Computer Architecture
(CSC-3501)
Lecture 27
(1 May 2008)

Seung-Jong Park (Jay)
http://www.csc.lsu.edu/~sjpark

Announcement

Chapter 11 Objectives

- Understand the ways in which computer performance is measured.
- Be able to describe common benchmarks and their limitations.
- Become familiar with factors that contribute to improvements in CPU and disk performance.

11.1 Introduction

- The ideas presented in this chapter will help you to understand various measurements of computer performance.
- You will be able to use these ideas when you are purchasing a large system, or trying to improve the performance of an existing system.
- We will discuss a number of factors that affect system performance, including some tips that you can use to improve the performance of programs.

11.2 The Basic Computer Performance Equation

- The basic computer performance equation has been useful in our discussions of RISC versus CISC:

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- To achieve better performance, RISC machines reduce the number of cycles per instruction, and CISC machines reduce the number of instructions per program.

11.2 The Basic Computer Performance Equation

- In short, using Amdahl's Law we know that we need to make the common case fast.
- So if our system is *CPU bound*, we want to make the CPU faster.
- A *memory bound* system calls for improvements in memory management.
- The performance of an *I/O bound* system will improve with an upgrade to the I/O system.

Of course, fixing a performance problem in one part of the sy stem can expose a weakness in another part of the system!

11.2 The Basic Computer Performance Equation

- We have also learned that CPU efficiency is not the sole factor in overall system performance. Memory and I/O performance are also important.
- Amdahl's Law tells us that the system performance gain realized from the speedup of one component depends not only on the speedup of the component itself, but also on the fraction of work done by the component:

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

11.3 Mathematical Preliminaries

- In comparing the performance of two systems, we measure the time that it takes for each system to do the same amount of work.
- Specifically, if System A and System B run the same program, System A is *n times as fast* as System B if:

$$\frac{\text{running time on system B}}{\text{running time on system A}} = n$$

- System A is x% faster than System B if:

$$\left[\frac{\text{running time on system B}}{\text{running time on system A}} - 1 \right] \times 100\% = x\%$$

11.3 Mathematical Preliminaries

- Suppose we have two racecars that have just completed a 10 mile race. Car A finished in 3 minutes, and Car B finished in 4 minutes. Using our formulas, Car A is 1.25 times as fast as Car B, and Car A is also 25% faster than Car B:

$$\frac{\text{time for Car B to travel 10 miles}}{\text{time for Car A to travel 10 miles}} = \frac{4}{3} = 1.25$$

$$\left[\frac{\text{time for Car B to travel 10 miles}}{\text{time for Car A to travel 10 miles}} - 1 \right] \times 100\% = \left[\frac{4}{3} - 1 \right] \times 100\% = 25\%$$

11.3 Mathematical Preliminaries

- When we are evaluating system performance we are most interested in its expected performance under a given workload.
- We use statistical tools that are *measures of central tendency*.
- The one with which everyone is most familiar is the *arithmetic mean* (or average), given by:

$$\frac{\sum_{i=1}^{n} x_i}{n}$$

11.3 Mathematical Preliminaries

- The arithmetic mean can be misleading if the data are skewed or scattered.
  - Consider the execution times given in the table below. The performance differences are hidden by the simple average.

Program	System A Execution Time	System B Execution Time	System C Execution Time
v	50	100	500
w	200	400	600
x	250	500	500
y	400	800	800
z	5000	4100	3500
Average	1180	1180	1180

11.3 Mathematical Preliminaries

- If execution frequencies (expected workloads) are known, a *weighted average* can be revealing.
  - The weighted average for System A is:
    $50 \times 0.5 + 200 \times 0.3 + 250 \times 0.1 + 400 \times 0.05 + 5000 \times 0.05 = 380$.

Program	Execution Frequency	System A Execution Time	System C Execution Time
v	50%	50	500
w	30%	200	600
x	10%	250	500
y	5%	400	800
z	5%	5000	3500
Weighted Average		380 seconds	695 seconds

11.3 Mathematical Preliminaries

- However, workloads can change over time.
  - A system optimized for one workload may perform poorly when the workload changes, as illustrated below.

Program	Execution Time	Execution Frequency #1	Execution Frequency #2
v	50	50%	25%
w	200	30%	5%
x	250	10%	10%
y	400	5%	5%
z	5000	5%	55%
Weighted Average		380 seconds	2817.5 seconds

11.4 Benchmarking

- Performance benchmarking is the science of making objective assessments concerning the performance of one system over another.
- *Price-performance ratios* can be derived from standard benchmarks.
- The troublesome issue is that there is no definitive benchmark that can tell you which system will run *your* applications the fastest (using the least *wall clock* time) for the least amount of money.

11.4 Benchmarking

- Many people erroneously equate CPU speed with performance.
- Measures of CPU speed include cycle time (MHz, and GHz) and millions of instructions per second (MIPS).
- Saying that System A is faster than System B because System A runs at 1.4GHz and System B runs at 900MHz is valid only when the ISAs of Systems A and B are identical.
  - With different ISAs, it is possible that both of these systems could obtain identical results within the same amount of wall clock time.

11.4 Benchmarking

- In an effort to describe performance independent of clock speed and ISAs, a number of *synthetic benchmarks* have been attempted over the years.
- Synthetic benchmarks are programs that serve no purpose except to produce performance numbers.
- The earliest synthetic benchmarks, *Whetstone, Dhrystone,* and *Linpack* (to name only a few) were relatively small programs that were easy to optimize.
  - This fact limited their usefulness from the outset.
- These programs are much too small to be useful in evaluating the performance of today's systems.

11.4 Benchmarking

- In 1988 the Standard Performance Evaluation Corporation (SPEC) was formed to address the need for objective benchmarks.
- SPEC produces benchmark suites for various classes of computers and computer applications.
- Their most widely known benchmark suite is the SPEC CPU benchmark.
- The SPEC CPU2000 benchmark consists of two parts, CINT2000, which measures integer arithmetic operations, and CFP2000, which measures floating-point processing.

11.4 Benchmarking

- The SPEC benchmarks consist of a collection of kernel programs.
- These are programs that carry out the core processes involved in solving a particular problem.
  - Activities that do not contribute to solving the problem, such as I/O are removed.
- CINT2000 consists of 12 applications (11 written in C and one in C++); CFP2000 consists of 14 applications (6 FORTRAN 77, 4 FORTRAN 90, and 4 C).

A list of these programs can be found in Table 10.7 on Pages 467 - 468.

11.4 Benchmarking

- On most systems, more than two 24 hour days are required to run the SPEC CPU2000 benchmark suite.
- Upon completion, the execution time for each kernel (as reported by the benchmark suite) is divided by the run time for the same kernel on a Sun Ultra 10.
- The final result is the geometric mean of all of the run times.
- Manufacturers may report two sets of numbers: The *peak* and *base* numbers are the results with and without compiler optimization flags, respectively.

11.4 Benchmarking

- The SPEC CPU benchmark evaluates only CPU performance.
- When the performance of the entire system under high transaction loads is a greater concern, the *Transaction Performance Council* (TPC) benchmarks are more suitable.
- The current version of this suite is the TPC-C benchmark.
- TPC-C models the transactions typical of a warehousing and distribution business using terminal emulation software.

11.4 Benchmarking

- The TPC-C metric is the number of new warehouse order transactions per minute *(tpmC*), while a mix of other transactions is concurrently running on the system.
- The tpmC result is divided by the total cost of the configuration tested to give a price-performance ratio.
- The price of the system includes all hardware, software, and maintenance fees that the customer would expect to pay.

11.4 Benchmarking

- The *Transaction Performance Council* has also devised benchmarks for decision support systems (used for applications such as data mining) and for Web-based e-commerce systems.
- For all of the TPC benchmarks, the systems tested must be available for general sale at the time of the test and at the prices cited in a full disclosure report.
- Results of the tests are audited by an independent auditing firm that has been certified by the TPC.

11.4 Benchmarking

- TPC benchmarks are a kind of simulation tool.
- They can be used to optimize system performance under varying conditions that occur rarely under normal conditions.
- Other kinds of simulation tools can be devised to assess performance of an existing system, or to model the performance of systems that do not yet exist.
- One of the greatest challenges in creation of a system simulation tool is in coming up with a realistic workload.

11.4 Benchmarking

- To determine the workload for a particular system component, system traces are sometimes used.
- Traces are gathered by using hardware or software probes that collect detailed information concerning the activity of a component of interest.
- Because of the enormous amount of detailed information collected by probes, they are usually engaged for only a few seconds.
- Several trace runs may be required to obtain statistically useful system information.

11.4 Benchmarking

- Devising a good simulator requires that one keep a clear focus as to the purpose of the simulator
- A model that is too detailed is costly and time-consuming to write.
- Conversely, it is of little use to create a simulator that is so simplistic that it ignores important details of the system being modeled.
- A simulator should be validated to show that it is achieving the goal that it set out to do: A simple simulator is easier to validate than a complex one.

Chapter 11 Conclusion

- Computer performance assessment relies upon measures of central tendency that include the arithmetic mean, weighted arithmetic mean, the geometric mean, and the harmonic mean.
- Each of these is applicable under different circumstances.
- Benchmark suites have been designed to provide objective performance assessment. The most well respected of these are the SPEC and TPC benchmarks.

Chapter 11 Conclusion

- CPU performance depends upon many factors.
- These include pipelining, parallel execution units, integrated floating-point units, and effective branch prediction.
- User code optimization affords the greatest opportunity for performance improvement.
- Code optimization methods include loop manipulation and good algorithm design.