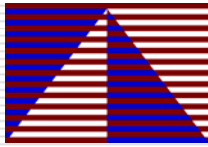


# ECE3055

## Computer Architecture and Operating Systems

### Chapter 4: Performance

---



Prof. Hsien-Hsin Sean Lee  
School of Electrical and Computer Engineering  
Georgia Institute of Technology

## Performance

---

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

*Why is some hardware better than others for different programs?*

*What factors of system performance are hardware related?  
(e.g., Do we need a new machine, or a new operating system?)*

*How does the machine's instruction set affect performance?*

---

## Which of these airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC-8?

## Computer Performance: TIME, TIME, TIME

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?
- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?
- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the lab what do we increase?*

## Execution Time

---

- Elapsed Time
  - counts everything (*disk and memory accesses, I/O , etc.*)
  - a useful number, but often not good for comparison purposes
- CPU time
  - doesn't count I/O or time spent running other programs
  - can be broken up into system time, and user time
- Our focus: user CPU time
  - time spent executing the lines of code that are "in" our program

## Book's Definition of Performance

---

- For some program running on machine X,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- "X is n times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = n$$

- Problem:
  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds

# Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Clock “ticks” indicate when to start activities (one abstraction):



- cycle time = time between ticks = seconds per cycle
- clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)

A 200 MHz clock has a \_\_\_\_\_ cycle time

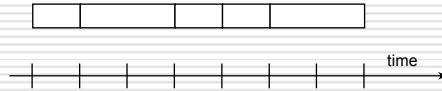
# How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

So, to improve performance (everything else being equal) you can either

- \_\_\_\_\_ the # of required cycles for a program, or
- \_\_\_\_\_ the clock cycle time or, said another way,
- \_\_\_\_\_ the clock rate.

## Different numbers of cycles for different instructions



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes (in general) more time than accessing registers
  
- Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*

## Now that we understand cycles

- A given program will require
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)  
*a floating point intensive application might have a higher CPI*
  - MIPS (millions of instructions per second)  
*this would be higher for a program using simple instructions*

## Program Execution time

$$\text{Execution Time} = \left[ \sum_{i=1}^n (C_i * CPI_i) \right] * \text{clock\_cycle\_time}$$

$$\sim = \underbrace{\text{Instruction\_count}}_{\text{compiler}} * \underbrace{CPI_{\text{avg}}}_{\text{architecture}} * \underbrace{\text{clock\_cycle\_time}}_{\text{technology}}$$

## Assessing Performance

- Ideal CPI is increased by dependencies
- Performance impact on CPI can be assessed by computing the impact on a per instruction basis

$$\text{Increase in CPI} = \text{Base CPI} + \text{Probability\_of\_event} * \text{penalty\_for\_event}$$

- For example, an event may be a branch mis-prediction or the occurrence of a data hazard
- The probability is computed for the occurrence of the event on an instruction
- Examples

## CPI Example

---

- Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0

Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

What machine is faster for this program, and by how much?

- *If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?*

## # of Instructions Example

---

- A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C

The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

Which sequence will be faster? How much?

What is the CPI for each sequence?

## MIPS example

- Two different compilers are being tested for a 100 MHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

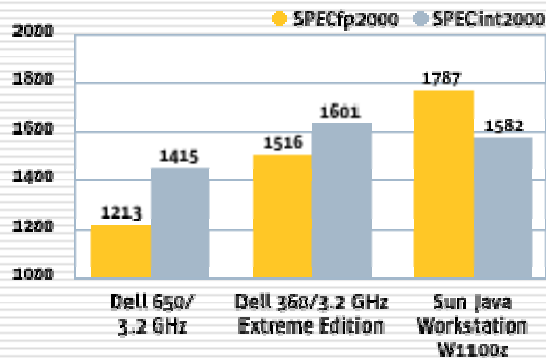
## Benchmarks

- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Typical of representative class of applications
    - Media: MP3 decoder, iTunes,
    - Games: Quake, Unreal, Call of Duty
    - Editing: Clone DVD, Photoshop, 3D Studio
- Synthetic benchmarks
  - Collection of common applications (i.e. Windows)
  - 3DMark, PC Mark, SiSoft Sandra,
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused (Intel's "other" bug)
  - valuable indicator of performance (and compiler technology)
  - Separate into integer benchmark and floating-point benchmark
  - Current version: SPEC2000

# SPEC CPU2000 Benchmark

Benchmark	Source	Application
164.gzip	C	Compression
175.vpr	C	FPGA circuit placement and routing
176.gcc	C	C programming language compiler
181.mcf	C	Combinatorial optimization
186.crafty	C	Game playing: chess
197.parser	C	Word processing
252.eon	C++	Computer visualization
253.perlbnk	C	PERL programming language
254.gap	C	Group theory, interpreter
255.vortex	C	Object-oriented database
256.bz2	C	Compression
300.twolf	C	Place and route simulator
168.wupwise	FORTRAN 77	Quantum chromodynamics
171.swim	FORTRAN 77	Shallow water modeling
172.mgrid	FORTRAN 77	Multi-grid solver
173.applu	FORTRAN 77	Parabolic/Elliptic PDEs
177.mesa	C	3D OpenGL graphics library
178.galgel	FORTRAN 90	Computational fluid dynamics
179.art	C	Image recognition / Neural networks
183.quake	C	Seismic wave propagation simulation
187.facerec	FORTRAN 90	Image processing: face recognition
188.ammp	C	Computational chemistry
189.lucas	FORTRAN 90	Number theory / primality testing
191.fma3d	FORTRAN 90	Finite-element crash simulation
200.sixtrack	FORTRAN 77	High energy nuclear physics accelerator design
301.apsi	FORTRAN 77	Meteorology: pollutant distribution

# SPEC CPU2000 Benchmark Sample Result



Source: Sun Microsystems  
W1100z uses AMD Opteron  
100 series CPU

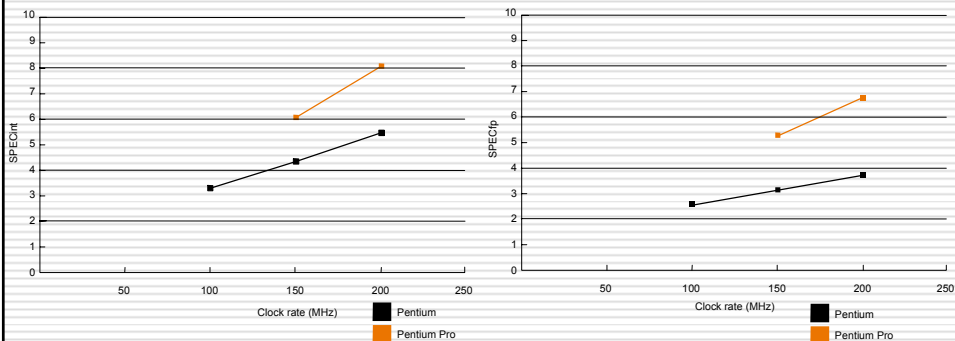
# SPEC '95

Benchmark	Description
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
jpeg	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
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Navier 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

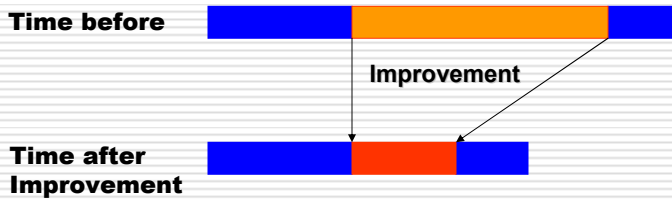
*Does doubling the clock rate double the performance?*

*Can a machine with a slower clock rate have better performance?*



# Amdahl's Law

Execution Time After Improvement =  
 Execution Time Unaffected + ( Execution Time Affected / Amount of Improvement )

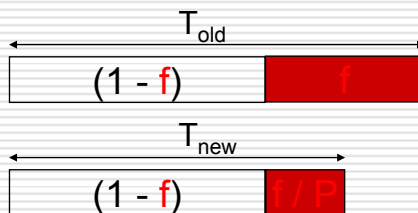


# Amdahl's Law

□ Speed-up =  

$$\text{Perf}_{\text{new}} / \text{Perf}_{\text{old}} = \text{Exec\_time}_{\text{old}} / \text{Exec\_time}_{\text{new}} = \frac{1}{(1-f) + \frac{f}{P}}$$

- Performance improvement from using faster mode is limited by the fraction the faster mode can be applied.



## Example

- "Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

- *Principle: Make the common case fast*

## Example

- Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?
- We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

# Remember

---

- Performance is specific to a particular program/s
    - Total execution time is a consistent summary of performance
  
  - For a given architecture performance increases come from:
    - increases in clock rate (without adverse CPI affects)
    - improvements in processor organization that lower CPI
    - compiler enhancements that lower CPI and/or instruction count
  
  - Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance
  
  - You should not always believe everything you read! Read carefully!  
(see newspaper articles, e.g., Exercise 2.37)
-