Computer Organization

Lecture 4 - Advanced Performance

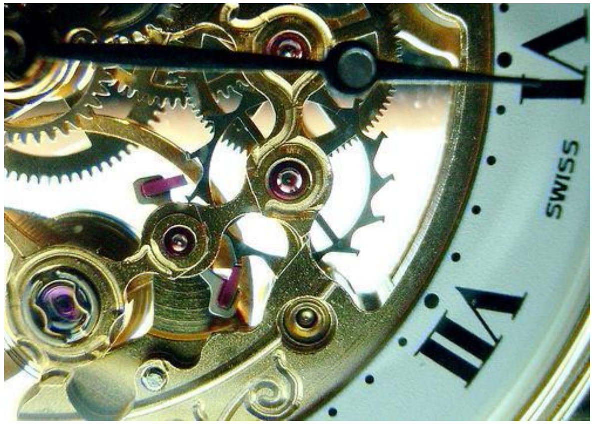


Photo credit:
Jurvetson via flickr

Reading: 1.7-1.9

Homework:

- **Given in uclass**

Roadmap for the term: major topics

- Computer Systems Overview
- Basic Performance
- **Advanced Performance**
- Instruction sets (and Software)
- Logic & Arithmetic
- Processor Implementation
- Memory Systems

Performance Summary (last lecture)

- **The "BIG PICTURE"**

$$\text{CPU time} = \frac{\#\,\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

- **Performance depends on**
  - **Algorithm: affects Instruction Count (IC), possibly CPI**
  - **Programming language: affects IC, CPI**
  - **Compiler: affects IC, CPI**
  - **Instruction set architecture: affects IC, CPI, T_{clock}**
  - **Implementation Technology and Design: affects CPI, T_{clock}**

Performance Outline

- Motivation
- Defining Performance
- Common Performance Metrics
- **Benchmarks**
- Fallacies and Pitfalls (오류와 함정)

Benchmarks : Programs to Evaluate Performance

- The book defines performance in terms of a <u>specific program</u>
- But, which program should you use?
  - Ideally, “real” programs
  - Ideally, programs you will use
  - But what if you don’t know or don’t have time to find out?
- Alternative - Benchmarks

Benchmark Suites

- A collection of small programs
- Summarize performance … how?
  - Total Execution Time
  - Arithmetic Mean
  - Weighted Arithmetic Mean
  - Geometric Mean

Total Execution Time

- **Suppose we have two benchmarks**

	Computer A	Computer B
Program 1 (seconds)	1	10
Program 2 (seconds)	1000	100
Total time (seconds)	1001	110

- **How do we compare computers A and B?**
  - **A is 10 times faster than B for Program 1**
  - **B is 10 times faster than A for Program 2**
- **Summarizing performance: total execution time**

$$\frac{\text{Performance}_B}{\text{Performance}_A} = \frac{\text{Execution time}_A}{\text{Execution time}_B} = \frac{1001}{110} = 9.1$$

  - **B is 9.1 times faster than A**

Think about faireness?

Summarizing Performance

- **Reasonable comparison for even workload**

  Arithmetic Mean $\mathrm{AM} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Time}_i$

  Example: $\mathrm{AM}_A = \frac{1}{2}(1+1000) = 500.5$ $\mathrm{AM}_B = \frac{1}{2}(10+100) = 55$

  $$\frac{Performance_B}{Performance_A} = \frac{500.5}{55} = 9.1$$

  Think about faireness?

- **When some programs run more often than others**

  Weighted Arithmetic Mean $\mathrm{WAM} = \sum_{i=1}^{n} \mathrm{Weight}_i \times \mathrm{Time}_i$

  Example:

  - **Program 1 is 80% of workload** $\mathrm{WAM}_A = 0.8 \times 1 + 0.2 \times 1000 = 200.8$
  - **Program 2 is 20% of workload** $\mathrm{WAM}_B = 0.8 \times 10 + 0.2 \times 100 = 28$

  $$\frac{Performance_B}{Performance_A} = \frac{200.8}{28} = 7.2$$

  Think about faireness?

The SPEC Benchmark Suite

- **System Performance Evaluation Corporation**
  - **Founded by workstation vendors with goal of realistic, standardized performance test**
  - **Philosophy: fair comparison between real systems**
  - **Multiple versions starting with SPEC89**

    most recent is SPECCPU 2006
- **Basic approach**
  - **Measure execution time of several small programs**
  - **Normalize each to performance on a reference machine**
  - **Combine performances using geometric mean**

$$\text{Exec. Time Ratio} = \frac{\text{Exec. Time}_{\text{ref}}}{\text{Exec. Time}_{\text{test}}}$$

$$GM = \sqrt[n]{\prod_{i=1}^{n} \text{Exec.Time Ratio}_i}$$

Why does SPEC use Geometric Mean?

- **“The geometric mean has the interesting property:**
  - **A certain percentage change in any one of the terms has the same effect as the same percentage change in any of the other terms**
  - **Even successive changes in the same term will have the same effect as if the changes were instead spread over other terms.**
  - **What this means in benchmarking terms is that a 10% improvement in one benchmark has the same effect on the overall mean as a 10% improvement on any of the other benchmarks, and that another 10% improvement on that benchmark will have the same effect as the last 10% improvement.**
  - **Thus no one benchmark in a suite becomes more important than any of the others in the suite.”**

SPEC glossary
(http://www.spec.org/spec/glossary/#geometricmean

Some SPEC's Benchmarks

- **Cloud**
  - **IaaS 2018Cloud**
- **SPEC CPU 2017**
  - SPECspeed 2017 Integer
  - SPECspeed 2017 Floating Point
  - SPECrate 2017 Integer
  - SPECrate 2017 Floating Point
  - Optional metric for measuring energy consumption
- **SPEC CPU2006**
  - **CINT2006 - integer performance**
  - **CFP2006 - floating point performance**
- **SPECWEB for webservers**
- **SPECJVM for Java Virtual Machine impl.**
- **SPECviewperf - 3D rendering under OpenGL**
- **SPECpower for power consumption**
- **For results and more info, see:**

`http://www.spec.org`

CINT2006 for an Opteron X4 2356

Name	Description	IC × 10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	725	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	834	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

$$GM = \sqrt[n]{\prod_{i=1}^{n} Exec.Time\ Ratio_i}$$

SPECpower

- **Power consumption of server at different workload levels**
  - **Performance: ssj_ops/sec - ("business operations"/sec)**
  - **Power: Watts (Joules/sec)**

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

ssj: server side java

SPECpower on Opteron X4 2356

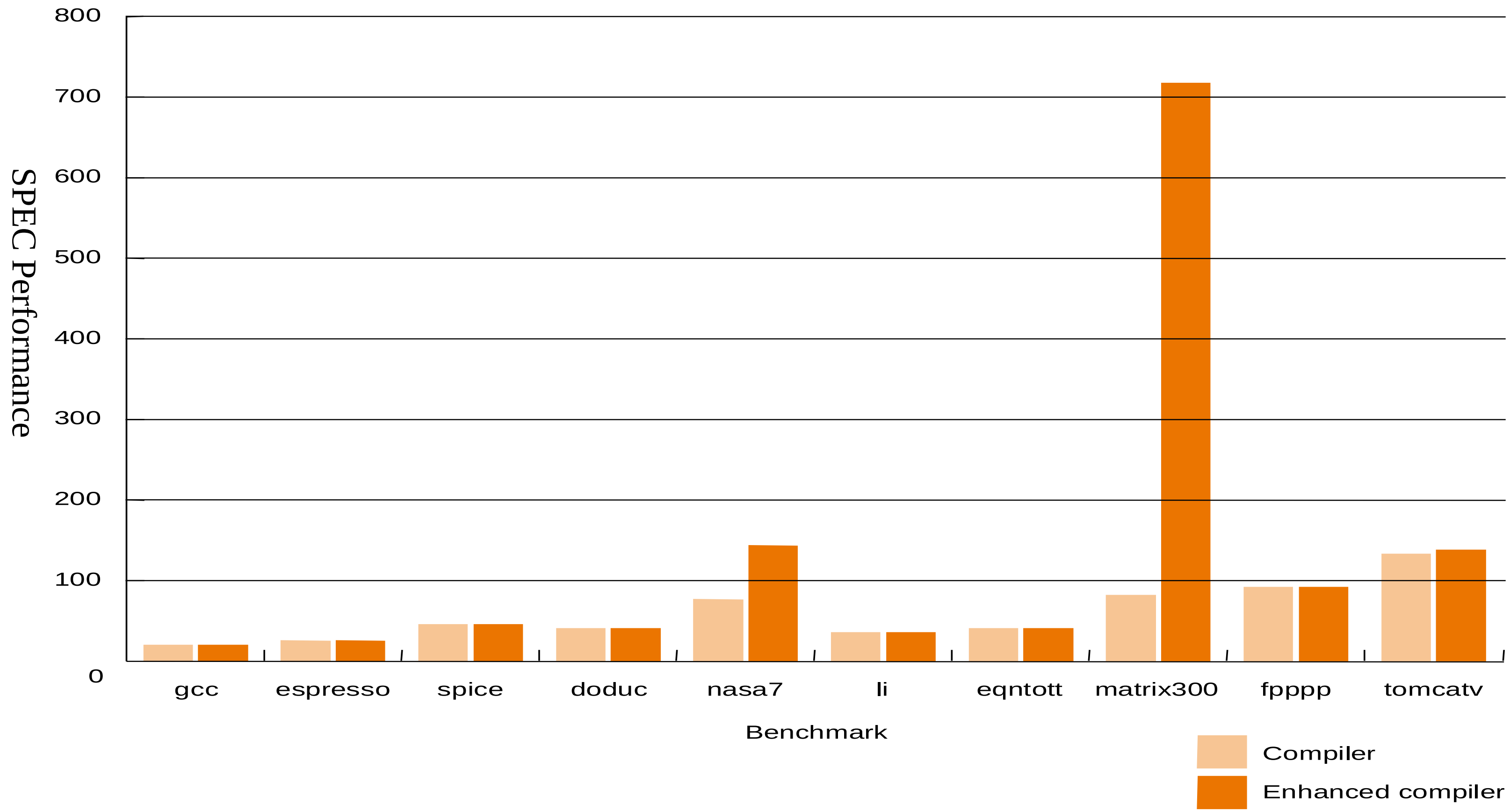
Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	920,35	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
∑ssj_ops/ ∑power		493

Performance Outline

- Motivation
- Defining Performance
- Common Performance Metrics
- Benchmarks
- **Fallacies and Pitfalls (오류와 함정)**

Pitfall (함정): Benchmark Tuning

- **Vendors sometimes focus on making specific benchmarks fast**
- **Example: tuning compiler (Old Figure 2.3)**

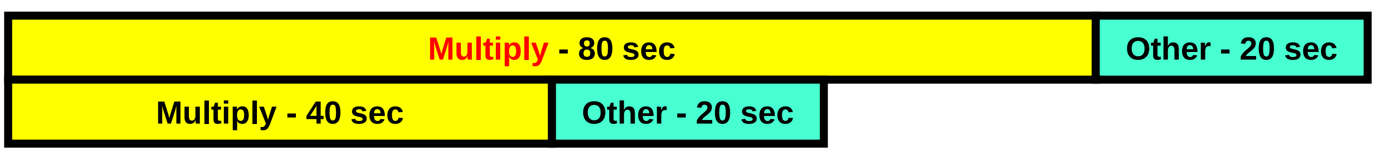


Pitfall: Amdahl's Law

- **Improving one part of performance by a factor of N doesn't increase overall performance by N**

$$\text{Execution time after improvement} = \left(\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution Time Unaffected} \right)$$

- **Book example: Suppose a program executes in 100 seconds where:**
  - **80 seconds are spent performing multiply operations**
  - **20 seconds are spent performing other operations**
- **What happens if we speed up multiply *2* times?**



Amdahl's Law Example (cont'd)

- **Execution time after speeding up multiply:**

$$\text{Execution time after improvement} = \left(\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution Time Unaffected} \right)$$

$$\text{Execution time after improvement} = \left(\frac{80 \text{ seconds}}{n} + 20 \text{ seconds} \right)$$

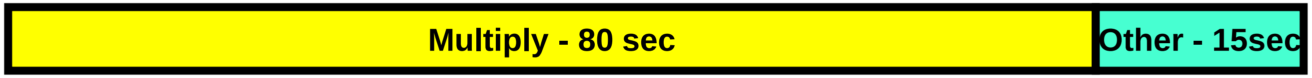
- **Bottom line:**
  - **No matter what we do to multiply, execution time will always be >20 seconds!**
  - **Speedup factor of 5 is not possible!**

Amdahl’s Law Corollary (추론)

- **Make the <u>common case</u> fast**
  - **In our example, biggest gains when we speed up multiply**
  - **Speeding up “other instructions” is not as valuable**

(1) 25% of improvement for multiply

(2) 25% of improvement for other



Fallacy - Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- See Fig. 1-22 (old fig) for additional examples
- Results from Google data center study
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Can processors be designed to make power proportional to load?

Fallacy - MIPS as Performance Metric

- **MIPS - millions of instructions per second**

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

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

- **Once used as a general metric for performance**
  - **But, not useful for comparing different architectures**
  - **Often ridiculed as "meaningless indicator of performance"**

Example - MIPS

대상 프로그램의 실행시간과 무관

Measurement	Computer A	Computer B
Instruction Count	10 billion	8 billion
Clock Rate	4 GHz	4 GHz
CPI	1.0	1.1

- **Which computer has the higher MIPS rating?**

$$MIPS_A = \frac{\text{Clock rate}}{\text{CPI} \times 10^6} = \frac{4 \times 10^9 \text{ cycles/sec}}{1 \text{ cycle/instr.} \times 10^6} = 4{,}000 \text{ MIPS}$$

$$\text{MIPS}_B = \frac{\text{Clock rate}}{\text{CPI} \times 10^6} = \frac{4 \times 10^9 \text{ cycles/sec}}{1.1 \text{ cycles/instr.} \times 10^6} = 3{,}636 \text{ MIPS}$$

Example - MIPS

대상 프로그램의 실행시간을 계산

Measurement	Computer A	Computer B
Instruction Count	10 billion	8 billion
Clock Rate	4 GHz	4 GHz
CPI	1.0	1.1

- **Which Computer is faster?**

학습도우미 1

- **Why we use a benchmark suite to compare the performance of computer systems?**
- **What are the key advantages of the geometric mean in evaluating performance?**
- **What is the pitfall in using benchmarks?**
- **What is the Amdahl's law**
  - **?**
- **What is the corollary from the Amdahl's law?**
  - **Make the common case fast!**
- **Why the MIPS as a performance measure does not work?**

Submit the answers for the above questions