Topic 1: **Fundamentals of Computer Design**

Fundamentals of Computer Design

- Introduction

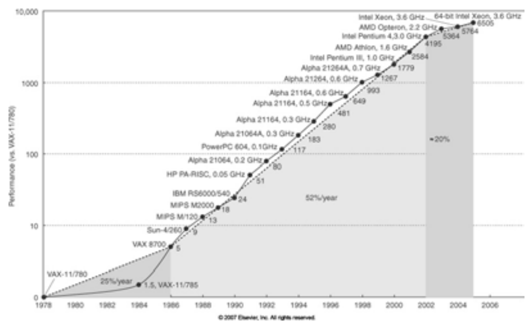
Pentium 3 Coppermine (2000)

Cray YMP (1988)

Fundamentals of Computer Design

	Pentium 3 (Coppermine)	Cray YMP
Type	Desktop	Supercomputer
Year	2000	1988
Clock	1130 MHz	167 MHz
MIPS	> 1000 MIPS	< 50 MIPS
Cache	256 KB	0.25 KB
Memory	512 MB	256 MB
Cost	$2000	$1,000,000

Fundamentals of Computer Design



Fundamentals of Computer Design

- 16 years: overall progress was a product of three factors of improvement:
  - Technology, Architecture, Compiler.
    - 1.15 x 1.15 x 1.15 = **1.52 / year**.
    - Architecture: Instruction Level Parallelism (ILP), caches, RISC
- Now: 20% / year
  1. power and heat
  2. running out of ILP to exploit
  3. memory latency
     - Reduce clock frequency
     - Multicore

Classes of computers

Feature	Desktop	Server	Embedded
Price of system	\$500-\$5000	\$5000-\$5,000,000	\$10-\$100,000 (including network routers at the high end)
Price of microprocessor module	\$50-\$500 (per processor)	\$200-\$10,000 (per processor)	\$0.01-\$100 (per processor)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Defining Computer Architecture

- ***Implementation***: how an abstract description is turned into hardware.
- The ***instruction set architecture*** (ISA) is such abstraction.
  - Specification of the functional behavior of a processor
  - HW / SW interface.
  - Assembly language, instruction format, addressing modes, programming model
    - ADD R1, R2, R3
- Examples of ISAs
  - MIPS64, ARM, PowerPC

Defining Computer Architecture

- 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

Defining Computer Architecture

– ***Organization***: high level aspects of the design
  - memory, bus structure, pipeline, cache, branch predictors…etc.
  - design of these sometimes called “micro-architecture”.

– It is possible to *implement* the same *instruction set architecture* using different *organizations*, resulting in different systems.
  - E.x. Different Bus, memory organization, pipeline structure and so-on. E.g. AMD Opteron 64 and Intel Pentium 4 (same ISA)

Defining Computer Architecture

– ***Hardware*** refers to the specifics of an implementation.
  - For example the Pentium 4 and the Mobile Pentium have different hardware.
  - Different clock frequencies, different memory systems
  - Other differences: fabrication technology, packaging, clock, etc…

– It is even possible to *emulate* a function normally carried out in hardware (say floating point calculations) using software (lists of instructions).

Task of the computer architect

Functional requirements	**Typical features required or supported**
Application area	*Target of computer*
General-purpose desktop	Balanced performance for a range of tasks, including interactive performance for graphics, video, and audio (Ch. 2,3,5, App. B)
Scientific desktops and servers	High-performance floating point and graphics (App. I)
Commercial servers	Support for databases and transaction processing; enhancements for reliability and availability; support of scalability (Ch. 4, App. B, E)
Embedded computing	Often requires special support for graphics or video (or other application-specific extension); power limitations and power control may be required (Ch. 2, 3, 5, App. B)
Level of software compatibility	*Determines amount of existing software for computer*
At programming language	Most flexible for designer; need new compiler (Ch. 4, App. B)
Object code or binary compatible	Instruction set architecture is completely defined-little flexibility-but no investment needed in software or porting programs
Operating system requirements	*Necessary features to support chosen OS (Ch. 5, App. E)*
Size of address space	Very important feature (Ch. 5); may limit applications
Memory management	Required for modern OS; may be paged or segmented (Ch. 5)
Protection	Different OS and application needs; page vs. Segment; virtual machines (Ch. 5)
Standards	*Certain standards may be required by marketplace*
Floating point	Format and arithmetic: IEEE754 standard (App. I), special arithmetic for graphics or signal processing
I/O interfaces	For I/O devices: Serial ATA, Serial Attach SCSI, PCI Express (Ch. 6, App. E)
Operating systems	UNIX, Windows, Linux, CISCO IOS
Networks	Support required for different networks: Ethernet, Infiniband (App. E)
Programming languages	Languages (ANSI C, C++, Java, FORTRAN) affect instruction set (App. B)

Trends in Technology

- Valid over long time periods, e.g. ISA can last decades
- Integrated circuit logic technology

• **Transistor Density:**	**~ 35%**
• **Die size:**	**~ 10-20%**
⇒Transistors per chip:	~ 40-55%

- Semiconductor DRAM

• **Capacity**	**~ 40%**

- Magnetic disk technology

• **Density**	**~ 30% (2004)**

- Network technology

Performance Trends: Bandwidth over Latency

- Compare ~1980 Archaic vs. ~2000 Modern
- 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

Performance Trends: Bandwidth over Latency

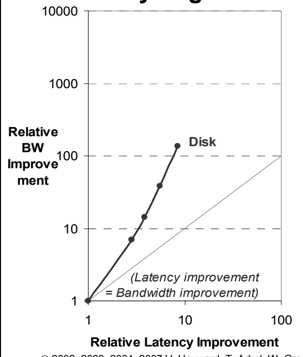
- Disks: Archaic v. Modern

- 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

Performance Trends: Bandwidth over Latency

- **Latency Lags Bandwidth (for last ~20 years)**



Performance Milestones

Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

Performance Trends: Bandwidth over Latency

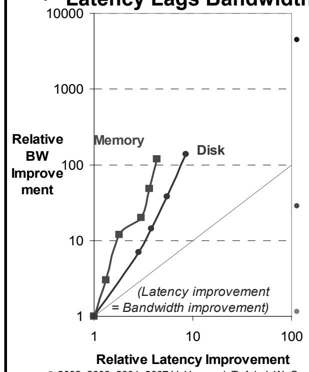
- Memory: Archaic v. Modern

- 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)

Performance Trends: Bandwidth over Latency

- **Latency Lags Bandwidth (for 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)

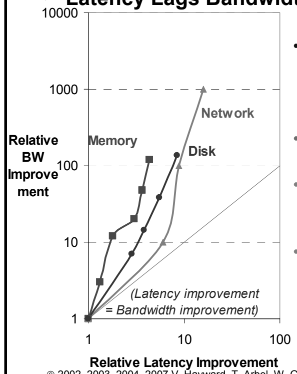
Performance Trends: Bandwidth over Latency

- LANs: Archaic v. Modern

Ethernet 802.3	Ethernet 802.3ae
Year of Standard: 1978	Year of Standard: 2003
10 Mbits/s link speed	10,000 Mbits/s link speed (1000X)
Latency: 3000 μsec	Latency: 190 μsec (15X)
Shared media	Switched media
Coaxial cable	Category 5 copper wire

Performance Trends: Bandwidth over Latency

- **Latency Lags Bandwidth (for 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)

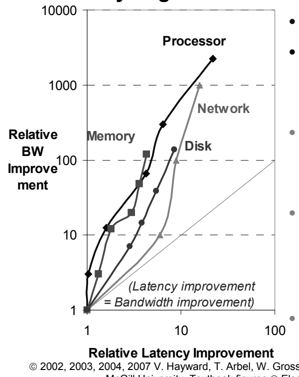
Performance Trends: Bandwidth over Latency

- CPUs: Archaic v. Modern

1982 Intel 80286	2001 Intel Pentium 4
12.5 MHz	1500 MHz (120X)
2 MIPS (peak)	4500 MIPS (peak) (2250X)
Latency 320 ns	Latency 15 ns (20X)
134,000 xtors, 47 mm^2	42,000,000 xtors, 217 mm^2
16-bit data bus, 68 pins	64-bit data bus, 423 pins
Microcode interpreter, separate FPU chip	3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
(no caches)	On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Performance Trends: Bandwidth over Latency

- **Latency Lags Bandwidth (for 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)

Trends in Technology

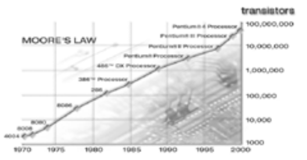
Moore's Law

2x number of transistors every 12-18 months

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain for nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.
"I believe that such a large circuit can be built on a single wafer."

--- Gordon Moore, 1965

	Year of Introduction	Transistors
4004	1971	2,250
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
286	1982	120,000
386™	1985	275,000
486™ DX	1989	1,180,000
Pentium®	1993	3,100,000
Pentium II	1997	7,500,000
Pentium III	1999	24,000,000
Pentium 4	2000	42,000,000



Source: Intel

Trends in Technology

- ***Scaling of Transistor Performance and Wires***
  - Feature size (min size of transistor or wire)
    - 1971: 10 μm
    - 2001: 0.18 μm
    - 2002: 0.13 μm
    - 2003: 0.10 μm
    - 2007: 65 nm
  - Quadratic increase in density, linear increase in performance

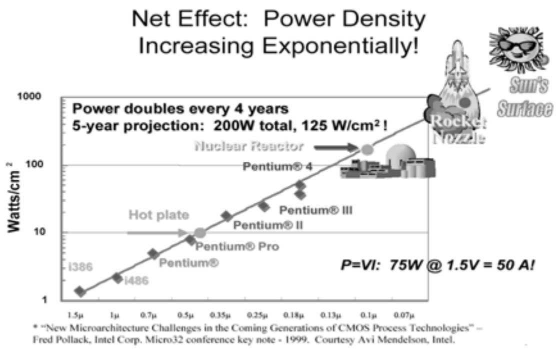
Trends in Technology

- ***Scaling of Transistor Performance and Wires***
  - $\Rightarrow$ *Architectural improvement!*
    - 8, 16, 32, 64 bit architectures (buses, ALU’s)
    - Pipelines and caches
    - Transistor performance benefits from smaller resistance and capacitance.
  - Interconnect propagation delay major problem.
    - e.g. Pentium 4 accounts for propagation of signals across chip.

Trends in Technology

- ***Trends in Power in Integrated Circuits***
  - Dynamic power: $P_{dynamic} = 1/2 \times f \times C \times V^2$
    - f = clock frequency, C = capacitance
    - V = voltage (5V-> 3.3V-> < 1 V)
  - 3.2 GHz Pentium 4 Extreme Edition
    - $\rightarrow$135 Watts: limits of air cooling
  - Temperature diodes used to regulate voltage and clock frequency
  - Shutdown parts of chip
  - Portable computing requires low power.
  - Static Power (leakage): P_{static} = Current $_{static}$ x V (25%!!!!!)

Trends in Technology



* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999. Courtesy Avi Mendelson, Intel.

Trends in Technology

- ***Example:***
- ***Some microprocessors have adjustable voltage. A 15% reduction in voltage results in a 15% reduction in frequency.***
- ***What is the impact on the dynamic power?***

Trends in Technology

- ***Other "Famous" Predictions***
  - **"There is no reason for any individual to have a computer in his home."**

    Kenneth H. Olson, President of DEC,

    Convention of the World Future Society, 1977
  - **"640 kilobytes (of computer memory) ought to be enough for anybody."**

    Bill Gates, Founder and head of Microsoft, 1981

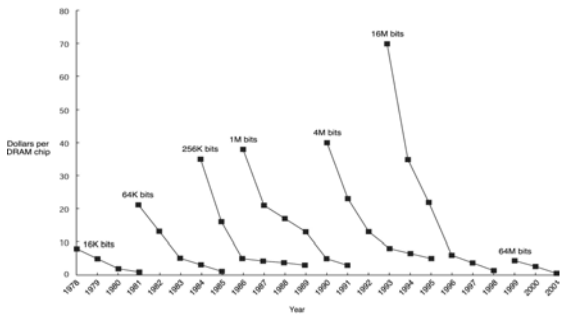
Trends in Cost

- What is the nature of the cost-performance tradeoff?
- Driven by cost of components
  - one important aspect is their change over time.
- ***Time and volume***
- Manufacturing learning curve –best measured by yield (# good chips / total # of chips made)

Trends in Cost

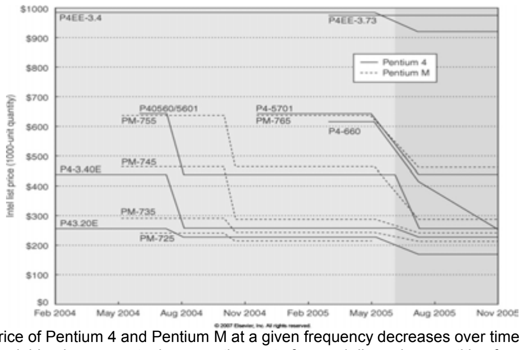
- ***Yield*** improves with time.
- Doubling the yield halves the cost.
  - Example: DRAM chips have strange business behaviors because of rapid changes (40 % / year drop in price per megabyte over long term) – sometimes sell at loss!
- Microprocessors are less predictable.
  - Roughly cost decreases 10% for volume doubling (learning curve improves with each chip made, efficiency increase, amortize development costs)
  - Expansion of low-end market has produced "commoditization" with fierce competition and razor-thin margins.

Trends in Cost



Prices of six generations of DRAMs over time since 1977 in dollars. This shows the importance of the learning curve.

Trends in Cost

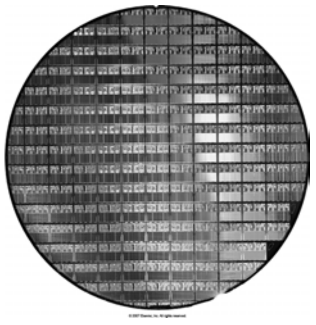


Price of Pentium 4 and Pentium M at a given frequency decreases over time as yield enhancements decrease the cost of a good die and competition forces price reductions.

Cost of an Integrated Circuit

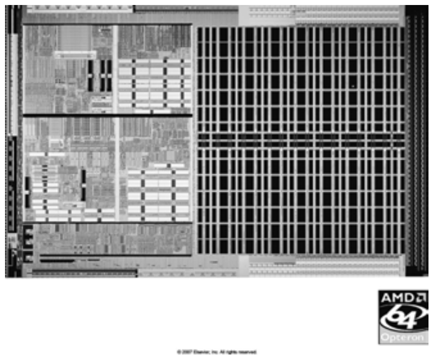
- **Manufacturing Steps.**
  - Silicon Crystal Growth extracted from molten silicon bath
  - Processed (cleaned to very high level of purity) into cylinder
  - Cylinder sliced to make wafers
  - Wafers cleaned, polished and chemically processed
  - Long sequence of steps involving deposit and removal of substances to etch the circuit according to patterns specified by optical masks.
  - Dies cut, tested and packaged.

Cost of an Integrated Circuit



300mm AMD Opteron WAFER in 90nm process(117 processors)

Cost of an Integrated Circuit



Cost of an Integrated Circuit

- ***Cost model***

$$ICCost = \frac{DieCost + DieTestCost + PackagingAndTestCost}{FinalYield}$$

$$DieCost = \frac{WaferCost}{DiesPerWafer \times DieYield}$$

$$DiesPerWafer = \frac{\pi \times WaferRadius^2}{DieArea} - \frac{\pi \times WaferDiameter}{\sqrt{2 \times DieArea}}$$

$$DieYield = WaferYield \times \left(1 + \frac{DefectDensity \times DieArea}{\alpha}\right)^{-\alpha}$$

DieYield is from an empirical formula where α reflect the number of process steps (complexity).
α can be of the order of 3 or 4.
DefectDensity is of the order of 0.4--0.8/cm^2.

Cost of an Integrated Circuit

- Example:
  - Find the number of dies per 300 mm wafer for a die 1.5 cm on a side

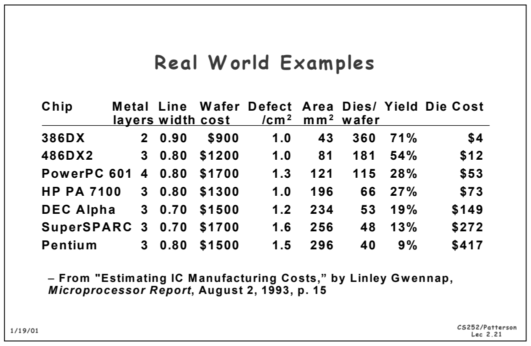
Cost of an Integrated Circuit

- Example:
  - Find the die yield for dies: i) 1.5 cm on a side, and ii) 1.0 cm on a side

Cost of an Integrated Circuit

- Example:
  - $2000 / wafer, 350 raw dies / wafer
  - 60% good dies, $80 to test wafer,
  - $4/unit to package and final test, 97% final test yield
- *DieCost* =
- *DieTestCost* (good dies) =
- *ICCost* =

Cost of an Integrated Circuit



Real World Examples

Chip	Metal layers	Line width	Wafer cost	Defect /cm^2	Area mm^2	Dies/ wafer	Yield	Die Cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

– From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

1/19/01 CS252/Patterson Lec 2.21

Cost of an Integrated Circuit

- **Dependability**
  - – Historically, integrated circuits were very reliable, error rates inside chips was very low.
  - – 65nm and smaller: transient and permanent faults
  - – Service Level Agreement (SLA) – provided pays customer a penalty if the system is down more than a certain number of hours a month
    - Two states:
      - – Service accomplishment (if service is delivers as specified)
      - – Service interruption (if service is different from SLA)
      - – Failures (state 1->2) and restorations (states 2->1).

Cost of an Integrated Circuit

- Quantify: Reliability and Availability
  - – **Module Reliability:** Mean Time To Failure (MTTF)
  - – MTTF^-1 → failure rate (Failures in Time or FIT) reported as failures per billion hours of operation
    - e.g. MTTF = 1,000,000 hours → 10^9/10^6 = 1000 FIT
  - – Service time: Mean Time To Repair (MTTR)

Cost of an Integrated Circuit

- Quantify: Reliability and Availability
  - – Mean Time Between Failures (MTBF)
  - – MTBF = MTTF + MTTR
- **Module Availability:**
  - – Module availability = MTTF / MTBF

Cost of an Integrated Circuit

- **Example:** Disk subsystem
    - 10 disks, each rated at 1,000,000 hour MTTF
    - 1 SCSI controller, 500,000 hour MTTF
    - 1 power supply, 200,000 hour MTTF
    - 1 fan, 200,000 hour MTTF
    - 1 SCSI cable, 1,000,000 hour MTTF
  - – Assume that failures are independent. Assume exponentially distributed lifetimes (the age of a module is not important in the probability of failure, hence the overall failure rate of the collection is the sum of the failure rates of the modules)
  - – Compute the MTTF of the disk subsystem.

Measuring and Reporting Performance

- How to measure performance?
  - There are two aspects:
    - *Response time* (latency) (needed to get a result from the givens).
    - *Throughput* (bandwidth) (how much computation per unit of time).
  - Example: Montreal to Paris

Aircraft	Time (response time)	Speed	Passengers	pmph (throughput)
747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

  - Computer user might be interested in response time, while manager of a server might be interested in throughput.

Measuring and Reporting Performance

- Relative performance: “X is n times faster than Y”

$$n = \frac{\text{ExecTime}_Y}{\text{ExecTime}_X} = \frac{1/\text{Performance}_Y}{1/\text{Performance}_X} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- How do you measure “time”?
- The only consistent and reliable measure of performance is the *execution time of real programs.*

Measuring and Reporting Performance

- ***Different “Times”***
  - *Wall-clock* time is the *elapsed time* to complete a task. This includes I/O, memory, OS overhead, …, everything. With multiprogramming and multitasking (as in UNIX), for a given task, it changes with the load.
  - *CPU time* is the time spent by the CPU on behalf of one task. It is subdivided into *user CPU time* (time spent by the CPU running user code) and *system CPU time* (time spent running OS code on the behalf of the user).

Measuring and Reporting Performance

- ***Different “Times”***
  - The UNIX time command reports all three.
    - For example
    - prompt$ time sleep 5
    - 0.00u 0.02s 0:05.02 0.3%
  - tells us that the sleep 5 command spent (almost) no CPU time to run, 20 milliseconds to execute OS code and that the elapsed was 5.02 s (per the definition of sleep). It also says that (0.0+0.02)/5.02=0.3% of the elapsed time was to do some work.

Measuring and Reporting Performance

- ***What is a task?***
  - We want to be able to predict the performance of a computer: *how do we evaluate performance?*
    1. *Real applications*: C compiler (if you are code developer), TeX if you are a typesetter, Photoshop if you are a graphic designer, Spice if you are an electronic engineer, MatLab, and so-on.
    2. *Kernels*: A general principle about computing (Knuth) is that programs tend to spend most of their time in a very small portion of the code. (For example, the integrator routine in MathLab, searching and sorting while compiling, manipulating matrices in scientific code).

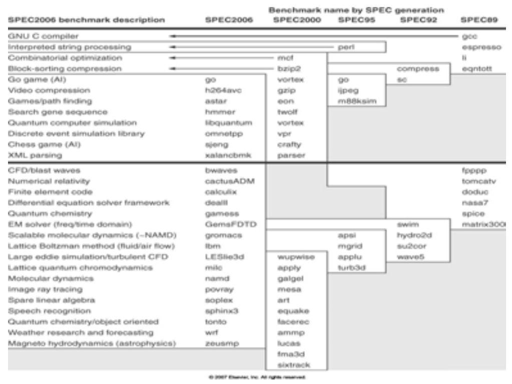
Measuring and Reporting Performance

3. *Toy benchmarks*: Small and interesting programs, Sieve of Eratosthenes (prime numbers), Towers of Hanoi, Puzzles, Quicksort.

- 4. *Synthetic Benchmarks*: Attempt of reproduce the load of a set of programs.

- Benchmark suites: a collection of benchmark applications.
  - SPEC (Standard Performance Evaluation Committee, www.spec.org) is a consortium dedicated to the design of documented benchmarks suites.

Measuring and Reporting Performance



	Benchmark name by SPEC generation				
SPEC2006 benchmark description	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

© 2007 Elsevier, Inc. All rights reserved.

Measuring and Reporting Performance

- ***Summarizing Performance Results of a Benchmark Suite in single number***
  - Consistent summary measure: total execution time
  - Example:

	Computer A	Computer B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- A is 10x faster than B for P1
- B is 10x faster than A for P2
  - Perf. B / Perf. A= Exec Time A / Exec Time B = 1001/110 = 9.1
- B is 9.1 x faster than A if Program 1 and Program 2 are run an equal # of times

Measuring and Reporting Performance

- The arithmetic mean tracks the total execution time. For n programs,

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

- What if some programs have a much longer execution time than others? This will bias the arithmetic mean towards those programs, so we can choose weights:

$$WeightedArithmeticMean = \sum_{i=1}^{n} weight_i time_i$$

Measuring and Reporting Performance

- The SPEC consortium is composed of competing companies who might have their own choice of favorite weights, so one approach is to choose weights to equalize running times:

$$\text{Weight}_i = \frac{1}{\text{Time}_i \times \sum_{i=1}^{n}\left(\frac{1}{\text{Time}_i}\right)}$$

Measuring and Reporting Performance

- **Ratios**
  - Another approach is to normalize execution times relative to a reference computer (SPECRatio = exec time ref / exec time).
  - i.e. if the SPECRatio n of computer A on a benchmark is 1.25 times higher than computer B then
    - 1.25 = SPECRatioA / SPECRatioB =(tref/tA)/(tref/tB)= =tB / tA = perf A / perf B
  - The choice of reference computer is not important.

Measuring and Reporting Performance

- **Ratios**
  - Normalized times to a reference machine must be averaged geometrically:

$$GeometricM\ ean = \sqrt[n]{\prod_{i=1}^{n} sample_i}$$

  - In the case of SPEC, $sample_i$ is the SPECRatio
  - Note that the geometric mean of the ratios is equal to the ratio of the geometric means and the relative results do not depend on the machine taken as reference.

Measuring and Reporting Performance

- We can use the standard deviation to characterize how much variability there is around the mean.

$$stdev = \sqrt{\sum_{i=1}^{n} (sample_i - Mean)^2}$$

$$GeometricMean = \exp(\frac{1}{n} \cdot \sum_{i=1}^{n} \ln(sample_i))$$

- The geometric standard deviation is:

$$gstdev = \exp(\sqrt{\frac{\sum_{i=1}^{n} (\ln(sample_i) - \ln(GeometricMean))^2}{n}})$$

Measuring and Reporting Performance

- Figure 1.14
  - Example on page 37: The geometric means are calculated from data in Figure 1.14 for an Opteron and an Itanium 2. The Itanium 2's mean is higher than the Opteron's (27.12 vs 20.86) but the standard deviation for the Itanium 2 is much higher (1.93 vs. 1.38) indicating that the results differ more widely from the mean and are therefore likely less predictable.

Quantitative Principles of Computer Design

- Take advantage of parallelism
- Principle of locality
- Focus on the common case
  - **Parallelism** is related to performing many operations simultaneously. It is applicable to single processor or to memory management as well. In fact, modern CPUs can executes 10's of instructions simultaneously and perform many memory transactions simultaneously. It is applicable to basic circuits (such as carry-look-ahead adders) to entire systems (many CPU or hard drives operating simultaneously).
  - One example of parallelism at the instruction level that we will exploit is called pipelining. We will overlap the execution of several instructions that the total time to execute the sequence is reduced.

Quantitative Principles of Computer Design

- **The principle of locality:**
  - Typically, a program spends 90% of its time executing only 10% of the code. Most programs are, by definition, highly structured. They rarely use data and instructions in a completely random fashion.
  - Using this principle, we will be able to predict which instructions and data a program will use in the near future with reasonable accuracy based on its accesses in the recent past.

Quantitative Principles of Computer Design

- **The principle of locality:**
  - **Temporal locality** states that recently accessed items are likely to be accessed again in the near future. **Spatial locality** says that items who's addresses (in memory) are near one another tend to be referenced close together in time.
- ***Make the common case fast*.**
  - The impact of an improvement to a system is higher if the occurrence is frequent. So, when making choices in design favor the frequent case over the infrequent ones.

61

Quantitative Principles of Computer Design

- Amdahl's law (1967) captures this quantitatively. It was used to make the case for single CPU processors.
  - Suppose we have an enhancement for a given design.

$$\text{SpeedUp} = \frac{\text{ExecTimeBase}}{\text{ExecTimeEnhanced}}$$

62

Quantitative Principles of Computer Design

- Define *FractionEnhanced*, the fraction of computation time concerned by an enhancement. This fraction is sped up by *SpeedUpEnhanced*.

$$\text{ExecTimeEnhanced} = \text{ExecTimeBase} \times \left((1 - \text{FractionEnhanced}) + \frac{\text{FractionEnhanced}}{\text{SpeedUpEnhanced}} \right)$$

$$\text{SpeedUp} = \frac{1}{(1 - \text{FractionEnhanced}) + \frac{\text{FractionEnhanced}}{\text{SpeedUpEnhanced}}}$$

- Remember, FractionEnhanced is the fraction of time that can be converted to use an enhancement, NOT the fraction of time that the enhanced portion takes after the enhancement.

63

Quantitative Principles of Computer Design

- Computer example:
  - Consider the enhancement to a processor for Web serving. New CPU is 10 x faster than original for Web serving. Original CPU busy with computation 40% of time and is waiting for I/O 60% of time. What is overall speedup gained by enhancement?
  - Only spends 40% doing work so limited by that amount.

64

Quantitative Principles of Computer Design

- **Reliability example**
  - The power supply of the disk subsystem in the last reliability example was improved from 200,000 hour to 830,000,000 hour MTTF (4150x better) by adding a redundant power supply. What is the reliability improvement by adding the redundant power supply?

Quantitative Principles of Computer Design

- Amdahl's Law express the low of diminishing returns
  - Incremental improvement in speedup gained by an improvement of just a portion of the computation diminishes as improvements are added.
- Corollary
  - If an enhancement is only usable for a fraction of a task, we can't speed up the task by more then the reciprocal of 1 minus that fraction.

Quantitative Principles of Computer Design

- ***CPU Performance***
  - Total CPU time for a task (CPU is a clock driven sequential circuit):

$$\text{CPU\_Time} = \text{ClockCycles} \times \text{ClockCycleTime} = \text{ClockCycles} \times \frac{1}{\text{ClockRate}}$$

  - Programs (tasks) are made of lists of instructions:

$$\text{ClockCyclesPerInstruction} = \text{CPI} = \frac{\text{ClockCycles}}{\text{InstructionCount}}$$

$$\text{CPU\_Time} = \text{InstructionCount} \times \text{CPI} \times \text{ClockCycleTime}$$

$$\text{CPU\_Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{ClockCycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{ClockCycle}}$$

Quantitative Principles of Computer Design

- ***CPU Performance***

$$\text{CPU\_Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{ClockCycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{ClockCycle}}$$

  - Significance:
    - First factor is a function of the ISA and of the compiler technology.
    - Second factor is a function of the organization and of the compiler.
    - Third factor is a function of the organization and of the technology.
  - <u>Faced with a giant tradeoff! The art of computer design is contained in this formula</u>.

Quantitative Principles of Computer Design

- ***How to improve a factor without affecting the others?***
- ***It is useful to breakdown this into more components, i.e. by classes of instructions.***

$$\text{ClockCycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPI} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{InstructionCount}} \times \text{CPI}_i$$

which breaks down the problem into the design of classes of instructions.

Quantitative Principles of Computer Design

- ***(e.g. Total CC = Number of FP operations occurring in program x number of clocks for a FP operation + number integer operations x clocks for integer operations …)***
- ***This also define the notion of instruction mix, which is the relative frequency of occurrence of classes of instructions (e.g. branches, FP, …), say in a given benchmark.***
- ***Ex. gcc***

Op	*Freq*	*CPIi*		*Term*
ALU	*50%*	*1*		*0.5*
Load	*20%*	*2*		*0.4*
Store	*10%*	*2*		*0.2*
Branch	*20%*	*2*		*0.4*
			CPI=	*1.5*