Chapter 4
Accessing and Understanding Performance

Performance

- Why do we care about performance evaluation?
  - Purchasing perspective
    - given a collection of machines, which has the
      - best performance ?
      - least cost ?
      - best performance / cost ?
  - Design perspective
    - faced with design options, which has the
      - best performance improvement ?
      - least cost ?
      - best performance / cost ?
- How to measure, report, and summarize performance?
  - Performance metric
  - Benchmark

Which of these airplanes has the best performance?

Airplane	Passenger Capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers x m.p.h.)
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 metric defines performance?**
  - **Capacity, cruising range, or speed?**
- **Speed**
  - **Taking one passenger from one point to another in the least time**
  - **Transporting 450 passengers from one point to another**

Two Notions of “Performance”

- 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

Definitions

- Performance is in units of things-per-second
  - bigger is better
- If we are primarily concerned with response time

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_Time}(Y)}{\text{Execution\_Time}(X)}$$

Which one is faster?
Concorde or Boeing 747

- Response Time of Concorde vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 times faster
- Throughput of Concorde vs. Boeing 747 ?
  - Boeing is 286,700 pmph/ 178,200 pmph= 1.6 “times faster”

- Boeing is 1.6 times (“60%”)faster in terms of throughput
- Concord is 2.2 times (“120%”) faster in terms of flying time

Example of Relative Performance

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?
  - (1) PerformanceA/PerformanceB=n
  - (2) Performance ratio: 15/10 = 1.5
  - (3) A is 1.5 times faster than B

We will focus primarily on execution time for a single job !

Metrics for Performance Evaluation

- Program execution time
  - Seconds for a program
  - Elapsed time
    - Total time to complete a task, including disk access, I/O, etc
- CPU execution 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

How about Embedded System?

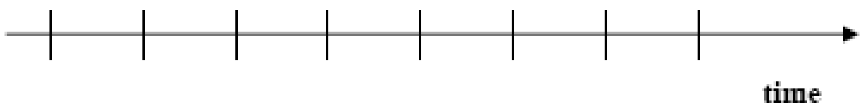
- Embedded system often limited by real-time constraint.
- Hard real time
  - A fixed bound on the time to respond to or process an event
- Soft real time
  - An average response or a response within a limited time to a large fraction of the events suffices.
- Designers often optimize throughput or try to reduce cost under certain response-time performance constraint.

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

$$\frac{1}{200 \times 10^{6}} \times 10^{9} = 5 \text{ nanoseconds}$$ 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.

Example of Improving Performance

- A program runs 10 second on 4GHz clock computer A. We are trying to help a computer designer build a computer, B, that will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

(1)CPU execute timeA

= CPU clock cyclesA/ Clock rateA

= 10 sec = CPU clock cyclesA/ 4GHz

CPU clock cyclesA=4GHz x 10sec =40G cycles

(2)CPU execute timeB

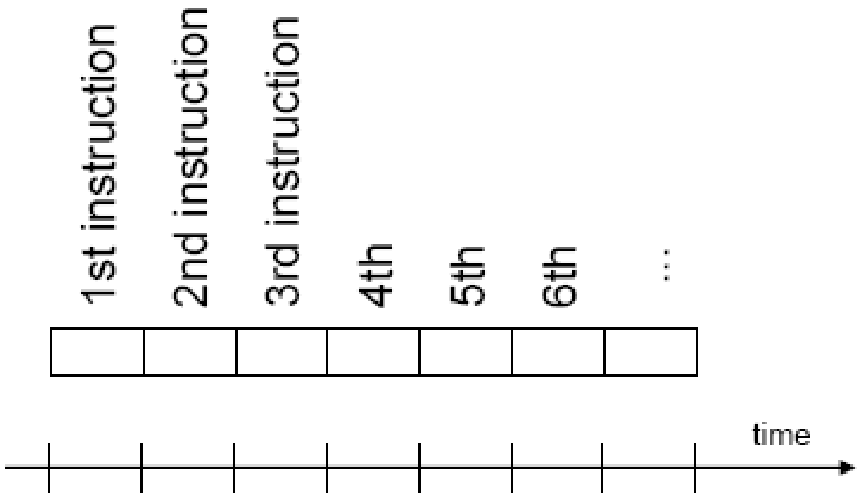
= 1.2 x CPU clock cyclesA/ Clock rateB

= 6 sec =1.2x40G cycles / Clock rateB

CPU rateB= 1.2x40G cycles / 6sec =8GHz

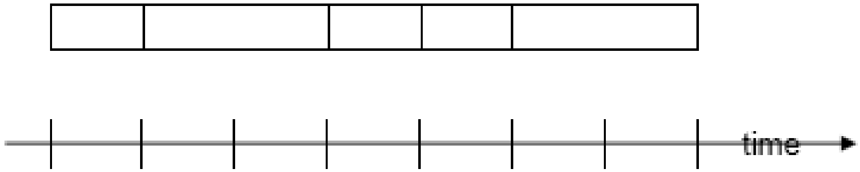
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



This assumption is incorrect,different instructions take different amounts of time on different machines.

Different numbers of cycles for different instructions



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- **CPI**(cycles per instruction)

CPU clock cycles = Instruction for a program x Average CPI

Performance Equation

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Performance is determined by execution time
- 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?
    - MIPS (million instructions per second)
    - When is it fair to compare two processors using MIPS?
- Common pitfall: thinking one of the variables is indicative of performance when it really isn't.

CPU time for a program

- **CUP execution time for a program**

  = CPU clock cycles for the program * Clock cycle time

  = CPU clock cycles for the program / Clock rate

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***

CPI

- **CPI: cycles per instruction**

$$CPU\ time = (\#\ \text{of inst.}) * (\text{CPI}) * (\text{cycle time})$$

$$CPU\ time = \frac{(\#\ \text{of inst.}) * \text{CPI}}{\text{clock rate}}$$

$$CPU\ time = \frac{\text{seconds}}{\text{program}} = \frac{\#\ \text{of inst.}}{\text{program}} \times \frac{\#\ \text{of cycle}}{\text{inst.}} \times \frac{\text{seconds}}{\text{cycle}}$$

Performance

- **Performance is determined by execution time**
- **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: thinking one of the variables is indicative of performance when it really isn't.**

Example #1

- Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for a program, and machine B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which machine is faster for this program, and by how much?

(1) CPU clock cyclesA=I x 2.0
CPU clock cyclesB=I x 1.2
I is the number of instructions for this program

(2) CPU timeA=I x 2.0 x 250 ps= 500 x I ps
CPU timeB=I x 1.2 x 500 ns = 600 x I ps

(3) CPU performanceA= 1 / timeA
CPU performanceB= 1 / timeB

(4) Speedup= performanceA/ performanceB=1.2

Example #2 MIPS Performance Measure

CPI : A = 1, B =2 , C = 3
Clock Rate 4 GHz

Code from	Instruction counts (in billions) for each instruction class		
	A	B	C
Compiler 1	5	1	2
Compiler 2	10	1	1

- Which code sequence will execute faster according to MIPS?
- Which code sequence will execute faster according to execution time?

$$CPU\ clock\ cycles = \sum_{i=1}^{n} CPI_i \times C_i$$

MIPS = Instruction Count / (Execution time x 10^6)

$$MIPS = \frac{\text{Instruction Count}}{\text{Execution time} \times 10^6}$$

Example of MIPS Performance Measure (cont.)

(1) CPU clock $cycles_1$=$(5x1+1x2+1x3)x10^9$=$10x10^9$

CPU clock $cycles_2$=$(10x1+1x2+1x3)x10^9$=$15x10^9$

(2) Execution $time_1$=$10x10^9/4x10^9$=2.5sec

Execution $time_2$=$15x10^9/4x10^9$=3.75sec

(3) $MIPS_1$=$(5+1+1)x10^9/2.5x10^6$=2800

$MIPS_2$=$(10+1+1)x10^9/3.75x10^6$=3200

So, the code from compiler 2 has a higher MIPS rating, but the code from compiler 1 runs faster!

Example #4

- 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?*

Example #4

- 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?

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\frac{\text{Execution\_Time (A)}}{\text{Execution\_Time (B)}} = \frac{\text{I x } 2.0 \text{ x } 10}{\text{I x } 1.2 \text{ x } 20}$$

Example #5

Instruction class	CPI for this instruction class
A	1
B	2
C	3

Code sequence	Instruction counts for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

- Which code sequence executes the most instructions?
- Which will be faster?
- What is the CPI for each sequence?

Example #5

- (1) Seq1 = 2 + 1 + 2 = 5

  Seq2 = 4 + 1 + 1 = 6

- (2) CPU clock cycles1=(2x1)+(1x2)+(2x3)=10

  CPU clock cycles2=(4x1)+(1x2)+(1x3)=9

- (3) CPI1= CPU clock cycles1/Instruction count1 = 10/5 = 2

  CPI2= CPU clock cycles2/Instruction count2 = 9/6 = 1.5

When comparing 2 machines, these "3 components" must be considered!

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Algorithm	X	X	
Programming Language	X	X	
Compiler	X	X	
ISA (instruction set architecture)	X	X	X

Basis for Evaluation: Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
    e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused
  - valuable indicator of performance (and compiler technology)
  - latest: spec2000

SPEC CPU 2000

Integer benchmarks		FP benchmarks	
Name	**Description**	**Name**	**Type**
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

SPEC CINT2000

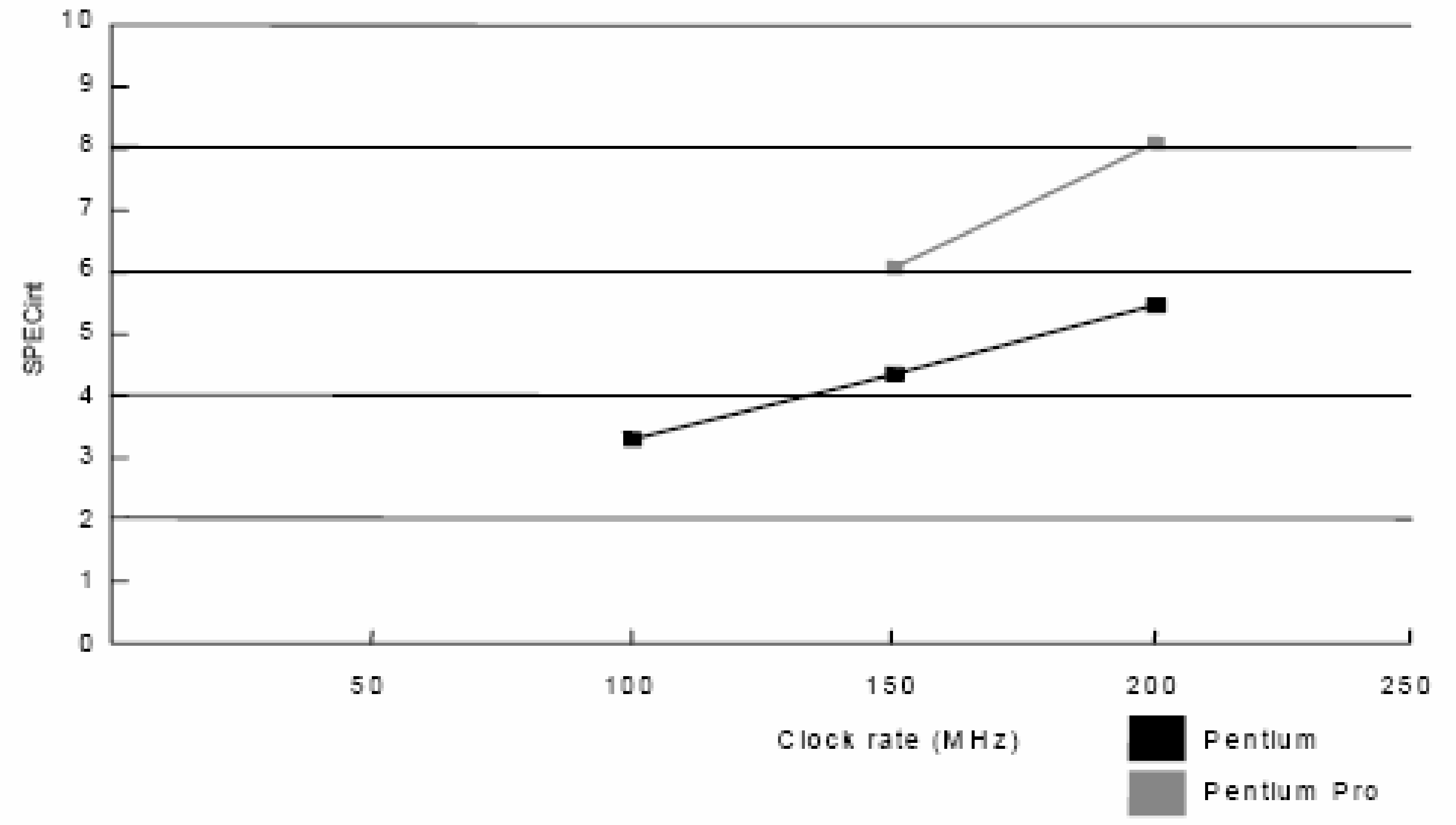
SPEC CINT2000 Benchmark Kernels				
Benchmark	Reference Time	Language	Application Class	General Description
164.gzip	1400	C	Compression	Compresses a TIFF (Tagged Image Format File), a Web server log, binary program code, "random" data, and a tar file source.
175.vpr	1400	C	Field Programmable Gate Array Circuit Placement and Routing	Maps FPGA circuit logic blocks and their required connections using a combinatorial optimization program. Such programs are found in integrated circuit CAD programs.
176.gcc	1100	C	C Programming Language Compiler	Compiles Motorola 88100 machine code from five different input source files using gcc.
181.mcf	1800	C	Combinatorial Optimization	Solves a single-depot vehicle scheduling problem of the type often found in the public transportation planning field.
186.crafty	1000	C	Chess	Solves five different chessboard input layouts to varying search tree "depths" for possible next moves.
197.parser	1800	C	Word Processing	Parses input sentences to find English syntax using a 60,000-word dictionary.
252.eon	1300	C++	Computer Visualization	Finds the intersection of three-dimensional rays using probabilistic ray tracing.
253.perlbmk	1800	C	PERL Programming Language	Processes five Perl scripts to create mail, HTML, and other output.
254.gap	1100	C	Group Theory Interpreter	Interprets a group theory language that was written to process combinatorial problems.
255.vortex	1900	C	Object-Oriented Database	Manipulates data from three object-oriented databases.
256.bzip2	1500	C	Compression	Compresses a TIFF, a binary program, and a tar source file.
300.twolf	3000	C	Place and Route Simulator	Approximates a solution to the problem of finding an optimal transistor layout on a microchip.

SPEC CFP2000

SPEC CFP2000 Benchmark Kernels				
Benchmark	Reference Time	Language	Application Class	General Description
168.wupwise	1600	FORTRAN 77	Quantum Chromodynamics	Simulates quark interactions as needed by physicists studying quantum chromodynamics.
171.swim	3100	FORTRAN 77	Shallow Water Modeling	Predicts weather using mathematical modeling techniques. Swim is often used as a benchmark of supercomputer performance.
172.mgrid	1800	FORTRAN 77	3D Potential Field Solver	Computes the solution of a three-dimensional scalar Poisson equation. This kernel benchmark comes from NASA.
173.applu	2100	FORTRAN 77	Parabolic-Elliptic Partial Differential Equations	Solves five nonlinear partial differential equations using sparse Jacobian matrices.
177.mesa	1400	C	3-D Graphics Library	Converts a two-dimensional graphics input to a three-dimensional graphics output.
178.galgel	2900	FORTRAN 90	Computational Fluid Dynamics	Determines the critical value of temperature differences in the walls of a fluid tank that cause convective flow to change to oscillatory flow.
179.art	2600	C	Image Recognition	Locates images of a helicopter and an airplane within an image. The algorithm uses neural networks.
183.equake	1300	C	Seismic Wave Propagation Simulation	Uses finite element analysis to recover the history of ground motion ensuing from a seismic event.
187.facerec	1900	FORTRAN 90	Face Recognition	Uses the "Elastic Graph Matching" method to recognize faces represented by labeled graphs.
188.ammp	2200	C	Computational Chemistry	Solves a molecular dynamics problem by calculating the motions of molecules within a system.
189.lucas	2000	FORTRAN 90	Primality Testing	Begins the process of determining the primality of a large Mersenne number (2^{P-1}). The result is not found; the intermediate results are measured instead.
191.fma3d	2100	FORTRAN 90	Finite-Element Crash Simulation	Simulates the effects of the collision of inelastic three-dimensional solids.
200.sixtrack	1100	FORTRAN 77	High Energy Nuclear Physics Accelerator Design	Simulates tracking particle behavior through a particle accelerator.
301.apsi	2600	FORTRAN 77	Pollutant Distribution	Finds the velocity of pollutant particles from a given source using parameters of initial velocity, wind speed, and temperature.

Now, you can answer this question..

- Q2: CPU frequency？ Performance



Performance Improvement

$$Time = \frac{Seconds}{pogram} = \underbrace{\frac{Instructions}{program}}_{\text{Instruction count for a program}} \times \underbrace{\frac{Clock\ cycles}{Instruction}}_{\text{CPI: clock per instruction}} \times \underbrace{\frac{Seconds}{Clock\ cycles}}_{\text{Clock rate}}$$

- A shown in the formula, given an ISA, increases in CPU performance can come from three source:
  1. Increases in clock rate
  2. Improvements in processor organization that lower the CPI
  3. Compiler enhancements that lower the instruction count or generate instructions with a lower average CPI (e.g., by using simpler instructions)

Amdahl's Law

- Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$

- Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected, then:

$$\text{ExTime(E)} = ((1\text{-}F) + F/S) \times \text{ExTime(without E)}$$

$$\text{Speedup(E)} = \frac{1}{(1\text{-}F) + F/S}$$

Amdahl's Law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times (0.9 + .1/2) = 0.95 \times \text{ExTime}_{\text{old}}$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{0.95} = 1.053$$

Example #3

- Our favorite program runs in 10 seconds on computer A, which has a 400 Mhz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?"

$$\frac{\text{Execution\_Time (A)}}{\text{Execution\_Time (B)}} = \frac{10}{6} = \frac{C \;\; x \;\; \dfrac{1}{400\ x10\text{^}\ 6}}{1.2C \;\; x \;\; \dfrac{1}{x}}$$

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