

---

## IC220

### Slide Set #12: Performance (Chapter 4)

Which of these airplanes has the best performance?



Airplane	Passengers	Range (mi)	Speed (mph)	Throughput
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

•What percentage faster is the Concorde compared to the 747?

–To the DC-8?

•How does throughput of Concorde compare to 747?

---

## 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?*

---

## Computer Performance:

- Execution / 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 improve?*
- *If we add a new machine to the lab what do we improve?*

## Execution Time

---

- Elapsed Time =
  - 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 is ?

## Clock Cycles

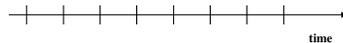
---

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

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

$$\text{CPUtime} = \text{CPUClockCycles} \times \text{ClockCycleTime}$$

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



- Clock Cycle time =
- Clock rate (frequency) =

What is the clock cycle time for a 200 Mhz. clock rate?

## Book's Definition of Performance

---

- For some program running on machine X,  
 $\text{Performance}_X =$
- "X is n times faster than Y"
- Example:
  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds
  - How much faster is A than B?

## Measuring Execution Time

---

EX: 4-1 ...

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

$$\text{CPUtime} = \text{CPUClockCycles} \times \text{ClockCycleTime}$$

Example: Some program requires 100 million cycles. CPU A runs at 2.0 GHz. CPU B runs at 3.0 GHz. Execution time on CPU A? CPU B?

## 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.

## Performance / Clock Cycle Review

---

1. Performance = 1 / Execution Time = 1/ CPU time

2. How do we compute CPU Time?

– CPU Time = CPU Clock Cycles \* Clock Cycle Time

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

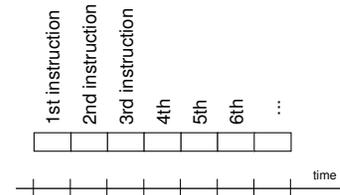
3. How do we get these?

- Clock Cycle Time = time between ticks (seconds per cycle)
  - Usually a given
  - Or compute from Clock Rate
- CPU Clock Cycles = # of cycles per program
  - Where does this come from?

## How many cycles are required for a program?

---

- Could assume that # of cycles = # of instructions



*This assumption is...  
Why?*

## Cycles Per Instruction (CPI)

---

CPU Clock Cycles

- = Total # of clock cycles
- = avg # of clock cycles per instruction \* program instruction count
- = CPI \* IC

What is CPI?

- Average cycle count of all the instruction executed in the program
- CPI provides one way of comparing 2 different implementations of the same ISA, since the instruction count for a program will be the same

New performance equation:

$$\text{Time} = \text{Instruction count} * \text{CPI} * \text{ClockCycleTime}$$

## Now that we understand cycles

---

- A given program will require
  - some number of
  - some number of
  - some number of
- We have a vocabulary that relates these quantities:
  - Instruction count
  - CPU clock cycles (cycles/program)
  - Clock cycle time
  - Clock rate
  - CPI

## 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?

## Performance

---

- Performance is determined by \_\_\_\_\_!
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?
- Common pitfall:

## # of Instructions Example

---

EX: 4-11 ...

- 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?

## Evaluating Performance

---

- Best scenario is head-to-head
  - Two or more machines running the same programs (workload), over an extended time
  - Compare execution time
  - Choose your machine
- Fallback scenario: **BENCHMARKS**
  - Packaged in ‘sets’
  - Programs specifically chosen to measure performance
    - Programs typical of \_\_\_\_\_
  - Composed of real applications
    - Specific to workplace environment
    - Minimizes ability to speed up execution

## Benchmark Games

---

- *An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error...was a sad commentary on a common industry practice of “cheating” on standardized performance tests...The error was pointed out to Intel two days ago by a competitor, Motorola ...came in a test known as SPECint92...Intel acknowledged that it had “optimized” its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing...At the heart of Intel’s problem is the practice of “tuning” compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...*

Saturday, January 6, 1996 New York Times

## Benchmarks

---

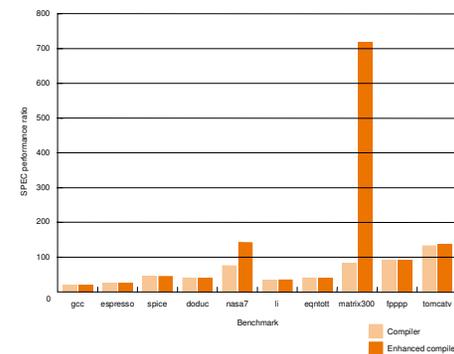
Types of Benchmarks used depend on position of development cycle

- Small benchmarks
  - Nice for architects and designers
  - Very small code segments
  - Easy to standardize
  - Can be abused
- SPEC (System Performance Evaluation Cooperative)
  - <http://www.specbench.org/>
  - Companies have agreed on a set of real program and inputs
  - Valuable indicator of performance (and compiler technology)
  - Latest: SPEC CPU2006

## SPEC “results”

---

- Compiler “enhancements” and performance



## SPEC CPU2006 (Integer)

400.perlbenc	C	Programming Language	Derived from Perl V5.8.7. The workload includes SpamAssassin, MhonArc (an email indexer), and speediff (SPEC's tool that checks benchmark outputs).
401.bzip2	C	Compression	Julian Seward's bzip2 version 1.0.3, modified to do most work in memory, rather than doing IO.
403.gcc	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron.
429.mcf	C	Combinatorial Optimization	Vehicle scheduling. Uses a network simplex algorithm (which is also used in commercial products) to schedule public transport.
445.gobmk	C	Artificial Intelligence: Go	Plays the game of Go, a simply described but deeply complex game.
456.hmmer	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models (profile HMMs)
458.sjeng	C	Artificial Intelligence: chess	A highly-ranked chess program that also plays several chess variants.
462.libquantum	C	Physics / Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
464.h264ref	C	Video Compression	A reference implementation of H.264/AVC, encodes a videostream using 2 parameter sets. The H.264/AVC standard is expected to replace MPEG2.
471.omnetpp	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
473.astar	C++	Path-finding Algorithms	Pathfinding library for 2D maps, including the well known A* algorithm.
483.xalanbmk	C++	XML Processing	A modified version of Xalan-C++, which transforms XML documents to other document types.

## SPEC CPU2006 (Floating point)

410.bwaves	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow.
416.gamess	Fortran	Quantum Chemistry	Gamess implements a wide range of quantum chemical computations. For the SPEC workload, self-consistent field calculations are performed using the Restricted Hartree Fock method, Restricted open-shell Hartree-Fock, and Multi-Configuration Self-Consistent Field.
433.milc	C	Physics / Quantum Chromodynamics	A gauge field generating program for lattice gauge theory programs with dynamical quarks.
434.zeusmp	Fortran	Physics / CFD	ZEUS-MP is a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, University of Illinois at Urbana-Champaign) for the simulation of astrophysical phenomena.
435.gromacs	C, Fortran	Biochemistry / Molecular Dynamics	Molecular dynamics, i.e. simulate Newtonian equations of motion for hundreds to millions of particles. The test case simulates protein Lysozyme in a solution.
436.cactusADM	C, Fortran	Physics / General Relativity	Solves the Einstein evolution equations using a staggered-leapfrog numerical method.
437.jeslie3d	Fortran	Fluid Dynamics	Computational Fluid Dynamics (CFD) using Large-Eddy Simulations with Linear-Eddy Model in 3D. Uses the MacCormack Predictor-Corrector time integration scheme.

(plus 10 more...)

## Amdahl's Law

EX: 4-21 ...

Execution Time After Improvement =

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

- **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?
- *Corollary: Make the common case fast*

## Remember

- Performance is specific to \_\_\_\_\_  
– Only total execution time is a consistent summary of performance
- For a given architecture performance increases come from:
  - Pitfall: expecting improvement in one aspect of a machine's performance to proportionally affect the total performance
  - You should not always believe everything you read! Read carefully!