G22.2243-001
High Performance Computer Architecture

Instructor: Vijay Karamcheti
vijayk@cs.nyu.edu

Lecture: Wednesdays, 5:00pm-7:00pm
102 CIWW

Office Hours: Tuesdays, 4:00pm-5:00pm
715 Broadway, Room 704
phone: 8-3496

Mailing List: g22_2243_001_fa02@cs.nyu.edu
Outline

• Introduction
• Administrative stuff
  – course organization
  – workload and grading
• SimpleScalar toolset

• Fundamentals of computer design
  – Target markets
  – Technology trends
  – Cost vs. price
  – Measuring and reporting performance
  – Quantitative principles of computer design
  – Price performance

[ Hennessy/Patterson CA:AQA (3rd Edition): Chapter 1 ]
Course Objective

- The course objective is to gain the knowledge required to design and analyze high-performance computer systems.
Computer Architecture Topics

Instruction Set Architecture
- Pipelining, Hazard Resolution, Superscalar, Reordering, ILP, Branch Prediction, Speculation

Memory Hierarchy
- DRAM
- L2 Cache
- L1 Cache
- Coherence, Bandwidth, Latency
- Cache Design, Block size, Associativity
- Addressing modes, formats

Input/Output and Storage
- Disks and Tape
- RAID
- Emerging Technologies, Interleaving

Memory
- DRAM

Processor Design

VLSI
Computer Architecture Topics

Networks, Interconnections, and Multiprocessors

Interconnection Network

Topologies, Routing, Bandwidth, Latency, Reliability

Shared Memory or Message Passing

Network Interfaces
### Growth in Microprocessor Performance

<table>
<thead>
<tr>
<th>Type</th>
<th>Pentium 3 (Coppermine)</th>
<th>Cray YMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2000</td>
<td>1988</td>
</tr>
<tr>
<td>Clock</td>
<td>1130 MHz</td>
<td>167 MHz</td>
</tr>
<tr>
<td>MIPS</td>
<td>&gt; 1000 MIPS</td>
<td>&lt; 50 MIPS</td>
</tr>
<tr>
<td>Cost</td>
<td>$2,000</td>
<td>$1,000,000</td>
</tr>
<tr>
<td>Cache</td>
<td>256 KB</td>
<td>0.25 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>512 MB</td>
<td>256 MB</td>
</tr>
</tbody>
</table>

**Legend:**
- Intel Pentium III
- Alpha

**Graph:**
- Relative performance over years:
  - 1.35x per year growth
Reasons for this Performance Improvement

- Technology
  - More transistors per chip
  - Faster logic
- Machine Organization/Implementation
  - Deeper pipelines
  - More instructions executed in parallel
- Instruction Set Architecture
  - Reduced Instruction Set Computers (RISC)
  - Multimedia extensions
  - Explicit parallelism
- Compiler technology
  - Finding more parallelism in code
  - Greater levels of optimization
From the Intel 386 to the Pentium III

Intel 386, introduced 1985
275,000 transistors, 1 micron
16 MHz clock speed

Intel Pentium III, introduced 1999
9.5M transistors, 0.25 micron
600 MHz clock speed
Intel Pentium III Microarchitecture

IFU: Instruction fetch unit
ID: Instruction dispatch
MIS: Micro-instruction sequencer
BTB: Branch target buffer
RAT: Register alias table
RS: Reservation station
IEU: Int. execution unit
FEU: FP execution unit
DCU: Data cache unit
L2 cache
What is Ahead?

- Today’s desktop microprocessors (e.g., Pentium 4)
  - 42 million transistors, 0.13 micron technology, 2.8 GHz clock speed
  - Internally: “hyper-pipelining”, multithreading, 128-bit SIMD instructions

- The future
  - Greater instruction level parallelism
  - Bigger caches, and more levels of cache
  - Multiple processors per chip
    - Complete systems on a chip
  - High performance interconnects
  - New application characteristics
    - Multimedia, new interface technologies (video, speech …)
  - Breakdown into desktop, enterprise, and embedded target markets
    - Different performance criteria

- This course provides the background for you to design, analyze, and effectively use such systems
Topic Coverage

- Textbook
  - Hennessy and Patterson,

- Fundamentals of Computer Design (Chapter 1) 1 lecture
- Instruction Set Architecture (Chapter 2) 0.5 lectures
- Pipelining Basics (Appendix A) 2 lectures
- Instruction Level Parallelism (Chapters 3 and 4) 5 lectures
- Memory Hierarchy (Chapter 5) 2 lectures
- Multiprocessors (Chapter 6) 1.5 lectures
- Input/Output and Storage (Chapter 7) 1 lecture
- Networks and Interconnection Technology (Chapter 8)

- Plus, relevant computer architecture research papers.
Course Workload

- Lectures (sometimes may be given by a guest lecturer)
  - Reading assignments from text

- Five programming assignments (60% of course grade)
  Build a simulator for a **multiple-issue, out-of-order execution** microprocessor in stages (Pentium III class)
  - Assignment 1: Pipelining
  - Assignment 2: Branch prediction
  - Assignment 3: Reservation stations (Multiple functional units)
  - Assignment 4: Multiple issue, out-of-order execution
  - Assignment 5: Cache hierarchies
  Use this simulator to analyze impact of architectural techniques

- Final exam (40% of course grade)
  - Sample questions will be provided in class (end of chapter exercises)
  - Turn-in is optional
SimpleScalar Toolset

- Comprehensive collection of tools for evaluating new architectural techniques
  - Possible to define new instruction-set architectures (support for Alpha, PISA)
  - Modules for writing own execution-driven simulators
    - bpred, caches, statistics collection, program loading, functional unit construction, …
  - More than 50% of papers at recent architecture conferences use SimpleScalar
SimpleScalar Toolset (cont’d)

- For the course assignments we will be using a small subset of the tools
  - You will be using an instructional ISA called PISA
    - Closely resembles MIPS 64 (described in the textbook)
    - PISA executables produced using GNU cross-compiler tools

- Goal of the assignments: To understand the issues involved in implementing and to assess the potential benefits from architectural techniques used in modern-day microprocessors
  - E.g., Branch prediction in modern-day microprocessors
    - At what stage during instruction execution is branch prediction used?
      - It takes some time to figure out that an instruction is a branch
    - How should the branch predictor be updated with information about seen branches?
    - What impact does prediction have on performance?
Background Survey

• Programming in C/C++
  – Required for using the SimpleScalar toolset

• Use of Unix systems
  – SimpleScalar installs available for SPARC/Solaris and x86/Linux

• Prior coursework
  – Logic design:
    • bits, gates, combinational and sequential logic
    • Adders, multipliers
  – Computer organization and assembly-level programming
    • ALUs, MIPS-like data path (without pipelining)
    • Register versus memory operations
    • Buses, I/O
Fundamentals of Computer Design
Context for Designing New Architectures

What is the target market?

• Desktop computing
  – General-purpose applications
  – Performance improvements must be traded off against cost
  – Performance metric of interest is usually response time

• Enterprise servers
  – Cost is important but not as much of a concern
  – Performance is paramount, but metric of interest is usually throughput

• Embedded computers
  – Cost is paramount
  – Power concerns
  – Specialized applications
Designing New Architectures: Other Considerations

- Application Area
  - Special Purpose / General Purpose
  - Scientific (FP intensive) / Commercial (Integer Intensive)

- Level of Software Compatibility Required
  - Object Code / Binary Compatible
  - Assembly Language
  - Programming Language

- Operating System Requirements
  - Size of Address Space
  - Memory Management / Protection
  - Context Switch Capability
  - Interrupts and Traps

- Standards:
  - IEEE 754 Floating Point
  - I/O Bus
  - Networks

- Technology Improvements
  - Increased Processor Performance
  - Larger Memory and I/O devices
  - Software/Compiler Innovations
Technology Trends

- Helps an architect plan for the evolution of an architecture

Four implementation technologies of interest

- **Integrated circuit logic**
  - Transistor density: increases by ~35% per year
  - Die size: increases by ~10-20% per year
  - Transistor count/die: increases by ~55% per year

- **Semiconductor DRAM**
  - Capacity increases by ~40-60% per year
  - Cycle time has not decreased as much: ~33% over 10 years
  - Bandwidth has increased: about ~66% more over 10 years
    - also, changes to the interface have helped further improve bandwidth

- **Magnetic disk technology**
  - Recently, capacity improving by ~100% every year

- **Network technology**
  - More improvements in bandwidth, less in latency
Technology Trends: Microprocessor Capacity

- Alpha 21264: 15 million
- Alpha 21164: 9.3 million
- PowerPC 620: 6.9 million
- Pentium Pro: 5.5 million
- Sparc Ultra: 5.2 million
Memory Capacity (Single Chip DRAM)

<table>
<thead>
<tr>
<th>Year</th>
<th>Size</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64 Kb</td>
<td>250 ns</td>
</tr>
<tr>
<td>1983</td>
<td>256 Kb</td>
<td>220 ns</td>
</tr>
<tr>
<td>1986</td>
<td>1 Mb</td>
<td>190 ns</td>
</tr>
<tr>
<td>1989</td>
<td>4 Mb</td>
<td>165 ns</td>
</tr>
<tr>
<td>1992</td>
<td>16 Mb</td>
<td>145 ns</td>
</tr>
<tr>
<td>1996</td>
<td>64 Mb</td>
<td>125 ns</td>
</tr>
<tr>
<td>2000</td>
<td>256 Mb</td>
<td>100 ns</td>
</tr>
</tbody>
</table>
Transistor Performance, Wires, and Power

- Driving factor in integrated circuit technology is **feature size**
  - Decreased from 10 microns (1971) to 0.13 microns (2002)
- Transistor count/density improves **quadratically** with feature size
  - But, performance only increases **linearly**
    - Need to reduce operating voltage to ensure correct operation at small sizes
- Unfortunately, wires do **not** improve in performance with decreasing feature size
  - Signal delay proportional to product of resistance and capacitance
  - Both increase as feature size decreases
  - Modern-day processors spend large fraction of time cycle just propagating signals between different portions of the chip
    - 2 stages of the Pentium 4 pipeline dedicated for this
- Power requirements grow dramatically with decreasing feature size
  - Pentium 4 consumes ~100 watts
  - Likely to become future bottleneck
Cost vs. Price

• Price: How much the finished good sells for
• Cost: Amount spent to produce it (including overhead)

• Price affected by
  – time (maturity of process)
  – volume (amortization of non-recurring costs)
  – commoditization (competition)

• Cost affected by
  – Yield
  – Direct costs (labor, purchasing, scrap, warranty)
  – Gross margin (R&D, marketing, sales, maintenance …)

• Read Section 1.4 for additional details
Measurement and Evaluation

- Architecture is an iterative process:
  - Search the possible design space
  - Make selections
  - Evaluate the selections made
- Good measurement tools are required to accurately evaluate the selection
- The book argues for what has now become widely accepted: A quantitative approach for evaluating selections
  - most accurate measure of performance is the execution time of representative real programs (benchmarks)
Performance Factors

CPU time = Seconds = Instructions x Cycles x Seconds
Program      Program     Instruction    Cycle

- Instruction Count (IC): Number of instructions/program
- Cycles per instruction (CPI)
  - Sometimes the reciprocal is used: Instructions per cycle (IPC)
- The number of seconds per cycle is the clock period
  - clock rate is the multiplicative inverse of the clock period
## Aspects of CPU Performance

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Instr. Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr. Set</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Comparing CPU Time

- A 500 MHz Pentium III processor takes 2 ms to run a program with 200,000 instructions.
- A 300 MHz UltraSparc processor takes 1.8 ms to run the same program with 230,000 instructions.

- Which processor is faster and by how much?
  - The UltraSparc is $\frac{2}{1.8} = 1.11$ times as fast, or 11% faster.

- What is the CPI for each processor for this program?
  - $\text{CPI} = \frac{\text{Cycles}}{\text{Instruction Count}}$
  - $\text{CPI}_{\text{Pentium}} = 2 \times 10^{-3} \times 500 \times 10^6 / 2 \times 10^5 = 5.00$
  - $\text{CPI}_{\text{SPARC}} = 1.8 \times 10^{-3} \times 300 \times 10^6 / 2.3 \times 10^5 = 2.35$
Cycles Per Instruction

“Average Cycles per Instruction”

\[
\text{CPU time} = \text{Cycle Time} \times \sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i
\]

“Instruction Frequency”

\[
\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i \times F_i \quad \text{where} \quad F_i = \frac{\text{IC}_i}{\text{Instruction Count}}
\]

- Implication: Invest resources where time is spent!
Example: Calculating CPI

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>$F_i \times CPI_i$</th>
<th>(% Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>0.5</td>
<td>(33%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>(27%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>0.2</td>
<td>(13%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>(27%)</td>
</tr>
</tbody>
</table>

Typical Mix

1.5
Example (cont’d)

- Add register / memory ALU operations:
  - One source operand in memory
  - One source operand in register
  - Cycle count of 2
- Branch cycle count to increase to 3
- What fraction of the loads must be eliminated for this to pay off, assuming the clock rate is not affected?

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5</td>
<td>.5 – X</td>
<td>1</td>
<td>.5 – X</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.4</td>
<td>.2 – X</td>
<td>2</td>
<td>.4 – 2X</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.2</td>
<td>.1</td>
<td>2</td>
<td>.2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.3</td>
<td>.2</td>
<td>3</td>
<td>.6</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>1.00</td>
<td>1.5</td>
<td></td>
<td>1 – X</td>
<td></td>
<td>(1.7 – X)/(1 – X)</td>
</tr>
</tbody>
</table>

What fraction of the loads must be eliminated for this to pay off, assuming the clock rate is not affected?
Programs to Evaluate Processor Performance

- **(Toy) Benchmarks**
  - 10-100 line program
  - e.g.: sieve, puzzle, quicksort
- **Synthetic Benchmarks**
  - Attempt to match average frequencies of real workloads
  - e.g., Whetstone, dhrystone
- **Kernels**
  - Time critical excerpts

- **Real Benchmarks**
  - Ideal: exactly the programs one would like to run on the architecture
  - More generally: a collection of programs that possess the characteristics of the real workload
SPEC: System Performance Evaluation Cooperative

- First Round SPEC CPU89
  - 10 programs yielding a single number

- Second Round SPEC CPU92
  - SPEC CINT92 (6 integer programs) and SPEC CFP92 (14 floating point programs)
  - Compiler flags can be set differently for different programs

- Third Round SPEC CPU95
  - new set of programs: SPEC CINT95 (8 integer programs) and SPEC CFP95 (10 floating point)
  - Single flag setting for all programs

- Fourth Round SPEC CPU2000
  - new set of programs: SPEC CINT2000 (12 integer programs) and SPEC CFP2000 (14 floating point)
  - Single flag setting for all programs
  - Programs in C, C++, Fortran 77, and Fortran 90
  - Report both baseline and best performance
SPEC 2000

- 12 integer programs:
  - 2 Compression
  - 2 Circuit Placement and Routing
  - C Programming Language Compiler
  - Combinatorial Optimization
  - Chess, Word Processing
  - Computer Visualization
  - PERL Programming Language
  - Group Theory Interpreter
  - Object-oriented Database.

- Written in C (11) and C++ (1)

- 14 floating point programs:
  - Quantum Physics
  - Shallow Water Modeling
  - Multi-grid Solver
  - 3D Potential Field
  - Parabolic / Elliptic PDEs
  - 3-D Graphics Library
  - Computational Fluid Dynamics
  - Image Recognition
  - Seismic Wave Simulation
  - Image Processing
  - Computational Chemistry
  - Number Theory / Primality Testing
  - Finite-element Crash Simulation
  - High Energy Nuclear Physics
  - Pollutant Distribution

- Written in Fortran (10) and C (4)
Other SPEC Benchmarks

• **JVM98:**
  - Measures performance of Java Virtual Machines
• **SFS97:**
  - Measures performance of network file server (NFS) protocols
• **Web99:**
  - Measures performance of World Wide Web applications
• **HPC96:**
  - Measures performance of large, industrial applications
• **APC, MEDIA, OPC**
  - Measures performance of graphics applications
• For more information about the SPEC benchmarks see: http://www.spec.org.
Implication of the Quantitative Approach

• Make the common case fast
  – One of the principles behind RISC: Reduced Instruction Set Computers
    • Identify most frequently-used instructions
      – Implement them in hardware
    • Emulate other instructions (slowly) in software
      – Pretty much every technique used in current-day microprocessors

• Is there a way of quantifying the gains one is likely to see by improving some portion of the design?
  – What is the best one can hope to do?

• General principle for the above: Amdahl’s Law
  (sometimes also called Amdahl’s curse)
Amdahl's Law

- Speedup due to enhancement E:

\[
\text{Speedup (E)} = \frac{\text{Execution time without E}}{\text{Execution time with E}} = \frac{\text{Performance with E}}{\text{Performance without E}}
\]

- Suppose that enhancement E accelerates a fraction \( f \) of the task by a factor \( s \), and the remainder of the task is unaffected

- New execution time and the overall speedup?

\[
\text{Exec. time}_{\text{new}} = \text{Exec. time}_{\text{old}} \times [(1 - f) + f/s]
\]

\[
\text{Speedup (E)} = \frac{\text{Exec. Time}_{\text{old}}}{\text{Exec. Time}_{\text{new}}} = \frac{1}{[1 - f + f/s]} \leq \frac{1}{(1 - f)}
\]
Example of Amdahl’s Law

- Floating point instructions improved to run 2x; but only 10% of the time was spent on these instructions
- How much improvement in performance should one expect?

\[
\text{Exec. time}_{\text{new}} = \text{Exec. time}_{\text{old}} \times [ (1 - f) + f/s ]
\]

\[
\text{Speedup (E)} = \frac{\text{Exec. Time}_{\text{old}}}{\text{Exec. Time}_{\text{new}}} = \frac{1}{[1 - f + f/s]} \leq \frac{1}{(1 - f)}
\]

\[
\text{Exec. time}_{\text{new}} = \text{Exec. time}_{\text{old}} \times [ (1 - 0.1) + 0.1/2 ] = \text{Exec. time}_{\text{old}} \times 0.95
\]

\[
\text{Speedup (E)} = \frac{\text{Exec. Time}_{\text{old}}}{\text{Exec. Time}_{\text{new}}} = \frac{1}{0.95} = 1.053
\]

- The new machine is 5.3% faster for this mix of instructions
Summary

- A fundamental rule in computer architecture is to make the common case fast.
- The most accurate measure of performance is the execution time of representative real programs (benchmarks).
- Execution time is dependent on the number of instructions per program, the number of cycles per instruction, and the clock rate.
- When designing computer systems, both cost and performance need to be taken into account.

- See Section 1.7 of the textbook for several examples of performance and price-performance in desktop, server, and embedded computers.