DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

5008: Computer Architecture

Chapter 1 – Fundamentals of Computer Design

Review from Last Lecture

- Computer Architecture >> instruction sets
- 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
- Computer Architecture skill sets are different
  - 5 Quantitative principles of design
  - Quantitative approach to design
  - Solid interfaces that really work
  - Technology tracking and anticipation
- Computer Science at the crossroads from sequential to parallel computing

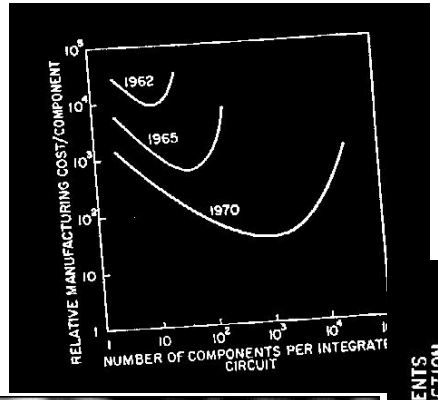
Outline

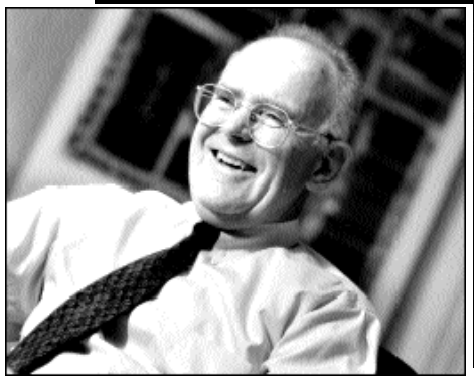
- Review
- Technology Trends:
  - Culture of tracking, anticipating and exploiting advances in technology
- Careful, quantitative comparisons:
  1. Define and quantity power
  2. Define and quantity dependability
  3. Define, quantity, and summarize relative performance
  4. Define and quantity relative cost

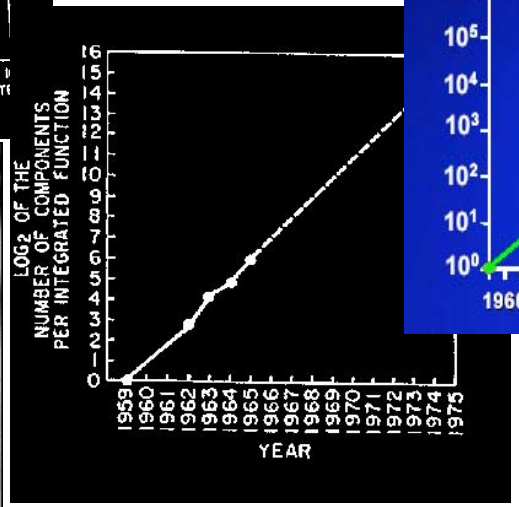
Tracking Technology Performance Trends

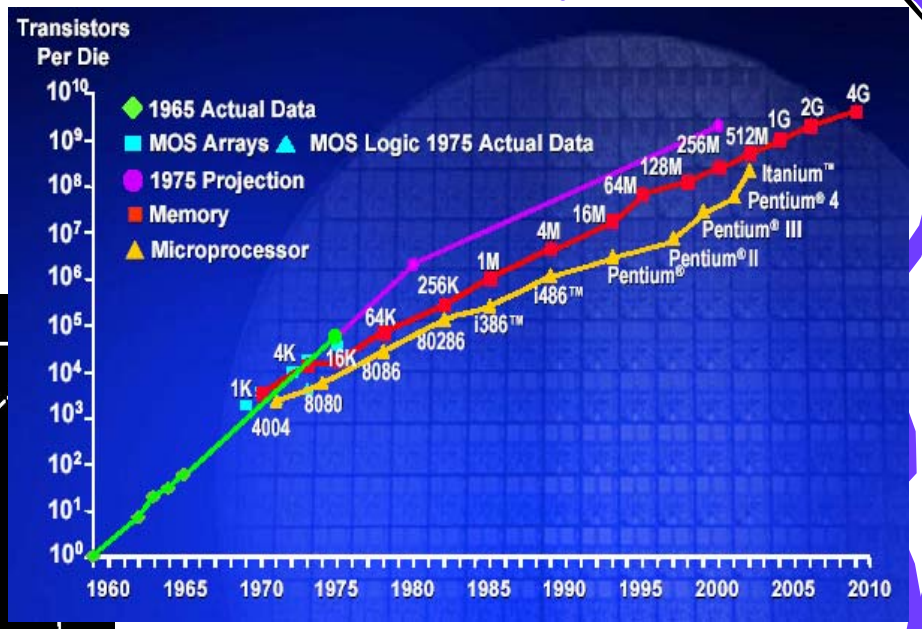
- Drill down into 4 technologies:
  - Disks,
  - Memory,
  - Network,
  - Processors
- Performance Milestones in each technology
  - Compare for Bandwidth vs. Latency improvements in performance over time
  - Bandwidth (Throughput): number of events per unit time
    - E.g., M bits / second over network,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

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



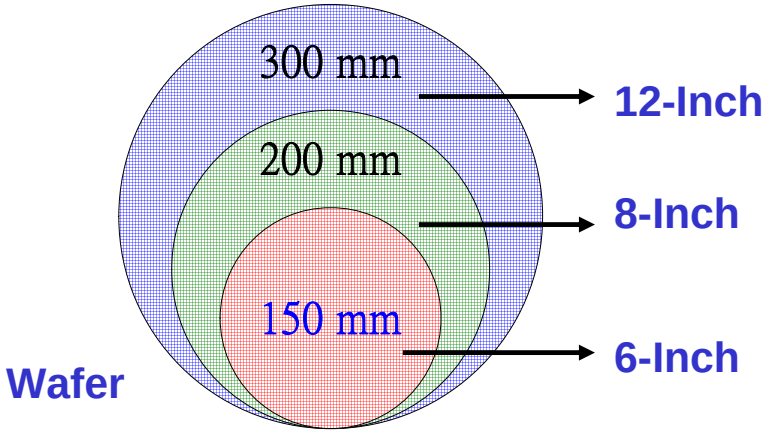


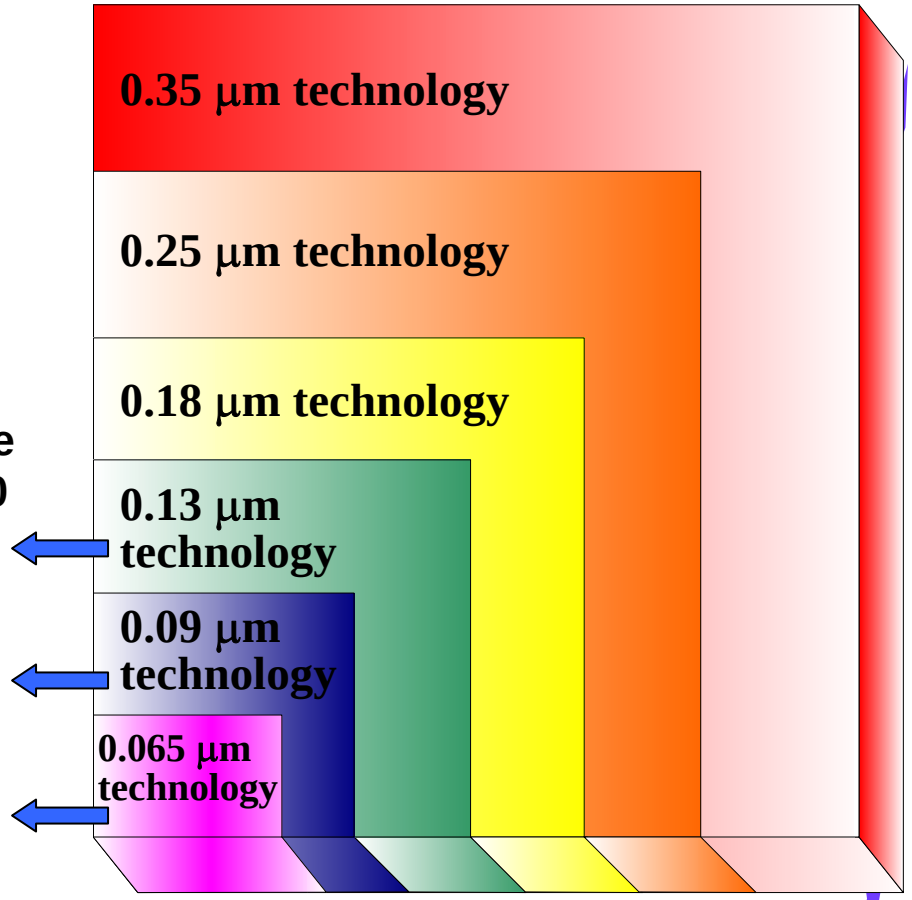




- Gordon Moore, Electronics, 1965
  - # on transistors / cost-effective integrated circuit double every N months ($12 \leq N \leq 24$)

Feature Technology and Size





When compared to the 0.18-micron process, the new 0.13-micron process results in less than 60 percent the die size and nearly 70 percent improvement in performance

The 90-nm process will be manufactured on 300mm wafers

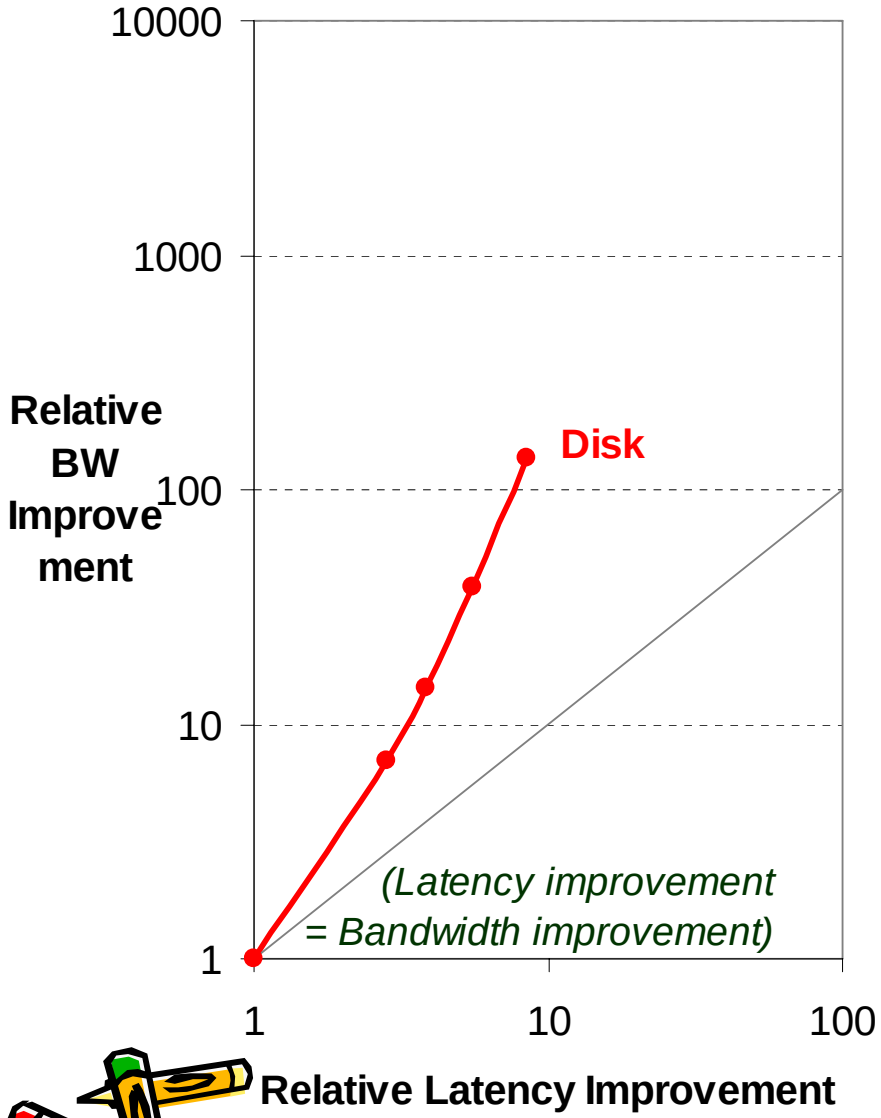
NEC devises low-k film for second-generation 65-nm process

Disks: Archaic vs. Modern

- 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)**

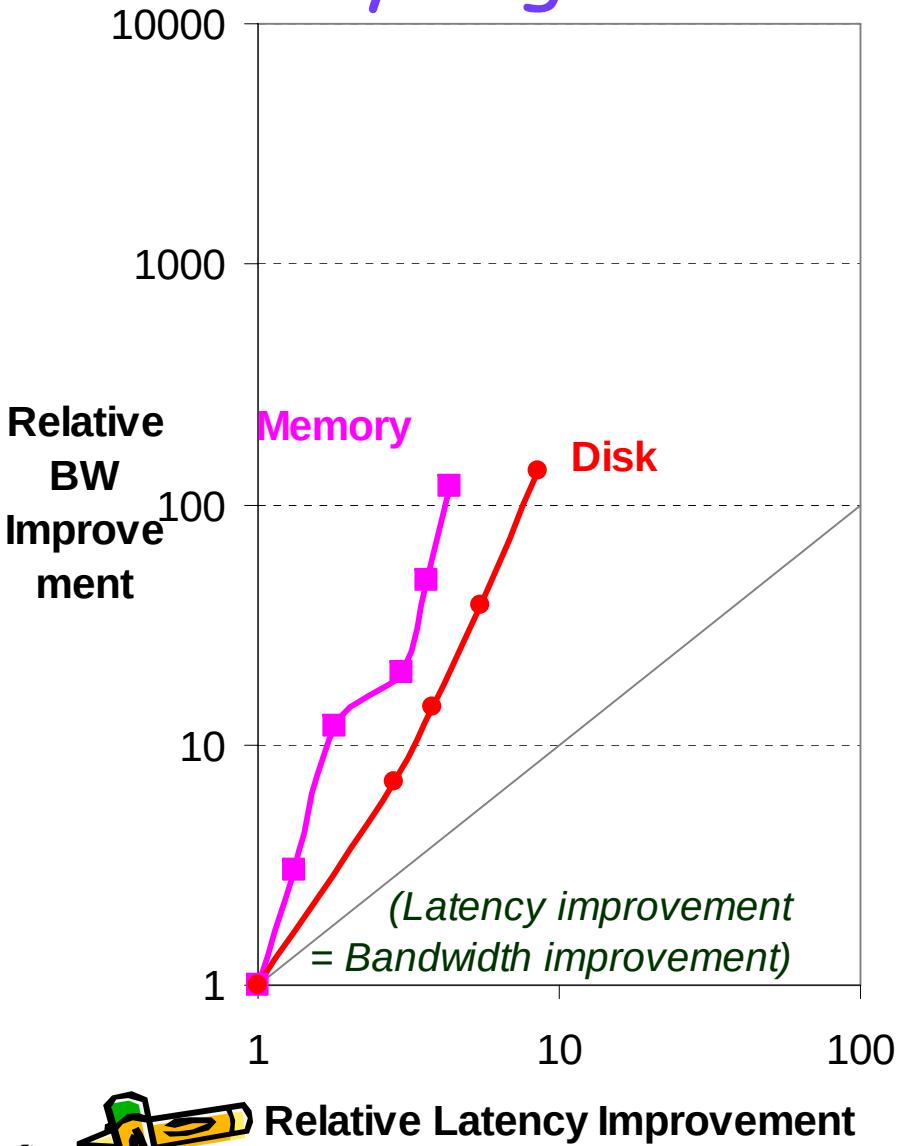
(latency = simple operation w/o contention
BW = best-case)

Memory: Archaic vs. Modern

- 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)**

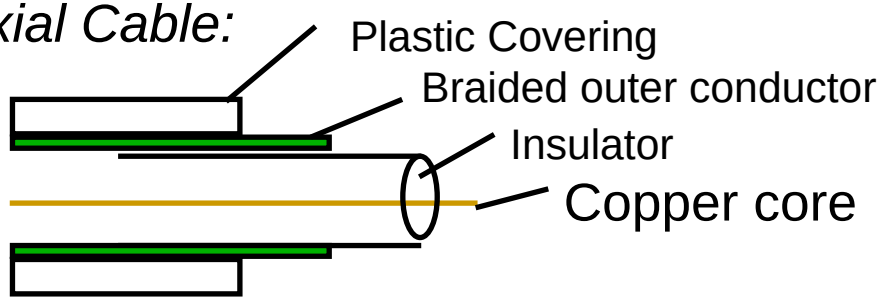
(latency = simple operation w/o contention
BW = best-case)

LANs: Archaic vs. Modern

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

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

Coaxial Cable:

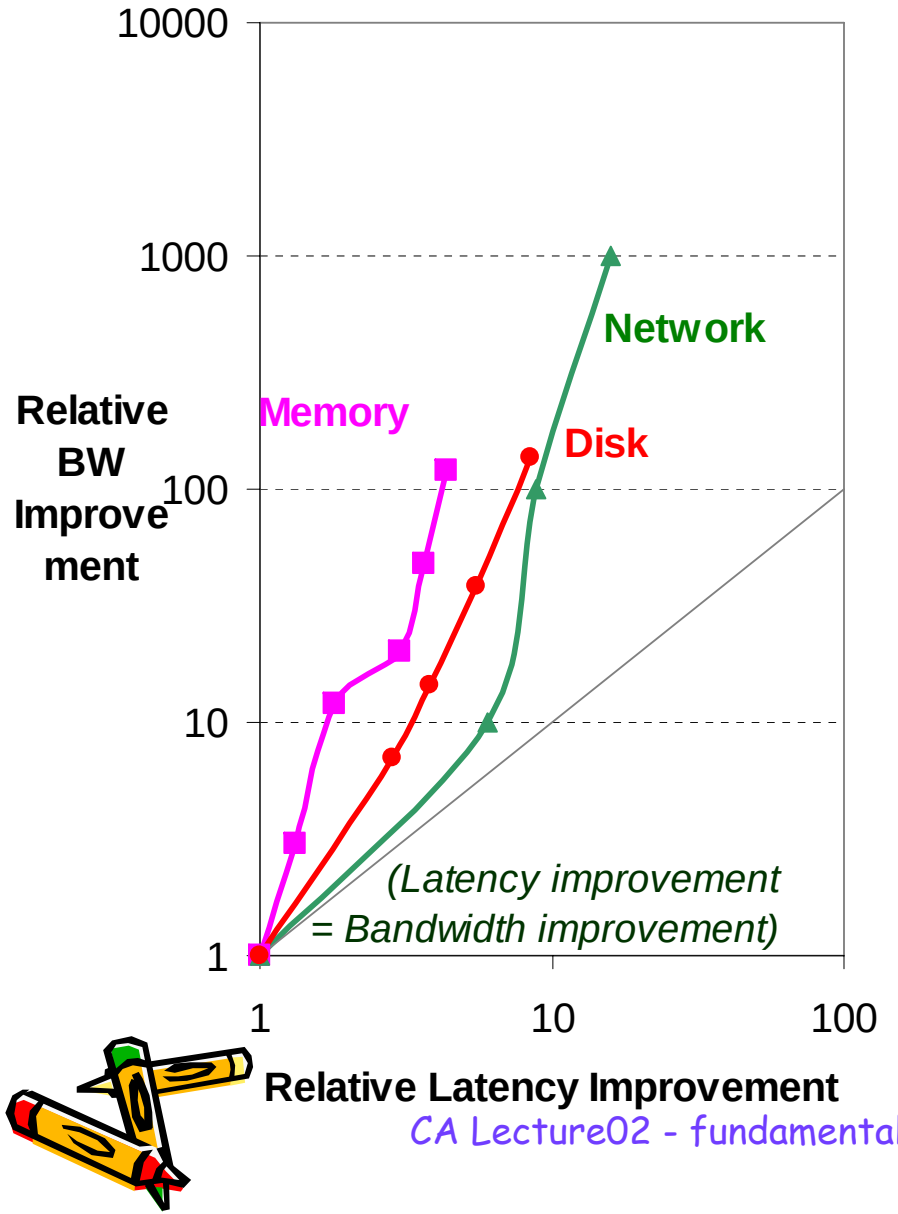


"Cat 5" is 4 twisted pairs in bundle

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)

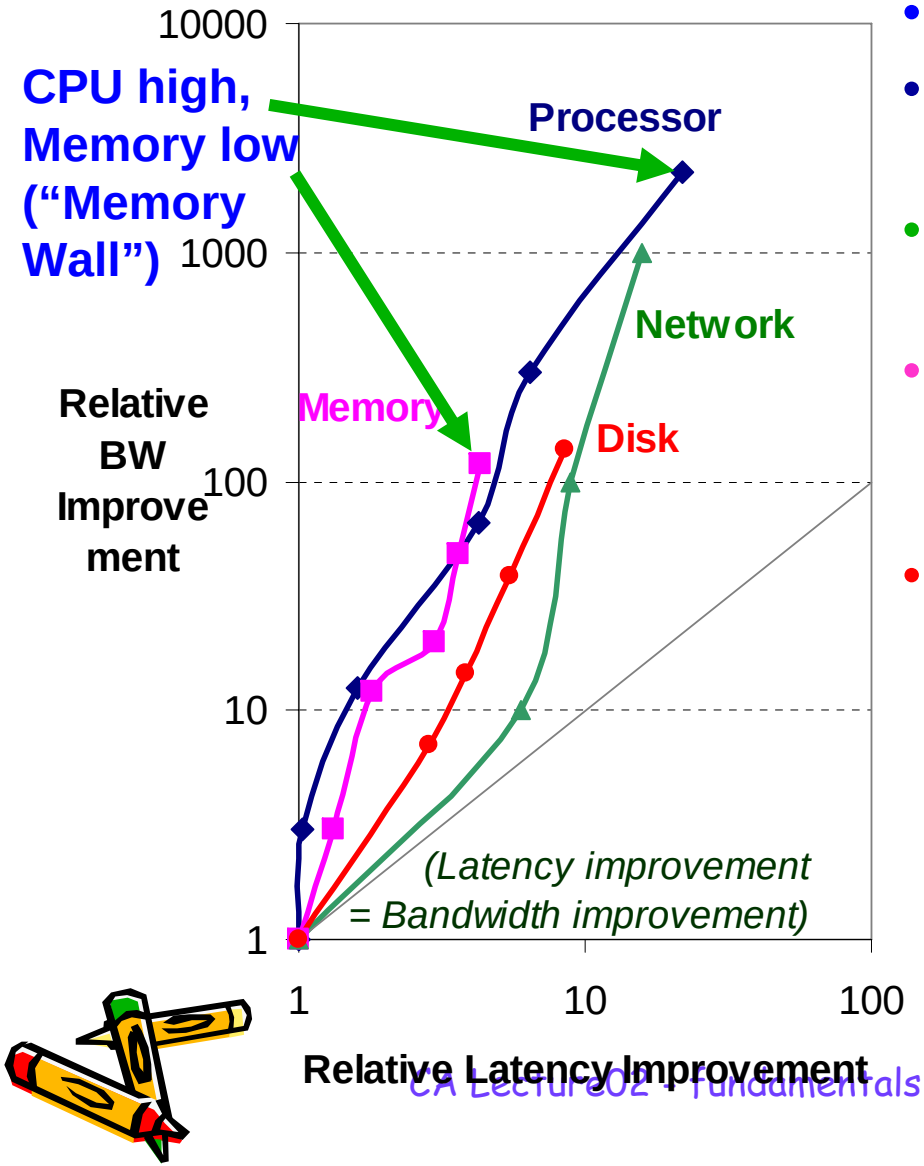
(latency = simple operation w/o contention
BW = best-case)

CPUs: Archaic vs. Modern

- 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)**

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

Trends in Technology

- 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

Summary of Technology Trends

- For disk, LAN, memory, and microprocessor, bandwidth improves by square of latency improvement
  - In the time that bandwidth doubles, latency improves by no more than 1.2X to 1.4X
- Lag probably even larger in real systems, as bandwidth gains multiplied by replicated components
  - Multiple processors in a cluster or even in a chip
  - Multiple disks in a disk array
  - Multiple memory modules in a large memory
  - Simultaneous communication in switched LAN
- HW and SW developers should innovate assuming Latency Lags Bandwidth
  - If everything improves at the same rate, then nothing really changes
  - When rates vary, require real innovation

Outline

- Review
- Technology Trends:
  - Culture of tracking, anticipating and exploiting advances in technology
- Careful, quantitative comparisons:
  1. Define and quantity power
  2. Define and quantity dependability
  3. Define, quantity, and summarize relative performance
  4. Define and quantity relative cost

Define and Quantity Power

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

$$P_{dynamic} = \frac{1}{2} \times C_{load} \times V^2 \times F$$

- For mobile devices, energy, instead of power, is the proper metric

$$E = C \times V^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Dropping voltage helps both, so went from 5V to 1V
- As moved from one process to the next, the increase in the number of transistors switching, the frequency, dominates the decrease in load capacitance and voltage

  ➔ an overall growth in power consumption and energy
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Define and Quantity Power

- Because leakage current flows even when a transistor is off, now *static power* important too

$$P_{static} = C_{static} \times V$$

- Increasing the number of transistors increases power even if they are turned off
- Leakage current increases in processors with smaller transistor sizes
- 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

Outline

- Review
- Technology Trends:
  - Culture of tracking, anticipating and exploiting advances in technology
- Careful, quantitative comparisons:
  1. Define and quantity power
  2. Define and quantity dependability
  3. Define, quantity, and summarize relative performance
  4. Define and quantity relative cost

Dependability ?

- Old CW: ICs are one of the most reliable components of a computer
- New CW: On the transistor feature size down to 65nm or smaller, both transient faults and permanent faults will become more commonplace.
- Computers are designed and constructed at different levels of abstraction

➔ One difficult question is deciding when a system is operating properly?

Define and quantity dependability

- How decide when a system is operating properly?
- [Internet Services] Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking 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
  - Transitions:
  - Failure = transition from state 1 to state 2
  - Restoration = transition from state 2 to state 1

Define and Quantity Dependability

- 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 = measure of the service accomplishment with respect to the alternation between the 2 states
- Module availability = MTTF / (MTTF + MTTR)

Outline

- Review
- Technology Trends:
  - Culture of tracking, anticipating and exploiting advances in technology
- Careful, quantitative comparisons:
  1. Define and quantity power
  2. Define and quantity dependability
  3. Define, quantity, and summarize relative performance
  4. Define and quantity relative cost

Measuring Performance

- One computer is faster than another
  1. Response time or execution time or latency
     - For computer user's interest
     - The time between the start and completion of an event
  2. Throughput or bandwidth
     - For computer center manager's interest
     - The total amount of work done in a given time
- Note, conventionally,
  - Response time, execution time, and throughput are used when an entire computing task is discussed
  - Latency and bandwidth are used when discussing a memory system

Performance Enhancements

1. Fast clock cycle time
   - improve execution time and throughput
2. Multiple processors for separate tasks
   - only throughput increases
3. Parallel processing
   - improve execution time and throughput

Performance Measurement

- Two different machines X and Y.
  X is *n* times faster than Y

$$\frac{\text{Execution time}_{Y}}{\text{Execution time}_{X}} = n$$

Since execution time is the reciprocal of performance

$$\frac{\text{Execution time}_{Y}}{\text{Execution time}_{X}} = n = \frac{\text{Performance}_{X}}{\text{Performance}_{Y}}$$

- Says *n* -1 = *m*/100
  This concludes that X is m% faster than Y

Measuring Performance

- Lower time → high performance ?
- Response time, elapsed time
  - The latency to complete a task, including disk accesses, memory accesses, I/O activities, OS overhead,…,-- everything!
  - Not an appropriate measure for multiprogramming
- CPU time
  - The time that the CPU is computing
  - User CPU time
    - spent in the user's program
  - System CPU time
    - spent in the OS
  - Example: UNIX time command

    90.7 u 12.9s 2:39 65%
    - User CPU time is 90.7 sec, system CPU time is 12.9 sec, elapsed time is 159sec
    - (90.7+12.9)/159 = 65%
    - For I/O or running other programs or both: ~1/3 elapsed time

5 Levels of Programs Used for Evaluation

» Listed below in decreasing order of accuracy of prediction……

- Real applications
  - Portability, compiler, OS
- Modified (or scripted) applications
  - To enhance portability or to focus on one particular aspect of system performance
- Kernels
  - Small, key pieces from real programs
  - Best way to isolate performance of individual features
- Toy benchmark
  - 10~100 code lines
  - Usually, the user already knows the evaluation results
- Synthetic benchmark
  - Whetstone, Dhrystone
  - Be created artificially to match an average execution profile
  - No user runs it

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

Reporting Performance Results

- Should be reproducibility
- To describe exactly the software system being measured and whether any special modifications have been made.
  - Baseline performance measurement
  - Optimized performance measurement
  - Source code modifications ?
  - Hand-generated assembly languages ?

Compare/Summarize Performance

- WLOG, 2 different ways
  1. Arithmetic mean

$$\frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

     - Time_i is the execution time for the ith program in the workload
     - **Weighted arithmetic mean**

$$\sum_{i=1}^{n}\text{Weight}_i \times \text{Time}_i$$

     - Weight_i factors add up to 1
  2. Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n}\text{Execution time ratio}_i}$$

     - To normalize to a reference machine (e.g. SPEC)
     - Execution time ratio$_i$ is the execution time normalized to the reference machine, for the ith program

Example

	Computers			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1	1.00	10.00	20.00	0.50	0.909	0.999
Program P2	1000.00	100.00	20.00	0.50	0.091	0.001
Arithmetic mean: W(1)	500.50	55.00	20.00			
Arithmetic mean: W(2)	91.91	18.19	20.00			
Arithmetic mean: W(3)	2.00	10.09	20.00			

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.00	10.00	20.00	0.10	1.00	2.00	0.05	0.50	1.00
Program P2	1.00	0.10	0.02	10.00	1.00	0.20	50.00	5.00	1.00
Arithmetic mean	1.00	5.05	10.01	5.05	1.00	1.10	25.03	2.75	1.00
Geometric mean	1.00	1.00	0.63	1.00	1.00	0.63	1.58	1.58	1.00
Total time	1.00	0.11	0.04	9.10	1.00	0.36	25.03	2.75	1.00

1. The arithmetic mean performance varies from ref. to ref.
2. The geometric mean performance is consistent

How Summarize Suite Performance (1/5)

- Arithmetic average of execution time of all pgms?
  - But they vary in operating 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 =

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

Summarizing Performance

Example:

	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Total Time	1001	110	40

- For single program
  - A is 10 times faster than B for program P1
  - C is 50 times faster than A for program P2
- Total execution time (of two programs)
  - If P1 and P2 are running equal times
    - B is 9.1 times faster than A for programs P1 and P2
    - C is 2.75 times faster than B for program P1 and P2

Remark

$$\text{e.g. } 1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

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

- SPECRatio is just a ratio rather than an absolute esecution time
- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

Geometric Mean

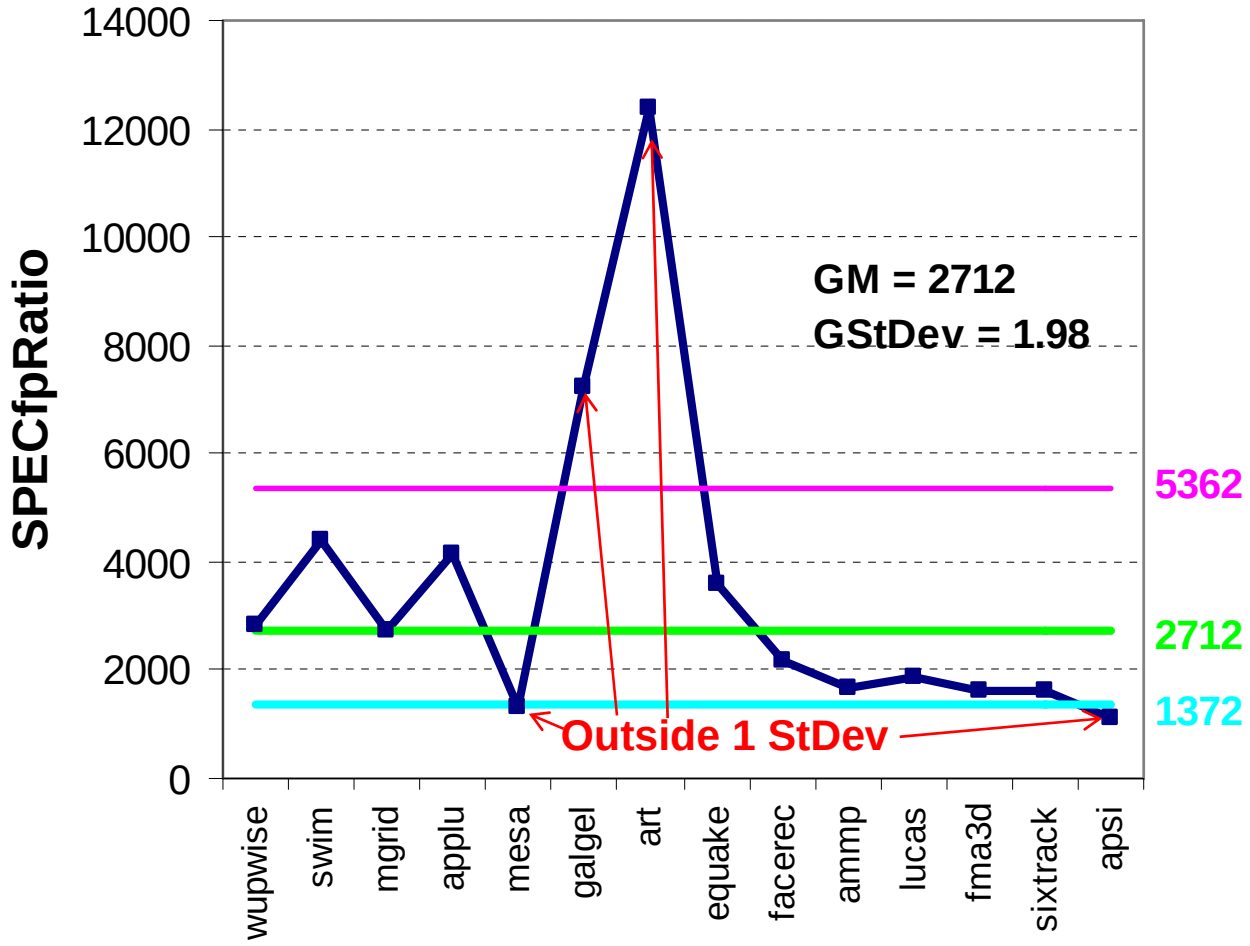
- Geometric mean of the ratios is the same as the ratio of the geometric means
- Choice of reference computer is irrelevant
  - The geometric mean would be less misleading than the arithmetic mean
- The geometric mean does not predict execution time, however.
  - In general, there is no real workload that will match the performance predicted by the geometric mean method.
  - It encourages the designer to pay more attentions to the benchmark where its performance is easiest to improve rather than on the benchmarks that are slowest.

How Summarize Suite Performance

- Does a single mean well summarize performance of programs in benchmark suite?
- Probability and Statistic Theory says:
  - If we characterize the variability of the distribution, using the standard deviation, we can decide whether the mean is likely to be a good predictor

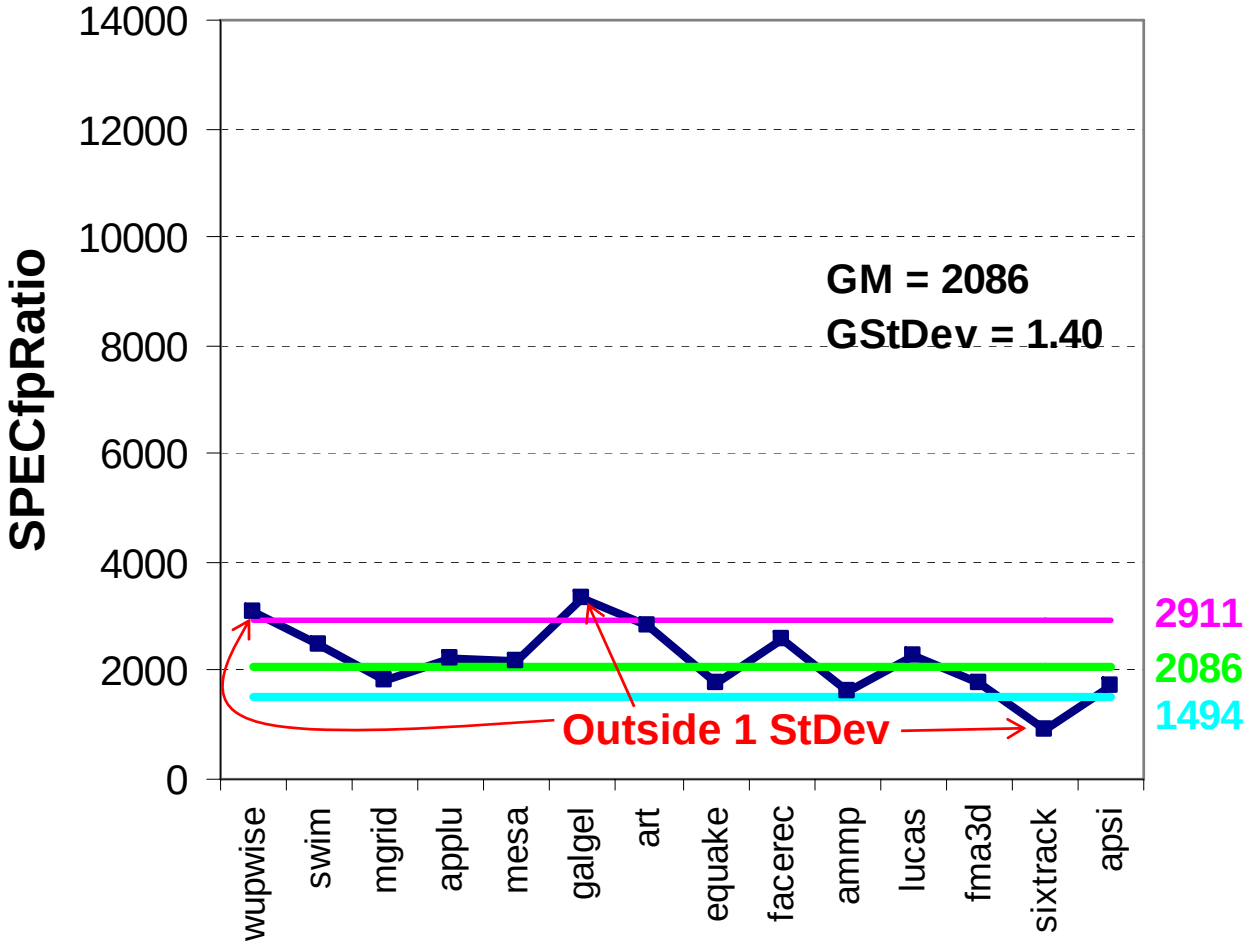
Example (1/2)

- Itanium 2



Example (2/2)

- AMD Athlon



Remarks

- 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 $[mean / gstdev^2, mean \times gstdev^2]$

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 with a lognormal distribution:
  - 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%)

Outline

- Review
- Technology Trends:
  - Culture of tracking, anticipating and exploiting advances in technology
- Careful, quantitative comparisons:
  1. Define and quantity power
  2. Define and quantity dependability
  3. Define, quantity, and summarize relative performance
  4. Define and quantity relative cost

Cost, Price, and Their Trends

- Price: what you sell a finished good for
- Cost: amount spent to produce it, including overhead
- The impact of time, volume, and commodification
  - Learning curve: manufacturing costs decrease over time (max. measured by yield)
  - Volume decreases the cost
  - Commodities: sell on the grocery stores, multiple suppliers.

Cost of an IC

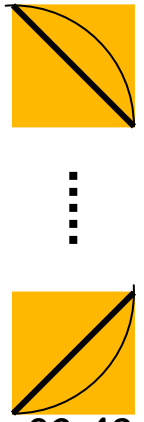
- A wafer is tested and chopped into dies

$$C_{\text{die}} = \frac{C_{\text{wafer}}}{\text{Die per wafer} \times \text{Die yield}}$$

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

- The die is still tested and packaged into IC

$$C_{\text{IC}} = \frac{C_{\text{die}} + C_{\text{testing die}} + C_{\text{packaging and final test}}}{\text{Final test yield}}$$



IC Yield

- A simple empirical model of IC yield (defect, randomly distributed):

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

- Wafer yield is almost 100%
- Defect per unit area ≈ 0.4~0.8 / cm²
- α = 4.0 for multi-level metal CMOS

Example

- A 30cm wafer, for a die that is 0.7cm on a side.
- Then

$$\text{Die per wafer} = \frac{\pi \times (30/2)^2}{0.49} - \frac{\pi \times 30}{\sqrt{2 \times 0.49}} = 1347$$

- Assume that a defect density of 0.6/cm², then

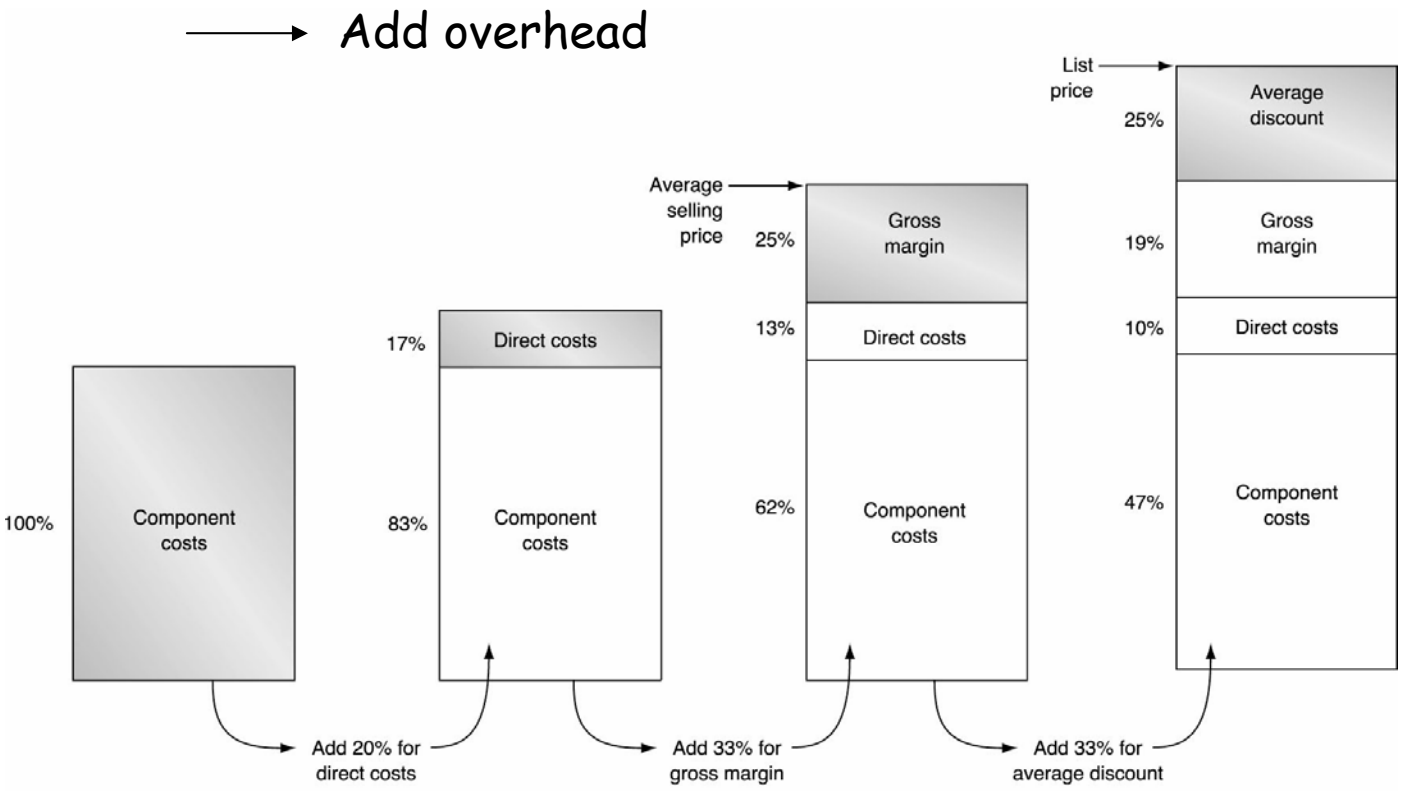
$$\text{Die yield} = \left(1 + \frac{0.6 \times 0.49}{4.0}\right)^{-4} = 0.75$$

About 1010 good dies

Cost Versus Price

- Direct costs (adds 10~30% to component cost)
  - recurring costs…
  - The costs directly related to making a product, including labor costs, purchasing components, warranty, …
- Gross margin (indirect cost)
  - non-recurring costs…
  - The company's overhead, such as R&D, marketing, sales, building rental, pretax profit, taxes, …
  - 10~45% of the ASP
- Average selling price (ASP)
  - Component cost + direct cost + gross margin
- List price
  - ASP is typically 50~75% of the list price
- Selling volume ↑ ⇒ cost ↓

The components of price for a $1000 PC



Concluding Remarks

- Tracking and extrapolating technology part of architect's responsibility
- Expect Bandwidth in disks, DRAM, network, and processors to improve by at least as much as the square of the improvement in Latency
- Quantify dynamic and static power
  - Capacitance x Voltage^2 x frequency, Energy vs. power
- Quantify dependability
  - Reliability (MTTF, FIT), Availability (99.9...)
- Quantify and summarize performance
  - Ratios, Geometric Mean, Multiplicative Standard Deviation
- Read Appendix A