EE282H: Computer Architecture and Organization

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

http://www-leland.stanford.edu/class/ee282h/

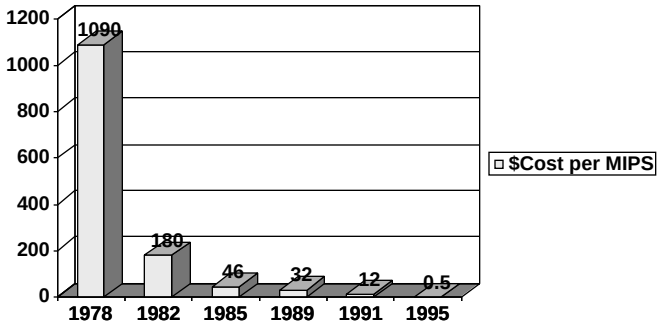
1

EE282H: Computer Architecture and Organization -- Course Overview

- Goals
  - » Understand how computers are organized and why they are organized that way?
  - » Understand and evaluate design trade-offs
- Outline
  - » Performance vs. cost evaluation measures
  - » Instruction set design: principles and examples
  - » Pipelining: basic and advanced
  - » Memory hierarchies: caches to bridge the processor/memory speed gap, virtual memory for the application/hardware gap
  - » I/O systems, esp. disk subsystem
- Theme: processor design using cost/performance as the basis

2

Faster and Cheaper



Today's microprocessors allow anyone to have supercomputer class performance.

3

Required Background: You should...

- know the basics of internal computer organization
  - » EE182 at Stanford, or the equivalent course or experience
  - » Breeze through the quiz
- understand basic logic design of synchronous systems
  - » Data path and control
  - » Gates, flip-flops, registers
  - » Clocking
  - » Multiplexers, ALUs
  - » state machines for control
- be familiar with assembly-language programming
  - » any mainframe or microprocessor

4

The textbook

- "Computer Architecture -- A Quantitative Approach", by John Hennessy (Stanford) and David Patterson (UC Berkeley)
  - » We will be using the *Second Edition*
- Chapter organization:
  - – Introduction
  - – Principles
  - – "Crosscutting issues": interaction of ideas between chapters
  - – "Putting it all together": real examples
  - – "Fallacies and pitfalls": what's wrong and what doesn't work
  - – History
- Do the reading before class

5

Course Mechanics

- Assignments
  - » Bi-weekly problem sets (4 or 5)
  - » Three programming assignments (2 in Verilog, 1 in C)
  - » You should work in pairs
  - » Late submission policy: 2 free late days for the quarter, 20% per day per assignment after that
- Exams
  - » Midterm: Nov 3, 7-9pm, on chapters 1-3
  - » Final: Dec 20, 7-10pm
- Approximate grading
  - » 10% for the problems sets
  - » 35% for the programming assignments
  - » 25% for the midterm
  - » 30% for the final
- Review sessions: four or five during quarter
- Honor code

6

Needed: Graders

- Grade a fraction of homework sets
- Benefits
  - » No need to turn in homework
  - » Full credit for homework
  - » $8.50 an hour pay

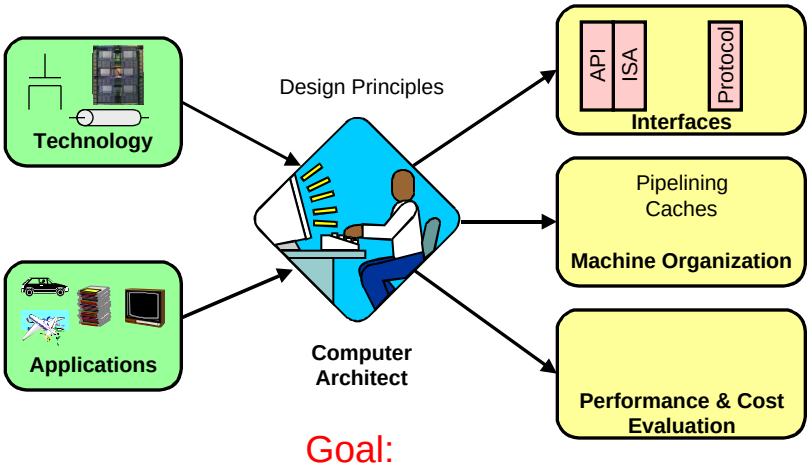
7

First Lecture

- What is computer architecture?
- Application requirements
- Technology constraints
- Performance
- HW/SW tradeoffs
- Optimizing the common case

8

What Does a Computer Architect Do?

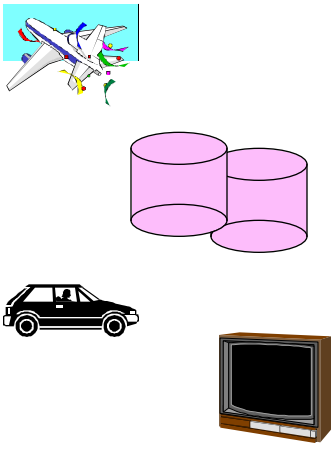


Goal:
Best Cost-Performance

9

Application Requirements

- Numerical simulations
  - floating-point performance
  - main memory bandwidth
- Transaction processing
  - I/Os per second
  - integer CPU performance
- Embedded control
  - Real-time performance
  - Low cost
- Media processing
  - low-precision ‘pixel’ arithmetic



10

Technology Impact

- Technology has a significant effect on architecture
- Technology trends:

	Density increase per year	Speed increase per year
IC Logic	60%	25%
DRAM	60% (4x every 3 yrs!)	10%
Disk	60%	10%

11

Technology Implications

- Think ahead of time about how the design will scale with advances in technology during its lifetime (ISAs may last 20 years e.g. x86)
- Processor/DRAM speed gap is a major challenge
- If it takes N years to design, target the technology and performance N years ahead: If 40%/year, in two years the competition will be 2x, so you should do better
- Shorten the design cycle

12

What is “performance”

- There are two ways to measure performance:
  - » Response time
    - – The time from start to completion of an event: the event’s latency
    - – Execution time vs. elapsed time -- what to count?
    - – Shared vs. dedicated system
  - » Throughput
    - – The amount of total work done per unit time: the bandwidth
    - – A better measure from the *system’s* point of view, but not from the *user’s*
- Example: A car wash starts a car every 30 seconds and holds 6 cars.
  - » Latency =
  - » Throughput =

13

Interface Design

- Software-software
  - » API (MS windows, unix)
- Hardware-software
  - » ISA (x86, MIPS)
- Hardware-hardware
  - » Bus (PCI, SCSI)
  - » Network protocol (Ethernet)

14

Hardware/Software tradeoffs

- Which operations are directly supported in “hardware” and which are synthesized in software

	Hardware	Software
Advantages	Speed, Consistency	Flexibility, easier/faster design (?), lower cost of errors
Disadvantages	Cost, Unchangeability	Slowness

- In reality, it’s a mixture of engineering, and aesthetics, and it’s hard to get right.

15

What counts the most: Make the common case fast

- Important, obvious, but overlooked!
- The frequent case is often simpler and can easily be done faster.
- Do the exceptional cases in software (eg: arithmetic overflow)
- A trap: making the common case slower to make a less common case faster (eg: Intel 432 (486 even?) procedure call), or to generalize
- Added benefit: easier, so faster design time
- Added benefit: flexibility of the software-handled cases
- The hardware should provide fast primitives, not complete solutions.

16

Making the common case fast: An example

- Say you could speed up floating point divide 8x, or floating point multiply 2x. Which is better?
- Measure applications! Suppose you find that:
  - » 30% of execution time is currently FP multiplication
  - » 8% of execution time is currently FP division
- If you speed up multiplication,
  - » 30% becomes 15%
  - » So new execution time is 85% of old
  - » So new hardware is 18% faster (not 15%!)
- If you speed up division,
  - » 8% becomes 1%
  - » So new execution time is 93% of old
  - » So new hardware is only 7% faster
- (All this assumes that the designs are correct. What about the risks inherent in clever designs?)

17

The limiting factor: Amdahl's Law

- The performance enhancement of an improvement is limited by how much the improved feature is used.
- In other words: Don't expect an enhancement proportional to how much you enhanced something. Quantitatively,

$$\text{Executiontime}_{\text{new}} = \text{Executiontime}_{\text{unaffacted}} + \frac{\text{Executiontime}_{\text{affected}}}{\text{Speedup}}$$

$$\text{Executiontime}_{\text{new}} = \text{Executiontime}_{\text{old}} \cdot \left(1 - \text{fraction}_{\text{enh}}\right) + \frac{\text{fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Executiontime}_{\text{old}}}{\text{Executiontime}_{\text{new}}}$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{fraction}_{\text{enh}}\right) + \frac{\text{fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}}$$

18

Example: Speedup using parallel processors

- Suppose an application is "almost all" parallel: 90%.
  What is the speedup using 10, 100, and 1000 processors?

Number of processors = P

$\text{fraction}_{\text{enhanced}} = 0.9$

$$Speedup_P = \frac{1}{0.1 + \frac{0.9}{P}}$$

$$Speedup_{10} = \frac{1}{0.1 + \frac{0.9}{10}} = \frac{1}{0.19} = 5.3$$

$$Speedup_{100} = \frac{1}{0.1 + \frac{0.9}{100}} = \frac{1}{0.109} = 9.1$$

$$Speedup_{1000} = 9.9$$

- (Not what the marketing department expected)

19

Moral: The common case can change

Number of processors = P

$\text{fraction}_{\text{enhanced}} = 0.9$

$$\text{Speedup}_P = \frac{1}{0.1 + \frac{0.9}{P}}$$

$$\text{Speedup}_{100} = \frac{1}{0.1 + \frac{0.9}{100}} = \frac{1}{0.109} = 9.1$$

Now instead, speedup the non-parallel part by a factor of 2:

$$\text{Speedup}_{100,2} = \frac{1}{\frac{0.1}{2} + \frac{0.9}{100}} = \frac{1}{0.05 + 0.009} = 16.95$$

(You get to keep your job)

20