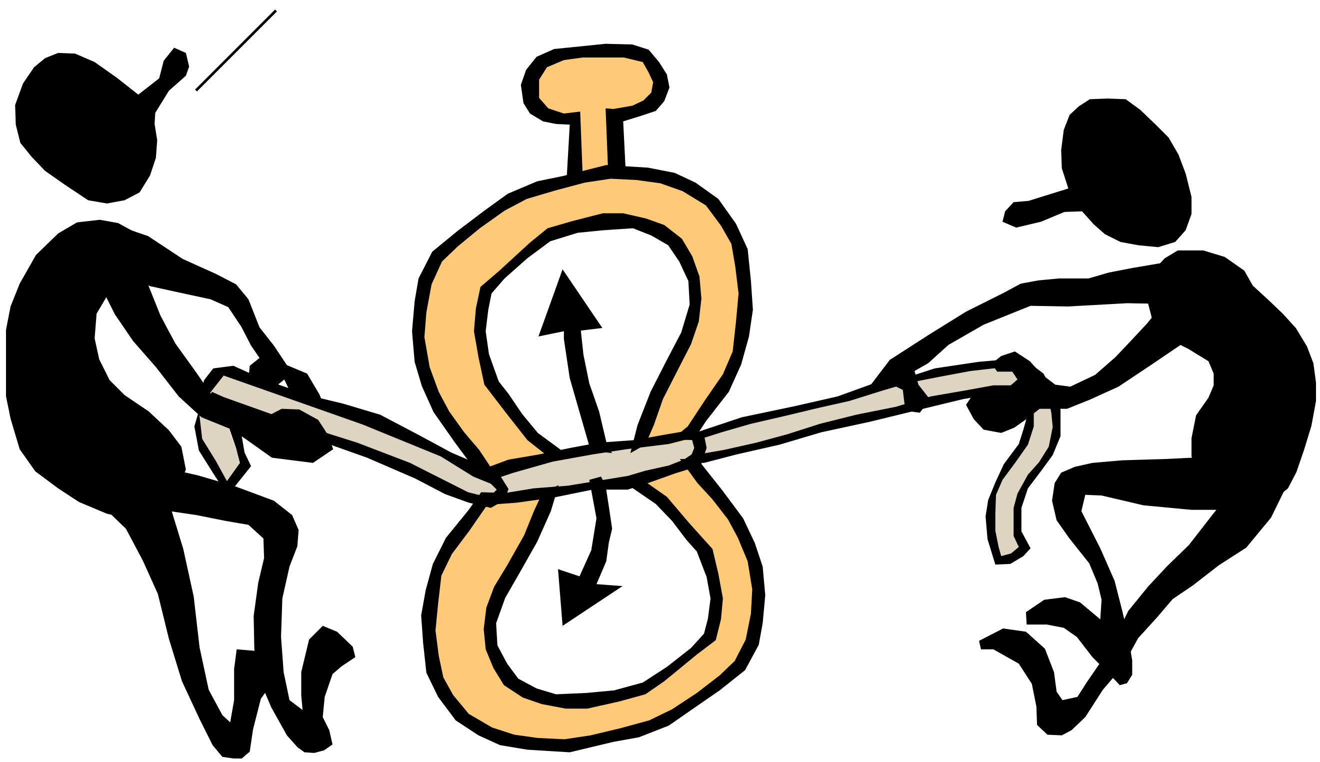
Computer Performance



- Reread Chapter 1.4-1.9

Why Study Performance?

Helps us to make intelligent design choices

Helps us see through marketing hype

Effects 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, or a new operating system?)*
- *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? 6.56 hours, 2.96 hours

Performance Metrics

Latency: Clocks from input to corresponding output

— How long does it take for my program to run?

— How long must I wait after typing return for the result?

Throughput: How many results per clock

— How many outputs can be generated per second?

— What is the average execution rate of my program?

— How much work is getting done?

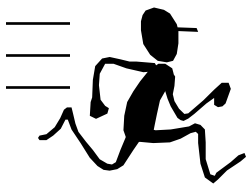
If we upgrade a machine with a new faster processor what do we improve?

Latency

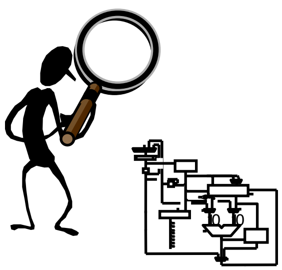
If we add a new machine to the lab what do we increase?

Throughput

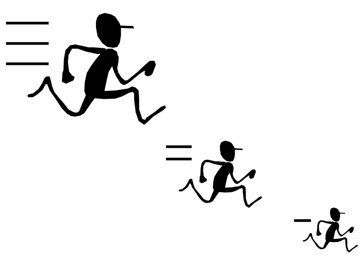
Design Tradeoffs



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



Minimum Cost: determined by either the size of the circuit or power/cooling costs



Best Price/Performance: measured by the ratio of size to MIPS.

In power-sensitive applications MIPS/Watt is important too.

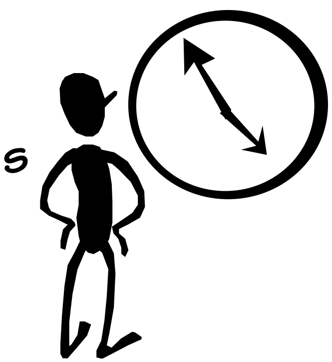
Execution Time

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

Book's 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

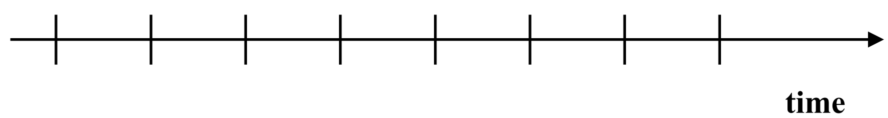
Machine A is (1/20)/(1/25) = 1.25 times faster than Machine B

Program Clock Cycles

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

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

Clock "ticks" indicate when machine-state changes (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 $\frac{1}{200 \times 10^{6}} = 5.0 \times 10^{9} = 5nS$ cycle time

Computer Performance Measure

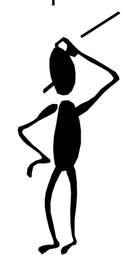
Millions of Instructions per Second

Frequency in MHz

$$\text{MIPS} = \frac{clocks/second}{\text{AVE}(clocks/instruction)}$$

CPI (Average Clocks Per Instruction)

Which of these terms are program dependent?



Historically:

PDP-11, VAX, Intel 8086	CPI > 1
Load/Store RISC machines	
MIPS, SPARC, PowerPC, miniMIPS:	CPI = 1
Modern CPUs, Pentium, Athlon, i7	CPI < 1

How to Improve Performance?

$$\frac{seconds}{program} = \boxed{\frac{clock\ cycles}{program}} \times \boxed{\frac{seconds}{cycle}} \qquad \text{MIPS} = \frac{\boxed{Freq}}{\boxed{CPI}}$$

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

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

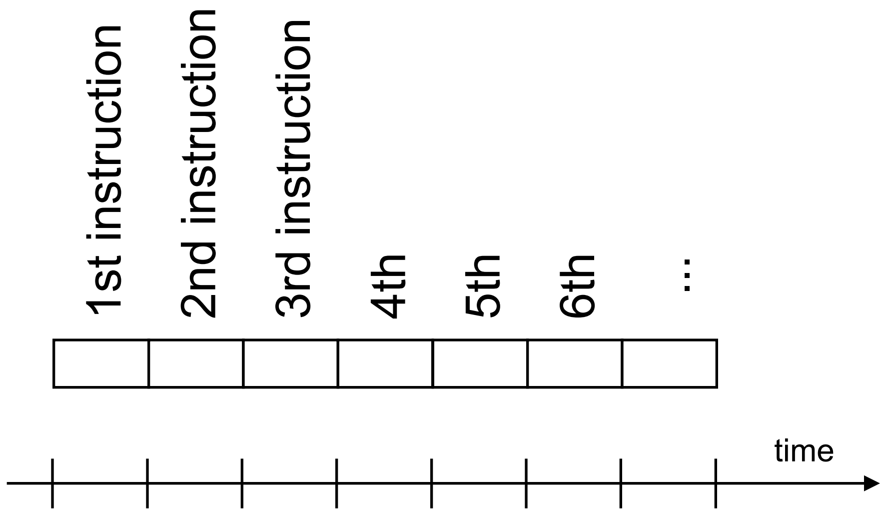
Decrease the clock cycle time or, said another way,

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



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

Performance is determined by the execution time of a program that you care about.

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 only one of the 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 10 ns. and a CPI of 0.5
Machine B has a clock cycle time of 3 ns. and a CPI of 1.5

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

$$\mathrm{MIPS}_A = \frac{freq}{CPI} = \frac{10^6/(10\times 10^9)}{0.5} = 200 \qquad \mathrm{MIPS}_B = \frac{freq}{CPI} = \frac{10^6/(3\times 10^9)}{1.5} = 222.2$$

$$Relative\ Performance = \frac{\mathrm{MIPS}_\mathrm{A}}{\mathrm{MIPS}_B} = \frac{200}{222.2} = 0.9$$

If two machines have the same ISA 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 a large piece of software. The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 2 million Class C instructions. The second compiler's code uses 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 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

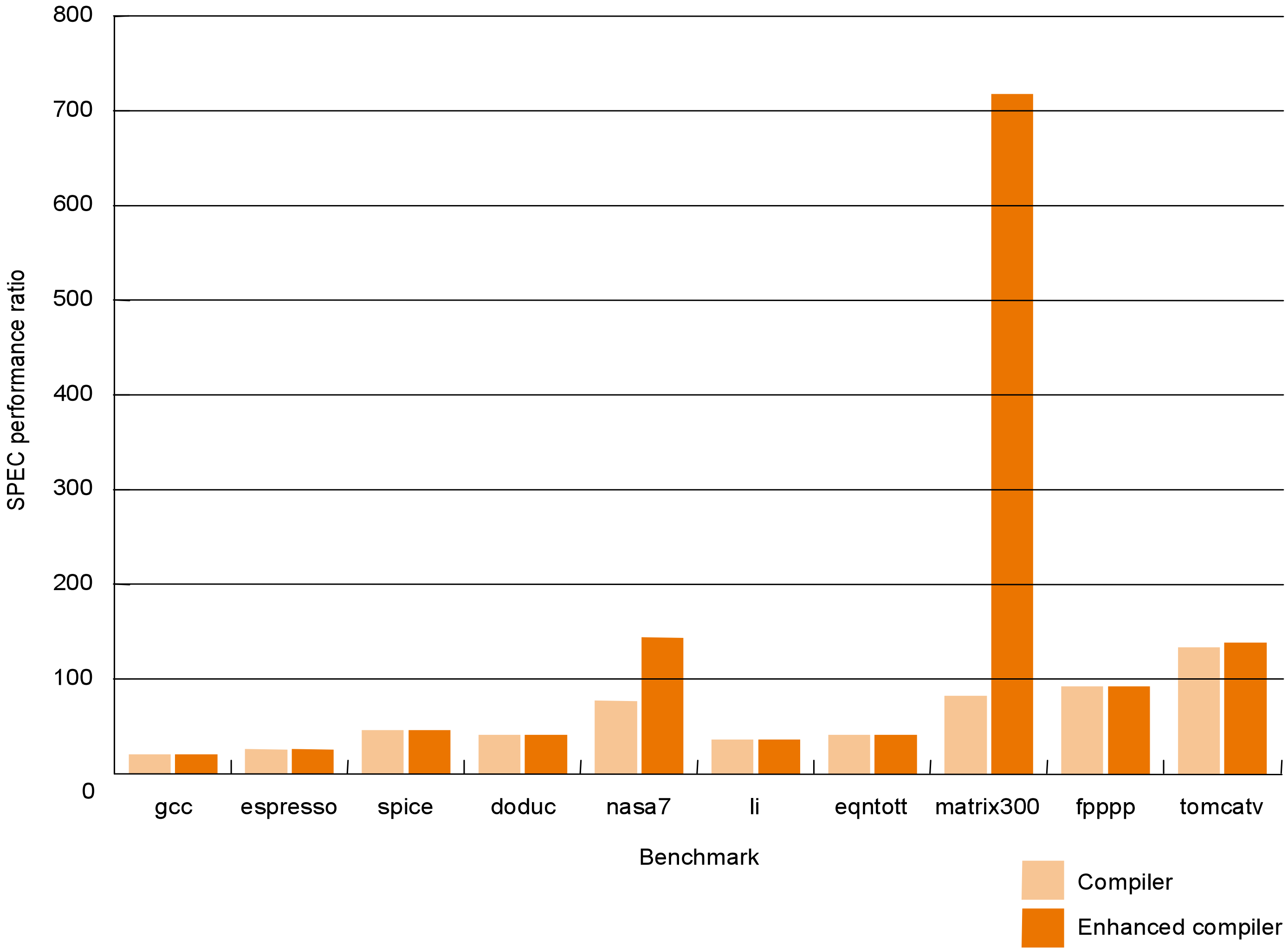
- nice for architects and designers
- easy to standardize
- can be abused

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)

SPEC '89

Compiler "enhancements" and performance



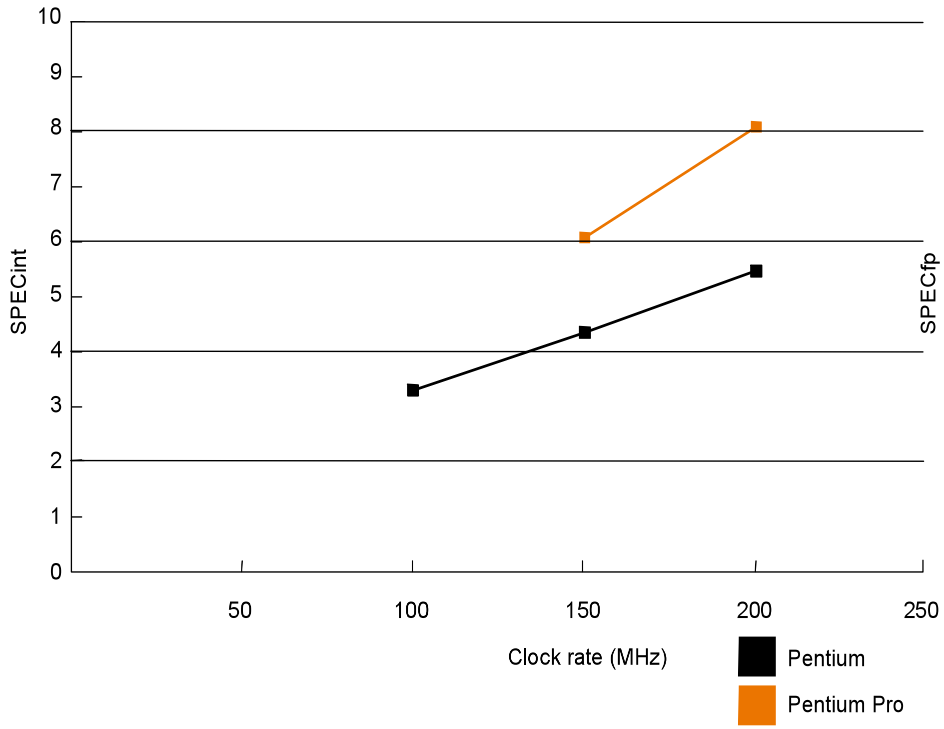
SPEC '95

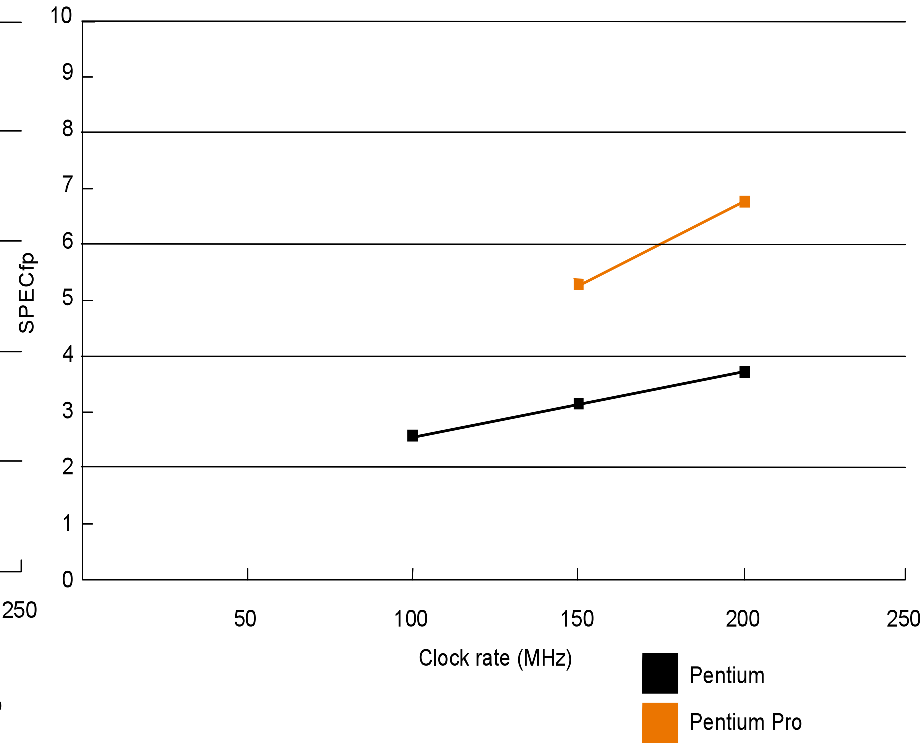
Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Image compression and decompression
perl	Manipulates 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 Naiver 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

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

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?"

$25 = 80/r + 20$ $\quad r = 16x$

How about making it 5 times faster?

$20 = 80/r + 20$ $\quad r = ?$

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 only half of the 10 seconds is spent executing floating-point instructions?

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

We are looking for a benchmark to show off the new floating -point unit described above, and want the overall benchmark to show at least a speedup of 3. What percentage 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?

$100/3 = f/5 + (100 - f) = 100 - 4f/5$ $f = 83.33$

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! Read carefully!