

X . s y s t e m s . p r e s s

X.systems.press ist eine praxisorientierte
Reihe zur Entwicklung und Administration von
Betriebssystemen, Netzwerken und Datenbanken.

Hans Günther Kruse

Leistungsbewertung bei Computersystemen

Praktische Performance-Analyse
von Rechnern und ihrer Kommunikation

 Springer

Hans Günther Kruse
Universität Mannheim
Rechenzentrum
68131 Mannheim
hans-guenther.kruse@rz.uni-mannheim.de

ISBN 978-3-540-71053-0

e-ISBN 978-3-540-71054-7

DOI 10.1007/978-3-540-71054-7

X.systems.press ISSN 1611-8618

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

© 2009 Springer-Verlag Berlin Heidelberg

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funk- sendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk be- rechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Einbandgestaltung: KünkelLopka, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

springer.de

Inhaltsverzeichnis

1	Vorbemerkungen zur Last mit der Leistung.....	1
2	Einfache Leistungsmaße.....	5
2.1	Takt, CPI und Ablaufzeit	5
2.2	Durchsatz, MIPS und MFLOPs	10
3	Die Gesetze von Amdahl und Gustafson.....	15
3.1	Das Gesetz von G. Amdahl.....	15
3.2	Amdahl und das Vektor-Pipeline-Prinzip	18
3.3	Das skalierte Gesetz von Gustafson	22
3.4	Befehlsverarbeitung nach dem Pipeline-Prinzip	24
3.5	Zusammenfassung	26
4	Die Gesetze von Little	27
4.1	Voraussetzungen.....	27
4.2	Die Herleitung der Gesetze	31
4.3	Zusammenfassung	35
5	Benchmarks oder die Last mit der Last.....	37
5.1	Historie und Voraussetzung	37
5.2	Ein explizites Benchmark-Beispiel	39
5.3	Vergleich von Benchmarks	45
5.4	Einige statistische Standardmethoden.....	48
6	Messbare Leistungsgesetze.....	53
6.1	Herleitung des globalen Auslastungsgesetzes.....	53
6.2	Lokale Auslastungsgesetze und die Flaschenhals-Analyse.....	55
6.3	Die Segmentierung von Systemen	57
6.4	Das Central-Server-System.....	61
6.5	Das verallgemeinerte Antwortzeit-Gesetz	64
6.6	Dialogsysteme.....	67

7	Einfache stochastische Modellvorstellungen	73
7.1	Voraussetzungen	73
7.2	Ableitung der Zeit-Dynamik und Skizze der Lösung	75
7.3	Zeitunabhängige Lösungen im Gleichgewicht.....	77
7.4	Weitere Leistungsgrößen	80
7.5	Endliche Modelle.....	83
8	Die Chapman-Kolmogoroff-Gleichungen.....	87
8.1	Der Übergang $0 \rightarrow 0$	88
8.2	Der Übergang $1 \rightarrow 1$	89
8.3	Der Übergang $0 \rightarrow 1$	90
8.4	Der Übergang $1 \rightarrow 0$	91
8.5	Zusammenfassung und Ableitung der Zustandsgleichungen.....	91
8.6	Zur Lösung der Zustandsgleichungen.....	93
8.7	Zwei konkrete Modelle über Reserve	96
9	Von der Lust am Modellieren	103
9.1	M/M/1/ ∞ und die Skalierung.....	103
9.2	Das Modell M/M/1/B mit endlichem Wartepool	104
9.3	Ein weiteres Verlust-Modell mit p Prozessoren.....	107
9.4	Der Fall unendlich vieler Prozessoren	111
9.5	Modelle mit variabler Last.....	112
9.6	Komposition von Submodellen.....	116
9.7	Leistungsunterschiede.....	118
9.8	Modell eines File-Servers	123
10	Leistungsanalyse auf Modellbasis.....	131
10.1	Die Diskussion des offenen Modellsystems	132
10.2	Über die Analyse von geschlossenen Systemen	139
11	Konzepte zur Analyse von Kommunikationssystemen.....	149
11.1	Allgemeine strukturelle Überlegungen	149
11.2	Analyse des ALOHA-Protokolls	153
11.3	Eine Verbesserung mit Slotted ALOHA.....	158
11.4	Eine anspruchsvollere Analyse von Slotted ALOHA	161
12	Modell und Wirklichkeit	165
12.1	Bemerkungen zu den Modellvoraussetzungen.....	165
12.2	Ein explizites Beispiel.....	167
12.3	Modelle des G/M/1/ ∞ -Typs mit nicht bekannter Ankunftsverteilung.....	170
12.4	Modell mit deterministischer Ankunftszeit.....	173
12.5	Allgemeine Bedienzeiten	176
12.6	Beispiel zu einem Modellvergleich.....	177

13 Über die Gültigkeit von Gesetzen	181
13.1 Die Skalierung nach Amdahl und Gustafson	181
13.2 Verallgemeinerte Durchsatzskalierung	184
13.3 Ein möglicher Ansatz über Prozessorkonfigurationen	186
13.4 Die Effizienz von p Prozessoren	188
13.5 Über eine mögliche Einbeziehung der Kommunikation	191
Literaturverzeichnis	195
Index	197

1 Vorbemerkungen zur Last mit der Leistung

Die Leistungsanalyse von Rechnern und Rechnernetzen ist im Prinzip aus der Notwendigkeit entstanden, vorhandene Systeme auf Schwachstellen hin zu untersuchen, um anschließend Verbesserungen konzipieren zu können.

Voraussetzung ist natürlich, dass man eine genaue Vorstellung von **Leistung** hat, und dies ist leider nicht immer gewährleistet. Im Rahmen eines nahe liegenden Ansatzes wäre es natürlich möglich, sich auf die Physik zu besinnen und Leistung als Arbeit pro Zeit zu definieren. Für viele Situationen und Systeme trifft diese einfache Vorgehensweise auch zu und wir werden sie auch später ausgiebig nutzen, jedoch immer die Grenzen aufzeigen.

Will man jedoch die Leistungsfähigkeit eines hochkomplexen Systems wie das eines heutigen Computers oder Rechnernetzes bewerten, so muss man sich genau überlegen, welche Größen bzw. Variablen die Leistung beschreiben.

Besteht die Leistung in einer möglichst schnellen Verarbeitung von Befehlen oder im Befehlsumfang, besteht sie aus der Größe der Speicherkapazitäten oder der Organisationsform wie SingleUser/SingleTask, MultiUser/MultiTask, Dialog- und Batch-Betrieb? Auch hier wird es auf die Gründe der Leistungsanalyse ankommen, welche letztlich die Auswahl der entsprechenden Variablen oder Messgrößen festlegen. So wird es einen großen Unterschied ausmachen, ob die Geschwindigkeit beim Übersetzen von Programmen oder die Verarbeitung einer großen Anzahl von Gleitkommaoperationen im Vordergrund steht. Beide Male sind Maschinenbefehle zu analysieren, wobei aber Organisation und Ablauf völlig verschieden sind. Bei Übersetzern/Compilern kommt es auf die effiziente Nutzung von Befehlen auf Zeichenketten und zum Lesen/Schreiben des Hauptspeichers an. Bei den Gleitkommaoperationen ist es häufig nur eine Addition, die gleichförmig über eine große Anzahl von Operanden abläuft.

Oder betrachten wir eine andere Anwendung, welche sehr stark vom Zugriff auf Datenbanken Gebrauch macht. Die dazugehörigen Transaktionen hängen von den Speicher- und I/O-Zugriffen ab, auch die Cache-Organisation spielt eine Rolle, numerische oder Zeichenkettenoperationen wie bei unseren beiden ersten Beispielen treten völlig in den Hintergrund. Natürlich kann man argumentieren, dass

die **Laufzeit z_{cpu} [sec]** aller drei Anwendungen die entscheidende Leistungsgröße ist – je kürzer, desto effizienter und leistungsfähiger –, die Ursache dafür sind jedoch so verschieden, dass kein Leistungsvergleich auf dieser Ebene einen Sinn macht.

Wir werden uns daher in den folgenden Ansätzen bemühen, solche Widersprüchlichkeiten zu vermeiden, und beginnen mit der Darstellung der grundsätzlichen Situation bei der Leistungsanalyse (s. Abb. 1.1).

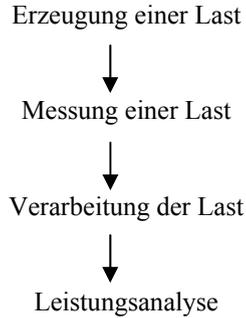


Abb. 1.1 Last/Leistungsmodell

Zentral sind hierbei die Begriffe **Leistung**, dessen Problematik schon skizziert wurde, und **Last**. Im einfachsten Fall wird dies – die **Last** – ein typisches Anwendungsprogramm sein, welches von einem Compiler in Maschineninstruktionen umgewandelt, in der Verarbeitungseinheit abläuft. Genauso denkbar ist aber ein Datenpaket mit definiertem Aufbau und Länge, welches in einem Router an eine bestimmte Adresse (forwarding) geschickt wird. In beiden Fällen besteht die Last aus einem Objekt, an welchem z. B. die **Ablaufzeit** (CPU-Zeit) bestimmt wird – man misst einen individuellen Auftrag.

Der weitaus schwierigere Fall besteht darin, die Last aus mehreren Programmen oder vielen Datenpaketen zu gestalten, die untereinander kein einheitliches Verhalten zeigen. So enthalten sie mal mehr, mal weniger I/O-Anweisungen oder Gleitkommaoperationen – auch können die Datenpakete im Routerbeispiel unterschiedlich lang sein, ähnlich ist es bei den Datenbanktransaktionen. Wenn es Variablen gibt, die eine gewisse statistische Gesetzmäßigkeit aufweisen, kann man diese zur Beschreibung der Gesamtlast heranziehen – allerdings sind dabei häufig gewisse vereinfachende Modellvorstellungen nicht zu vermeiden. Als Leistungskenngröße bietet sich hier der **Durchsatz** an, d. h. die Anzahl der bearbeiteten Aufträge oder Datenpakete pro Zeiteinheit. Interessanterweise kommen wir mit diesem Begriff „Durchsatz“ auch an die physikalische Definition Arbeit pro Zeit nahe heran.

Bei den vorausgegangenen Ausführungen war immer von den DV-Systemen als Ganzes ausgegangen und die entsprechenden Leistungsmaße wie Systemzeit oder Durchsatz definiert worden. Nun kann man die Komponenten wie E/A-Subsystem,

Bus, Caches oder Platten entsprechend behandeln und Leistungsmaße wie Bandbreite (Mbit/sec), Hitraten oder Umdrehungen pro Sekunde (rps) einführen – aber es wird sehr schwierig, Maße für das Zusammenwirken solcher Komponenten zu finden. Dies verkompliziert sich noch, wenn zur Hardware noch Software wie Betriebssysteme oder Netzprotokolle treten.

Aus diesem Grund verzichten wir in den folgenden Analysen und Diskussionen weitgehend auf die Berücksichtigung von Details und bleiben bei der bereits anfangs konzipierten „Black-Box“. Das bedeutet die Lastdefinition und Lastmessung „vor“ der Box und die Leistungsmessung „hinter“ der Box. Dabei beachten wir folgende Grundsätze:

1. Last- und Leistungsmaß müssen miteinander korrespondieren, z. B. wird man ein interaktives System anders als ein Batch-System ausmessen.
2. Sorgfältige Auswahl geeigneter statistischer Techniken, um die Messungen zu validieren und zu vergleichen.
3. Vergleich der gemessenen Werte mit entsprechenden Ergebnissen analytischer oder simulativer Modelle.

Der letzte Punkt mag vor dem Hintergrund der reinen Messung nicht ganz einseitig erscheinen, tatsächlich zeigt aber die praktische Erfahrung, wie wichtig gut strukturierte und validierte Modelle für das Verständnis von Systemleistung sind.

Die Leistungsanalyse und Modellierung haben ihren Ursprung in der Telefonie, erst nach dem Zweiten Weltkrieg und vor allem zu Beginn der 70er-Jahre des letzten Jahrhunderts wurden sie mit Erfolg auf dem Feld der Rechner und Netze angewandt. Da die analytischen Methoden für die immer komplexer werdenden Systeme bald nicht mehr ausreichten, entstanden umfangreiche und leistungsfähige numerisch orientierte Algorithmen und Programme. Vor allem die Entwicklung der Internettechnologie und der Druck, eine effiziente Steuerung der Mainframes/Großrechner, die damals die Hauptstütze der Datenverarbeitung darstellten, zur Verfügung zu haben, hatten einen entscheidenden Anteil daran.

In dieser Zeit entstand auch das klassische Werk von L. Kleinrock [1], welches bis in die heutige Zeit ein Standard hinsichtlich der grundsätzlichen analytischen Methodik geblieben ist. Wir werden uns in den Ausführungen der anschließenden Kapitel, vor allem bei der Modellierung häufig darauf beziehen. In den letzten 20 Jahren sind zahlreiche Lehrbücher zur Thematik „Leistungsanalyse“ erschienen, wir werden häufig Bezug nehmen auf die Werke von Raj Jain [2], von M. Haas und W. Zorn [3], die ausgezeichneten Werke von Tran-Gia [4] und G. Bolch [5]. Eine herausragende Rolle spielt das Lehrbuch von Hennessy & Patterson [6], dem viele moderne Anregungen und instruktive Beispiele entnommen wurden. Gerade dieses Buch zeigt, wie man analytisch-mathematische Methoden zum Verständnis sinnvoll nutzen kann, ohne die Methodik und Mathematik als Selbstzweck (*l'art pour l'art*) darzustellen. Ein ähnliches Ziel, allerdings ohne Berücksichtigung von Hardware-Details, ist hier auch ins Auge gefasst. Meinen Kollegen Dr. Erich

Strohmaier in Berkeley und Dr. Heinz Kredel in Mannheim verdanke ich ebenfalls in dieser Richtung viele wertvolle Diskussionsbeiträge sowie Herrn Prof. Effelsberg/Mannheim die günstigen Rahmenbedingungen.

Ich habe mich, zumindest in den ersten Kapiteln, bemüht, mit einer elementaren Mathematik auszukommen, bis auf einige „Ausrutscher“ bei Statistik-Problemen im Kap. 5 über Benchmarks sollte dies im Großen und Ganzen gelungen sein. In den folgenden Ausführungen des Kap. 6 über messbare Leistungsgesetze kommt die Technik der linearen Gleichungssysteme zum Einsatz, während ab Kap. 7 einfache Differenzialgleichungen 1. Ordnung in der Zeit eine Rolle spielen. Hier wurden einmal bei der Lösung einer speziellen Gleichung etwas ausführlicher als notwendig mathematische Details skizziert, diese können jedoch ohne Probleme übersprungen werden. Die Theorie der Markoff'schen Prozesse, welche der sogenannten stochastischen Modellierung zugrunde liegt, wird nur in Ansätzen aufgezeigt und darüber hinaus auf jede mathematische Strenge verzichtet – der Schwerpunkt liegt eindeutig auf der Nachvollziehbarkeit der Beispiele. Dies gilt auch für die Skizze der Chapman-Kolmogoroff-Gleichungen und ihrer Herleitung, welche mir nicht nur wegen der Vollständigkeit, sondern wegen ihrer praktischen Anwendung unverzichtbar erscheint.

Im Kap. 10 über Leistungsgesetze und Modellierung wird der Versuch unternommen, Messung und Theorie in Einklang zu bringen, die Beschränkung auf den Algorithmus der Mittelwertanalyse (MVA) ist vor dem Hintergrund eines Einstiegs in diese Art von Verfahren zu sehen. Der an mehr und detaillierteren Informationen interessierte Leser sei an dieser Stelle auf das ausgezeichnete Buch von G. Bolch [5] verwiesen.

Das folgende Kap. 11 beschäftigt sich mit einigen Ideen und Konzepten zur Leistungsanalyse von Kommunikationssystemen, wobei zunächst weitgehend auf bereits diskutierte Schemata zurückgegriffen wird. Da die stochastische Modellierung für Protokolle sehr viel an mathematischer Technik verlangt, wurde zunächst ein relativ simpler Ansatz gewählt, der anschließend verfeinert wird, und deshalb etwas aufwendiger ist. Auch hier können die mathematischen Details aber ohne Probleme übergangen werden.

Die beiden letzten Kap. 12 und 13 beschäftigen sich mit dem Spannungsverhältnis von Modell und Wirklichkeit sowie der Gültigkeit von zwei wichtigen Leistungsgesetzen bei modernen Clusterarchitekturen. Da es sich gezeigt hat, dass in vielen Fällen die diskutierten Modellierungstechniken nicht ausreichen, möchte ich hier Hinweise geben, wie man in diesen Fällen zu verfahren hat, aber auch nicht verschweigen, dass ab jetzt doch Einiges mehr an Mathematikkenntnissen notwendig ist.

An der schriftlichen Form dieser Ausarbeitung waren eine Reihe von Studierenden als Hilfskräfte beteiligt. Erwähnen möchte ich Florian Strasser, Julia Doll, Bernhard Kusch und besonders Sabrina Hauser – ihnen allen gilt mein Dank für ihre Geduld und ihre Mühe.

2 Einfache Leistungsmaße

In diesem Kapitel geht es um einfache, direkt messbare Leistungsgrößen oder -maße, welche eine erhebliche Bedeutung über alle Systemklassen und Größenordnungen hinweg haben.

2.1 Takt, CPI und Ablaufzeit

Das am leichtesten zugängliche Leistungsmaß einer Hardware, um genauer zu sein, eines Prozessors, ist das diskrete feste Zeitraster, nach dem sich alle Aktionen ausrichten – der Takt!

So entspricht der 100-MHz-Takt längst Vergangenheit gewordener Maschinen einem Zeitabstand zwischen zwei Ereignissen – der Basistaktzeit $bzeit$ – von 10 nsec, das sind $10 \cdot 10^{-9}$ sec. Die heutigen Gigahertz-Prozessoren entsprechen einem Abstand von 1 nsec. Zählen wir die Takte, welche eine vorgegebene Last benötigt, mit der Variable **#takte** (# bezeichnet immer eine Anzahl), dann ergibt sich die Ablaufzeit $zcpu$ der Last zu

$$zcpu = \#takte * bzeit = \frac{\#takte}{\frac{1}{bzeit}} = \frac{\#takte}{takt}$$

denn klarerweise ist **takt** = $1/bzeit$. Hierbei ist die Variable **#takte** zwar unmittelbar einsichtig, aber schlecht einer direkten Messung zugänglich, weiterhin ist damit die Last nicht vor, sondern nach der Verarbeitung definiert – eine zu große Abweichung von unserem Konzept. Deshalb wird wie folgt verfahren.

Die Zahl der Instruktionen der Last **#last** lässt sich im Prinzip vor der Verarbeitung bestimmen. Wir messen die Zahl der benötigten Takte **#takte** und bestimmen mit der Größe **cpi**, wie viel Takte für eine Instruktion benötigt werden:

$$cpi = \frac{\# \text{ takte}}{\# \text{ last}}$$

Für die Ablaufzeit folgt danach:

$$zcpu = \frac{\# \text{ takte}}{\text{takt}} = \frac{\# \text{ takte}}{\# \text{ last}} * \frac{\# \text{ last}}{\text{takt}}$$

$$zcpu = cpi * \frac{\# \text{ last}}{\text{takt}} = \# \text{ last} * cpi * \text{ bzeit}$$

Diese Umformungen sind keine überflüssige Spielerei, sondern widerspiegeln die drei wichtigsten Ebenen in einem DV-System:

- **#last** = Zahl der Instruktionen, hängt von der Anwendung und Compilertechnologie ab;
- **cpi** = Zahl der Basistakte, die für eine Instruktion benötigt wurde, hängt von der Maschinenarchitektur ab;
- **bzeit** = Basistakt, ist direkt von der eingesetzten Hardware- Technologie abhängig.

So verkürzt eine steigende Taktrate die Basiszeit **bzeit** und damit in der Theorie die Ablaufzeit **zcpu**. Eine RISC-Architektur versucht, **cpi** auf den Wert 1 zu drücken, also einen Takt pro Instruktion – in superskalaren Maschinen liegt dieser Wert sogar darunter –, und hochoptimierende Compiler versuchen die Anzahl der Maschineninstruktionen **#last** zu minimieren.

Die Größe **cpi** muss noch etwas genauer analysiert werden. Hintergrund ist die Tatsache, dass Maschinenbefehle unterschiedliches Zeitverhalten von den Hardware-Ressourcen her haben, und es daher kein exaktes globales **cpi** geben kann. Vielmehr muss man die Maschinenbefehle in Gruppen von Instruktionen einteilen, welche dann innerhalb der Gruppe k alle die gleiche Taktzahl pro Befehl besitzen:

$$\{\text{Instruktionen}\}_{k=1\dots n} = > cpi_{k=1\dots n}$$

Kennt man dann das prozentuale Gewicht p_k jeder Gruppe mit $p_1 + p_2 + p_3 + \dots + p_n = 1$ (n ist die Gruppenanzahl), dann gilt für die globale Größe

$$cpi = \sum_{k=1}^n p_k * cpi_k$$

Das folgende Beispiel soll dies etwas verdeutlichen.

B1

Wir betrachten eine Modell-RISC-Maschine mit den Befehlsgruppen

		cpi_k	p_k
Arithmetisch-logisch	ALU	1	0,4 (40%)
Speicher	LOAD/STORE	2	0,4 (40%)
Test & Verzweigung	JMP	3	0,2 (20%)

$$cpi = 1 * 0,4 + 2 * 0,4 + 3 * 0,2 = 1,8$$

An der Größe **cpi** kann man jetzt sehr schön den Einfluss verschiedener Maßnahmen analysieren. Angenommen wir ändern durch den Einsatz eines besseren Compilers das Gewicht $p_n = I_n / I$ der n-ten Gruppe $I_n \rightarrow \alpha * I_n$, wobei $0 \leq \alpha \leq 1$.

Aus dem alten Wert

$$cpi = \sum_{k=1}^n cpi_k \left(\frac{I_k}{I} \right), \quad I = \sum_{k=1}^n I_k$$

wird dann

$$\begin{aligned} cpi' &= \sum_{k=1}^{n-1} cpi_k \left(\frac{I_k}{I - (I_n - \alpha I_n)} \right) + cpi_n * \left(\frac{\alpha * I_n}{I - (I_n - \alpha I_n)} \right) \\ &= \sum_{k=1}^{n-1} cpi_k * \left(\frac{I_k}{I} \right) * \left(\frac{1}{1 - (1 - \alpha) * p_n} \right) + \left(\frac{\alpha * p_n * cpi_n}{1 - (1 - \alpha) * p_n} \right) \\ &= \frac{1}{1 - (1 - \alpha) * p_n} \left(\underbrace{\sum_{k=1}^{n-1} p_k * cpi_k + p_n * cpi_n}_{cpi} - p_n * cpi_n + \alpha * p_n * cpi_n \right) \end{aligned}$$

Also folgt

$$cpi' = \frac{1}{1 - (1 - \alpha) * p_n} (cpi - (1 - \alpha) * p_n * cpi_n)$$

Da $0 \leq p_n$, $1 - \alpha \leq 1$, ist der Vorfaktor $1 / (1 - (1 - \alpha) * p_n) \geq 1$ und signalisiert damit erst einmal eine Verschlechterung des neuen cpi-Wertes. Diese kann nur durch die Subtraktion in der Klammer kompensiert werden.

Diese Formel können wir jetzt benutzen, um Änderungen im Instruktionsumfang abzuschätzen.

B2

In der Beispielmachine von B1 wird p_{ALU} um die Hälfte auf 0,2 reduziert, wie ändert sich cpi ?

$$p_{ALU} = 2/5; p'_{ALU} = 1/5 \text{ d.h. } \alpha = 1/2$$

$$cpi' = \frac{1}{1 - (1 - \frac{1}{2}) * \frac{2}{5}} \left(\frac{9}{5} - (1 - \frac{1}{2}) * \frac{2}{5} * 1 \right) = \frac{5}{4} * \frac{8}{5} = 2$$

Dieses Beispiel zeigt, dass eine an sich positive Maßnahme sogar einen schlechten cpi -Wert zur Folge haben kann. Dies ist aber klar, weil die größeren Werte ($cpi_{IS} = 2$, $cpi_{JMP} = 3$) stärker gewichtet werden.

Reduzieren wir etwa in der Gruppe der Sprungbefehle um die Hälfte (0,2 \rightarrow 0,1), folgt $\overline{cpi} = \frac{1}{1 - (1 - \frac{1}{2}) * \frac{1}{5}} * \left(\frac{9}{5} - (1 - \frac{1}{2}) * \frac{1}{5} * 3 \right) = \frac{10}{9} * \frac{15}{10} = \frac{5}{3}$ und die Ver-

besserung beträgt immer hin ca. 8%.

Der andere Parametertyp, den man verändern kann, ist cpi_k von einzelnen Gruppen, ohne dass der Umfang der Maschineninstruktionen verändert wird. Im Prinzip bedeutet dies eine Architekturänderung im Leitwerk (μ -Programm). Wir greifen wieder die n -te Gruppe heraus und reduzieren über $cpi'_n = \alpha * cpi_n$, $0 \leq \alpha \leq 1$:

$$\begin{aligned} &= \sum_{k=1}^{n-1} p_k * cpi_k + p_n * \alpha * cpi_n \\ \langle cpi' \rangle &= \sum_{k=1}^{n-1} p_k * cpi_k + \underbrace{p_n * cpi_n}_{cpi} - p_n * cpi_n + p_n * \alpha * cpi_n \\ \langle cpi' \rangle &= cpi - (1 - \alpha) * p_n * cpi_n \end{aligned}$$

Hier fehlt der vergrößernde Faktor $1/(1 - (1 - \alpha) p_n)$, sodass deutlich höhere Chancen bestehen, den cpi -Wert mit dieser Maßnahme zu verkleinern.

B2'

Wir setzen die Situation wie in B1 und B2 voraus, schaffen es durch eine Änderung des Mikroprogramms, den Wert von cpi_{JMP} von 3 auf 1,5 zu reduzieren. Dann folgt

$$\begin{aligned}\overline{cpi} &= cpi_{ALU} * p_{ALU} + cpi_{LS} * p_{LS} + cpi_{JMP} * p_{JMP} \\ &= 1 * 0,4 + 2 * 0,4 + 1,5 * 0,2 \\ &= 1,5\end{aligned}$$

Dies bedeutet eine 20-prozentige Verbesserung und liegt damit deutlich über den Anstrengungen von B2!

Das dritte Beispiel zu der cpi-Thematik ist etwas ausführlicher und diskutiert auch die Abhängigkeit der Ablaufzeit.

B3

In zwei Maschinen A und B braucht die Verzweigung zwei Zyklen und alle anderen Befehle einen Zyklus. In der auszumessenden Last sind 20% Verzweigungen. Die Taktzeit (Basiszeit) von A ist 25% günstiger als die von B. Welche Maschine ist hinsichtlich dieses Benchmarks günstiger, wenn ein optimierender Compiler 20% der Instruktionen auf B in der Gruppe der Einzyklusbefehle einspart?

$$\begin{aligned}zcpu_A &= \#last_A * cpi_A * bzeit_A \\ cpi_A &= 2 * p_{JMP} + 1 * p_{Rest} \\ &= 2 * 0,2 + 1 * 0,8 = 1,2 = \frac{6}{5} \Rightarrow zcpu_A = \frac{6}{5} * \#last_A * bzeit_A\end{aligned}$$

Für die Maschine B schließen wir analog

$$\begin{aligned}zcpu_B &= \#last_B * cpi_B * bzeit_B \\ cpi_B &= 2 * p_{JMP} + 1 * p_{Rest} = 1,2 = \frac{6}{5} \\ bzeit_B &= \frac{5}{4} * bzeit_A \text{ nach Voraussetzung}\end{aligned}$$

Damit folgt ohne optimierenden Compiler ($\#last_B = \#last_A$):

$$zcpu_B = \left(\frac{5}{4}\right) * \#last_B * \left(\frac{6}{5}\right) * bzeit_B = \left(\frac{5}{4}\right) zcpu_A$$

Der Gewinn (Speed-Up) $Sp(B/A) = (zcpu_B/zcpu_A)$ berechnet sich zu $5/4$, d. h. die Maschine B ist um 25% langsamer.

Der Einsatz des optimierenden Compilers bringt $\#last_B = 0,8 * last_A$, allerdings muss cpi_B neu berechnet werden. Wie in B2 folgt

$$cpi'_B = \frac{1}{1-0,2} \left(\frac{6}{5} - 0,2 * 1 \right) = \frac{5}{4}$$

$$\begin{aligned} zcpu_B &= 0,8 \#last_A * \left(\frac{5}{4}\right) * \left(\frac{5}{4}\right) bzeit_A \\ &= \left(\frac{4}{5}\right) * \left(\frac{5}{4}\right) * \left(\frac{5}{4}\right) * \#last_A * bzeit_A \end{aligned}$$

Damit können wir wie oben die Rechner A und B vergleichen:

$$Sp(B/A) = \frac{zcpu_B}{zcpu_A} = \frac{\left(\frac{5}{4}\right) * \#last_A * bzeit_A}{\left(\frac{6}{5}\right) * \#last_A * bzeit_A} = \frac{25}{24} > 1$$

Rechner B wird also um ca. 5% langsamer sein, obwohl er 20% weniger Instruktionen verarbeiten muss.

Konzentriert man sich statt der eben diskutierten Maßnahme auf eine 20%ige Verbesserung von cpi_{REST} , dann resultiert ein neuer cpi_B -Wert von $26/25$. Der Instruktionsumfang bleibt gleich, sodass ein Vergleich der $zcpu$ -Zeiten $13/12$ ergibt, d. h. die Maschine ist um 8% langsamer.

2.2 Durchsatz, MIPS und MFLOPs

Unter dem Aspekt, die Leistung direkt aus der Taktzeit und der Anzahl der Takte pro Instruktionen zu erschließen und das physikalische Leistungsmaß „Arbeit pro Zeiteinheit“ möglichst gut abzubilden, sind zwei Begriffe entwickelt worden, die

ein gewisses Renommee erworben haben und damit etwas eingehender diskutiert werden sollen.

Das eine Maß ist **MIPS** (**M**illionen **I**nstruktionen **P**ro **S**ekunde) und das heißt, wenn man ein Programm als mögliche Last vorgibt (#last ist die Zahl der Instruktionen), dann folgt:

$$\begin{aligned} \text{MIPS} &= \frac{\# \text{last}}{\text{zcpu}} * 10^{-6} \\ &= \frac{\# \text{last} * 10^{-6}}{\# \text{last} * \text{cpi} * \text{bzeit}} \\ &= \frac{10^{-6}}{\text{cpi} * \frac{1}{\text{takt}}} = \frac{\text{takt}}{\text{cpi}} * 10^{-6} \\ &= \frac{\text{takt}}{\text{cpi}} * 10^{-6} \end{aligned}$$

Dies sieht als Leistungsmaß sehr vernünftig aus, es wächst mit dem Takt (der Taktrate) und fällt mit der Komplexität der Architektur – dann nämlich, wenn ich im Mittel immer mehr Takte für einen Maschinenbefehl verbrauche (z. B. CISC-Maschinen).

Von Hennessy & Patterson [6] werden eine Reihe von Einwänden gegenüber diesem Maß angeführt, die wichtigsten sind:

- MIPS ist abhängig von der Menge der Instruktionen, also der Architektur.
- MIPS ist extrem lastabhängig.
- MIPS kann invers zur gemessenen Leistung sein.

Während die beiden ersten Punkte eigentlich nur eine Reflexion des Standardproblems beim Benchmarking darstellen, muss der dritte Punkt wesentlich ernster genommen werden.

Betrachten wir Gleitkommaoperationen, diese können in Hardware oder Software realisiert sein. Nehmen wir zunächst die Hardware-Variante; im Allgemeinen bedeuten sie ein komplexeres Mikroprogramm, d. h. der Wert von cpi wird größer und damit MIPS kleiner – obwohl die Ausführungszeit sinkt.

Bei der Software-Variante werden einfache Instruktionen mit kleinerem cpi verwendet, dieses bewirkt ein Anwachsen von MIPS – obwohl die Ausführungszeit **zcpu** ansteigt.

B4

Wir greifen auf unsere RISC-Maschine von Beispiel B1 zurück, unterstellen eine Taktrate von 1 GHz = takt. Der Einsatz eines optimierenden Compilers bringt eine 50-prozentige Reduktion der ALU-Operationen. Man berechne den Effekt auf MIPS und Ablaufzeit!

$$cpi_{\text{normal}} = 1,8$$

$$\begin{aligned} MIPS_{\text{normal}} &= \frac{\text{takt}}{cpi_{\text{normal}}} * 10^{-6} = \frac{1 * 10^9}{1,8} * 10^{-6} \\ &= \frac{1000}{1,8} = \frac{10000}{18} = \frac{5000}{9} \end{aligned}$$

$$\begin{aligned} cpi_{\text{opt}} &= \frac{\left(\frac{0,4}{2}\right) * 1 + 0,4 * 2 + 0,2 * 3}{1 - (0,4/2)} \\ &= \frac{0,2 + 0,8 + 0,6}{0,8} = \frac{1,6}{0,8} = 2 \end{aligned}$$

$$MIPS_{\text{opt}} = \frac{\text{takt}}{cpi_{\text{opt}}} * 10^{-6} = \frac{1 * 10^9}{2} * 10^{-6} = \frac{1000}{2} = 500$$

$$\frac{MIPS_{\text{opt}}}{MIPS_{\text{normal}}} = \frac{500}{\frac{5000}{9}} = 1 - \frac{1}{10} = \frac{9}{10}$$

Die optimierte Version hat also eine um 10% kleinere MIPS-Rate!

Wie sieht es bei den Ausführungszeiten aus?

$$\begin{aligned} zcpu_{\text{normal}} &= \#last * cpi_{\text{normal}} * bzeit [\text{sec}] \\ &= \#last * 1,8 * 1,0 * 10^{-9} = 1,8 * 10^{-9} * \#last [\text{sec}] \end{aligned}$$

$$\begin{aligned} zcpu_{\text{opt}} &= 0,8 * \#last * cpi_{\text{opt}} * bzeit \\ &= 0,8 * \#last * 2 * 1 * 10^{-9} = 1,6 * 10^{-9} * \#last [\text{sec}] \end{aligned}$$

D. h. obwohl die optimierende Version mehr als 10% schneller läuft, hat sie eine kleinere MIPS-Rate → Widerspruch im Leistungsmaß!

Das zweite populäre Maß, welches eingeführt wurde, um die Leistungsfähigkeit von Maschinen zu beurteilen, ist **MFLOPS** (**M**illion **F**loating **P**oint **O**perations per **S**econd) und in ähnlicher Weise wie MIPS definiert:

$$\text{MFLOPS} = \frac{\#\text{FLOPS}}{z_{\text{cpu}}} * 10^{-6}$$

Es hängt jedoch noch weit mehr als MIPS von der Hardware und dem Anwendungsprogramm ab. Enthält dies keine oder wenig Gleitkommaoperationen (FLOPS), so wird die gemessene Leistung ziemlich irrelevant sein. Es wird mit MFLOPS/GFLOPS/TFLOPS halt nur eine einzige Komponente eines Rechnersystems ausgemessen und hinsichtlich dieser kann man die Leistung vergleichen. Für wissenschaftliche Anwendungen mit stark numerischem Charakter, z. B. aus der Flüssigkeits- und Gasdynamik der Theoretischen Chemie und Theoretischen Physik, der Astronomie und der Gentechnik, sind TFLOPS allerdings ein geeignetes Maß.

Standard-Last ist der sogenannte Linpack-Benchmark, ein Paket zur Lösung linearer Gleichungssysteme, welches von Jack Dongarra/Knoxville-Tennessee geschrieben wurde. Es erlaubt nicht nur die maximale Leistung R_{max} , sondern auch die Leistung N_{max} bei der halben Problemgröße (Matrixdimension) zu messen; die Implementierung muss so erfolgen, dass die Zahl der Gleitkommaoperationen sich wie $(2/3)n^3 + 0(n^2)$ verhält.

Wir führen einige Beispiele aus der Liste von November 2004 an:

1.	IBM BlueGene/L (32768 * PowerPC)	70,72 TFLOPS
2.	SGI Columbia (10160 * SGI Altix)	51,87 TFLOPS
3.	NEC Earth Simulator (5120 * NEC)	35,86 TFLOPS
10.	NCSA Tungsten (2500 * Dell PowerEdge/Xeon)	9,819 TFLOPS
85.	MPI Garching (822 * IBM p Serie 690 Turbo)	2,198 TFLOPS
168.	Hitachi Leibniz-RZ München (168 * Hitachi)	1,653 TFLOPS

Um unter die Top 10 zu kommen, brauchte eine Installation zu diesem Zeitpunkt mehr als $3 \text{ TeraFLOPS} = 3 * 10^{12}$ Gleitkommaoperationen pro Sekunde. Im Frühsommer 2008 waren ca. neun TFLOPS nötig, um überhaupt in die Liste zu kommen, und das IBM-Roadrunner-System durchbrach die Peta-FLOP-Schranke.

Man muss sich aber stets vor Augen halten, dass dies kein allgemeingültiges Leistungsmaß darstellt – die Gründe sind ähnlich wie bei MIPS gelagert.

Abbildung 2.1 zeigt die MFLOP-Raten der jeweils leistungsfähigsten Gleitkommamaschinen (auch Supercomputer genannt) im Zeitraum 1982–2002. Man beachte den logarithmischen Maßstab auf der vertikalen Achse und somit den Technologiesprung um drei Zehnerpotenzen in einer Dekade.

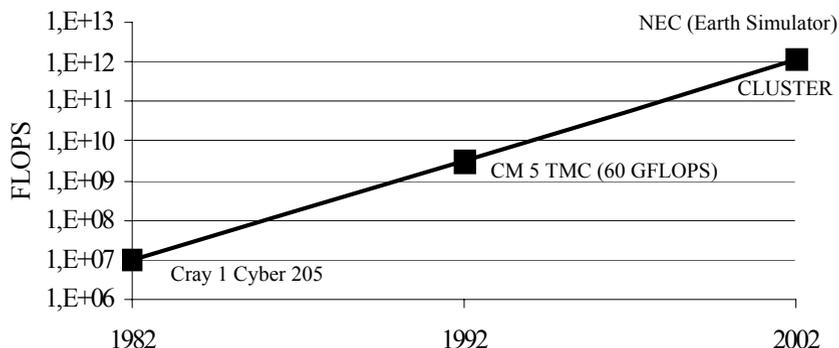


Abb. 2.1 Leistungsentwicklung von Supercomputern

Ohne auf Einzelheiten einzugehen, widerspiegelt diese Entwicklung auch die Veränderung von einer monolithischen klassischen Architektur wie die Cray 1 oder Cyber 205 über die einheitliche Vernetzung einer großen Zahl identischer Prozessoren wie in der CM 5 von Thinking Machines bis hin zur aktuellen Clusterung kompletter Systeme wie bei NEC Earth Simulator oder IBM SP2 Power3 in Livermore (über 6000 Knoten).

Damit einher geht auch eine Veränderung in der Herstellerlandschaft. Waren es zu Beginn der 1980er-Jahre noch Konstrukteure, die aus dem Großrechnerbereich kamen, änderte sich dies zu Beginn der 1990er fundamental. Eine Reihe junger innovativer Firmen und Designer erschien auf der Bühne und löste mit neuen Architekturkonzepten einen Leistungsboom aus. Viele von diesen sind Ende der Dekade schon wieder verschwunden; durchgesetzt hat sich das Cluster-Prinzip, nach dem eine große Anzahl konventioneller Billigprozessoren oder Standardsysteme (dazu zählt auch der α -Prozessor von DEC, später Compaq bzw. hp) durch ein Hochleistungsnetz verbunden wird.

Einen sehr schönen Überblick der Szene gibt die

www.top500.org

Website, welche zweimal jährlich die 500 leistungsfähigsten Maschinen listet und vergleicht – wie bereits erwähnt auf der Basis des Linpack-Benchmarks.