Performance

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2020

CPU Performance

Chap. 1.6, 2.13

Performance Issues

- Measure, analyze, report, and summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

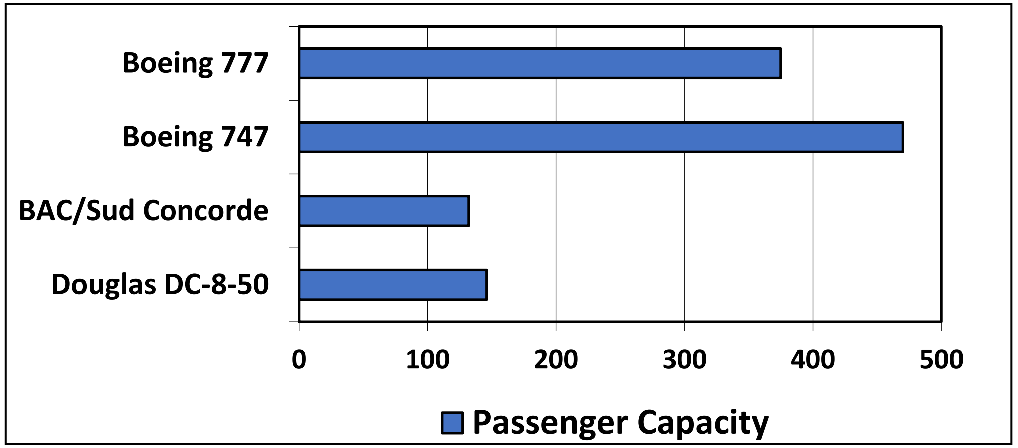
- Questions
  - Why is some hardware better than others for different programs?
  - What factors of system performance are hardware related?
    (e.g., Do we need a new machine or a new operating system?)
  - How does the machine's instruction set affect performance?

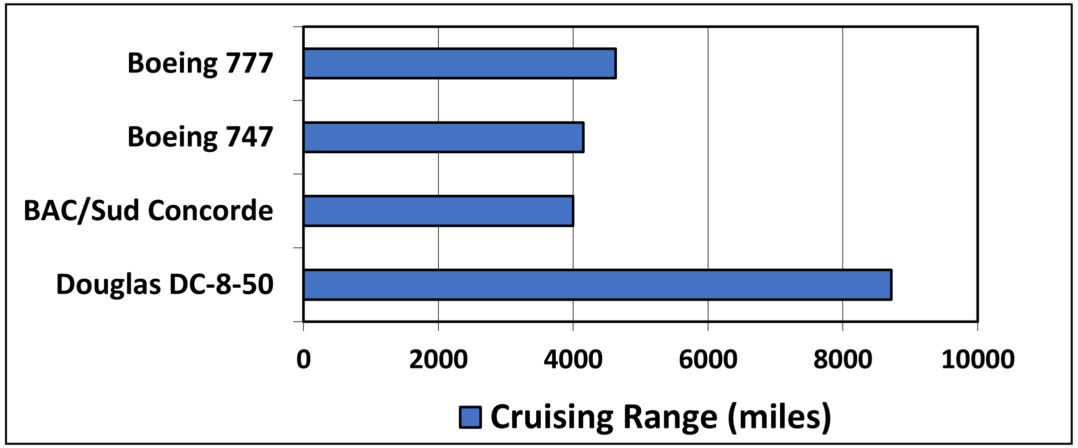
Understanding Performance

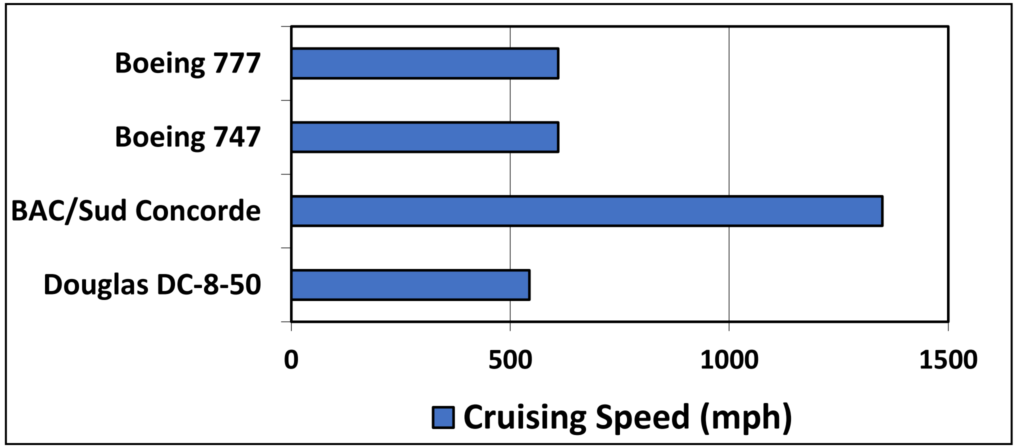
- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system (microarchitecture)
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

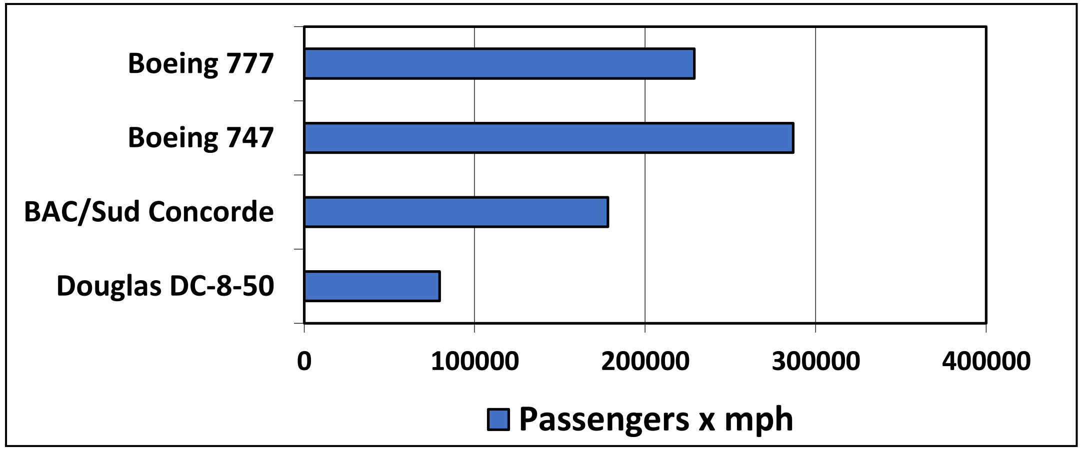
Defining Performance

- Which airplane has the best performance?









Performance Metric

- Response time (≈ execution time, latency)
  - The time between the start and completion of a task
  - How long does it take for my job to run?
  - How long must I wait for the database query?
- Throughput (≈ bandwidth)
  - The total amount of work done in a given time
  - How much work is getting done per unit time?
  - What is the average execution rate?
- What if …
  - We replace the processor with a faster version?
  - We add more processors?

Relative Performance

- Define

$$Performance = 1/Execution\ Time$$

- "X is *n* times faster than Y"

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ time_Y}{Execution\ time_X} = n$$

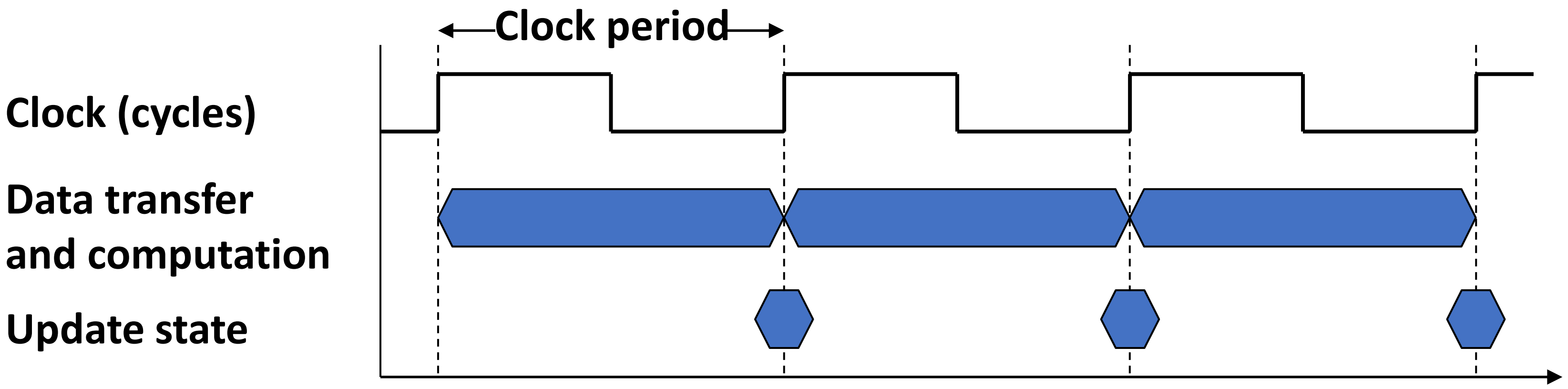
- Example: time taken to run a program
  - 10s on machine A, 15s on machine B
  - Execution $Time_B$ / Execution $Time_A$ = 15s / 10s = 1.5
  - Machine A is 1.5 times faster than machine B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - e.g., processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance
- Our focus: User CPU time

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12} s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^9 Hz

CPU Time

$$CPU\ Time = CPU\ Clock\ Cycles \times Clock\ Cycle\ Time$$

$$= \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate (or decreasing the clock cycle time)
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2x clock cycles

- How fast must Computer B's clock be?

$$Clock\ Cycles_A = CPU\ Time_A \times Clock\ Rate_A = 10s \times 2GHz = 20 \times 10^9$$

$$Clock\ Rate_B = \frac{Clock\ Cycles_B}{CPU\ Time_B} = \frac{1.2 \times Clock\ Cycles_A}{6s} = \frac{1.2 \times 20 \times 10^9}{6s} = 4GHz$$

Instruction Count and CPU

$$Clock\ Cycles = \ Instruction\ Count \times \text{CPI (Cycles Per Instruction)}$$

$$CPU\ Time = \ Instruction\ Count \times \text{CPI} \times Clock\ Cycle\ Time$$

$$= \frac{Instruction\ Count \times \text{CPI}}{Clock\ Rate}$$

- Instruction count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPU, average CPI affected by instruction mix

CPI Example

- Computer A: Cycle time = 250ps, CPI = 2.0
- Computer B: Cycle time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$CPU\ Time_A = Instruction\ Count\ (IC) \times CPI_A \times Cycle\ Time_A$$
$$= IC \times 2.0 \times 250ps = IC \times 500ps$$
$$CPU\ Time_B = Instruction\ Count\ (IC) \times CPI_B \times Cycle\ Time_B$$
$$= IC \times 1.2 \times 500ps = IC \times 600ps$$
$$\frac{CPU\ Time_B}{CPU\ Time_A} = \frac{IC \times 600ps}{IC \times 500ps} = 1.2$$

A is faster than B by 1.2 times

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$Clock\ Cycles = \sum_{i=1}^{n} (CPI_i \times Instruction\ Count_i)$$

- Weighted average CPI

$$CPI = \frac{Clock\ Cycles}{Instruction\ Count} = \sum_{i=1}^{n} \left(CPI_i \times \underbrace{\frac{Instruction\ Count_i}{Instruction\ Count}}_{\textbf{Relative frequency}} \right)$$

CPI Example

- Alternative compiled code sequences A and B

Class	ALU	Load/Store	Branch
CPI for class	1	2	3
IC in sequence A	200	100	200
IC in sequence B	400	100	100

- Sequence A: IC = 500
  - Clock cycles
    = 200 x 1 + 100 x 2 + 200 x 3
    = 1000
  - Average CPI = 1000 / 500 = 2.0

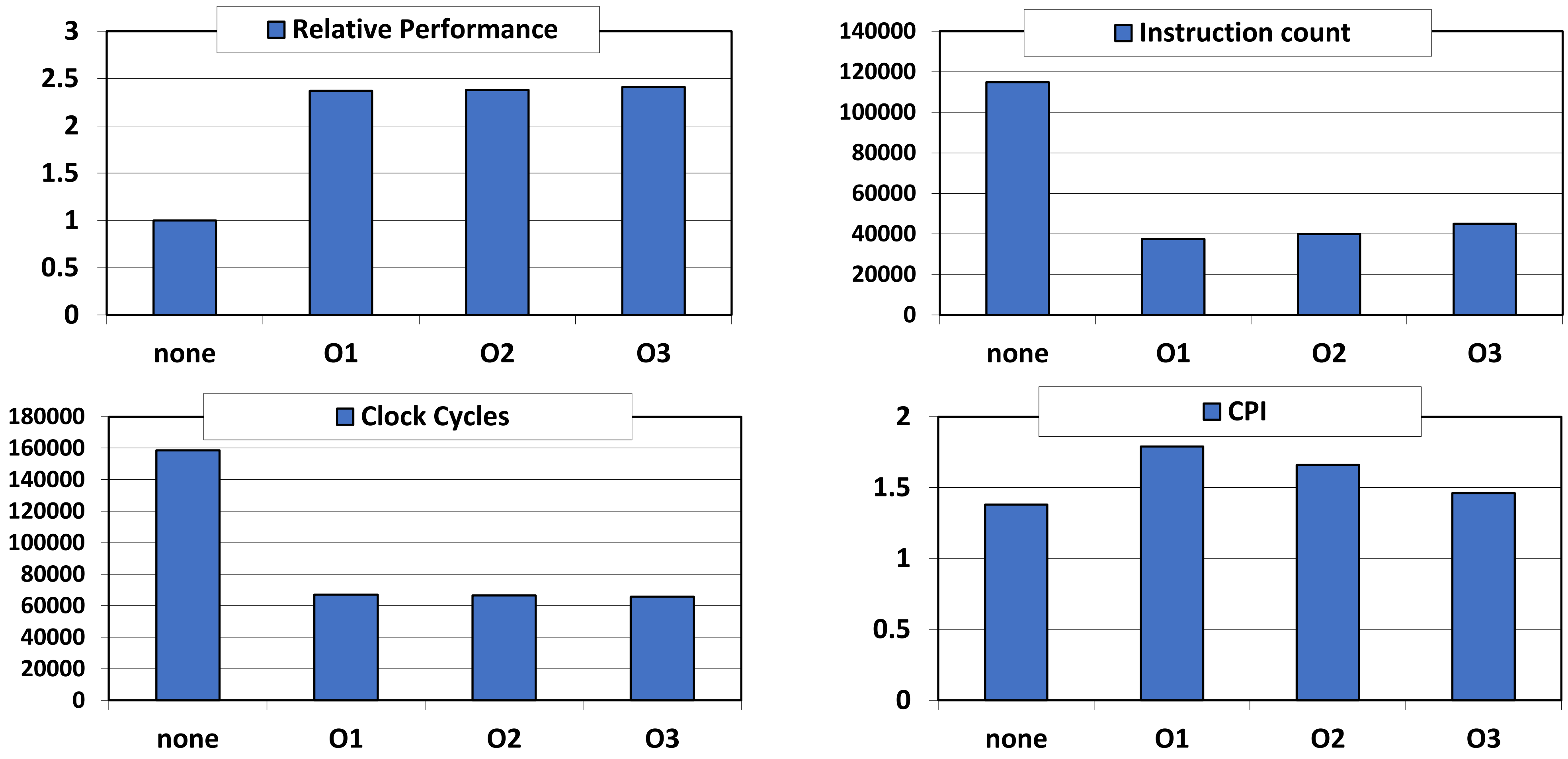
- Sequence B: IC = 600
  - Clock cycles
    = 400 x 1 + 100 x 2 + 100 x 3
    = 900
  - Average CPI = 900 / 600 = 1.5

C Sort Example (Revisited)

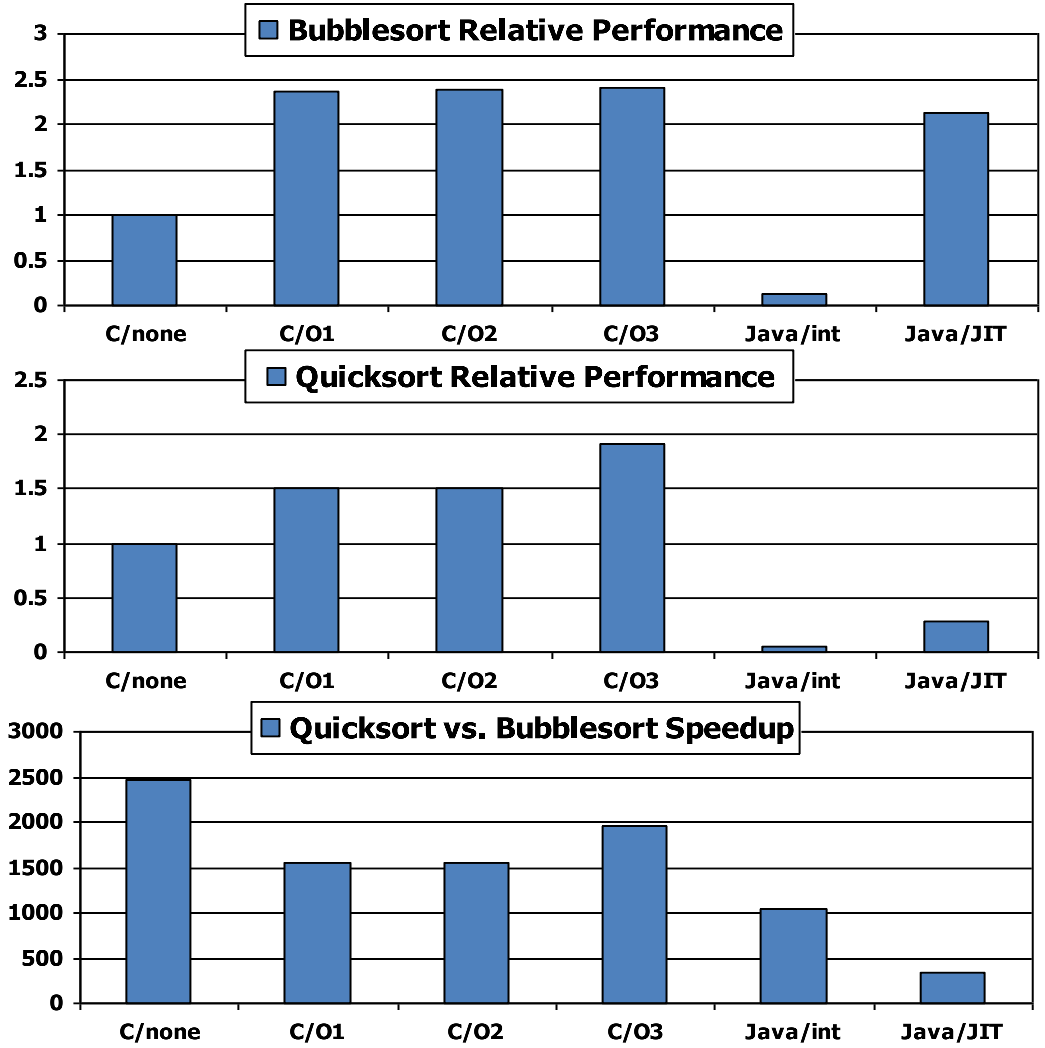
```
void swap(long long v[], long long k)
{
   long long temp;

   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}

void sort(long long v[], size_t n)
{
    size_t i, j;
    for (i = 1; i < n; i++)
        for (j = i - 1; j >= 0 && v[j] > v[j+1]; j--) {
            swap(v, j);
}
```

Effect of Compiler Optimization

- Compiled with gcc on Pentium 4 under Linux



Effect of Language and Algorithm



Lessons

- Instruction count and CPI are not good performance indicators in isolation
- Compiler optimizations are sensitive to the algorithm
- Java/JIT compiled code is significantly faster than JVM interpreted
  - Comparable to optimized C in some cases
- Nothing can fix a dumb algorithm!

Iron Law of CPU Performance

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

$$= Instruction\ Count \times CPI \times Clock\ Cycle\ Time$$

	Instruction Count	CPI	Clock Cycle
Algorithm	○	△	
Programming language	○	○	
Compiler	○	○	
ISA	○	○	○
Microarchitecture		○	○
Technology			○

Fallacy: MIPS as a Performance Metric

- MIPS (Millions of Instructions Per Second) doesn't account for
  - Differences in ISAs between computers
  - Differences in complexity between instructions

$$MIPS = \frac{Instruction\ count}{Execution\ time\ \times\ 10^6}$$

$$= \frac{Instruction\ count}{\frac{Instruction\ count\ \times CPI}{Clock\ rate} \times\ 10^6} = \frac{Clock\ rate}{CPI\ \times\ 10^6}$$

  - CPI varies between programs on a given CPU

Benchmarking

Chap. 1.9

Benchmarks

- How to measure the performance?
  - Performance best determined by running a real application
  - Use programs typical of expected workload

- Small benchmarks
  - Nice for architects and designers
  - Easy to standardize
  - Can be abused

SPEC CPU Benchmark

- SPEC (Standard Performance Evaluation Corporation)
  - A non-profit organization that aims to "produce, establish, maintain and endorse a standardized set" of performance benchmarks for computers
  - CPU, Power, HPC (High-Performance Computing), Web servers, Java, Storage, …
  - http://www.spec.org

- SPEC CPU benchmark
  - An industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler
    - Companies have agreed on a set of real program and inputs
    - Valuable indicator of performance (and compiler technology)
  - CPU89 → CPU92 → CPU95 → CPU2000 → CPU2006 → CPU2017
  - Can still be abused

Benchmark Games

An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error…was a sad commentary on a common industry practice of "cheating" on standardized performance tests…The error was pointed out to Intel two days ago by a competitor, Motorola …came in a test known as SPECint92…Intel acknowledged that it had "optimized" its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing…At the heart of Intel's problem is the practice of "tuning" compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code…

Saturday, January 6, 1996 New York Times

SPEC CPU2006

- Elapsed time to execute a selection of programs
  - Negligible I/O, so focuses on CPU performance
- Normalize relative to reference machine
  - Sun's historical "Ultra Enterprise 2" introduced in 1997
  - 296MHz UltraSPARC II processor
- Summarize as geometric mean of performance ratios
  - CINT2006: 12 integer programs written in C and C++
  - CFP2006: 17 FP programs written in Fortran and C/C++

$$\sqrt[n]{\prod_{i=1}^{n} Execution\ time\ ratio_i}$$

SPEC CPU2006 Suites

Integer Benchmarks (CINT2006)			Floating Point Benchmarks (CFP2006)		
perlbench	C	Perl programming language	bwaves	Fortran	Fluid dynamics
bzip2	C	Compression	gamess	Fortran	Quantum chemistry
gcc	C	C compiler	milc	C	Physics: Quantum chromodynamics
mcf	C	Combinatorial optimization	zeusmp	Fortran	Physics / CFD
gobmk	C	Artificial intelligence: Go	gromacs	C/Fortran	Biochemistry / Molecular dynamics
hmmer	C	Search gene sequence	cactusADM	C/Fortran	Physics / General relativity
sjeng	C	Artificial intelligence: Chess	leslie3d	Fortran	Fluid dynamics
libquantum	C	Physics: Quantum computing	namd	C++	Biology / Molecular dynamics
h264ref	C	Video compression	dealII	C++	Finite element analysis
omnetpp	C++	Discrete event simulation	soplex	C++	Linear programming, optimization
astar	C++	Path-finding algorithms	povray	C++	Image ray-tracing
xalancbmk	C++	XML processing	calculix	C/Fortran	Structural mechanics
			GemsFDTD	Fortran	Computational electromagnetics
			tonto	Fortran	Quantum chemistry
			lbm	C	Fluid dynamics
			wrf	C/Fortran	Weather prediction
			sphinx3	C	Speech recognition

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC Power Benchmark

- SPECpower_ssj2008
  - The first industry-standard SPEC benchmark for evaluating the power and performance characteristics of server class computers
  - Initially targets the performance of server-side Java
  - Power consumption of server at different workload levels (0% ~ 100%)
    - Performance: ssj_ops/sec
    - Power: Watts (Joules/sec)

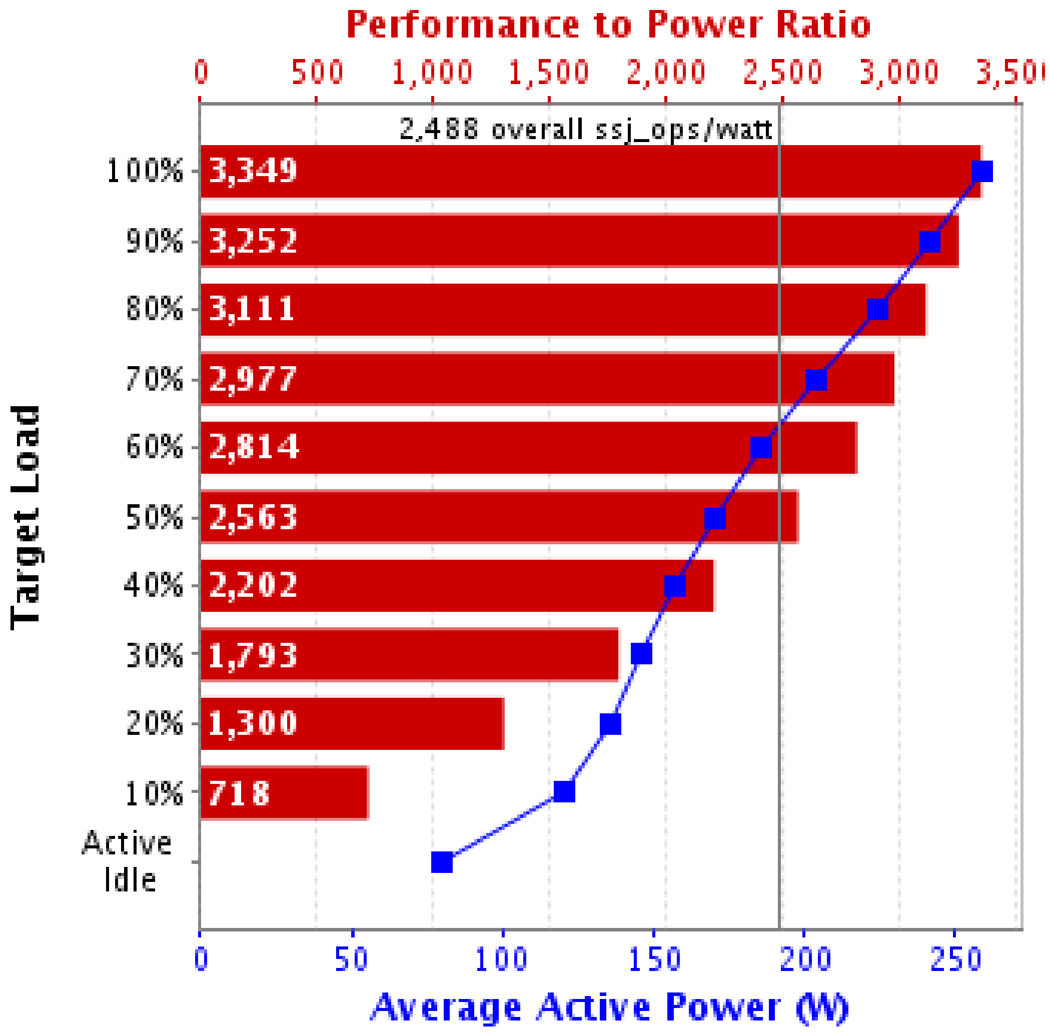
$$Overall\ ssj\_ops\ per\ Watt = \left(\sum_{i=0}^{10} ssj\_ops_i\right) \Big/ \left(\sum_{i=0}^{10} power_i\right)$$

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Fallacy: Low Power at Idle

- Look back at Xeon power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% - 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load



Summary

- Performance is specific to a particular program(s)
  - Total execution time is a consistent summary of the performance

- For a given architecture, performance increases come from
  - Increases in clock rate (without adverse CPI effects)
  - Improvements in processor organization that lower CPI
  - Compiler enhancements that lower CPI and/or instruction count
  - Algorithm/language choices that affect instruction count

- Pitfall: Using a subset of the performance equation as a performance metric