Early Adopter - Fundamentals of Computer Systems

Martha A. Kim
Computer Science Department
Columbia University

EduPar 2009
May 16, 2011

Computer Science @ Columbia University

Columbia University

Four year, research-intensive university
Columbia College (CC) (~4k total enrollment)
School of Engineering and Applied Sciences (SEAS) (~1.5K total enrollment)
School of General Studies (GS) (~1.5K total enrollment)

Computer Science Curriculum

Undergraduate majors from CC, SEAS, and GS
Two years of computer science core + two years in selected track)

CS Core

- Intro programming
- Data structures and algorithms
- Complexity theory

-Fundamentals of computer systems

- Computational linear algebra
- Probability and statistics

CS Tracks

-Foundations of CS
(algorithms, computational complexity, scientific computing, and security)

-Systems
(networking, programming languages, operating systems, and software systems)

-Artificial Intelligence
(machine learning, robots, and systems capable of exhibiting ''human-like'' intelligence)

-Applications
(interactive multimedia applications for the Internet and wireless networks)

-Vision and Graphics
(vision, graphics, and advanced forms of human-computer interaction)

Fundamentals of Computer Systems Overview

- Full title: CSEE 3827: Fundamentals of Computer Systems
- Required course, part of CS core (as well as EE and CE core)
- Students typically take this course in their second or third year
- Taught every semester
  - Fall enrollment typically ~40 students
  - Spring enrollment typically ~80 students
- Textbooks
  - Until Spring 2011: Logic and Computer Design Fundamentals, Mano & Kime + Computer Organization & Design, Patterson & Hennessy
  - Currently: Digital Design and Computer Architecture, Harris & Harris
- Website: `http://www.cs.columbia.edu/~martha/courses/3827/sp11`

Course Syllabus - Part I: Digital Logic

Approximate lecture hours spent

Bloom level

Information Representation	1	A
BCD		
2's Complement		
ASCII		
general coding		
Boolean Logic and Algebra	5	A
Algebraic rules		
Standard forms (SOP, POS)		
Karnaugh Maps		
Timing		
Combinational Circuit Design	3	A
Multiplexers		
Encoders and Decoders		
Arithmetic Circuits		

Sequential Circuit Design	3	A
Latches		
Flip-Flops		
Timing Constraints		
Registers		
Finite State Machine Design	3	A
Moore		
Mealy		
Storage Architectures	2	C
Base cell abstraction		
Array organization		
Register v. SRAM v. DRAM		

Course Syllabus - Part II: Machine Organization

Instruction Set Architectures	1	C
Benefit		
RISC v. CISC		
MIPS ISA	5	A
Arithmetic instructions		
Load & stores		
Branches and control		
Calling conventions		
Stack management		
Instruction formats		
Performance Analysis	1	A
Throughput v. latency		
CPI, etc.		
Speedup & Amdahl's Law		
Single Cycle MIPS Processor	2	A
Datapath		
Control		
Performance analysis		

Pipeline MIPS Processor	3	A
Datapath & control		
Data hazards		
Control hazards		
Performance analysis		
Caches	2	C
Memory hierarchy		
Performance analysis		
Direct mapped cache		
N-way associative cache		
Fully associative cache		
LRU replacement		
Advanced Microarchitectures	1	K
CPI, and other components		
Speedup and Amdahl's Law		
Modern Architectures	1	K
Multicore (motivation)		
GPUs		

General Strategy for Incorporating Parallelism

- The syllabus already contained a number of parallelism related concepts

- To emphasize them for the students, made special effort to highlight parallelism and other recurring themes in CS as they arose over the semester, and to draw analogies to where students may already have seen them.

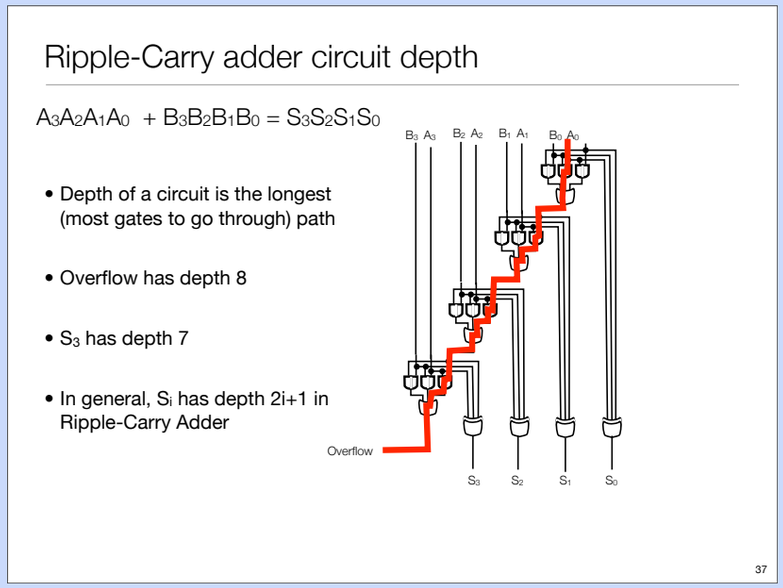
Theme	In this course	Seen before
Abstraction	Hierarchical circuit design	Functional decomposition of software
Interface v. implementation	ISA v. microarchitecture	Public v. private methods and fields in Java
Serial v. parallel	Combinational v. sequential circuits	?

- Finally, made the occasional passing reference to relevant advanced feature of current topic (e.g., the value of non-blocking cache design when teaching caches)

Example 1: Circuits Compute in Parallel

- When teaching **combinational circuits**, heavy emphasis on parallel nature of their operation.
    - Logic gates are always computing, all at once, but not necessarily synchronized
    - Signals propagate constantly, but not in synchronized fashion
- **Observation**: students assume synchronous circuitry (likely from sequential programming training) before they've ever seen sequential synchronous circuits.

Combinational Circuit Timing Analysis
A circuit has many paths from input to output
Signals are constantly propagating along these paths
Correct signals start propagating from correct input values
Some signals are faster than others (they race)
Computation done when last correct signal arrives at output
This is the critical path



Example 2: Pipelines = Parallel Execution

- Emphasized that the benefit of pipelining was **improved throughput** thanks to parallel **instruction execution**

- Extended discussion of throughput v. latency, and how parallel resources can boost the former. Employed analogy to web host adding servers. Can serve more clients, but latency is unchanged for each client.

- **Observation**: as with first serial then parallel programming instructions, students struggle to shift their mental model from a serial (single cycle processor) to a parallel processor with multiple instructions in-flight at once.

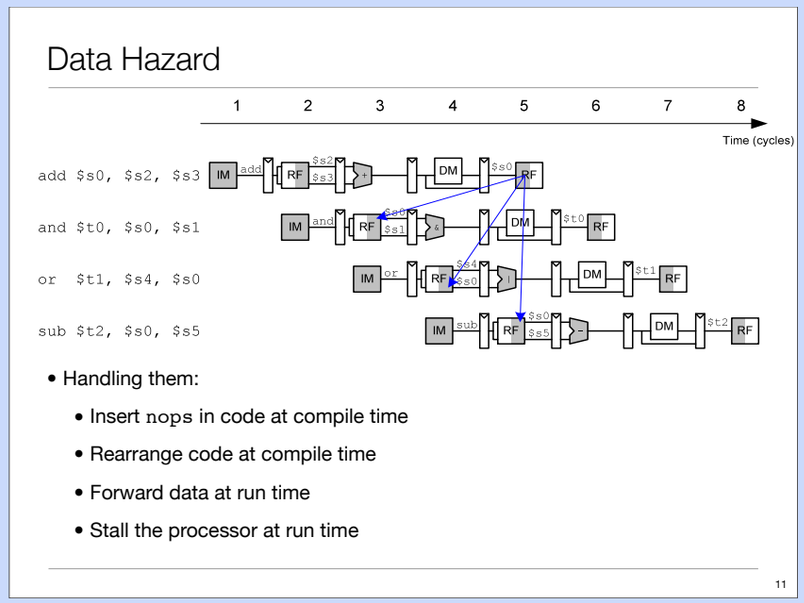
Pipeline Hazards

-Hazards arise because a pipeline executes multiple instructions at once

-At runtime processors detect conflicts between concurrent instructions

-When possible resolve with bypassing (allowing all to proceed in parallel)

-When not possible stall processor (making execution more serial)



Example 3: Advanced Microarchitectures

- At the end of the semester, build on course foundations to expose student to advanced microarchitectures. Emphasize that vast majority of **optimizations are designed with the goal of doing more work in parallel**.

- **Observation**: students are excited by topics they have heard the name of (GPUs, multicore). Despite these topics not being on the exam, students still asked more questions in these lectures than the others.

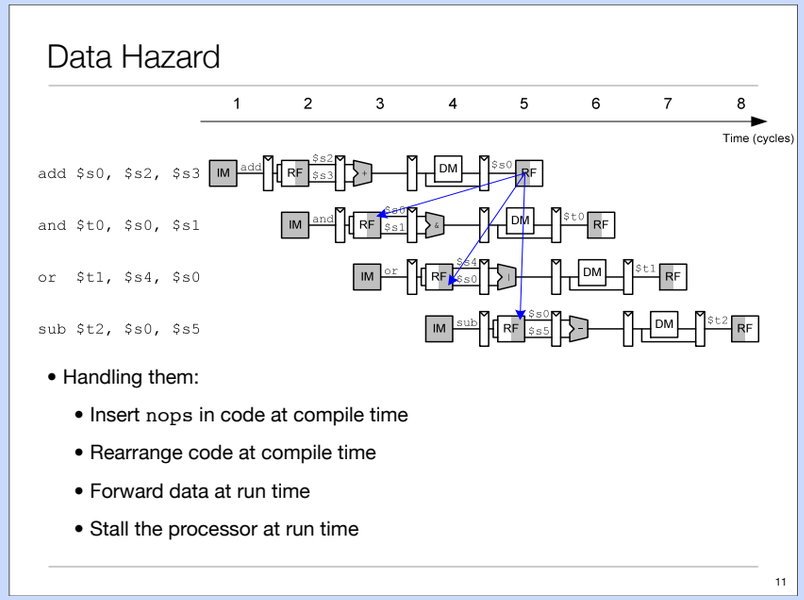
Pipeline Hazards

-Hazards arise because a pipeline executes multiple instructions at once

-At runtime processors detect conflicts between concurrent instructions

-When possible resolve with bypassing (allowing all to proceed in parallel)

-When not possible stall processor (making execution more serial)



Example 3: Advanced Microarchitectures

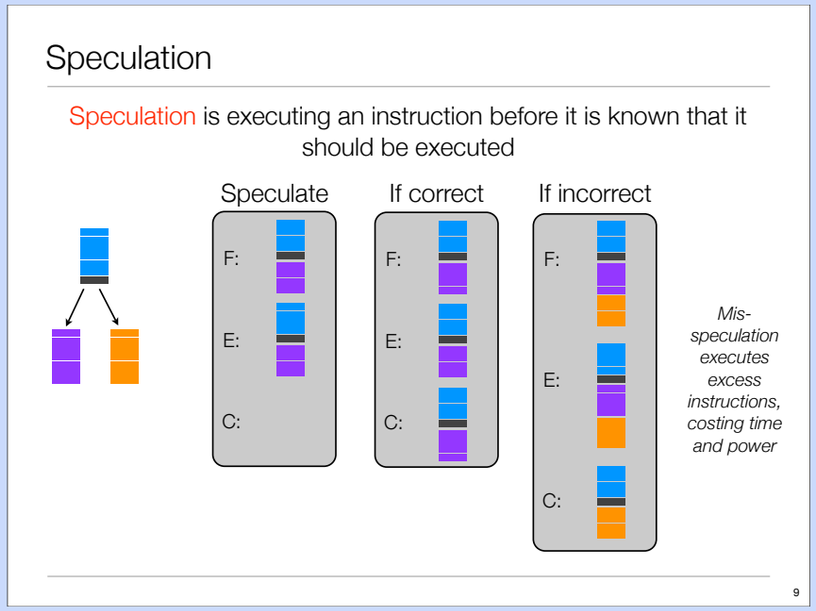
- At the end of the semester, build on course foundations to expose student to advanced microarchitectures. Emphasize that vast majority of **optimizations are designed with the goal of doing more work in parallel**.

- **Observation:** Given basic foundation and time constraints, must introduce these topics only at the conceptual level.

- **Observation**: students are excited by topics they have heard the name of (GPUs, multicore).

Speculative Execution

- ILP is limited by branches (students learned this when we taught control hazards in pipelines)
-To find more instructions to keep a pipeline full, some processors guess the branch outcome
-Brief overview branch prediction strategies
-If the prediction is correct - hooray! the pipeline stayed full of useful work
-If the prediction was wrong - we lost time doing unnecessary work

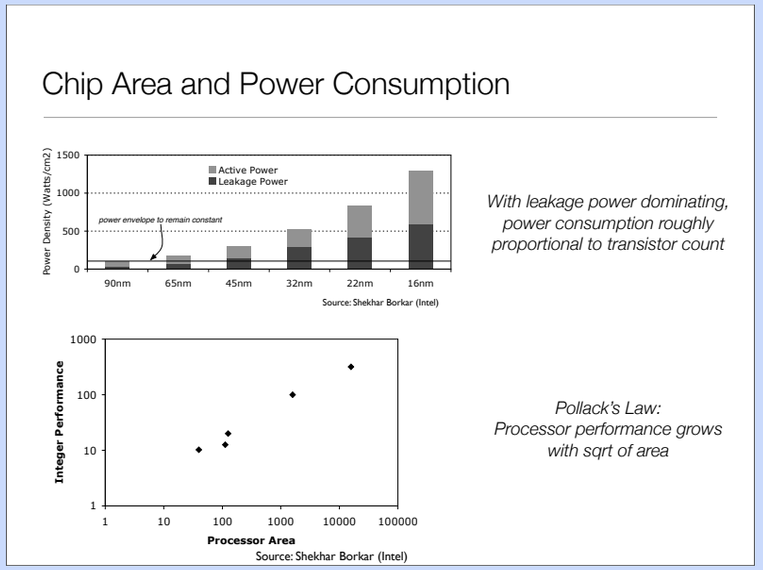


Example 4: Multicore

- The students have all heard of multicore, and often come in thinking more cores is better. They are very surprised to hear about why multicore can make things more difficult (on the software side) and then curious to understand why we went multicore anyway.

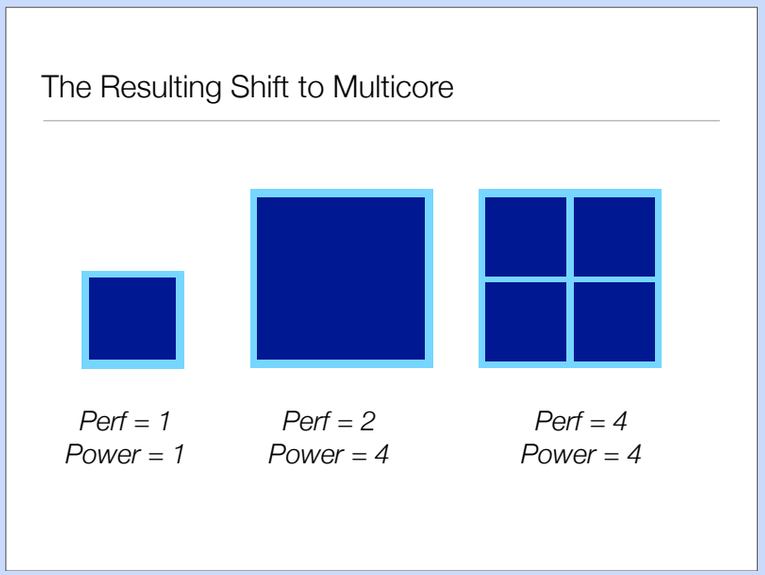
Models of uniprocessor performance and power

- Must be extremely simple, but also scientifically sound
- Performance proportional to SQRT(Area)
- Power proportional to Area



Comparison of abstract processor scenarios

- Assume a base processor with unit performance and area
- Given 4x more transistors you have two options
  - Massive unicore with Power=4, Performance=2
  - 4way multicore with Power=4, Performance=4
- From efficiency standpoint multicores make more sense than massive cores
- However (bit caveat) performance relies on there being software to exploit it



Evaluation

- Evaluation primarily conducted via comparison to prior years offerings.
- Important concepts in parallelism that were emphasized in the first half of the semester deliberately tested on the midterm exam (example below).
- Midterm student survey indicated that despite a very full syllabus, addition of new material was acceptable thanks to integration with existing syllabus. I.e., One cannot bolt new topics on to crowded courses, but shifted emphasis on topics already integrated seems to be working OK.

Dynamic circuit and timing behavior

Students were asked to (and successfully did) analyze the timing characteristics of an ALU containing four parallel computational paths.

2. **(10 points)** Design a 32-bit ALU (arithmetic logic unit) which computes either $+$, $-$, $\times$, $\div$ of two 32-bit values A and B putting the result on 32-bit output C. You may assume the presence of existing a 32-bit adder, subtractor, multiplier, and divider, and you may incorporate them in your design as black boxes. The particular operation is controlled by an additional two bit input S, where

   NB: In addition to the arithmetic modules, you may incorporate muxes, decoders, and other standard components as you find helpful.

   - if $S = 00$, $C = A + B$
   - if $S = 01$, $C = A - B$
   - if $S = 10$, $C = A \times B$
   - if $S = 11$, $C = A \div B$

   (a) Draw a schematic for your ALU design.

   (b) Give an expression for the propagation delay (TP_{ALU}) and contamination delay (TC_{ALU}) of your ALU, in terms of the propagation and contamination delays of the sub-modules. For example, TP_+ is the propagation delay of the adder, and TC_{MUX} is the contamination delay of a mux.