CS 211: Computer Architecture

Instructor: Prof. Bhagi Narahari
Dept. of Computer Science
Course URL: www.seas.gwu.edu/~narahari/cs211/

Computer Architecture – Course Objectives

- Examine the role of computer architecture (CA) in system/program performance
  - What are the key components of CA ?
  - What are the architectures of today's processors ?
  - What aspects of architecture design affect performance of application and how ?
  - How to extract max performance out of today's CAs ?
  - Role of software in architecture performance
  - What are the emerging trends in CA ?
- quantitative approach to CA

What it is not..

- What the course is not
  - Detailed exposition on hardware design
  - Semiconductor technology details
  - Case studies
  - How to assemble/buy a new computer

Perspective

- Computer architecture design is directly linked to underlying technology
  - Semiconductor
  - Compiler technology
  - Computational models
- Goal of software designers is to run an application program efficiently on the architecture
  - Compiler plays a key role
  - interplay between architecture features and application program properties
  - Bottom line is performance of application

Let's look at Architecture Trends, Technologies

- Interplay between hardware and software
- Implications of technology trends on emerging architecture designs

Today

- What is Computer Architecture
  - Architecture levels and our focus
- Technology Trends
  - Summary of what has happened in CA
    - Hardware performance trends and designs
  - Impact of current trends on new designs
- Performance models
  - What to measure and how
  - Models linking hardware and software
  - Thumb rules for CA design
- Read Chapter 1

An Important Idea: what are Computers meant to do ?

- We will be solving problems that are describable in English (or Greek or French or Hindi or Chinese or ...) and using a box filled with electrons and magnetism to accomplish the task.
  - This is accomplished using a system of well defined (sometimes) transformations that have been developed over the last 50+ years.
  - As a whole the process is complex, examined individually the steps are simple and straightforward

Hardware Vs. Software

Hardware

- Medium to compute functions

Software

- Functions to compute

Computational Model connects them

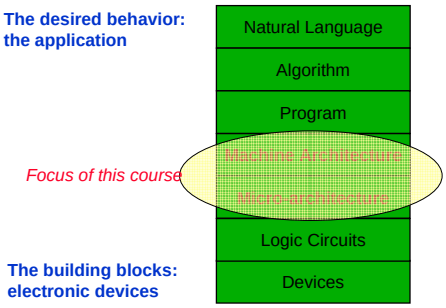
Two pillars of Computing

- **Universal Computational Devices**
  - Given enough time and memory, all computers are capable of computing exactly the same things (irrespective of speed, size or cost).
    - Turing's Thesis: every computation can be performed by some "Turing Machine" - a theoretical universal computational device
- **Problem Transformation**
  - The ultimate objective is to transform a problem expressed in natural language into electrons running around a circuit!
    - That's what Computer Science and Computer Engineering are all about: a continuum that embraces software & hardware.
    - Note the role of compilers/translators

Making the Electrons Work

- Problems
  - application expressed in a natural language
  - Find the quickest way to get from Network Node A to Node B
- Algorithms to solve the problem
  - Djikstra's shortest path algorithm
- Programming Language to implement algo
  - Program is the output of this state
  - C program with relevant data structures
- Machine (ISA) Architecture
  - describes functions/capability of the HW
  - IA-32 architecture (Pentium)
- Microarchitecture
  - how is the ISA implemented on the chip
  - Pipelined units, superscalar processor
- Circuits
  - Basic building blocks – gates, buses
- Devices
  - Transistors, semiconductor principles

Problem Transformation - levels of abstraction



The Machine Level - 1

- **Machine Architecture**
  - This is the formal specification of all the functions a particular machine can carry out, known as the ***Instruction Set Architecture*** **(ISA).**
    - We focus on the ISA level
- **Microarchitecture**
  - The implementation of the ISA in a specific CPU - i.e. the way in which the specifications of the ISA are actually carried out.
    - We will touch on some aspects of this level to examine how ISA solutions are implemented … pre-req material

The Machine Level - 2

- Logic Circuits
  - Each functional component of the microarchitecture is built up of circuits that make "decisions" based on simple rules
    - Not the focus of this course – prerequisite material
- Devices
  - Finally, each logic circuit is actually built of electronic devices such as CMOS or NMOS or GaAs (etc.) transistors.
    - Device electronics – not in this course

Alternate Definitions: The Multi-Level Concept

- Different levels, each with its unique functionality
  - Problem-Oriented Language Level (prog languages)
  - Assembly Language Level
  - Operating System machine level
  - Conventional Machine Level (Instruction Set Architecture -- ISA)
  - Micro-architecture level (Microprogramming level)
  - Digital Logic Level (program in VHDL, Verilog)
    - Device & Semiconductor Level

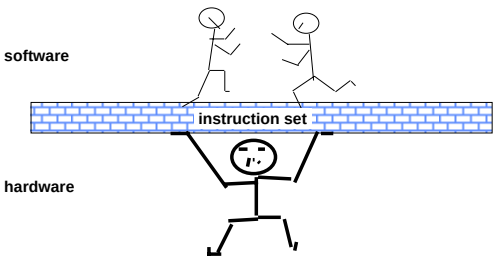
For us, Computer Architecture is ...

Instruction Set Architecture

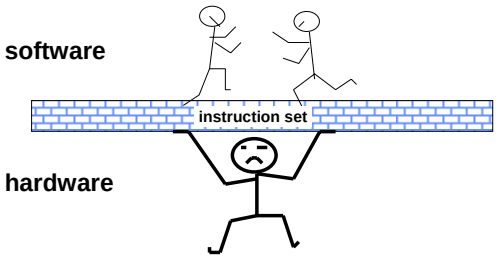
Organization (MicroArchitecture)

(Logic Circuits) Hardware

Instruction Set Architecture (ISA)



The hardware/software interface: Instruction Set Architecture (ISA)



Which is easier to change/design???

The Backdrop: Users

- Who will program these machines?
  - Programmers
- What do they expect?
  - Performance
  - Correctness
- How?
  - Write HLL program and Compile
- Compilation is key to performance
  - Requires Hardware/Software interaction at ISA level
  - Knowledge of architecture, application, algorithm

Architecture: Introduction

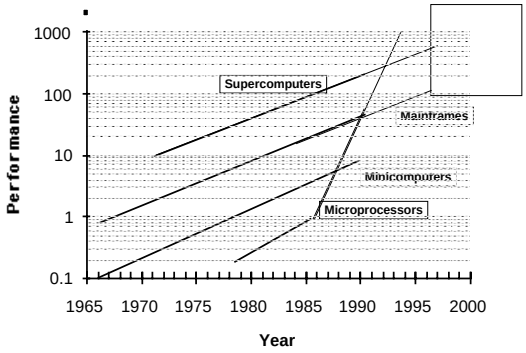
- What is Computer Architecture
  - Architecture levels and our focus
- Technology Trends
  - Summary of what has happened in CA
    - Hardware performance trends and designs
  - Impact of current trends on new designs
- Performance models
  - What to measure and how
  - Models linking hardware and software
  - Thumb rules for CA design

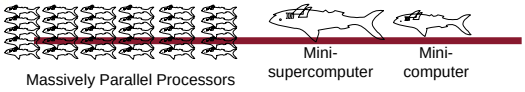
Trends In Technology, Applications,Architectures

Performance: Original Food Chain Picture

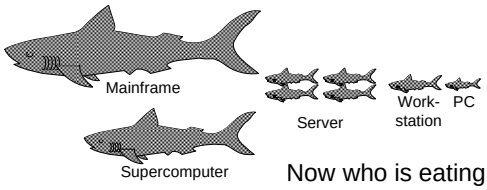
Big Fishes Eating Little Fishes

Processor Performance Trends





1998 Computer Food Chain: Cost/Performance



Now who is eating whom?

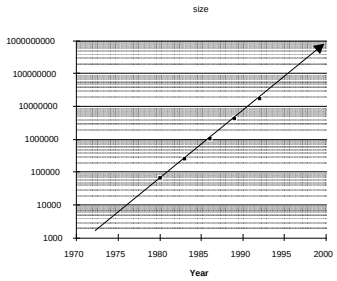
Computer Architecture: Over the years

- Microprocessors today (Intel, PowerPC,etc.) faster than first Cray supercomputer CRAY-1
- ENIAC filled a room, MicroProc today fit on palm
- Big increase in functionality
  - "old" days, one had to buy separate Math co-processor for Intel PCs
  - Now, even separate special purpose engines (graphics co-proc., network proc. etc.) are standard

Why Such Change?

- **Performance**
  - **Technology Advances- Moore's Law**
    - **CMOS VLSI dominates older technologies (TTL, ECL) in cost *AND* performance and is progressing rapidly**
  - **Computer architecture advances improves low-end**
    - **RISC, superscalar, RAID, ...**
- **Price: Lower costs due to …**
  - **Simpler development, volumes, lower margins**
- **Function**
  - **Rise of networking/local interconnection technology**

Memory Capacity (Single Chip DRAM)



year	size(Mb)	cyc time
1980	0.0625	250 ns
1983	0.25	220 ns
1986	1	190 ns
1989	4	165 ns
1992	16	145 ns
1996	64	120 ns
2000	256	100 ns

Technology Trends summary

	Capacity	Speed (latency)
Logic	2x in 2 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years

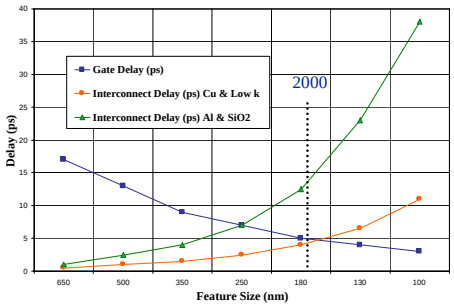
Performance Trends: Summary

- **Workstation performance (measured in Spec Marks) improves roughly 50% per year (2X every 18 months)**
- **Improvement in cost performance estimated at 70% per year**

Emerging trends in Processor Design

- CISC to RISC
  - Based on speeding up common instructions
    - Shall return to this later
- What's the trend in Semiconductor technology and its impact on new types of processor architectures ?
  - some aspects to consider:
    - Delay: switching time of transistor – impacts clock cycle
    - Feature size: size of transistor – impacts amount of logic in processor
    - Interconnect delay: clock cycle/delay in sending signal across the interconnect lines on a chip

Delay vs. Feature Size



Bohr, M. T., "Interconnect Scaling - The Real Limiter To High Performance ULSI", Proceedings of the IEEE International Electron Devices, pages 241-242.

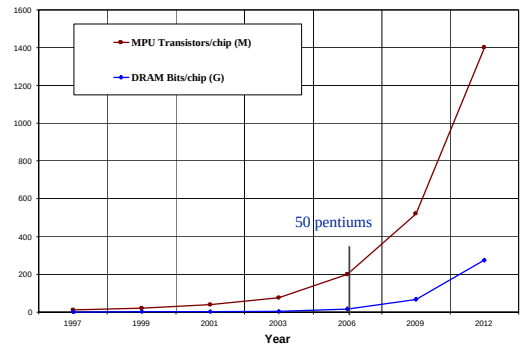
As Wire Delays Become Significant..

- Focus on architectures that
  - do not involve long distance communication
  - distribute control and data processing logic

Verification And Test

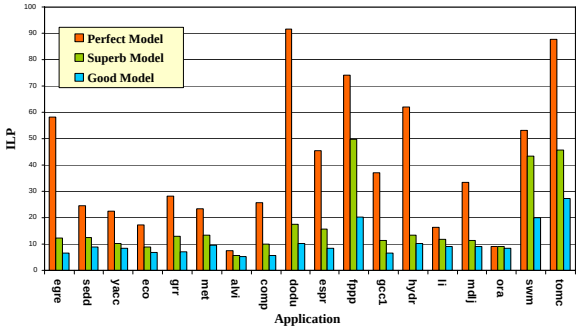
- With increasing chip complexity, verification and test costs form a significant component of the overall cost
- Long testing process will also affect time to market
- Impact of high costs ?
  - Keep architecture simple and regular

Transistors / Chip



Available instruction-level parallelism

[Wall'93, DECWRL]



From Previous Slides...

- **Lots of hardware parallelism available**
  - **can accommodate approx. 50 pentiums on one die in few years**

However,

- **Conventional architectures and compilation**
  - **cannot expose enough parallelism in applications**
  - **even the "superb" model yields an ILP < 10 on average**

- **Need for new architectures and compilation techniques!**

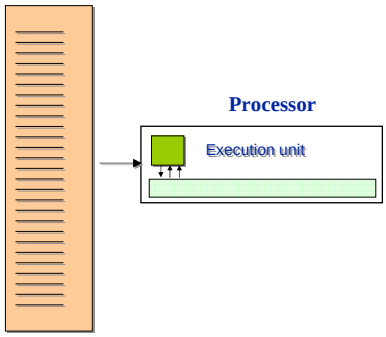
Current Architecture Designs

- **Reconfigurable Processors—better for special purpose applications**
  - **let compiler handle everything**
  - **no commitment to a particular architecture**
  - **compiler generates architecture and code for it**
  - **Example: FPGA based processors**

- **ILP Architecture: instruction level parallelism**
  - **Superscalar**
  - **Explicitly Controlled Architectures (Very Large Instruction Word - VLIW)**
    - **simplify architectures as much as possible**
    - **compiler handles a lot of processor's decision making**
      - **explicitly control issue, scheduling, allocation**
    - **Explicitly Parallel Instruction Computing (EPIC)- Intel's IA-64, Itanium**
- **Multi-Core Processors**
  - **The ILP "wall": ILP processors cannot expose enough parallelism..**
  - **so move to multi-threaded/multiprocessor on chip**

Sequential Processor

Sequential Instructions

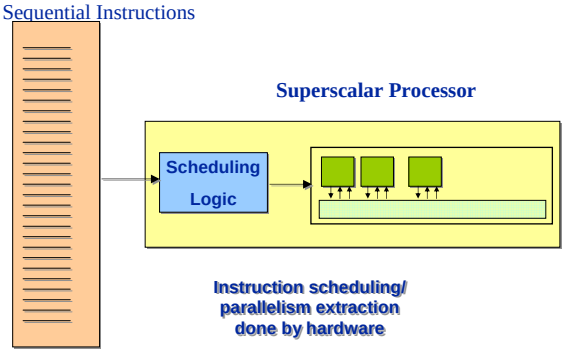


Instruction Level Parallelism: Shrinking of the Parallel Processor

- Put multiple processors into one chip
- execute multiple instructions in each cycle
- move from multiple processor architectures to multiple issue processors
- Two classes of Instruction Level Parallel (ILP) processors
  - Superscalar processors
  - Explicitly Parallel Instruction Computers (EPIC)
    - also known as Very Large Ins Word (VLIW)

ILP Processors:Superscalar

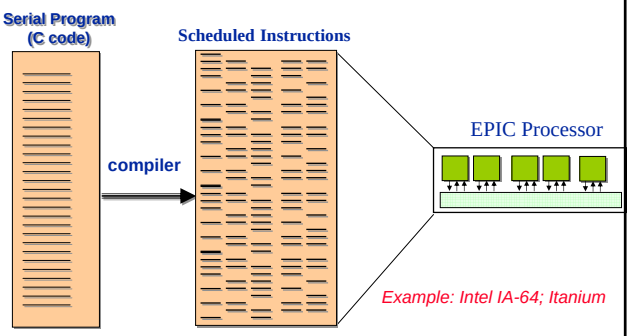
Sequential Instructions



Instruction scheduling/ parallelism extraction done by hardware

Example: Intel IA-32/Pentium

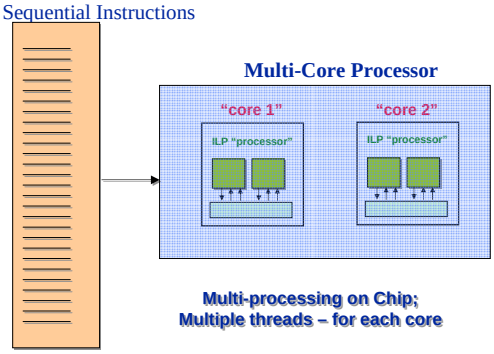
ILP Processors:EPIC/VLIW



Example: Intel IA-64; Itanium

Multi-Core Processors

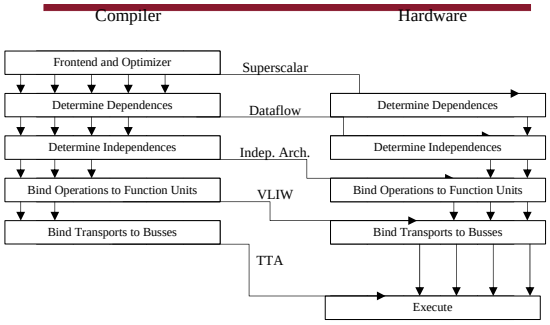
Sequential Instructions



Multi-processing on Chip;
Multiple threads – for each core

Example: Intel Core 2 Duo

Who is doing what: Compiler vs. Processor



B. Ramakrishna Rau and Joseph A. Fisher. Instruction-level parallel: History overview, and perspective. The Journal of Supercomputing, 7(1-2):9-50, May 1993.

Importance of Compilers in ILPArchitectures

- Role of compiler more important than ever
  - optimize code
  - analyze dependencies between instructions
  - extract parallelism
  - schedule code onto processors
  - EPIC processors does not have any hardware utilities for scheduling, conflict resolution etc.
    - has to be done by the compiler

Another aspect: Quantifying Power Consumption

- What else is an issue in processor/system design/performance
- Power consumption/heat dissipation
  - Limited energy source (battery) in embedded systems (or even laptops)
    - Apple switch to Intel chips in 2005 ?

Power Equation

$$P_{AVG} = \frac{1}{2} N_G f_{clk} C_L V_{DD}^2$$

- P_{AVG} - the average dynamic power consumed by the gates
- N_G - the number of gates that transition
  - This is usually dropped from the equation
- f_{clk} - the frequency of the system clock
- C_L - the average capacitive load per gate
- V_{DD} - the supply voltage

- For mobile devices, energy better metric

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

Define and quantify power

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*
- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5V to 1V
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Example of quantifying power

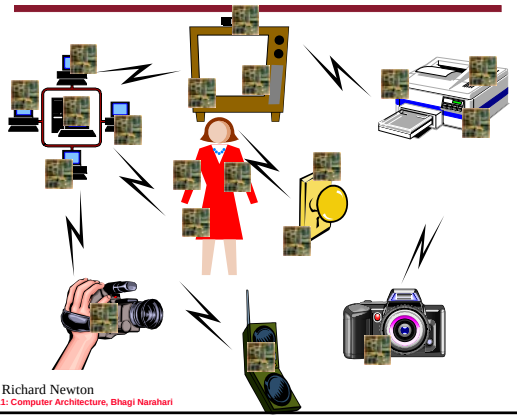
- Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?

$$\begin{aligned} Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched \\ &= 1/2 \times .85 \times CapacitiveLoad \times (.85 \times Voltage)^2 \times FrequencySwitched \\ &= (.85)^3 \times OldPower_{dynamic} \\ &\approx 0.6 \times OldPower_{dynamic} \end{aligned}$$

Power

- Because leakage current flows even when a transistor is off, now *static power* important too $Power_{static} = Current_{static} \times Voltage$
- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

What about the embedded processor ?



Source: Richard Newton

Summary: What's up with Architecture Trends ?

- Moore's law: density doubles every 18-24 months
  - smaller processors, faster clocks
  - leads to more powerful and smaller processors!
    - Small computing platforms like Palmtop computers, Palm, WinCE
- Trends/Lessons/Limits ?

Crossroads: Conventional Wisdom in Comp. Arch

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Xtors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)

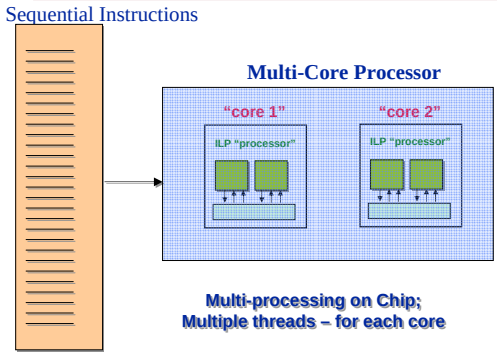
Conventional Wisdom…

- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)

- More simpler processors are more power efficient

Multi-Core Processors



Multi-processing on Chip;
Multiple threads – for each core

Example: Intel Core 2 Duo

Déjà vu all over again?

- Multiprocessors imminent in 1970s, '80s, '90s, …
- "… today's processors … are nearing an impasse as technologies approach the speed of light.."
  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature
  ⇒ Custom multiprocessors strove to lead uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"
  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies switch to multiprocessors (AMD, Intel, IBM, Sun; all new Apples 2 CPUs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: 1 to 2 CPUs

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip,
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved *without* participation of computer architects

Course Information

- Course materials placed at
  - www.seas.gwu.edu/~bhagiweb/cs211/
  - All lecture notes, homeworks, simulator s/w info, and announcements
  - Check at least once a week – before class.
  - Strong pre-requisite: CS135 or equivalent first course in Computer Organization/Systems
  - Programming skills and basic system skills

Course Information

- Textbook: Hennessy and Patterson, Computer Architecture: A quantitative approach; 4th Edition, Pub. Morgan Kauffman
  - If you have 3rd Edition that will work fine.
- course topic to book chapter mapping placed on website
- Website will contain lecture materials and homeworks, as well as references
- Homework & Project submissions will use Blackboard

Course Requirements

- Prerequisites: data structures, discrete math, computer organization
- Requirements:
  - Exams: 65%
    - Midterm and Final
  - Homework assignments: 10%
    - Work individually
  - Projects – 15%
    - Work in teams of 3 persons
    - Students *may* be permitted to
      - substitute term paper or project for some of the projects—will have to meet me before October 1.
      - Substitute different project for assigned project
  - Class discussions & presentations
    - Readings will be assigned to teams; present and lead discussion in class
- Academic Integrity Policy
  - Absolutely no collaboration of any kind on homeworks
    - No outside sources (people or content)
  - Programming projects can be done in 2-3 person teams – no collaboration between teams

Programming projects

- Projects require programming using Simple Scalar simulator
  - Some homeworks may also require use of this
  - Students placed into teams (3 person teams; 2 also allowed) for programming projects – team selection target date is October 1.
- www.simplescalar.com
- Objective of using Simplescalar
  - Connect concepts covered with 'real' implementations and study impact of architecture techniques on actual applications.
- Machines in Academic Center, 7th Floor Terminal Room 724.
  - Linux machines
  - Grad student (part-time TA) will cover this in office hours
- No regular TA for course

Course Outline

- Computer Organization Review – Mostly Self study
- Architecture challenges, design objectives, thumb rules, emerging issues
- (I) Processor architectures:
  - Instruction level parallel (ILP) processors
  - Pipelined, superscalar, and EPIC/VLIW..vector
  - Midterm – date to be decided…plan for 8th or 9th week
- (II) Components:
  - Compiler Optimization
  - Memory Design: cache optimizations
  - I/O system
- (III) Multi-core and Multiprocessors:
  - Multiprocessor Architectures overview
  - Introduction to Multi-core computing
- Other topics time permitting

Architecture: Introduction

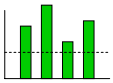
- What is Computer Architecture
  - Architecture levels and our focus
- Technology Trends
  - Summary of what has happened in CA
    - Hardware performance trends and designs
  - Impact of current trends on new designs
- **Performance models**
  - **What to measure and how**
  - **Models linking hardware and software**
  - **Thumb rules for CA design**

Recurring Theme

Performance

– Calculating & measuring performance

– Designing & tuning software



Performance

- **How do you measure performance?**
  - **Throughput**
    - **Number of tasks completed per time unit**
  - **Response time/latency**
    - **time taken to complete the task**
  - **metric chosen depends on user community**
    - **System admin vs single user submitting homework**

The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers	Performance ?
Boeing 747	6.5 hours	610 mph	470	
BAD/Sud Concodre	3 hours	1350 mph	132	

The Bottom Line: Performance (and Cost)

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

- Time to run the task (Execution Time/Response Time/Latency)
  - Time to travel from DC to Paris
- Tasks per unit time (Throughput/Bandwidth)
  - Passenger miles per hour; how many passengers transported per unit time

The Bottom Line: Performance (and Cost)

"X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

- Speed of Concorde vs. Boeing 747
- Throughput of Boeing 747 vs. Concorde

How to Model Performance

- What are we trying to model ?
  - Time taken to run an application program
- Why not just use "time" function in Unix?

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU = IC * CPI * Clk

Holy grail of CS 211 @

CPU time and Architecture Interplay

- 3 components to CPU time: IC, CPI, Clk
  - Factors that affect these components

	Inst. Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set	X	X	(X)
Organization		X	X
MicroArch		X	X
Technology			X

- Consider all three components when optimizing
- Workloads change!

CPI: Cycles per instruction

- **Depends on the instruction executed**
  - Can have different times for diff. inst.

CPI_i = Execution time of instruction i / Cycle time

- **Average cycles per instruction**

$$CPI = \sum_{i=1}^{n} CPI_i * F_i \quad \text{where } F_i = \frac{IC_i}{IC_{tot}}$$

- **Example:**

Op	Freq	Cycles	CPI_i	%time
ALU	50%	1	0.5	33%
Load	20%	2	0.4	27%
Store	10%	2	0.2	13%
Branch	20%	2	0.4	27%
		CPI_{total}	1.5	

Measurement Tools

- Benchmarks, Traces, Mixes
- Hardware: Cost, delay, area, power estimation
- Simulation (many levels)
  - ISA, RT, Gate, Circuit
- Queuing Theory
- Rules of Thumb
- Fundamental "Laws"/Principles

Measuring IC/CPI/Clk

- **Existing Processors**
  - IC: most processors have performance counters
  - CPI: calculate from IC, Clk, and execution time
  - Clk: known
- **New Designs**
  - IC: functional simulation or analyze static instructions
  - CPI: simple models or execution-driven simulation
  - Clk: estimate from simple structures or ??

Measure performance of what applications?

- CPU A versus CPU B
  - How to compare ?

Performance Evaluation

- "For better or worse, benchmarks shape a field"
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Execution time is the measure of computer performance!

SPEC: System Performance Evaluation Cooperative

- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECInt92 (6 int. programs) and SPECfp92 (14 flt pt.)
- Third Round 1995
  - SPECint95 (8 int programs) and SPECfp95 (10 flt pt)
- Fourth Round 2000: SPEC CPU2000
  - 12 Integer, 14 Floating point
  - 2 choices on compilation; "aggressive" or "conservative"
  - multiple data sets so that can train compiler if trying to collect data for input to compiler to improve optimization
- Why SPEC: characterization of wide spectrum of use

What other benchmarks ?

- What if you are targeting the design for an application domain
- Some domains have well-defined/accepted benchmarks
  - Media Bench– for multimedia apps
  - Data Intensive Sys. (DIS) – for embedded systems that process input data
  - MI Bench – for embedded systems
  - TPC- transaction processing benchmarks to measure trans. proc. systems

How to Summarize Performance

- Arithmetic mean (weighted arithmetic mean) tracks execution time:
  $\Sigma(T_i)/n$ or $\Sigma(W_i*T_i)$
- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time:
  $n/\Sigma(1/R_i)$ or $n/\Sigma(W_i/R_i)$
- Normalized execution time is handy for scaling performance (e.g., X times faster than SPARCstation 10)

Performance

- How do you measure performance?
  - Throughput, Response time/latency
  - metric chosen depends on user community
    - System admin vs single user submitting homework
- Models for performance
  - CPU time equation
- What to measure
  - Benchmarks- SPEC, MIBench, etc.
- Next: How to improve performance – thumb rules

Performance: The AAA rule for designers

- Application
- Algorithm
- Architecture

Quantitative Principles of Computer Architecture Design (Thumb Rules)

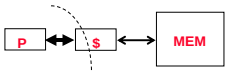
- Performance equation
- Common case fast
  - Focus on improving those instructions that are frequently used
- Amdahl's Law
  - Fraction enhanced/optimized runs faster
  - Parts of program that cannot be enhanced
- Locality
  - Spatial
  - Temporal
- Concurrency/Parallelism – overlap instruction execution

Parallelism

- Increasing throughput of server computer via multiple processors or multiple disks
- Detailed HW design
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.

The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If you use something then you will use it again soon
    - If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If you use something then you will use something nearby
    - If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Last 30 years, HW relied on locality for memory perf.



Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

Common Case

- 90% time spent on 10% of code
- Examples: Word proc, CAD
  - 80% of program instructions executed were from 3-5% of the code
  - 90% of inst. executed were from 9-12% code

Amdahl's Law: Speedup

- Application takes X time
- How to run it faster
  - Enhance/optimize a portion of it
    - Which portion
  - Can we enhance all of it
  - Note that we are talking of solving the enhanced part in a different way, and possibly using different (more costly) resources
- Eg: Getting from A to B, B to C.
  - Two portions to the task (A-B) and (B-C)

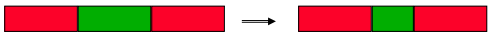
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O
  - Implies can "enhance"/optimize only 40% of code

$$Speedup_{overall} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster ☺

Architecture Design: Summary

- Design to last through trends
- Understand the principles
  - Make common case fast
  - Amdahl's law
  - Locality
  - Parallelism/concurrency