CS 61C: Great Ideas in Computer Architecture

Course Summary & Celebration!

Instructor: Justin Hsia
Agenda

• Course Summary
• Administrivia
• What’s Next?
• Acknowledgements
Six Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Technology Trends
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy
Great Idea #1: Levels of Representation/Interpretation

Higher-Level Language Program (e.g. C)

Compiler

Assembly Language Program (e.g. MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g. block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
Number Representation

- *Anything* can be represented as a number!
  - With $n$ digits in base $B$, can represent $B^n$ things
- IEC (vs. SI) prefixes ($2^{10} \approx 10^3$)
- Signed and unsigned integers
  - Addition, subtraction, overflow, sign extension
  - Two’s complement (better than 1’s and sign&mag)
- Floating point (sign, biased exp, significand)
  - Inf, NaN, 0, denorms
  - Precision and truncation
Higher-Level Language (HLL)

• We studied C because exposes more of hardware (particularly memory)
  – Compiled language is machine-dependent
• Arrays and strings
  – Don’t run off the end or forget null terminator
• Pointers hold addresses, used to pass by ref
  – Pointer arithmetic
  – Array vs. pointer syntax
• Structs are padded collections of variables
Assembly Language

• Close to the level that a machine understands
  – ISA in human-readable format
  – TAL vs. MAL (pseudo-instructions)
• RISC vs. CISC and effects
• MIPS Instruction Formats: R, I, J
  – Meaning and limitations of the fields
  – Relative (branch) vs. absolute (jump) addressing
  – Register conventions (saved/volatile; caller/callee)
• Assembler: instr translation, sym/rel tables
Machine Language

- Everything is just 0’s and 1’s!
  - Usually done by assembler & linker, but can also do manually
  - At this level, just raw bits! No differentiation between instructions and different types of data
  - Alignment matters

- Executable produced by linker, run by loader
  - OS creates new process (PT, swap space)
  - Instructions loaded and run by processor
Hardware Architecture Description

• Processor split into Datapath and Control
  – *Datapath components*: I$\$/D$, RegFile, ALU, Extender, MUXes
    • Datapath Stages: IF, ID, EX, MEM, WB
  – *Controller*: Use “AND” and “OR” Logic blocks to determine control signal values for *each* instruction

• Can build/design components hierarchically

• Behavior of many circuits/programs can be represented using Finite State Machines
  – States, transition function, initial state
Logic Circuit Description

• Build Synchronous Digital Systems out of combinational and sequential logic
• Equivalence between Circuit Diagrams, Truth Tables, and Boolean Expressions
  – Can convert between all representations
• Boolean algebra allows for circuit simplification (Karnaugh maps, too)
• FSMs built with registers and CL
• In reality, everything wires and transistors
  – Voltage-controlled switches (1: high, 0: low)
Great Idea #2: Technology Trends

Predicts: Transistor count per chip doubles every 2 years

Gordon Moore
Intel Cofounder
B.S. Cal 1950
Technology Trends

- Dynamic power = \( C \times V^2 \times f \)
  - Capacitance, voltage, switching frequency
- In WSC: Power Usage Effectiveness (PUE) = Total building power / IT equipment power
- Technology growth is slowing, processors have hit a power wall
  - Everywhere: transistor density, CPU speed, disk and memory capacity
  - Performance improvements now coming from parallelism and multicore processors
Transition to Multicore
Great Idea #3: Principle of Locality/Memory Hierarchy

- Trade-off in speed and cost vs. capacity!
- Increasing distance from application/user
Memory

• Programmer treats as one long array
  – You know that this is just an illusion (VM)!

• Memory is *byte-addressed*
  – Most data (including instructions) in words and *word-aligned*, so all word addresses are multiples of 4 (end in **0b00**)

• Multicore systems use shared memory
  – Synchronization/cache coherence necessary
    • e.g. MOESI snooping protocol
Memory Management

- Program’s *address space* contains four regions:
  - **Stack**: local variables, grows downward
  - **Heap**: space requested for pointers via `malloc()`; resizes dynamically, grows upward
  - **Static Data**: global and static variables, does not grow or shrink
  - **Code**: loaded when program starts, does not change size

![Diagram of memory regions]

```
~FFFF FFFF_hex

stack

heap

static data

code
```

8/13/2013
Summer 2013 -- Lecture #29
Typical Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology.
Accessing Data

1) Check TLB/PT if page is in main memory
   – Page fault to load from disk (swap space) if not

2) Check cache for data
   – Fetch from main memory on cache miss
   – Return data to processor
Caching Details (1/2)

• Move data in contiguous *blocks*
  – Store data in cache, closer to processor

• Cache organization
  – Map data addresses to the limited number of slots in cache
    – Set associativity: # of slots per set

• TIO breakdown
  – Tag as block identifier, Index to find set in cache,
    Offset to find data within block

• Also store management bits for *each* slot
  – Valid bit, (Dirty bit), Tag bits; replacement bits per set
Caching Details (2/2)

• Cache parameters affect performance
  – Block size, cache size, set associativity
  – Write-back/write-through policies
  – Write allocate/no-write allocate policies
  – Block replacement policy (Least Recently Used)

• Source of cache misses: The 3 C’s
  – Compulsory, capacity, conflict

• Multilevel caches reduce miss penalty
Virtual Memory Details (1/3)

• Give main memory effective size of disk without major penalty to performance
  – Move data in contiguous *pages* from disk to main memory
  – Assumption is that memory is small compared to both disk and virtual address space (or many processes)

• Also provide protection for multiple processes
  – Requires a lot of work by operating system
Virtual Memory Details (2/3)

• Paging requires address translation
  – Can run programs larger than main memory
  – Hides variable machine configurations (RAM/HDD)
  – Solves fragmentation problem

• Address mappings stored in page tables in memory
  – Additional memory access mitigated with TLB, which is a cache for page table
  – Management bits: Valid, Dirty, Ref, Access Rights
Virtual Memory Details (3/3)

• Running multiple processes:
  – Each process has its own page table and swap space in disk
  – OS can switch between by saving and loading process states (PC, reg vals, page table address)
  – On context switch, flush TLB and pipeline
Input/Output

• Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead
• Processor must synchronize with I/O devices before use due to difference in data rates:
  – Polling works, but expensive due to repeated queries
  – Exceptions are “unexpected” events in processor
  – Interrupts are asynchronous events that are often used for interacting with I/O devices
• In SW, need special handling code
Great Idea #4: Parallelism

Software

- Parallel Requests
  Assigned to computer
  e.g. search “Garcia”

- 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

- Warehouse Scale Computer

Leverage Parallelism & Achieve High Performance

- Computer
  - Core
  - ... Core
  - Memory
  - Input/Output

- Instruction Unit(s)
  - A<sub>0</sub>+B<sub>0</sub>
  - A<sub>1</sub>+B<sub>1</sub>
  - A<sub>2</sub>+B<sub>2</sub>
  - A<sub>3</sub>+B<sub>3</sub>

- Functional Unit(s)

- Cache Memory

- Logic Gates

8/13/2013
Summer 2013 -- Lecture #29
Types of Parallelism (1/4)

• Request-Level Parallelism (RLP)
  – Handling many requests per second
    (e.g. web search)

• Data-Level Parallelism (DLP)
  – Operate on many pieces of data at once
  – SIMD: at the level of single instructions
  – MapReduce: at the level of programs (split into map and reduce)
Types of Parallelism (2/4)

• Thread-Level Parallelism (TLP)
  – Have many processors, run either different programs or different parts of same program at same time
  – If same program, need to deal with shared memory (cache coherence and synchronization primitives to prevent data races)
  – Splitting up work properly is difficult!
    • Shared vs. private variables in OpenMP
    • Often requires re-designing your algorithm
Types of Parallelism (3/4)

• Thread-Level Parallelism (TLP)
  – Synchronization requires hardware support
    • Test-and-set mechanism
    • 11 and sc in MIPS
  – OpenMP directives
    • parallel, for, sections, single, etc.
    • critical, atomic, master
Types of Parallelism (4/4)

• Instruction Level Parallelism (ILP)
  – Pipelining: increase throughput by adding registers
    • Reduce critical path, increase max frequency
    • Working on multiple instructions at once introduces hazards (structural, data, control)
    • Forwarding, delay slots, branch prediction
  – Multiple instruction issue (superscalar)
    • Register renaming, speculation, out-of-order execution
    • Loop unrolling
Great Idea #5: Performance Measurement and Improvement

- Allows direct comparisons of architectures and quantification of improvements
- It is all about *time to finish* (latency)
  - Includes both *setup* and *execution*.
- Match application and hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g. matrix manipulation)
Performance Measurements

• Execution time (latency) and work per time (throughput)
  – CPU Time = Instructions × CPI × Clock Cycle Time

• Memory Access:
  – AMAT, CPI_{stall} use hit time, miss rate, miss penalty
  – Definitions recursive back to last level in hierarchy

• Amdahl’s Law
  – Speedup = 1 / [ (1-F) + F/S ]
  – Why we almost never get max possible speedup
Performance Programming

• **Key challenge:** Craft parallel programs that scale well (weak/strong scaling)
  – Scheduling, load balancing, time for synchronization, overhead for communication

• Some techniques:
  – Register/Cache Blocking
  – Data Parallelism & Loop Unrolling
  – Multithreading
Dependability Measures

• Failure rates and time
  – MTTF, MTTR, MTBF

• Availability = MTTF / (MTTF + MTTR)
  – Use measures in # of 9s

• Annualized Failure Rate (AFR)
Great Idea #6: Dependability via Redundancy

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

• 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)
Redundant Arrays of Inexpensive Disks

- Simulate behavior of single larger disk with an array of smaller disks
  - Cheaper, higher bandwidth, more resistant to failure
- RAID 0 – Disk striping, no redundancy
- RAID 1 – Mirroring for redundancy
- RAID 2 – Bit-level striping with ECC parity disks
- RAID 3 – Byte-level striping with dedicated parity disk
- RAID 4 – Block-level striping with dedicated parity disk
- RAID 5 – Block-level striping with interleaved parity
- RAID 6 – RAID 5 + extra parity blocks for DEC
Error Detection & Correction

• Even parity using XOR

• Hamming Distance
  – Distance 2 can detect 1-bit error
  – Distance 3 can detect & correct 1-bit error
  – Distance 4 can correct 1-bit error and detect 2-bit errors

• Hamming ECC
  – Introduce extra parity bits (one per group)
  – Sum of group errors indicates corrupted bit
Agenda

- Course Summary
- Administrivia
- What’s Next?
- Acknowledgements
Administrivia

• TAs will be available during normal Discussion and Lab times
• Optional lecture tomorrow
  – Memory Security given by Albert
• Final Review tonight @ 7-10pm, 10 Evans
• Final Exam Fri @ 9am-12pm, 155 Dwinelle
  – Two two-sided handwritten cheat sheets
  – Green sheet provided
• Extra OH: Justin Thu 1-7pm 200 SDH
  – Others, see Piazza @1061
Project 1 EC Warriors

• *memory.c + next-fit:*

1) Joshua Lam 110 pts
2) Gani Bhaskara 110 pts
3) Jesse Halim 108.5 pts
4) Alex Cruz 108.5 pts
5) Kevin Casey 107 pts
6) Leyuan Jia 107 pts
Project 2: sgemm-small.c

Average Speed on Small Matrices m=[32,100] by n=[32,300]

Mean: 7.0
Std Dev: 1.5
Project 2: sgemm-small.c

Average Speed on 36 x 36 Matrices

- Mean: 12.8
- Std Dev: 1.9
Project 2: sgemm-openmp.c

Average Speed on Large Matrices m=[1000,10000] by n=[32,100]

Mean: 54.4
Std Dev: 11.2
Project 2 Fastest Submissions

• **sgemm-small (small):**
  
  1) 9.79 Gflop/s  Samuel Leung / Luke Song
  
  2) 9.62 Gflop/s  Andy La / Ada Lin
  
  3) 9.43 Gflop/s  Vikram Arouza / Arun Jandaur

• **sgemm-small (36×36):**
  
  1) 16.97 Gflop/s  HongJun Jin / Ki Hyun Won
  
  2) 15.63 Gflop/s  Ashleigh Cushman / Lisa Li
  
  3) 15.55 Gflop/s  <14-way tie!>
Project 2 Fastest Submissions

• **sgemm-openmp:**
  1) 80.07 Gflop/s  Ashleigh Cushman / Lisa Li
  2) 73.69 Gflop/s  Arjun Parthiban / Baljot Singh
  3) 72.49 Gflop/s  Tammuz Dubnov / Jack Wilson

• **sgemm-threads:**
  1) 69.96 Gflop/s  Kevin Casey / William Huang
  2) 68.34 Gflop/s  Alex Chu / Jonghyun Ahn
  3) 61.03 Gflop/s  Gani Bhaskara / Milkent Samsurya
Participation Awards

• **Piazza answers:**
  1) William Huang 208
  2) Abhinav Gautam 74
  3) Kevin Casey 59

• **Peer Instruction Scores:**
  1) Edward Lai 106.2
  2) Samuel Leung 105.8
  3) David Dinh 103.5
Agenda

• Course Summary
• Administrivia
• What’s Next?
• Acknowledgements
What’s Next?

• Take classes from great teachers! (teacher > class)
  – Distinguished Teaching Award (very hard to get)
  – HKN Course evaluations (≥ 6 is very good)
  – Upcoming instructors for classes: (CS / EE)

• Classes related to CS 61C
  – CS169 Software Engineering (for SaaS, Fox/Patterson Fall 13)
  – CS194-15 Engineering Parallel Software
  – CS164 Programming Languages and Compilers
  – CS162 Operating Systems and Systems Programming
  – CS152 Computer Architecture and Engineering (Sp14)
  – CS150 Components and Design Techniques for Digital Systems
Opportunities in Teaching

• Interest in joining the CS staff?
  – Applies for CS 10, 61A, 61B, 61C
  – **Usual path:** Lab Assistant → Reader → TA
  – **Also:** Self-Paced Center Tutor

• Requirements:
  – Interest in teaching
  – Stricter grade requirements based on where you want to jump in

• Applying:
  – **Application form** (for TA, Reader, or Lab Assistant)
  – Doesn’t hurt to e-mail professor as well
Opportunities at Cal

• Why are we a top university in the WORLD?
  – Research, research, research!
  – Classes are just the tip of the iceberg
  – Whether you want to go to grad school or industry, you need someone to vouch for you
  – Won’t know if you like it or not until you try

• Find out what you like, do lots of web research (read published papers), hit OH of professor, show enthusiasm & initiative
Agenda

- Course Summary
- Administrivia
- What’s Next?
- Acknowledgements
Special Thanks

Professor
Randy Katz

Professor
David Patterson

Lecturer SOE
Dan Garcia

Su11
Instructor
Michael Greenbaum
Special Thanks to the Staff:

- **TAs:** Albert Magyar, Jeffrey Dong, Justin Fu, Kevin Yeun, Sagar Karandikar, Shaun Benjamin
- **Readers:** Albert Lu, Alvin Wong, Jay Patel, Jhoong Roh, Kelvin Chou, Peter Chen
- **Lab Assistants:** Emily Shiue, Eric Oh, James Yang, Jeffrey Li, Michael Huston, Tomo Ueda, Shahin Ashrafzadeh