

---

## EE282 Computer Systems Architecture

Christos Kozyrakis

Department of Electrical Engineering  
Stanford University

<http://eeclass.stanford.edu/ee282>

- 
- Instructor: Christos Kozyrakis, assistant professor, EE and CS
    - <http://csl.stanford.edu/~christos>
  - Teaching assistants:
    - Dawson Wong ([djwong@stanford.edu](mailto:djwong@stanford.edu))
    - Jacob Leverich ([leverich@stanford.edu](mailto:leverich@stanford.edu))
      - Jacob will focus only on infrastructure for the programming assignments
  - Administrative support: Teresa Lynn
  - Contact info & office hours on class information sheet
    - Up-to-date info on class webpage
    - <http://eeclass.stanford.edu/ee282>

---

## Class Basics

- 
- Lectures: Mo & Wed, 11am – 12.15pm, Skilling Auditorium
    - Also televised and on the web, but come to class & ask questions
    - There will also be ~5 discussion sessions
      - Fridays 11 – 11.50am, location TBA
  - Web page: <http://eeclass.stanford.edu/ee282>
    - Announcements, handouts, office hours, latest schedule, bulletin board
    - Check regularly
    - Signup with webpage for on-line access to grades
      - We will let you know when registration is open...
  - Read the information sheet for details

---

## The Class Bulletin Board

- 
- Available at <http://eeclass.stanford.edu/ee282>
  - The preferred way to ask class-related questions
    - We promise to check & answer very often, especially close to deadlines
    - We will post answer to newsgroup and email a copy to you
  - The rules
    - Before posting a new question
      - Check if question has already been asked or even answered
      - Check the FAQ page for the assignment
    - Choose an appropriate subject for your question
      - E.g. "HW2, problem 3, definition of memory latency"
  - For questions not appropriate for the public: send us an email
    - We do encourage critical feedback
    - The earlier you tell us the more likely we'll address it

## Textbook and Prerequisite

- “*Computer Architecture: A Quantitative Approach*”, 4th Edition, by John Hennessy & David Patterson
  - Do not buy/use the 3<sup>rd</sup> or 2<sup>nd</sup> edition!
  - We will also use a small number of papers and other handouts
- Prerequisite: EE108b or EE182 or equivalent
  - You should know
    - machine organization, basic pipelining & caching, digital logic, assembly programming, basic IO concepts, virtual memory, simple OS concepts
  - If you don't have the prerequisite, this class will likely be too hard for you
    - It will be difficult to catch up and you will likely get a bad grade...
  - If you need to, refresh your memory check the EE108b notes
    - 1st discussion session on Friday will quickly review prerequisite material
- EE282 is offered again in Fall 2009

## Quizzes

- **Quiz 1:** Wed 10/22<sup>nd</sup>, 6pm – 9pm, place TBA
- **Quiz 2:** Mo 12/10<sup>th</sup>, 8.30am – 11.30am, place TBA
- Typical quiz rules
  - Closed books, 1 page of notes, calculator
  - Local SCPD students must come to campus
- Alternative quiz days/time
  - Only for legitimate conflicts with other Stanford class or documented health problems, death in the family, etc
  - If you cannot make it, do not take the class...
- If you have a disability
  - Contact the Disability Resource Center (DRC) asap

## Homework and Programming Assignments

- 3 homework assignments
- 2 programming assignments
  - Optimize the performance of a program on a modern server
  - Develop a cluster application using MapReduce
- Policies (in addition to Stanford Honor Code)
  - You will work in groups of 3 students
  - All deadlines are final, no extensions, no exceptions
    - If you cannot make it, don't take the class...
  - Regrading requests within one week from original grade
- Grade breakdown (tentative)
  - Quizzes ~45%, homework 15%, assignments 40%
  - Uniform grading rules for all of you (grad/ugrad/SCPD)

## EE282 Lecture 1: Computer Architecture Fundamentals

Department of Electrical Engineering  
Stanford University

<http://eeclass.stanford.edu/ee282>

## Today's Menu

- Class goals and overview
- What is a System and what is Computer Architecture
- Application & technology trends
- Basic architecture techniques and metrics

## EE282 Goals

- Our focus: processor-based digital systems
- Our goals
  - Understand *how* and *why* they are organized the way they are
  - Study common issues and advanced solutions
    - Applicable to systems from data-centers to deeply embedded devices
  - Understand interactions with technology and applications
  - Understand interactions between hardware and software
  - Get practical experience using systems efficiently
    - How to make use of memory hierarchy
    - How to make use of parallelism

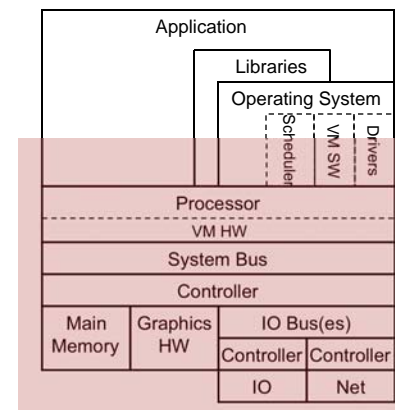
## Processor-based Digital Systems

- Systems with a programmable, general-purpose processor
  - Advantages ??
- Computers are the canonical example
  - PCs, laptops, workstations, ...
- However, most processors are embedded or in servers
  - Game consoles, PDAs, cell phones, ...
  - Printers, car electronics system, ...
  - Web servers, database servers, ...



## Overall System Architecture

- Multiple interacting layers
  - Term “architecture” used with all of them
- This class focuses on
  - Hardware architecture
    - Memory hierarchy and I/O
    - Clusters
    - Reliability & energy-efficiency
  - Hardware-software interaction
    - Programming for performance
    - OS support & virtual machines
    - Cluster programming



## System Architecture Vs Processor Microarchitecture

- EE282 does not cover processor architecture!
  - No coverage of superscalar, multi-core, multithreading, ...
  - You know the basics from EE108b, learn more in EE382A, CS315A, EE382C, ...
- Rational
  - System architecture is becoming the main differentiating factor
    - Few processor vendors, many systems vendors or service providers...
  - Cluster are widespread; you should know how to build & use them
  - Your own needs
    - Few of you will build processors, many will build systems
    - Most of you will have to understand and use systems efficiently

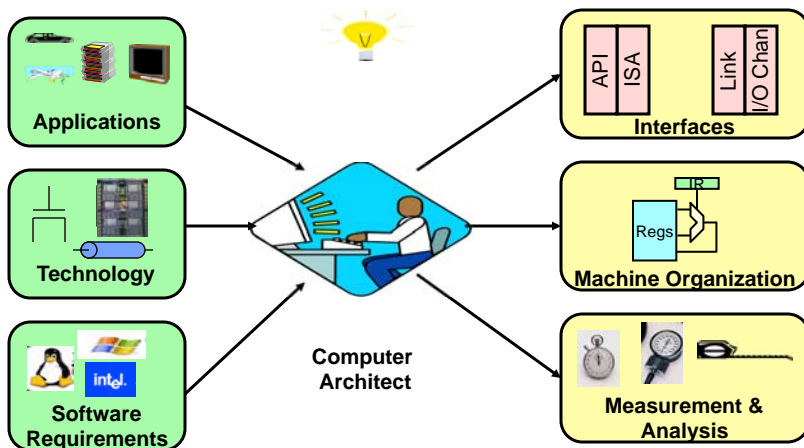
## EE282 Schedule

- Introduction
- Statistics for architects
- Advanced caching (x2)
- Main memory system
- Software optimizations for memory
- Advanced IO systems
- System software support
- Parallel architecture basics
- Cluster architecture
- MapReduce programming
- Cluster software
- Virtual machines (+guest)
- Reliability (x2, + guest)
- Energy efficiency (x2, + guest)
- Class summary

Quiz 1

Quiz 2

## What do Computer Architects Do?



The science/art of constructing efficient systems for computing tasks

## Applications

## Application: Constraints & Opportunities

- Applications drive machine 'balance'
  - Scientific computations
    - Floating-point performance
    - Main memory bandwidth
  - Transaction/web processing
    - ??
  - Multimedia processing
    - ??
  - Embedded control
    - ??



Architecture concepts typically exploit application behavior

## Applications Change over Time

- Data-sets & memory requirements → larger
  - Cache & memory architecture become more critical
- Single task → multiple tasks
  - Parallel architectures become critical
- Standalone → networked
  - IO integration & system software become more critical
- Limited IO requirements → rich IO requirements
  - 60s: tapes & punch cards
  - 70s: character oriented displays
  - 80s: video displays, audio, hard disks
  - 90s: 3D graphics; networking, high-quality audio
  - 00s: software as a service, online gaming, ...

## Application Properties to Exploit in Computer Design

- Locality in memory and IO references
  - A subset of code & data are actively used at any time
  - Temporal and spatial locality
- Parallelism
  - Data-level (DLP): same operation on sequences of elements
  - Instruction-level (ILP): independent instructions within sequential code
  - Thread-level (TLP): independent tasks within one program
  - Multi-programming: multiple independent programs
- Predictability
  - Control-flow direction, memory references, data values

## Technology

## Key Technology Trends

- Semiconductors (processor & memory chips)
  - Moore's Law: # of devices per chip doubles every ~1.5 years
  - Device speed improves by ~15% per year
  - Long on-chip wires get worse
- Storage (hard disks)
  - Disk capacity increases by 100x per decade
  - Disk throughput improves by 10x per decade
  - Disk cost/bit is 100x better than that of DRAM
- Network
  - Gilder's law: Deployed bandwidth triples every year
  - Link bandwidth improves 4x every 3 years
- Implications?

## Challenges and Opportunities

- Latency lags bandwidth
  - In the time that that bandwidth doubles, latency improves by 1.2 to 1.4x
- Non-volatile memory (e.g. Flash)
  - Close to disk cost/bit, close to DRAM latency
- Energy and power limitations
  - Cannot supply energy or cool a chip if all transistors are active
- Implications?

## Historical Perspective: Changes in Technology lead to Changes in Architecture

- 1970s
  - Multi-chip CPUs
  - Semiconductor memory very expensive
  - Complex instruction sets (good code density)
  - Microcoded control
- 1980s
  - 5K – 500 K transistors
  - Single-chip, pipelined CPUs
  - On-chip memory possible
  - Simple, hard-wired control
  - Simple instruction sets
  - Small on-chip caches
- 1990s
  - 1 M - 64M transistors, 64b CPUs
  - Complex control to exploit instruction-level parallelism
  - Deep pipelines
  - Multi-level caches
- 2000s
  - 100 M - 5 B transistors
  - Slow wires, power consumption, design, complexity, memory latency, IO bottlenecks, ...
  - Multicore & heterogeneous systems
  - Support & programming for parallelism?
  - <<your Ph.D. thesis goes here>>

Keeps computer architecture interesting and challenging

## Metrics, Tools, and Techniques

## Metrics of Efficiency

- Desktop computing (\$500 - \$3K)
  - Metrics: ??
  - Prominent processors: Intel Pentium, AMD Athlon, PowerPC G5
- Server computing (\$3K - \$1M)
  - Metrics: ??
  - Prominent processors: IBM Power5, Sun UltraSparc, AMD Opteron
- Embedded computing (\$10 - \$500)
  - Metrics: ??
  - Prominent processors: ARM, MIPS, Motorola 68K, many others

Diversity in requirements leads to diversity in architectures

## Performance Metrics

- Latency or execution time or response time
  - Wall-clock time to complete a task
  - Important if all we have to run is a single or a time-critical time to run
- Bandwidth or throughput or execution rate
  - Number of tasks completed per unit of time
    - $\text{Bandwidth} = \frac{\text{total amount of work}}{\text{total execution time}}$
  - Metric is independent of exact number of tasks executed
  - Important when we have many tasks to run

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

## Examples

- Latency metric: program execution time in seconds

$$\begin{aligned}
 CPUtime &= \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Cycles}}{\text{Program}} \cdot \frac{\text{Seconds}}{\text{Cycle}} \\
 &= \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Seconds}}{\text{Cycle}} \\
 &= IC \cdot CPI \cdot CCT
 \end{aligned}$$

- Your system architecture can affect all of them
  - CPI: memory latency, IO latency, ...
  - CCT: cache organization, ...
  - IC: OS overhead, compiler choice ...
- Bandwidth metrics:
  - Network bandwidth: 1 Gb/s ethernet
  - Database server throughput:  $10^6$  transactions/sec

## A is Faster than B?

- Given the CPUtime for machines A and B, A is X times faster than B means:

$$X = \frac{CPUtime_B}{CPUtime_A}$$

- Example, CPUtimeA=3.4sec & CPUtimeB=5.3sec then
  - A is  $5.3/3.4=1.55$  times faster than B or 55% faster
- If you start with bandwidth metrics of performance, use inverse ratio

$$X = \frac{BandWidth_A}{BandWidth_B}$$

## Speedup and Amdahl's Law

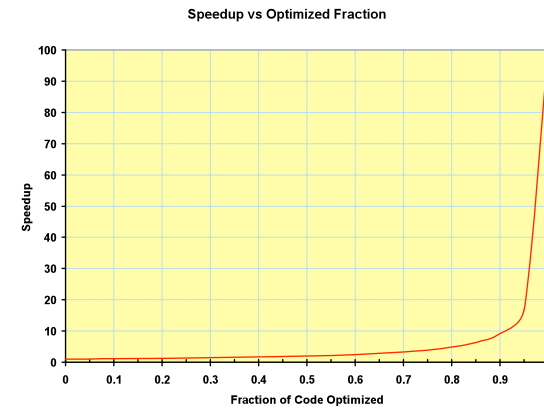
- Speedup = CPUtime<sub>old</sub> / CPUtime<sub>new</sub>
- Given an optimization x that accelerates fraction f<sub>x</sub> of program by a factor of S<sub>x</sub>, how much is the overall speedup?

$$Speedup = \frac{CPUtime_{old}}{CPUtime_{new}} = \frac{CPUtime_{old}}{CPUtime_{old}[(1-f_x) + \frac{f_x}{S_x}]} = \frac{1}{(1-f_x) + \frac{f_x}{S_x}}$$

- Lesson's from Amdahl's law
  - Make common cases fast: as f<sub>x</sub>→1, speedup→S<sub>x</sub>
  - But don't overoptimize common case: as S<sub>x</sub>→∞, speedup→1 / (1-f<sub>x</sub>)
    - Speedup is limited by the fraction of the code that can be accelerated
    - Uncommon case will eventually become the common one
  - Amdahl's law applies on other metrics too: cost, power consumption, ...

## Amdahl's Law Example

- If S<sub>x</sub>=100, what is the overall speedup as a function of f<sub>x</sub>?



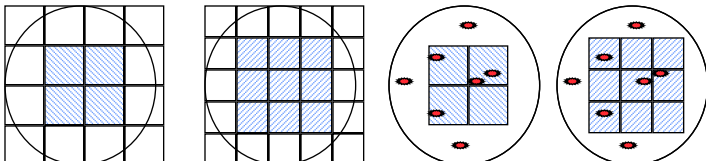
## Digital System Cost

- For semiconductor chips, cost = f(area<sup>4</sup>)

$$IC\ cost = \frac{Die\ cost + Testing\ cost + Packaging\ cost}{Final\ test\ yield}$$

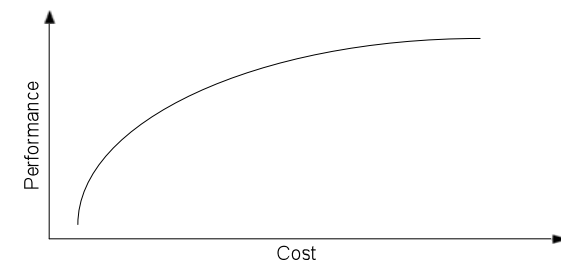
$$Die\ cost = \frac{Wafer\ cost}{Dies\ per\ Wafer \times Die\ yield}$$

$$Die\ Yield = Wafer\_yield \times \left\{ 1 + \left( \frac{Defect\_Density \times Die\_area}{\alpha} \right)^{-\alpha} \right\}$$



## Cost – Performance Tradeoff

- The trade-off
  - Chip cost is primarily a function of die area<sup>4</sup>
  - But bigger dies provide more resources for higher performance
- The goal of a good architect
  - Find the knee of the performance-cost curve OR
  - Get maximum performance for a fixed cost target



## Other Cost Contributors

---

- Testing cost:  $\text{cost/die} = (\text{cost/hour} \times \text{test time}) / \text{yield}$ 
  - Could be \$10-\$20 or more for complex chips
- IC Packaging
  - Depends on die size, number of pins, and power dissipation
- Cost of cooling system
  - <2W no heat-sink, <10W no fan, >100+W liquid/spray cooling
- And most of all, do not forget VOLUME
  - Cost of a modern IC fabrication facility: >\$2B
  - Cost of a set of masks for a wafer: \$0.5M - \$1M
  - Design NRE cost: often ~\$10M
  - Need volume to amortize all this cost...
- And then there is total cost of ownership (TCO)
  - More on this later

## Key Architecture Techniques

---

## Key Architecture Techniques (1)

---

- Pipelining
  - What is it and what does it work?
  - Does it improve latency or bandwidth?
- Parallelism processing
  - What makes it possible?
  - What should be aware of?
- Out-of-order execution
  - In what order should you execute the instructions in a program?
- Speculation
  - Why would speculation be useful?

## Key Architecture Techniques (2)

---

- Caching
  - Why do caches reduce memory access latency?
  - Are caches applicable only to processor design?
- Indirection
  - How do you find your doctor's number?
  - Examples in system design?
- Amortization
  - How can you amortize the high cost of memory accesses?
  - Other examples?

# Summary

---

- Computer architecture:
  - Design of efficient systems given the requirements of applications and the capabilities/constraints of technology
  - Need to look a few years ahead with both applications & technology
- Applications
  - Look for locality, parallelism, and predictability
- Technology
  - Dealing with latency, power, and reliability are the upcoming challenges
- Efficiency metrics
  - Performance (latency & bandwidth), cost, power, reliability, ...
- Key architecture techniques for the rest of the quarter
  - Pipelining, parallel processing, out-of-order execution, speculation, caching, indirection, amortization