Die Bedeutung von Performance

Martin Gaitzsch, Mirek Hancl, Davood Kheiri

8.11.2000

A. Einleitung, 'relative' Performance

Performance interessiert sowohl den Systementwickler als auch den Kunden. Auf der Entwicklungsseite kommt z.B. neues Chiplayout der Performance-Optimierung zugute, der Anwender profitiert von Performanceangaben beim z.B. Kauf eines Neusystems (Benchmarks).
Für Performancebestimmung ist Zeit das Maß aller Dinge. Systemadministratoren interessiert der Durchsatz, d.h. wie viele Anfragen an ein System gleichzeitig gestellt werden können, für Einzelplatzsysteme ist die verstrichene Zeit eines einzelnen Programms aussagend. Man unterscheidet zwischen

- Antwortzeit: Zeit zwischen Starten und Beenden eines Programms incl. Wartezeit für I/O (wall-clock time, response time, elapsed time)
- Ausführungszeit: CPU-time
- User CPU-time: Zeit für Programmablauf
- System CPU-time: Zeit für Betriebssystemaktivitäten

'Relative' Performance gibt einen einfachen Leistungsvergleich zwischen zwei Systemen durch Zeitmessung eines gleichen Programms auf beiden Systemen.

$$\frac{Performance_x}{Performance_y} = \frac{Ausführungszeit_y}{Ausführungzeit_x} = n$$

Beispiel: Rechner A benötigt 10s für ein Programm, Rechner B 15s für dasselbe Programm.
Rechner A ist 15/10=1,5mal schneller als Rechner B.

B. Taktzyklus, Berechnung Ausführungszeit u. CPU-time

Ein Quarz in einem System liefert konstante Zeitintervalle, sog. Taktzyklen (clock cycles, ticks,...).
Eine Taktperiode ist die Zeit eines Taktzyklus bzw. die Taktrate, welche die Inverse des Taktzyklus ist.
Damit läßt sich die Ausführungszeit für ein Programm berechnen:

Ausführungszeit für ein Programm	=	benötigte Taktzyklen	mal Zeit eines Taktzyklus *oder* geteilt durch Taktrate

Beispiel: Benötigte Taktzyklen für Programm X: 10^{10}, Zeit eines Taktzyklus: 2ns, Taktrate: 500 MHz
Ausführungszeit für Programm X: 10^{10} / 500 MHz = 20s

→ Mehr Performance durch weniger Taktzyklen und der jeweiligen Zeit eines Taktzyklus

Um die CPU-time errechnen zu können muß man wissen, wie viele Taktschritte ein Programm benötigt. Da viele Maschinenbefehle mehr als einen Taktzyklus benötigen (Multiplikation, Gleitkomma-Arithmetik, Speicherzugriffe), wird eine ‘mittlere‘ Anzahl von Taktschritten je Befehl angegeben. Man spricht von clock cycles per instruction, kurz: CPI. Berechnet wird diese durch gewichteten Mittelwert aller individuellen CPIs der Befehle im Programm.

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i * C_i)}{\sum_{i=1}^{n} C_i}$$

Befehle der Klasse i mit CPI_i kommen im betrachteten Programm C_i mal vor.

Beispiel: Befehl A: 1 Taktzyklus, kommt 65mal im Programm X vor.

Befehl B: 2 Taktzyklen, kommt 35mal im Programm X vor.

$$CPI_x = \frac{65*1+35*2}{35+65} = 1{,}35$$

Die CPU-time ergibt sich nun aus folgendem grundlegenden Zusammenhang:

$$CPU\ time = \frac{Instructions}{Pogram}\ X\ \frac{Clock\ Cycles}{Instruction}\ X\ \frac{Seconds}{Clock\ Cycle}$$

⇩ ⇩ ⇩

Befehlsanzahl * CPI * Takperiode

C. Arithmetisches Mittel, Geometrisches Mittel

Mit dem arithmetischen Mittel lassen mehrere Einzelmessungen der Ausführungszeiten einfach zusammenfassen:

$$\frac{1}{n} * \sum_{i=1}^{n} Time_i$$

n = Anzahl der Programme; $Time_i$ = Ausführungszeit de i-ten Programms

Beispiel: Programm 1 läuft auf Rechner A in 1s, auf Rechner B in 10s.

Programm 2 läuft auf Rechner A in 1000s, auf Rechner B in 100s.

Gesamtausführungszeit Rechner A: 1001s, Rechner B: 110s.

Arithmetisches Mittel Rechner A: 500,5s, Rechner B: 55s

→ Besser lassen sich mehrere Einzelmessungen über das geometrische Mittel der Ausführungszeiten zusammenfassen:

$$\sqrt[n]{\prod_{i=1}^{n} Time_i}$$

n = Anzahl der Programme, $Time_i$ = Ausführungszeit des i-ten Programms

Für das obige Beispiel ergibt sich somit ein geometrisches Mittel von 31.6 für beide Rechner.

D. Amdahlsches Gesetz (vereinfacht)

Alt:

$$\text{Optimierte Ausführungszeit} = \frac{\text{optimierter Zeitanteil}}{\text{Verbesserungsfaktor}} + \text{restl. Zeitanteil}$$

Neu:

$$\text{Speedup} = \frac{\text{Performance nach Verbesserung}}{\text{Performance vor Verbesserung}} = \frac{\text{Ausführungszeit vor Verbesserung}}{\text{Ausführungszeit nach Verbesserung}}$$

Konsequenz: Optimiere den häufigsten Fall !

E. Benchmark, Performance-Maße

Zur Messung der Performance eignen sich besonders echte Anwenderprogramme, die fertig vorliegen und nicht verändert werden können. Jedoch sind diese anwenderabhängig, d.h. ein CAD-Programm eines Architekten interessiert einen Programmierer zur Performancemessung seines Systems wenig, eher schon ein Compiler oder Texteditor. Programme zur Messung sollen einfach strukturiert sein, allerdings sind sie betrugsanfällig durch z.B. den Einsatz eines optimierten Compilers. Als allgemein akzeptierten Kompromiß gilt SPEC (System Performance Evaluation Cooperative), gegründet von Apollo/HP, DEC, MIPS und Sun im Jahre 1989. Als Referenz diente bis 1995 eine VAX 11/780, gemessen dient ein Softwarebündel mit unterschiedlichsten Operationen. Durch einen optimierten Compiler war es möglich, den Benchmark der Programme matrix300 und nasa7 um ein vielfaches zu beeinflussen. Als Konsequenz wurden diese 1992 gestrichen und integer- von floating point-Operationen getrennt. Seit 1995 dient eine SunSparcStation 10/40, etw 40mal schneller als die VAX11/780, als Referenzsystem. Neuere Programme wie Webapplikationen sind seitdem dazugekommen.

Die Specratio ergibt sich durch einfaches Dividieren der gemessenen Ausführungszeit durch die des Referenzsystems.

Performance wird des weiteren in folgenden Maßen gemessen:

MIPS, d.h. ‘million instructions per second‘ (native MIPS).

$$\text{native MIPS} = \frac{\text{Anz.d..Befehle}}{\text{Ausf.zeit} * 10 \text{ hoch } 6}$$

Bei optimiertem Compiler mit minimierter CPI spricht man von peak MIPS. Wird gerne von Chipherstellern zu Werbezwecken benutzt.

Bei Vergleich mit einem Referenzrechner spricht man von relative MIPS.

$$\text{relative MIPS} = \frac{\text{Ausf.zeit Referenzrechner}}{\text{Ausf.zeit zu testender Rechner}} * \text{MIPS reference}$$

MOPS, d.h. ‘million operations per second‘

MFLOPS, d.h. ‘million FP operations per second‘

$$\text{MFLOPS} = \frac{\text{Anzahl der Gleitkommaoperationen im Programm}}{\text{Ausf.zeit} * 10 \text{ hoch } 6}$$

Ebenfalls gibt es peak MFLOPS.

Whetstone,Drystone,.... sind synthetische Benchmarks, die speziell im High-End Bereich zum Einsatz kommen.

F. Performance-Optimierung, Zusammenfassung

Performance kann optimiert werden durch höhere Taktrate und/oder geringere CPI. Während letzteres in den Bereich Software fällt und Sache eines Compilers ist, muß für eine höhere Taktrate die Hardware weiterentwickelt werden (Technologischer Optimierung). Einseitig kann selten etwas optimiert werden, es sind oftmals Kompromisse erforderlich.
Performance ist also abhängig vom gewählten Programm, zum besten Vergleich kommen reale Applikationen zum Einsatz. Zeit ist das objektivste Performance-Maß und der häufigste Fall ist zu optimieren (Amdahl'sches Gesetz). Performance-Optimierung ist kompromißgebunden zwischen Taktrate, CPI und Befehlsanzahl. Hersteller geben oft verzerrte Performanceangaben an. Schließlich ist Optimierung auch eine Kostenabwägung.

Quellenangaben:

[1]David A. Patterson, John L. Hennesy. *Computer Organisation & Design, The Hardware/Software Interface*. Morgan Kaufmann, 1998.

Die Bedeutung von

Performance

Martin Gaitzsch, Mirek Hancl, Davood Kheiri

Überblick

Performance richtig

- verstehen
- bestimmen
- angeben

- Entscheidung bei Kauf, Design, Optimierung
- optimale Nutzung der HW bei Programmierung

- besseres Verständnis für Rechnerachitekturen

Performance definieren

Flugzeug	Kapazität	Reichweite (Meilen)	Geschw. (m.p.h.)	Durchsatz
Boeing 777	375	4630	610	228750
Boeing 747	470	4150	610	286700
Concorde	132	4000	1350	178200

Douglas DC-8-50	146	8720	544	79424

Kenngrößen für Performance

- Antwortzeit (wall-clock time, response time, elapsed time): Zeit zwischen Starten und Beenden eines Programms incl. Wartezeit für I/O
- Ausführungszeit: CPU-time
- User CPU-time: Zeit für Programmablauf

- System CPU-time: Zeit für OS – Aktivitäten
- Durchsatz

$$Performance_x = \frac{1}{Ausführungszeit_x}$$

‘relative‘ Performance:

$$\frac{Performance_x}{Performance_y} = \frac{Ausführungszeit_y}{Ausführungszeit_x} = n$$

Beispiel:
Rechner A benötigt 10s für ein Programm,

Rechner B benötigt 15s für dasselbe Programm.
Rechner A ist 15/10 = 1,5 mal schneller als Rechner B.

Taktzyklen

Taktgeber (Quarz) liefert konstante Zeitintervalle, sog. Taktzyklen (clock cycles, ticks,...)
Taktperiode: Zeit eines Taktzyklus (z.B. 2ns) bzw. Taktrate (z.B. 500 Mhz), Inverse des Taktzyklus

Ausführungszeit für ein Programm	=	benötigte Taktzyklen	X	Zeit eines Taktzyklus
Ausführungszeit	=	benötigte	/	Taktrate

für ein Programm Taktzyklen
→ Mehr Performance durch weniger Taktzyklen und der Zeit eines Taktzyklus

CPI: “clock cycles per instruction“

Wieviele Taktschritte benötigt ein Programm ?

Viele Maschinenbefehle benötigen mehr als einen Taktzyklus:

- Multiplikation, Division
- FP-Arithmetik
- Speicherzugriffe

Angabe einer “mittleren“ Anzahl von Taktschritten je Befehl: CPI

Bsp: Performance-Vergleich

- Prozessor A: Taktzyklus = 1ns, CPI = 2.0 für Progr. X
- Prozessor B: Taktzyklus = 2ns, CPI = 1.2 für Progr. X

A und B haben gleichen Befehlssatz.

Welche CPU ist um wieviel schneller ?

CPI-Berechnung

Gewichteter Mittelwert aller individuellen CPIs der Befehle im Programm:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i * C_i)}{\sum_{i=1}^{n} C_i}$$

Befehle der Klasse i mit CPI_i kommen im betrachteten Programm C_i mal vor.

Bsp: CPI-Berechnung

CPU:

Befehlsklasse	A	B	C
CPI	1	2	3

Compiler:

	Häufigkeit der Klasse		
Code	A	B	C
X	2	1	2
Y	4	1	1

Welcher Code ist schneller ?

Performance-Faktoren

Grundlegender Zusammenhang:

Performance-Optimierung

- höhere Taktrate (Technologie)
- geringere CPI (Compiler)
- weniger Befehle pro Programm bzw. Befehle mit geringerer CPI (Compiler)

einseitige Verbesserung eines Faktors ist selten möglich, meist sind Kompromisse erforderlich.

Der ideale Benchmark

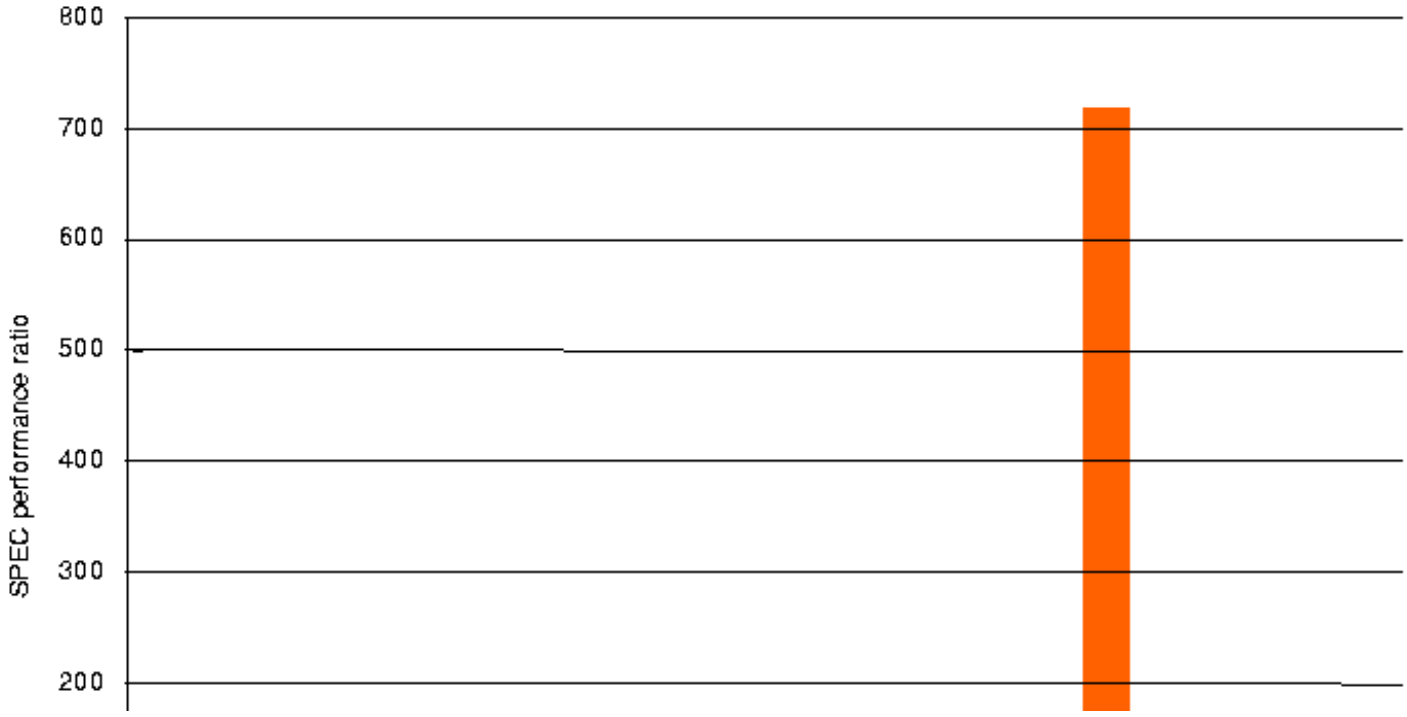
- Echtes Anwenderprogramm
  unbestechlich aber anwenderabhängig

- Einfaches Programm

Einfach nachvollziehbar aber “betrugsanfällig“

- SPEC (System Performance Evaluation Cooperative) allgemein akzeptierter Kompromiß

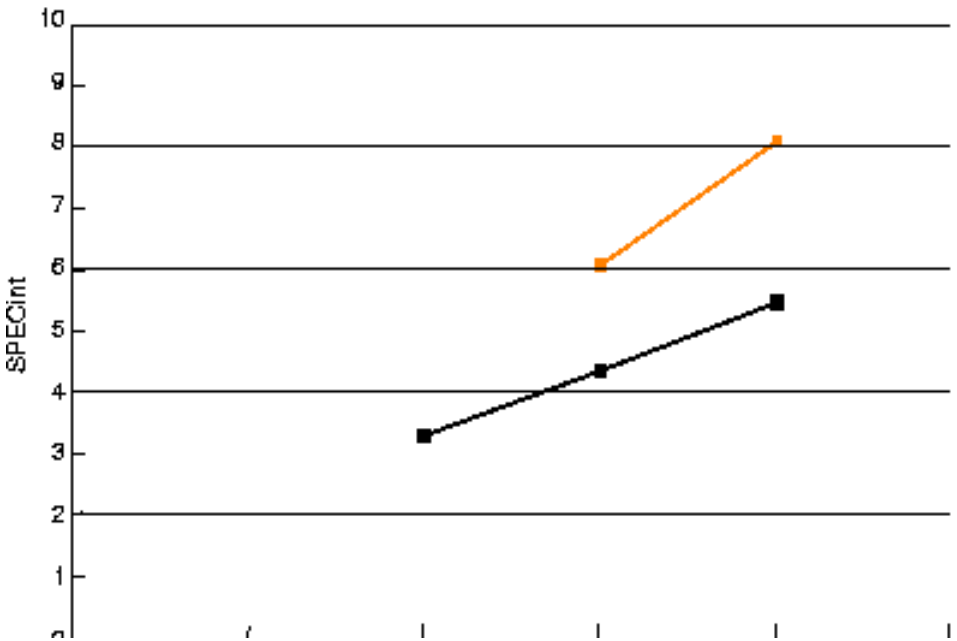
Compileroptimierungen

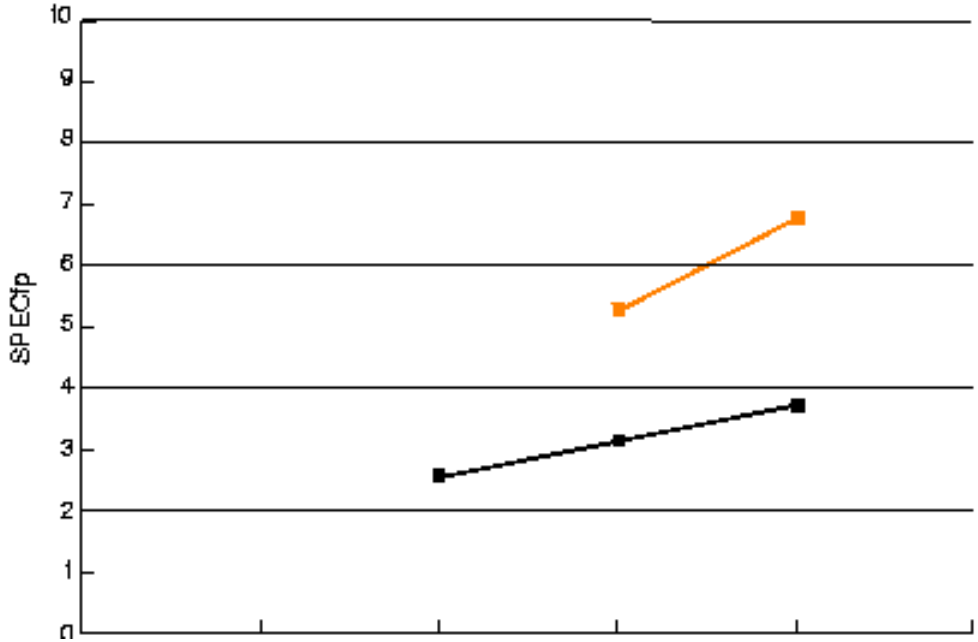


SPEC'95 Benchmark

	Benchmark	Description
int	go	Artificial intelligence; plays the game of Go
	m88ksim	Motorola 88k chip simulator; runs test program
	gcc	The Gnu C compiler generating SPARC code
	compress	Compresses and decompresses file in memory
	li	Lisp interpreter
	ijpeg	Graphic compression and decompression
	perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
	vortex	A database program
FP	tomcatv	A mesh generation program
	swim	Shallow water model with 513 x 513 grid
	su2cor	quantum physics; Monte Carlo simulation
	hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
	mgrid	Multigrid solver in 3-D potential field

Pentium Performance (SPEC’95)





“Gesamt“-Performance

Beispiel:

	Computer A	Computer B
Prog 1	1 s	10 s
Prog 2	1000 s	100 s
Summe	1001 s	110 s

→Der einfachste Weg zur Zusammenfassung mehrerer Einzelmessungen ist das arithmetische Mittel der Ausführungszeiten:

$$\frac{1}{n} * \sum_{i=1}^{n} Time_i$$

N = Anzahl der Programme
$Time_i$ = Ausführungszeit des *i* -ten Programms

Amdahl'sches Gesetz

$$Optimierte\ Ausführungzeit = \frac{optimierter\ Zei\tan teil}{Verbesserungsfaktor} + restl.\ Zei\tan teil$$

Konsequenz:

Optimiere den häufigsten Fall !

Verbreitete Performance-Maße

MIPS
- native: “million instructions per second“
- peak: MIPS für Code mit min CPI
- relative: bezogen auf Referenzrechner

MOPS = “million operations per second“

MFLOPS = “million FP operations per second“

Whetstone, Dhrystone,... synthet. Benchmarks

Beispiel: Native MIPS

Zwei unterschiedliche Compiler generieren für die gleiche Applikation folgende Befehlssequenzen:

	Häufigkeit der Befehlsklassen		
Code von	1 CPI	2 CPI	3 CPI
Compiler 1	$5*10^9$	$1*10^9$	$1*10^9$

Compiler 2	$10*10^9$	$1*10^9$	$1*10^9$

Welcher Code ist schneller ?
Wie sind die entsprechenden native MIPS bei 500 MHz Prozessortakt ?

Zusammenfassung

- Performance abhängig vom gewählten Programm
- Reale Applikationen sind die besten Benchmarks
- Herstellerangaben über Performance oft verzerrt
- Zeit ist das objektivste Performance-Maß
- Optimiere den häufigsten Fall (Amdahl’sches Gesetz)

- Performance-Optimierung erfolgt im Spannungsfeld zwischen Taktrate, CPI und Befehlsanzahl
- Kostenabwägung