UT 1. Introduction to Computer Architecture

Lecture 1.2 Performance Evaluation

J. Duato, J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Department of Computer Engineering
Universitat Politècnica de València

Contents

Bibliography


John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

Contents

Time and Throughput

Two viewpoints

User point of view: *Users want to finish as soon as possible*

↓ *Response time* or *Execution time*: time to complete a task.

System manager point of view: *System managers want to complete as many tasks as possible per unit of time*

↑ *Throughput*: Operations/executions completed per time unit.

Relationship between viewpoints:

$$\text{Throughput} = \frac{1}{\text{Execution time}}$$

Comparisons (1/2)

Comparing alternatives:

- When two computers (X and Y) are compared, the slowest one Y is taken as reference
- When a set of designs ($X_1, \ldots, X_n$) is studied, a computer Y (in the set or not) is selected as reference

In order to compare computers X and Y, a workload must be selected in order to measure their respective performances under similar execution conditions. Attending to the type of measure, the result could be:

- Execution time: T_X and T_Y
- Throughput: P_X and P_Y

Comparisons (2/2)

The relation S is computed as:

$$S = \frac{T_Y}{T_X} = \frac{P_X}{P_Y} = 1 + \frac{n}{100}$$

S represents the *speedup* and it must be interpreted as follows:

- "X is S times faster than Y"
- "X is n % faster than Y"

Contents

Execution time equation (1/2)

$$T_{\text{exec}} = \frac{\text{sec}}{\text{program}} = \frac{\text{num. of instr.}}{\text{program}} \times \frac{\text{cycles}}{\text{num. of instr.}} \times \frac{\text{sec}}{\text{cycle}} = I * CPI * T$$

- All three parameters are related:
  - I = f(instruction set architecture, compiler)
  - CPI = f(instruction set architecture, organization)
  - T = f(technology, organization)

  $\Rightarrow$ It is not possible to reduce one of these parameters without affecting the others.

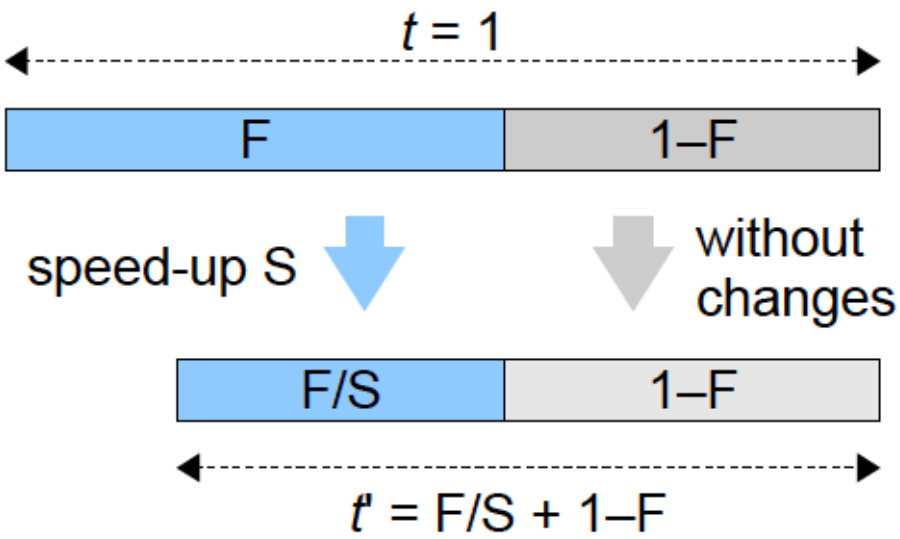
Execution time equation (2/2)

Examples:

- A richer instruction set can reduce I, but it can make the organization and implementation of the computer more complex, thus increasing CPI and/or T.
- A more complex computer organization can reduce CPI, but can increase the number of logic gates traversed per cycle, thus increasing T.

Amdahl's law (1/4)

How does a change in a part of a system/process affect the whole?



- F is the fraction of time affected by the improvement
- S is the applied speed-up, i.e. the factor improving F

In general, for values of $t \neq 1$, the execution time is:

$$t' = t \cdot (1 - F) + \frac{t}{S} \cdot F$$

Amdahl's law (2/4)

The resulting global speed-up S' will be:

$$S' = \frac{t}{t'} = \frac{1}{(1-F) + \frac{F}{S}}$$

The fraction of time that is not improved $(1 - F)$ defines an upper bound to the maximum reachable speed-up:

$$S'_{\infty} = \lim_{S \to \infty} S' = \frac{1}{1-F}$$

Amdahl's law (3/4)

Amdahl's law can be generalized for multiple fractions (n), each of them being accelerated with a different speedup. Thus,

$$S' = \frac{1}{\frac{F_1}{S_1} + \frac{F_2}{S_2} + ... + \frac{F_n}{S_n}}$$

where $F_1 + F_2 + ... + F_n = 1$.

Additionally, a given speedup S_i can be the result of combining several independent speedups (m_i). Thus,

$$S_i = S_{i,1} \times S_{i,2} \times ... \times S_{i,m_i}$$

Amdahl's law (4/4)

Example: The execution time of a program P is composed of two fractions F_1 and F_2, where F_2 is parallelizable. Originally, P is executed on a processor with 2 cores working at 2.2 GHz. To speedup the execution, the processor is replaced with a 4-core processor running at 3.3 GHz. What is the expression to compute the global speedup of the update?

Solution:

$$S' = \frac{1}{\frac{F_1}{S_1} + \frac{F_2}{S_2}}$$

$$S_1 = \frac{3.3}{2.2} = 1.5 \qquad S_2 = \frac{3.3}{2.2} \times \frac{4}{2} = 3$$

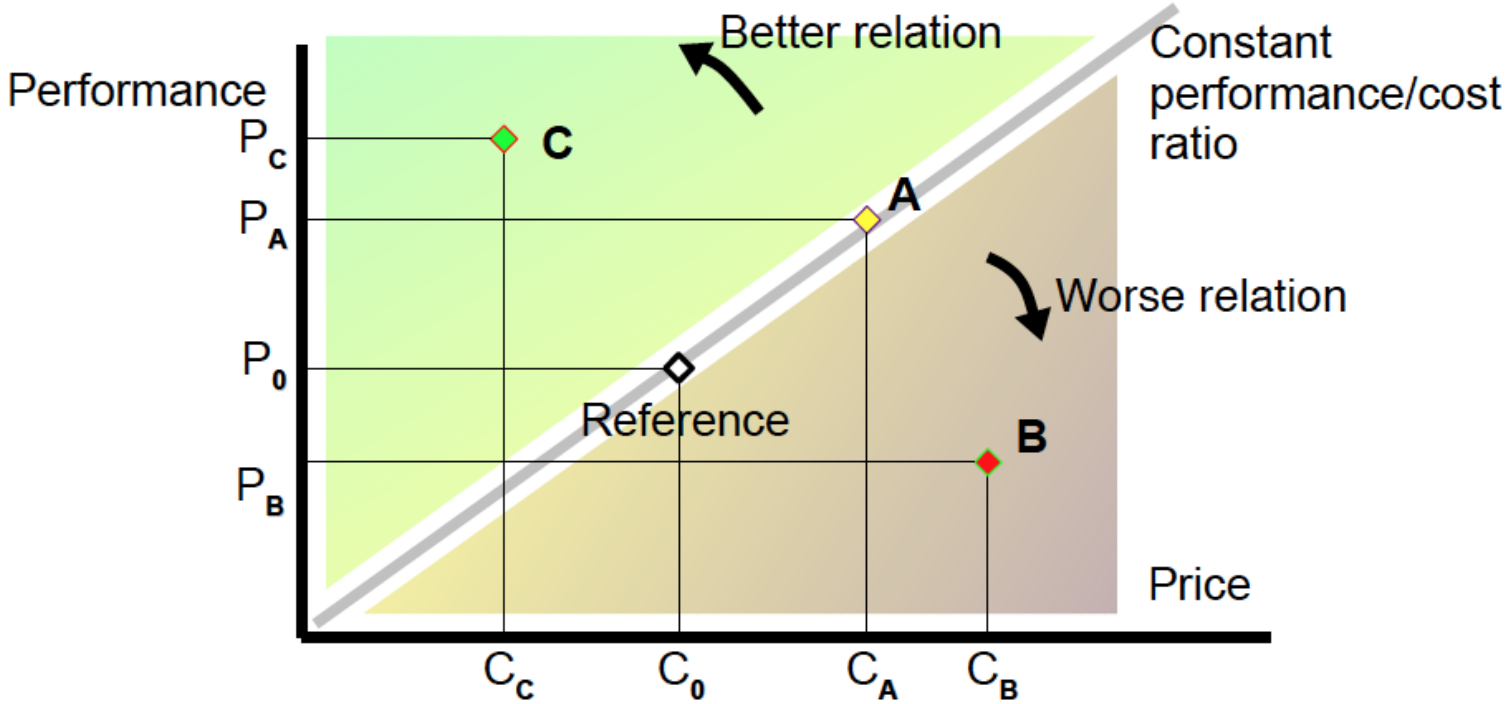
Amdahl's law examples

- In programming: Most of the execution time of a program is spent running a particular fragment of code.
  Locality principle: 90% of the time, the computer executes 10% of the code
  $\Rightarrow$ It is convenient to optimize the most frequently executed code
- Instruction set design: Which instructions are more frequently executed?
- Multiprocessor systems: Which fraction of a program can be executed in parallel?
- In general, Amdahl's law applies to the design of the various parts of a computer

Relation between Cost and Performance

Measures the relation between performance and cost (price, wats, . . .) provided by a system configuration.
It enables the comparison of different alternatives.

Example: Given two system configurations



Contents

Workloads

Performance is measured with a program or a collection of programs that are likely relevant to the user.
The source code of such program(s) must be available. It must be compiled for each computer under analysis.
The best option:

- Real programs

Alternative options (not so popular anymore):

- *Kernels*. Fragments of code extracted from real programs. Examples: *Livermore Loops* and *Linpack*.
- *Toy benchmarks*. Simple programs with well-known execution results. Examples: *Quicksort*, *Puzzle*, etc.
- *Synthetic Benchmarks*. Programs written in order to represent the average program typically run in a system. Examples: *Whetstone*, *Dhrystone*, etc.

Benchmark suites

Basic structure: Typically kernels and real non-interactive programs defined to measure performance attending to a given user profile.
Some examples: SPEC (CPU, graphics, servers, etc), MediaBench (multimedia), TPC-xx (Transactions), EEMBC (embedded) ...

Must be easy to update: Programs in the suite must represent *at any moment* the type of tasks typically run in the system by a regular users.
$\rightarrow$ Benchmarks are periodically updated.

Reproducibility: Measures must be reproducible
$\rightarrow$ All details must be clearly defined:

- hardware: processor, cache, memory, disk, ...
- software: operating system, programs and versions, input data, execution/compilation options, ...

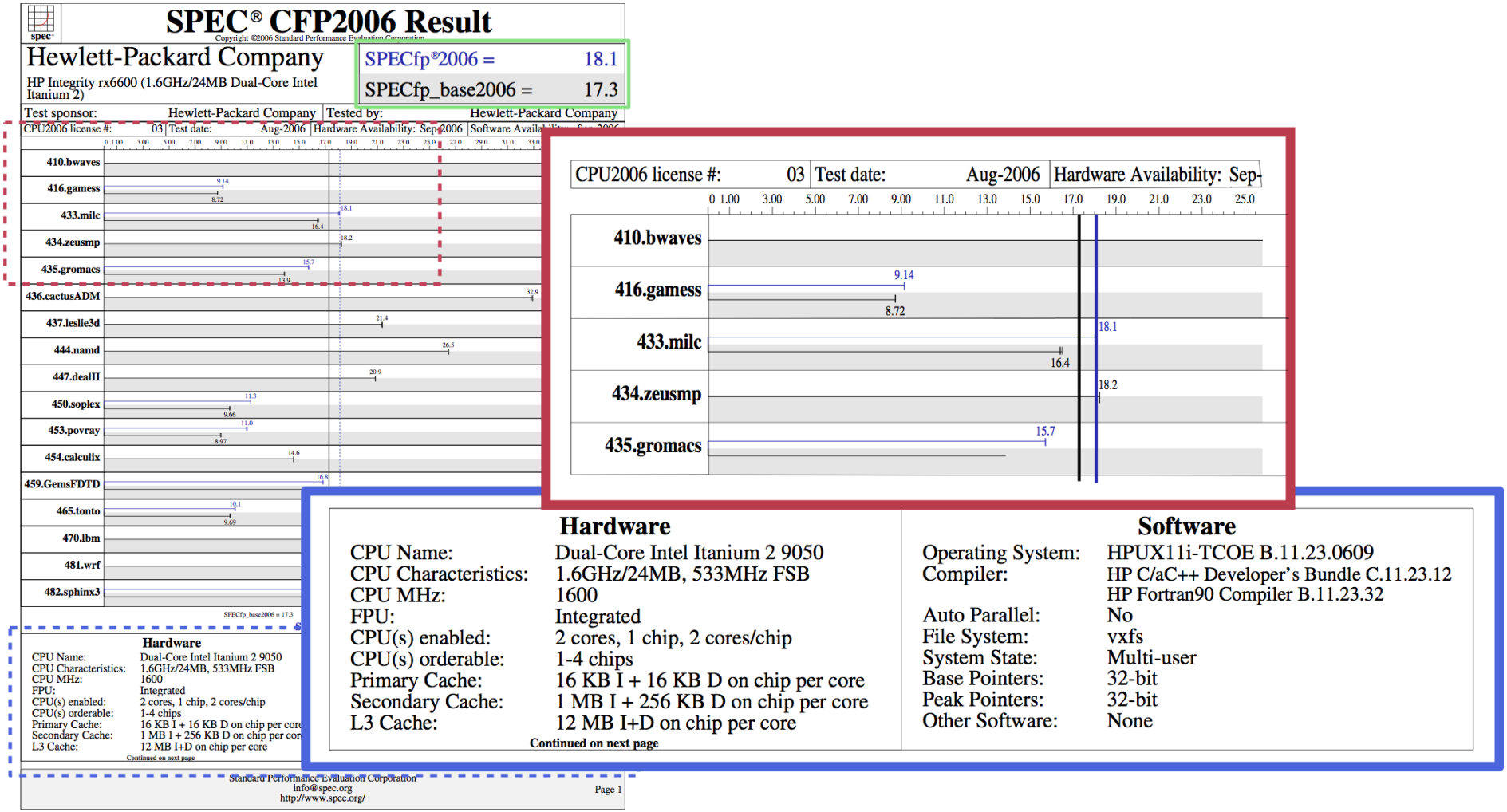
Example: SPEC CPU95-**CINT95** Benchmarks

Benchmark	Application Area	Specific Task
099.go	Game playing	Plays the game Go against itself
124.m88ksim	Simulation	Simulates the M88100 processor running test programs
126.gcc	Program compilation	Compiles pre-processed source code into optimized SPARC assembly code
129.compress	Compression	Compresses large text files (about 16MB) using adaptive Limpel-Ziv coding
130.li	Language interpreter	Lisp interpreter
132.ijpeg	Imaging	Performs jpeg image compression with various parameters
134.perl	Shell interpreter	Performs text and numeric manipulations (anagrams/prime number factoring)
147.vortex	Database	Builds and manipulates three interrelated databases

Example: SPEC CPU95-**CFP95** Benchmarks

Benchmark	Application Area	Specific Task
101.tomcatv	Fluid Dynamics & Geometric Translation	Generation of 2D coordinate system around general geometric domains
102.swim	Weather Prediction	Solves shallow water equations using finite difference approximations (SP)
103.su2cor	Quantum Physics	Masses of elementary particles are computed in the Quark-Gluon theory
104.hydro2d	Astrophysics	Hydrodynamical Navier Stokes equations are used to compute galactic jets
107.mgrid	Electromagnetism	Calculation of a 3D potential field
110.applu	Fluid Dynamics/Math	Solves matrix system with pivoting
125.turb3d	Simulation	Simulates turbulence in a cubic area
141.apsi	Weather Prediction	Calculates statistics on temperature and pollutants in a grid
145.fpppp	Chemistry	Performs multi-electron derivatives
146.wave	Electromagnetics	Solves Maxwell's equations on a cartesian mesh

Example: Results



SPEC® CFP2006 Result

Copyright ©2006 Standard Performance Evaluation Corporation

Hewlett-Packard Company

HP Integrity rx6600 (1.6GHz/24MB Dual-Core Intel Itanium 2)

SPECfp®2006 = 18.1

SPECfp_base2006 = 17.3

Test sponsor: Hewlett-Packard Company | Tested by: Hewlett-Packard Company

CPU2006 license #: 03 | Test date: Aug-2006 | Hardware Availability: Sep-2006 | Software Availability: Sep-2006

410.bwaves
416.gamess 9.14 8.72
433.milc 18.1 16.4
434.zeusmp 18.2
435.gromacs 15.7 13.9
436.cactusADM 32.9
437.leslie3d 21.4
444.namd 26.5
447.dealII 20.9
450.soplex 11.3 9.66
453.povray 11.0 8.97
454.calculix 14.6
459.GemsFDTD 16.8
465.tonto 10.1 9.69
470.lbm
481.wrf
482.sphinx3

SPECfp_base2006 = 17.3

Hardware

CPU Name:	Dual-Core Intel Itanium 2 9050
CPU Characteristics:	1.6GHz/24MB, 533MHz FSB
CPU MHz:	1600
FPU:	Integrated
CPU(s) enabled:	2 cores, 1 chip, 2 cores/chip
CPU(s) orderable:	1-4 chips
Primary Cache:	16 KB I + 16 KB D on chip per core
Secondary Cache:	1 MB I + 256 KB D on chip per core
L3 Cache:	12 MB I+D on chip per core

Continued on next page

Software

Operating System:	HPUX11i-TCOE B.11.23.0609
Compiler:	HP C/aC++ Developer's Bundle C.11.23.12 HP Fortran90 Compiler B.11.23.32
Auto Parallel:	No
File System:	vxfs
System State:	Multi-user
Base Pointers:	32-bit
Peak Pointers:	32-bit
Other Software:	None

Standard Performance Evaluation Corporation
info@spec.org
http://www.spec.org/

Page 1

Comparison of computers (1/2)

How to capture the behavior of several programs with a single measure?

$\rightarrow$ Feature of a good time metric: the average value must be directly proportional to the execution time

- Total execution time (Sum of execution times)

$$T_T = \sum_{i=1}^{n} \text{Time}_i$$

- Arithmetic mean

$$T_A = \frac{1}{n} \sum_{i=1}^{n} \text{Time}_i$$

Comparison of computers (2/2)

- Sum of weighted execution times

$$T_W = \sum_{i=1}^{n} w_i * \text{Time}_i$$

where w_i represents the frequency of program i in the considered workload.

- (SPEC) the geometric mean of execution times normalized wrt a reference machine $\rightarrow$ R times faster than the reference:

$$R = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Time}_{\text{ref}}}{\text{Time}_i}}$$

Contents

MIPS (1/3)

MIPS = Millions of Instructions per Second

$$\text{MIPS} = \frac{\text{number of instructions executed}}{T_{\text{execution}} * 10^6} =$$

$$= \frac{I}{I * CPI * T * 10^6} = \frac{1}{CPI * T * 10^6} = \frac{f}{CPI * 10^6}$$

- It is an intuitive measure proportional to performance.
- It does not account for the number of executed instructions
- Depends on the considered programs. Different programs execute different instructions, of different complexity and execution time.
  $\rightarrow$ ¡But the executed program is rarely mentioned!

MIPS (2/3)

- Depends on the instruction set. The same program executes different number of instructions in each machine, according to the complexity of its instruction set
  $\rightarrow$ not suitable for comparing machines with different instruction set.
- ¡May be inversely proportional to performance!
  Example: comparing two computers, one with, and another without, a floating point co-processor $\rightarrow$ more performance: **with** coprocessor; more MIPS **without** coprocessor.

MIPS (3/3)

Given a program executing n millions of instructions with a coprocessor and $m > n$ millions of instructions without coprocessor. Times required by a coprocessor instruction and a regular one are t_c and t, respectively. As a result:

	Without coprocessor	With coprocessor
Num. of instr.:	m instr	n instr
T_{exec}	mt	nt_c
MIPS	$\frac{m}{mt \times 10^6}$	$\frac{n}{nt_c \times 10^6}$

As $t_c > t$, $\frac{1}{t} > \frac{1}{t_c} \Rightarrow \text{MIPS}_{without\_coprocessor} > \text{MIPS}_{with\_coprocessor}$

MFLOPS (1/2)

Millions of floating point operations per second.

$$\text{MFLOPS} = \frac{\text{num. of floating point operations in the program}}{T_{\text{execution}} * 10^6}$$

- It accounts for operations instead of instructions: the execution of the same program on different architectures will require a different number of instructions but the same number of FP operations.
- It cannot be applied to programs that are not carrying out any floating point operations. This is the case of text processors and compilers

MFLOPS (2/2)

- It depends on the FP instruction set of each computer, which is not always the same. Example: CRAY-2 provides no division; M68882 supports division in addition to squares, sines and cosines.
  $\rightarrow$ the number of FP operations is not kept constant.
  Solution: Rely on *source code* FP operations.
- Different programs ejecute different FP operations and they usually have different costs.