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?

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passenger Capacity</th>
<th>Cruising range (miles)</th>
<th>Cruising speed (m.p.h.)</th>
<th>Passenger throughput (passengers x m.p.h.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 777</td>
<td>375</td>
<td>4630</td>
<td>610</td>
<td>228,750</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
<td>286,700</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
<td>178,200</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
<td>79,424</td>
</tr>
</tbody>
</table>

- 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

\[
\frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_Time}(Y)}{\text{Execution\_Time}(X)}
\]
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 ppmh / 178,200 ppmh = 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) Performance\textsubscript{A}/Performance\textsubscript{B} = 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

\[
\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds } \text{ cycle time}
\]
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 time A
   = CPU clock cycles A / Clock rate A
   = 10 sec = CPU clock cycles A / 4GHz
   CPU clock cycles A = 4GHz x 10sec = 40G cycles

2. CPU execute time B
   = 1.2 x CPU clock cycles A / Clock rate B
   = 6 sec = 1.2 x 40G cycles / Clock rate B
   CPU rate B = 1.2 x 40G 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)

\[
\text{CPU clock cycles} = \text{Instruction for a program} \times \text{Average CPI}
\]
Performance Equation

<table>
<thead>
<tr>
<th>CPU time</th>
<th>=</th>
<th>Seconds</th>
<th>= Instructions</th>
<th>x</th>
<th>Cycles</th>
<th>x</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td>Program</td>
<td>Instruction</td>
<td></td>
<td>Cycle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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
  - $= \text{CPU clock cycles for the program} \times \text{Clock cycle time}$
  - $= \frac{\text{CPU clock cycles for the program}}{\text{Clock rate}}$
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

\[ CPU \text{ time} = (\# \text{ of inst.}) \times \text{(CPI)} \times \text{(cycle time)} \]

\[ CPU \text{ time} = \frac{(\# \text{ of inst.}) \times \text{CPI}}{\text{clock rate}} \]

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

<table>
<thead>
<tr>
<th>Code from</th>
<th>Instruction counts (in billions) for each instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Compiler 1</td>
<td>5</td>
</tr>
<tr>
<td>Compiler 2</td>
<td>10</td>
</tr>
</tbody>
</table>

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

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

\[
CPU \text{ clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i
\]

\[
MIPS = \frac{\text{Instruction Count}}{\text{Execution time} \times 10^6}
\]
Example of MIPS Performance Measure (cont.)

(1) CPU clock cycles\(_1\) = \((5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9\) = \(10 \times 10^9\)
CPU clock cycles\(_2\) = \((10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9\) = \(15 \times 10^9\)

(2) Execution time\(_1\) = \(10 \times 10^9 / 4 \times 10^9\) = 2.5 sec
Execution time\(_2\) = \(15 \times 10^9 / 4 \times 10^9\) = 3.75 sec

(3) MIPS\(_1\) = \((5 + 1 + 1) \times 10^9 / 2.5 \times 10^6\) = 2800
MIPS\(_2\) = \((10 + 1 + 1) \times 10^9 / 3.75 \times 10^6\) = 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?

<table>
<thead>
<tr>
<th>CPU time</th>
<th>= Seconds</th>
<th>= Instructions x Cycles x Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Program</td>
<td>Instruction</td>
</tr>
</tbody>
</table>

\[
\text{Execution Time (A)} = \frac{I \times 2.0 \times 10}{I \times 1.2 \times 20}
\]
Example #5

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>CPI for this instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code sequence</th>
<th>Instruction counts for instruction class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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

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

<table>
<thead>
<tr>
<th></th>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Programming Language</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(instruction set architecture)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

**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.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Reference Time</th>
<th>Language</th>
<th>Application Class</th>
<th>General Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>1400</td>
<td>C</td>
<td>Compression</td>
<td>Compresses a TIFF (Tagged Image Format File), a Web server log, binary program code,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em>random</em> data, and a tar file source.</td>
</tr>
<tr>
<td>175.vpr</td>
<td>1400</td>
<td>C</td>
<td>Field Programmable Gate Array Circuit</td>
<td>Maps FPGA circuit logic blocks and their required connections using a combinatorial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Placement and Routing</td>
<td>optimization program. Such programs are found in integrated circuit CAD programs.</td>
</tr>
<tr>
<td>176.gcc</td>
<td>1100</td>
<td>C</td>
<td>C Programming Language Compiler</td>
<td>Compiles Motorola 88100 machine code from five different input source files using gcc.</td>
</tr>
<tr>
<td>181.mcf</td>
<td>1800</td>
<td>C</td>
<td>Combinatorial Optimization</td>
<td>Solves a single-depot vehicle scheduling problem of the type often found in the public</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>transportation planning field.</td>
</tr>
<tr>
<td>186.crafty</td>
<td>1000</td>
<td>C</td>
<td>Chess</td>
<td>Solves five different chessboard input layouts to varying search tree &quot;depths&quot; for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>possible next moves.</td>
</tr>
<tr>
<td>197.parser</td>
<td>1800</td>
<td>C</td>
<td>Word Processing</td>
<td>Parses input sentences to find English syntax using a 60,000-word dictionary.</td>
</tr>
<tr>
<td>252.eon</td>
<td>1300</td>
<td>C++</td>
<td>Computer Visualization</td>
<td>Finds the intersection of three-dimensional rays using probabilistic ray tracing.</td>
</tr>
<tr>
<td>253.perlbnk</td>
<td>1800</td>
<td>C</td>
<td>PERL Programming Language</td>
<td>Processes five Perl scripts to create mail, HTML, and other output.</td>
</tr>
<tr>
<td>254.gap</td>
<td>1100</td>
<td>C</td>
<td>Group Theory Interpreter</td>
<td>Interprets a group theory language that was written to process combinatorial problems.</td>
</tr>
<tr>
<td>255.vortex</td>
<td>1900</td>
<td>C</td>
<td>Object-Oriented Database</td>
<td>Manipulates data from three object-oriented databases.</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>1500</td>
<td>C</td>
<td>Compression</td>
<td>Compresses a TIFF, a binary program, and a tar source file.</td>
</tr>
<tr>
<td>300.twolf</td>
<td>3000</td>
<td>C</td>
<td>Place and Route Simulator</td>
<td>Approximates a solution to the problem of finding an optimal transistor layout on a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>microchip.</td>
</tr>
</tbody>
</table>
# SPEC CFP2000

## SPEC CFP2000 Benchmark Kernels

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

- Q2: CPU frequency? Performance
A shown in the formula, given an ISA, increases in CPU performance can come from three sources:

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:

  \[
  \text{ExTime}(E) = ((1-F) + F/S) \times \text{ExTime(without } E) \\
  \text{Speedup}(E) = \frac{1}{(1-F) + F/S}
  \]
Amdahl’s Law

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

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times (0.9 + \frac{1}{2}) = 0.95 \times \text{ExTime}_{\text{old}}
\]

\[
\text{Speedup}_{\text{overall}} = \frac{1}{0.95} = 1.053
\]
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?

\[
\frac{\text{Execution Time (A)}}{\text{Execution Time (B)}} = \frac{10}{6} = \frac{C \times \frac{1}{400 \times 10^6}}{1.2C \times \frac{1}{x}}
\]

\[
\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}
\]