CS 61C: Great Ideas in Computer Architecture (a.k.a. Machine Structures)

Lecture 1: Course Introduction, Number Representation

Instructor: Sagar Karandikar (call me “Sagar”) sagark@eecs.berkeley.edu

About Your Instructor

• I graduated in May 2015 with a B.S. in EECS at Berkeley
• I am starting the Ph.D. program in CS at Berkeley in Fall 2015, focusing in Computer Architecture

My CS61C History:

<table>
<thead>
<tr>
<th>Position</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Fall 2012</td>
</tr>
<tr>
<td>TA</td>
<td>Spring 2013</td>
</tr>
<tr>
<td>Head TA</td>
<td>Summer 2013</td>
</tr>
<tr>
<td>TA</td>
<td>Fall 2013</td>
</tr>
<tr>
<td>TA</td>
<td>Spring 2014</td>
</tr>
<tr>
<td>Head TA</td>
<td>Fall 2014</td>
</tr>
<tr>
<td>Head TA</td>
<td>Spring 2015</td>
</tr>
</tbody>
</table>

Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What you need to know about this class
• Everything is a Number

CS61C is NOT really about C Programming

• It is about the hardware-software interface
  – What does the programmer need to know to achieve the highest possible performance?
• C is closer to the underlying hardware, unlike languages like Scheme, Python, Java!
  – Allows us to talk about key hardware features in higher level terms
  – Allows programmer to explicitly harness underlying hardware parallelism for high performance

Old School CS61C

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What you need to know about this class
• Everything is a Number
New School CS61C (1/3)

Personal Mobile Devices

New School CS61C (2/3)

Old School Machine Structures

New School CS61C (3/3)

My other computer is a data center

Old School Machine Structures

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

• Parallel Requests
  Assigned to computer
  e.g., Search “Katz”

• Parallel Threads
  Assigned to core
  e.g., Lookup, Ads

• Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions

• Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words

• Hardware description
  All gates functioning in parallel at same time

Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What you need to know about this class
• Everything is a Number
Great Idea #1: Abstraction (Levels of Representation/Interpretation)

- High Level Language Program (e.g., C)
  - Compiler
  - Assembly Language Program (e.g., MIPS)
  - Machine Language Program (MIPS)

Great Idea #3: Principle of Locality/ Memory Hierarchy

Interesting Times

Moore’s Law was based on how many transistors/chip at cheapest cost/transistor as technology scaled.

BUT newest, smallest fabrication processes <14nm, might have greater cost/transistor !!!!

So, why shrink????

Jim Gray’s Storage Latency Analogy: How Far Away is the Data?

- Tape/Optical Robot
- Disk
- Memory
- On Board Cache
- On Chip Cache
- Registers

Great Idea #2: Moore’s Law

Predicts: 2X Transistors / chip every 2 years

Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969

6 Great Ideas in Computer Architecture

1. Abstraction
   (Layers of Representation/Interpretation)
2. Moore’s Law (Designing through trends)
3. Principle of Locality (Memory Hierarchy)
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy
Great Idea #4: Parallelism

- Jane
  - Research
  - Composing
  - Typing
- Sue
  - Research
  - Composing
  - Typing
- Tom
  - Research
  - Composing
  - Typing

Caveat: Amdahl’s Law

Gene Amdahl
Computer Pioneer

Great Idea #5: Performance Measurement and Improvement

- Tuning application to underlying hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g., matrix manipulation)
- Latency
  - How long to set the problem up
  - How much faster does it execute once it gets going
  - It is all about time to finish

Coping with Failures

- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?
  a) 1 / month
  b) 1 / week
  c) 1 / day
  d) 1 / hour

NASA Fixing Rover’s Flash Memory

Opportunity still active on Mars after >10 years
But flash memory worn out
New software update will avoid using worn out memory banks

http://www.engadget.com/2014/12/30/nasa-opportunity-cover-flash-fix/
Great Idea #6: Dependability via Redundancy

- Redundancy so that a failing piece doesn’t make the whole system fail

2 of 3 agree

Increasing transistor density reduces the cost of redundancy.

1+1=2 1+1=2 1+1=2

FAIL!

Great Idea #6: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Agenda

- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- What you need to know about this class
- Everything is a Number

Yoda says...

“Always in motion, the future is...”

Our schedule may change slightly depending on some factors. This includes lectures, assignments & labs...

Weekly Schedule

Course Information

- Course Web: [http://inst.eecs.berkeley.edu/~cs61c/](http://inst.eecs.berkeley.edu/~cs61c/)
- Instructor:
  - Sagar Karandikar
- Teaching Assistants, Tutors, Readers: (see next slide)
- Textbooks: Average 15 pages of reading/week (can rent!)
  - Patterson & Hennessey, Computer Organization and Design, 5/e
  - Barroso & Holzle, The Datacenter as a Computer, 2nd Edition
- Piazza:
  - Every announcement, discussion, clarification happens there
Teaching Assistants

Head TA: Jay Patel
Head TA: Nathan Mailoa
TA: Derek Ahmed
TA: Rebecca Herman
TA: Harrison Wang
TA: Jeffrey Weitbliser

Tutors

Michelle Tsai
Alex Sung
Austin Tai
Brenton Chu
Nicolas Stone

Readers

Dashteng Chen
Molly Zhai

Course Grading

• EPA: Effort, Participation and Altruism (5%)
• Homework (10%) – graded on completion
• Labs (5%)
• Projects (20%) – graded on correctness
  1. Intro to C (beargit)
  2. C/MIPS (MIPS assembler/linker)
  3. Computer Processor Design (Logisim MIPS Processor)
  4. Performance/Parallel Programming
• Two midterms (15% each): 7/9 and 7/28, in-class, can be clobbered!
• Final (30%): 2015/8/13 @ 9am-12pm
• Performance Competition for honor (and EPA)

Tried-and-True Technique: Peer Instruction

• Increase real-time learning in lecture, test understanding of concepts vs. details
• As complete a “segment” ask multiple-choice question
  – 1-2 minutes to decide yourself
  – 2 minutes in pair/triples to reach consensus.
  – Teach others!
  – 2 minute discussion of answers, questions, clarifications
• You can get iClickers from the ASUC bookstore
  – We’ll start this next week!
  – No web-based clickers, sorry!
• Register clickers on bCourses (not the iClicker website, which may charge you money)
  – See https://go.berkeley.edu/zN6mR for more instructions

EECS Grading Policy

• http://www.eecs.berkeley.edu/Policies/ugrad_grading.shtml
  “A typical GPA for courses in the lower division is 2.7. This GPA would result, for example, from 17% A’s, 50% B’s, 20% C’s, 10% D’s, and 3% F’s. A class whose GPA falls outside the range 2.5 - 2.9 should be considered atypical.”
• Fall 2010: GPA 2.81
  26% A’s, 47% B’s, 17% C’s, 3% D’s, 6% F’s
• Job/Intern Interviews: They grill you with technical questions, so it’s what you say, not your GPA
  (New 61C gives good stuff to say)
My Goal as an Instructor

- To make your experience in CS61C as enjoyable & informative as possible
  - Humor, enthusiasm & technology-in-the-news in lecture
  - Fun, challenging projects & HW
  - Pro-student policies (exam clobbering)
- To maintain Berkeley & EECS standards of excellence
  - Projects & exams will be as rigorous as every year.
- Score 7.0 on HKN:
  - Please give feedback so we can improve! Why are we not 7.0 for you? We will listen!!

Late Policy ... Slip Days!

- Projects due at 11:59:59 PM
- You have 3 slip day tokens (NOT hour or min)
- Every day your project is late (even by a minute) we deduct a token
- After you've used up all tokens, 1/3 of the potential points are deducted per day.
  - No credit if more than 3 days late
  - No need for sob stories, just use a slip day!

Policy on Assignments and Independent Work

- ALL PROJECTS WILL BE DONE WITH A PARTNER
- We expect that projects you turn in are the work of your team and YOUR TEAM ALONE. You should not at any point share code or pseudocode
- PARTNER TEAMS MAY NOT WORK WITH OTHER PARTNER TEAMS
- You are encouraged to discuss your assignments with other students, and extra credit will be assigned to students who help others, particularly by answering questions on Piazza, but we expect that what you hand in is yours.
- It is NOT acceptable to copy solutions from other students.
- It is NOT acceptable to copy (or start your) solutions from the Web.
- It is NOT acceptable to use PUBLIC github archives (giving your answers away)
- We have software tools for detecting plagiarism and they are extremely effective. You WILL be caught, and the penalties WILL be severe.
- At the minimum F in the course, and a letter to your university record documenting the incidence of cheating.
- If we've caught people in recent semesters)
- Both Giver and Receiver are equally culpable and suffer equal penalties
- If in doubt, ask the instructor or a TA!

Architecture of a typical Lecture

```
Full Attention
Clickers
Adminstrivia
Clickers
Fun/News
```

“And in conclusion...”

<table>
<thead>
<tr>
<th>Time (minutes)</th>
<th>0</th>
<th>10</th>
<th>35</th>
<th>60</th>
<th>85</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Comments on the Summer Variant

- Summer is incredibly hectic
  - We run at 2x the standard pace of the class
  - Falling behind just a little can be disastrous
    - If the course begins to overwhelm you, don’t wait, contact me or your TA immediately
  - The first week will go slowly (only homework, no project), but we will ramp up to full speed starting next week
    - Project 1 will release on Sunday
Agenda

- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- What you need to know about this class
- Everything is a Number

Key Concepts

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, ...
- Integer and floating-point operations can lead to results too big to store within their representations: overflow/underflow

Number Representation

- Value of i-th digit is \( d \times Base^i \) where \( i \) starts at 0 and increases from right to left:
  - \( 123_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \)
  - \( = 100_{10} + 20_{10} + 3_{10} \)
  - \( = 123_{10} \)
- We will frequently use 3 bases to represent integers: Binary (Base 2), Hexadecimal (Base 16), and Decimal (Base 10)

- Hexadecimal digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - \( FFF_{hex} = 15_{10} \times 16^2 + 15_{10} \times 16^1 + 15_{10} \times 16^0 \)
  - \( = 3840_{10} + 240_{10} + 15_{10} \)
  - \( = 4095_{10} \)
- May put blanks every group of binary, octal, or hexadecimal digits to make it easier to parse, like commas in decimal
- Lots of conversion practice in discussion today

Signed and Unsigned Integers

- C, C++, and Java have **signed integers**, e.g., 7, -255:
  - int x, y, z;
- C, C++ also have **unsigned integers**, which are used for addresses
- 32-bit word can represent \( 2^{32} \) binary numbers
- Unsigned integers in 32 bit word represent 0 to \( 2^{32} - 1 \) (4,294,967,295)

Unsigned Integers

- \( 00000000000000000000000000000000_{bin} = 0_{dec} \)
- \( 00000000000000000000000000000001_{bin} = 1_{dec} \)
- \( 00000000000000000000000000000010_{bin} = 2_{dec} \)
- \( 01111111111111111111111111111111_{bin} = 2,147,483,647_{dec} \)
- \( 01111111111111111111111111111110_{bin} = 2,147,483,646_{dec} \)
- \( 01111111111111111111111111111101_{bin} = 2,147,483,645_{dec} \)
- \( 01111111111111111111111111111100_{bin} = 2,147,483,644_{dec} \)
- \( 11111111111111111111111111111111_{bin} = 4,294,967,295_{dec} \)
- \( ... \)

- \( 11111111111111111111111111111111_{bin} = 4,294,967,295_{dec} \)
- \( ... \)
- \( 11111111111111111111111111111111_{bin} = 4,294,967,295_{dec} \)
Signed Integers and Two's-Complement Representation

- Signed integers in C; want \( \frac{1}{2} \) numbers <0, want \( \frac{1}{2} \) numbers >0, and want one 0
- Two's complement treats 0 as positive, so 32-bit word represents \( 2^{32} \) integers from \(-2^{31} \) to \( 2^{31} - 1 \) (2,147,483,647)
  - Note: one negative number with no positive version
  - Book lists some other options, all of which are worse
- Every computer uses two's complement today

Most-significant bit (leftmost) is the sign bit, since 0 means positive (including 0), 1 means negative
- Bit 31 is most significant, bit 0 is least significant

Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented

Ways to Make Two’s Complement

- For N-bit word, complement to \( 2^N \)
  - For 4 bit number \( 3_{\text{ten}} = 0011_{\text{two}} \), two's complement (i.e. \( -3_{\text{ten}} \)) would be
    \[ 16_{\text{ten}} 3_{\text{ten}} = 13_{\text{ten}} \text{ or } 1000_{\text{two}} - 0011_{\text{two}} = 1101_{\text{two}} \]
  - Here is an easier way:
    \[ 3_{\text{ten}} = 0011_{\text{two}} \]
    - Invert all bits and add 1
    - Computers actually do it like this, too

Suppose we had a 5-bit word. What integers can be represented in two's complement?

- -32 to +31
- 0 to +31
- -16 to +15
- -15 to +15
Wrap-Up Administrivia

• HW0 will be out shortly, due on Sunday
  – We will create edX accounts for everyone
• HW0-mini-bio is also out, due to your TA in lab on Tuesday 6/30
• Must notify us by the end of this week about any exam conflicts
• DSP: please have your letters sent to us ASAP
• Discussions begin today, labs begin tomorrow
• Read the full course policies: https://goo.gl/dtv71A
• Obtain iClickers by Monday 6/29
• Register your proj1 teams (more on Piazza)

Summary

• CS61C: Learn 6 great ideas in computer architecture to enable high performance programming via parallelism, not just learn C
  1. Abstraction
     (Layers of Representation/Interpretation)
  2. Moore’s Law
  3. Principle of Locality/Memory Hierarchy
  4. Parallelism
  5. Performance Measurement and Improvement
  6. Dependability via Redundancy
• Everything is a Number!