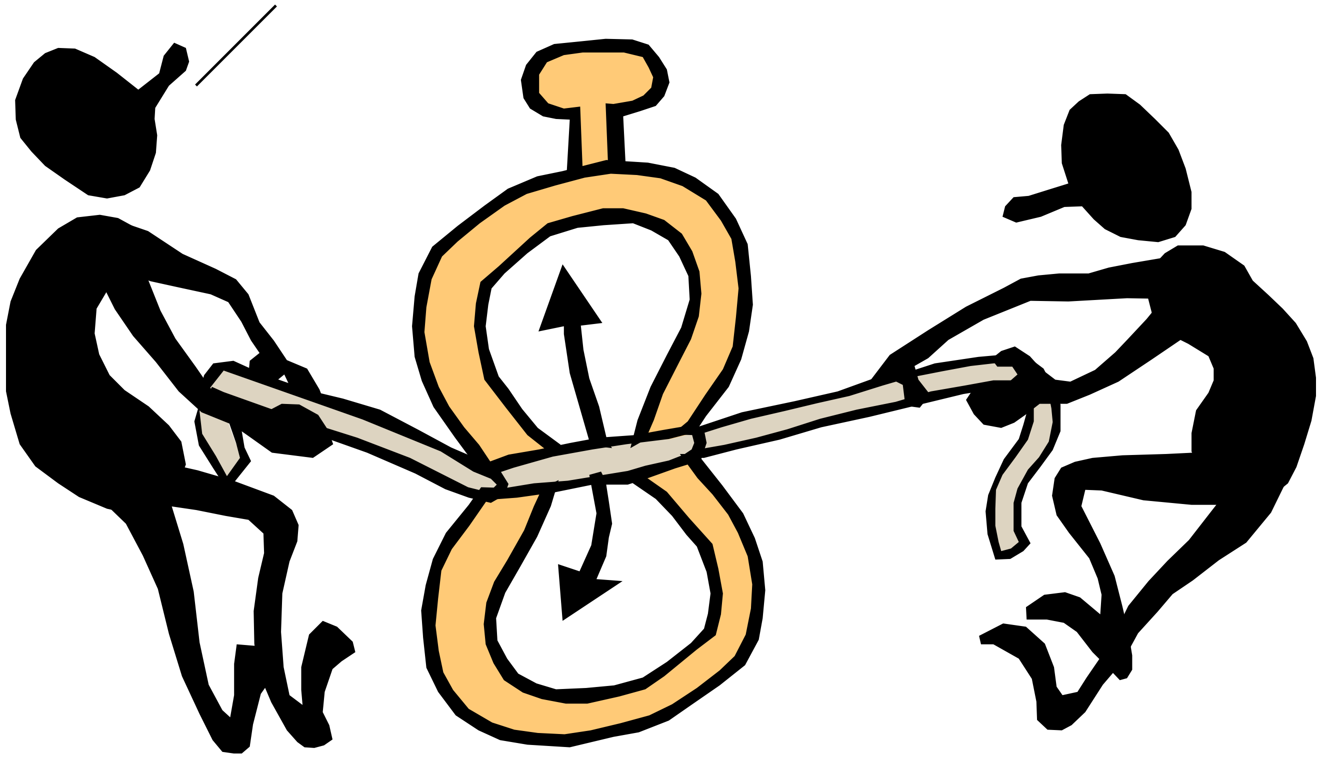
Computer Performance

He said, to speed
things up we need
to squeeze the clock



Why Study Performance?

Helps us to make intelligent design choices

Helps us see through marketing hype

Affects computer organization
(pipelining, caches, etc.)

- *Why is some hardware faster than others for different programs?*
- *What factors of system performance are hardware related? (e.g., Do we need a new machine, more memory, a better compiler, or a new OS?)*
- *How does a machine's instruction set affect its performance?*

Which Airplane has the Best Performance?

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

How much faster is the Concorde than the 747? **2.213 X**

How much larger is the 747's capacity than the Concorde? **4.65 X**

It is roughly 4000 miles from Raleigh to Paris. What is the throughput of the 747 in passengers/hr? The Concorde?

$$470 \times \frac{610}{4000} = 71.65 \text{ passengers/hr} \qquad 101 \times \frac{1350}{4000} = 34.0875 \ \frac{\text{pass}}{\text{hr}}$$

What is the latency of the 747? The Concorde? **4000/610 = 6.56 hours/pass**
4000/1350 = 2.96 hours/pass

Performance Metrics

Latency: Time from an input to its corresponding output

- How long does it take for my program to run?
- How long must I wait after typing return for the result?

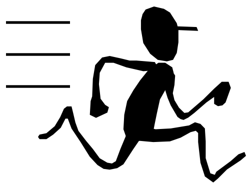
Throughput: The rate at which new outputs are generated

- How many calculations per second?
- What is the average execution rate of my program?
- How much work is getting done?

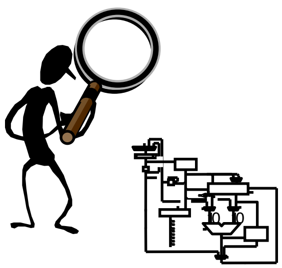
By running a program on 20 different input files on the fastest processor, what performance metric do we improve? Latency

By running our program simultaneously on 20 CPU's for all of the assigned input files, what performance metric do we improve? Throughput

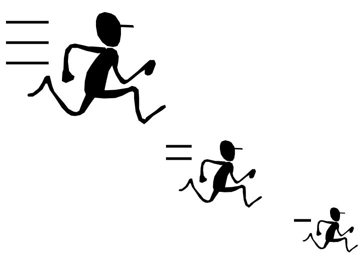
Design Tradeoffs



Maximum Performance: measured by the numbers of instructions executed per second



Minimum Cost: determined by the size-of-the-circuit/number-of-components-used plus power/cooling costs



Best Price/Performance: measured by the ratio of CPU-cost to MIPS.

For many applications **Performance/Watt** has become important too.

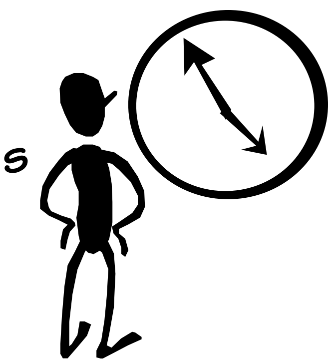
Execution Time

Elapsed Time/Wall Clock Time

counts everything *(disk and memory accesses, I/O , etc.)*

a useful number, but often not good for comparison purposes

CPU time



Doesn't include 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 actual instructions of "our" program

Definition of Performance

For some program running on machine X,

$Performance_X$ = Program Executions / $Time_X$ (executions/sec)

"X is n times faster than Y"

$Performance_X$ / $Performance_Y$ = n

Problem:

Machine A runs a program in 20 seconds

Machine B runs the same program in 25 seconds

$Performance_A$ = 1/20 $Performance_B$ = 1/25

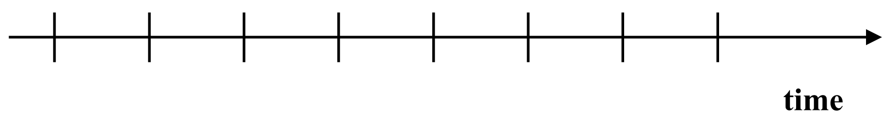
Machine A is (1/20)/(1/25) = 1.25 times faster than Machine B

Program Clock Cycles

Instead of reporting execution time in seconds, we can also use cycle counts

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

Clock "ticks" are when machine-state changes (synchronous abstraction):



cycle time = time between rising edges of the clock = seconds per cycle

clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)

A 200 Mhz. clock has a $\frac{1}{200 \times 10^{6}} = 5.0 \times 10^{9} = 5nS$ cycle time

OVERCLOCKing improves performance (seconds/program) by decreasing the cycle time (seconds/cycle), while hoping that the functional blocks continue to operate as specified.

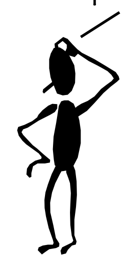
Computer Performance Measure

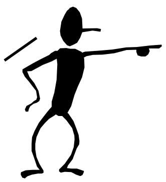
Millions of Instructions per Second

Frequency in Hz

Which of these terms are ISA/Architecture dependent?

$$\text{MIPS} = \frac{1}{10^6} \frac{clocks/second}{\text{AVE}(clocks/instruction)}$$





This term is canceled if the frequency is given in Megahertz (MHz)

CPI (Average Clocks Per Instruction)

Historically:

70's -80's	PDP-11, VAX, Intel 8086	CPI > 1
90's	Load/Store RISC machines	
	MIPS, SPARC, PowerPC, miniMIPS:	CPI = 1
Your Century	Modern CPUs, Pentium, Athlon, i7	CPI < 1

How to Improve Performance?

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

$$MIPS = \frac{1}{10^6}\frac{Freq}{CPI}$$

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

Decrease the # of required cycles for a program, or (improve ISA/Compiler)

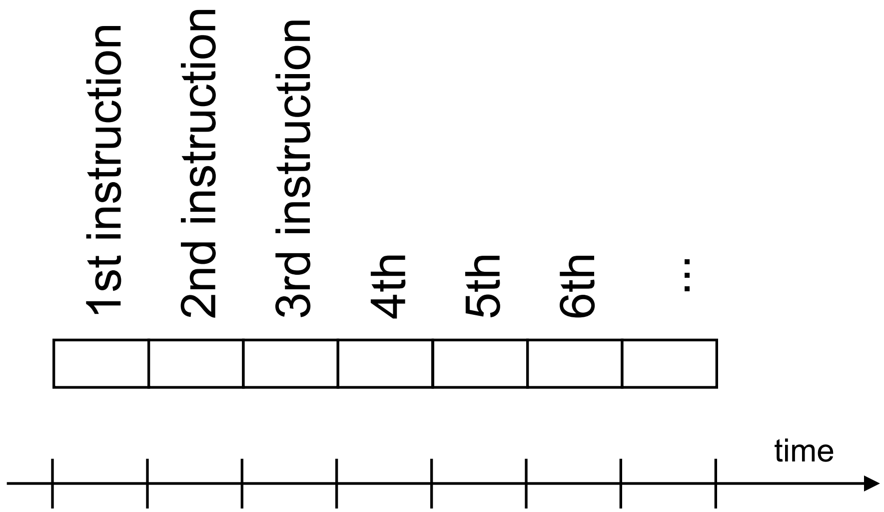
Decrease the clock cycle time or, said another way,

Increase the clock rate. (reduce propagation delays or use pipelining)

Decrease the CPI (average clocks per instruction) (new H/W)

How Many Cycles in a Program?

Could assume that # of cycles = # of instructions (True of the miniMIPS implementation developed last lecture).



This assumption can be incorrect,

- *Different instructions take different amounts of time on different machines.*
- *Memory accesses might require more cycles than other instructions.*
- *Floating-Point instructions might require multiple clock cycles to execute.*
- *Branches might stall execution rate*

Example

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, to 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?

$$\frac{cycles}{program} = \left(\frac{seconds}{program}\right)_A \times \frac{cycles}{second} = 10 \times 400 \times 10^6 = 4 \times 10^9$$

$$\frac{cycles}{second} = \frac{cycles/program}{(seconds/program)_B} = \frac{1.2 \times 4 \times 10^9}{6} = 800 \times 10^6$$

Don't panic, can easily work this out from basic principles

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 (average clocks 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

Performance Traps

Actual performance is determined by the execution time of a program that you care about, not a benchmark nor a clock rate.

Variables that impact 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 only one of these variables is indicative of performance when it really isn't.

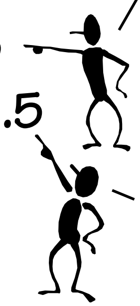
CPI Example

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

For some program,

Machine A has a clock cycle time of 1 ns and a CPI of 0.5

Machine B has a clock cycle time of 0.4 ns and a CPI of 1.5



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

$$\text{MIPS}_A = \frac{1}{10^6}\frac{freq}{CPI} = \frac{1}{10^6}\frac{1/(1\times10^{-9})}{0.5} = 2000 \qquad \text{MIPS}_B = \frac{1}{10^6}\frac{freq}{CPI} = \frac{1}{10^6}\frac{1/(0.4\times10^{-9})}{1.5} = 1666.66$$

$$Relative\ Performance = \frac{\text{MIPS}_\text{A}}{\text{MIPS}_B} = \frac{2000}{1666.6} = 1.2$$

If two machines have the same ISA and run the same program, which quantity (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?

Compiler's Performance Impact

Two different compilers are being tested for a 500 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 the same a large program. The first compiler's code executes 5 million Class A instructions, 1 million Class B instructions, and 2 million Class C instructions. The second compiler's code executes 7 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

Which program uses the fewest instructions?

$$Instructions_1 = (5+1+2) \times 10^6 = 8 \times 10^6$$

$$Instructions_2 = (7+1+1) \times 10^6 = 9 \times 10^6$$

Which sequence uses the fewest clock cycles?

$$Cycles_1 = (5(1)+1(2)+2(3)) \times 10^6 = 13 \times 10^6$$

$$Cycles_2 = (7(1)+1(2)+1(3)) \times 10^6 = 12 \times 10^6$$

Benchmarks

- Performance is 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
  - but can be easily abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real programs and inputs
  - can still be abused
  - valuable indicator of performance (and compiler technology)

SPEC CPU 2006

CINT2006 (Integer Component of SPEC CPU2006):

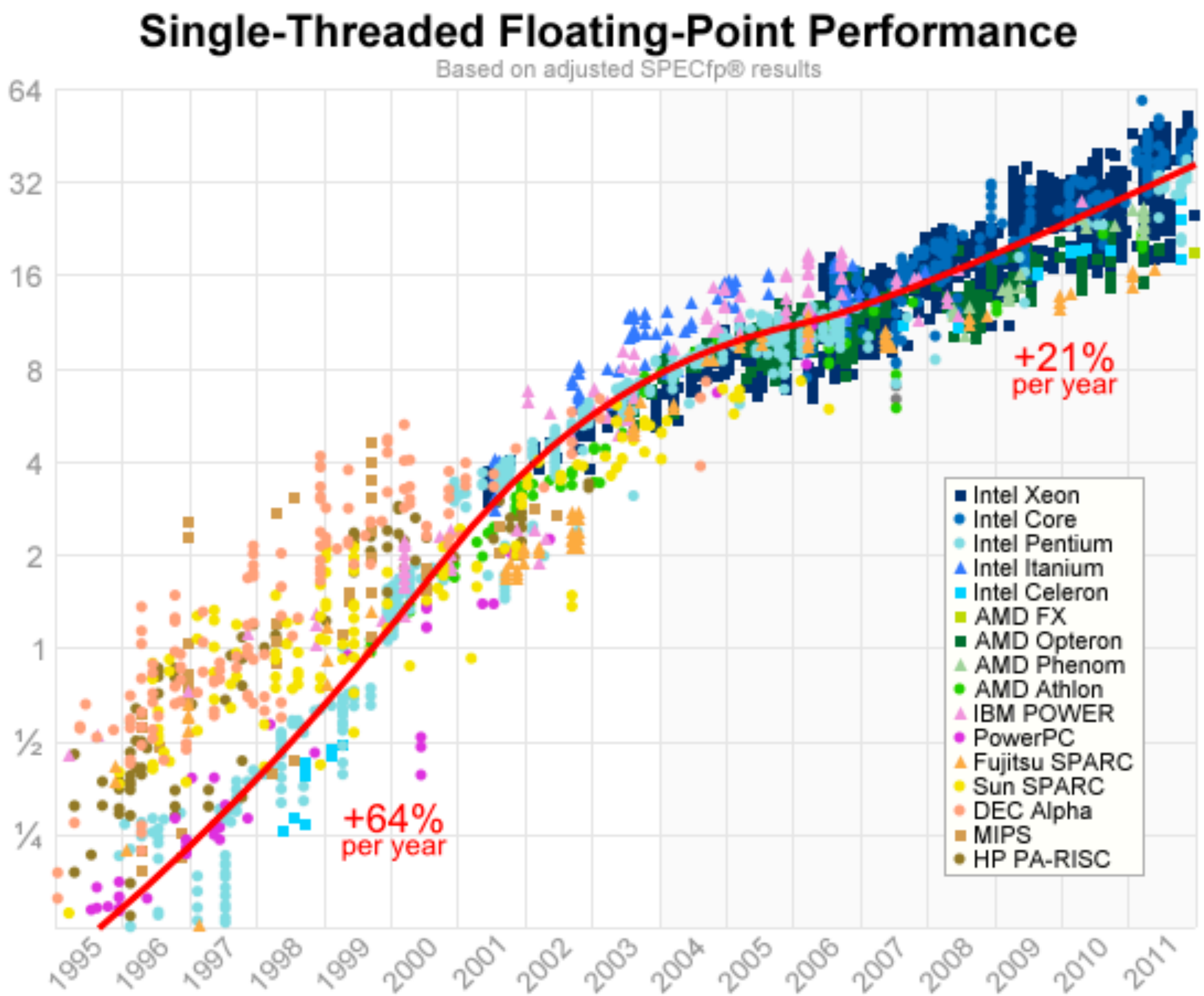
Benchmark	Language	Application Area	Brief Description
400.perlbench	C	Programming Language	Derived from Perl V5.8.7. The workload includes SpamAssassin, MHonArc (an email indexer), and specdiff (SPEC's tool that checks benchmark outputs).
401.bzip2	C	Compression	Julian Seward's bzip2 version 1.0.3, modified to do most work in memory, rather than doing I/O.
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.xalancbmk	C++	XML Processing	A modified version of Xalan-C++, which transforms XML documents to other document types.

SPEC CPU 2006

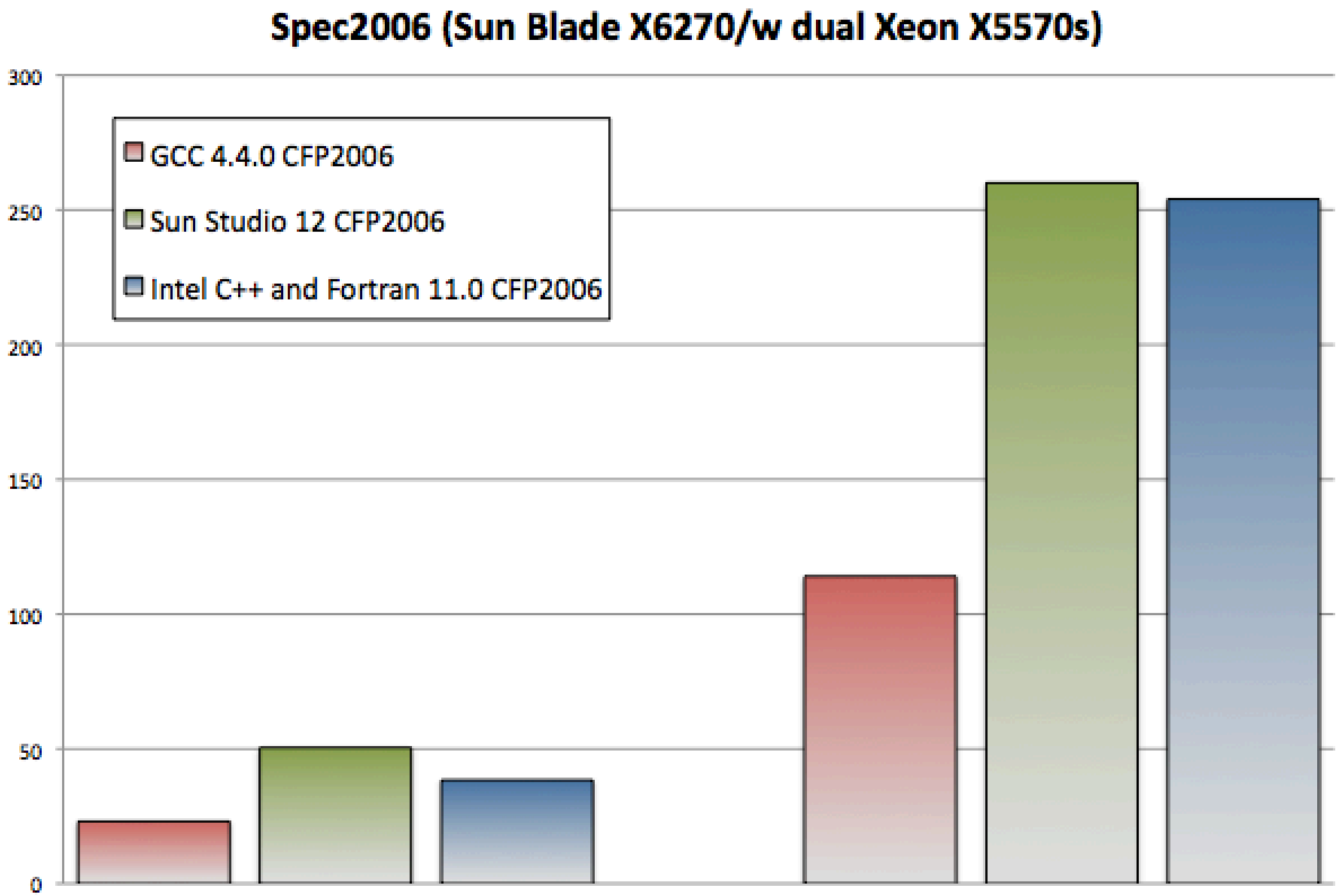
CFP2006 (Floating Point Component of SPEC CPU2006):

Benchmark	Language	Application Area	Brief Description
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.leslie3d	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.
444.namd	C++	Biology / Molecular Dynamics	Simulates large biomolecular systems. The test case has 92,224 atoms of apolipoprotein A-I.
447.dealII	C++	Finite Element Analysis	deal.II is a C++ program library targeted at adaptive finite elements and error estimation. The testcase solves a Helmholtz-type equation with non-constant coefficients.
450.soplex	C++	Linear Programming, Optimization	Solves a linear program using a simplex algorithm and sparse linear algebra. Test cases include railroad planning and military airlift models.
453.povray	C++	Image Ray-tracing	Image rendering. The testcase is a 1280x1024 anti-aliased image of a landscape with some abstract objects with textures using a Perlin noise function.
454.calculix	C, Fortran	Structural Mechanics	Finite element code for linear and nonlinear 3D structural applications. Uses the SPOOLES solver library.
459.GemsFDTD	Fortran	Computational Electromagnetics	Solves the Maxwell equations in 3D using the finite-difference time-domain (FDTD) method.
465.tonto	Fortran	Quantum Chemistry	An open source quantum chemistry package, using an object-oriented design in Fortran 95. The test case places a constraint on a molecular Hartree-Fock wavefunction calculation to better match experimental X-ray diffraction data.
470.lbm	C	Fluid Dynamics	Implements the "Lattice-Boltzmann Method" to simulate incompressible fluids in 3D
481.wrf	C, Fortran	Weather	Weather modeling from scales of meters to thousands of kilometers. The test case is from a 30km area over 2 days.
482.sphinx3	C	Speech recognition	A widely-known speech recognition system from Carnegie Mellon University

Stories Benchmarks Tell



It's not just CPUs



Amdahl's Law

(a.k.a where to spend your efforts when improving performance)

$$t_{improved} = \frac{t_{affected}}{r_{speedup}} + t_{unaffectted}$$

Example:

"Suppose a program runs in 100 seconds on a machine, where multiplies are executed 80% of the time. How much do we need to improve the speed of multiplication if we want the program to run 4 times faster?"

25 = 80/r + 20 r = 16x

How about making it 5 times faster?

20 = 80/r + 20 r = ?

Principle: Focus on making the most common case fast.

Amdahl's Law applies to H/W and S/W!

Example

Suppose we enhance a machine by making all floating-point instructions run 5 times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if only ½ of the 10 seconds is spent executing floating-point instructions?

$6 = 5/5 + 5$ Relative Perf $= 10/6 = 1.67$ x

Marketing is looking for a benchmark to show off the new floating -point unit described above, and wants the overall benchmark to show at least a speedup of 3. What percentage of the execution time would floating-point instructions have to be to account in order to yield our desired speedup on this benchmark?

$33.33 = f/5 + (100 - f) = 100 - 4f/5$ $f = 83.33$

Remember

- When performance is specific to a particular program
  - Total execution time is a consistent summary of performance
- For a given architecture performance comes from:

  1) increases in clock rate (without adverse CPI affects)

  2) improvements in processor organization that lower CPI

  3) compiler enhancements that lower CPI and/or instruction count
- Pitfall: Expecting improvements in one aspect of a machine's performance to affect the total performance
- You should not always believe everything you read! So read carefully!