Lecture 1: What is Computer Architecture?

Karu Sankaralingam
University of Wisconsin-Madison
karu@cs.wisc.edu

Slides courtesy of Stephen W. Keckler, UT-Austin

Goals

• Understand the "how" and "why" of computer architectures
  - Instruction Set Architecture
  - Microarchitecture (advanced techniques for concurrency)
  - System Organization (processor, memory, interconnection networks)

• Learn methods of measuring and improving performance
  - Metrics, Benchmarks, Performance methods

• Discover the state of the art in computer architectures
  - Readings
  - Hands-on project
Logistics

Lectures  T, Th 9:30-10:45, 2540 ENGR HALL  
Lecturer  Prof. Karu Sankaralingam  
TA  Lena Olson  

Grading  
Reviews  20%  
Homework  12.5%  
Class Participation  5%  
Final Project  32.5%  
Exam 1 & 2  30%  

Text  
Hill, Joupi, & Sohi, *Readings in Computer Architecture*  
Course Reader  

CS/ECE 752 Online

URL:  http://www.cs.wisc.edu/~karu/courses/cs752/fall2010  
my e-mail: karu@cs.wisc.edu  
Include 752 in subject line  
e-mail: ece752-1-f07@lists.wisc.edu  
You should already be subscribed.  
Test message?  
Learn@UW: http://learnuw.wisc.edu  
Grades
Pre-Requisites

I expect you to *already* know the following:
• Boolean logic and gate-level design (Verilog?)
• Assembly language programming
• Basics of how a computer works
  - (fetch, decode, execute, memory, write-back)
• Pipelining
• Caches
  - Principles for why they work
• Virtual memory
  - How translation works, why it is useful

CS752 Topics

• Technology Trends
• Instruction set architectures
• Instruction-level parallelism
  - Dynamic ILP machines (OOO, speculation)
  - Static ILP machines (VLIW, SW pipelining)
• Concurrency
  - Multithreading, graphics pipelines
• Memory hierarchy
  - Cache optimizations, caching and virtual memory
• Multiprocessors
• Low power processors & GPUs
Reading List

• Reader 1 & 2
  - 23 papers to be reviewed total
  - Additional reference reading material
• H&P chapter readings
• Reviews due 5pm previous day
  - Submit to www.cs.wisc.edu/~karu/courses/cs752/fall2010/reviews
  - Before class skim other reviews
• In class we will discuss all

What is Computer Architecture?

Technology
Applications

API
ISA
Link
IO

Machine Organization

Measurement & Evaluation

Computer Architect

Interfaces
Layers of Abstraction

- Application
- Language
- Compiler
- ISA
- Microarchitecture
- Circuits
- Devices
- Solid-state physics

Technology Constraints

- Yearly improvement
  - Semiconductor technology
    - 60% more devices per chip (doubles every 18 months)
    - 15% faster devices (doubles every 5 years)
    - Slower wires
    - Leaky transistors
    - Unreliable transistors
  - Magnetic Disks
    - 60% increase in density
  - Circuit boards
    - 5% increase in wire density
  - Cables
    - no change

2000x more devices since 1989
10x faster devices

2005
Changing Technology leads to Changing Architecture

- **1970s**
  - multi-chip CPUs
  - semiconductor memory very expensive
  - microcoded control
  - complex instruction sets (good code density)

- **1980s**
  - single-chip CPUs, on-chip RAM feasible
  - simple, hard-wired control
  - simple instruction sets
  - small on-chip caches

- **1990s**
  - lots of transistors
  - complex control to exploit instruction-level parallelism

- **2000s**
  - even more transistors
  - slow wires
    - 1) More complex architectures (Pentium IV)
    - 2) Way more cache (24MB on Itanium II)
    - 3) Multiple cores/multiple threads (32 threads on Sun Niagara)
  - More processors (Nvidia GeForce 7800)
  - What else???

**Intel 4004 - 1971**

- The first microprocessor
- 2,300 transistors
- 108 KHz
- 10μm process
  - Human hair: approx ~20μm
Some Recent Chips!

Intel Pentium IV (1999)
- 42 million transistors
- 4.2 GHz
- 0.13µm process
- Could fit ~15,000 4004s on this chip!

NVidia - GeForce 6800
- 222 million transistors
- 400MHz
- 0.13µm process

Intel Atom (2009)
- 45nm process
- Only 47 million

IBM Cell
- 8 vector processors + 1 PPC
- 4 GHz
- 90nm process

Application Constraints

- Applications drive machine 'balance'
  - Numerical simulations
    - floating-point performance
    - main memory bandwidth
  - Transaction processing
    - I/Os per second
    - integer CPU performance
  - Decision support
    - I/O bandwidth
  - Embedded control
    - I/O timing
    - RTOS
  - Media processing
    - low-precision 'pixel' arithmetic
What is Computer Architecture?

Interface Design

- A good interface
  - lasts through several generations of implementations
    - IBM 360 and x86 ISAs, DOS APIs
  - is simple - 'economy of mechanism'
- Interfaces are visible, Implementations generally aren't
- 3 Types of Interfaces
  - Between Layers
    - API, ISA
  - Between Modules
    - Network protocol (Ethernet), I/O channel or bus (SCSI or PCI), USB
  - Standard Representation
    - ASCII, IEEE floating-point, Unicode
Instruction-Set Architecture

- Hardware/Software interface
  - Software impact
    - support OS functions
      - restartable instructions
      - memory relocation and protection
    - a good compiler target
      - simple
      - orthogonal
    - dense
  - Hardware impact
    - admits efficient implementation
      - across generations
    - admits parallel implementation
      - no 'serial' bottlenecks
    - Abstraction without interpretation

Microarchitecture

- Register-transfer-level (RTL)
  design
- Implement instruction set
- Exploit capabilities of technology
  - locality and concurrency
- Iterative process
  - generate proposed architecture
  - estimate cost
  - measure performance
- Still emphasis is on overcoming sequential nature of programs
  - deep pipelining
  - multiple issue
  - dynamic scheduling
  - branch prediction/speculation
System-Level Organization

- Design at the level of processors, memories, and interconnect.
- More important to application performance than CPU design
- Feeds and speeds
  - constrained by IC pin count, module pin count, and signaling rates
- System balance
  - for a particular application
- Driven by
  - performance/cost goals
  - available components (cost/perf)
  - technology constraints

An Example

- Google Project 02: Oregon/Washington border
- Cheap electricity: Columbia river
- Cheap b/w: surplus optic network from tech-boom era
- Google, Yahoo, and Microsoft
The Architecture Process

New concepts created

Estimate Cost & Performance

Sort

Bad ideas
Mediocre ideas
Good ideas

Performance Measurement and Evaluation

- Many dimensions to computer performance
  - CPU execution time
    - by instruction or sequence
      - floating point
      - integer
      - branch performance
  - Cache bandwidth
  - Main memory bandwidth
  - I/O performance
    - bandwidth
    - seeks
    - pixels or polygons per second
- Relative importance depends on applications
Evaluation Tools

- **Benchmarks, traces, & mixes**
  - macrobenchmarks & suites
  - application execution time
  - microbenchmarks
    - measure one aspect of performance
  - traces
    - replay recorded accesses
      - cache, branch, register

- **Simulation at many levels**
  - ISA, cycle accurate, RTL, gate, circuit

- **Area and delay estimation**

- **Analysis**
  - e.g., queuing theory


Metrics of Evaluation

- **Level of design ⇒ performance metric**

- **Examples**
  - Applications perspective
    - Time to run task (Response Time)
    - Tasks run per second (Throughput)
  - Systems perspective
    - Millions of instructions per second (MIPS)
    - Millions of FP operations per second (MFLOPS)
  - Bus/network bandwidth: megabytes per second
  - Function Units: cycles per instruction (CPI)
  - Fundamental elements (transistors, wires, pins): clock rate
Some Warnings about Benchmarks

- Benchmarks measure the whole system
  - application
  - compiler
  - operating system
  - architecture
  - implementation

- Popular benchmarks typically reflect yesterday's programs
  - what about the programs people are running today?
  - need to design for tomorrow's problems

- Benchmark timings are sensitive
  - alignment in cache
  - location of data on disk
  - values of data

- Danger of inbreeding or positive feedback
  - if you make an operation fast (slow) it will be used more (less) often
    - therefore you make it faster (slower)
      - and so on, and so on...
    - the optimized NOP

Performance Comparison Terminology

- "Y is n% slower than X" means:

  \[
  \frac{ExTime(Y)}{ExTime(X)} = 1 + \frac{n}{100}
  \]

- Example: Y takes 15 seconds to complete task, X takes 10 seconds
  - How much faster is X?
  - How much slower is Y?

- Same definitions apply to other metrics, such as throughput
Brief History of Benchmarking

• Early days (1960s)
  - Single instruction execution time
  - Average instruction time [Gibson 1970]
  - Pure MIPS (1/AIT)

• Simple programs (early 70s)
  - Synthetic benchmarks (Whetstone, etc.)
  - Kernels (Livermore Loops)

• Relative Performance (late 70s)
  - VAX 11/780 = 1-MIPS but was it?
  - MFLOPs

• “Real” Applications (late 80s-now)
  - SPEC
    • Scientific
    • Irregular
  - TPC
    • Transaction Processing
  - Winbench
    • Desktop
  - Graphics
    • Doom
    • MediaBench

Amdahl’s Law: Example

• FP instructions improved by 2x
• But….only 10% of instructions are FP

\[
ExTime_{new} = ExTime_{old} \times \left( 0.9 + \frac{0.1}{2} \right) = 0.95 \times ExTime_{old}
\]

\[
\text{Speedup}_{\text{total}} = \frac{1}{0.95} = 1.053
\]

• Speedup bounded by \( \frac{1}{\text{fraction of time not enhanced}} \)
Amdahl’s Law: Example 2

\[ T_1 = T_0 \left(1 - p\right) + \frac{p}{S} \]

Amdahl’s Law: Summary message

- **Make the Common Case fast**

- **Examples:**
  - All instructions require instruction fetch, only fraction require data
    \( \Rightarrow \) optimize instruction access first
  - Data locality (spatial, temporal), small memories faster
    \( \Rightarrow \) storage hierarchy: most frequent accesses to small, local memory
CPU Performance Equation

- 3 components to execution time:

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

- Factors affecting CPU execution time:

<table>
<thead>
<tr>
<th>Inst. Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Inst. Set</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MicroArch</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Consider all three elements when optimizing
- Workloads change!

Cycles Per Instruction (CPI)

- Depends on the instruction

\[CPI_i = \text{Execution time of instruction } i \times \text{Clock Rate}\]

- Average cycles per instruction

\[CPI = \sum_{i=1}^{n} CPI_i \times F_i \quad \text{where } F_i = \frac{IC_i}{IC_{\text{tot}}}\]

- Example:

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>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>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPI(total)</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Means

Arithmetic mean \( \frac{1}{n} \sum_{i=1}^{n} T_i \)  
Can be weighted: \( a_i T_i \)  
Represents total execution time

Harmonic mean \( \frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}} \)  
\( R_i = 1/T_i \)

Geometric mean \( \left( \prod_{i=1}^{n} \frac{T_i}{T_R} \right)^{\frac{1}{n}} \)  
Consistent independent of reference  
Best for combining results

Next Time

- Computer components  
  - Transistors and wires
- ISA overview/review  
  - MIPS ISA
- Reading assignment  
  - Course webpage and syllabus  
  - H&P Chapter 1, Appendix B  
  - Review due: Moore paper  
  - Also read:  
    - Mollick’s “Establishing Moore’s law”
- HW0 posted due 09/07