2. Performance

- Wichtiges Unterscheidungsmerkmal für Computer
- Schwierig zu erfassen
  - Hardware-Optimierungen machen die Maschinen immer komplexer.
  - Es ist praktisch unmöglich, die Ausführungszeit für ein gegebenes Programm anhand der Datenblätter für den Prozessor zu ermitteln.
- Schlüssel zum Verständnis der zugrunde liegenden Computer-Organisation
  - Wir wollen verstehen, wie Architekturmerkmale die Performance beeinflussen.
    - Wieso ist eine Hardware besser als eine andere für bestimmte Programme?
    - Wie beeinflusst der Befehlssatz einer Maschine die Performance?

Beispiel: Performance von Flugzeugen

Flugzeug	Passagiere	Reich-weite (mi)	Geschwindig-keit (mph)	Passagier-durchsatz (Passagiere * mph)
Boeing 777	375	4630	610	228750
Boeing 747	470	4150	610	286700
BAC/Sud Concorde	132	4000	1350	178200
Douglas DC-8-50	146	8720	544	79424

- **Verschiedene Fragen können gestellt werden.**
  - Wie viel schneller ist die Concorde verglichen mit einer 747?
  - Wie viel mehr Passagiere kann eine 747 transportieren als eine DC-8?
  - Wie viele Passagiere schafft man von A nach B?

Computer Performance

- **Antwortzeit (Latenzzeit, latency)**
  - Wie lange benötigt mein Programm für **einen** Durchlauf?
  - Wie lange muss ich warten, bis mein Programm startet?
  - Wie lange muss ich auf eine Antwort von der Datenbank warten?
- **Durchsatz (throughput)**
  - Welche Arbeit bekomme ich in welcher Zeit erledigt?
  - Wie lange braucht meine Maschine im Durchschnitt?
- **Fragen**
  - Was wird verbessert, wenn wir einen neuen, schnelleren Prozessor in unseren Computer stecken?
  - Was wird verbessert, wenn wir einen weiteren Computer ins Labor stellen?
  - Wieso beschreiben Antwortzeit und Durchsatz verschiedene Dinge?

Definition der Performance

- **Performance**
  - Für ein Programm, das auf einer Maschine A läuft, definieren wir

$$\text{Performance}_A = 1 / \text{Ausführungszeit}_A$$

- **Relative Performance**
  - "A ist *n*-mal schneller als B"

$$\text{Performance}_A \ / \ \text{Performance}_B = \text{Ausführungszeit}_B / \text{Ausführungszeit}_A = n$$

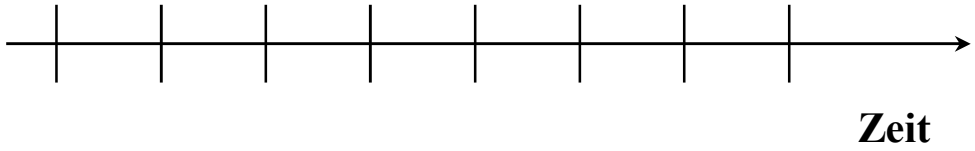
- **Wie misst man Ausführungszeiten?**
  - verschiedene Möglichkeiten

Programm-Ausführungszeiten

- Verstrichene Zeit (*elapsed time, response time*)
  - wichtig, falls die Antwortzeit entscheidend ist
  - es zählt alles (Platten- und Speicherzugriffe, Ein-/Ausgabe, etc.)
  - nützliche Angabe, aber schlecht für Vergleiche von Prozessorarchitekturen geeignet
- CPU Zeit
  - zählt nicht Ein-/Ausgabe oder die Rechenzeit, die für die Ausführung anderer Programme aufgewandt wird
  - setzt sich zusammen aus
    - Systemzeit (*system time*)
    - Benutzerzeit (*user time*)
- Unser Fokus: Benutzer-CPU-Zeit
  - Zeit, die der Prozessor für die Ausführung **unserer** Codezeilen benötigt

Taktzyklen

- Statt Ausführungszeit in Sekunden verwendet man auch Taktzyklen.
- Zeitmarken (*ticks*) zeigen an, wann Aktivitäten *beginnen*.



- Zykluszeit := Zeit zwischen zwei Ticks
- Taktrate (Frequenz, *clock rate*) := Anzahl der Zyklen pro Sekunde (1Hz = 1 Zyklus/Sekunde)
- Beispiel: ein 2,6 GHz Takt hat eine Zykluszeit von

$$\frac{1}{2{,}6\times10^{9}\,\mathrm{Hz}} = 0{,}385\times10^{-9}\,\mathrm{s} = 385\mathrm{ps}$$

Performance-Steigerung

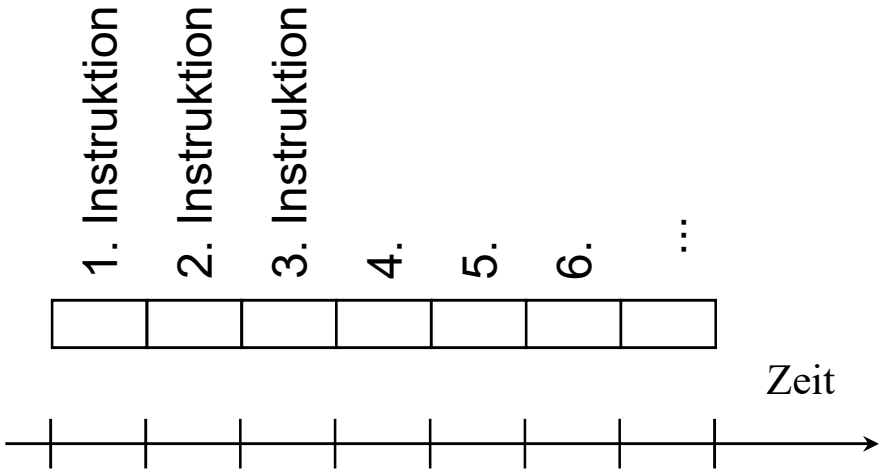
– CPU Laufzeit

$$\text{CPU Laufzeit} = \text{CPU Taktzyklen} \times \text{Zykluszeit}$$
$$= \frac{\text{CPU Taktzyklen}}{\text{Taktfrequenz}}$$

– Um die Performance zu steigern, kann man
  • Anzahl der Taktzyklen für das Programm reduzieren
  • Zykluszeit reduzieren (bzw. die Taktfrequenz erhöhen)
– Beide Ziele widersprechen sich teilweise.
  • Um die Zykluszeit zu reduzieren, kann es notwendig sein, die Anzahl der Zyklen für einige Befehle zu erhöhen.
  • Guter Kompromiss muss gefunden werden.

Instruktionen und Zyklen

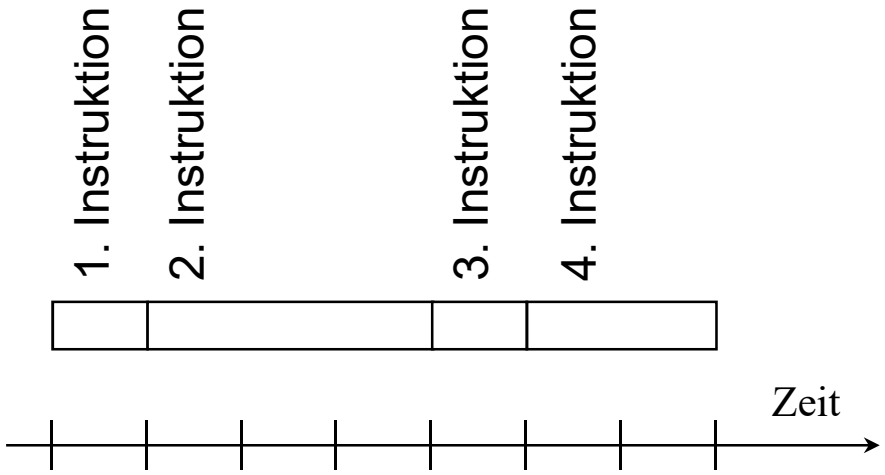
– Anzahl der Taktzyklen = Anzahl Instruktionen (Maschinenbefehle)?



- Im Allgemeinen ist das **falsch.**
  - Verschiedene Maschinenbefehle brauchen unterschiedlich lange.
  - Hängt von der konkreten Maschine ab.

Instruktionen und Zyklen (2)

- **Typisch**



  - Multiplikation dauert länger als Addition.
  - Floating Point Operationen benötigen länger als Integer Operationen.
  - Zugriff auf Speicher dauert länger als Zugriff auf Register.
- **Parallelisierung**
  - z.B. Pipelining
    - Befehle werden schon begonnen, während andere Befehle noch bearbeitet werden.

Zyklen pro Instruktion

$$\text{CPU Taktzyklen} = \text{Anzahl Instruktionen} \times \frac{\text{durchschnittl. Anzahl der Zyklen}}{\text{pro Instruktion}}$$

- **Durchschnittliche Anzahl der Zyklen pro Instruktion**
  - abgekürzt: CPI (*cycles per instruction*)
  - ist ein Durchschnittswert **bezogen auf ein gegebenes Programm**
  - dient zum Vergleich verschiedener Implementierungen derselben ISA (*instruction set architecture*)

Weitere Maßzahlen

- **MIPS (Million Instructions Per Second)**
  - Millionen Instruktionen pro Sekunde
  - bezieht sich auf Maschinen-Befehle und Integer-Operationen
- **FLOPS (Floating Point Operations Per Second)**
  - Anzahl Gleitkommaoperationen pro Sekunde
  - wichtig für Number-Crunching Anwendungen
    - Wettervorhersage
    - 3d-Simulationen aller Art
    - etc.
  - heute eher in Giga- oder Teraflops angegeben

Zusammenfassung Taktzyklen

- Ein gegebenes Programm benötigt
  - gewisse Anzahl von Maschinenbefehlen
  - gewisse Anzahl von Taktzyklen
  - gewisse Anzahl von Sekunden
- Maßzahlen
  - Zykluszeit (Sekunden pro Taktzyklus)
  - Taktfrequenz (Zyklen pro Sekunde)
  - CPI (Zyklen pro Instruktion)
  - MIPS (Millionen Instruktionen pro Sekunde)
  - FLOPS (Floating Point Operationen pro Sekunde, z.B. Teraflops)

$$\text{CPU Zeit} = \frac{\text{Instruktionen}}{\text{Programm}} \times \frac{\text{Taktzyklen}}{\text{Instruktion}} \times \frac{\text{Sekunden}}{\text{Taktzyklus}}$$

Benchmarks

- engl.: *bench*
  - deutsch: Bank, Werkbank, *Richterbank*, *Richter*
- Am besten bestimmt man die Performance durch Messungen an einer echten Anwendung.
  - dient zum Vergleich verschiedener Architekturen
  - z.B. als Kriterium für eine Kaufentscheidung
- Testprogramme sollten ähnliches Verhalten wie die eigene Anwendung haben.
  - z.B. Compiler, wissenschaftliche Anwendungen (*number crunching*), Grafikprogramme, etc.

Benchmarks (2)

- Benchmarks
  - gut für Hardware-Designer und –Architekten
  - können aber auch missbraucht werden
    - Hardware/Compiler gezielt für spezielle Benchmarks entwickelt.
    - Hardware/Compiler erkennt Benchmark und gibt das richtige Ergebnis zurück, ohne es aufwändig zu berechnen.
    - Hardware/Compiler erkennt Benchmark und führt spezielle Optimierungen durch, die im allgemeinen nicht benutzt werden können.
    - Beispiel Intel (im Jahr 2003)
      - » Hoch optimierender Intel Compiler fragte Herstellername des Prozessors ab und compilierte Benchmarkprogramme anders (ohne SSE2, mit langsamen Floating-Point Operationen), wenn der Prozessor, der zwar SSE2 hatte, nicht von Intel kam.
      - » Sollte offenbar AMD-Prozessoren schlechter aussehen lassen.

Beispiel für Grafikkarten Benchmark-Betrug

- **Meldung vom 26.5.2003**
  - siehe: http://www.heise.de/newsticker/data/tol-26.05.03-001/
  - Nvidia nutzte in Treibern für GeForce FX Chips offenbar aus, dass beim Benchmark „Mother Nature“ die Kameraposition bekannt ist.
  - Verdeckte Szenen werden gar nicht gerendert.
  - Wird sichtbar, wenn Kameraposition in Debug-Versionen des Benchmarks verändert wird.

Beispiel für Benchmark-Betrug (2)

- Unzulässiger Vorteil gegenüber anderen Grafikchipherstellern
- Nach Patch, der bewirkt, dass der Benchmark nicht erkannt wird:
  - Leistung der Nvidia GeForce FX 5900 Ultra geht von 37 auf 19 Bilder/s zurück
  - zum Vergleich: ATI Radeon 9800 Pro: 34 Bilder/s

Benchmarks (3)

- **Zusammenfassen von Performance-Angaben**
  - Daten stammen von mehreren Einzelprogrammen.
  - Interesse an einer einzelnen Zahl, die die Gesamtperformance ausdrückt.
  - Wie fasst man die Einzelergebnisse zusammen?

	Computer A	Computer B
Programm 1 [s]	**1**	**8**
Programm 2 [s]	**100**	**50**
Gesamtzeit [s]	**101**	**58**

Benchmarks (4)

- Gesamtlaufzeit ist die einzig relevante Größe
  - arithmetischer Mittelwert ist proportional zur Gesamtlaufzeit

$$\text{AM} = \frac{1}{n}\sum_{\text{i=1}}^{\text{n}} \text{Zeit}_i$$

  - gewichteter Mittelwert, falls bekannt ist, wie groß die Anteile der einzelnen Programme am Gesamtrechenaufwand sind

$$\text{AM} = \sum_{\text{i=1}}^{\text{n}} w_i \times \text{Zeit}_i \quad \text{mit} \quad \sum_{i=1}^{n} w_i = 1$$

- Alternative: geometrischer Mittelwert

$$\text{GM} = \sqrt[n]{\prod_{i=1}^{n} \text{Zeit}_i}$$

Benchmarks (5)

- Performance-Angaben werden häufig auf eine Referenz-Maschine bezogen (siehe z.B. SPEC, s.u.)
- arithmetische Mittelwerte von relativen Rechenzeiten hängen davon ab, auf welche Maschine normalisiert wurde
- Beispiel

	Zeit auf A	Zeit auf B	Normalisiert auf A		Normalisiert auf B	
			A	B	A	B
Progr. 1	**1**	**8**	**1**	**8**	**0,125**	**1**
Progr. 2	**100**	**50**	**1**	**0,5**	**2**	**1**
Arith. Mittel	**50,5**	**29**	**1**	**4,25**	**1,0625**	**1**
Geom. Mittel	**10**	**20**	**1**	**2**	**0,5**	**1**

- arithmetisch: inkonsistent: einmal ist B schneller als A, einmal A schneller als B
- geometrisch: konsistente Ergebnisse A doppelt so schnell wie B

Benchmarks (6)

- Geometrischer Mittelwert
  - Vorteil: unabhängig von Bezugsmaschine, wegen

$$\frac{\mathrm{GM}(A_i)}{\mathrm{GM}(B_i)} = \frac{\sqrt[n]{\prod_{i=1}^{n} A_i}}{\sqrt[n]{\prod_{i=1}^{n} B_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{A_i}{B_i}} = \mathrm{GM}\left(\frac{A_i}{B_i}\right)$$

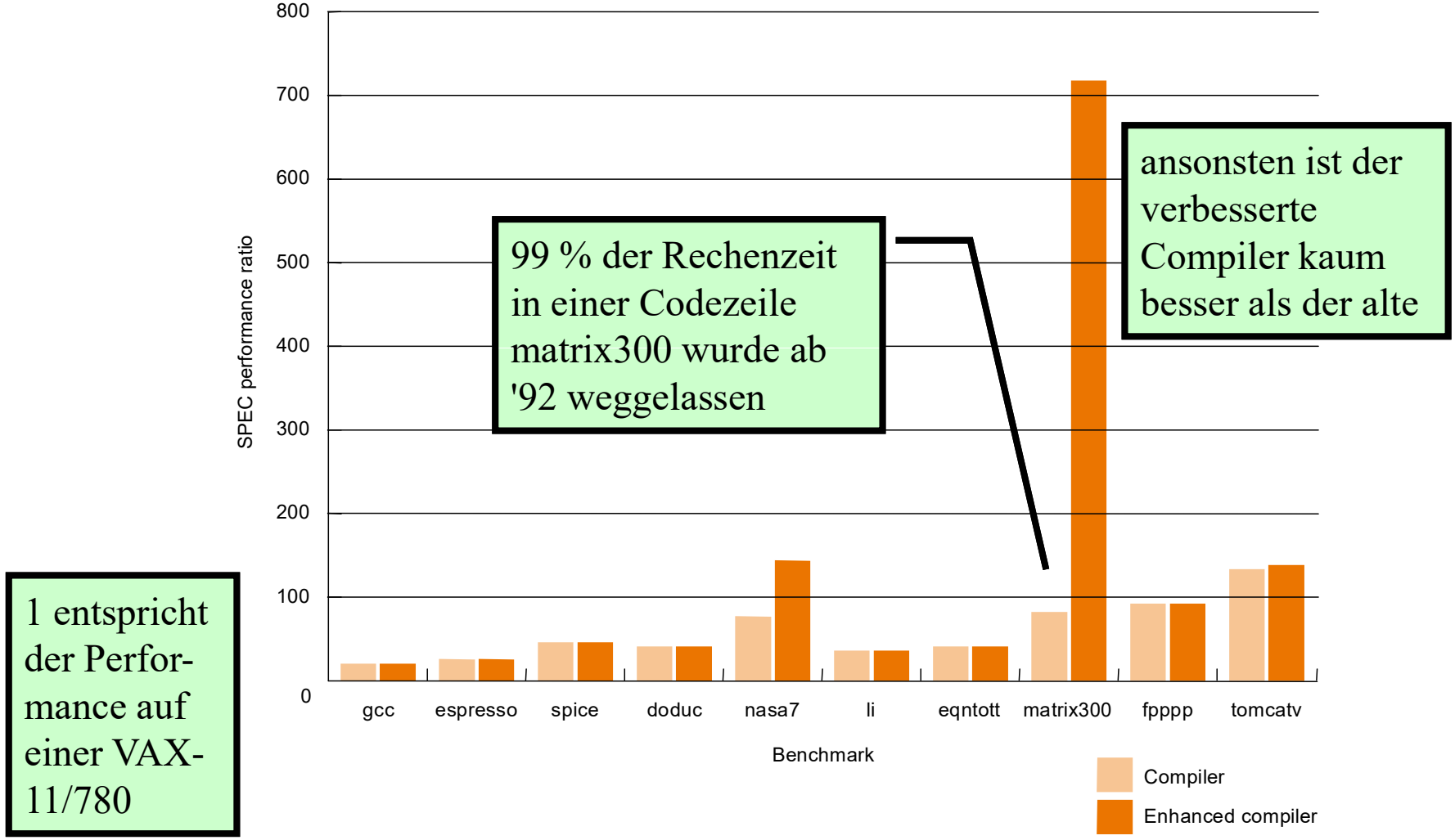
  - wird deshalb in SPEC-Benchmarks (s.u.) benutzt
  - Nachteil: sagt nichts über die absolute Rechenzeit aus
- Arithmetischer Mittelwert
  - Vorteil: proportional zur Rechenzeit
  - Ist aber auch nur richtig, wenn man die konkreten Gewichte für die Benchmarks kennt.

SPEC

- *System Performance Evaluation Cooperative*
  - Firmenkonsortium hat sich auf einen Satz von real-world-Programmen und Eingabedaten geeinigt, mit denen die Performance verschiedener Maschinen gemessen werden soll.
  - Wertvoller Indikator für Rechner Performance und Compiler-Technologie

SPEC '89

- **Beispiel: Verbesserungen an einem Compiler**

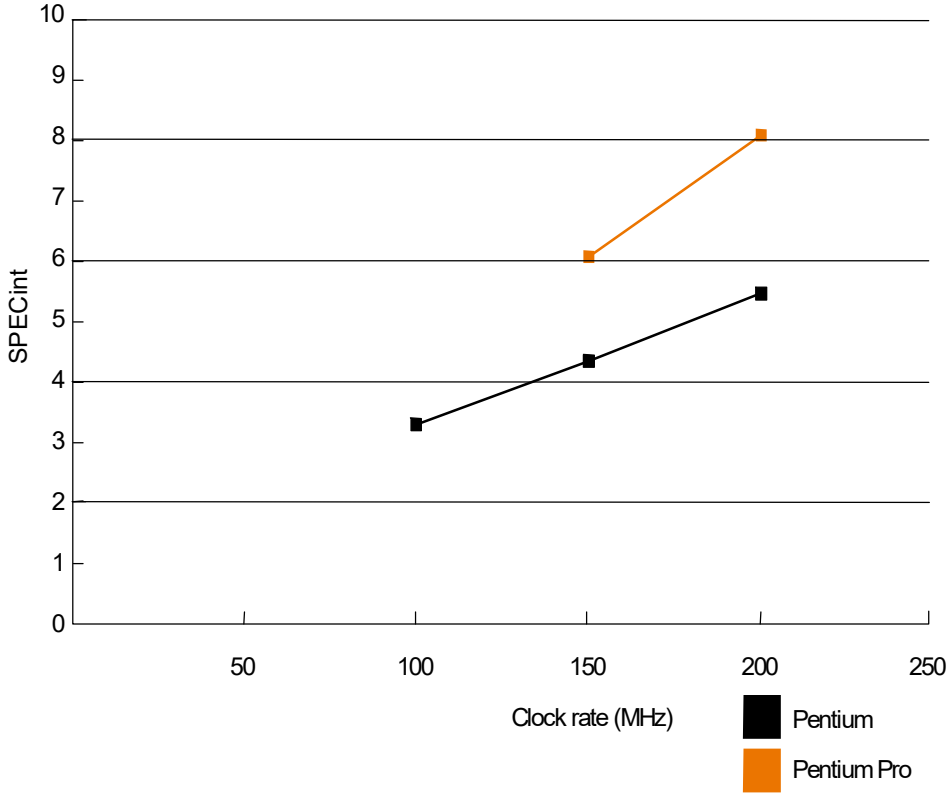


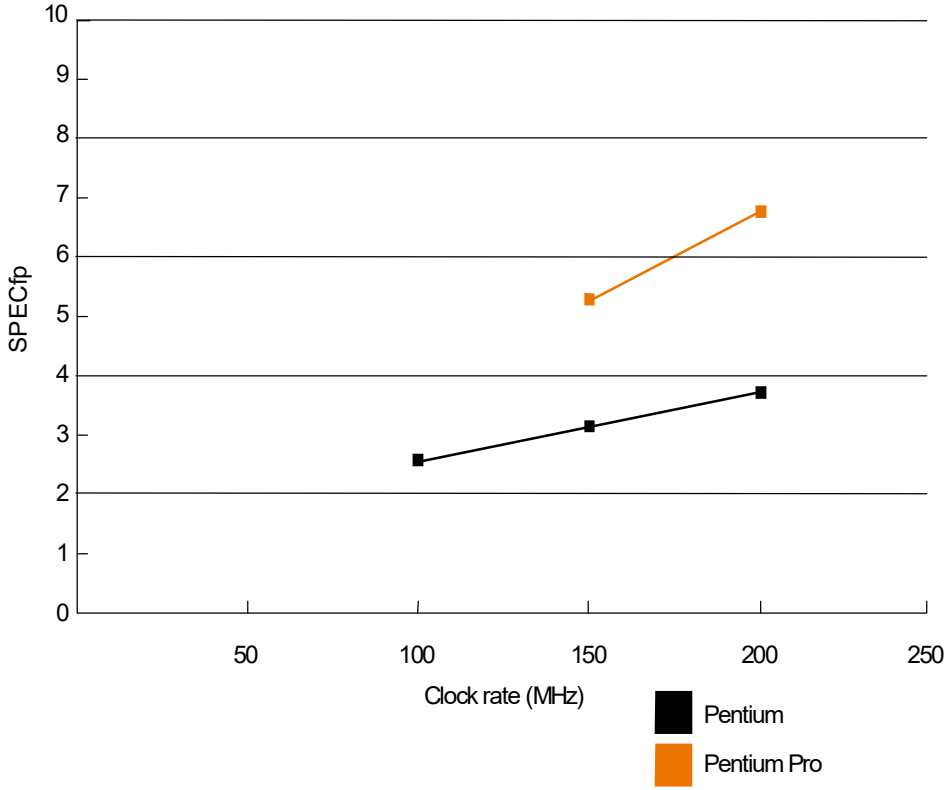
SPEC '95

Benchmark	Description	
go	Artificial intelligence; plays the game of Go	int
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	
tomcatv	A mesh generation program	fp
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	
applu	Parabolic/elliptic partial differential equations	
trub3d	Simulates isotropic, homogeneous turbulence in a cube	
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant	
fpppp	Quantum chemistry	
wave5	Plasma physics; electromagnetic particle simulation	

SPEC '95 (2)

- Führt eine Verdopplung der Taktrate zu einer Verdopplung der Performance?
- Kann eine Maschine mit geringerer Taktrate eine höhere Performance haben?





Amdahl's Law

Ausführungszeit nach einer Verbesserung
= Ausführungszeit der nicht verbesserten Teile
+ Ausführungszeit der verbesserten Teile / Verbesserungsfaktor

– Beispiel:
  - Ein Programm läuft 100s, davon werden 80s durch Multiplikationen verbraucht.
  - Wie schnell müssen wir die Multiplikation machen, damit das Programm 4 mal schneller läuft?

16 mal: 20s + 80s/16 = 25s

  - Wie ist es mit 5 mal schneller?

∞ mal: 20s + 80s/ ∞ = 20s