EI 338: Computer Systems Engineering
(Operating Systems & Computer Architecture)

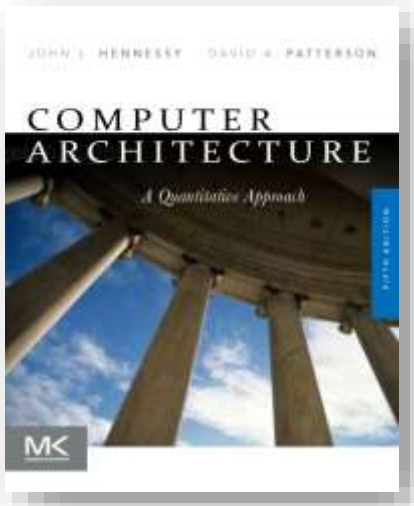
Dept. of Computer Science & Engineering
Chentao Wu
wuct@cs.sjtu.edu.cn

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Download lectures

- ftp://public.sjtu.edu.cn
- User: wuct
- Password: wuct123456

- http://www.cs.sjtu.edu.cn/~wuct/cse/



Chapter 1

Fundamentals of Quantitative Design and Analysis

上海交通大學

Outline

Introduction

Quantitative Principles of Computer Design

Classes of Computers

Computer Architecture

Trends in Technology

Power in Integrated Circuits

Trends in Cost

Dependability

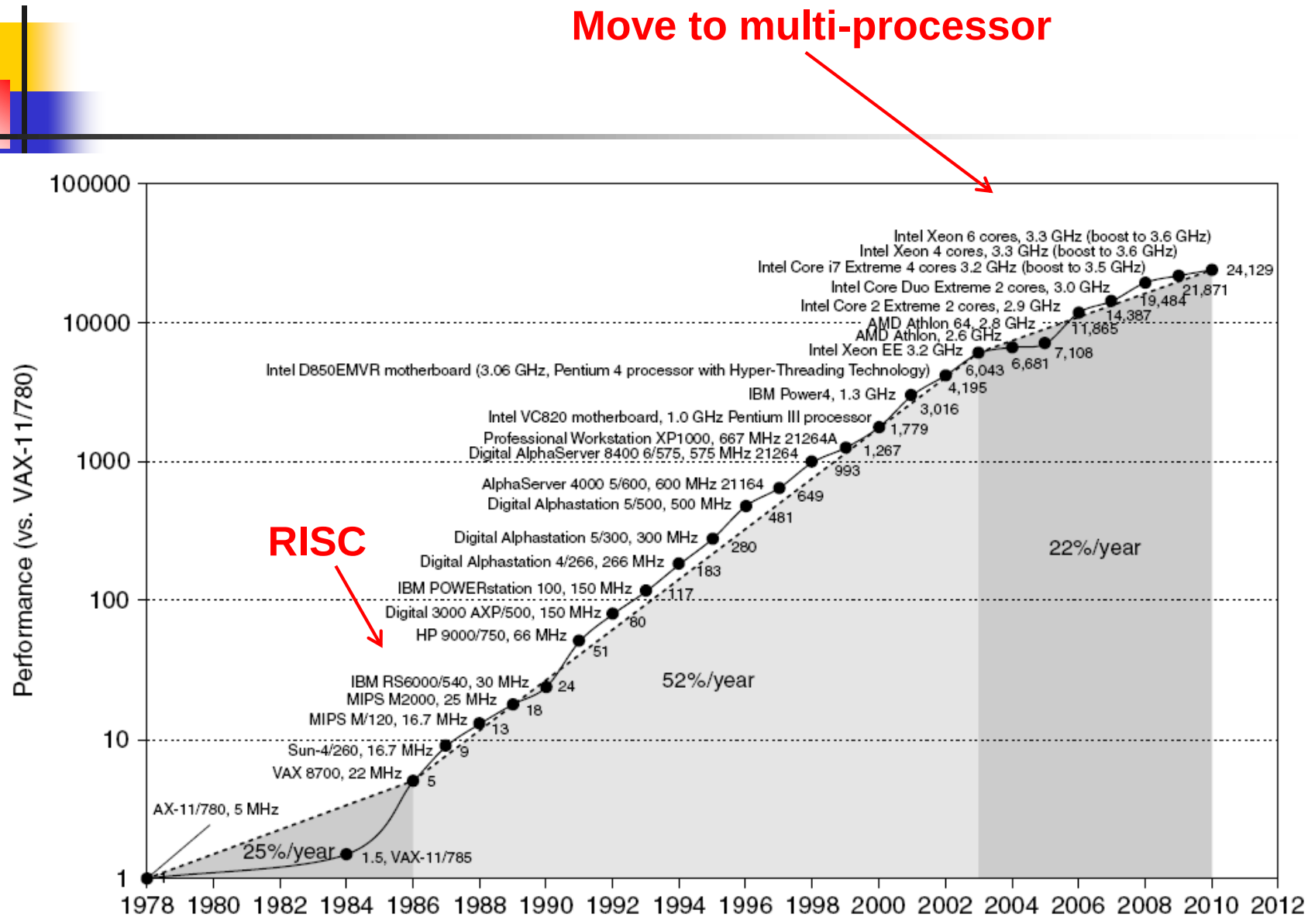
Performance

Fallacies and Pitfalls

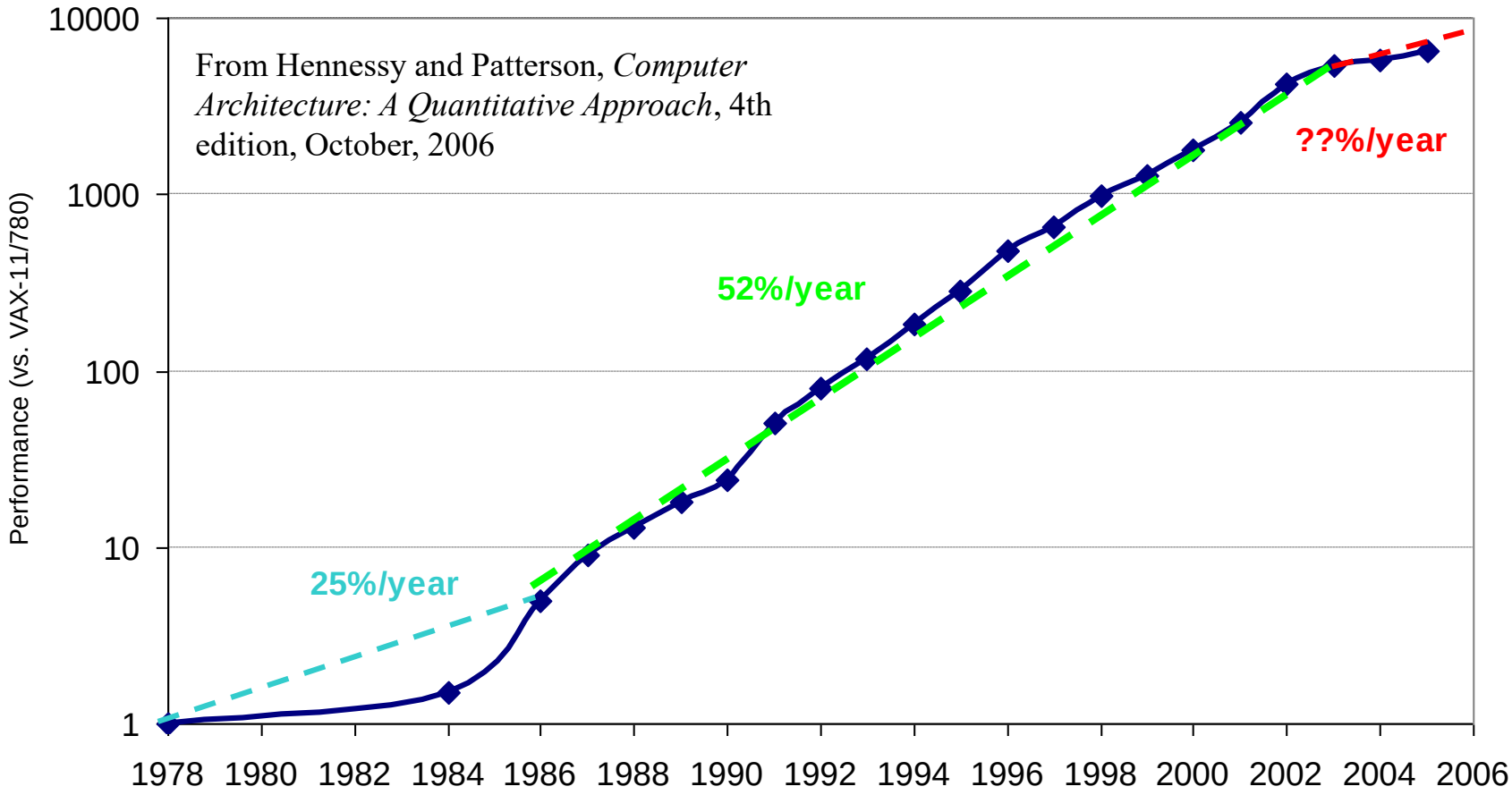
Computer Technology

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by High-Level Language (HLL) compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

Single Processor Performance



Crossroads: Uniprocessor Performance



- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

Less than 20%

Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application

Crossroads: Conventional Wisdom in Computer Architecture

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Transistors 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, …)
- 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)

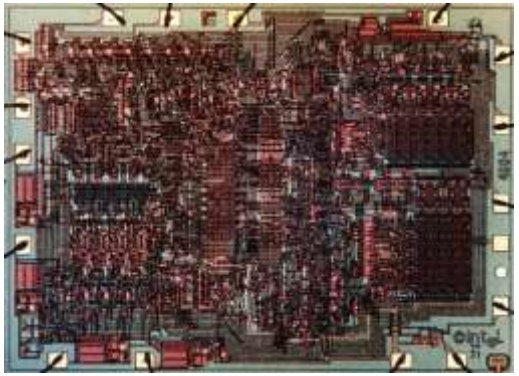
Crossroads: Conventional Wisdom in Computer Architecture

- 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

Sea Change in Chip Design

- **Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm^2 chip**



- **RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip**

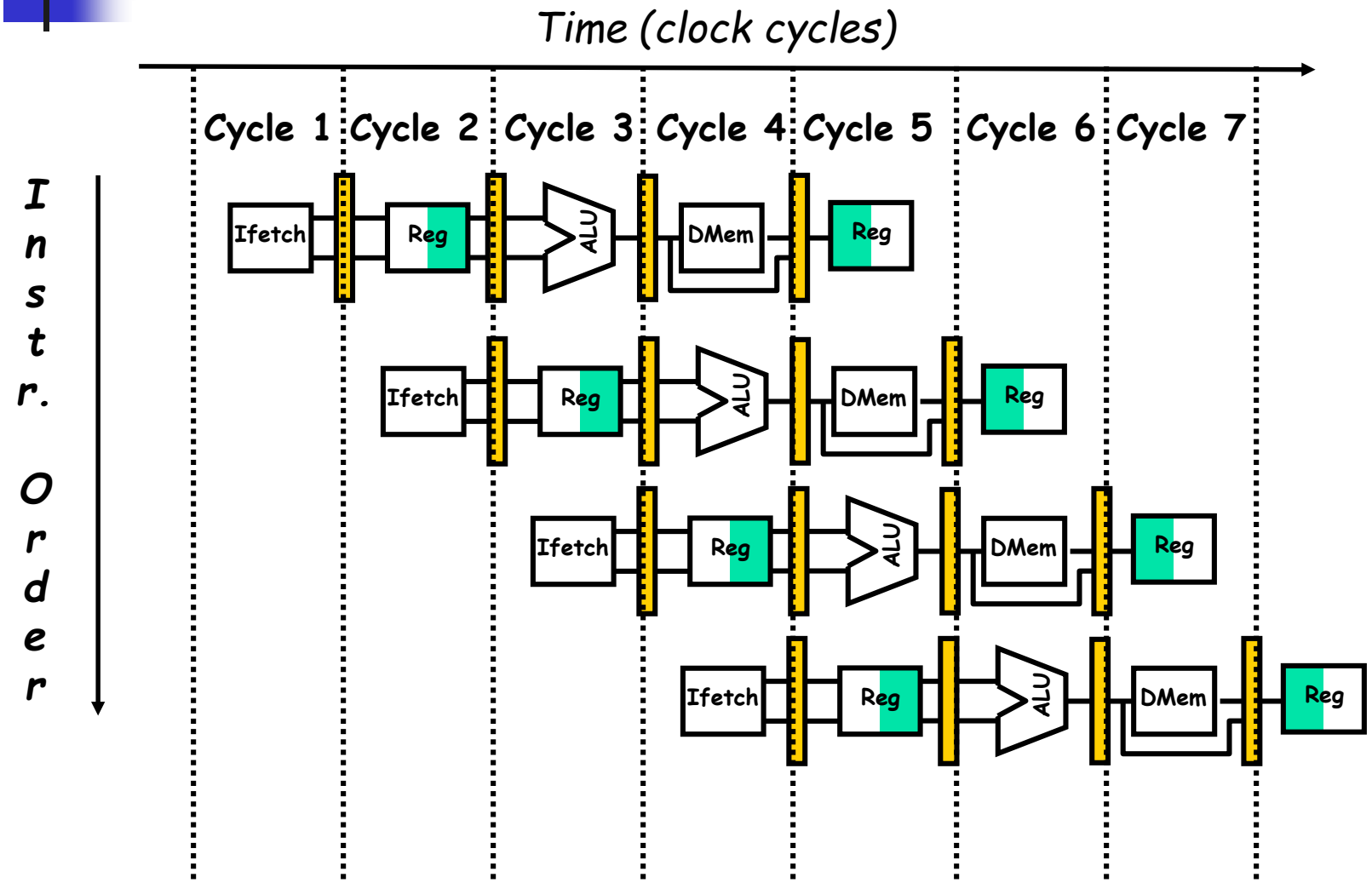
- **125 mm^2 chip, 0.065 micron CMOS = 2312 RISC II+FPU+Icache+Dcache**
  - **RISC II shrinks to ~ 0.02 mm^2 at 65 nm**
  - **Caches via DRAM or 1 transistor SRAM (www.t-ram.com) ?**
  - **Proximity Communication via capacitive coupling at > 1 TB/s ? (Ivan Sutherland @ Sun / Berkeley)**

- **Processor is the new transistor?**

Taking Advantage of Parallelism

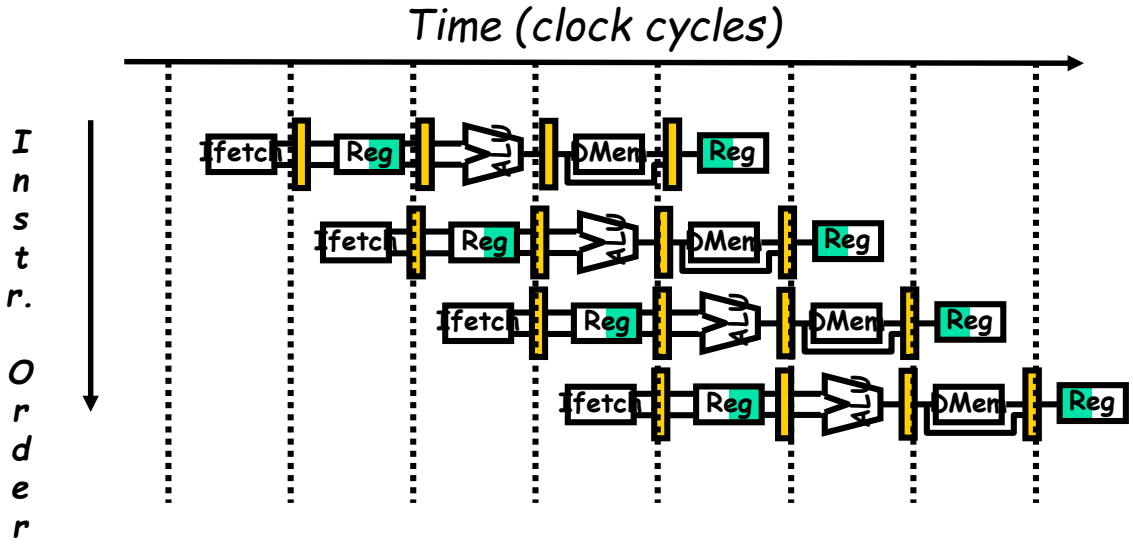
- Increasing throughput of server computer via multiple processors or multiple disks
- Detailed HW design
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.
  - Not every instruction depends on immediate predecessor ⇒ executing instructions completely/partially in parallel possible
  - Classic 5-stage pipeline:
    1) Instruction Fetch (Ifetch),
    2) Register Read (Reg),
    3) Execute (ALU),
    4) Data Memory Access (Dmem),
    5) Register Write (Reg)

Pipelined Instruction Execution



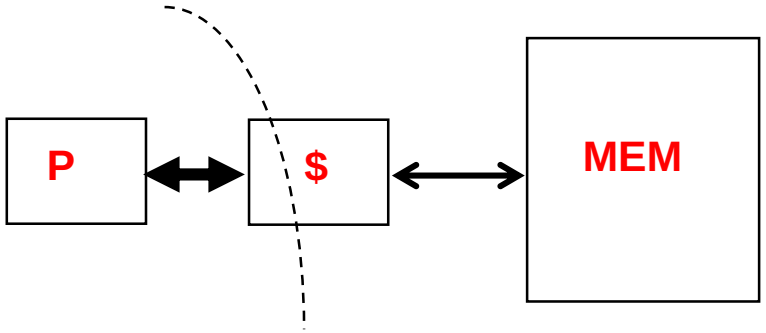
Limits to pipelining

- **Hazards prevent next instruction from executing during its designated clock cycle**
  - **Structural hazards: attempt to use the same hardware to do two different things at once**
  - **Data hazards: Instruction depends on result of prior instruction still in the pipeline**
  - **Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).**



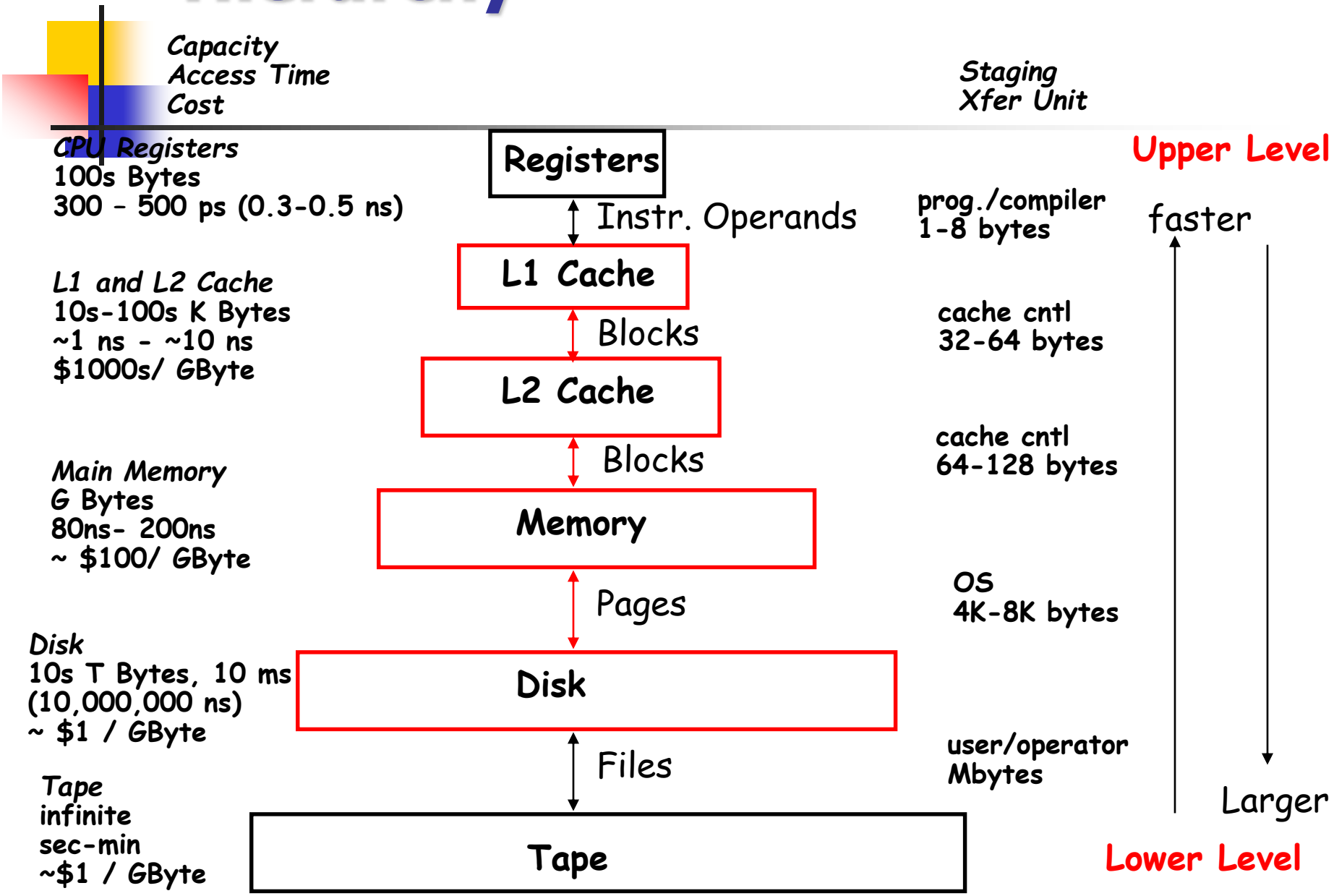
The Principle of Locality

- 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 an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): 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.



Levels of the Memory Hierarchy

上海交通大學



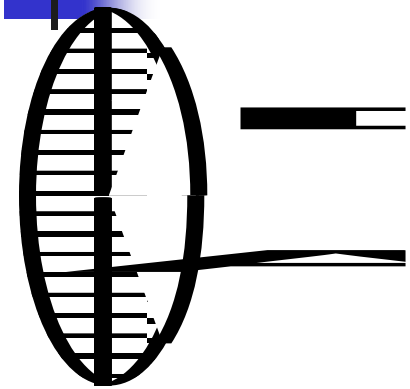
What Computer Architecture brings to Table

- Other fields often borrow ideas from architecture
- Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl's Law
  5. The Processor Performance Equation
- Careful, quantitative comparisons
  - Define, quantity, and summarize relative performance
  - Define and quantity relative cost
  - Define and quantity dependability
  - Define and quantity power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked

Comp. Arch. is an Integrated Approach

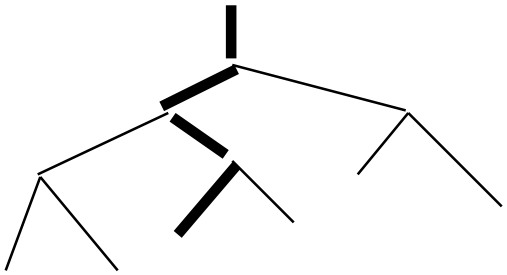
- What really matters is the functioning of the complete system
  - hardware, runtime system, compiler, operating system, and application
  - In networking, this is called the "End to End argument"
- Computer architecture is not just about transistors, individual instructions, or particular implementations
  - E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions

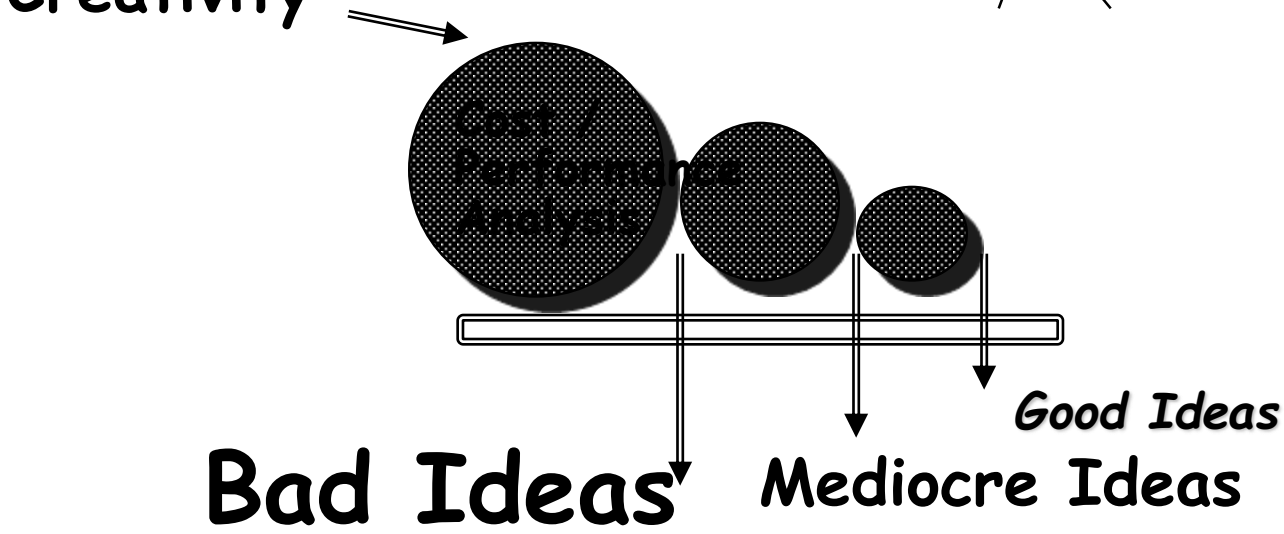
Computer Architecture is Design and Analysis



Architecture is an iterative process:

- Searching the space of possible designs
- At all levels of computer systems





上海交通大學

Outline

Introduction

Quantitative Principles of Computer Design

Classes of Computers

Computer Architecture

Trends in Technology

Power in Integrated Circuits

Trends in Cost

Dependability

Performance

Fallacies and Pitfalls

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

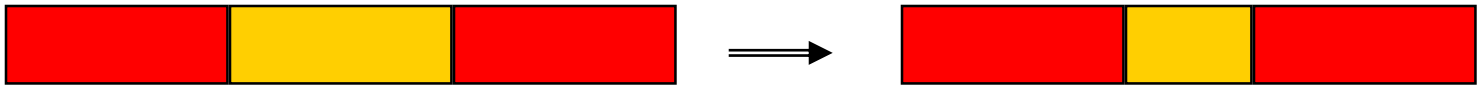
Amdahl's Law

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

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

Best you could ever hope to do:

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



Amdahl's Law example

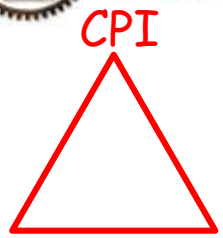
- **New CPU 10X faster**
- **I/O bound server, so 60% time waiting for I/O**

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

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

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

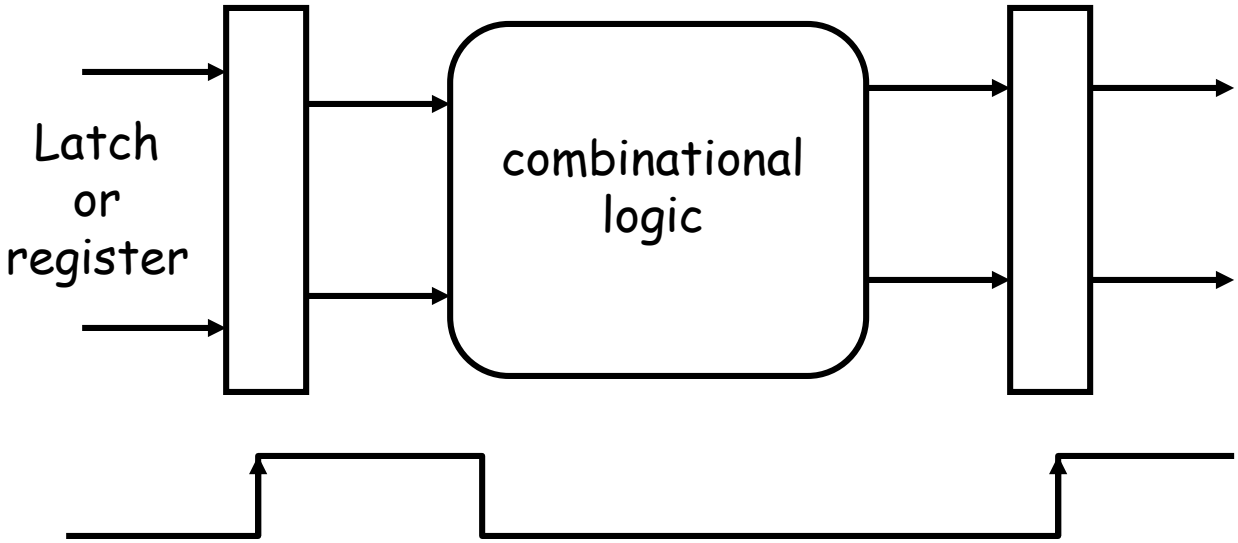
Processor performance equation



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

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

What's a Clock Cycle?



- **Old days: 10 levels of gates**
- **Today: determined by numerous time-of-flight issues + gate delays**
  - **clock propagation, wire lengths, drivers**

Examples on Pages 47-48

Example A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

Examples on Pages 47-48

Answer We can compare these two alternatives by comparing the speedups:

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

Improving the performance of the FP operations overall is slightly better because of the higher frequency.

Principles of Computer Design

- The Processor Performance Equation

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

Principles of Computer Design

- Different instruction types having different CPIs

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

Examples on Pages 50-51

Example Suppose we have made the following measurements:

Frequency of FP operations = 25%

Average CPI of FP operations = 4.0

Average CPI of other instructions = 1.33

Frequency of FPSQR = 2%

CPI of FPSQR = 20

Assume that the two design alternatives are to decrease the CPI of FPSQR to 2 or to decrease the average CPI of all FP operations to 2.5. Compare these two design alternatives using the processor performance equation.

Answer First, observe that only the CPI changes; the clock rate and instruction count remain identical. We start by finding the original CPI with neither enhancement:

$$\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times \left(\frac{\text{IC}_i}{\text{Instruction count}} \right)$$

$$= (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

We can compute the CPI for the enhanced FPSQR by subtracting the cycles saved from the original CPI:

$$\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{of new FPSQR only}})$$

$$= 2.0 - 2\% \times (20 - 2) = 1.64$$

We can compute the CPI for the enhancement of all FP instructions the same way or by summing the FP and non-FP CPIs. Using the latter gives us:

$$\text{CPI}_{\text{new FP}} = (75\% \times 1.33) + (25\% \times 2.5) = 1.625$$

Since the CPI of the overall FP enhancement is slightly lower, its performance will be marginally better. Specifically, the speedup for the overall FP enhancement is

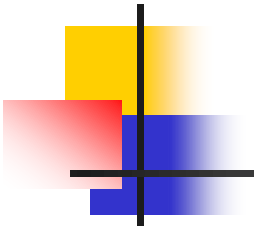
$$\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{new FP}}}$$

$$= \frac{\text{CPI}_{\text{original}}}{\text{CPI}_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23$$

Happily, we obtained this same speedup using Amdahl's law on page 46.

上海交通大學

Outline



Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for "Software as a Service (SaaS)"
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

上海交通大學

Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

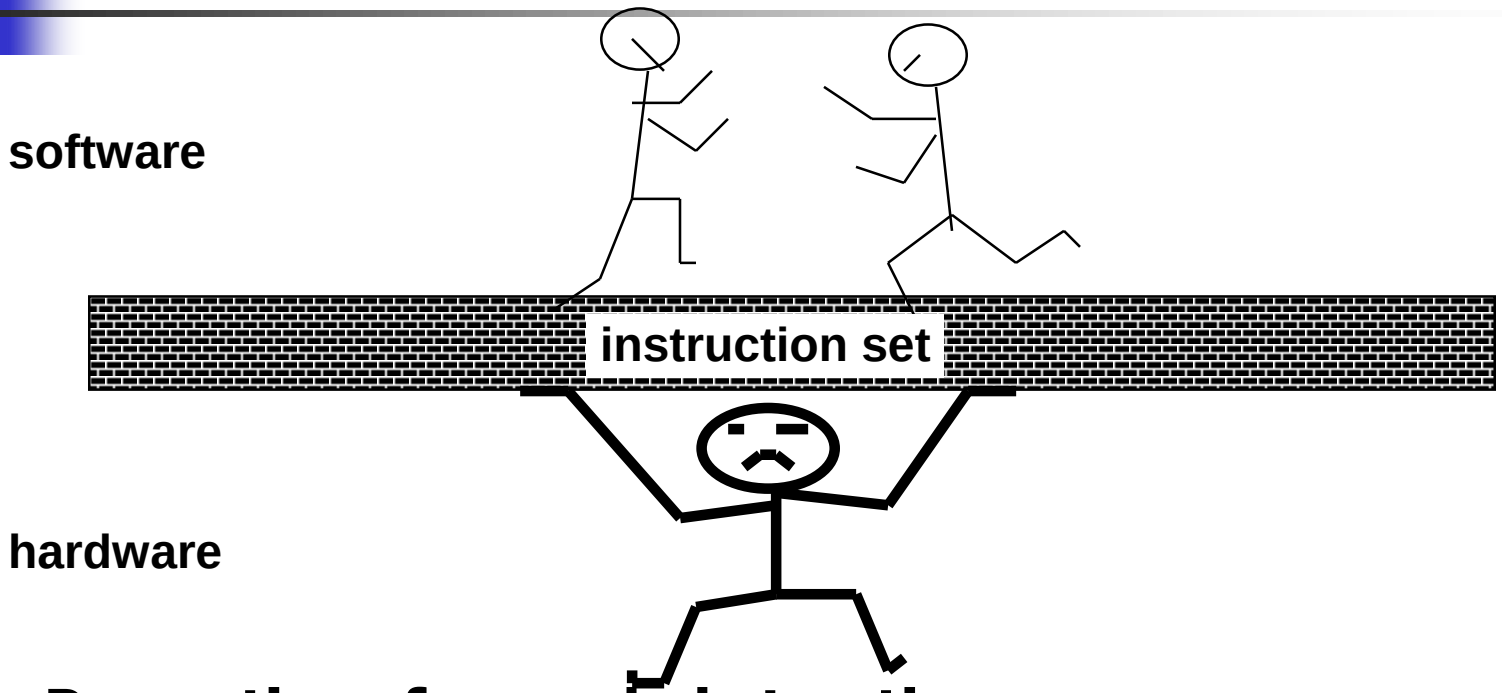
Parallelism

- Classes of parallelism in applications:
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)
- Classes of architectural parallelism:
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

Flynn's Taxonomy

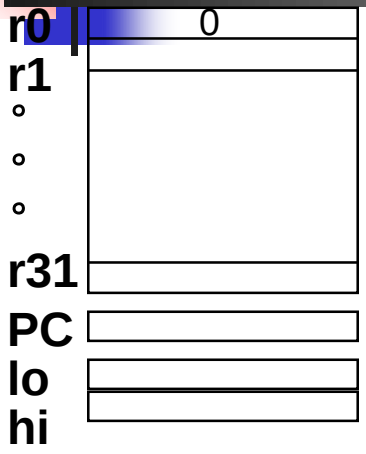
- Single instruction stream, single data stream (SISD)
  - a single processor executes a single instruction stream
  - Instruction Level Parallelism (ILP): pipelining
- Single instruction stream, multiple data streams (SIMD)
  - Multiple processors perform an instruction steam on multiple data stream simultaneously
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

Instruction Set Architecture: Critical Interface



- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

Example: MIPS architecture



Programmable storage

2^32 x bytes

31 x 32-bit GPRs (R0=0)

32 x 32-bit FP regs (paired DP)

HI, LO, PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

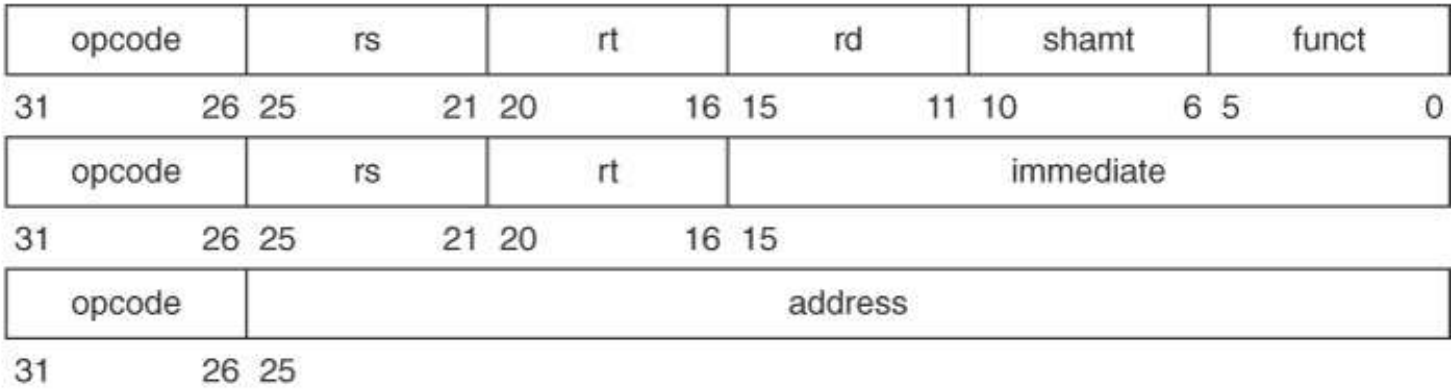
J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

32-bit instructions on word boundary

MIPS architecture instruction set format

Basic instruction formats

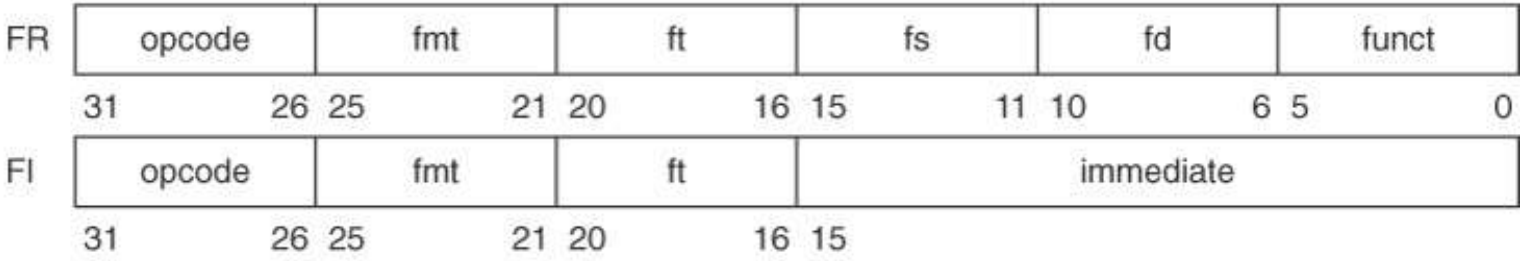
Register to register



Transfer, branches

Jumps

Floating-point instruction formats



ISA vs. Computer Architecture

- Old definition of computer architecture = instruction set design
  - Other aspects of computer design called implementation
  - Insinuates implementation is uninteresting or less challenging
- Our view is computer architecture >> ISA
- Architect's job much more than instruction set design; technical hurdles today *more* challenging than those in instruction set design
- Since instruction set design not where action is, some conclude computer architecture (using old definition) is not where action is
  - We disagree on conclusion
  - Agree that ISA not where action is (ISA in CA:AQA 4/e appendix)

Defining Computer Architecture

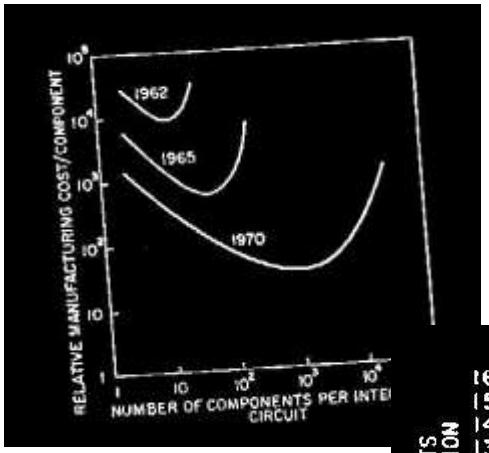
- "Old" view of computer architecture:
  - Instruction Set Architecture (ISA) design
  - i.e. decisions regarding:
    - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding
- "Real" computer architecture:
  - Specific requirements of the target machine
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware

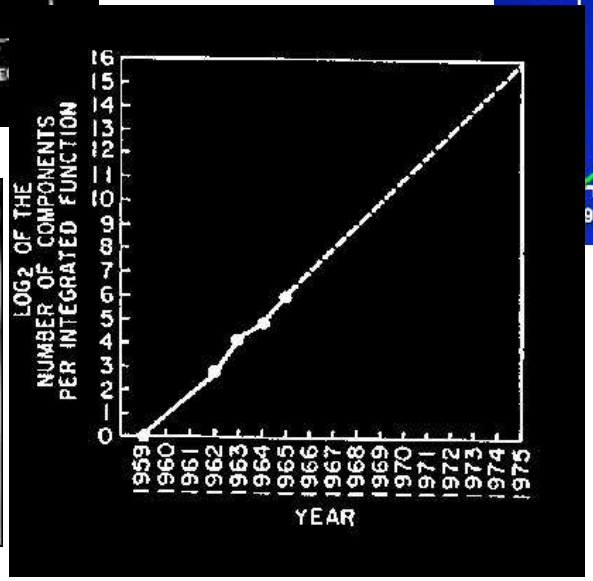
上海交通大學

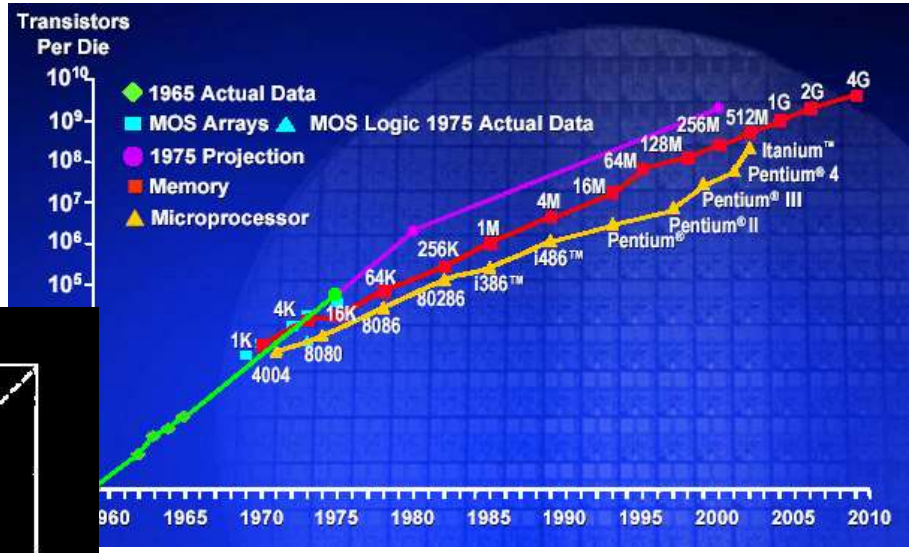
Outline

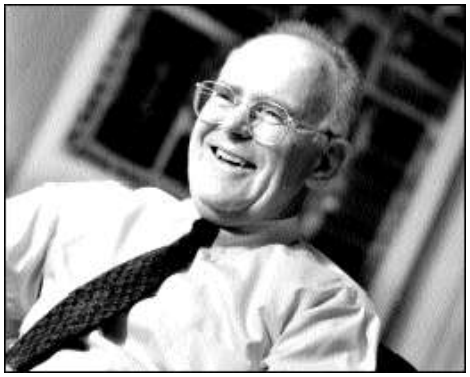
Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Moore's Law: 2X transistors / "year"









- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors / cost-effective integrated circuit double every N months (12 ≤ N ≤ 24)

Tracking Technology Performance Trends

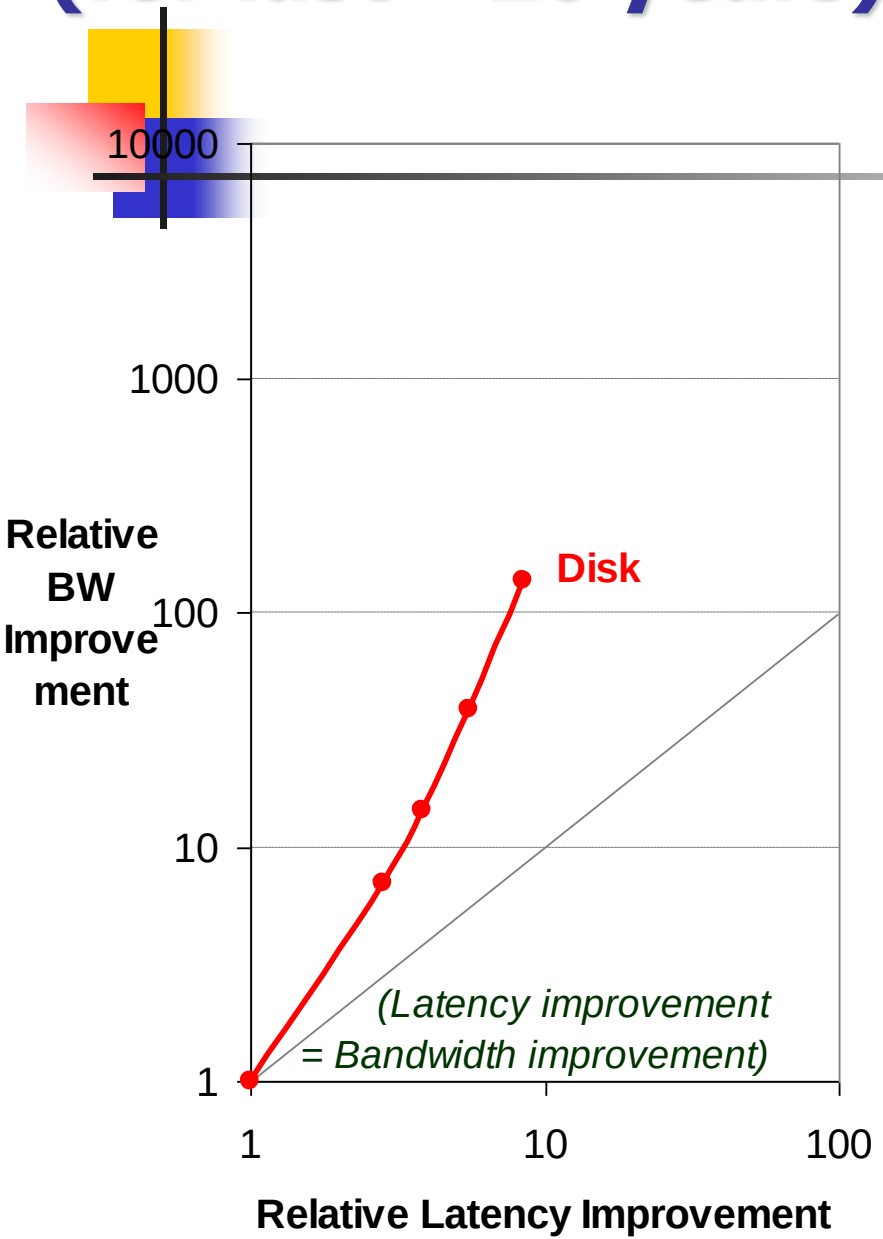
- Drill down into 4 technologies:
  - Disks,
  - Memory,
  - Network,
  - Processors
- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., M bits / second over network, M bytes / second from disk
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds

Disks: Archaic(Nostalgic) vs. Modern(Newfangled)

- CDC Wren I, 1983
- 3600 RPM
- 0.03 GBytes capacity
- Tracks/Inch: 800
- Bits/Inch: 9550
- Three 5.25” platters
- Bandwidth: 0.6 MBytes/sec
- Latency: 48.3 ms
- Cache: none

- Seagate 373453, 2003
- 15000 RPM (4X)
- 73.4 GBytes (2500X)
- Tracks/Inch: 64000 (80X)
- Bits/Inch: 533,000 (60X)
- Four 2.5” platters (in 3.5” form factor)
- Bandwidth: 86 MBytes/sec (140X)
- Latency: 5.7 ms (8X)
- Cache: 8 MBytes

Latency Lags Bandwidth (for last ~20 years)



- **Performance Milestones**

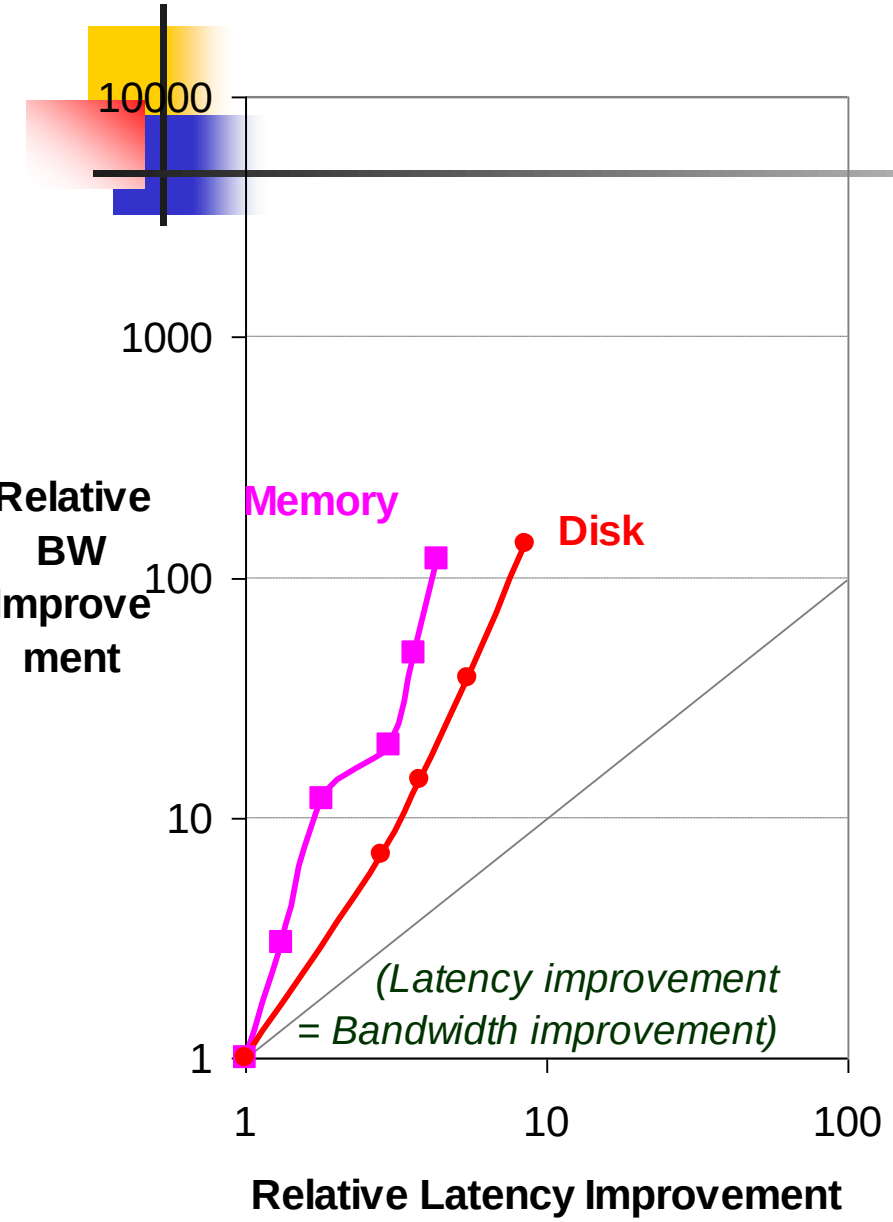
- **Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)**
  (latency = simple operation w/o contention
  BW = best-case)

Memory: Archaic (Nostalgic) vs. Modern (Newfangled)

- 1980 DRAM (asynchronous)
- 0.06 Mbits/chip
- 64,000 xtors, 35 mm^2
- 16-bit data bus per module, 16 pins/chip
- 13 Mbytes/sec
- Latency: 225 ns
- (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
- 256.00 Mbits/chip (4000X)
- 256,000,000 xtors, 204 mm^2
- 64-bit data bus per DIMM, 66 pins/chip (4X)
- 1600 Mbytes/sec (120X)
- Latency: 52 ns (4X)
- Block transfers (page mode)

Latency Lags Bandwidth (last ~20 years)



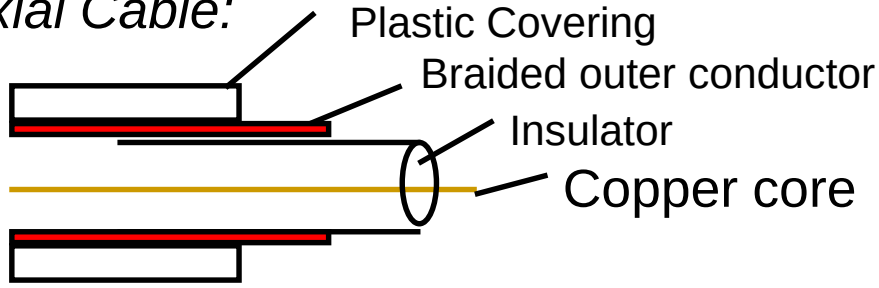
- **Performance Milestones**
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)**
- **Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)**

(latency = simple operation w/o contention
BW = best-case)

LANs: Archaic (Nostalgic) vs. Modern (Newfangled)

- Ethernet 802.3
- Year of Standard: 1978
- 10 Mbits/s link speed
- Latency: 3000 μsec
- Shared media
- Coaxial cable

Coaxial Cable:



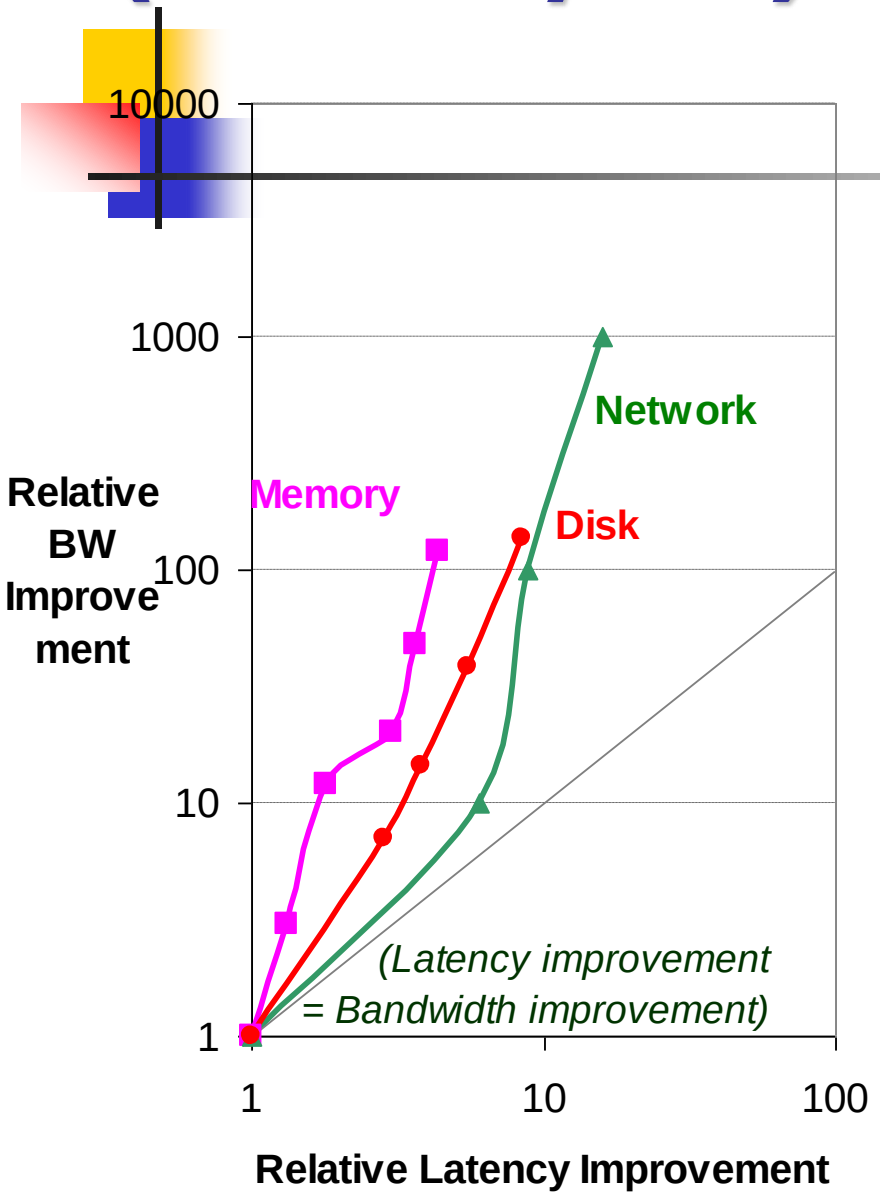
- Ethernet 802.3ae
- Year of Standard: 2003
- 10,000 Mbits/s link speed (1000X)
- Latency: 190 μsec (15X)
- Switched media
- Category 5 copper wire

"Cat 5" is 4 twisted pairs in bundle

Twisted Pair:

Copper, 1mm thick,
twisted to avoid antenna effect

Latency Lags Bandwidth (last ~20 years)



- **Performance Milestones**
- **Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s** (16x,1000x)
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM** (4x,120x)
- **Disk: 3600, 5400, 7200, 10000, 15000 RPM** (8x, 143x)

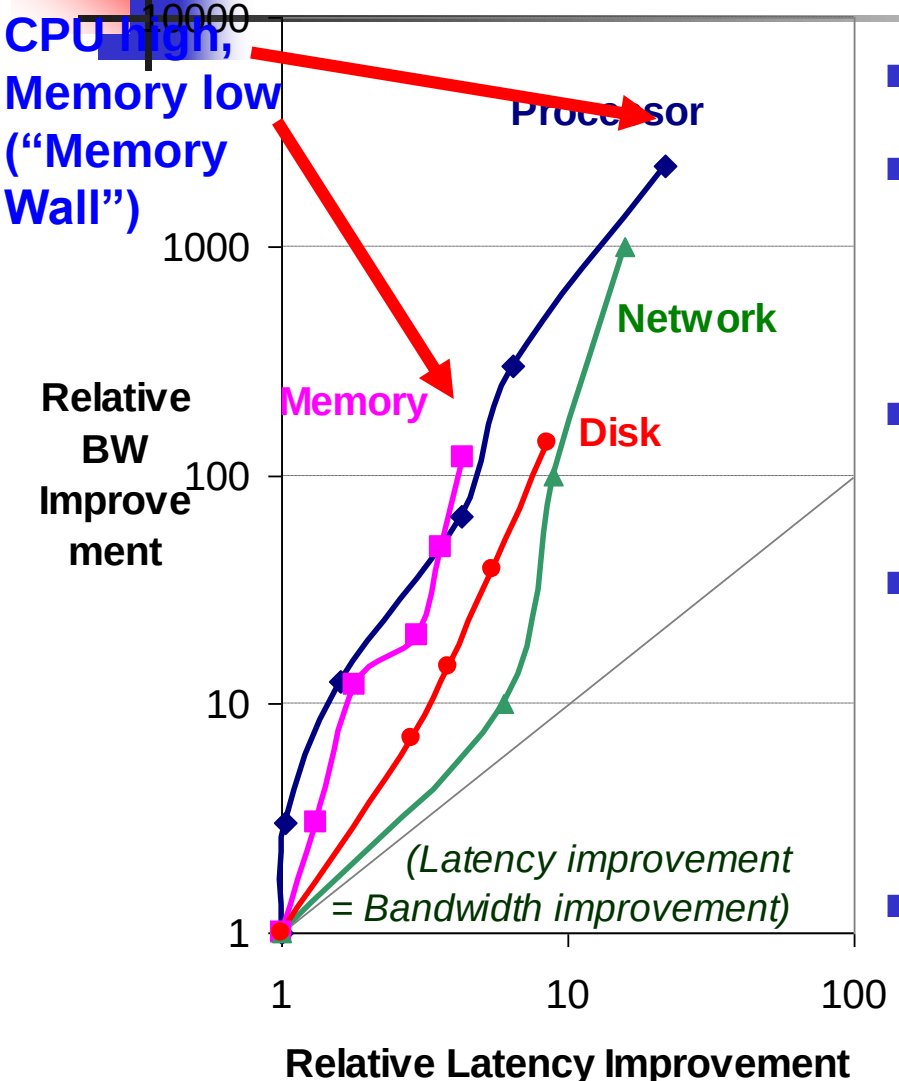
(latency = simple operation w/o contention
BW = best-case)

CPUs: Archaic (Nostalgic) vs. Modern (Newfangled)

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm^2
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)

- 2001 Intel Pentium 4
- 1500 MHz (120X)
- 4500 MIPS (peak) (2250X)
- Latency 15 ns (20X)
- 42,000,000 xtors, 217 mm^2
- 64-bit data bus, 423 pins
- 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
- On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Latency Lags Bandwidth (last ~20 years)



- **Performance Milestones**
- **Processor: '286, '386, '486, Pentium, Pentium Pro, Pentium 4** (21x,2250x)
- **Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s** (16x,1000x)
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM** (4x,120x)
- **Disk : 3600, 5400, 7200, 10000, 15000 RPM** (8x, 143x)

Rule of Thumb for Latency Lagging BW

- In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4

  (and capacity improves faster than bandwidth)

- Stated alternatively:
  Bandwidth improves by more than the square of the improvement in Latency

6 Reasons Latency Lags Bandwidth

1. Moore's Law helps BW more than latency

- Faster transistors, more transistors, more pins help Bandwidth
  - MPU Transistors: 0.130 vs. 42 M xtors (300X)
  - DRAM Transistors: 0.064 vs. 256 M xtors (4000X)
  - MPU Pins: 68 vs. 423 pins (6X)
  - DRAM Pins: 16 vs. 66 pins (4X)
- Smaller, faster transistors but communicate over (relatively) longer lines: limits latency
  - Feature size: 1.5 to 3 vs. 0.18 micron (8X,17X)
  - MPU Die Size: 35 vs. 204 mm^2 (ratio sqrt $\Rightarrow$ 2X)
  - DRAM Die Size: 47 vs. 217 mm^2 (ratio sqrt $\Rightarrow$ 2X)

6 Reasons Latency Lags Bandwidth (cont'd)

2. Distance limits latency

- Size of DRAM block ⇒ long bit and word lines
  ⇒ most of DRAM access time
- Speed of light and computers on network
- 1. & 2. explains linear latency vs. square BW?

3. Bandwidth easier to sell ("bigger=better")

- E.g., 10 Gbits/s Ethernet ("10 Gig") vs.
  10 μsec latency Ethernet
- 4400 MB/s DIMM ("PC4400") vs. 50 ns latency
- Even if just marketing, customers now trained
- Since bandwidth sells, more resources thrown at bandwidth, which further tips the balance

6 Reasons Latency Lags Bandwidth (cont'd)

4. Latency helps BW, but not vice versa

- Spinning disk faster improves both bandwidth and rotational latency
  - 3600 RPM ⇒ 15000 RPM = 4.2X
  - Average rotational latency: 8.3 ms ⇒ 2.0 ms
  - Things being equal, also helps BW by 4.2X
- Lower DRAM latency ⇒ More access/second (higher bandwidth)
- Higher linear density helps disk BW (and capacity), but not disk Latency
  - 9,550 BPI ⇒ 533,000 BPI ⇒ 60X in BW

6 Reasons Latency Lags Bandwidth (cont'd)

5. Bandwidth hurts latency

- Queues help Bandwidth, hurt Latency (Queuing Theory)
- Adding chips to widen a memory module increases Bandwidth but higher fan-out on address lines may increase Latency

6. Operating System overhead hurts Latency more than Bandwidth

- Long messages amortize overhead; overhead bigger part of short messages

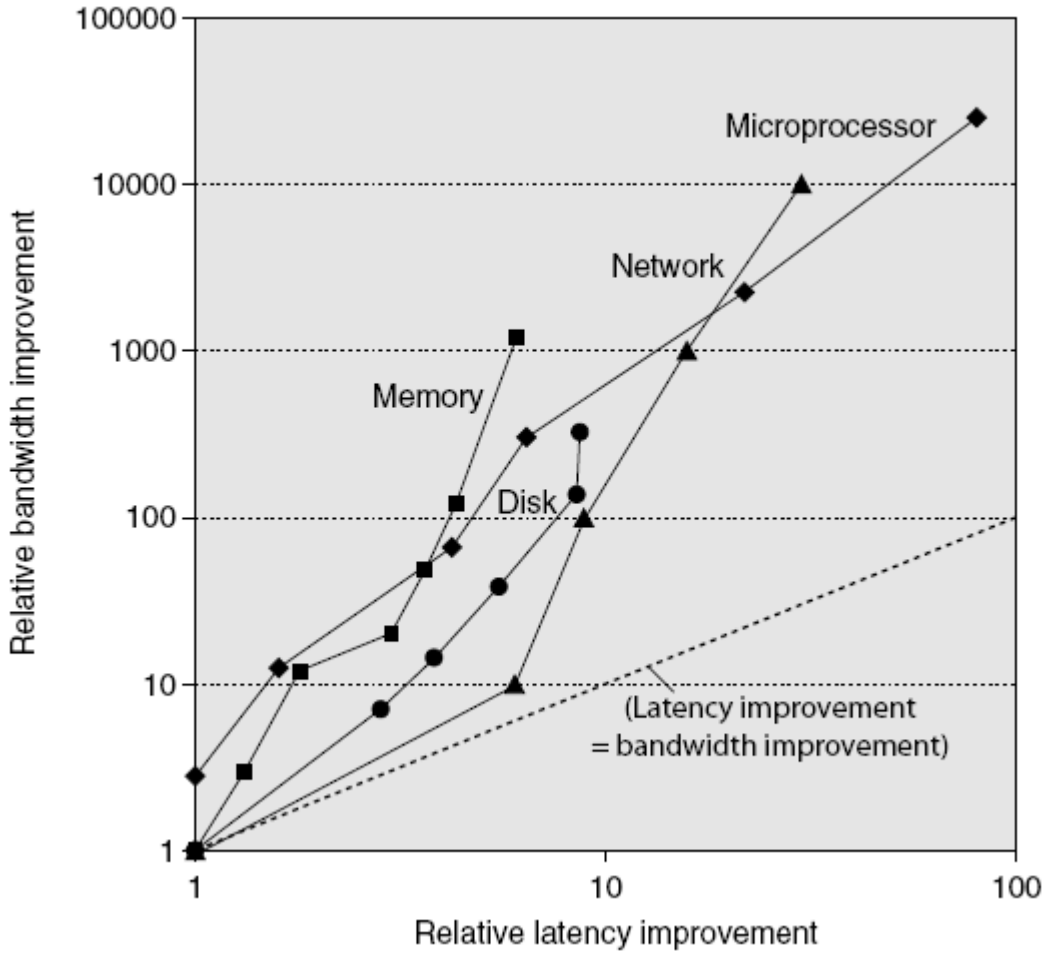
Trends in Technology

- Integrated circuit technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year
  - 15-25X cheaper/bit then Flash
  - 300-500X cheaper/bit than DRAM

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

上海交通大學

Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Transistors and Wires

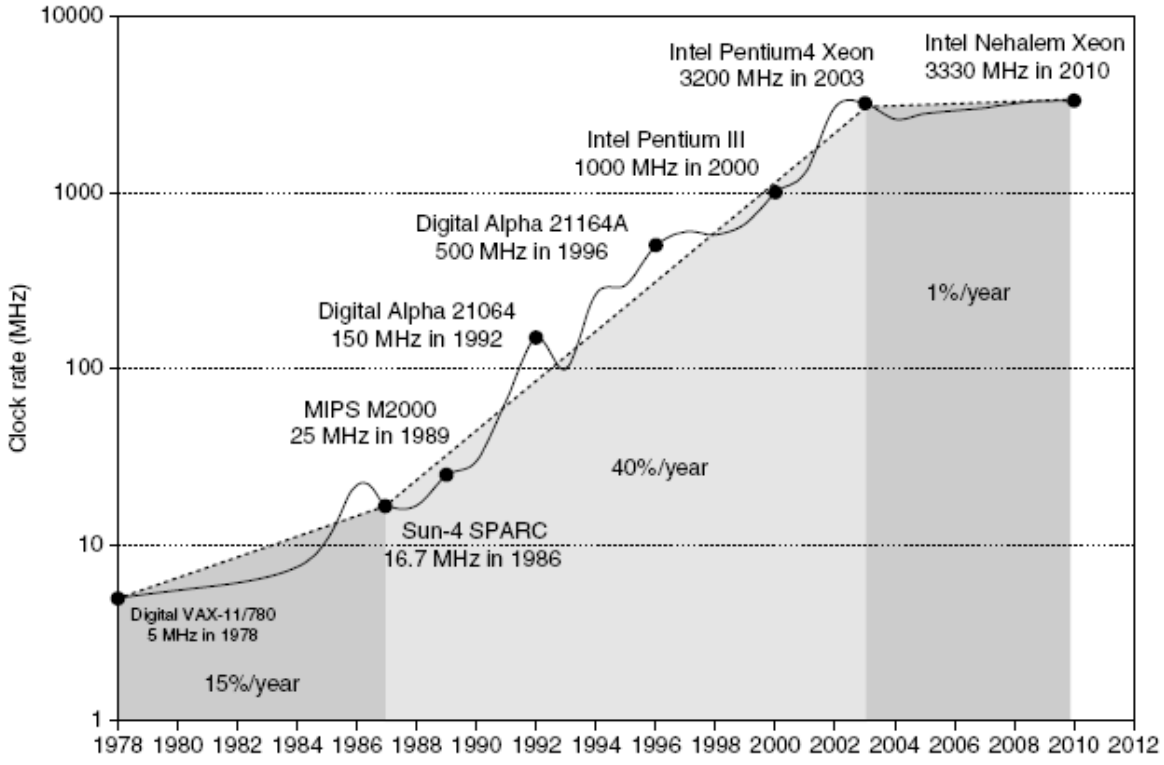
- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - 10 microns in 1971 to .032 microns in 2011
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically

Power and Energy

- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption

Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



Define and quantity power (1 / 2)

- **For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power:***

$$\text{Power}_{dynamic} = 0.5 \times \text{CapacitiveLoad} \times \text{Voltage}^2 \times \text{FrequencySwitched}$$

- For mobile devices, energy better metric

$$\text{Energy}_{dynamic} = \text{CapacitiveLoad} \times \text{Voltage}^2$$

- 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?**

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

Define and quantity power (2 / 2)

- **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

Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling
  - Low power state for DRAM, disks
  - Overclocking, turning off cores

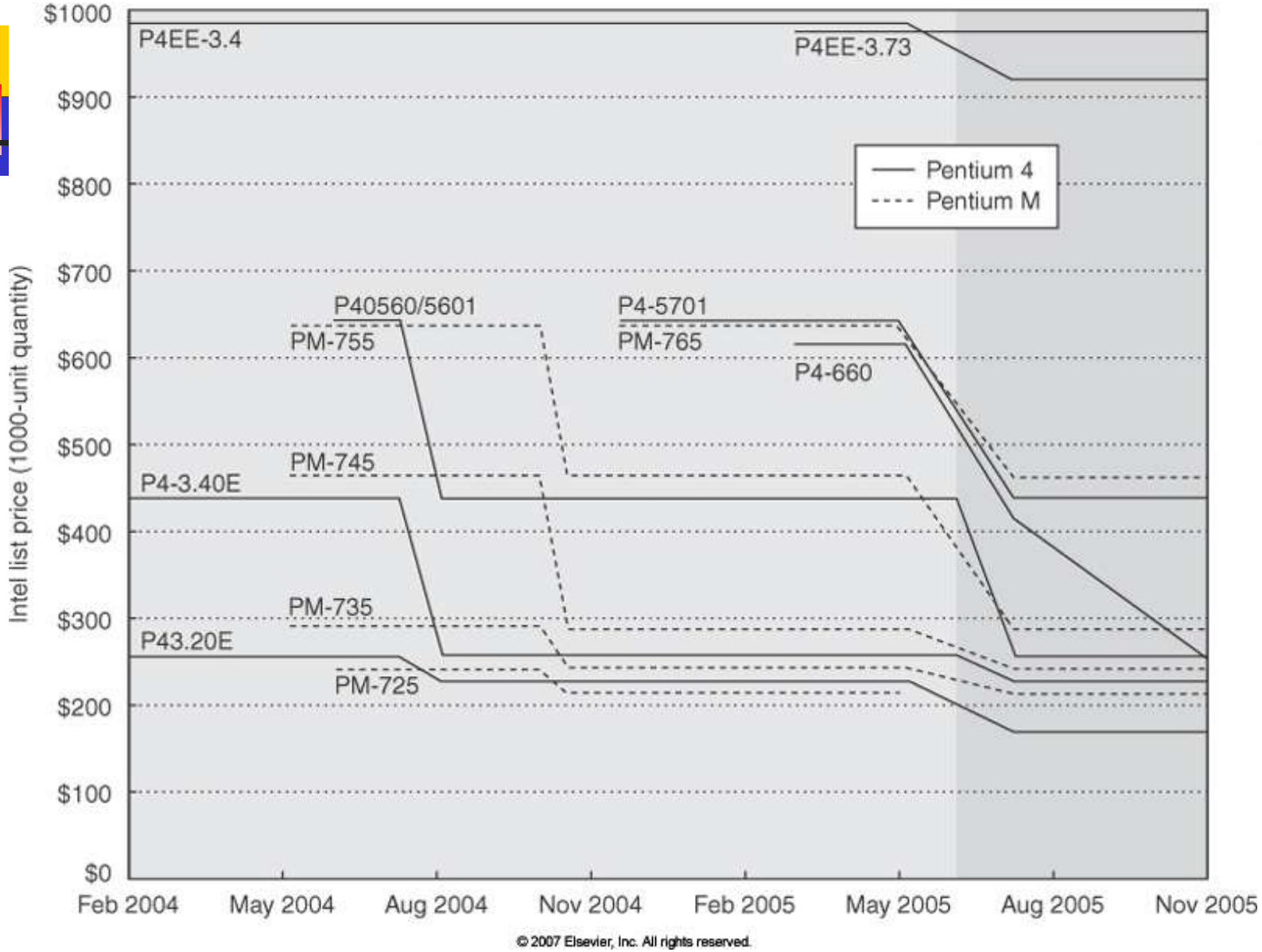
Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

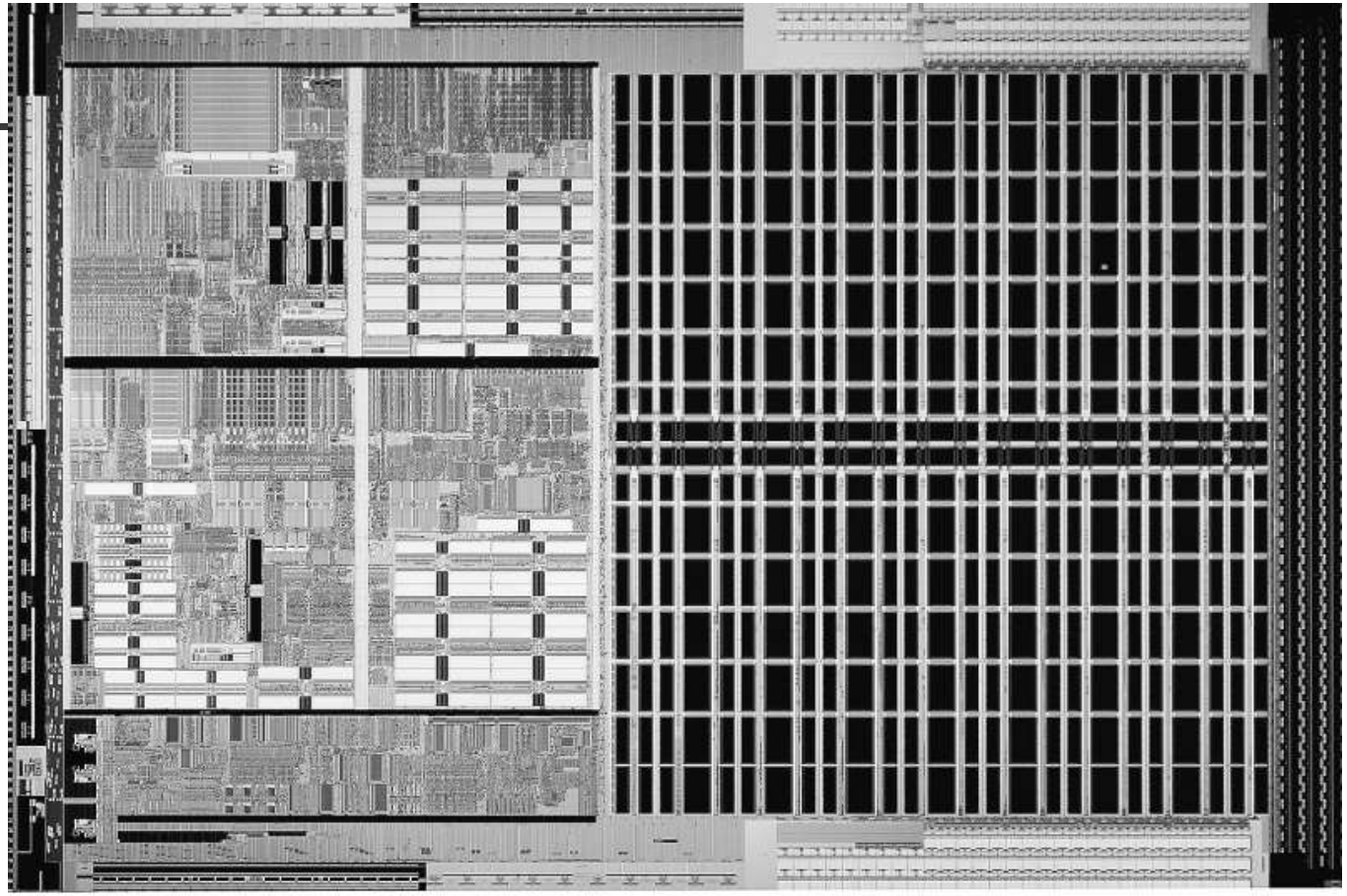
Trends in Cost

- Cost driven down by learning curve
  - Yield
- DRAM: price closely tracks cost
- Microprocessors: price depends on volume
  - 10% less for each doubling of volume

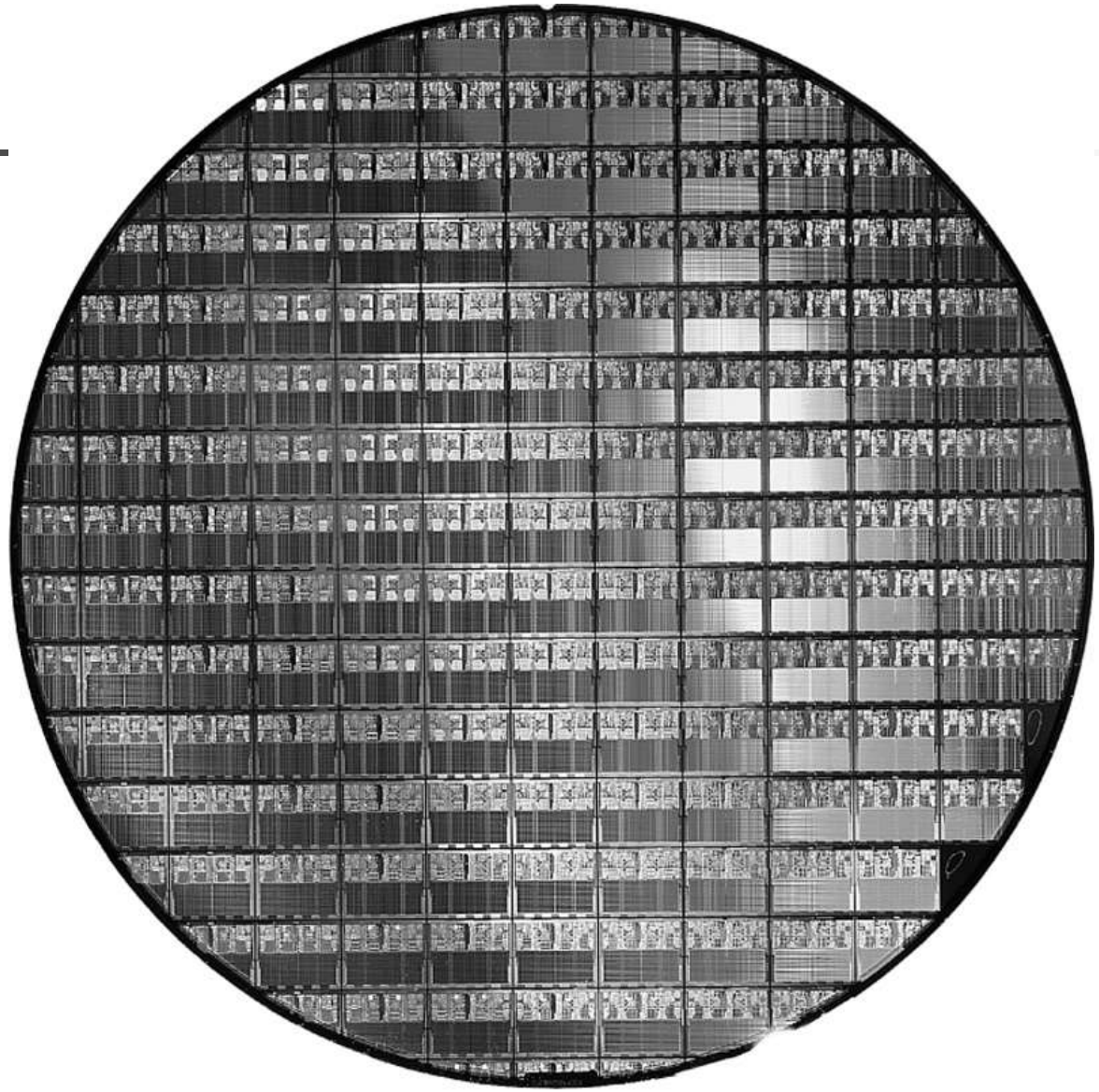
The price of Intel Pentium 4 and Pentium M



AMD Opteron Microprocessor Die



A 300mm silicon wafer contains 117 AMD Opteron microprocessor chips in a 90nm process



Integrated Circuit Cost

- **Integrated circuit**

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- Bose-Einstein formula:

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- Defects per unit area = 0.016-0.057 defects per square cm (2010)
- N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

Examples on Page 31

Example Find the number of dies per 300 mm (30 cm) wafer for a die that is 1.5 cm on a side and for a die that is 1.0 cm on a side.

Answer When die area is 2.25 cm^2:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2 \times 2.25}} = \frac{706.9}{2.25} - \frac{94.2}{2.12} = 270$$

Since the area of the larger die is 2.25 times bigger, there are roughly 2.25 as many smaller dies per wafer:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{1.00} - \frac{\pi \times 30}{\sqrt{2 \times 1.00}} = \frac{706.9}{1.00} - \frac{94.2}{1.41} = 640$$

Bose-Einstein formula

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{a}\right)^{-a}$$

Wafer yield: measures how many wafers are completely bad

$a = 4$ corresponds to masking levels in manufacturing process

Example:

Defect density = 0.4 per cm^2

Die area = 1.5cm X 1.5 cm = 2.25cm^2

Die yield = 0.44

Die area = 1.0cm X 1.0 cm = 1cm^2

Die yield = 0.68

Smaller die area gives more die yield

上海交通大學

Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Define and quantity dependability (1/3)

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:

1. Service accomplishment, where the service is delivered as specified in SLA
2. Service interruption, where the delivered service is different from the SLA

- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

Define and quantity dependability (2/3)

- ***Module reliability*** = measure of continuous service accomplishment (or time to failure).
  2 metrics
1. ***Mean Time To Failure*** (***MTTF***) measures Reliability
2. ***Failures In Time*** (***FIT***) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation
- ***Mean Time To Repair*** (***MTTR***) measures Service Interruption
  - ***Mean Time Between Failures*** (***MTBF***) = MTTF+MTTR
- ***Module availability*** measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- ***Module availability = MTTF / (MTTF + MTTR)***

Examples on Pages 34-35

Example Assume a disk subsystem with the following components and MTTF:

- 10 disks, each rated at 1,000,000-hour MTTF
- 1 ATA controller, 500,000-hour MTTF
- 1 power supply, 200,000-hour MTTF
- 1 fan, 200,000-hour MTTF
- 1 ATA cable, 1,000,000-hour MTTF

Using the simplifying assumptions that the lifetimes are exponentially distributed and that failures are independent, compute the MTTF of the system as a whole.

Examples on Pages 34-35

Answer The sum of the failure rates is

$$\text{Failure rate}_{\text{system}} = 10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$$

$$= \frac{10 + 2 + 5 + 5 + 1}{1{,}000{,}000 \text{ hours}} = \frac{23}{1{,}000{,}000} = \frac{23{,}000}{1{,}000{,}000{,}000 \text{ hours}}$$

or 23,000 FIT. The MTTF for the system is just the inverse of the failure rate:

$$\text{MTTF}_{\text{system}} = \frac{1}{\text{Failure rate}_{\text{system}}} = \frac{1{,}000{,}000{,}000 \text{ hours}}{23{,}000} = 43{,}500 \text{ hours}$$

or just under 5 years.

上海交通大學

Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput
- Speedup of X relative to Y
  - Execution time$_Y$ / Execution time$_X$
- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications, called *benchmark suites*, are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPECint2000 has 12 integer, SPECfp2000 has 14 integer pgms
  - SPECCPU2006 to be announced Spring 2006
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-H models ad hoc decision support
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

How Summarize Suite Performance (1/5)

- Arithmetic average of execution time of all pgms?
  - But they vary by 4X in speed, so some would be more important than others in arithmetic average
- Could add a weights per program, but how pick weight?
  - Different companies want different weights for their products
- SPECRatio: Normalize execution times to reference computer, yielding a ratio proportional to performance

$$\frac{\text{time on reference computer}}{\text{time on computer being rated}}$$

How Summarize Suite Performance (2/5)

- **If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then**

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\frac{ExecutionTime_{reference}}{ExecutionTime_A}}{\frac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

  - **Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant**

How Summarize Suite Performance (3/5)

- **Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)**

$$GeometricMean = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

1. **Geometric mean of the ratios is the same as the ratio of the geometric means**
2. **Ratio of geometric means = Geometric mean of performance ratios ⇒ choice of reference computer is irrelevant!**

- **These two points make geometric mean of ratios attractive to summarize performance**

How Summarize Suite Performance (4/5)

- Does a single mean well summarize performance of programs in benchmark suite?
- Can decide if mean a good predictor by characterizing variability of distribution using standard deviation
- Like geometric mean, geometric standard deviation is multiplicative rather than arithmetic
- Can simply take the logarithm of SPECRatios, compute the standard mean and standard deviation, and then take the exponent to convert back:

$$GeometricMean = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln\left(SPECRatio_i\right)\right)$$

$$GeometricStDev = \exp\left(StDev\left(\ln\left(SPECRatio_i\right)\right)\right)$$

How Summarize Suite Performance (5/5)

- **Standard deviation is more informative if know distribution has a standard form**
  - ***bell-shaped normal distribution*, whose data are symmetric around mean**
  - ***lognormal distribution*, where logarithms of data--not data itself--are normally distributed (symmetric) on a logarithmic scale**
- **For a lognormal distribution, we expect that**

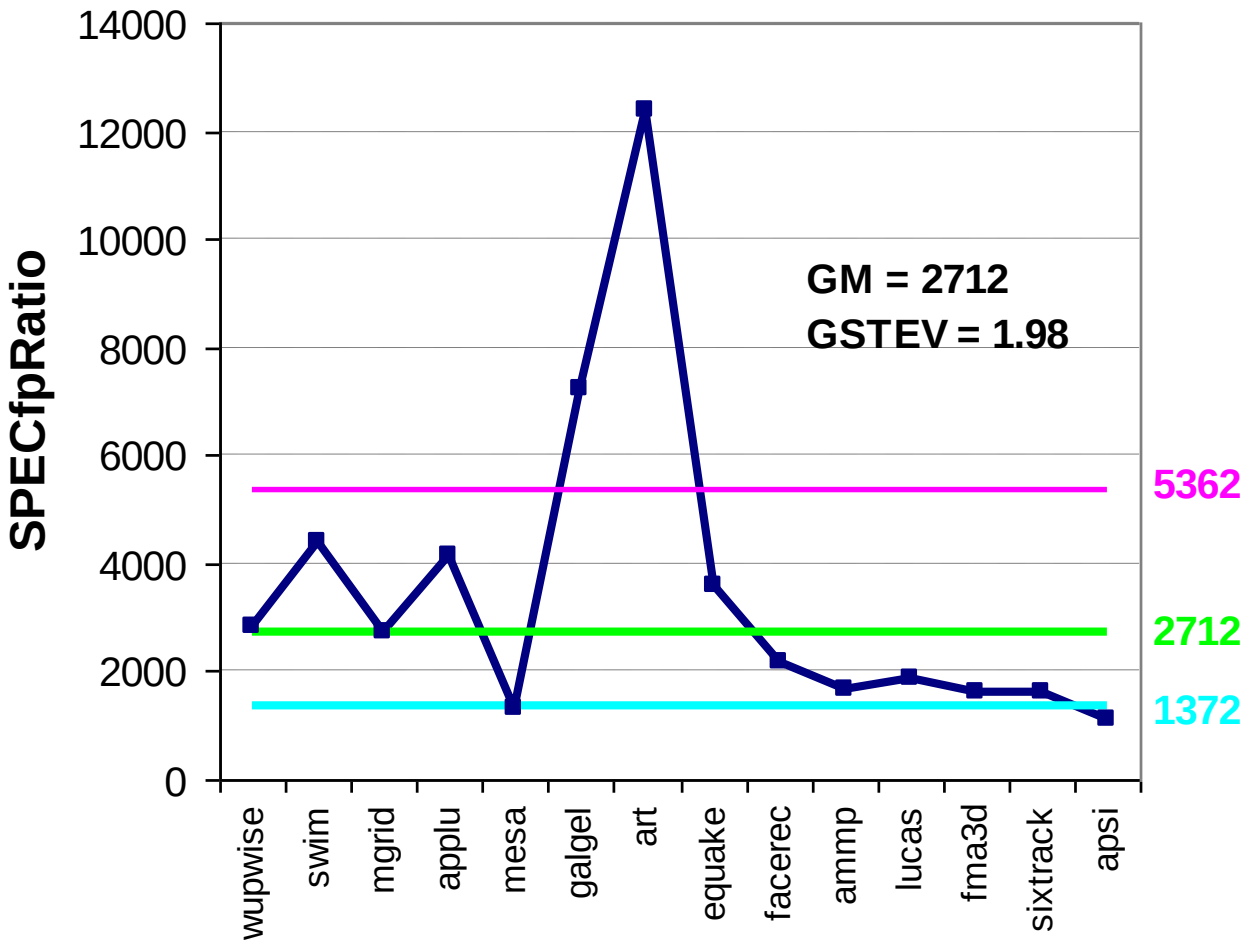
68% of samples fall in range $[mean / gstdev, mean \times gstdev]$

95% of samples fall in range $[mean / gstdev^2, mean \times gstdev^2]$

- **Note: Excel provides functions EXP(), LN(), and STDEV() that make calculating geometric mean and multiplicative standard deviation easy**

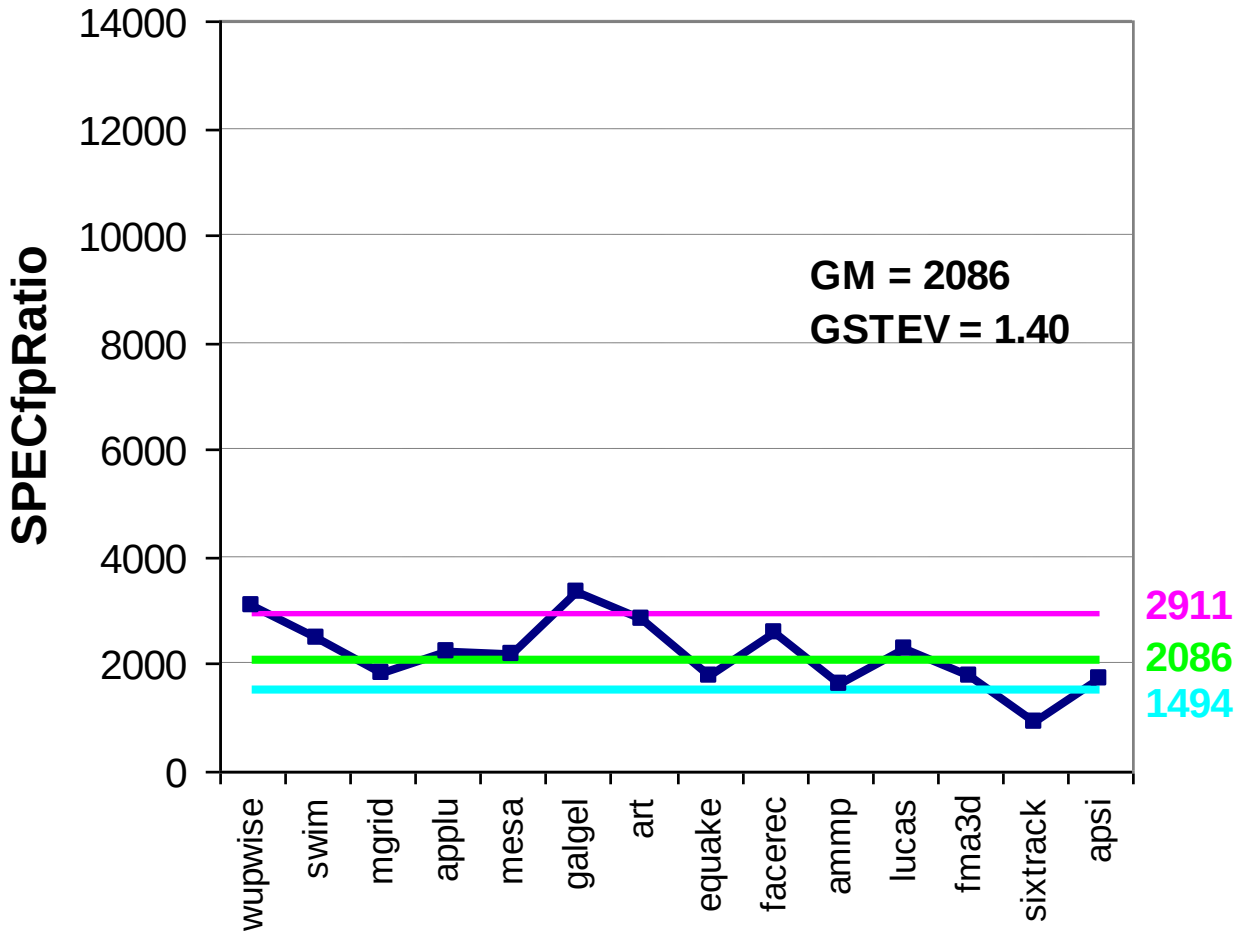
Example Standard Deviation (1/2)

- **GM and multiplicative StDev of SPECfp2000 for Itanium 2**



Example Standard Deviation (2/2)

- **GM and multiplicative StDev of SPECfp2000 for AMD Athlon**



Comments on Itanium 2 and Athlon

- Standard deviation of 1.98 for Itanium 2 is much higher-- vs. 1.40--so results will differ more widely from the mean, and therefore are likely less predictable
- Falling within one standard deviation:
  - 10 of 14 benchmarks (71%) for Itanium 2
  - 11 of 14 benchmarks (78%) for Athlon
- Thus, the results are quite compatible with a lognormal distribution (expect 68%)

上海交通大學

Outline

Introduction
Quantitative Principles of Computer Design
Classes of Computers
Computer Architecture
Trends in Technology
Power in Integrated Circuits
Trends in Cost
Dependability
Performance
Fallacies and Pitfalls

Fallacies and Pitfalls

- Fallacies - commonly held misconceptions
  - When discussing a fallacy, we try to give a counterexample.
- Pitfalls - easily made mistakes.
  - Often generalizations of principles true in limited context
  - Show Fallacies and Pitfalls to help you avoid these errors
- Fallacy: Benchmarks remain valid indefinitely
  - Once a benchmark becomes popular, tremendous pressure to improve performance by targeted optimizations or by aggressive interpretation of the rules for running the benchmark: "benchmarksmanship."
  - 70 benchmarks from the 5 SPEC releases. 70% were dropped from the next release since no longer useful
- Pitfall: A single point of failure
  - Rule of thumb for fault tolerant systems: make sure that every component was redundant so that no single component failure could bring down the whole system (e.g, power supply)

- Fallacy - Rated MTTF of disks is 1,200,000 hours or ≈ 140 years, so disks practically never fail
- But disk lifetime is 5 years ⇒ replace a disk every 5 years; on average, 28 replacements wouldn't fail
- A better unit: % that fail (1.2M MTTF = 833 FIT)
- Fail over lifetime: if had 1000 disks for 5 years
  = 1000*(5*365*24)*833 /10^9 = 36,485,000 / 10^6 = 37
  = 3.7% (37/1000) fail over 5 yr lifetime (1.2M hr MTTF)
- But this is under pristine conditions
  - little vibration, narrow temperature range ⇒ no power failures
- Real world: 3% to 6% of SCSI drives fail per year
  - 3400 - 6800 FIT or 150,000 - 300,000 hour MTTF [Gray & van Ingen 05]
- 3% to 7% of ATA drives fail per year
  - 3400 - 8000 FIT or 125,000 - 300,000 hour MTTF [Gray & van Ingen 05]

Homework

- Read 1.11
- Question 1.8 & 1.11