Computer Systems Architecture CMSC 411 Unit 1 – Computer Design and Evaluation

Alan Sussman
August 30, 2004

Administrivia

- Class web pages are at http://www.cs.umd.edu/class/fall2004/cmsc411
  - and linked in from CS dept. class web pages
- Class accounts for project will be on CSIC Linux cluster
- First homework, for Unit 1, announced Thursday, 9/2
- Start reading Ch. 1 of H&P

Introduction

- Why are you taking this course?
  - You really liked the material in 311 and want to learn more?
  - The course time fit into your schedule well?
  - You needed upper level CS courses and chose this one at random?
  - All the courses you really wanted to take were filled?

What can you expect to learn in this course?

- What to look for in buying a PC (brag to parents and friends!)
- How computer architecture affects programming style
- How programming style affect computer architecture
- Variations on interconnection networks
- A great deal of jargon

Syllabus

- More on web page
- Importance of *doing the homework.*
- Lecture notes available on the web - you can print them before coming to class and then add your own notes (I will try to put them up in time)
- I'll correct any typos/errors after each lecture and re-post
- I'll post homework (and exam) answers after the due date – password protected, and you'll get email with the password – account name cmsc411

The Textbook – H&P

- Everyone complains about it
- Virtually everyone uses it
- You can handle it, but you have to work at it – *do the reading*
- Through lecture notes, other references, etc., I'll try to help you put it all together

Chapter 1 of H&P

- Read Chapter 1; you can skip the economics of *cost vs. price*
- Historical Perspective - Section 1.11
  - Computers as we know them are roughly 60 years old
  - The *von Neumann machine* model that underlies computer design is only partially von Neumann's
  - Why does Konrad Zuse say he had ``the bad luck of being too early''?
    - Optional: Read his own recollections in TR 180 of ETH, Zürich, http://www.inf.ethz.ch/research/publications/show.php?what=180&lang=en&type=tr (contains both German and English)
  - No one was able to successfully patent the idea of a stored-program computer, much to the dismay of Eckert and Mauchly

Early development steps

- Make input and output easier than wiring circuit boards and reading lights
- Make programming more easy by developing higher level programming languages, so that users did not need to use binary machine code instructions
- Develop storage devices

Later development steps

- Faster
- More storage
- Cheaper
- Networking and parallel computing
- Better user interfaces
- Ubiquitous applications
- Development of standards

Perspective: An example

- Most powerful computer in 1988: CRAY Y-MP
- 1993: a desktop workstation (IBM Power-2) matched its power at less than 10% of the cost
- How did this happen?
  - hardware improvements, e.g., squeezing more circuits into a smaller area
  - improvements in instruction-set design, e.g., making the machine faster on a small number of frequently used instructions
  - improvements in compilation, e.g., optimizing code to reduce memory accesses and make use of faster machine instructions

Computer Architecture

- instruction set architecture:
  - which elementary instructions are basic to programming the machine
- organization:
  - memory systems to store information,
  - bus systems to move information,
  - logical design of CPU (recall from 311)
- hardware:
  - what physical components are used to form the computer

Why a computer scientist should care

- to choose an architecture that matches a user's needs
- to compare and benchmark different computers
- to decide whether an upgrade is cost-efficient
- to write a program that exploits the strengths of a given computer

Technology is rapidly changing

- From Sections 1.3 & 1.4
  - The average program hogs twice as much memory each year
    - Example: Microsoft
  - Transistor density increases by 50% each year
  - Dynamic random access memory (DRAM) density increases by 60% each year
  - Disk density increases by 50% each year
  - Lifespan of a computer is about 5 years
  - Cost of DRAM drops by about 40% each year

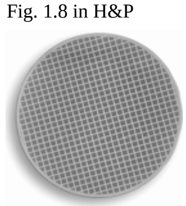
Costs

- From Figure 1.9 in H&P
- The cost of components in a $1000 PC in 2001 are:
  - CPU – 22%
  - Monitor – 19%
  - Hard drive – only 9%
  - DRAM – only 5% (for 128MB)
  - Software – 20% (OS & basic office suite)

Manufacture of DRAM and other chips

- Chips are manufactured on **wafers** - circular disks containing many dies (chips).
- The wafer is tested and chopped into **dies**.

Fig. 1.8 in H&P



Wafers and dies

- To find the cost of a die:
  - Number of dies per wafer is *at most* the area of the wafer divided by the area of the die.
  - The cost of the wafer divided by the number of **working** dies per wafer is the cost of each die.
- The fraction of working dies is called the *die yield*, which decreases as the area of the die increases.
- Rule of thumb (p. 20): Cost of die is proportional to the square of the die area

Computer Systems Architecture CMSC 411 Unit 1 – Computer Design and Evaluation

Alan Sussman
September 2, 2004

Administrivia

- First homework, for Unit 1, posted
  - due Sept. 14

Last time

- Technology trends
  - smaller, better, faster, cheaper
  - processor, memory, disks, networks, etc.
- Wafer yields
  - Rule of thumb is that cost of die (chip) is proportional to the square of the die area

Comparing performance of two machines

- Definition: **Performance** is equal to 1 divided by execution time
- Problem: How to measure execution time?

What is time?

- Unix time command example (p.26):
  - 90.7u 12.9s 2:39 65%
  - The user used the CPU for 90.7 seconds (user CPU time)
  - The system used it for 12.9 seconds (system CPU time)
  - Elapsed time from the user's request to completion of the task was 2 minutes, 39 seconds (159 seconds)
  - And (90.7 + 12.9)/159 = 65%
    - the rest of the time was spent waiting for I/O or running other programs

Time (cont.)

- Usual measurements of time:
  - **system performance** measures the elapsed time on unloaded (single user) system
  - **CPU performance** measures user CPU time on unloaded system

How to measure CPU performance

- **Benchmark**: a program used to measure performance
  - real programs - what is reality?
  - kernels - loops in which most of time is spent in a real program
  - toy programs
  - synthetic programs
- Fact: Computer manufacturers tune their product to the popular benchmarks
  - ``Your results may vary,'' unless you run benchmark programs and nothing else
  - See Figure 1.12 listing programs in the SPEC CPU2000 benchmark suite

Reproducibility

- Benchmarking is a laboratory experiment, and needs to be documented as fully as a well-run chemistry experiment
  - Identify each variable hardware component
  - Identify compiler flags and measure variability
  - Verify reproducibility and provide data for others to reproduce the benchmarking results

Reporting results

- Example of parameters for SPEC CINT2000 benchmark is in H&P Figure 1.14
  - for a Dell workstation running Windows NT4.0
  - data on hardware - CPU (how many, primary and secondary caches), memory, disk
  - data on software – OS, compilers, file system type, and compiler flags used (very important!)
- How to summarize results?

Sample results

From Figure 1.15 in H&P – which machine is fastest?

	Computer A	Computer B	Computer C
Program P1 (sec)	1	10	20
Program P2 (sec)	1000	100	20
Program P3 (sec)	1001	110	40

Statistical reporting

- "It is rare indeed when advertisers, politicians, pop economists, and drumbeaters for medical programs offer a statistical argument that is not either misleading or downright deceptive." - Martin Gardner
- "... in mathematics you don't understand things, you just get used to them." - John von Neumann
- "... if you torture your data long enough, they will tell you what you want to hear." - James L. Mills, M.D.

Statistical reporting (cont.)

- The **average** execution time is the **arithmetic mean**:
  - sum the execution times for each program and divide by the number of programs *n*
- The **harmonic mean** is *n* divided by the sum of some function of each execution time
- Often, both of these means are modified to give more weight to more important programs

Statistical reporting (cont.)

- Arithmetic mean is good if the weights represent your own workload - then you can compare how each machine will perform for you
  - *Don't* set the weights based on performance on any particular machine; Figure 1.16 illustrates confusing results obtained by doing that
- Harmonic mean gives same sort of information, but is used when rates (e.g., instructions per second) are measured instead of times

Geometric mean

- To measure relative performance, use the **geometric mean**
  - Choose a *reference machine*, divide all execution times by the corresponding times on the reference machine, multiply those ratios together, and take the *n*th root of the product
- Geometric means have the nice property that you get consistent results regardless of which machine is used to normalize (Figure 1.17)

How to make computers faster

- **Make the common case faster!**
- Example: Put more effort and funds into optimizing the hardware for addition than to optimize square root
- **Amdahl's law** quantifies this principle:
  - Define **speedup** as the time the task took originally divided by the time the task takes after improvement

Amdahl's Law

- Then Amdahl tells us what the speedup of a particular task is, given
  - fraction f of the original execution time that the task could use the improvement
  - the speedup s of a task that always uses the improvement
- Then what is the speedup of the task?

Amdahl's Law (cont.)

- Suppose that the original task runs for 1 second so it takes f seconds in the critical piece, and $1-f$ in other things
- Then the task on the improved machine will take only f/s seconds in the critical piece, but will still take $1-f$ seconds in other things
- speedup = $\frac{old\_time}{new\_time} = \frac{1}{(1-f)+\frac{f}{s}}$

Example 1

- Suppose we work very hard improving the square root hardware, and that our task originally spends *1%* of its time doing square roots. Even if the improvement reduces the square root time to zero, the speedup is no better than
- speedup = $\frac{1}{1-f} = \frac{1}{.99} = 1.01$
- *And we might be better off putting our effort into a more important part of the task*

Example 2

- Suppose that, for the same cost, we can speed up integer arithmetic by a factor of 20, or speed up floating point arithmetic by a factor of 2. If our task spends *10%* of its time in integer arithmetic, and *40%* of its time in floating point arithmetic, which should we do?

Example 2 (cont.)

- Option 1:
  speedup = $\frac{1}{.9+\frac{.1}{20}} = 1.105$
- Option 2:
  speedup = $\frac{1}{.6+\frac{.4}{2}} = 1.25$

CPU Performance

- More jargon …
- Something new happens in a computer during every **clock cycle**
- The clock rate is what manufacturers usually advertise to indicate a chip's speed:
  - e.g., a 2.8GHz Pentium 4
- But how fast the machine is depends on the clock rate *and* the number of clock cycles per instruction (CPI)

CPU Performance (cont.)

- So total CPU time = instruction count (IC) times CPI divided by clock rate (MHz/GHz)
  - IC * CPI / MHz
- There's an example on p. 44 that illustrates the overall effect
- Note: CPI is not a good measure of performance all by itself since it varies by type of instruction!

How to design a fast computer

- Amdahl's law says put your effort into optimizing instructions that are often used.
- **The locality of reference** rule-of-thumb says that a program spends *90%* of its time in *10%* of the code, so we should put effort into taking advantage of this, through a **memory hierarchy**
- For data accesses, 2 types of locality
  - temporal
  - spatial

Take advantage of parallelism

- At system level
  - e.g., use multiple CPUs (Unit 7), disks (Unit 6)
- At processor level
  - among instructions (Unit 4)
  - pipelining (Unit 3)
- At digital design level
  - e.g., set associative caches (Unit 5), carry-lookahead in ALU design

Computer Systems Architecture CMSC 411 Unit 1 – Computer Design and Evaluation

Alan Sussman
September 7, 2004

Administrivia

- Unit 1 homework due in 1 week, Sept. 14
- Start reading Chapter 2

Last time

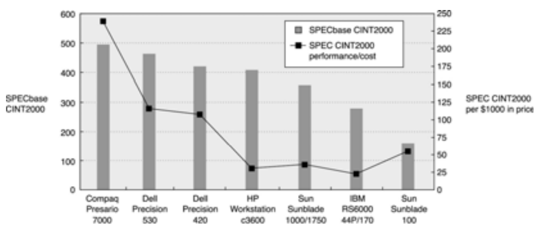
- Benchmark programs
  - use your own programs, or ones that have similar characteristics to what you want to run
- Use statistical methods to aggregate performance results
  - arithmetic mean for time, harmonic for rates, geometric for relative performance
- Speedup = old time/new time
  - how long does a program take after a hardware improvement that gives a speedup of 5?
- Amdahl's law – speedup = $1/((1-f) + f/s)$
  - what are f and s?
- CPU equation – time = IC*CPI/MHz

Performance/Price Performance

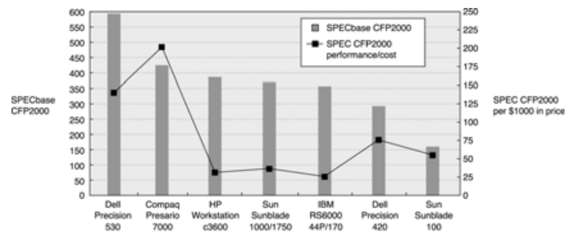
For desktop systems (H&P Figure 1.18) – 512MB ECC SDRAM, July 2001

Model	Processor	Clock rate (MHz)	Price
Compaq 7000	AMD Athlon	1400	$2091
Dell 420	Pentium III	1000	$3834
Dell 530	Pentium 4	1700	$4175
HP c3600	PA 8600	552	$12,631
IBM RS6000 44P/170	IBM Power III-2	450	$13,889
Sunblade 100	UltraSPARC II-e	500	$2950
Sunblade 1000	UltraSPARC III	750	$9950

SPEC CINT2000 – H&P 1.19



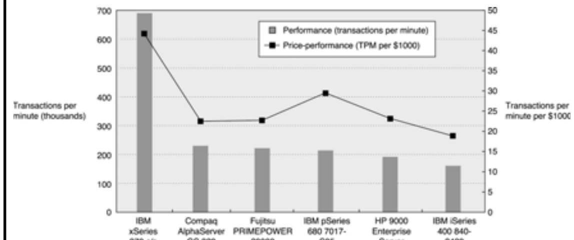
SPEC CFP2000 – H&P 1.20



Performance/Price Performance

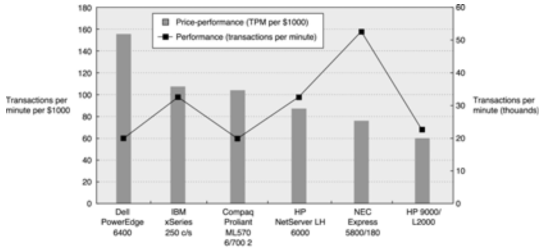
- For OLTP systems
- See Figure 1.21 (and www.tpc.org) for system details
  - both high total performance, and high price performance systems
- No uniprocessor systems
  - one cluster (IBM with 280 Pentium IIIs)
  - others multiprocessors (SMPs)

High performing OLTP systems



H&P Figure 1.22

High price/performance systems



H&P Figure 1.23

Fallacies

- MIPS (Millions of Instructions Per Second) or MFLOPS (Millions of Floating Point Operations) is a good way to compare performance
- **Fallacy**: One computer may have more powerful instructions than another, and MIPS/MFLOPS depends on the instruction mix that you choose

Fallacies (cont.)

- Relative performance of 2 processors with same ISA can be judged by clock rate or by single benchmark suite
- **Fallacy**: other factors matter too, e.g., pipeline structure, memory system

Fallacies (cont.)

- Synthetic benchmarks predict performance of real programs
- **Fallacy**: they don't reflect program behavior for factors not measured, and are very susceptible to compiler and hardware optimizations that may not work on real programs

Fallacies (cont.)

- Benchmarks remain valid indefinitely
- **Fallacy**: Needs change, hardware changes, languages change, compilers change (esp. better optimization), ...

Fallacies (cont.)

- Peak performance tracks observed performance
- **Fallacy**: Peak performance is a speed limit, the *guaranteed-not-to-be-exceeded* performance, and virtually never seen in practice

Pitfalls

- Comparing hand-coded assembly and compiler-generated high-level language performance

- **Pitfall**: hand-coding most applications not possible – because of high software development costs – even though it is possible for key loops, performance critical kernels

CMSC 411 - Alan Sussman 55

Pitfalls (cont.)

- Neglecting software costs in evaluating a system or for cost-performance analysis

- **Pitfall**: software costs can be a large fraction of both purchase cost and operational (maintenance) costs of a system
  - e.g., Microsoft Windows + Office costs ~$300 for a PC system with hardware costing ~$1000, so software costs are ~25% of the total cost (and higher if not bundled with the hardware)

CMSC 411 - Alan Sussman 56