The Role of Performance

Performance

- *Performance is the key to understanding underlying motivation for the hardware and its organization*
- Measure, report, and summarize performance to enable users to
  - make intelligent choices
  - see through the marketing hype!

- *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?*

What do we measure? Define performance….

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud 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 Boeing 747 than the Douglas DC-8?

- *So which of these airplanes has the best performance?!*

Computer Performance: TIME, TIME, TIME!!!

- *Response Time* (*elapsed time, latency*):
  - how long does it take for *my* job to run?
  - how long does it take to execute (start to finish) *my* job?
  - how long must *I* wait for the database query?

Individual user concerns…

- *Throughput*:
  - how *many* jobs can the machine run at once?
  - what is the *average* execution rate?
  - how *much* work is getting done?

Systems manager concerns…

- *If we upgrade a machine with a new processor what do we increase?*

Execution Time

- *Elapsed Time*
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

  elapsed time = CPU time + wait time (I/O, other programs, etc.)

- *CPU time*
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

  CPU time = user CPU time + system CPU time

  ⇒ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

Definition of Performance

- For some program running on machine X:

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

- *X is n times faster than Y* means:

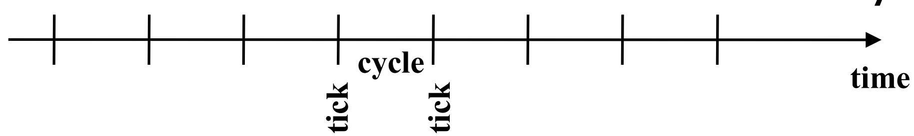
$$\text{Performance}_X \; / \; \text{Performance}_Y = n$$

Clock Cycles

- Instead of reporting execution time in seconds, we often use *cycles*. In modern computers hardware events progress cycle by cycle: in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles

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

- *Clock ticks* indicate start and end of cycles:



- *cycle time* = time between ticks = seconds per cycle
- *clock rate* (*frequency*) = cycles per second (1 Hz. = 1 cycle/sec, 1 MHz. = 10^6 cycles/sec)
- *Example*: A 200 Mhz. clock has a cycle time

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

Performance Equation I

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

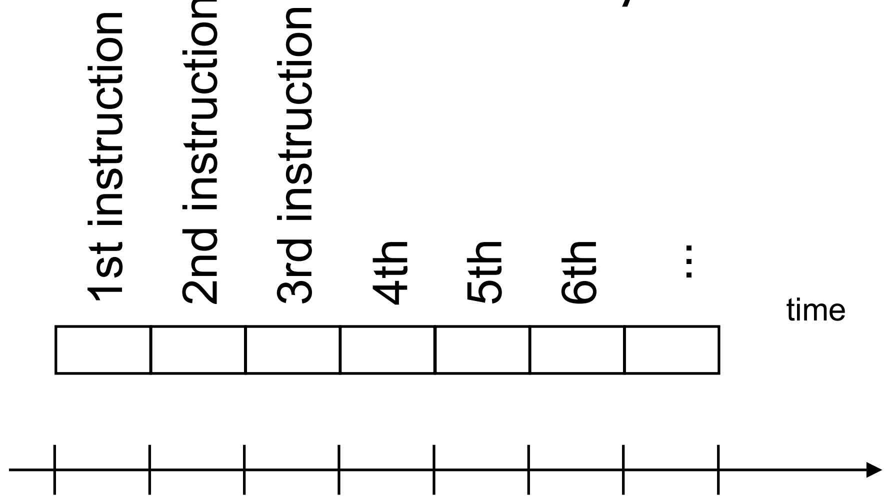
equivalently

$$\begin{array}{l}\text{CPU execution time} \\ \text{for a program}\end{array} = \begin{array}{l}\text{CPU clock cycles} \\ \text{for a program}\end{array} \times \text{Clock cycle time}$$

- So, to improve performance one can either:
  - reduce the number of cycles for a program, or
  - reduce the clock cycle time, or, equivalently,
  - increase the clock rate

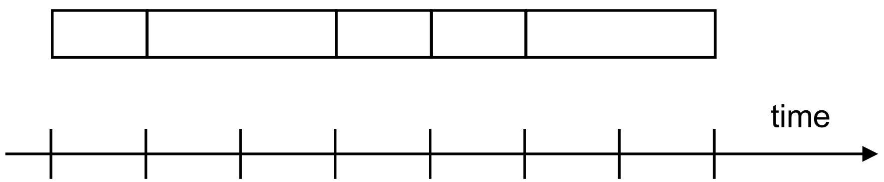
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



- *This assumption is incorrect!* Because:
  - Different instructions take different amounts of time (cycles)
  - Why…?

How many cycles are required for a program?



- Multiplication takes more time than addition
- Floating point 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 because it means changing the hardware design. More later…

Example

- Our favorite program runs in 10 seconds on computer A, which has a 400Mhz. 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?*

Terminology

- 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)
  - (*average*) *CPI* (cycles per instruction)
    - a floating point intensive application might have a higher average CPI
  - *MIPS* (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance Measure

- *Performance is determined by execution time*

- Do any of these 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

Performance Equation II

$$\text{CPU execution time for a program} = \text{Instruction count for a program} \times \text{average CPI} \times \text{Clock cycle time}$$

- *Derive the above equation from Performance Equation I*

CPI Example I

- 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

- *Which 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?*

CPI Example II

- 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 1, 2 and 3 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?*

MIPS Example

- 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 1, 2 and 3 cycles (respectively). Both compilers are used to produce code for a large piece of software.
- Compiler 1 generates code with 5 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- Compiler 2 generates code with 10 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.

- *Which sequence will be faster according to MIPS?*
- *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 standardize
  - can be abused!

- Benchmark suites
  - Perfect Club: set of application codes
  - Livermore Loops: 24 loop kernels
  - Linpack: linear algebra package
  - SPEC: mix of code from industry organization

SPEC (System Performance Evaluation Corporation)

- Sponsored by industry but independent and self-managed – trusted by code developers and machine vendors
- Clear guides for testing, see www.spec.org
- Regular updates (benchmarks are dropped and new ones added periodically according to relevance)
- Specialized benchmarks for particular classes of applications
- Can still be abused..., by selective optimization!

SPEC History

- First Round: SPEC CPU89
  - 10 programs yielding a single number
- Second Round: SPEC CPU92
  - SPEC CINT92 (6 integer programs) and SPEC CFP92 (14 floating point programs)
  - compiler flags can be set differently for different programs
- Third Round: SPEC CPU95
  - new set of programs: SPEC CINT95 (8 integer programs) and SPEC CFP95 (10 floating point)
  - single flag setting for all programs
- Fourth Round: SPEC CPU2000
  - new set of programs: SPEC CINT2000 (12 integer programs) and SPEC CFP2000 (14 floating point)
  - single flag setting for all programs
  - programs in C, C++, Fortran 77, and Fortran 90

CINT2000 (Integer component of SPEC CPU2000)

Program	Language	What It Is
164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

CFP2000 (Floating point component of SPEC CPU2000)

Program	Language	What It Is
168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.equake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

SPEC CPU2000 reporting

- Refer SPEC website www.spec.org for documentation
- Single number result – geometric mean of normalized ratios for each code in the suite
- Report precise description of machine
- Report compiler flag setting

CFP2000 Result

Advanced Micro Devices

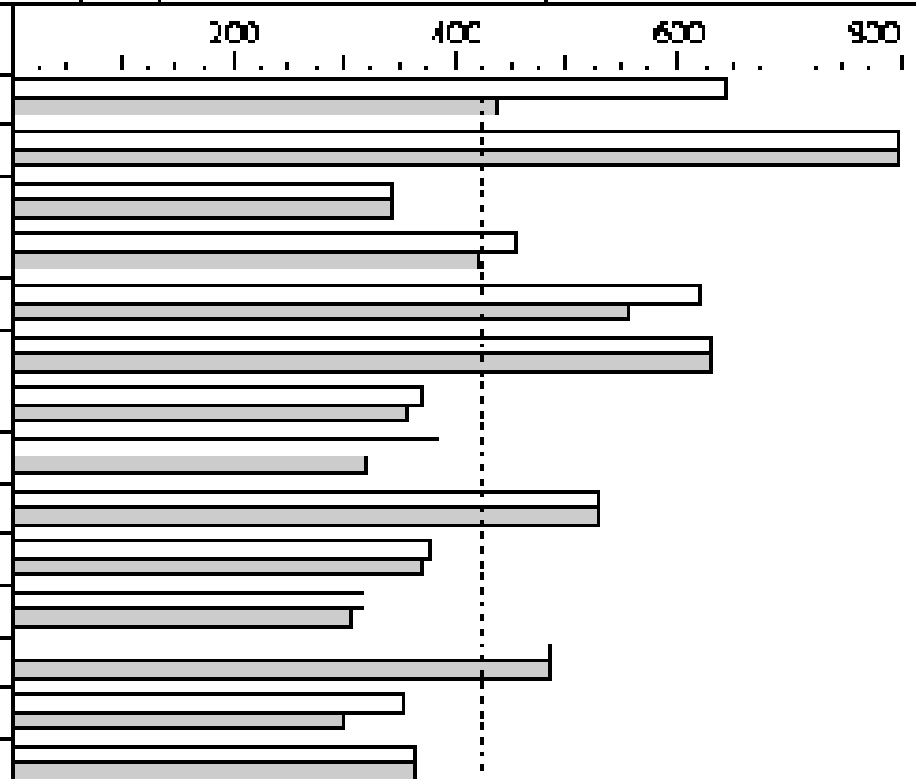
Gigabyte GA-7DX Motherboard, 1.4GHz Athlon Processor

SPECfp2000 = 458

SPECfp_base2000 = 426

SPEC license #: 49 | Tested by: AMD Austin TX | Test date: May-2001 | Hardware Avail: Jun-2001 | Software Avail: Jun-2001

Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio
168.wupwise	1600	366	438	249	644
171.swim	3100	389	797	389	797
172.mgrid	1800	525	343	525	343
173.applu	2100	500	420	463	453
177.mesa	1400	252	556	225	621
178.galgel	2900	461	629	461	630
179.art	2600	725	359	701	371
183.equake	1300	403	323	337	386
187.facerec	1900	358	531	357	531
188.ammp	2200	591	372	585	376
189.lucas	2000	654	306	641	312
191.fma3d	2100	434	483	434	483
200.sixtrack	1100	366	301	311	354
301.apsi	2600	714	364	714	364

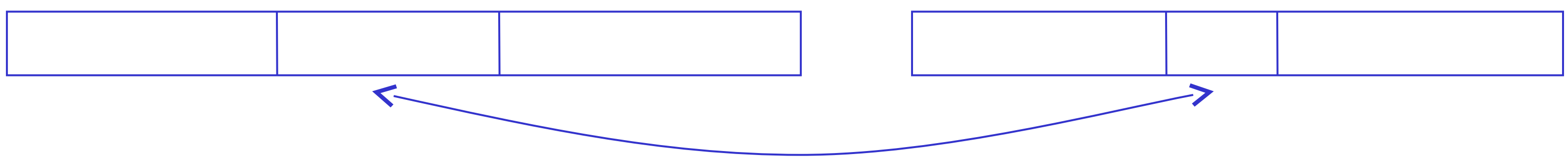


Specialized SPEC Benchmarks

- I/O
- Network
- Graphics
- Java
- Web server
- Transaction processing (databases)

Amdahl's Law

- Execution Time After Improvement =
  Execution Time Unaffected + (Execution Time Affected / Rate of Improvement)



- *Example*:
  Improved part of code
  - 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?*

- <u>Design Principle</u>: *Make the common case fast*

Examples

- Suppose we enhance a machine making all floating-point instructions run five times faster. 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?*

Summary

- Performance is specific to a particular program
  - 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 organization 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! See newspaper articles, e.g., Exercise 2.37!!