



香港中文大學

計算機科學與工程學系

CSC 3420

Computer Systems Architecture

Chapter 2: Cost and Performance

Performance

- **Measure, Analyse, Summarise and Report**
- **Make intelligent choices**
- **See through the marketing hype**
- **Key to understanding underlying organisational 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?*

Which of these aeroplanes has the best performance?

<u>Aeroplane</u>	<u>Passengers</u>	<u>Range (mile)</u>	<u>Speed (mph)</u>
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC-8?
- What is the advantage of Douglas DC-8-50?

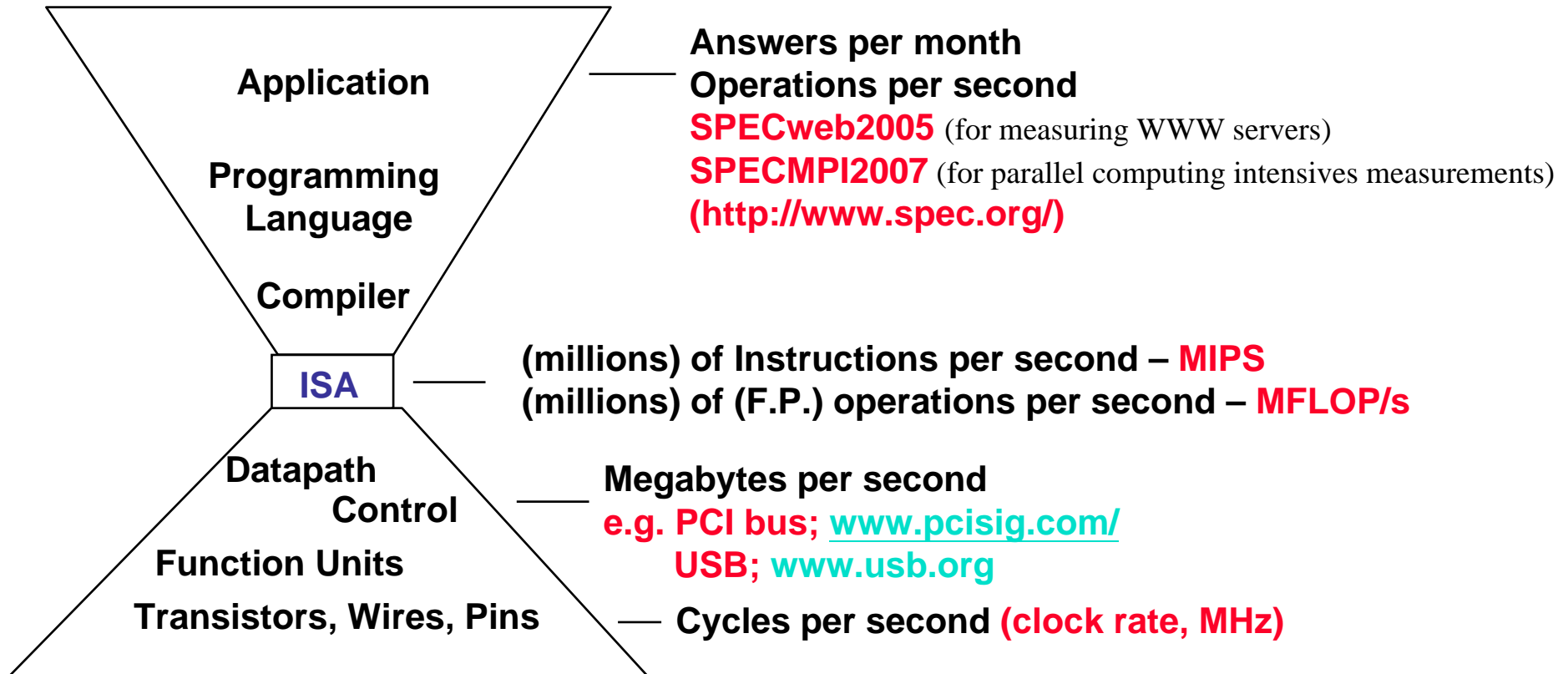
Computer Performance: TIME, TIME, TIME

- **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 in a fixed period?

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?

Metrics of performance



Relating Processor Metrics

- **CPU execution time = CPU clock cycles/program X clock cycle time**
= CPU clock cycles/program ÷ clock rate
- **CPU clock cycles/program = Instructions/program**
X average clock cycles per instruction (CPI)
- **or CPI = CPU clock cycles/program ÷ Instructions/program**
- **CPI (Cycles per Instruction) tells us something about the Instruction Set Architecture, the Implementation of that architecture, and the program measured.**

Execution Time

- **Elapsed Time**
 - counts everything (*disk and memory accesses, I/O , etc.*)
 - a useful number, but often not good for comparison purposes (**Why?**)
- **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

Definition of Performance

- For some program running on machine X,

$$\text{Performance}_x = 1 / \text{Execution time}_x$$

- Machine X is n times faster than machine Y

$$\text{Performance}_x / \text{Performance}_y = n$$

- Compare the performances
 - Machine A runs a program in 20 seconds
 - Machine B runs the same program in 25 seconds
 - How much is machine A faster than machine B?

Aspects of CPU Performance

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

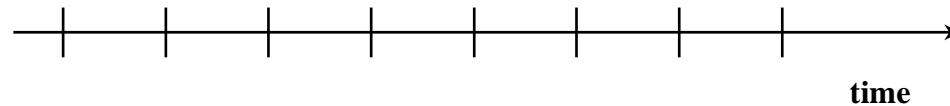
	Instruction Count	CPI	Clock Rate
Program (Algorithm)	X		
Compiler	X	(X)	
Instruction Set	X	X	(X)
Organization		X	X
Technology			X

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 = (nano; pico) seconds per cycle**
- **clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)**

A 2 Ghz. clock has a $\frac{1}{2 \times 10^9} \times 10^9 = 0.5 \text{ ns} = 500 \text{ ps}$ cycle time

How to Improve Performance ?

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

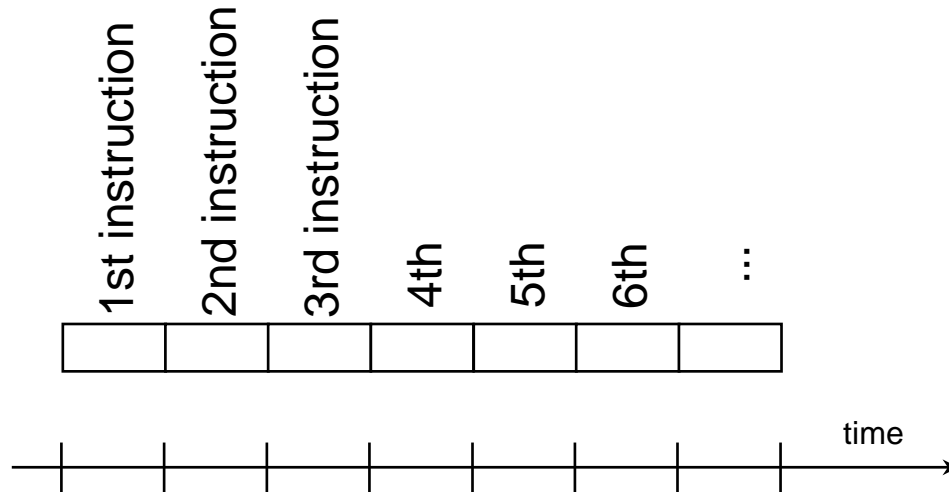
So, to improve performance (everything else being equal) you can either

Reduce the number of required cycles for a program, or

Reduce the clock cycle time or, said another way, increase the clock rate.

How many cycles are required for a program?

- Could we assume that # of cycles = # of instructions ?



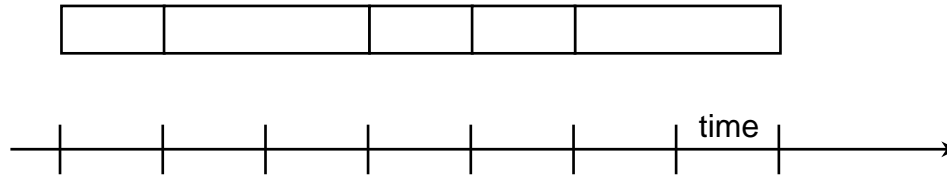
This assumption is incorrect,

different instructions take different amounts of time on different machines.

Why?

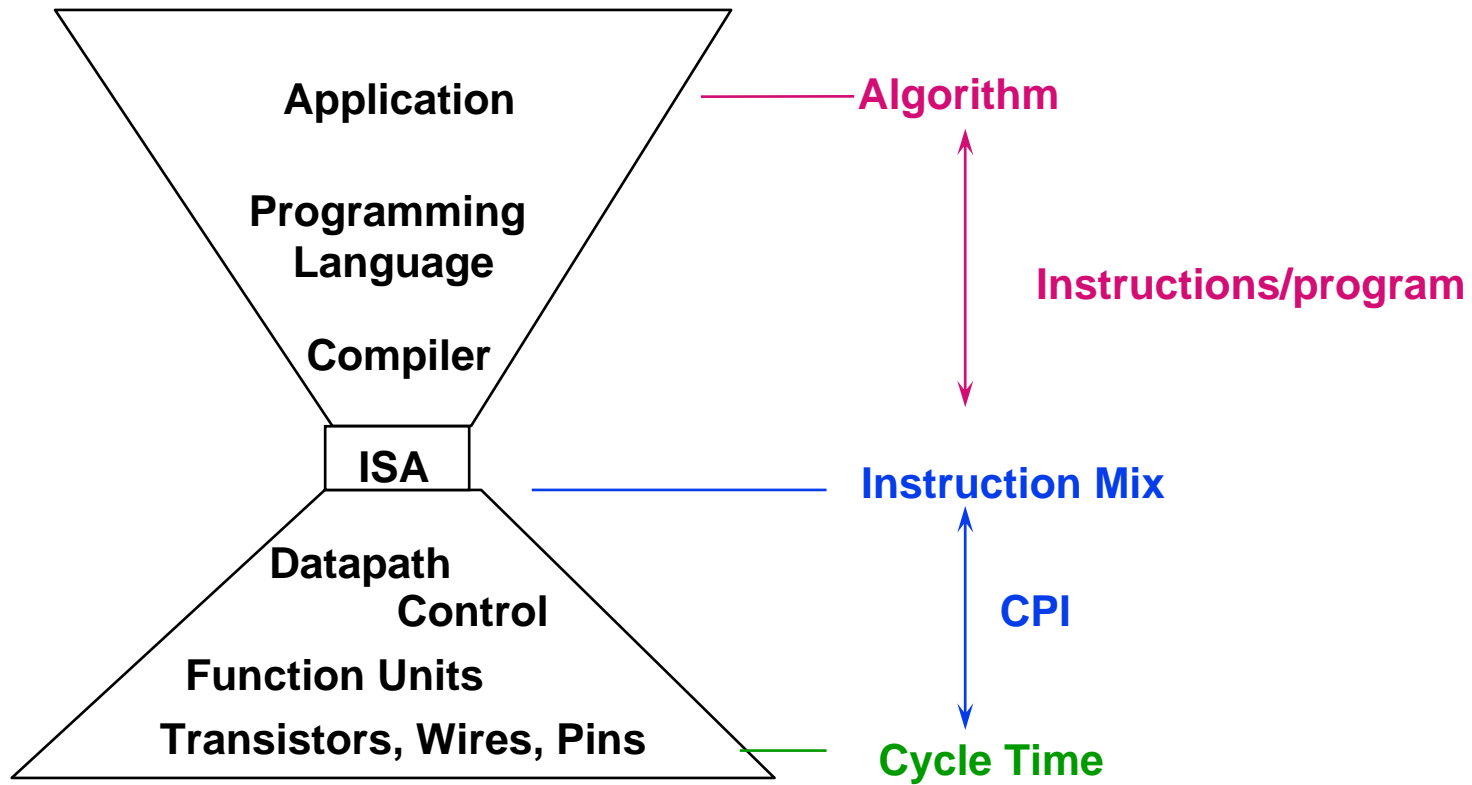
hint: remember that these are machine instructions, not lines of C code

Different numbers of cycles for different instructions



- **Multiplication takes more time than addition**
- **Floating point (Scientific Notation) operations take longer than integer ones**
- **Accessing memory takes more time than accessing registers**
- *Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*

Organisational Trade-offs



CPI (Cycle per Instruction)

“Average cycles per instruction”

$$\begin{aligned} \text{CPI} &= (\text{CPU Time} * \text{Clock Rate}) / \text{Instruction Count} \\ &= \text{Clock Cycles} / \text{Instruction Count} \end{aligned}$$

$$\text{CPU time} = \text{ClockCycleTime} \times \sum_{i=1}^n \text{CPI}_i \times \mathbf{I}_i$$

(\mathbf{I} : instruction frequency)
(i : different types of instructions)

$$\text{CPI} = \sum_{i=1}^n \text{CPI}_i \times \mathbf{F}_i \quad \text{where} \quad \mathbf{F}_i = \mathbf{I}_i \div (\text{instruction count})$$

Invest Resources where time is Spent!

Example:

Base Machine (Reg / Reg)

Op	Freq	Cycles	$CPI_i * F_i$	% Time
ALU	50%	3	1.5	44%
Load	20%	4	0.8	24%
Store	10%	5	0.5	15%
Branch	20%	3	0.6	18%

Typical Mix

Average CPI = 3.4

- How much faster would the machine be if a better data cache reduced the average load time to 2 clocks?
- How does this compare with using branch prediction to shave a cycle off the branch time?
- What if the machine could execute 2 ALU instructions in parallel?

Marketing Metrics

MIPS = Millions Instructions Per Second
= Instruction Count / (Time * 10⁶)
= Clock Rate / (average CPI * 10⁶)

Problems!

- machines with different instruction sets ?
- programs with different instruction mixes ?
 - Worse: dynamic frequency of instructions
- uncorrelated with performance

MFLOP/S = FP Operations / (Time * 10⁶) (millions FP operations per sec.)

- machine dependent
- often not where time is spent; application types dependent

Example

- **Our favourite 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?"**

Hint, can easily work this out from basic principles

Solution:

No of machines cycles needed = 10 (sec) * 400*10⁶ (Hz) * 1.2

To finish it in 6 sec., the clock rate = [10*400*10⁶*1.2] / 6 = 800 MHz.

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*

Performance

- Performance is determined by execution time
- Can any one of these variables on its own representing 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.

CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA).
- For some programs,
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
- Which machine is faster for this program, and by how much?

*Equivalent Inst. Time : Machine A = 10 * 2.0 ns = 20 ns*

*Machine B = 20 * 1.2 ns = 24 ns*

Therefore A is 24/20 = 1.2 times faster than B

What if A's CPI = 2.5?

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?

Number of Instructions Example

- 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?

Effective CPI: Sequence A = $(2*1 + 1*2 + 2*3) / 5 = 2.0$

Sequence B = $(4*1 + 1*2 + 1*3) / 6 = 1.5$

Sequence B is 1.33 (2.0/1.5) times faster

(even though there are 6 instructions in sequence B and 5 in sequence A)

MIPS example

- **Two different compilers are being tested for a 100 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, 2 million Class B instructions, and 2 million Class C instructions.

The second compiler's code uses 8 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- 1. Which sequence will be faster according to the no. of Instructions?**
- 2. Which sequence will be faster according to execution time?**

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 standardise
 - can be abused
- **SPEC (System Performance Evaluation Cooperative)**
 - <http://www.spec.org>
 - companies have agreed on a set of real program and inputs
 - can still be abused
 - valuable indicator of performance (and compiler technology)

Why Do Benchmarks?

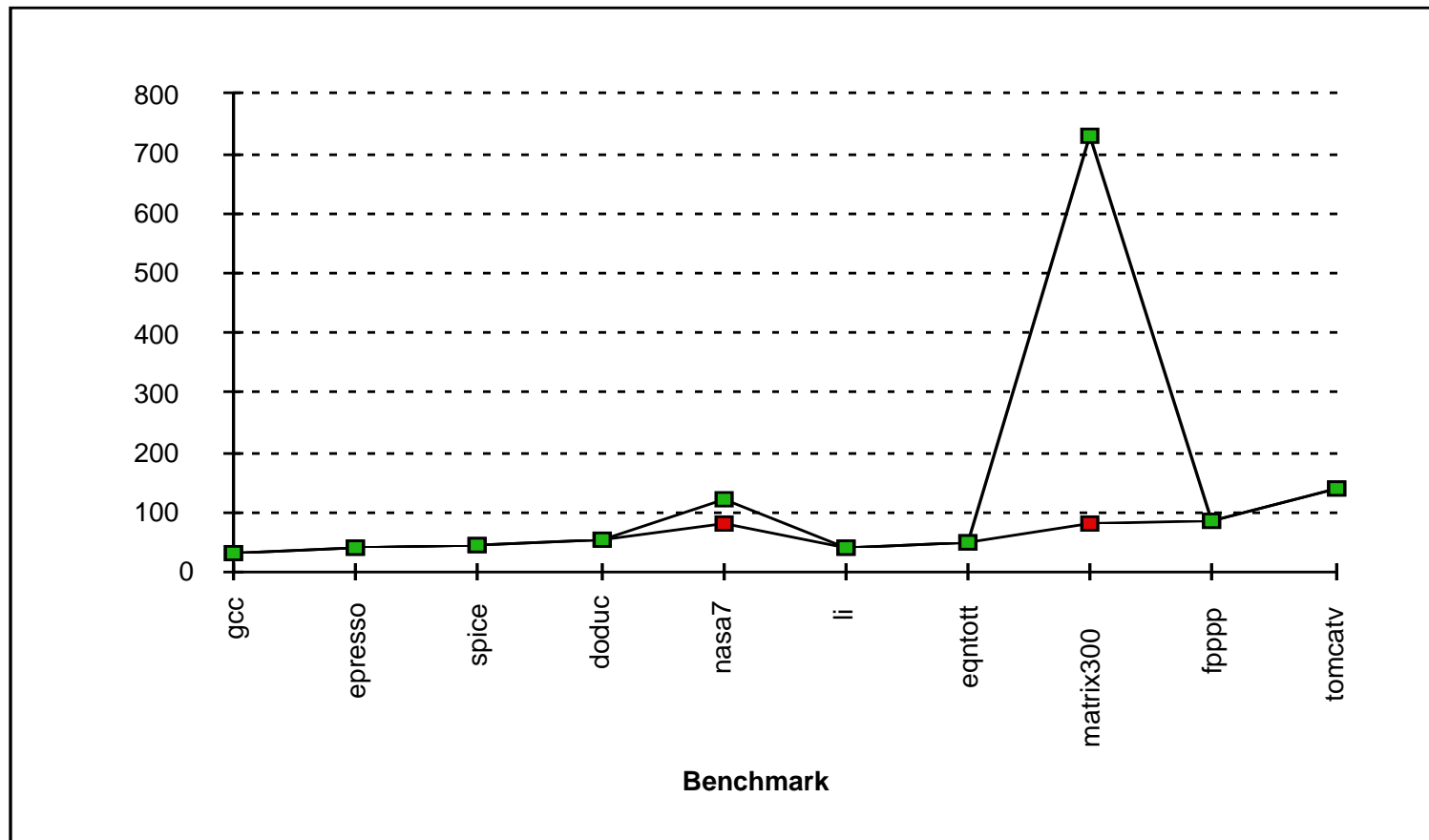
- How we **evaluate differences**?
 - Different systems
 - Changes to a single system
- **Provide a target**
 - Benchmarks should represent large class of important programs
 - Improving benchmark performance should help many programs
- For better or worse, **benchmarks shape a field**
- **Good ones accelerate progress**
 - good target for development
- **Bad benchmarks hurt progress**
 - Help real programs versus sell machines (or papers?)
 - Inventions that help real programs don't always help benchmark

Programs to Evaluate Processor Performance

- **(Toy) Benchmarks**
 - Useful early in design to identify peak performance and bottlenecks
 - 10-100 line, e.g., sieve, puzzle, quicksort
- **Synthetic Benchmarks**
 - Attempt to match average frequencies of real workloads
 - Easy to run in design cycle to get a feel for actual performance
 - e.g., Whetstone, dhrystone
- **Kernels**
 - Time critical excerpts of real programs
 - Give indication of performance in specific application area
 - e.g., Livermore loops
- **Real programs**
 - Useful real performance figures for specific program
 - e.g., gcc, spice
- **Actual target workload**
 - Representative, but non-portable, difficult to run or measure

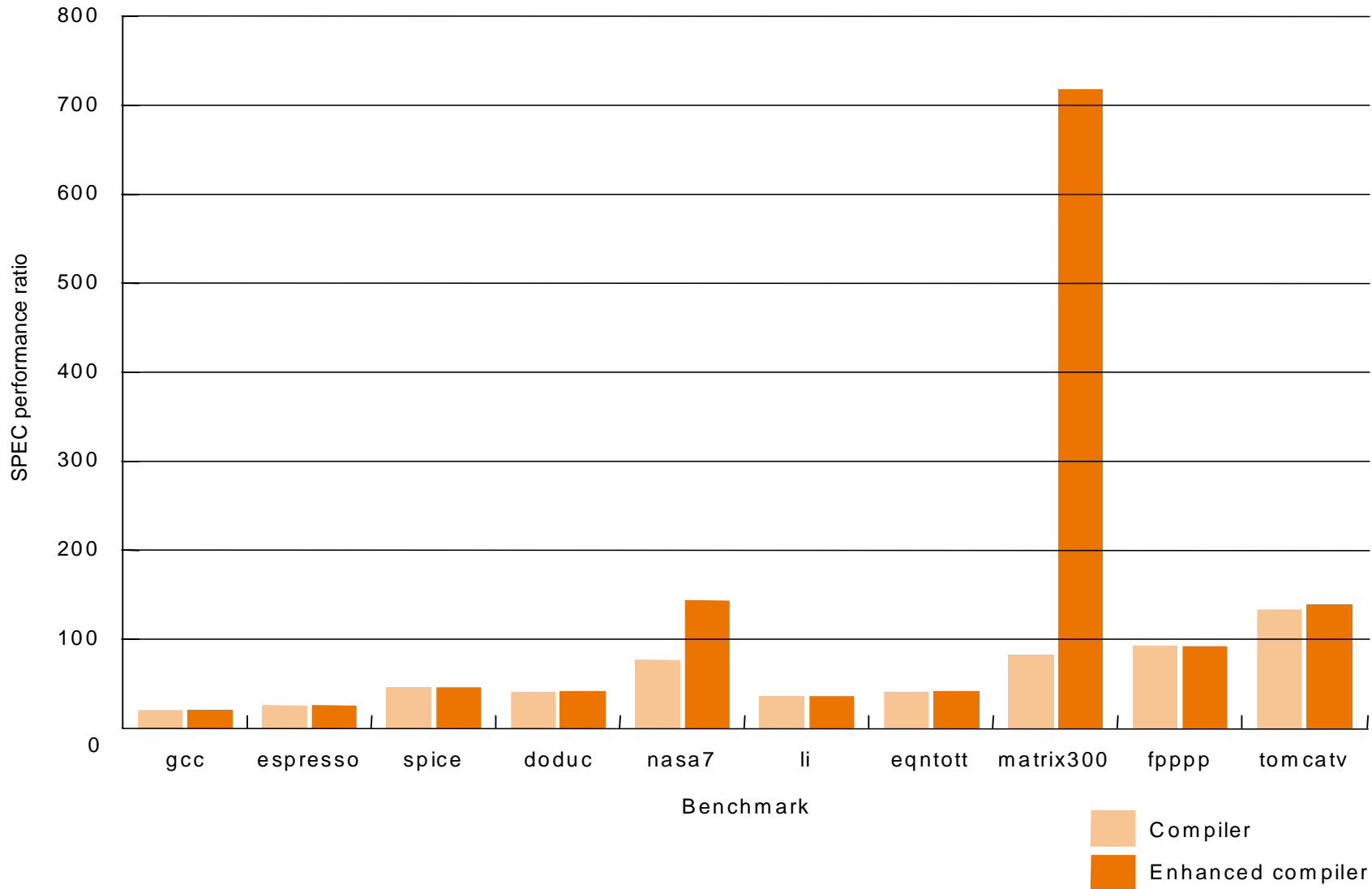
SPEC first round

- First round 1989; 10 programs, single number to summarise performance
- One program: 99% of time in a single line of code
- New front-end compiler could improve dramatically



SPEC '89

- **Compiler “enhancements” and performance**



SPEC Evolution

- Second round; SpecInt92 (6 integer programs) and SpecFP92 (14 floating point programs)

Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

spice:

```
unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)=memcpy(b,a,c)"
```

```
wave5:/ali=(all,dcom=nat)/ag=a/ur=4/ur=200
```

```
nasa7:/norecu/ag=a/ur=4/ur2=200/lc=blas
```

- Add SPECbase: one flag setting for integer programs & one for FP
- Third round; 1995; SPEC95 (retired in 2000)
(<http://www.spec.org/cpu95/news/cpu95descr.html#benchmarks>)
- More recent: SPEC CPU2000 (retired Feb. 2007) & CPU2006
- **“benchmarks useful for 3 years”**

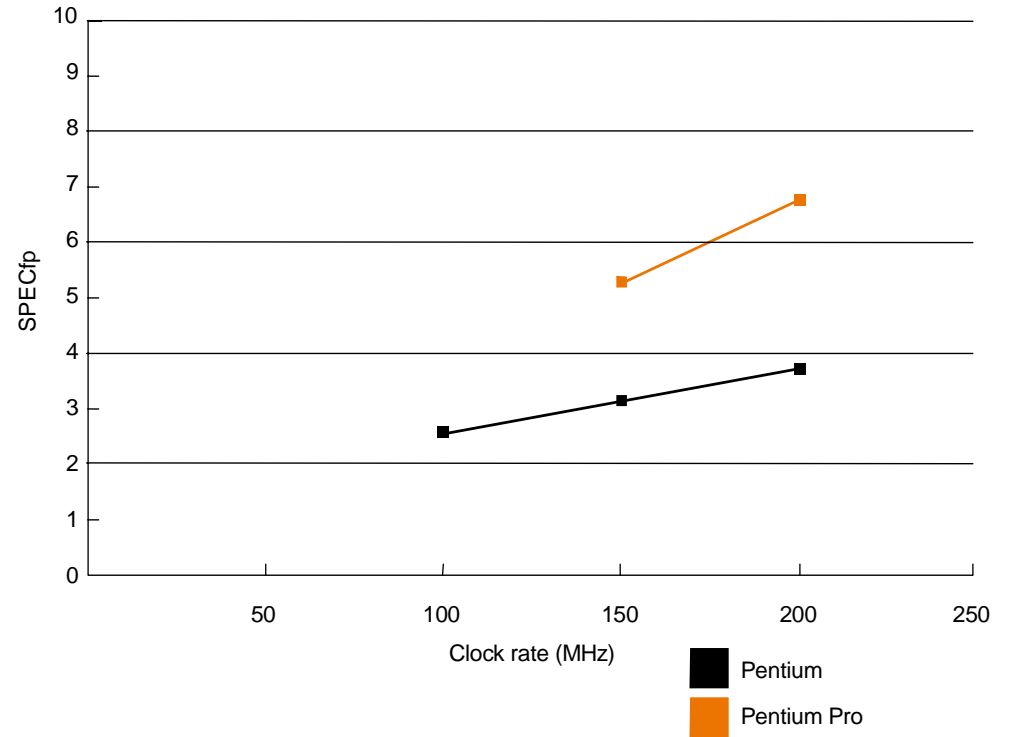
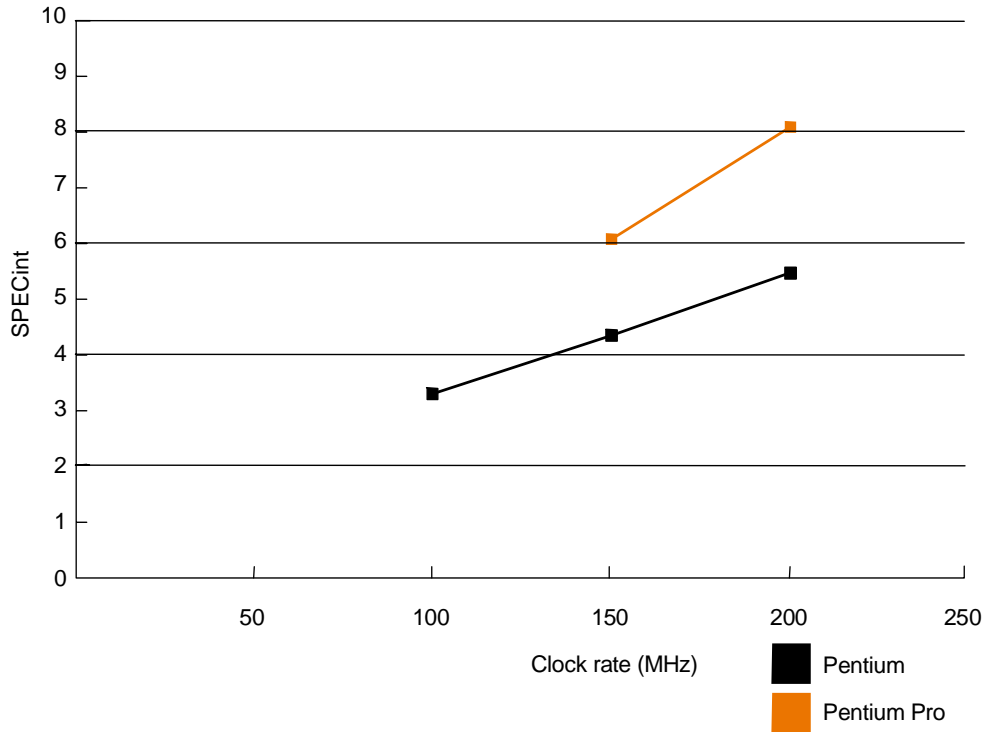
How to Summarise Performance

- Arithmetic mean (or weighted arithmetic mean) tracks execution time: $\text{SUM}(T_i) / n$ or $\text{SUM}(W_i * T_i)$
- Harmonic mean (or weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution rate: $n / \text{SUM}(1/R_i)$ or $n / \text{SUM}(W_i/R_i)$
- Normalised execution time is handy for scaling performance (e.g., time on reference machine \div time on measured machine)

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
jpeg	Graphic 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 Navier 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?

CINT2000

Name	Language	Remarks
164.gzip	C	Data compression utility
175.vpr	C	FPGA circuit placement and routing
176.gcc	C	C compiler
181.mcf	C	Minimum cost network flow solver
186.crafty	C	Chess program
197.parser	C	Natural language processing
252.eon	C++	Ray tracing
253.perlbm	C	Perl
254.gap	C	Computational group theory
255.vortex	C	Object Oriented Database
256.bzip2	C	Data compression utility
300.twolf	C	Place and route simulator

CFP2000

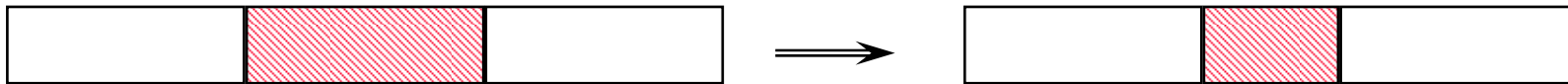
Name	Language	remarks
168.wupwise	Fortran 77	Quantum chromodynamics
171.swim	Fortran 77	Shallow water modeling
172.mgrid	Fortran 77	Multi-grid solver in 3D potential field
173.applu	Fortran 77	Parabolic/elliptic partial differential equations
177.mesa	C	3D Graphics library
178.galgel	Fortran 90	Fluid dynamics: analysis of oscillatory instability
179.art	C	Neural network simulation; adaptive resonance theory
183.quake	C	Finite element simulation; earthquake modeling
187.facerec	Fortran 90	Computer vision: recognizes faces
188.amp	C	Computational chemistry
189.lucas	Fortran 90	Number theory: primality testing
191.fma3d	Fortran 90	Finite element crash simulation
200.sixtrack	Fortran 77	Particle accelerator model
301.apsi	Fortran 77	Solves problems regarding temperature, wind, velocity and distribution of pollutants

Remember

- **Performance is specific to a particular program/s**
 - **Total execution time is a consistent summary of performance**
- **For a given architecture performance increases come from:**
 - **increases in clock rate (without adverse CPI affects)**
 - **improvements in processor organisation that lower CPI**
 - **compiler enhancements that lower CPI and/or instruction count**
- **Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance**
- **You should not always believe everything you read! Read carefully!**
(read papers & newspaper articles)

Amdahl's Law

**Execution Time After Improvement = Execution Time Unaffected
+ (Execution Time Affected / Amount of Improvement)**



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,

$$\text{ExTime (with } E) = \left((1 - F) + \frac{F}{S} \right) \times \text{ExTime (without } E)$$

$$\text{Speedup (with } E) = \frac{\text{ExTime (without } E)}{\text{ExTime (with } E)} = \frac{1}{(1 - F) + \frac{F}{S}}$$

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?

- ***Principle: Make the common case fast***

Further Examples

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

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much 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?

Performance Evaluation Summary

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

- Time is the measure of computer performance!
- Good products created when have:
 - Good benchmarks
 - Good ways to summarise performance
- However, sometimes choice is between improving product for real programs and improving product to get more sales
 - sales almost always wins
- Remember Amdahl's Law : Speedup is limited by unimproved part of program