

   was viel kleiner ist als der Zeitgewinn im ersten Fall.

3.2 Gesetz des abnehmenden Gewinns

Eine weitere Folgerung aus dem Amdahlschen Gesetz ist die Tatsache, dass die Hinzunahme eines weiteren Prozessors nur noch einen sehr geringen Geschwindigkeitsvorteil bringt, wenn bereits eine große Anzahl an Prozessoren verwendet wird. Selbst eine *Verdoppelung* der Prozessorenanzahl N hilft für große Werte von N nur noch sehr wenig.

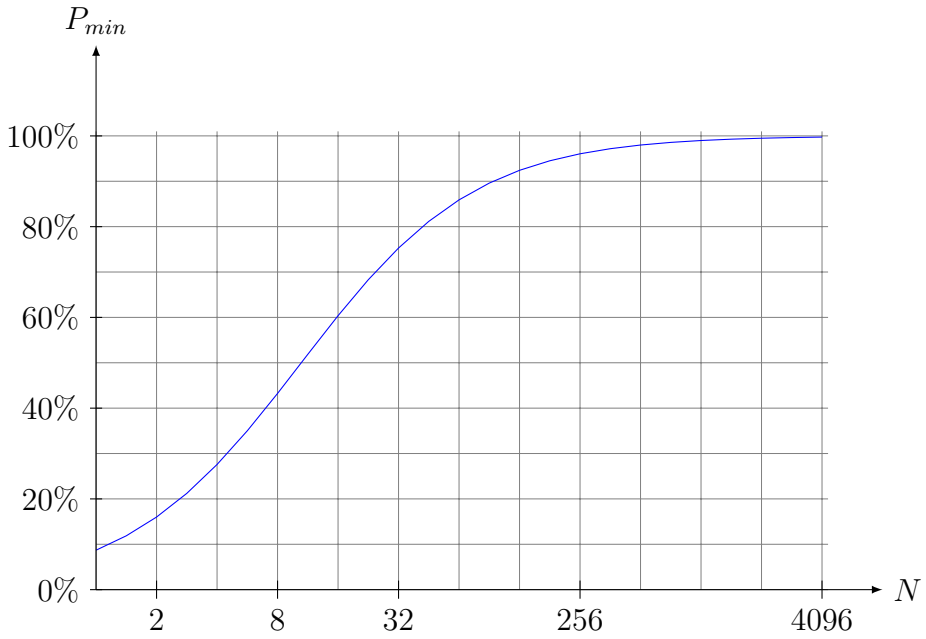
In der folgenden Graphik ist der erwartete Geschwindigkeitsgewinn bei einem parallelisierbaren Anteil von 95% und einer Verdoppelung der Prozessorenanzahl von N auf $2N$ dargestellt.



3.3 Höchster vertretbarer Aufwand

Wenn bei einer ausreichend großen Prozessorenanzahl eine weitere Verdoppelung der Prozessorenanzahl keinen großen Zeitgewinn mehr erhoffen lässt, so stellt sich die Frage, ab welcher Prozessorenanzahl dieser Effekt eintritt. Selbstverständlich hängt diese Grenze von der Größe des parallelisierbaren Anteils des Programms ab.

Wenn man eine Leistungssteigerung um mindestens 5% als lohnend ansieht, so erkennt man, dass ab etwa 1000 Prozessoren nahezu das gesamte Programm parallelisierbar sein müsste, damit eine Verdoppelung von N noch einen lohnenenden Geschwindigkeitszuwachs verursacht:



4 Das Gustafsonsche Gesetz

Das Gustafsonsche Gesetz [Gus88] entstand als Gegenstück zum Amdahlschen Gesetz. Es zeigt, dass sich massive Parallelisierung eben doch lohnt! Zwar können wir nicht beliebig schneller werden, aber in gleicher Zeit beliebig große Probleme lösen. Gustafson widerspricht damit Amdahl in keiner Weise, sondern zeigt lediglich einen Aspekt auf, den Amdahl übersehen hat.

Leider formulierte Gustafson sein Gesetz unzulässigerweise als „Zeitgewinn“. So provozierte er Missverständnisse, die bis hin zu einem vermeintlichen Widerspruch zum Amdahlschen Gesetz reichten.

Um möglichst viel Klarheit zu schaffen, werden wir Schritt für Schritt die einzelnen Gedanken von Gustafson nachvollziehen. Dabei werden wir all seine guten Ideen zusammentragen und in nützliche Formeln gießen. Erst zum Schluss unternehmen wir den fatalen Schritt, daraus einen fiktiven Zeitgewinn abzuleiten, und landen beim Gustafsonschen Gesetz.

4.1 Einbeziehung der Problemgröße (nach Gustafson)

Die wichtigste Idee von Gustafson ist die Einbeziehung der *Problemgröße* in das Modell. Genauer gesagt geht es um die Eingangsdaten, deren Größe wir mit K bezeichnen wollen. Das ursprüngliche Problem habe die Größe $K = 1$.

Die Laufzeit des Programms bei Problemgröße $K = 1$ auf einer 1-Prozessor-Maschine normieren wir wieder auf 1. Der parallelisiere Programmteil habe bei Problemgröße $K = 1$ einen Anteil von P' an der Laufzeit.

Wie ändert sich die Laufzeit, wenn wir die Problemgröße erhöhen? Gustafson argumentiert, dass normalerweise nur der parallelisierbare Programmteil mehr zu tun bekommt. Denn dieser besteht typischerweise aus vielen Vektor-Operationen, deren Dimension sich direkt aus der Größe der Eingabedaten ergibt. Der sequenzielle Programmteil hingegen besteht überwiegend aus Initialisierungs-Schritten, die

unabhängig von der Problemgröße sind. Dies führt zu folgender Laufzeit:

$$\underbrace{(1-P')}_{\text{sequenziell}} \quad + \quad \underbrace{K \cdot P'}_{\text{parallel}}$$

Wie beim Amdahlschen Gesetz stellen wir uns nun die Frage, wieviel schneller das Programm auf einem N-Prozessor-System läuft. Dort arbeitet der sequenzielle Programmteil unverändert, während der parallelisierbare Programmteil N-mal so schnell wird. Als neue Laufzeit ergibt sich:

$$\underbrace{(1-P')}_{\text{sequenziell}} \quad + \quad \underbrace{\frac{K \cdot P'}{N}}_{\text{parallel}}$$

Der Zeitgewinn lautet entsprechend:

$$\text{Zeitgewinn} = \frac{(1-P') + K \cdot P'}{(1-P') + \frac{K \cdot P'}{N}}$$

Wir haben somit den Zeitgewinn in Abhängigkeit von der Problemgröße ermittelt. Diese Formel wurde von Gustafson leider nicht herausgearbeitet, entspricht aber genau seinem Modell.

Das Amdahlsche Gesetz finden wir in dieser Formel als Spezialfall $K = 1$ wieder. Umgekehrt können wir auch die neue Formel aus dem Amdahlschen Gesetz herleiten. Dazu müssen wir für lediglich für P den parallelisierbaren Anteil bei Problemgröße K einsetzen:

$$P = \frac{\text{parallelisierbarer Teil der Laufzeit}}{\text{gesamte Laufzeit}} = \frac{K \cdot P'}{(1-P') + K \cdot P'}$$

Es ergibt sich:

$$\begin{aligned}
\text{Zeitgewinn} &= \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{\left(1 - \frac{K \cdot P'}{(1-P') + K \cdot P'}\right) + \frac{\frac{K \cdot P'}{(1-P') + K \cdot P'}}{N}} \\
&= \frac{(1-P') + K \cdot P'}{\left((1-P') + K \cdot P' - K \cdot P'\right) + \frac{K \cdot P'}{N}} = \frac{(1-P') + K \cdot P'}{(1-P') + \frac{K \cdot P'}{N}}
\end{aligned}$$

Wir erhalten die selbe Formel.

4.2 Gustafsonsches Gesetz – Gute Idee ...

Die zweite gute Idee, die Gustafson einbringt, ist folgende: Wenn wir bessere Rechentechnik erhalten, freuen wir uns, dass unsere alten Programme schneller laufen. Doch ihre Laufzeit war gar nicht so schlecht. Sie war vielleicht hart an der Grenze, aber immer noch in Ordnung. Also lassen wir auf dem neuen System immer größere Probleme rechnen, bis wir wieder die Grenze stoßen, an die ursprüngliche Laufzeit. Gustafson hat es so formuliert:

> „Hence, it may be most realistic to assume that run time, not problem size, is constant.“

Die Frage ist also, welche Problemgröße wir in der ursprünglichen Laufzeit 1 auf einem N-Prozessor-System handhaben können:

$$\begin{aligned} \text{Laufzeit} &= 1 \\ (1 - P') + \frac{K \cdot P'}{N} &= 1 \\ \frac{K \cdot P'}{N} &= P' \\ K &= N \end{aligned}$$

Ein erstaunliches Ergebnis! Unserem Modell zufolge können wir stets die N-fache Problemgröße berechnen, unabhängig vom parallelisierbaren Anteil P'.

4.3 ... aber schlechte Umsetzung

Schauen wir uns die bisherigen Erkenntnisse noch einmal an:

$$\begin{aligned} \text{Zeitgewinn} &= \frac{1}{(1 - P) + \frac{P}{N}} && \text{bei Problemgröße 1} \\ K &= N && \text{bei Laufzeit 1} \end{aligned}$$

Mit diesen Formeln ist eigentlich alles gesagt: Durch Parallelisierung erhalten wir zwar nur einen begrenzten Zeitgewinn, aber einen beliebig hohen Kapazitätsgewinn. Also lohnt sich massive Parallelisierung doch!

Leider hat es Gustafson nicht bei der einfachen Formel $K = N$ belassen. Stattdessen stellt er Überlegungen an, welche Zeit das neue, größere Problem auf einer 1-Prozessor-Maschine benötigt hätte. Der daraus resultierende, fiktive Zeitgewinn lautet dann:

$$\begin{aligned} \text{Zeitgewinn} &= \frac{(1 - P') + K \cdot P'}{(1 - P') + \frac{K \cdot P'}{N}} && K = N \text{ einsetzen} \\ &= \frac{(1 - P') + N \cdot P'}{(1 - P') + \frac{N \cdot P'}{N}} \\ &= \frac{(1 - P') + N \cdot P'}{(1 - P') + P'} \\ &= \frac{(1 - P') + N \cdot P'}{1} \end{aligned}$$

Im Nenner steht die ursprüngliche Laufzeit, die sich wie erwartet auf 1 zusammenkürzt. Damit ergibt sich das Gustafsonsche Gesetz:

$$\text{Zeitgewinn} = (1 - P') + N \cdot P'$$

Die Absurdität tritt deutlich zutage, wenn man bedenkt, dass dieser „Zeitgewinn“ letztlich auf der Annahme basiert, dass die Laufzeit konstant gehalten wird.

Stellen wir uns folgendes Sonderangebot im Supermarkt vor: „Zwei Stück zum Preis von einem!“ Kaufen wir nun zwei Stück statt einem, hätten wir nach Gustafsons Argumentation 50% Geld gespart. Aber wir haben genauviel Geld ausgegeben wie sonst auch! Wir haben überhaupt kein Geld gespart, sondern 1 Stück extra erhalten. Wir haben einen materiellen Gewinn, aber keinen finanziellen!

Genauso ist der von Gustafson formulierte „Zeitgewinn“ unzulässig. Dieser völlig unnötige Schritt in Gustafsons Arbeit [Gus88] sorgte für viel Verwirrung.

5 Widerlegungen

Seit der Formulierung des Amdahlschen Gesetzes gab es viele Hinweise darauf, dass es widerlegt worden sei. In diesem Kapitel untersuchen wir, was an diesen Hinweisen dran ist.

5.1 Widerspruch zwischen den Gesetzen?

Scheinbar gibt es einen grundlegenden Widerspruch zwischen dem Amdahlschen und dem Gustafsonschen Gesetz:

Das Amdahlsche Gesetz besagt, dass sich schon ab einer recht geringen Prozessorenanzahl kein weiter Geschwindigkeitsgewinn mehr erwarten lässt, wohingegen das Gustafsonsche Gesetz von einem nahezu linearen Geschwindigkeitsgewinn spricht. Die Ursache dieses scheinbaren Widerspruchs ist jedoch lediglich, dass bei dem Amdahlschen Gesetz die Problemgröße und vor allem auch die Größe des parallelisierbaren Anteils konstant sind.

Gustafson geht jedoch von einem mit wachsender Problemgröße verschwindend geringem sequentiellen Anteil aus. Damit kann dann in der gleichen Zeit ein linear mit der Anzahl der Prozessoren wachsendes Problem lösen. Bei dem Gustafsonschen Gesetz wird dann das Verhältnis aus der theoretischen Laufzeit dieses (größeren) Problems auf einem Prozessor und der Laufzeit auf $N \gg 1$ Prozessoren gebildet. Es wird hier also nur ein vollkommen anderer Aspekt betrachtet.

Amdahl sagt letztendlich, dass sich massive Parallelisierung nicht lohnt, weil sich unsere bisherigen Programme nur bis zu einer bestimmten Grenze beschleunigen lassen. Gustafson erwidert, dass sich Parallelisierung dennoch lohnt, weil wir dadurch größere Probleme berechnen können.

Parallelisierung bringt zwar *wenig Zeitgewinn*, aber einen *großen Kapazitätsgewinn.*

5.2 Algorithmen ändern sich durch Parallelisierung

In den vergangenen Jahren hat sich gezeigt, dass bei der Parallelisierung einiger Programme Geschwindigkeitsgewinne erreicht wurden, welche dem Amdahlschen Gesetz zu widersprechen scheinen.

Ein Grund für diesen unerwarteten Zeitgewinn liegt darin, dass ein sequentieller Algorithmus nicht ohne weiteres auf mehrere Prozessoren aufgeteilt werden kann. Meist ist hierfür eine Anpassung nötig, die unter Umständen zu einer grundlegenden Änderung des Algorithmus' führen kann. Dabei ist es dann auch möglich, dass sich sogar die Komplexitätsklasse des Algorithmus ändert, was dann den beobachteten Effekt erklärt.

Betrachten wir hierzu beispielsweise den folgenden Algorithmus:

Der Algorithmus erhält eine sortierte Liste $x_1, ..., x_n$ paarweise verschiedener Elemente als Eingabe und soll in ihr ein bestimmtes Element x finden bzw. "nicht gefunden" ausgeben, wenn das Element nicht enthalten ist.

Im Groben führt der Algorithmus einfach eine lineare Suche auf der gesamten Liste aus, um das gesuchte Element zu finden. Das die Liste sortiert ist, wird dabei nicht beachtet.

Lediglich vor dem Start der linearen Suche führt der Algorithmus einen Randtest aus, d.h. er überprüft, ob das gesuchte Element mindestens so groß wie das erste und höchstens so groß wie das letzte Element der Liste ist. Damit lässt sich unter Umständen sehr schnell feststellen, dass das gesuchte Element nicht in der Liste enthalten sein kann. (Dies ist der Fall, wenn $x < x_1$ oder $x > x_n$ gilt.)

Wenn das gesuchte Element wirklich in der Liste enthalten ist, bringt dieser Randtest keinen Vorteil. Damit ist bei einem Prozessor die Laufzeit $O(n)$.

Nun parallelisieren wir den Algorithmus wie folgt:

Wenn noch $N > 1$ freie Prozessoren zur Verfügung stehen, so wird die Liste in N gleichgroße Stücke zerlegt und jedem dieser N Prozessoren wird eines dieser Stücke zur Bearbeitung übergeben. Nehmen wir einmal an, dass wir nur 2 Prozessoren haben. Dann wird die Liste zu Beginn in zwei Teillisten zerlegt.

Durch den anfänglichen Randtest wird einer der Prozessoren sofort feststellen, dass seine Teilliste das gesuchte Element nicht enthalten kann. Also kann er die Bearbeitung seines Teilproblems sofort beenden. Damit stehen für den zweiten Teil der Liste wieder zwei Prozessoren zur Verfügung, womit sie erneut geteilt werden kann.

Durch Iteration dieses Verfahrens entartet die anfängliche lineare Suche bei einem Prozessor zu einer Binärsuche, sobald man 2 Prozessoren zur Verfügung hat. Damit reduziert sich auch die Komplexität auf $O(\log n)$.

5.3 Wirkliche Widerlegung

Ein weiterer Grund für den Geschwindigkeitszuwachs liegt in der Hardware.

Die vereinfachenden Grundannahmen des Amdahlschen Gesetzes ignorieren zum Beispiel, dass mit einer Erhöhung der Prozessorenanzahl meistens auch eine Vergrößerung des zur Verfügung stehendes Hauptspeichers und Caches einher geht. Insbesondere kann es zu dem Effekt kommen, dass ein großes Problem, welches unter Verwendung eines einzigen Prozessors vielleicht nicht einmal in den Speicher passte, nach der Parallelisierung in so viele Teilprobleme zerlegt wurde, dass diese zumindest zu einem großen Teil (wenn nicht sogar ganz) im Cache gehalten werden können. Damit ist dann natürlich eine viel höhere Verarbeitungsgeschwindigkeit der Teilprobleme und somit auch des gesamten Problems möglich.

Literatur

[Amd67] Amdahl, Gene M.: *Validity of the single processor approach to achieving large scale computing capabilities*. 30:483–485, 1967.

[Gus88] Gustafson, John L.: *Reevaluating Amdahl's Law*. Comm. ACM, 31:532–533, 1988.

[Shi96] Shi, Yuan: *Reevaluating Amdahl's Law and Gustafson's Law*. `http://www.cis.temple.edu/~shi/docs/amdahl/amdahl.html`, 1996.

[Wik08] Wikipedia: *Amdahl's law*. `http://en.wikipedia.org/w/index.php?title=Amdahl%27s_law&oldid=210767896`, 2008.