Lecture 14

- Today's lecture:
  - Another look at performance

Performance

- Now we'll discuss issues related to performance:
  - Latency/Response Time/Execution Time vs. Throughput
  - How do you make a reasonable performance comparison?
  - The 3 components of CPU performance
  - The 2 laws of performance

Why know about performance

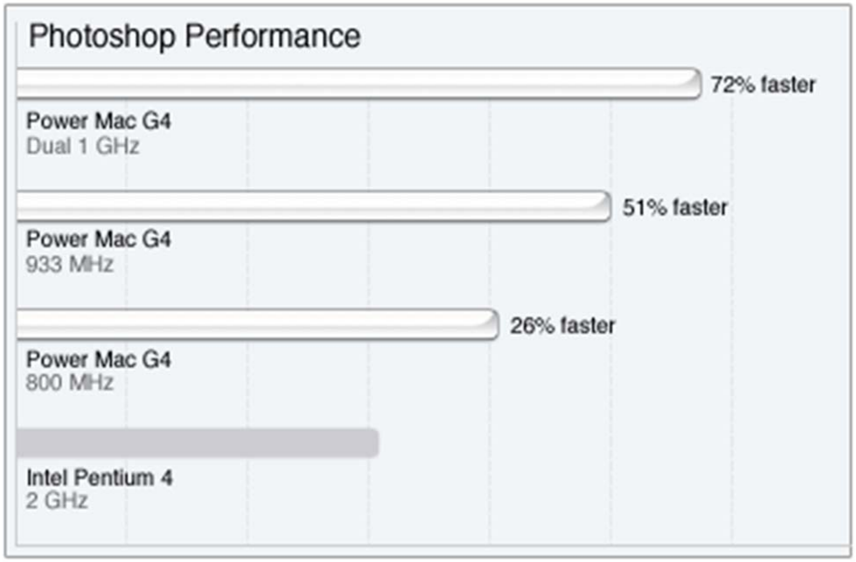
- Purchasing Perspective:
  - Given a collection of machines, which has the
    - Best Performance?
    - Lowest Price?
    - Best Performance/Price?
- Design Perspective:
  - Faced with design options, which has the
    - Best Performance Improvement?
    - Lowest Cost?
    - Best Performance/Cost ?
- Both require
  - Basis for comparison
  - Metric for evaluation

Many possible definitions of performance

- Every computer vendor will select one that makes them look good. How do you make sense of conflicting claims?

Introducing the 2.20 GHz Pentium®4 Processor

Built with Intel's 0.13 micron technology, the new 2.20 GHz Pentium® 4 processor delivers significant performance gains.



AMD

Q: *Why do end users need a new performance metric?*
A: End users who rely only on megahertz as an indicator for performance do not have a complete picture of PC processor performance and may pay the price of missed expectations.

Two notions of performance

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Which has higher performance?
  - Depends on the metric
    - Time to do the task (Execution Time, Latency, Response Time)
    - Tasks per unit time (Throughput, Bandwidth)
  - Response time and throughput are often in opposition

Some Definitions

- Performance is in units of things/unit time
  - E.g., Hamburgers/hour
  - Bigger is better

- If we are primarily concerned with response time
  - $$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

- Relative performance: "X is N times faster than Y"

$$N = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{execution\_time}(Y)}{\text{execution\_time}(X)}$$

Basis of Comparison

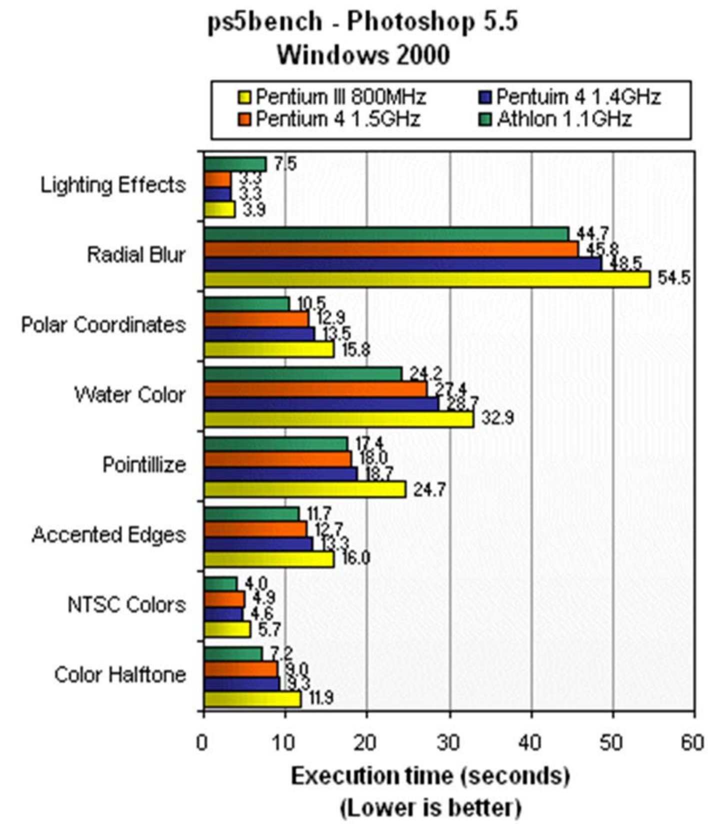
- When comparing systems, need to fix the workload
  - Which workload?

Workload	Pros	Cons
Actual Target Workload	Representative	Very specific Non-portable Difficult to run/measure
Full Application Benchmarks	Portable Widely used Realistic	Less representative
Small "Kernel" or "Synthetic" Benchmarks	Easy to run Useful early in design	Easy to "fool"
Microbenchmarks	Identify peak capability and potential bottlenecks	Real application performance may be much below peak

Benchmarking

- Some common benchmarks include:
  - Adobe Photoshop for image processing
  - BAPCo Sysmark for office applications
  - Unreal Tournament 2003 for 3D games
  - SPEC2000 for CPU performance

- The best way to see how a system performs for a variety of programs is to just show the execution times of all of the programs.
- Here are execution times for several different Photoshop 5.5 tasks, from http://www.tech-report.com

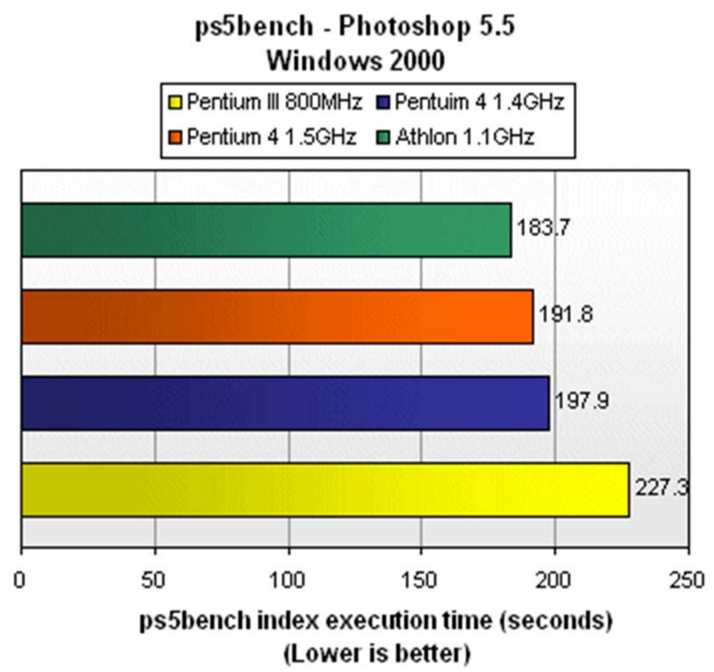


Summarizing performance

- Summarizing performance with a single number can be misleading—just like summarizing four years of school with a single GPA!
- If you must have a single number, you could sum the execution times.

  This example graph displays the total execution time of the individual tests from the previous page.
- A similar option is to find the average of all the execution times.

  For example, the 800MHz Pentium III (in yellow) needed 227.3 seconds to run 21 programs, so its average execution time is 227.3/21 = 10.82 seconds.



- A weighted sum or average is also possible, and lets you emphasize some benchmarks more than others.

The components of execution time

- Execution time can be divided into two parts.
  - User time is spent running the application program itself.
  - System time is when the application calls operating system code.
- The distinction between user and system time is not always clear, especially under different operating systems.
- The Unix time command shows both.

```
salary.125 > time distill 05-examples.ps
Distilling 05-examples.ps (449,119 bytes)
10.8 seconds (0:11)
 449,119 bytes PS  =>  94,999 bytes PDF  (21%)
10.61u 0.98s 0:15.15 76.5%
```

User time

System time

"Wall clock" time (including other processes)

CPU usage = (User + System) / Total

Three Components of CPU Performance

$\text{CPU time}_{X,P} = \text{Instructions executed}_P * \text{CPI}_{X,P} * \text{Clock cycle time}_X$

Cycles Per Instruction

CPI (Review)

- The average number of clock cycles per instruction, or CPI, is a function of the machine and program.
  - The CPI depends on the actual instructions appearing in the program—a floating-point intensive application might have a higher CPI than an integer-based program.
  - It also depends on the CPU implementation. For example, a Pentium can execute the same instructions as an older 80486, but faster.
- Initially we assumed each instruction took one cycle, so we had CPI = 1.
  - The CPI can be >1 due to memory stalls and slow instructions.
  - The CPI can be <1 on machines that execute more than 1 instruction per cycle (superscalar).

Example: Comparing across ISAs

- Intel's Itanium (IA-64) ISA is designed to facilitate executing multiple instructions per cycle. If an Itanium processor achieves an average CPI of .3 (3 instructions per cycle), how much faster is it than a Pentium4 (which uses the x86 ISA) with an average CPI of 1? (assume same freq)

a) Itanium is three times faster
b) Itanium is one third as fast
c) Not enough information

Improving CPI

- Many processor design techniques we'll see improve CPI
  - Often they only improve CPI for certain types of instructions

$$CPI = \sum_{i=1}^{n} CPI_i \times F_i \qquad \text{where } F_i = \frac{I_i}{\text{Instruction Count}}$$

- Fi = Fraction of instructions of type i

- First Law of Performance:

Make the common case fast

Example: CPI improvements

- Base Machine:

Op Type	Freq (fi)	Cycles	CPIi
ALU	50%	3	
Load	20%	5	
Store	10%	3	
Branch	20%	2	

- How much faster would the machine be if:
  - — we added a cache to reduce average load time to 3 cycles?
  - — we added a branch predictor to reduce branch time by 1 cycle?
  - — we could do two ALU operations in parallel?

Amdahl's Law

- Amdahl's Law states that optimizations are limited in their effectiveness.

$$\text{Execution time after improvement} = \frac{\text{Time affected by improvement}}{\text{Amount of improvement}} + \text{Time unaffected by improvement}$$

- For example, doubling the speed of floating-point operations sounds like a great idea. But if only 10% of the program execution time T involves floating-point code, then the overall performance improves by just 5%.

$$\text{Execution time after improvement} = \frac{0.10\,T}{2} + 0.90\,T = 0.95\,T$$

- What is the maximum speedup from improving floating point?
  - Second Law of Performance:

Make the fast case common

Summary

- Performance is one of the most important criteria in judging systems.
- There are two main measurements of performance.
  - Execution time is what we'll focus on.
  - Throughput is important for servers and operating systems.
- Our main performance equation explains how performance depends on several factors related to both hardware and software.

$$\text{CPU time}_{X,P} = \text{Instructions executed}_{P} * \text{CPI}_{X,P} * \text{Clock cycle time}_{X}$$

- It can be hard to measure these factors in real life, but this is a useful guide for comparing systems and designs.
- Amdahl's Law tell us how much improvement we can expect from specific enhancements.
- The best benchmarks are real programs, which are more likely to reflect common instruction mixes.