CS 61C:
Great Ideas in Computer Architecture
Compilers, Components
Instructor:
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/sp12

New-School Machine Structures
(It’s a bit more complicated!)

Levels of Representation/Interpretation

Review

Agenda

What is Typical Benefit of Compiler Optimization?

Compilers, Optimization, Interpreters, Just-In-Time Compiler
Administrivia
Dynamic Linking
Technology Trends Revisited
Technology Break
Components of a Computer

• Everything is a (binary) number in a computer
  — Instructions and data; stored program concept
• Assemblers can enhance machine instruction set to help assembly-language programmer
• Translate from text that easy for programmers to understand into code that machine executes efficiently:
  Compilers, Assemblers
• Linkers allow separate translation of modules
• Interpreters for debugging, but slow execution
• Hybrid (Java): Compiler + Interpreter to try to get best of both
• Compiler Optimization to relieve programmer

• What is a typical program?
• For now, try a toy program:
  BubbleSort.c
  #define ARRAY_SIZE 20000
  int main() {
    int iarray[ARRAY_SIZE], x, y, holder;
    for(x = 0; x < ARRAY_SIZE; x++)
      for(y = 0; y < ARRAY_SIZE-1; y++)
        if(iarray[y] > iarray[y+1]) {
          holder = iarray[y+1];
          iarray[y+1] = iarray[y];
          iarray[y] = holder;
        }
  }

Hardware
Software
...
Unoptimized MIPS Code

-O2 optimized MIPS Code

Compiler vs. Interpreter

Advantages

- Faster Execution
- Single file to execute
- Compiler can do better diagnosis of syntax and semantic errors, since it has more info than an interpreter (interpreter only sees one line at a time)
- Can find syntax errors before run program
- Compiler can optimize code

Bootstrap

Why Bytecodes?

- Platform-independent
- Load from the Internet faster than source code
- Interpreter is faster and smaller than it would be for Java source
- Source code is not revealed to end users
- Interpreter performs additional security checks, screens out malicious code

Java's Hybrid Approach: Compiler + Interpreter

- A Java compiler converts Java source code into instructions for the Java Virtual Machine (JVM)
- These instructions, called bytecodes, are same for any computer / OS
- A CPU-specific Java interpreter interprets bytecodes on a particular computer

Compilation:

- Takes longer to change source code, recompile, and relink

Interpreter:

- Slower execution times
- No optimization
- Need all of source code available
- Source code larger than executable for large systems
- Interpreter must remain installed while the program is interpreted

- Code compiled to MIPS
  
  - Assembly
  
  - Unoptimized
  
  - Optimized

- Code interpreted by a Java Virtual Machine (JVM)

---

---
JVM uses Stack vs. Registers

```
a = b + c;
=>
iload b ; push b onto Top Of Stack (TOS)
iload c ; push c onto Top Of Stack (TOS)
iadd ; Next to top Of Stack (NOS) = 
    Top Of Stack (TOS) + NOS
istore a ; store TOS into a and pop stack
```

Dynamic Linking

- Only link/load library procedure after it is called
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
  - Automatically picks up new library versions
  - Requires procedure code to be relocatable
- Dynamic linking is default on UNIX and Windows Systems

Dynamic Linking Idea

- 1st time pay extra overhead of DLL (Dynamically Linked Library), subsequent times almost no cost
- Compiler sets up code and data structures to find desired library first time
- Linker fixes up address at runtime so fast call subsequent times
- Note that return from library is fast every time
Administrivia

- Labs 5 and 6 posted, Project 2 posted
- Homework, Proj 2-Part 1 Due Sunday @ 11:59:59
- Midterm is now on the horizon:
  - No discussion during exam week
  - TA Review: Su, Mar 4, starting 2 PM, 2050 VLSB
  - Exam: Tu, Mar 6, 6:40-9:40 PM, 2050 VLSB (room change)
- Small number of special consideration cases, due to class conflicts, etc.—contact me

CSUA Github Help Session

- Wednesday 2/15, 6-8pm, 380 Soda.
- Learn about source control, git, setting up your Github account, and using GitHub for your CSUA Hackathon submission.
- Bring laptops.
- The presentation will be from 6:10-7. Individual troubleshooting help will be from 7-8.
- This helpsession will be especially useful for those attending CSUA’s Hackathon @436 on Friday. http://tinyurl.com/csuaHackathon

Projects

- Project 2: MIPS ISA simulator in C
  - Add ~ 200 (repetitive) lines of C code to framework
  - Lots of Cut & Paste
  - Appendix B describes all MIPS instructions in detail
  - Make your own unit test!

61C in the News

- “Erasing the Boundaries,” NY Times, 2/12/12
- The new strategy is to build a device, sell it to consumers and then sell them the content to play on it. ... Google is preparing its first Google-branded home entertainment device — a system for streaming music in the house —...fits solidly into an industry wide goal in which each tech company would like to be all things to all people all day long.
- Their job boards,...are brimming with positions for people with degrees in electrical engineering and hardware design.
- On Amazon's Web site, for example, the boards have dozens of listings for jobs with titles you might expect at a hardware company. Among them: Senior Hardware Engineering Manager, Director, Hardware Platforms and Systems, and Hardware EE Reliability Engineer. (EE is short for electrical engineer.)

Technology Cost over Time: What does Improving Technology Look Like?

Technology Cost over Time Successive Generations

How Can Tech Gen 2 Replace Tech Gen 1?
Moore’s Law

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year...That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000." (from 50 in 1965)

Gordon Moore, “Cramming more components onto integrated circuits,” Electronics, Volume 38, Number 8, April 19, 1965

"Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.”

Memory Chip Size

Growth in memory capacity slowing

End of Moore’s Law?

• It’s also a law of investment in equipment as well as increasing volume of integrated circuits that need more transistors per chip
• Exponential growth cannot last forever
• More transistors/chip will end during your careers
  – 2020? 2025?
  – (When) will something replace it?

Technology Trends: Uniprocessor Performance (SPECint)

Improvements in processor performance have slowed. Why?

Limits to Performance: Faster Means More Power

$P = CV^2$
\[ P = C \ V^2 \ f \]

- Power is proportional to Capacitance * Voltage*2 * Frequency of switching
- What is the effect on power consumption of:
  - “Simpler” implementation (fewer transistors)?
  - Smaller implementation (shrunk down design)?
  - Reduced voltage?
  - Increased clock frequency?

- Traditional processors consume about two thirds as much power at idle (doing nothing) as they do at peak
- Higher performance (server class) processors approaching 300 W at peak
- Implications for battery life?

Computer Technology: Growing, But More Slowly

- Processor
  - Speed: 2x / 1.5 years (since ’85-’05) *slowing*
  - Now 12 cores / 2 years
  - When you graduate: 3-4 GHz, 6-8 Cores in client, 10-14 in server
- Memory (DRAM)
  - Capacity: 2x / 2 years (since ’96) *slowing*
  - Now 20X-3-4 years
  - When you graduate: 8-16 Gigabytes
- Disk
  - Capacity: 2x / 1 year (since ’97)
  - 250X size last decade
  - When you graduate: 6-12 Terabytes
- Network
  - Core: 2x every 2 years
  - Access: 100-1000 mbps from home, 1-10 mbps cellular

Internet Connection Bandwidth Over Time

- Annualized growth rate per year

![Internet Bandwidth Growth, 1970-2012](chart)

- Logarithmic Fit

![Internet Bandwidth Growth, 1965-2005](chart)
Five Components of a Computer

- Control
- Datapath
- Memory
- Input
- Output

Reality Check: Typical MIPS Chip Die Photograph

Example MIPS Block Diagram

A MIPS Family (Toshiba)

The Processor

- Processor (CPU): the active part of the computer, which does all the work (data manipulation and decision-making)
  - Datapath: portion of the processor which contains hardware necessary to perform operations required by the processor (“the brawn”)
  - Control: portion of the processor (also in hardware) which tells the datapath what needs to be done (“the brain”)
Stages of the Datapath : Overview

• Problem: a single, atomic block which “executes an instruction” (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient
• Solution: break up the process of “executing an instruction” into stages or phases, and then connect the phases to create the whole datapath
  – Smaller phases are easier to design
  – Easy to optimize (change) one phase without touching the others

Project 2 Warning

• You are going to write a simulator in C for MIPS, implementing these 5 phases of execution

Phases of the Datapath (1/5)

• There is a wide variety of MIPS instructions: so what general steps do they have in common?
• Phase 1: Instruction Fetch
  – No matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)
  – Also, this is where we increment PC (that is, PC = PC + 4, to point to the next instruction: byte addressing so + 4)
• Simulator: Instruction = Memory[PC]; PC+=4;

Phases of the Datapath (2/5)

• Phase 2: Instruction Decode
  – Upon fetching the instruction, we next gather data from the fields (decode all necessary instruction data)
  – First, read the opcode to determine instruction type and field lengths
  – Second, read in data from all necessary registers
    • For add, read two registers
    • For addi, read one register
    • For jal, no reads necessary

Simulator for Decode Phase

Register1 = Register[rsfield];
Register2 = Register[rtfield];
if (opcode == 0) ...
  else if (opcode >5 && opcode <10) ...
    else if (opcode ...) ...
      else if (opcode ...) ...
  • Better C statement for chained if statements?

Phases of the Datapath (3/5)

• Phase 3: ALU function
  – Once the instruction is decoded, we next proceed to the ALU (or its equivalent) to perform the appropriate operation (add, subtract, shift, etc.)
  – For add, ALU adds the two registers
  – For addi, ALU adds the register and immediate value
  – For jal, no ALU operations necessary

Phases of the Datapath (4/5)

• Phase 4: Memory Access
  – Once the instruction is decoded, we next proceed to the memory unit (or its equivalent) to fetch the required data
  – For add, Memory fetches the second register
  – For addi, Memory fetches the immediate value
  – For jal, no Memory Access operations necessary

Phases of the Datapath (5/5)

• Phase 5: Write Back
  – Once the instruction is decoded, we next proceed to the registers to write back the results
  – For add, Write Back updates Register1
  – For addi, Write Back updates Register1
  – For jal, no Write Back operations necessary

Instruction Level Parallelism

P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12

Simulator: Instruction = Memory[PC]; PC+=4;
Phases of the Datapath (3/5)

- Phase 3: **ALU** (Arithmetic-Logic Unit)
  - Real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (slt)
  - What about loads and stores?
    - lw: $t0, 40($t1)
    - Address we are accessing in memory = the value in $t1 PLUS the value 40
    - So we do this addition in this stage
  - Simulator: Result = Register1 op Register2; Address = Register1 + Addressfield

Phases of the Datapath (4/5)

- Phase 4: **Memory Access**
  - Actually only the load and store instructions do anything during this phase; the others remain idle during this phase or skip it all together
  - Since these instructions have a unique step, we need this extra phase to account for them
  - (As a result of the cache system, this phase is expected to be fast: talk about next week)
  - Simulator: Register[rfield] = Memory[Address] or Memory[Address] = Register[rfield]

Phases of the Datapath (5/5)

- Phase 5: **Register Write**
  - Most instructions write the result of some computation into a register
  - E.g.,: arithmetic, logical, shifts, loads, slt
  - What about stores, branches, jumps?
    - Don’t write anything into a register at the end
    - These remain idle during this fifth phase or skip it all together
  - Simulator: Register[rfield] = Result
The ARM Inside the iPhone

iPhone Innards

You will about multiple processors, data level parallelism, caches in 61C

ARM Architecture

Review

- Key Technology Trends and Limitations
  - Transistor doubling BUT power constraints and latency considerations limit performance improvement
  - (Single Processor) computers are about as fast as they are likely to get, exploit parallelism to go faster
- Five Components of a Computer
  - Processor/Control + Datapath
  - Memory
  - Input/Output: Human interface/KB + Mouse, Display, Storage ... evolving to speech, audio, video
- Architectural Family: One Instruction Set, Many Implementations

• http://en.wikipedia.org/wiki/ARM_architecture