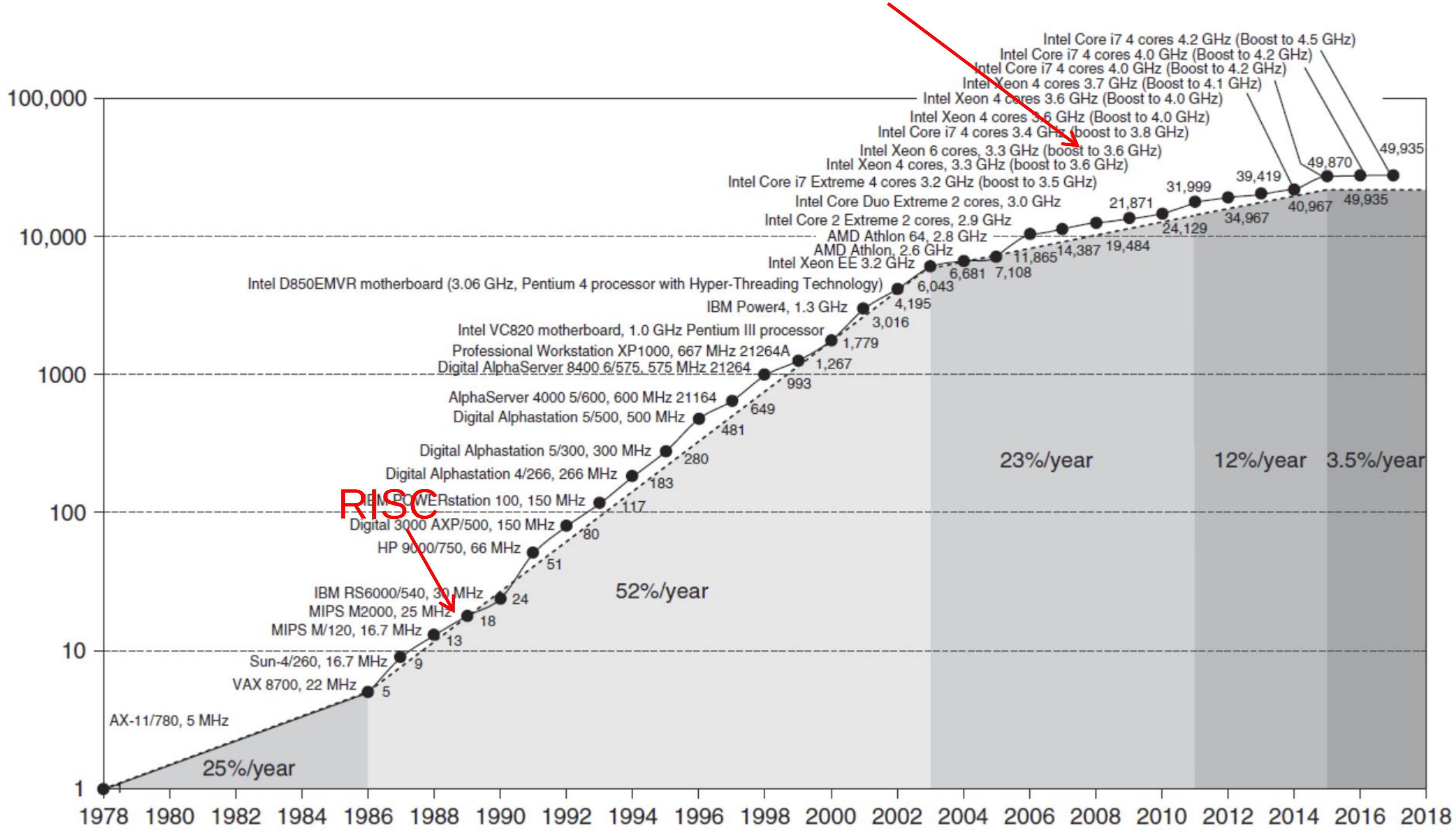
Chapter 1
Fundamentals of Quantitative Design and Analysis

Computer Technology

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by HLL compilers, UNIX
    - Lead to RISC architectures

- Together have enabled:
  - Lightweight computers
  - Productivity-based managed/interpreted programming languages

Single Processor Performance



Move to multi-processor

RISC

Fourfold Dramatic Growth

- Microprocessors outperformed the supercomputers
- Rapid improvement in cost-performance led to new classes of computers

Changing Face of Computing

- Mainframes in 1960
- Minicomputer in 1970
- Desktop computer in 1980
- Internet and the first PDA in 1990
- Embedded systems in 2000

Summary of Three Computing Classes

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/ warehouse-scale computer	Embedded
Price of system	$100-$1000	$300-$2500	$5000-$10,000,000	$100,000-$200,000,000	$10-$100,000
Price of microprocessor	$10-$100	$50-$500	$200-$2000	$50-$250	$0.01-$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy, proportionality	Price, energy, application-specific performance

Personal Mobile Device (PMD)

- A collection of wireless devices with multimedia user interface such as cell phones, tablet computers, and so on
- Cost is a prime concern
- Real-time performance is very critical
  - Soft real-time
  - Hard real-time

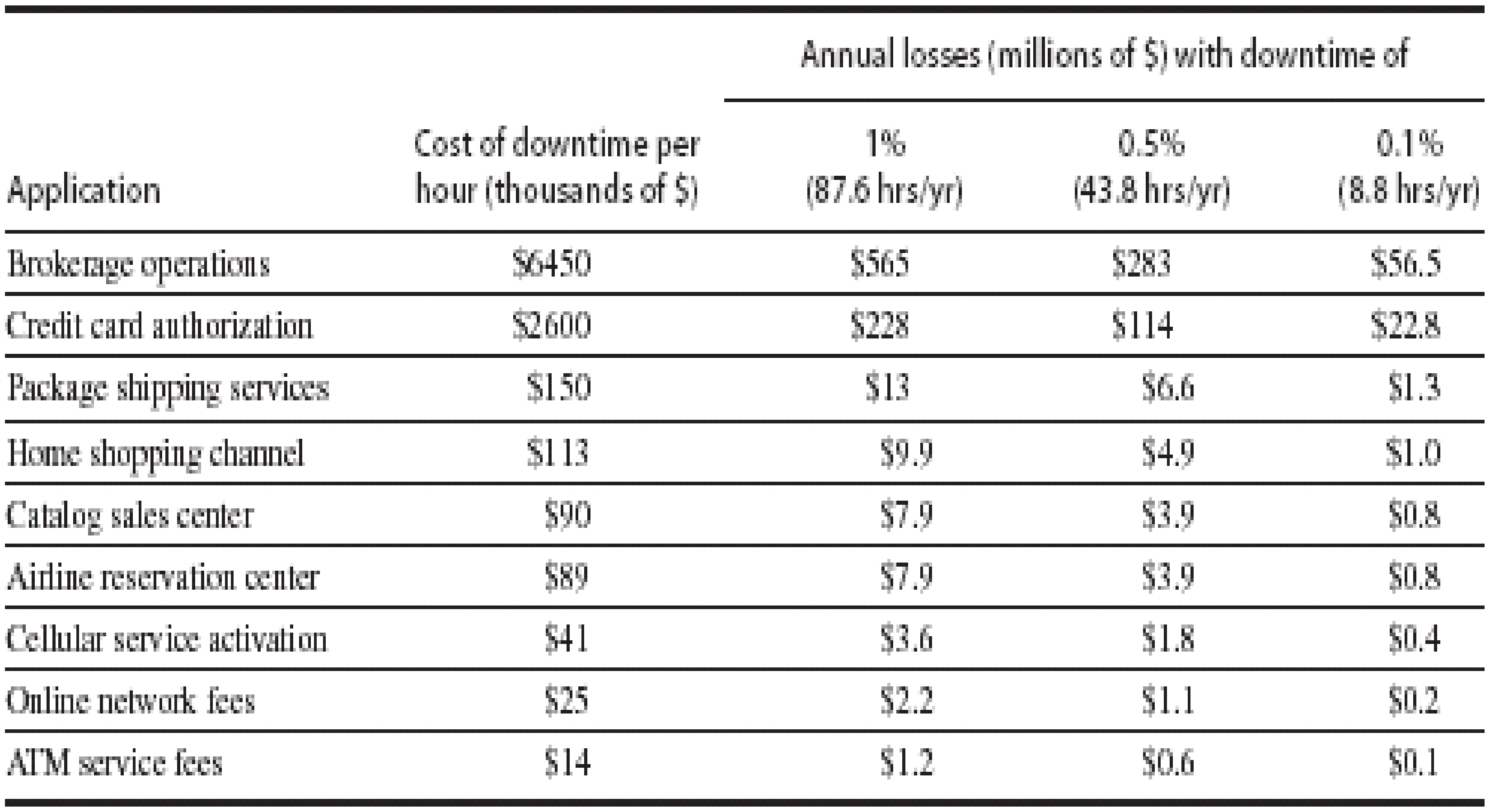
Desktop Computing

- The desktop market tends to be driven to optimize price-performance
- Desktop computing also tends to be reasonably well characterized in terms of applications and benchmarking, though
  - the increasing use of Web-centric
  - interactive applications poses new challenges in performance evaluation

Servers

- Different characteristics are important
    - Dependability
    - Scalability
    - Throughput
- A related category is *supercomputers*

Cost of Downtime



		Annual losses (millions of $) with downtime of		
Application	Cost of downtime per hour (thousands of $)	1% (87.6 hrs/yr)	0.5% (43.8 hrs/yr)	0.1% (8.8 hrs/yr)
Brokerage operations	$6450	$565	$283	$56.5
Credit card authorization	$2600	$228	$114	$22.8
Package shipping services	$150	$13	$6.6	$1.3
Home shopping channel	$113	$9.9	$4.9	$1.0
Catalog sales center	$90	$7.9	$3.9	$0.8
Airline reservation center	$89	$7.9	$3.9	$0.8
Cellular service activation	$41	$3.6	$1.8	$0.4
Online network fees	$25	$2.2	$1.1	$0.2
ATM service fees	$14	$1.2	$0.6	$0.1

Clusters/ Warehouse-Sacle Computers

- The growth of as a service (SaaS) for applications like searach, social networking, video sharing, multiplayer games, online shopping, and so on has led to the growth of clusters
- What are clusters?
- The largest clusters are called warehouse-scale computers (WSCs): tens of thousands of servers can act as one
- Price-performance and power are critical to WSCs
- WSCs are related to servers, in that availability is critical
- Supercomputers are related to WSCs in that they are equally expensive

Embedded Computers

- What is embedded system?
- Requirements of embedded systems
- Some characteristics:
  - Real-time constraint
  - Low power
  - Code density
- Processor core
- IP

Approaches for Embedded Systems

- System on a chip
- Custom software runs on off-the-shelf embedded processor
- Integrate with custom software and DSP with processor core

Class of Parallelism

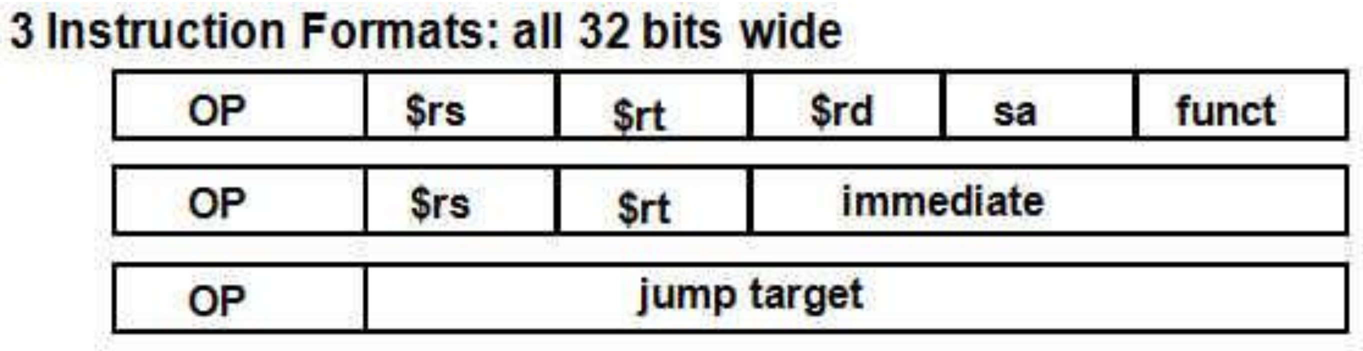
- Two kinds of parallelism in applications
  - Data-Level parallelism (DLP)
  - Task-Level parallelism (TLP)
- Computer hardware in turn can exploit these two kinds of application parallelism in four major ways
  - Instruction-Level parallelism
  - Vector Architectures and Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

Parallel Architectures

- At 40 years ago, Flynn proposed a model of categorizing all computers that is still useful today
  - —*Single instruction stream*, *single data stream (SISD)*
  - —*Single instruction stream*, *multiple data streams (SIMD)*
  - —*Multiple instruction streams*, *single data stream (MISD)*
  - —*Multiple instruction streams*, *multiple data streams (MIMD)*
- In fact, some multiprocessors are hybrids of these categories

Instruction Set Architecture (ISA)

- Class of ISA: RISC and CISC
- Memory addressing
- Addressing modes
- Types and sizes of operands
- Operations
- Control flow instructions
- Encoding on ISA



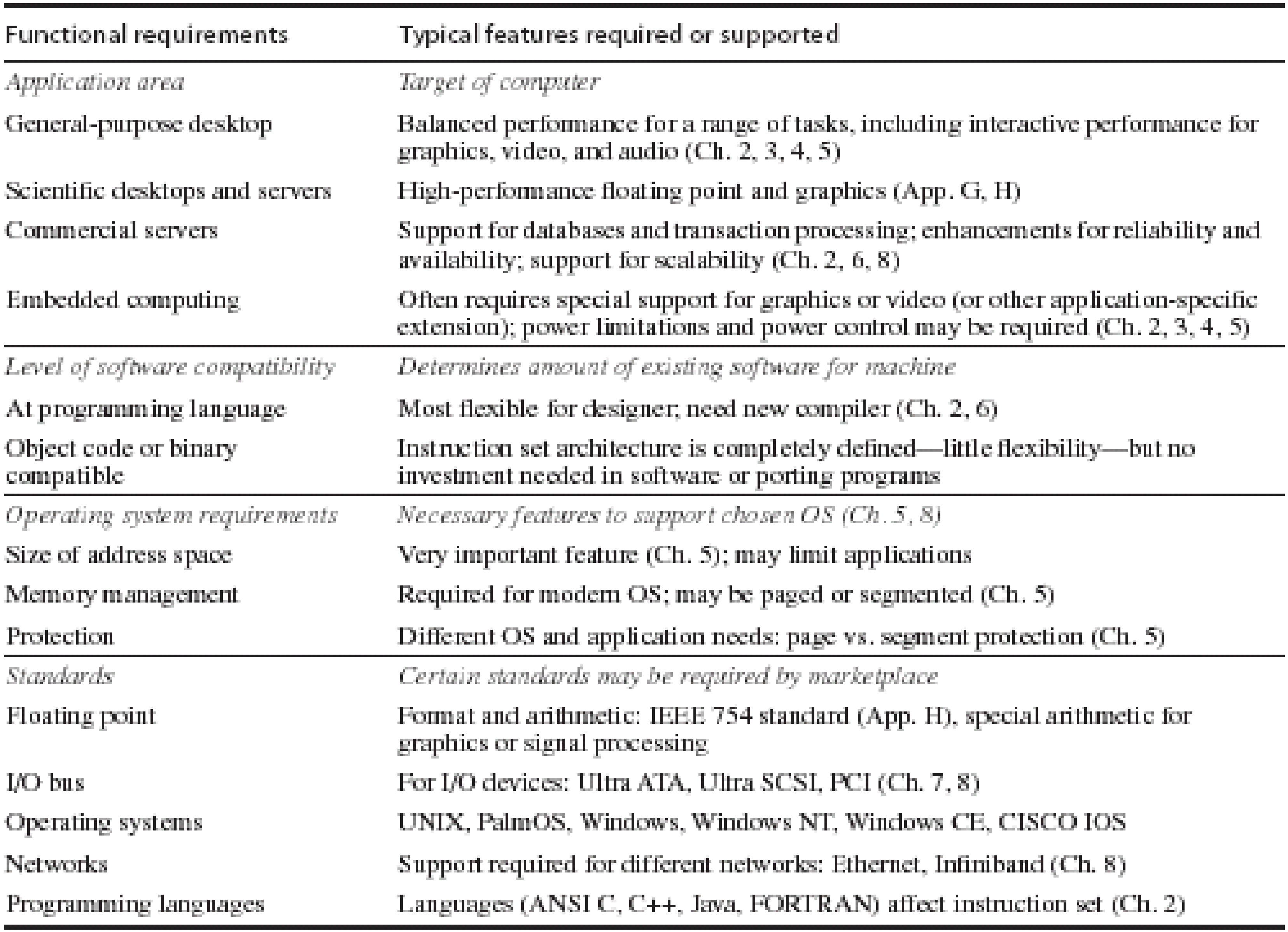
3 Instruction Formats: all 32 bits wide

OP	$rs	$rt	$rd	sa	funct
OP	$rs	$rt	immediate		
OP	jump target				

The Task of Computer Designer

- The task includes instruction set design, functional organization, logical design and implementation.
- Instruction set architecture refers to the actual programmer-visible instruction set.
- Organization includes the high-level aspects like the memory system, the bus architecture, and the design of internal CPU.
- Hardware refers to the specifics of a machine, including the detailed logic design and the packaging technology of the machine.
- Architecture consists of the above three.

Summary of Functional Requirements



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, 4, 5)
Scientific desktops and servers	High-performance floating point and graphics (App. G, H)
Commercial servers	Support for databases and transaction processing; enhancements for reliability and availability; support for scalability (Ch. 2, 6, 8)
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, 4, 5)
Level of software compatibility	*Determines amount of existing software for machine*
At programming language	Most flexible for designer; need new compiler (Ch. 2, 6)
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, 8)*
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 protection (Ch. 5)
Standards	*Certain standards may be required by marketplace*
Floating point	Format and arithmetic: IEEE 754 standard (App. H), special arithmetic for graphics or signal processing
I/O bus	For I/O devices: Ultra ATA, Ultra SCSI, PCI (Ch. 7, 8)
Operating systems	UNIX, PalmOS, Windows, Windows NT, Windows CE, CISCO IOS
Networks	Support required for different networks: Ethernet, Infiniband (Ch. 8)
Programming languages	Languages (ANSI C, C++, Java, FORTRAN) affect instruction set (Ch. 2)

Technology Trends

- If an instruction set architecture is to be successful, it must be designed to survive rapid changes in computer technology
- Four implementation technologies
  - —IC logic technology
  - —DRAM technology
  - —Magnetic disk technology
  - —Network technology
- Scaling of transistor performance, wires, and power in ICs

Trends in Technology

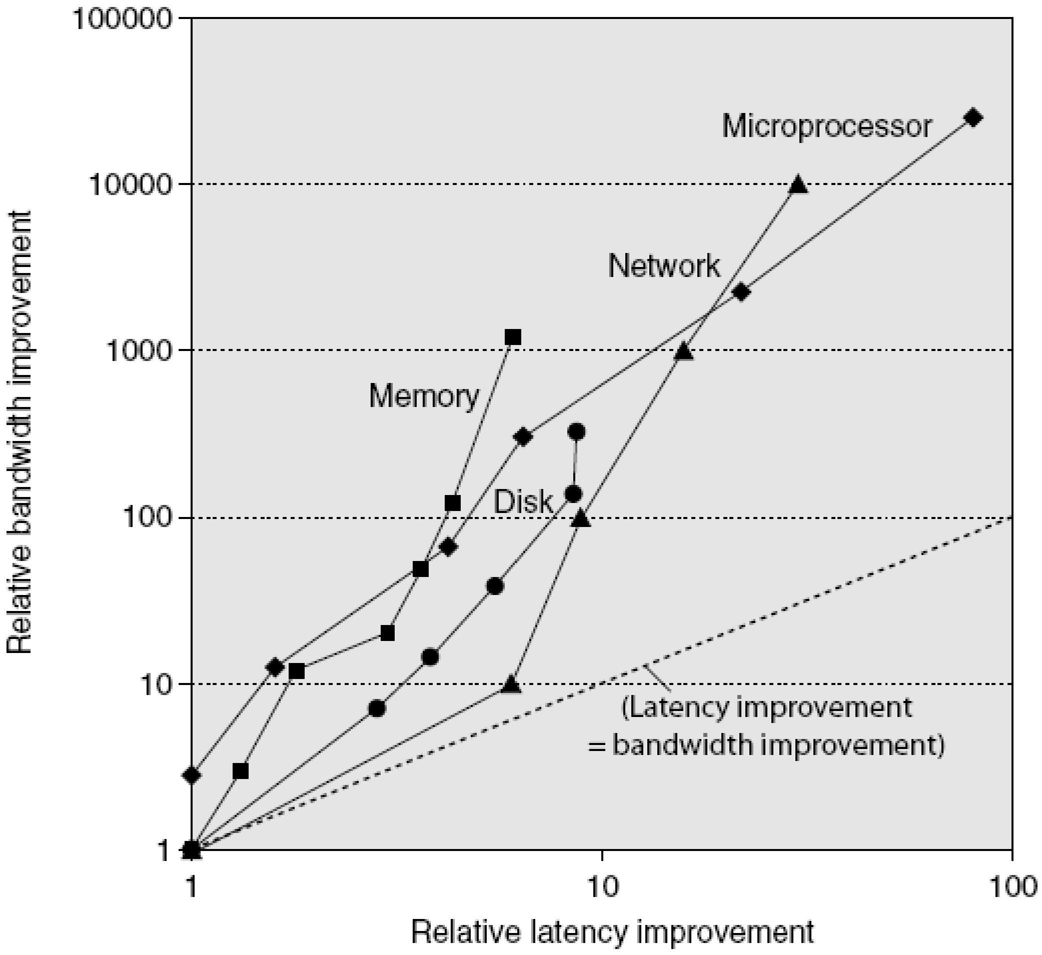
- Integrated circuit technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year
  - 15-25X cheaper/bit then Flash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

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

Trends in Power in IC

- Applications of 3C are getting popular
- Portable digital devices are widely used in the world.
- Power consumption has become one of important issues in embedded systems.

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

- Energy per task is often a better measurement

Dynamic Energy and Power

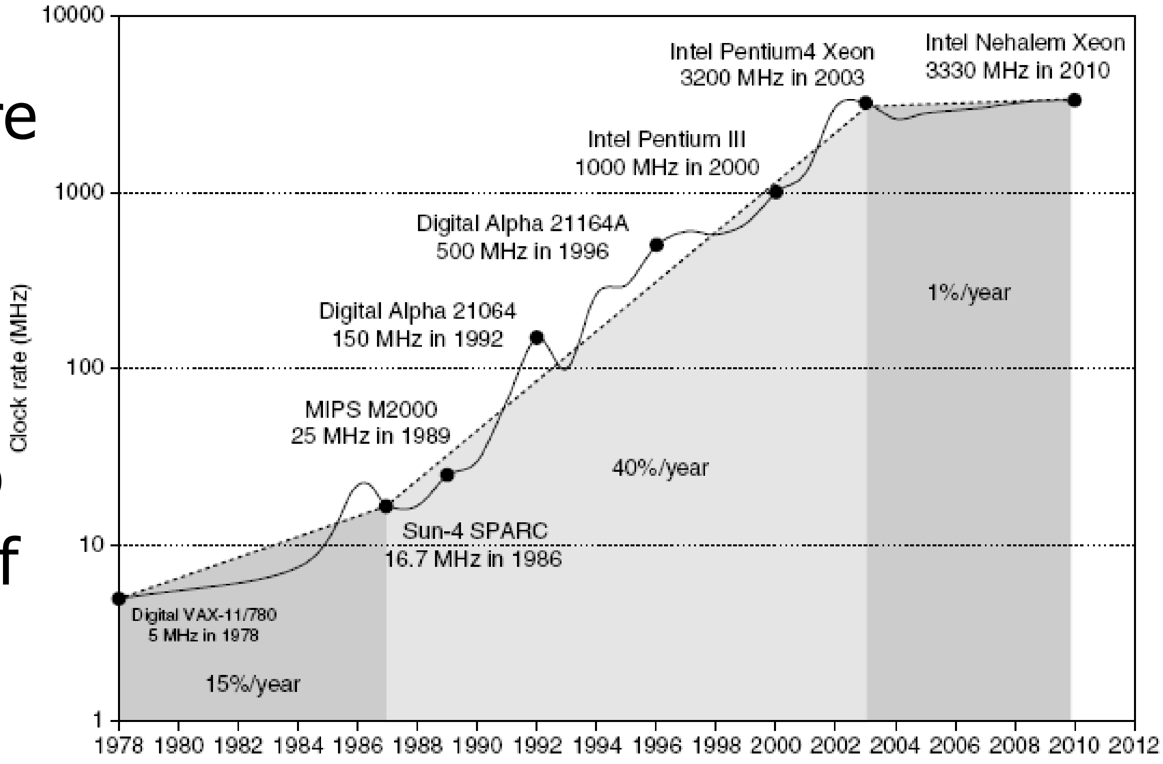
- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0
  - ½ x Capacitive load x Voltage^2

- Dynamic power
  - ½ x Capacitive load x Voltage^2 x Frequency switched

- Reducing clock rate reduces power, not energy

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



Reducing Power

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

Static Power

- Static power consumption
  - $\text{Current}_{\text{static}}$ x Voltage
  - Scales with number of transistors
  - To reduce: power gating

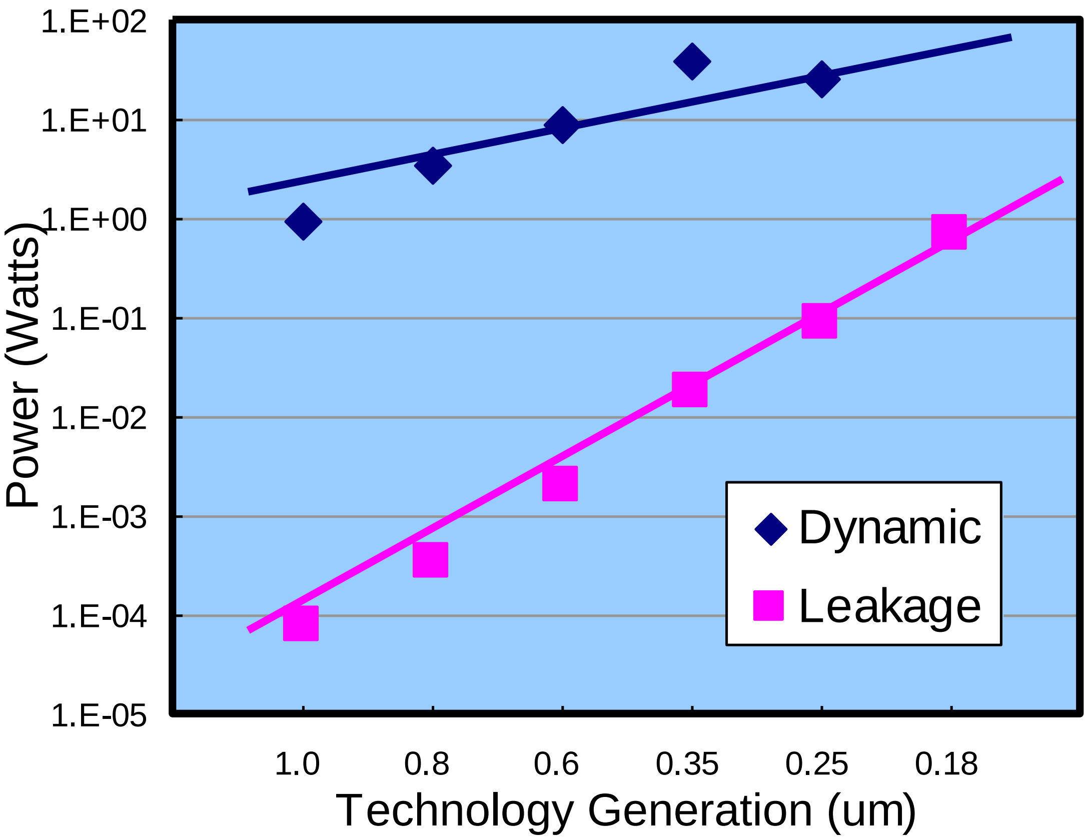
Power Consumption

- $\text{Power} = \text{power}_{\text{dynamic}} + \text{power}_{\text{static}}$
- $P_d = \alpha \cdot f_{\text{clk}} \cdot C_L \cdot V_{DD}^2$
    - Reduce weight
    - Package cost
    - Cooling
    - Relaiabilty
- $\text{Energy}_{\text{dynamic}} = C_L \cdot V_{DD}^2$

Frying an egg on the CPU in 11 minutes

Power Consumption (Cont'd)

- $P_s = I_s \cdot V_{DD}$

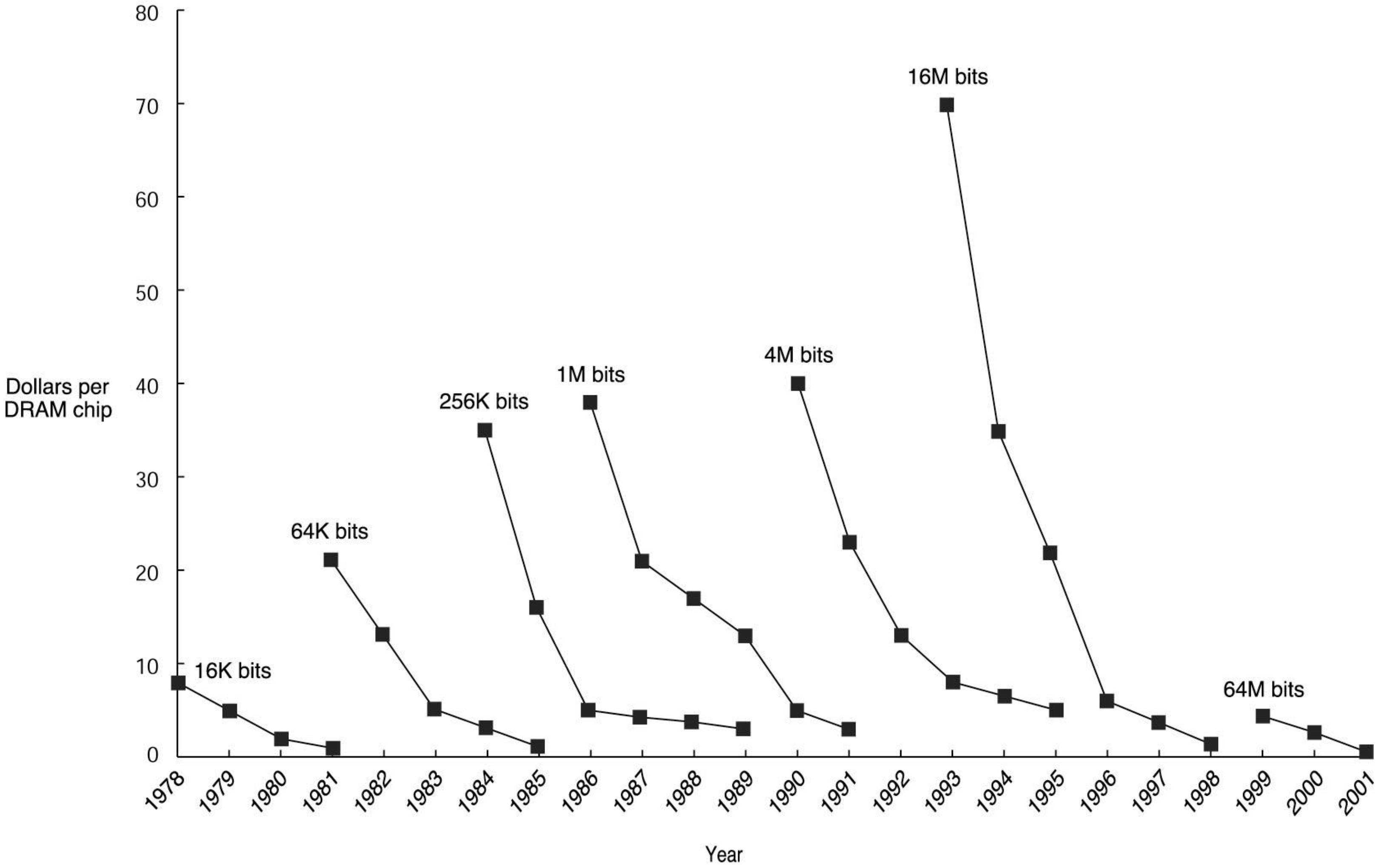


source: intel

Cost, Price, and Their Trends

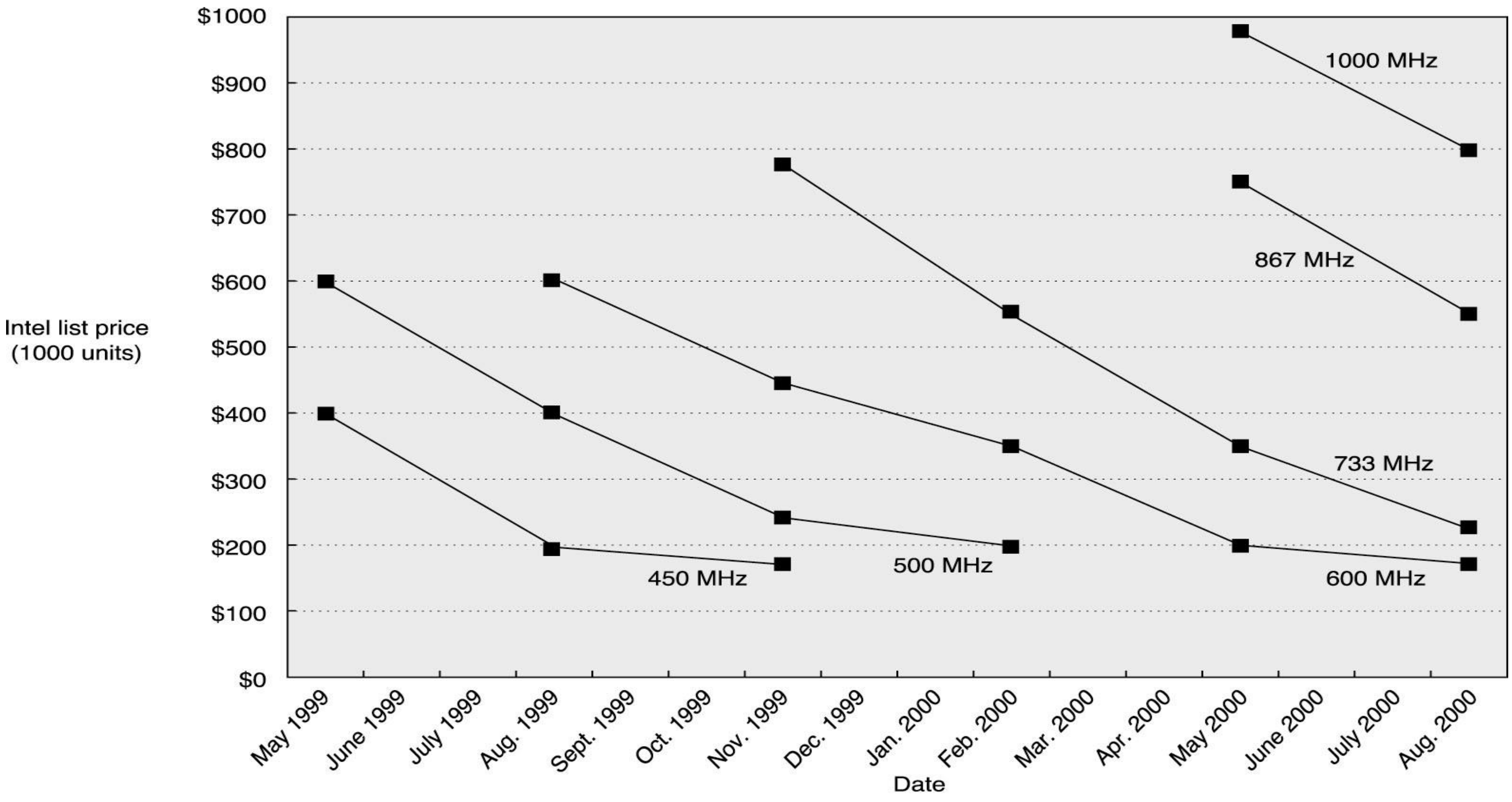
- Cost
- Price
- Key factors
  - Yield
  - Volume
- Commodities

Example 1



Dollars per DRAM chip
80
70
60
50
40
30
20
10
0
16K bits
64K bits
256K bits
1M bits
4M bits
16M bits
64M bits
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
Year

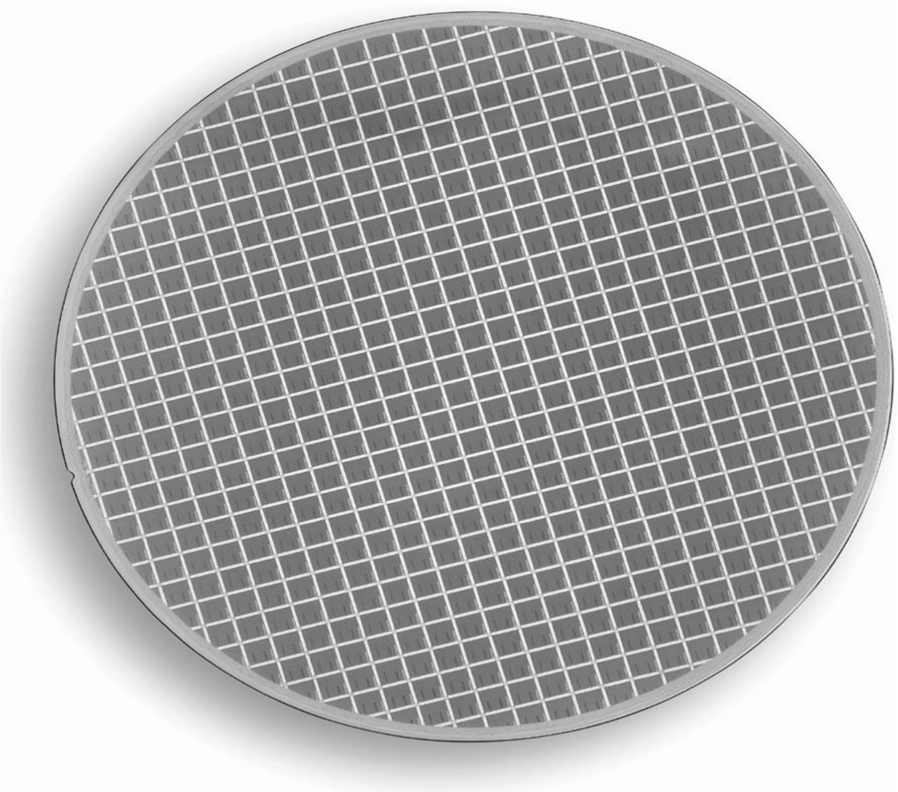
Example 2



$1000
$900
$800
$700
$600
$500
$400
$300
$200
$100
$0
Intel list price
(1000 units)
1000 MHz
867 MHz
733 MHz
600 MHz
500 MHz
450 MHz
May 1999
June 1999
July 1999
Aug. 1999
Sept. 1999
Oct. 1999
Nov. 1999
Dec. 1999
Jan. 2000
Feb. 2000
Mar. 2000
Apr. 2000
May 2000
June 2000
July 2000
Aug. 2000
Date

Wafer and Die

- Wafer
- Die



Integrated Circuit Cost

- Integrated circuit

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

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

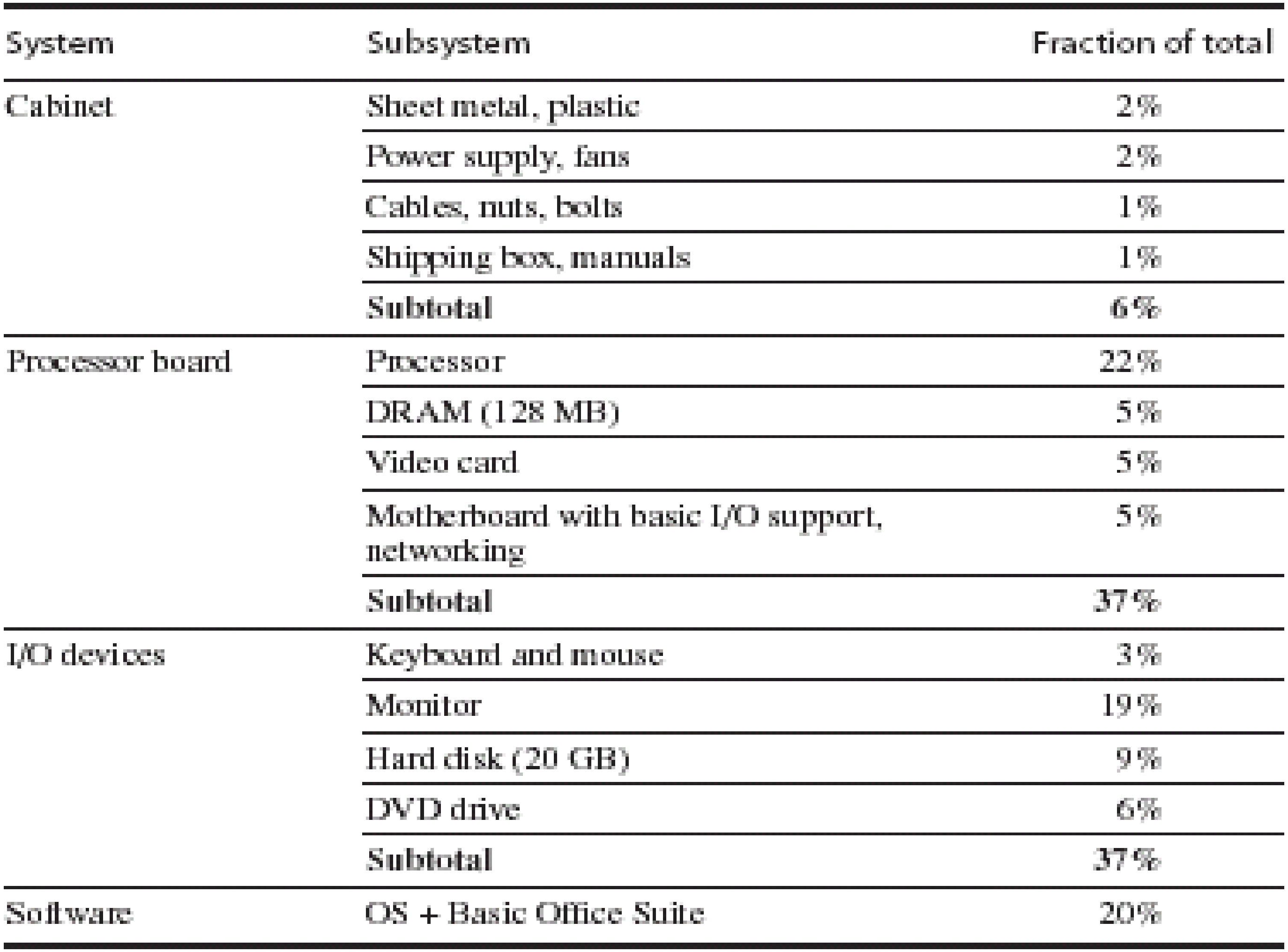
$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- 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)

Distribution of Cost in a System

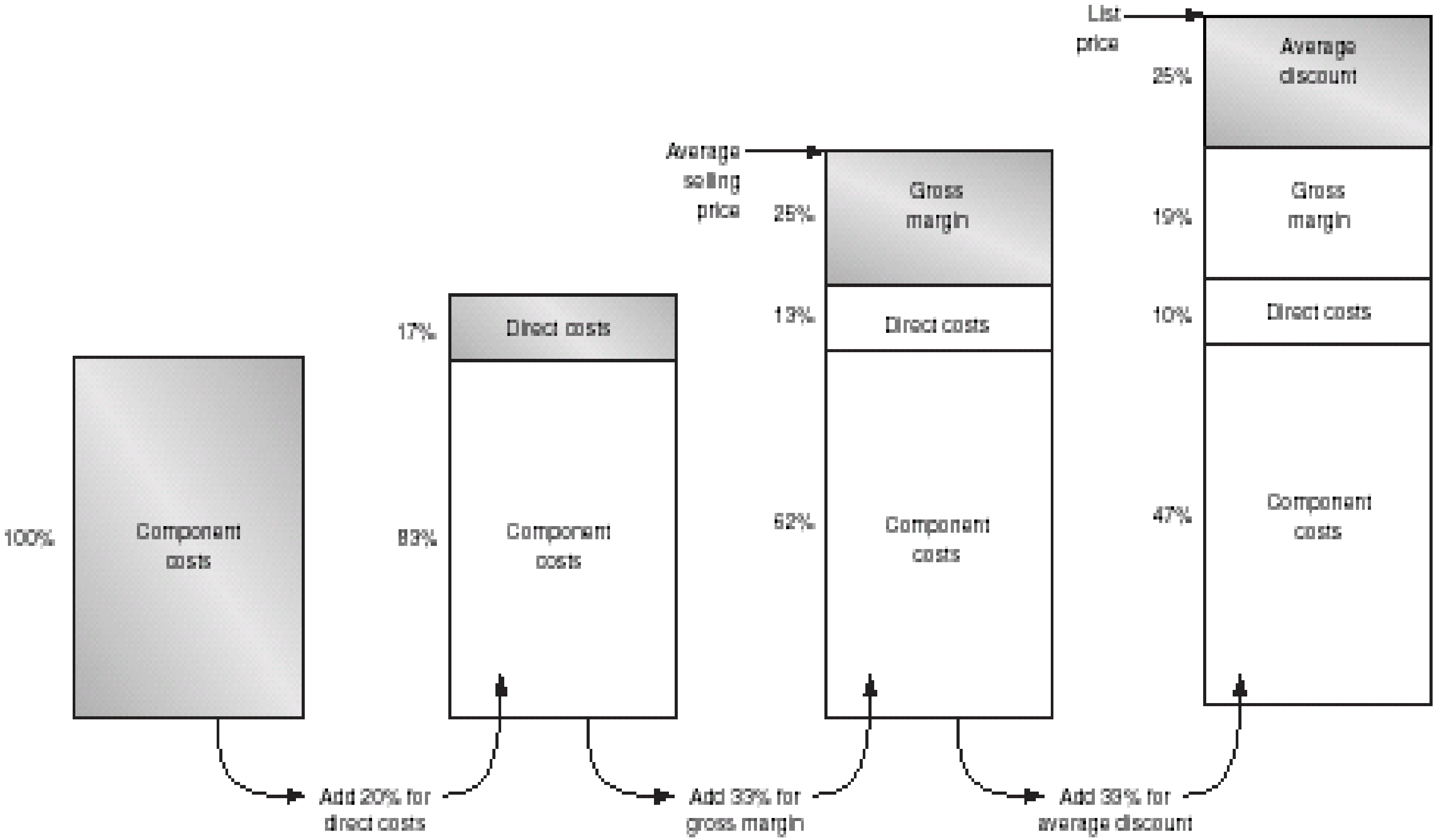


System	Subsystem	Fraction of total
Cabinet	Sheet metal, plastic	2%
	Power supply, fans	2%
	Cables, nuts, bolts	1%
	Shipping box, manuals	1%
	Subtotal	**6%**
Processor board	Processor	22%
	DRAM (128 MB)	5%
	Video card	5%
	Motherboard with basic I/O support, networking	5%
	Subtotal	**37%**
I/O devices	Keyboard and mouse	3%
	Monitor	19%
	Hard disk (20 GB)	9%
	DVD drive	6%
	Subtotal	**37%**
Software	OS + Basic Office Suite	20%

Cost versus Price

- Understand the relationship of cost to price and the impact on price of adding, deleting, or replacing components
- Direct costs: the costs directly related to making a product
- Gross margin
- Average selling price = component cost + direct cost + gross margin
- List price

The Components of Price for a $1000 PC



Some Issues

- Pricing is sensitive to competition
- How much should a company invest in R&D?
- Expensive machines usually cost more

Goals of Design

- Cost vs cost-performance
- High-performance design
  - e.g. supercomputers
- Low cost performance
  - e.g. embeded products
- Cost-performance design
  - e.g. PC, workstations, and servers

Dependability

- One difficult question is deciding when a system is operating properly
- Infrastructure providers started offering *service level agreements* (SLAs) or *service level objectives* (SLOs)
- Systems alternate between *Service accomplishment* and *Service interruption* with respect to SLA
  - Transitions between these two states are caused by failures
- Two main measures of dependability
  - Module reliability
  - Module availability

Define and quantity dependability (1/3)

- 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)*

Measuring and Reporting Performance

- Response time: the time between the start and the completion of an event
- Execution time
- Throughput
- “X is n times faster than Y” means

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\dfrac{1}{\text{Performance}_Y}}{\dfrac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

Measuring Performance

- Wall-clock time
- Response time
- Elapsed time
- CPU time
  - User CPU time
  - System CPU time
- System performance

Choosing Programs to Evaluate Performance

- Real applications
- Modified (or scripted) applications
- Kernels
- Toy benchmarks
- Synthetic benchmarks

Real Applications

- Buyers only know that some users will run them to solve real problems
- Examples: C compilers, text-processing software like Word, and other applications like Photoshop
- Real applications have I/O, and options that a user can select when running the pro
- Downside trends: Real APs often encounter portability problems arising from dependences on the OS or compiler
- Enhancing portability often means modifying the source and sometimes eliminating some important activity, such as interactive graphics, which tends to be more system dependent

Modified Applications

- Real APs are used as the building blocks for a benchmark, either with modifications to the AP or with a script that acts as stimulus to the application
- Reasons to modify real Aps:
  - To enhance portability or
  - To focus on one particular aspect of system performance
- E.g., to remove I/O to create a CPU-oriented benchmark
- Scripts are used to simulate
  - application programs to reproduce interactive behavior occurring on a desktop system
  - complex multiuser interaction occurring in server system

Kernels

- Attempts have been made to extract small, key pieces from real programs and use them to evaluate performance
- Kernels are best used to isolate performance of individual features of a machine to explain the reasons for differences in performance of real programs
- Two best known examples:
  - Livermore Loops
  - Linpack
- Unlike real programs, no user would run kernel programs; they exist solely to evaluate performance

Toy Benchmarks

- Toy benchmarks are typically between 10 and 100 lines of code
- Toy benchmarks produce a result the user already knows before running the toy program
- They are small, easy to type, and run on almost any computer
- Examples are
  - Sieve of Eratosthenes
  - Puzzle
  - Quicksort

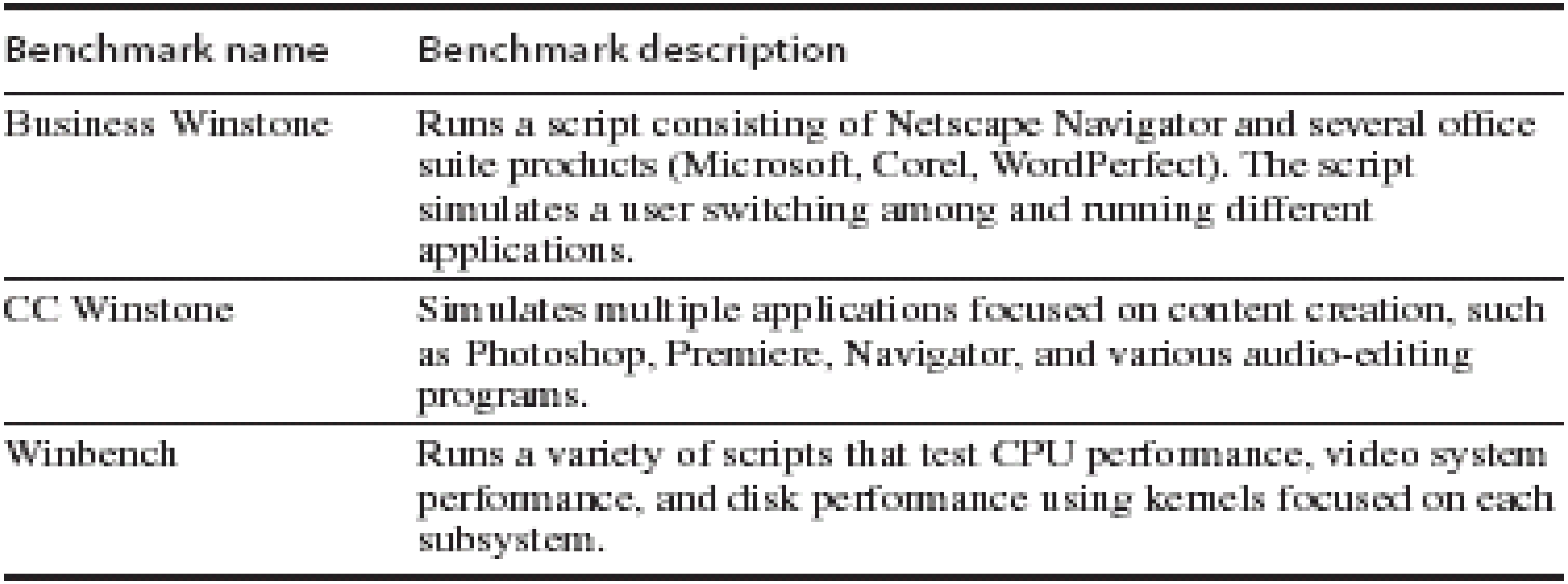
Synthetic Benchmarks

- Similar in philosophy to kernels, synthetic benchmarks try to match the average frequency of operations and operands of a large set of programs
- Two most popular synthetic benchmarks
  - Whetstone
  - Dhrystone
- Synthetic code is created artificially to match an average execution profile
- Synthetic benchmarks are not even pieces of real programs, although kernels might be

Benchmark Suites

- Good reputation is the most important
- It has become popular to put together collections of benchmarks to measure the performance of processors with a variety of applications
- A key advantage is to lessen the weakness of any one benchmark
- SPEC benchmarks cover different application classes
- A large set of benchmarks developed for PCs running the Windows OS, covering a variety of different application environments

Some PC Benchmarks



Benchmark name	Benchmark description
Business Winstone	Runs a script consisting of Netscape Navigator and several office suite products (Microsoft, Corel, WordPerfect). The script simulates a user switching among and running different applications.
CC Winstone	Simulates multiple applications focused on content creation, such as Photoshop, Premiere, Navigator, and various audio-editing programs.
Winbench	Runs a variety of scripts that test CPU performance, video system performance, and disk performance using kernels focused on each subsystem.

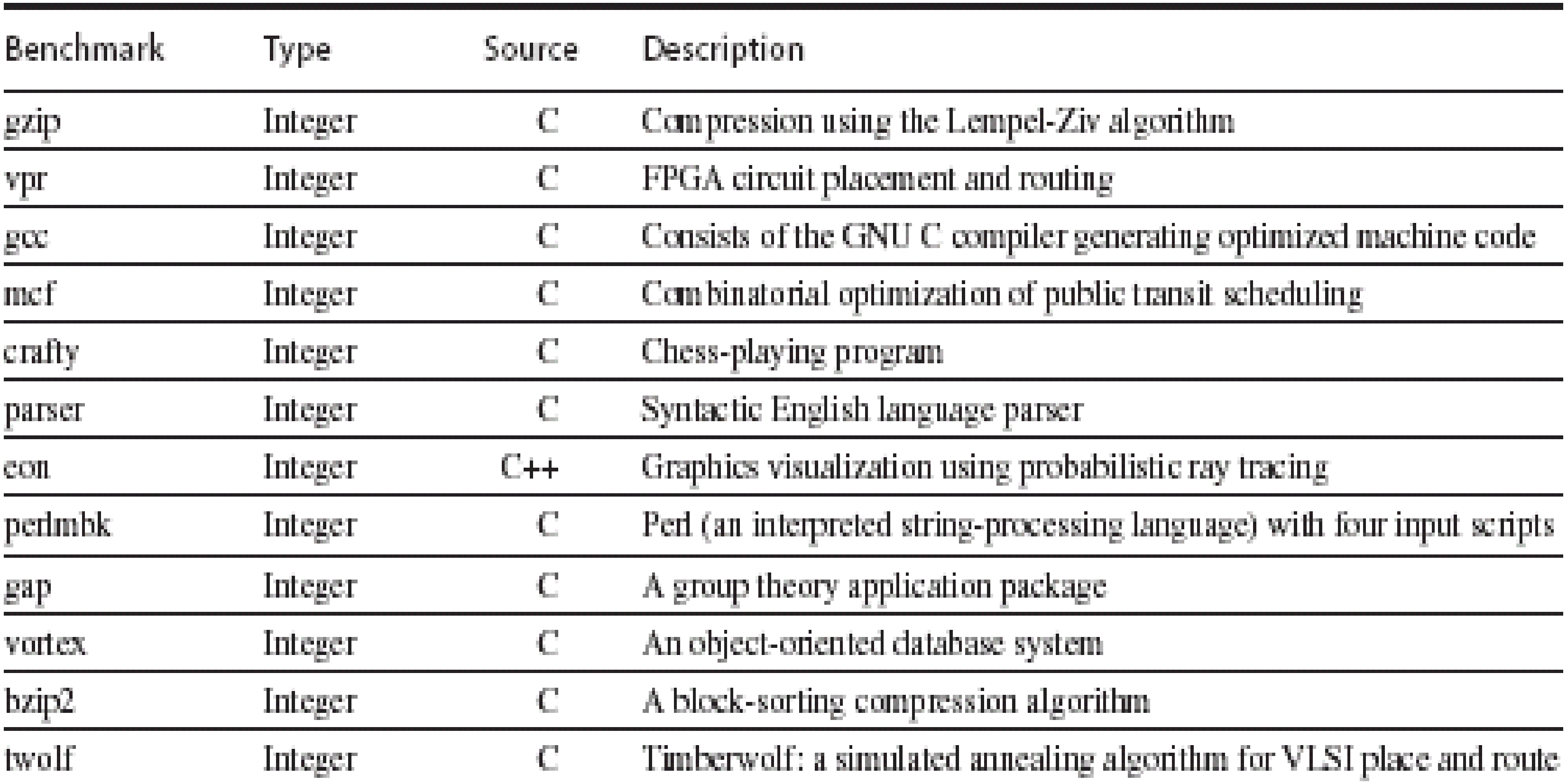
SPEC Benchmark

- SPEC is originally for CPU performance
- SPEC benchmarks are real programs, modified for portability and to minimize the role of I/O in overall benchmark performance
- SPEC CPU2000 is aimed at CPU performance

Two SPEC Graphics Benchmarks

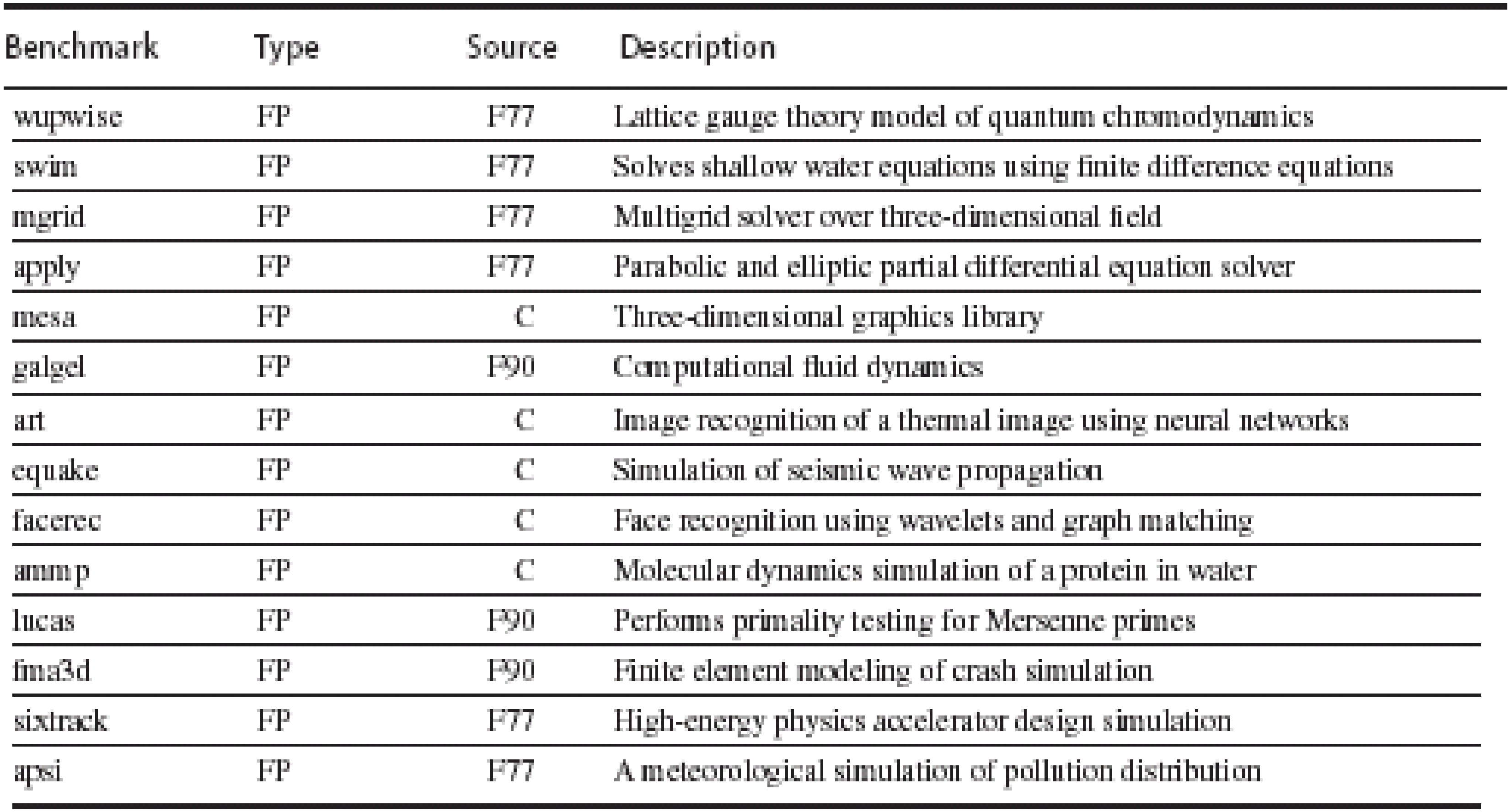
- SPECviewperf
  - measures the 3D rendering performance of systems running under OpenGL using
    - 3D model
    - a series of OpenGL calls transforming 3D model
- SPECapc consists of runs of several large applications, including
  - Pro/Engineer: A solid modeling application that does extensive 3D rendering
  - SolidWorks 2001: A 3D CAD/CAM design tool running a series of five tests varying from I/O intensive to CPU intensive
  - Unigraphics V15: Based on an aircraft model and covering a wide spectrum of Unigraphics functionality, including assembly, drafting, numeric control machining, solid modeling, and optimization

CINT2000



Benchmark	Type	Source	Description
gzip	Integer	C	Compression using the Lempel-Ziv algorithm
vpr	Integer	C	FPGA circuit placement and routing
gcc	Integer	C	Consists of the GNU C compiler generating optimized machine code
mcf	Integer	C	Combinatorial optimization of public transit scheduling
crafty	Integer	C	Chess-playing program
parser	Integer	C	Syntactic English language parser
eon	Integer	C++	Graphics visualization using probabilistic ray tracing
perlmbk	Integer	C	Perl (an interpreted string-processing language) with four input scripts
gap	Integer	C	A group theory application package
vortex	Integer	C	An object-oriented database system
bzip2	Integer	C	A block-sorting compression algorithm
twolf	Integer	C	Timberwolf: a simulated annealing algorithm for VLSI place and route

CFP2000



Benchmark	Type	Source	Description
wupwise	FP	F77	Lattice gauge theory model of quantum chromodynamics
swim	FP	F77	Solves shallow water equations using finite difference equations
mgrid	FP	F77	Multigrid solver over three-dimensional field
apply	FP	F77	Parabolic and elliptic partial differential equation solver
mesa	FP	C	Three-dimensional graphics library
galgel	FP	F90	Computational fluid dynamics
art	FP	C	Image recognition of a thermal image using neural networks
equake	FP	C	Simulation of seismic wave propagation
facerec	FP	C	Face recognition using wavelets and graph matching
ammp	FP	C	Molecular dynamics simulation of a protein in water
lucas	FP	F90	Performs primality testing for Mersenne primes
fma3d	FP	F90	Finite element modeling of crash simulation
sixtrack	FP	F77	High-energy physics accelerator design simulation
apsi	FP	F77	A meteorological simulation of pollution distribution

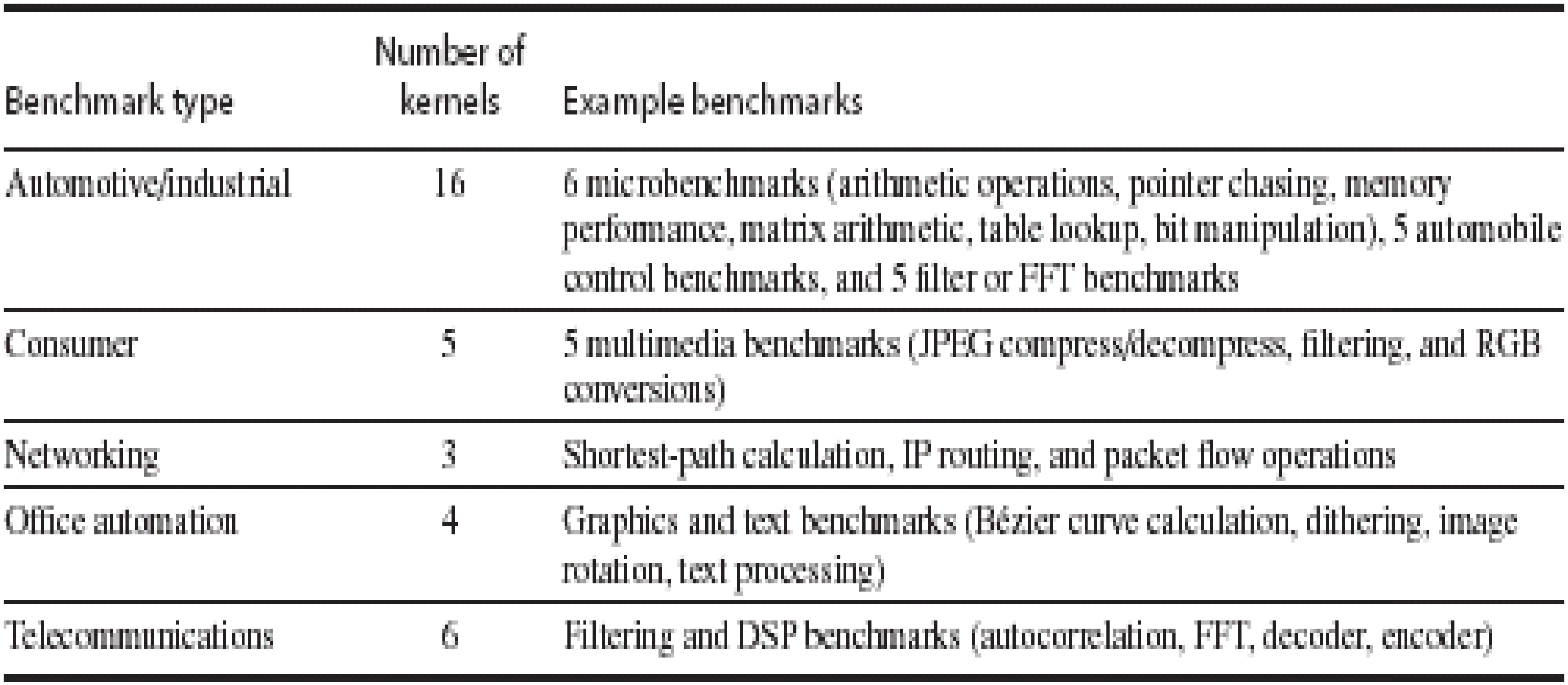
Server Benchmarks

- SPECrate: run SPEC CPU and convert the CPU tome into a rate
- SPECSFS for file servers
- SPECWeb for web servers
- TP measures the system ability to handle transactions
- TCP measures performance in transaction per second
  - TPC-A
  - TPC-C
  - TPC-H
  - TPC-R
  - TPC-W

Embedded Benchmarks

- Benchmarks for embedded computing systems
- They are far more nascent
- Difficulties:
  - Enormous variety
  - Different requirements
- EEMBC
- Assessments for both small kernel and the entire application are important

EEMBC Benchmark Suite



Benchmark type	Number of kernels	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions)
Networking	3	Shortest-path calculation, IP routing, and packet flow operations
Office automation	4	Graphics and text benchmarks (Bézier curve calculation, dithering, image rotation, text processing)
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, encoder)

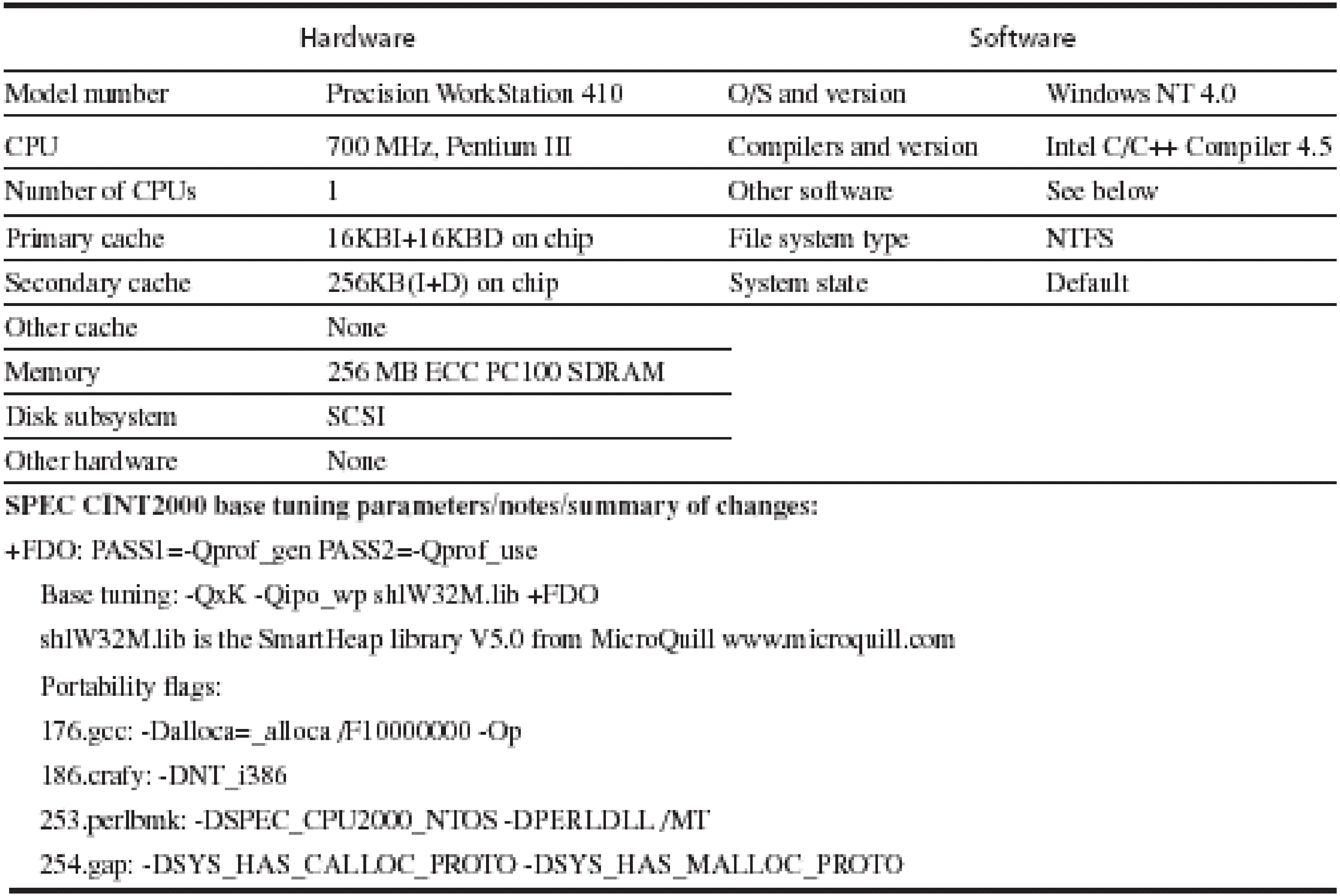
Reporting Performance Reports

- A SPEC benchmark report requires
  - A fairly complete description of the machine
  - Compiler flags
  - Publication of both baseline and optimization results
- Software configurations can greatly affect the result for a benchmark
- It is important to describe exactly the system being made and the nonstandard parts

Reporting Performance Reports (cont'd)

- Another key question: source code modifications or hand-generated assembly language are allowed
  - No source code modifications are allowed
  - Source code modifications are allowed, but are essentially difficult or impossible
  - Source code modifications are allowed
  - Hand-coding is allowed
- Key issue: what modifications really reflect
  - Real practice
  - Useful insights to users
  - Reduce the accuracy of the benchmarks

CINT2000 Report



Hardware		Software	
Model number	Precision WorkStation 410	O/S and version	Windows NT 4.0
CPU	700 MHz, Pentium III	Compilers and version	Intel C/C++ Compiler 4.5
Number of CPUs	1	Other software	See below
Primary cache	16KBI+16KBD on chip	File system type	NTFS
Secondary cache	256KB(I+D) on chip	System state	Default
Other cache	None		
Memory	256 MB ECC PC100 SDRAM		
Disk subsystem	SCSI		
Other hardware	None		

SPEC CINT2000 base tuning parameters/notes/summary of changes:

+FDO: PASS1=-Qprof_gen PASS2=-Qprof_use

Base tuning: -QxK -Qipo_wp shlW32M.lib +FDO

shlW32M.lib is the SmartHeap library V5.0 from MicroQuill www.microquill.com

Portability flags:

176.gcc: -Dalloca=_alloca /F10000000 -Op

186.crafy: -DNT_i386

253.perlbmk: -DSPEC_CPU2000_NTOS -DPERLDLL /MT

254.gap: -DSYS_HAS_CALLOC_PROTO -DSYS_HAS_MALLOC_PROTO

Comparing and Summarizing Performance

A is 10 times faster than B for program P1.

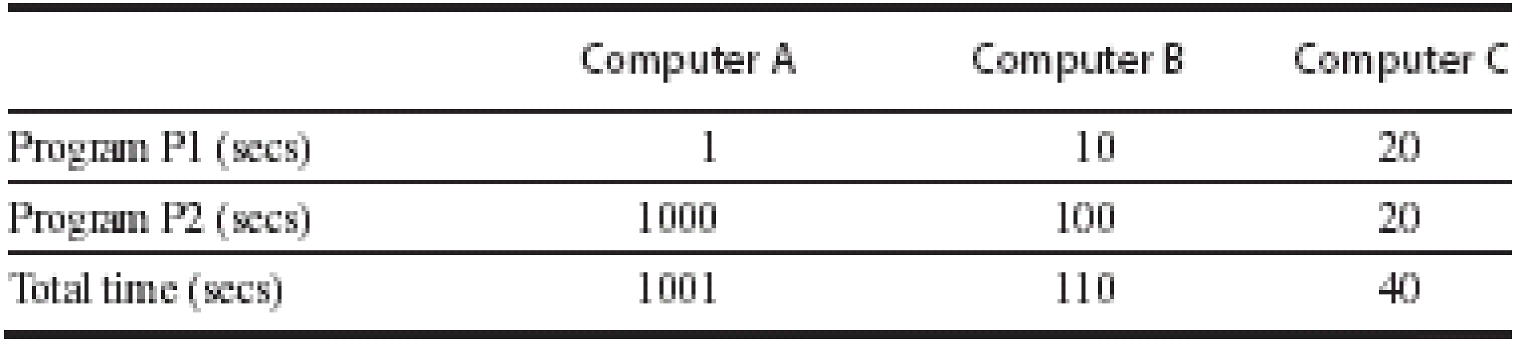
B is 10 times faster than A for program P2.

A is 20 times faster than C for program P1.

C is 50 times faster than A for program P2.

B is 2 times faster than C for program P1.

C is 5 times faster than B for program P2.



	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40

Total Execution Time: A Consistent Summary Measure

- B is 9.1 times faster than A for programs P1 and P2.
- C is 25 times faster than A for programs P1 and P2.
- C is 2.75 times faster than B for programs P1 and P2.

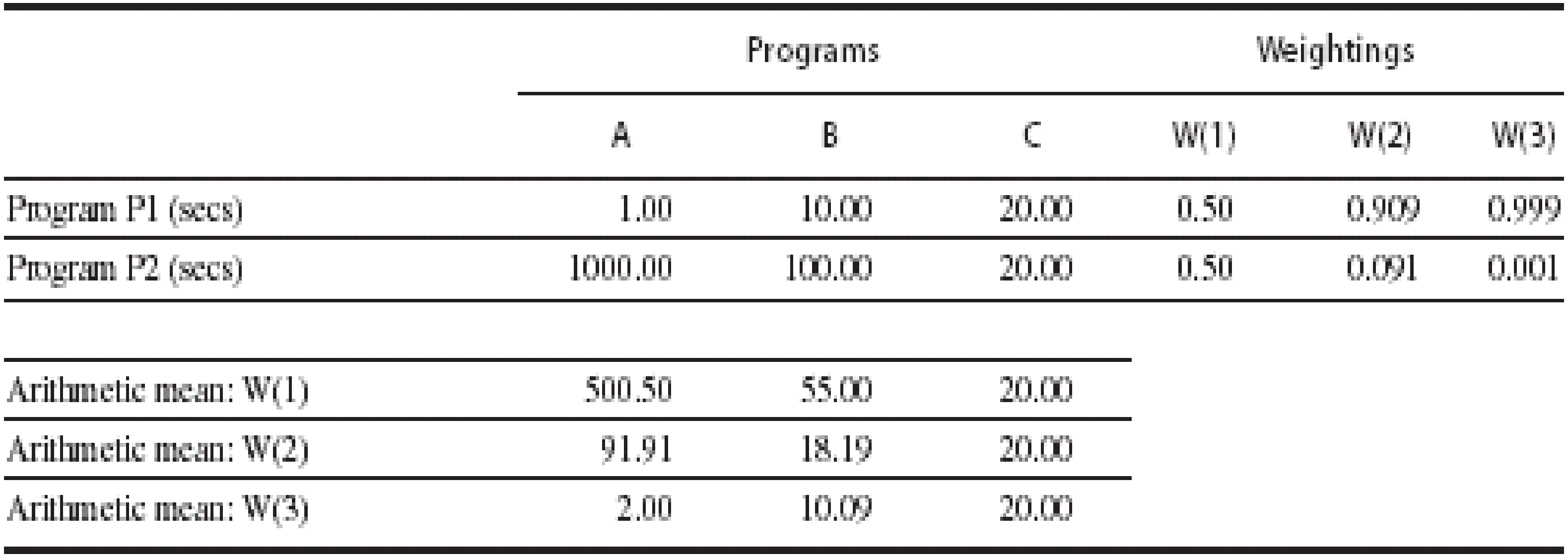
- An average of the execution times that tracks total execution time is the arithmetic mean:

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

Weighted Execution Time

- The formula is

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



	Programs			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1 (secs)	1.00	10.00	20.00	0.50	0.909	0.999
Program P2 (secs)	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 Execution Time of Geometric Means

- The formula is

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

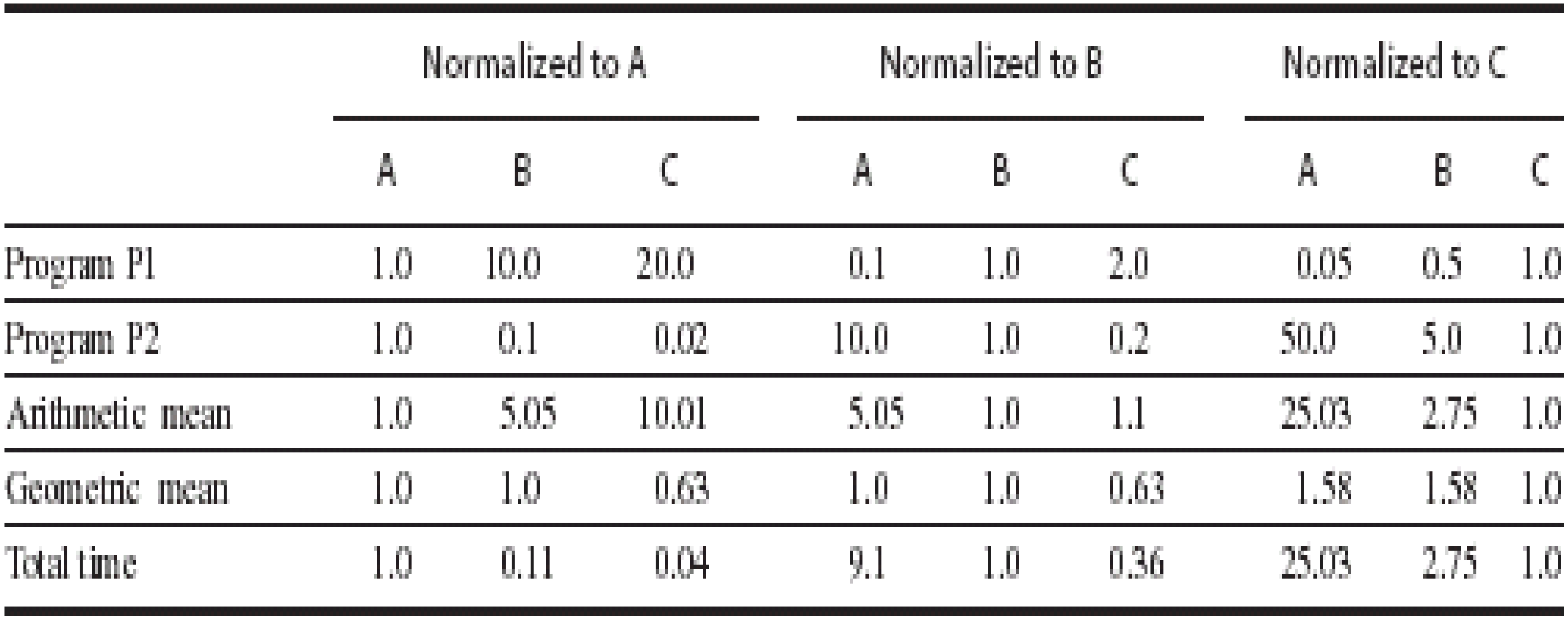
- A nice property of geometric means

$$\frac{\text{Geometric mean}(X_i)}{\text{Geometric mean}(Y_i)} = \text{Geometric mean}\left(\frac{X_i}{Y_i}\right)$$

Pros and Cons of Geometric Means

- Pros
  - —Taking either the ratio of the means or the mean of the ratios yields the same result
  - —It is independent of the running times of the individual programs, and it doesn't matter which machine is used to normalize
  - —The geometric mean would be less misleading than the arithmetic mean
- Cons
  - —The geometric mean do not predict execution time
  - —It encourages designers to focus their attention on the sweet spots

Execution Times Measured by Geometric Means



	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arithmetic mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

Quantitative Principles of Computer Design

- Make the common case fast
- Amdahl’s Law
- The CPU performance equation
- Principle of locality
- Take advantage of parallelism

Principle of Locality

- Other important fundamental observations come from properties of programs but Amdahl’s Law
- Programs tend to reuse data and instructions they have used recently
- A rule of thumb is: a program spends 90% of its execution time in only 10% of the code
- Principle of locality also applies to data accesses

Two Different Types of Locality

- Temporal locality
  - —Recently accessed items are likely to be accessed in the near future
- Spatial locality
  - —Items whose addresses are near one another tend to be referenced close together in time

Take Advantage of Parallelism

- It is one of the most important methods for improving performance
- One of the simplest ways to do this is through pipelining
- Parallelism can also be exploited at the level of detailed digital design, e.g. set-associative caches
- Modern ALUs use carry-lookahead, which uses parallelism to speed the process of computing sums from linear to logarithmic in the number of bits per operand

Amdahl’s Law

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used
- What is speedup?

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Amdahl’s Law (Cont’d)

- Amdahl’s Law can serve as a guide to
  - How much an enhancement will improve performance
  - How to distribute resources to improve cost-performance
- It is particularly useful for comparing
  - The overall system performance of two alternatives
  - Two CPU design alternatives

Key Factors of Speedup

- The fraction of the computation time converted to take advantage of the enhancement
  - Fraction$_{\text{enhanced}}$: if 20 seconds is enhanced in a program that takes 60 seconds in total, the fraction is 20/60
- How much faster the task would run if the enhanced mode were used for the entire program
  - Speedup$_{\text{enhanced}}$: If the enhanced mode takes 2 seconds for some portion of the program that can completely use the mode, while the original mode took 5 seconds for the same portion, the improvement is 5/2

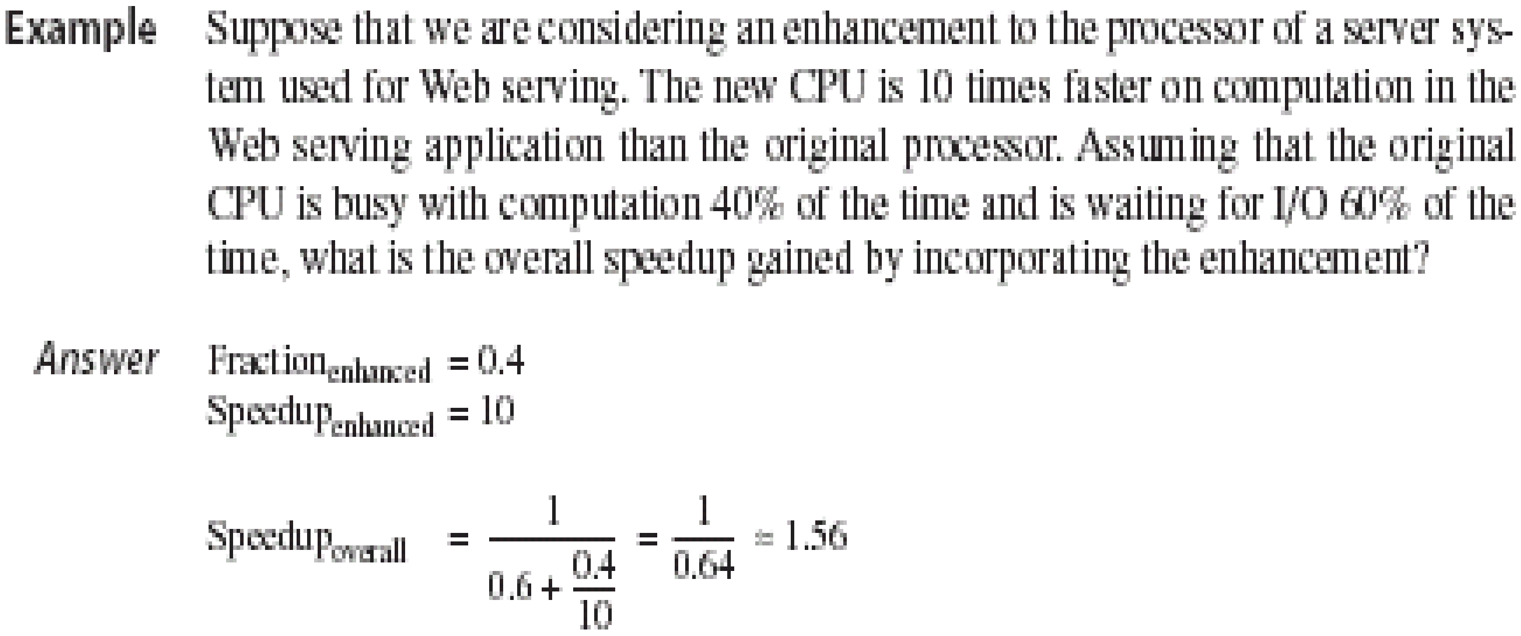
Speedup

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example 1



Example Suppose that we are considering an enhancement to the processor of a server system used for Web serving. The new CPU is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original CPU is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

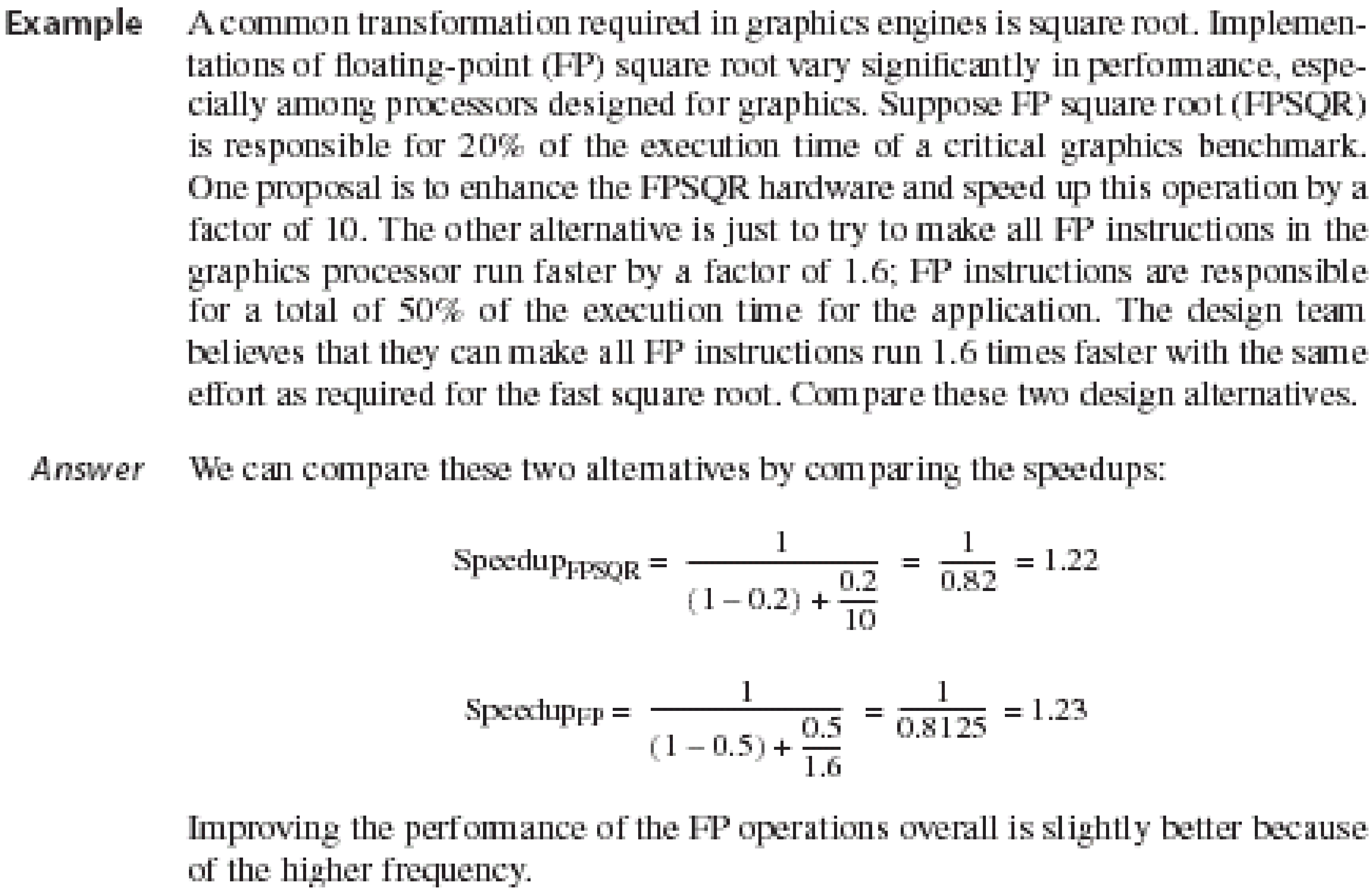
Answer

$$\text{Fraction}_{\text{enhanced}} = 0.4$$

$$\text{Speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

Example 2



Example A common transformation required in graphics engines 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 a total of 50% 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.

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.

The CPU Performance Equation

- Computer designers refer to the time of a clock period by its duration (e.g., 1 ns) or by its rate (e.g., 1 GHz)
  - —ticks, clock ticks, clock periods, clocks, cycles, or clock cycles
- CPU time for a program can then be expressed two ways:

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

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

The CPU Performance Equation (Cont’d)

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

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

or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{Clock cycle time}}{\text{Clock rate}}$$

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

The CPU Performance Equation (Cont'd)

- The Processor Performance Equation

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

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

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

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

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

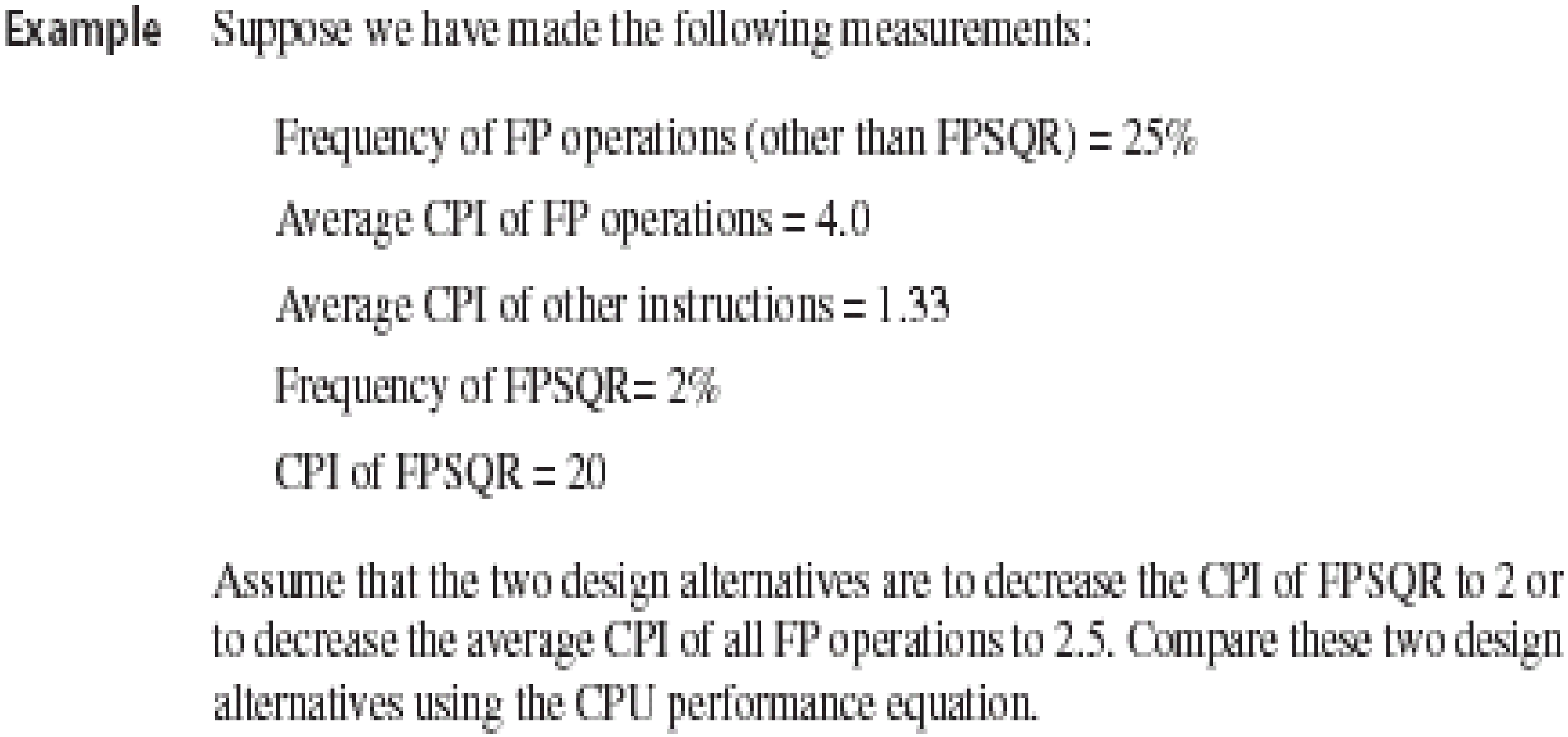
Principles of Computer Design

- Different instruction types having different CPIs

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

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

Example 3



Example Suppose we have made the following measurements:

Frequency of FP operations (other than FPSQR) = 25%

Average CPI of FP operations = 4.0

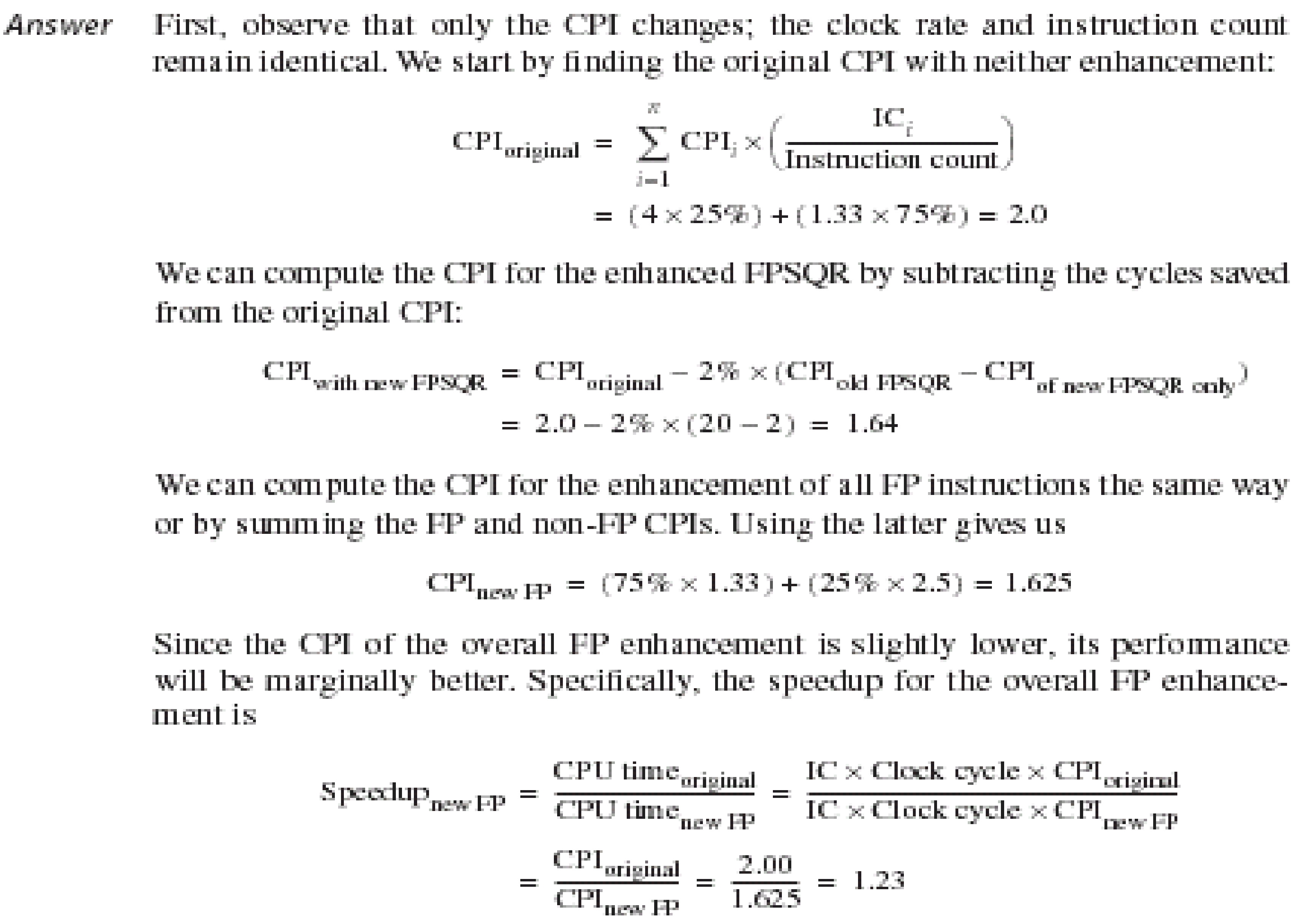
Average CPI of other instructions = 1.33

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 CPU performance equation.

Example 3 (Cont'd)



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:

$$\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{of new FPSQR only}})$$
$$= 2.0 - 2\% \times (20 - 2) = 1.64$$

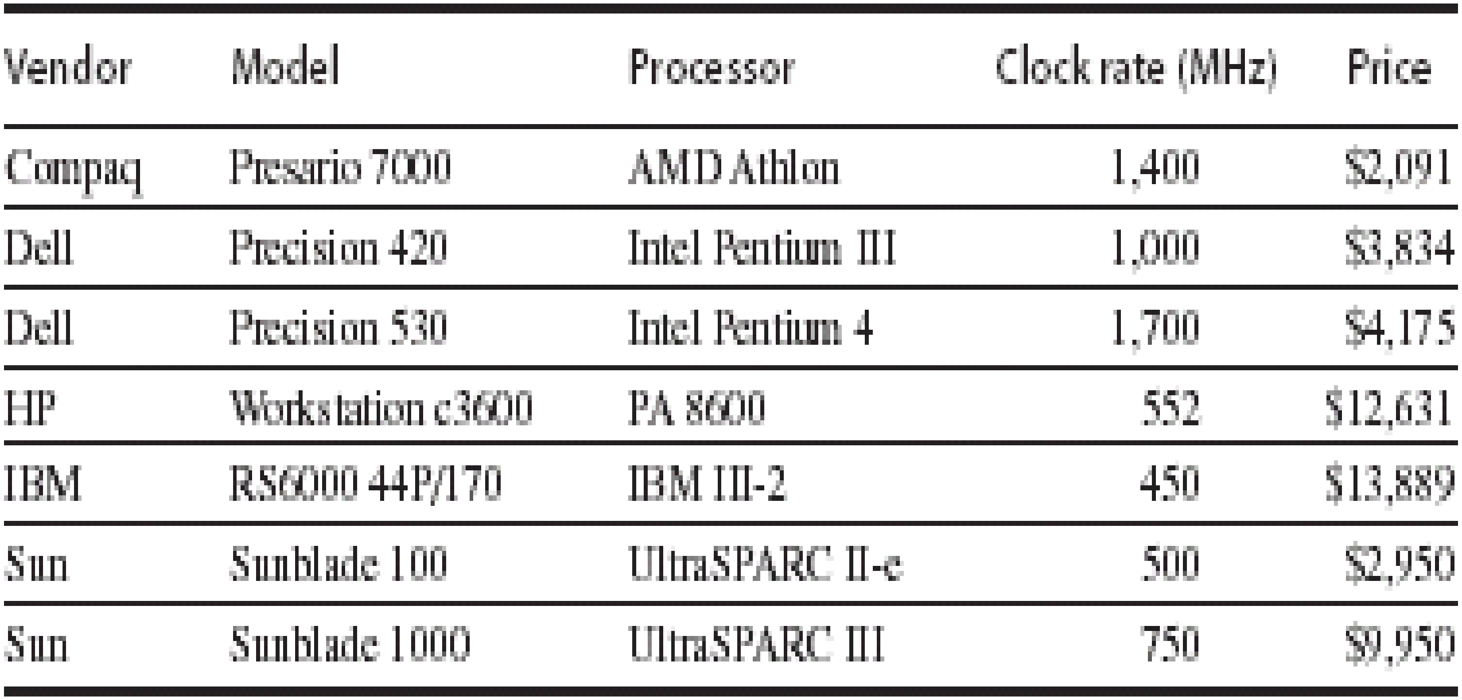
We can compute the CPI for the enhancement of all FP instructions the same way or by summing the FP and non-FP CPIs. Using the latter gives us

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

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

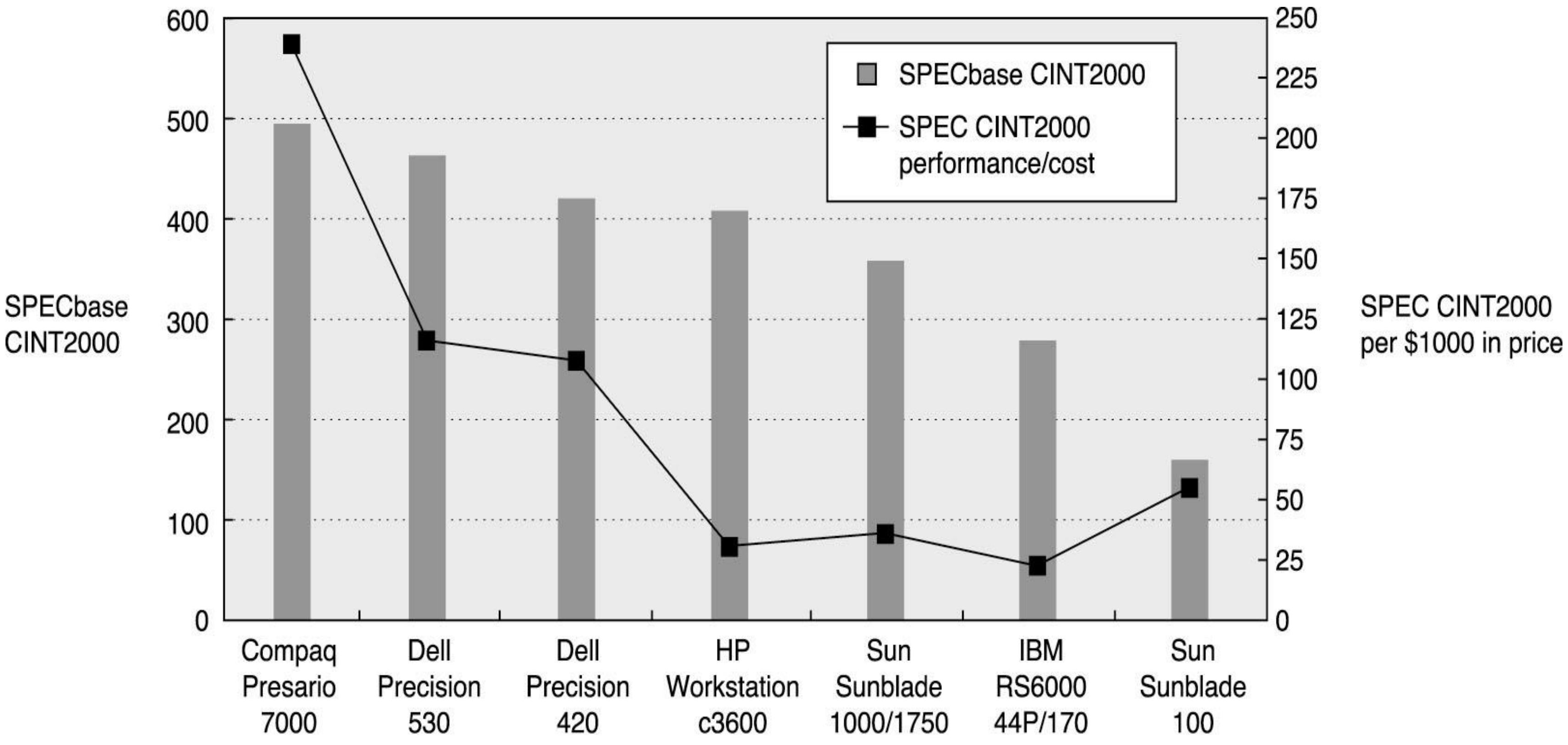
$$\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{new FP}}}$$
$$= \frac{\text{CPI}_{\text{original}}}{\text{CPI}_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23$$

Performance and Price-Performance for Desktop Systems

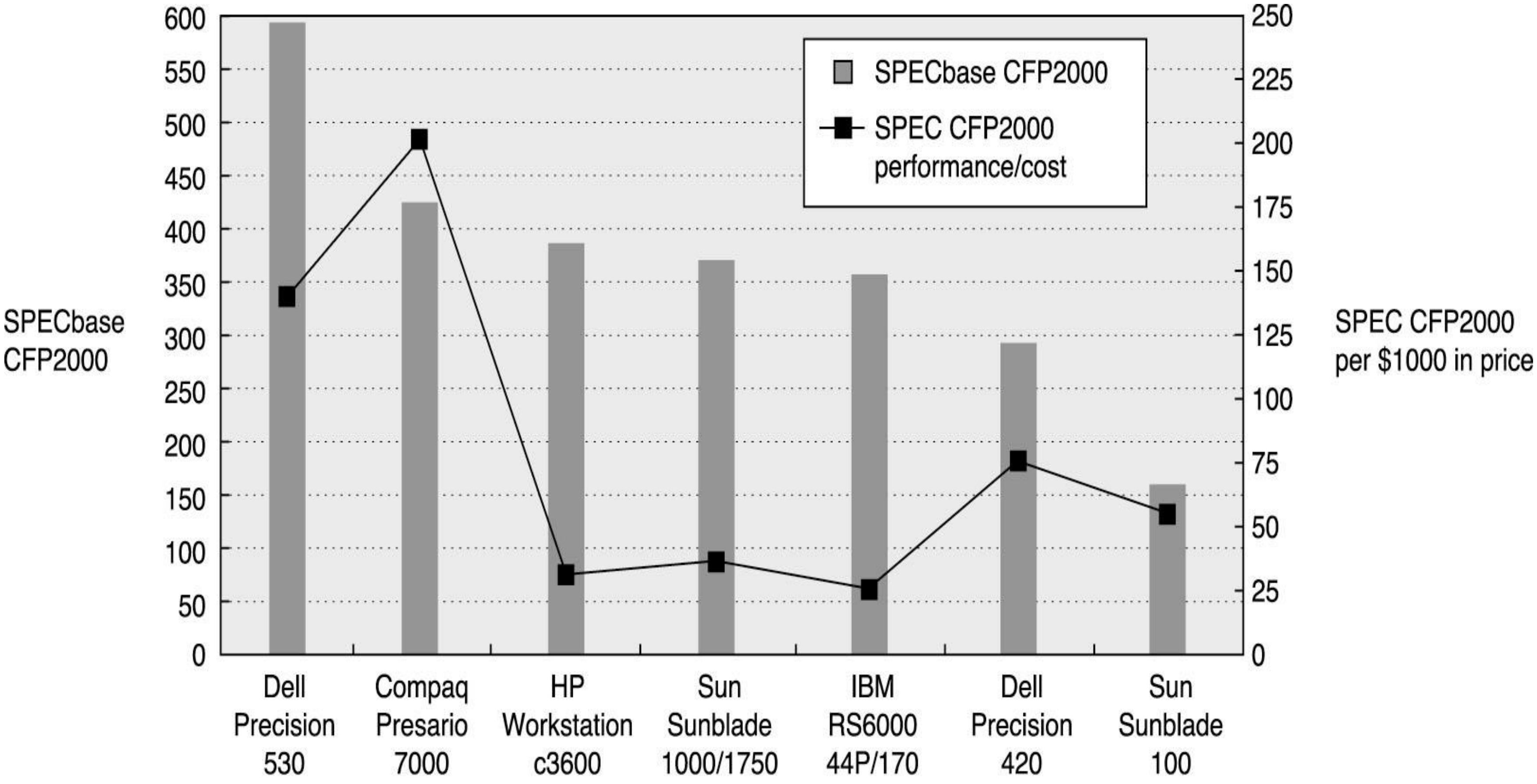


Vendor	Model	Processor	Clock rate (MHz)	Price
Compaq	Presario 7000	AMD Athlon	1,400	$2,091
Dell	Precision 420	Intel Pentium III	1,000	$3,834
Dell	Precision 530	Intel Pentium 4	1,700	$4,175
HP	Workstation c3600	PA 8600	552	$12,631
IBM	RS6000 44P/170	IBM III-2	450	$13,889
Sun	Sunblade 100	UltraSPARC II-e	500	$2,950
Sun	Sunblade 1000	UltraSPARC III	750	$9,950

Performance and Price-Performance Using SPEC CINT2000



Performance and Price-Performance Using SPEC CFP2000



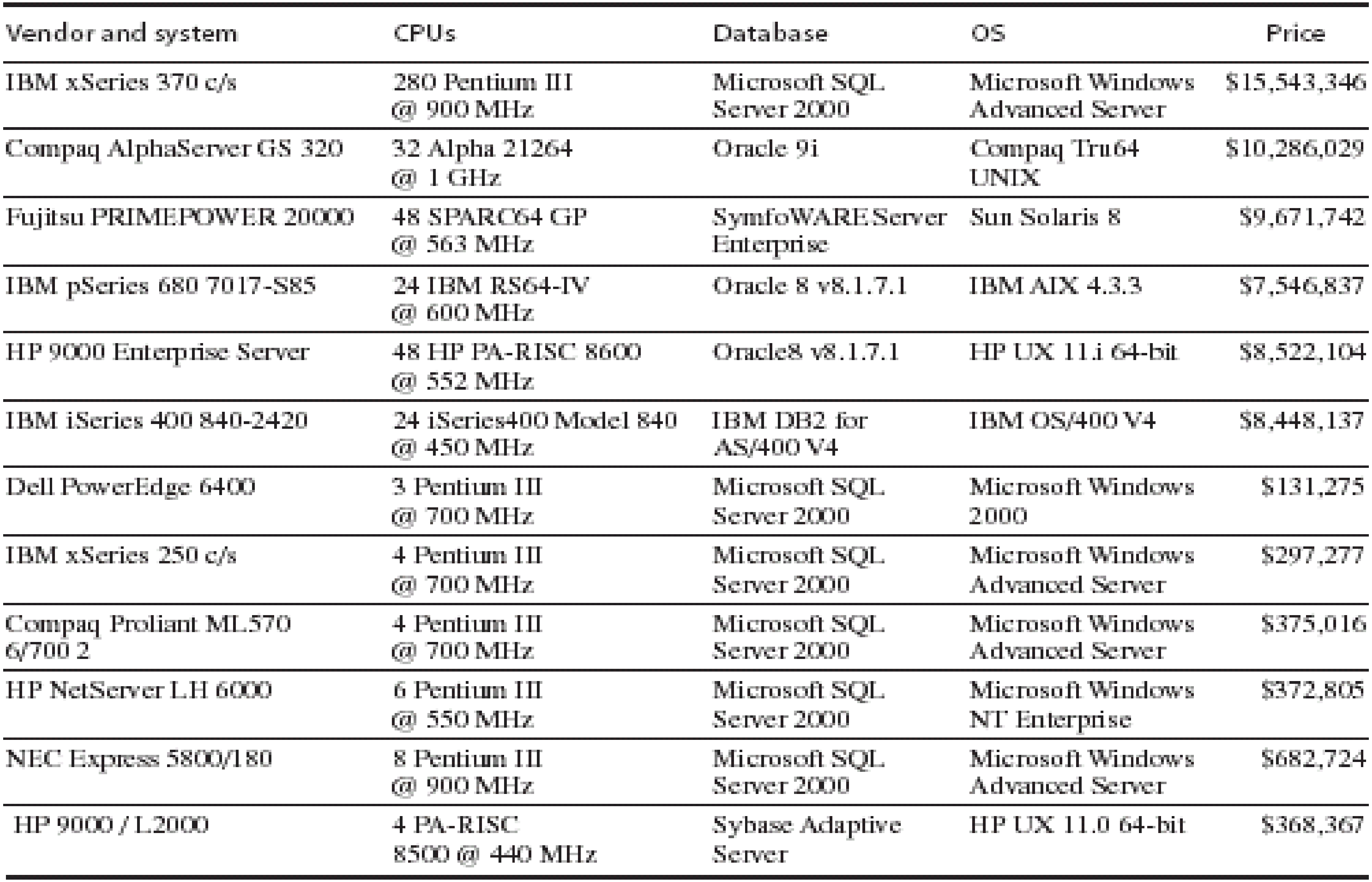
Performance and Price-Performance for Transaction-Processing Servers

- The standard industry benchmark for OLTP is TPC-C, which relies on a database system to perform queries and updates
- OLTP market is large and quite varied
- An incredible range of computing systems used for these applications
- For TPC-C
  - —performance is measured in transactions per minute (TPM)
  - —price-performance is measured in TPM per dollar

Factors Make TPC-C Interesting

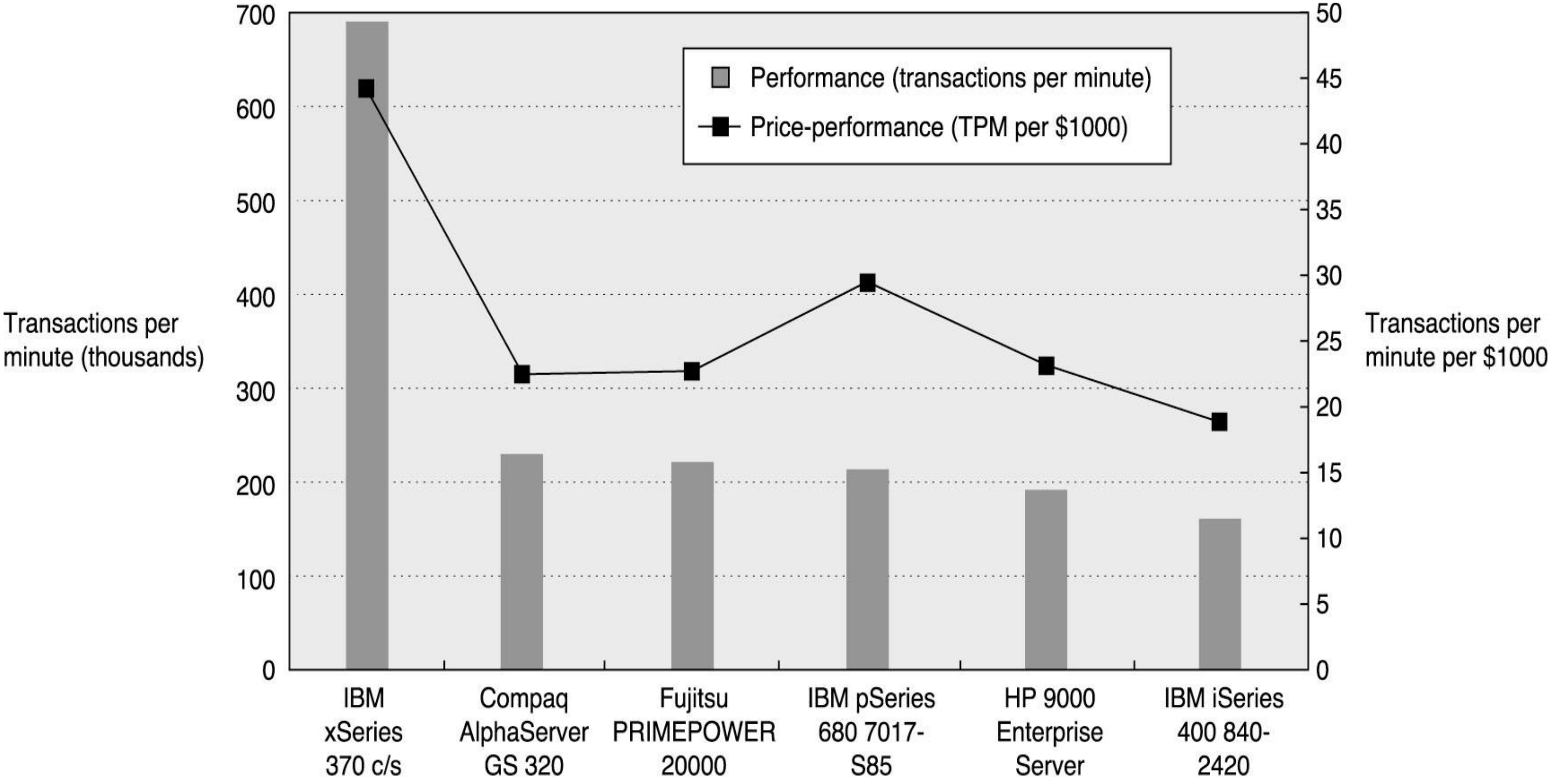
- TPC-C is a reasonable approximation to a real OLTP application
- TPC-C measures total system performance
- The rules and reporting execution time are very complete, resulting in more comparable numbers
- Computer system vendors devote significant effort to making TPC-C run well
- Vendors are required to report both performance and price-performance, enabling us to examine both

Characteristics of OLTP Systems

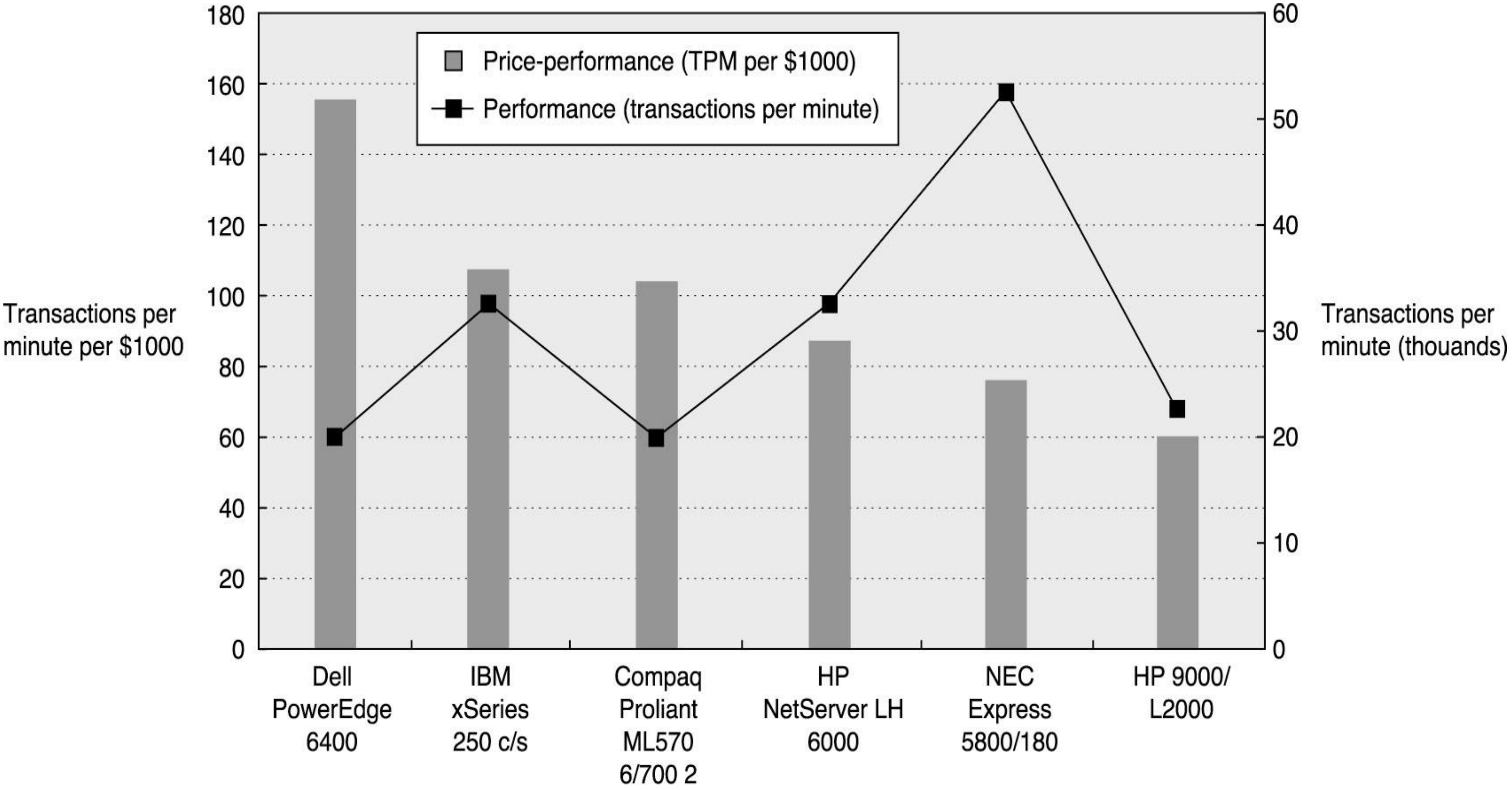


Vendor and system	CPUs	Database	OS	Price
IBM xSeries 370 c/s	280 Pentium III @ 900 MHz	Microsoft SQL Server 2000	Microsoft Windows Advanced Server	$15,543,346
Compaq AlphaServer GS 320	32 Alpha 21264 @ 1 GHz	Oracle 9i	Compaq Tru64 UNIX	$10,286,029
Fujitsu PRIMEPOWER 20000	48 SPARC64 GP @ 563 MHz	SymfoWARE Server Enterprise	Sun Solaris 8	$9,671,742
IBM pSeries 680 7017-S85	24 IBM RS64-IV @ 600 MHz	Oracle 8 v8.1.7.1	IBM AIX 4.3.3	$7,546,837
HP 9000 Enterprise Server	48 HP PA-RISC 8600 @ 552 MHz	Oracle8 v8.1.7.1	HP UX 11.i 64-bit	$8,522,104
IBM iSeries 400 840-2420	24 iSeries400 Model 840 @ 450 MHz	IBM DB2 for AS/400 V4	IBM OS/400 V4	$8,448,137
Dell PowerEdge 6400	3 Pentium III @ 700 MHz	Microsoft SQL Server 2000	Microsoft Windows 2000	$131,275
IBM xSeries 250 c/s	4 Pentium III @ 700 MHz	Microsoft SQL Server 2000	Microsoft Windows Advanced Server	$297,277
Compaq Proliant ML570 6/700 2	4 Pentium III @ 700 MHz	Microsoft SQL Server 2000	Microsoft Windows Advanced Server	$375,016
HP NetServer LH 6000	6 Pentium III @ 550 MHz	Microsoft SQL Server 2000	Microsoft Windows NT Enterprise	$372,805
NEC Express 5800/180	8 Pentium III @ 900 MHz	Microsoft SQL Server 2000	Microsoft Windows Advanced Server	$682,724
HP 9000 / L2000	4 PA-RISC 8500 @ 440 MHz	Sybase Adaptive Server	HP UX 11.0 64-bit	$368,367

Performance and Price-Performance Using TPC-C



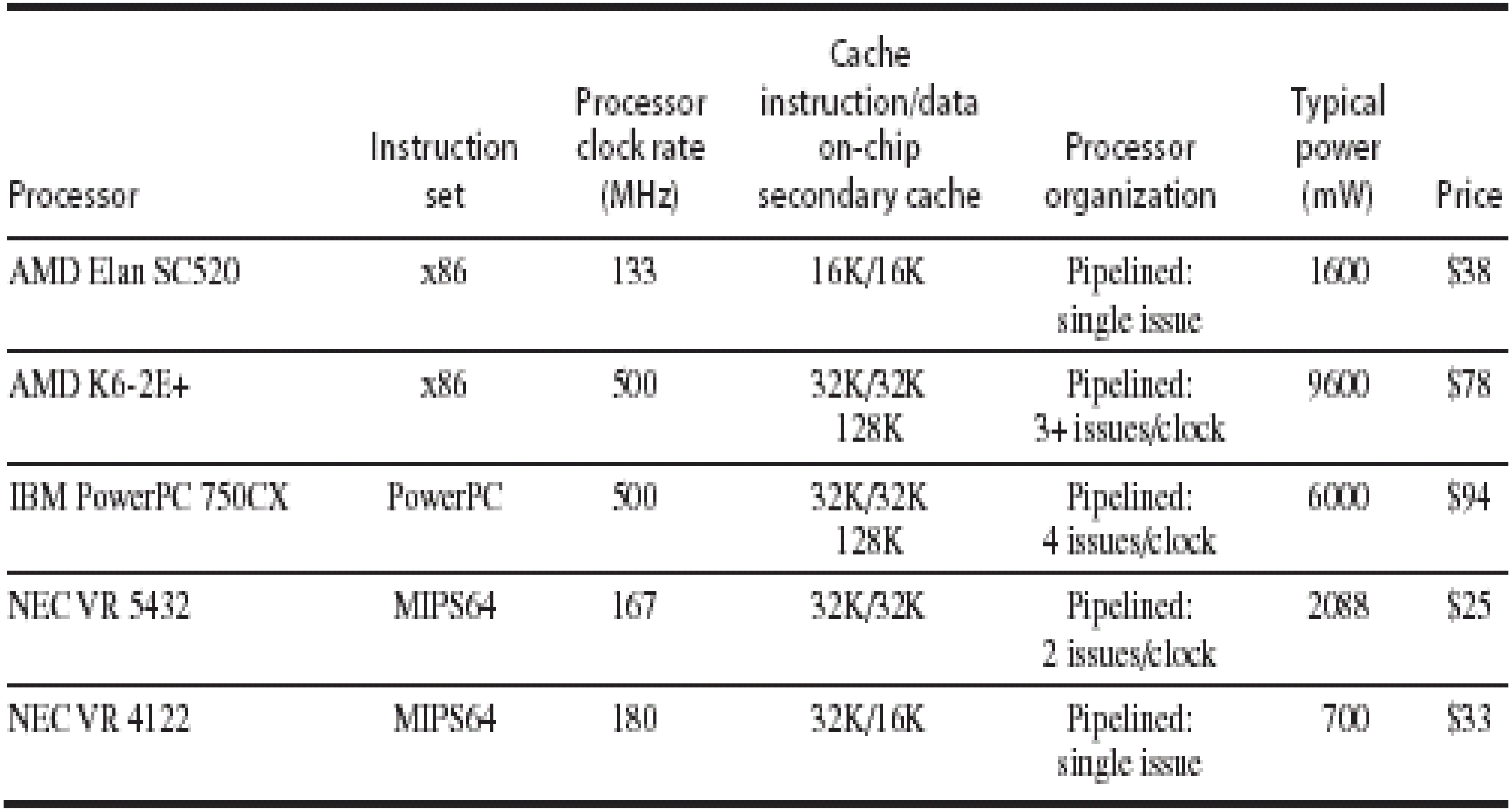
Price-Performance and Overall Performance



Performance and Price-Performance for Embedded Processors

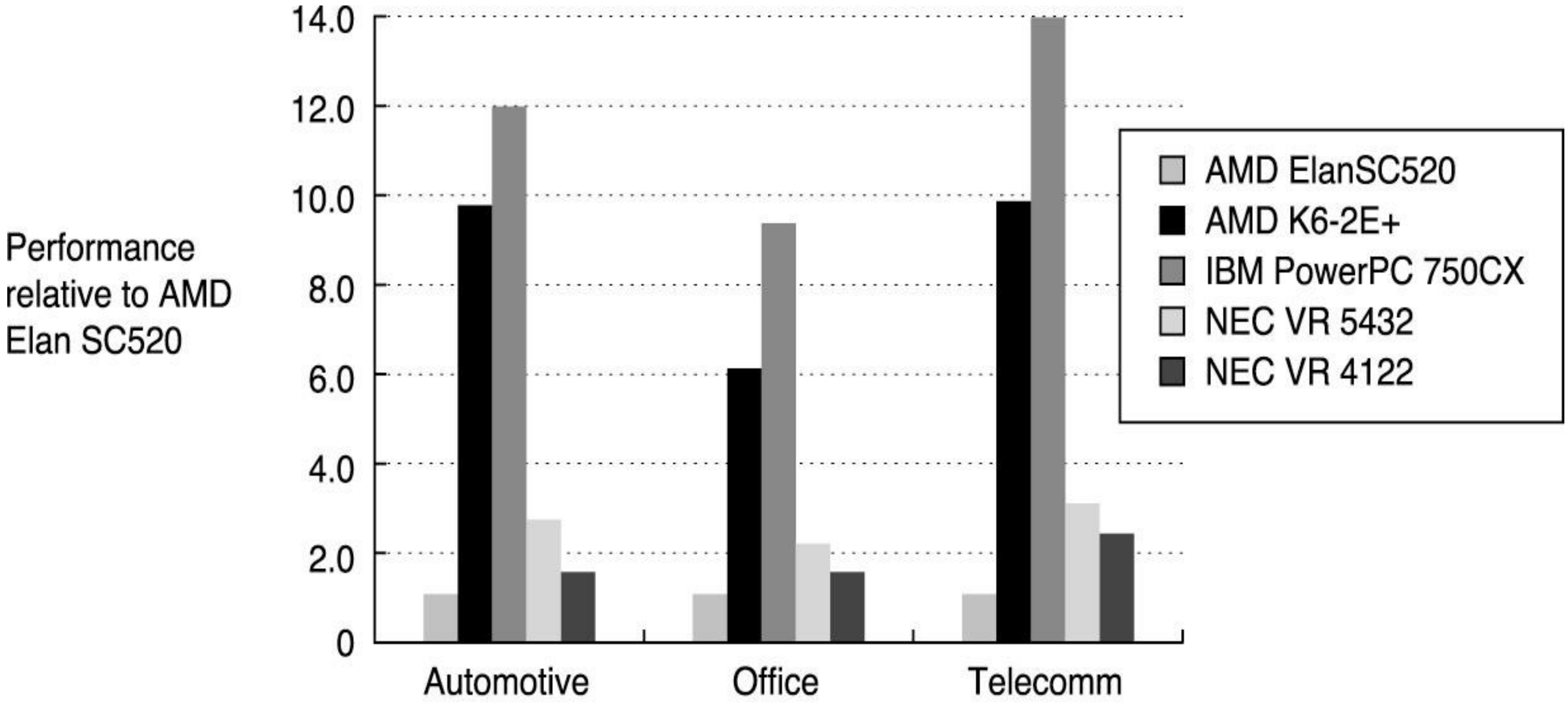
- Comparing performance and price-performance of embedded processors is more difficult than for the desktop or server environments
  - —Benchmarking is in its comparative infancy in the embedded space
  - —Embedded processors are often designed for a particular class of applications
  - —Cost and power are often the most important factors for an embedded application
  - —Power is the critical constraint in embedded systems

Characteristics of Five Processors

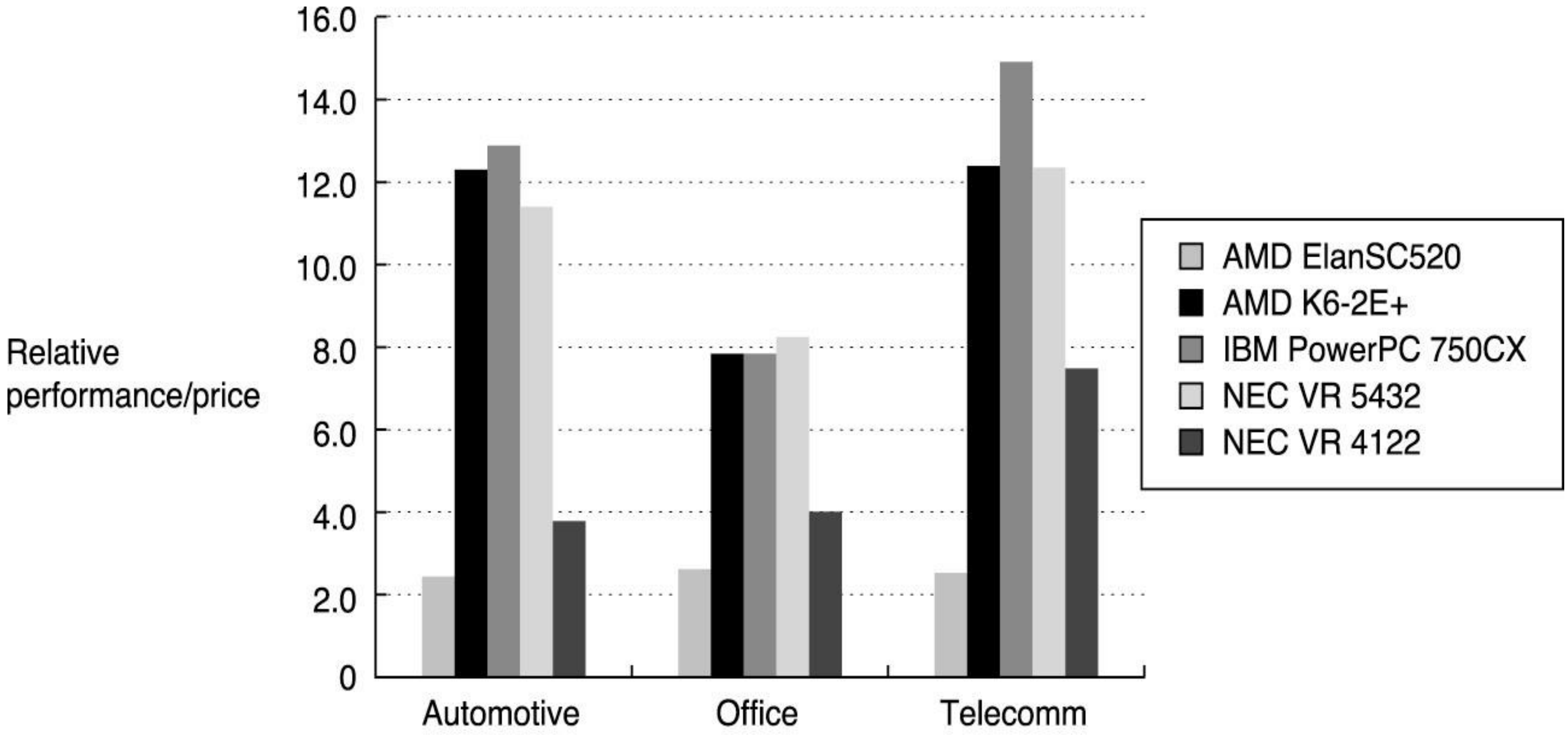


Processor	Instruction set	Processor clock rate (MHz)	Cache instruction/data on-chip secondary cache	Processor organization	Typical power (mW)	Price
AMD Elan SC520	x86	133	16K/16K	Pipelined: single issue	1600	$38
AMD K6-2E+	x86	500	32K/32K 128K	Pipelined: 3+ issues/clock	9600	$78
IBM PowerPC 750CX	PowerPC	500	32K/32K 128K	Pipelined: 4 issues/clock	6000	$94
NEC VR 5432	MIPS64	167	32K/32K	Pipelined: 2 issues/clock	2088	$25
NEC VR 4122	MIPS64	180	32K/16K	Pipelined: single issue	700	$33

Relative Performance Using Three EEMBC Benchmark Suites



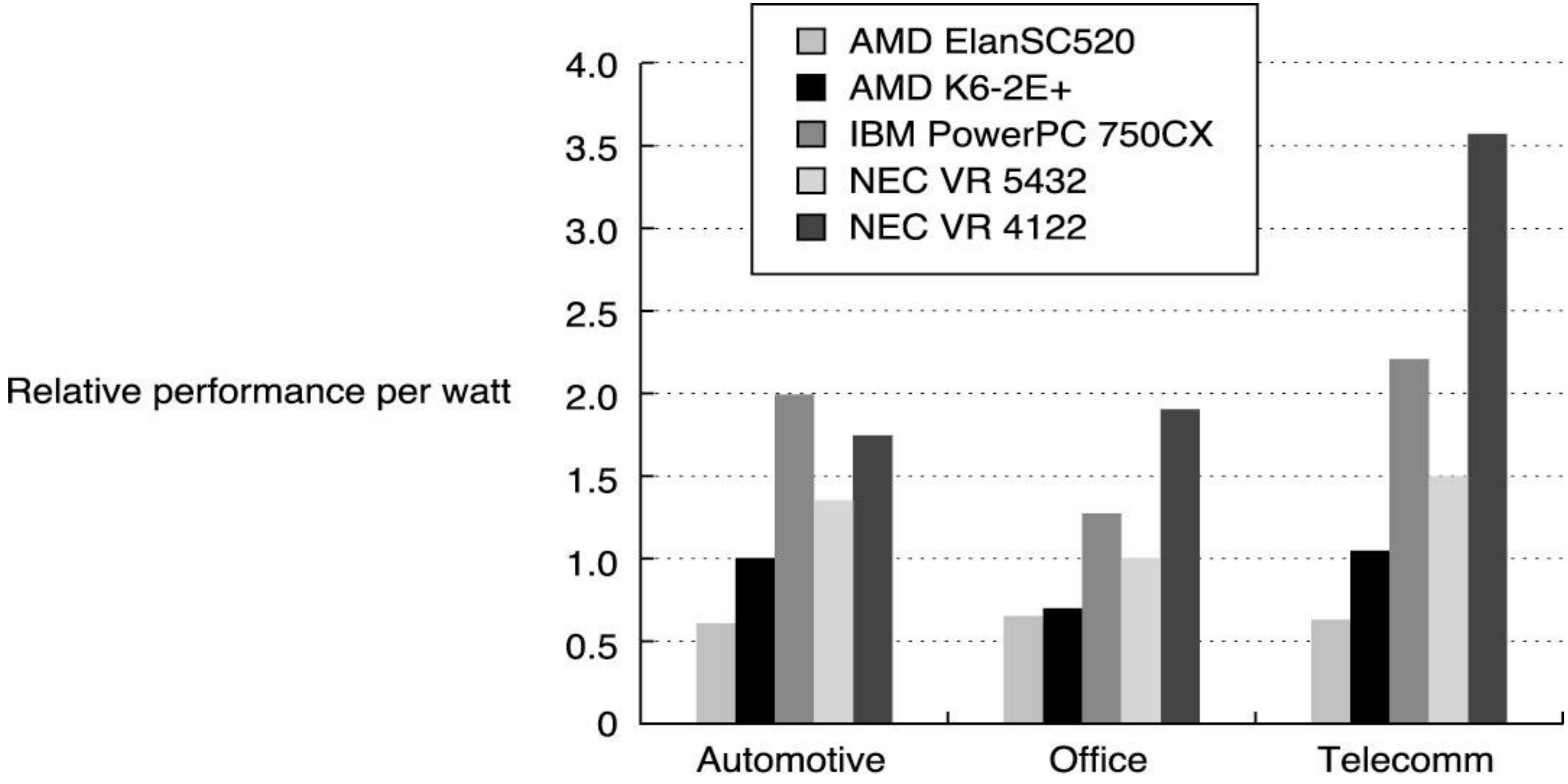
Relative Price-Performance Using Three EEMBC Benchmark Suites



Power Consumption and Efficiency

- Cost and power are often at least as important as performance in the embedded market
- Memory is often the next most costly part of an embedded system
- The amount of memory needed for the application is critical
- Power is often a determining factor in choosing a processor, especially for battery-powered systems

Relative Performance Per Watt for the Five Embedded Processors



Fallacy 1

- Multiprocessors are a silver bullet

Pitfall 1

- Falling prey to Amdahl’s heartbreaking law

Pitfall 2

- A single point of failure

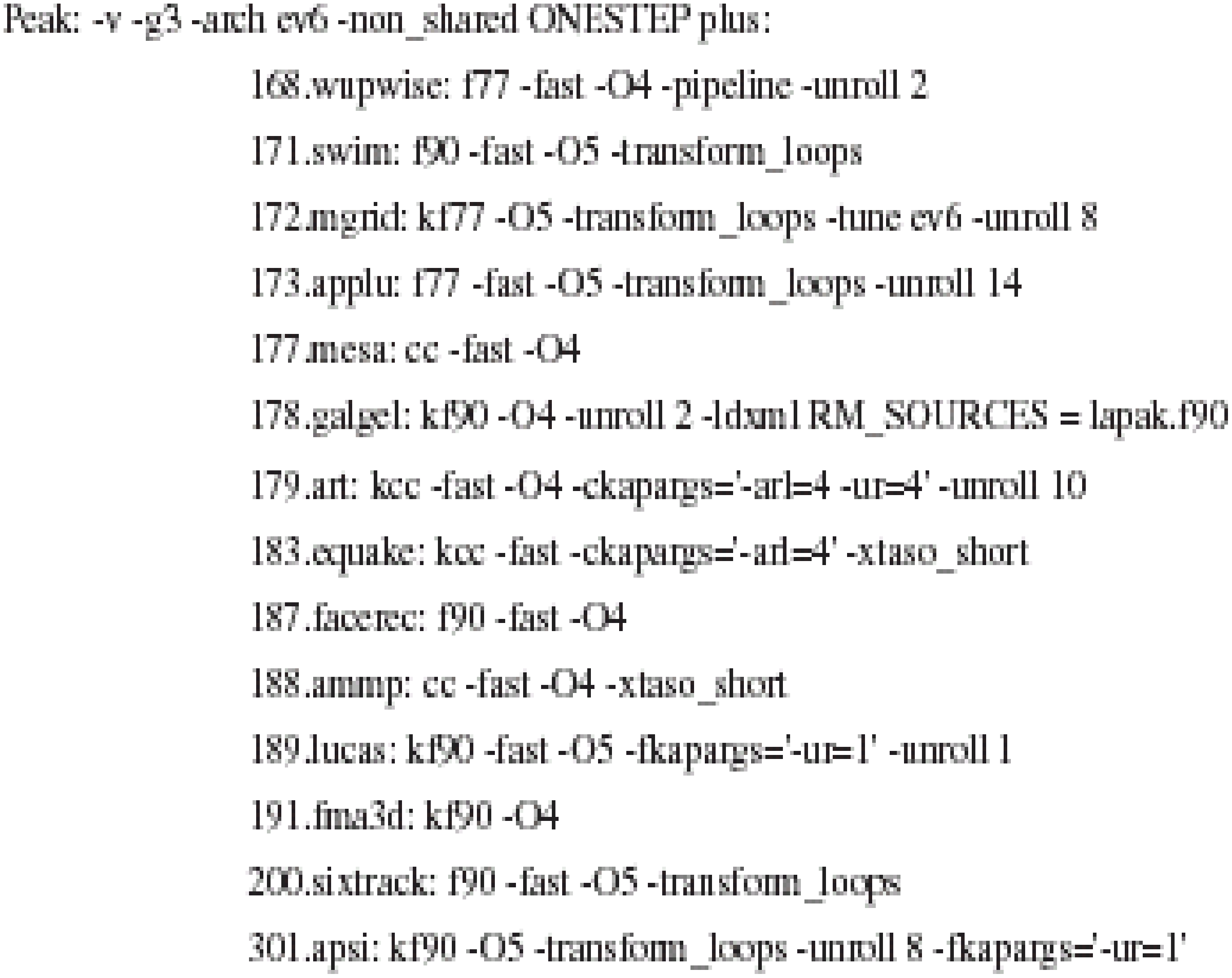
Fallacy 2

- Hardware enhancements that increase performance improve energy efficiency or are at worst energy neutral

Fallacy 3

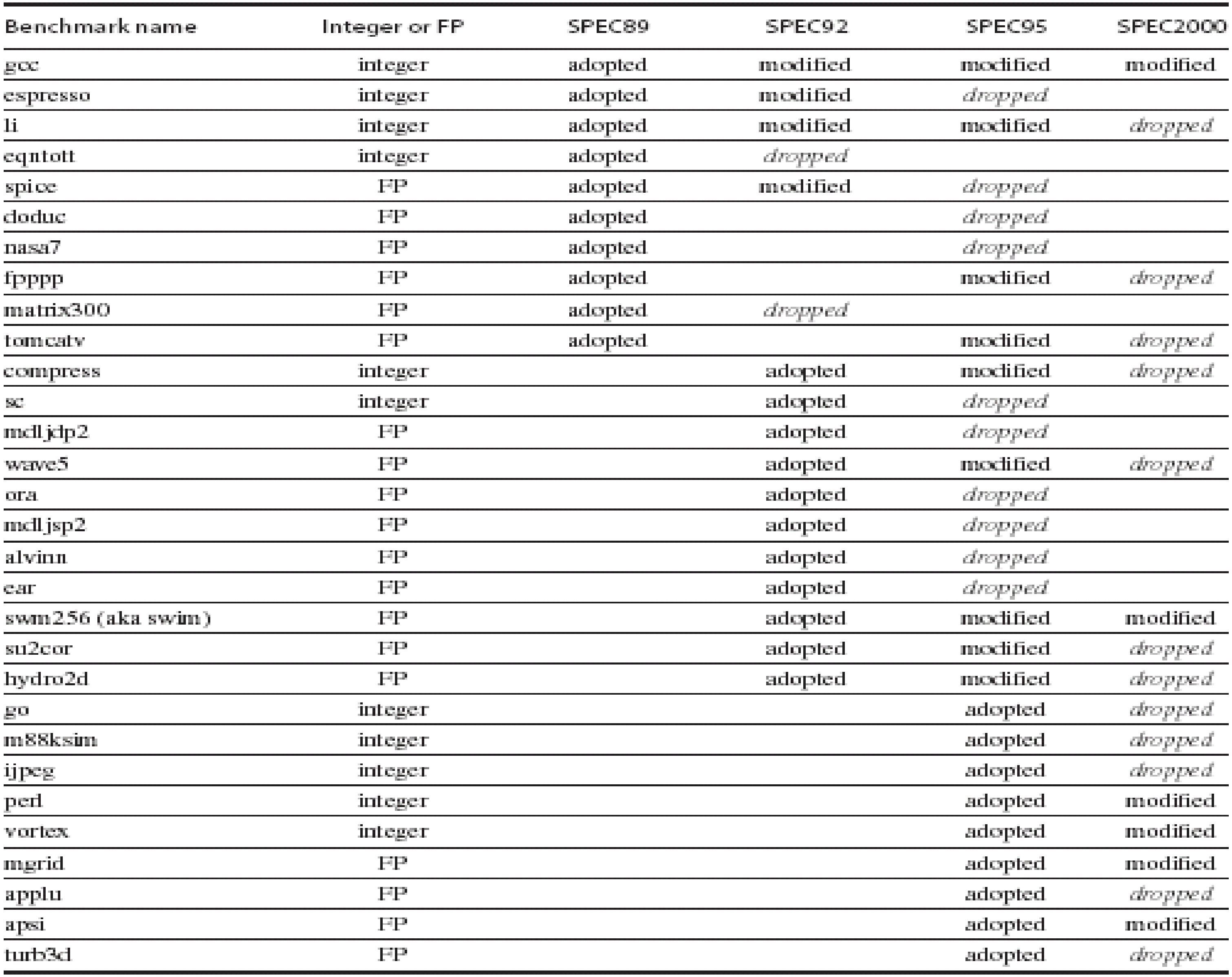
- Benchmarks remain valid indefinitely
- Small kernels or programs that spend their time in a very small number of lines of code are particularly vulnerable
- Benchmark-specific flags are used
- The performance difference between the baseline and tuned numbers can be substantial
- Ongoing improvements in technology can also change what a benchmark measures
- Over a long period of time, changes may make even a well-chosen benchmark obsolete

Fallacy 3 (Cont'd)



```
Peak: -v -g3 -arch ev6 -non_shared ONESTEP plus:
          168.wupwise: f77 -fast -O4 -pipeline -unroll 2
          171.swim: f90 -fast -O5 -transform_loops
          172.mgrid: kf77 -O5 -transform_loops -tune ev6 -unroll 8
          173.applu: f77 -fast -O5 -transform_loops -unroll 14
          177.mesa: cc -fast -O4
          178.galgel: kf90 -O4 -unroll 2 -ldxml RM_SOURCES = lapak.f90
          179.art: kcc -fast -O4 -ckapargs='-arl=4 -ur=4' -unroll 10
          183.equake: kcc -fast -ckapargs='-arl=4' -xtaso_short
          187.facerec: f90 -fast -O4
          188.ammp: cc -fast -O4 -xtaso_short
          189.lucas: kf90 -fast -O5 -fkapargs='-ur=1' -unroll 1
          191.fma3d: kf90 -O4
          200.sixtrack: f90 -fast -O5 -transform_loops
          301.apsi: kf90 -O5 -transform_loops -unroll 8 -fkapargs='-ur=1'
```

Fallacy 3 (Cont’d)



Benchmark name	Integer or FP	SPEC89	SPEC92	SPEC95	SPEC2000
gcc	integer	adopted	modified	modified	modified
espresso	integer	adopted	modified	*dropped*	
li	integer	adopted	modified	modified	*dropped*
eqntott	integer	adopted	*dropped*		
spice	FP	adopted	modified	*dropped*	
doduc	FP	adopted		*dropped*	
nasa7	FP	adopted		*dropped*	
fpppp	FP	adopted		modified	*dropped*
matrix300	FP	adopted	*dropped*		
tomcatv	FP	adopted		modified	*dropped*
compress	integer		adopted	modified	*dropped*
sc	integer		adopted	*dropped*	
mdljdp2	FP		adopted	*dropped*	
wave5	FP		adopted	modified	*dropped*
ora	FP		adopted	*dropped*	
mdljsp2	FP		adopted	*dropped*	
alvinn	FP		adopted	*dropped*	
ear	FP		adopted	*dropped*	
swm256 (aka swim)	FP		adopted	modified	modified
su2cor	FP		adopted	modified	*dropped*
hydro2d	FP		adopted	modified	*dropped*
go	integer			adopted	*dropped*
m88ksim	integer			adopted	*dropped*
ijpeg	integer			adopted	*dropped*
perl	integer			adopted	modified
vortex	integer			adopted	modified
mgrid	FP			adopted	modified
applu	FP			adopted	*dropped*
apsi	FP			adopted	modified
turb3d	FP			adopted	*dropped*

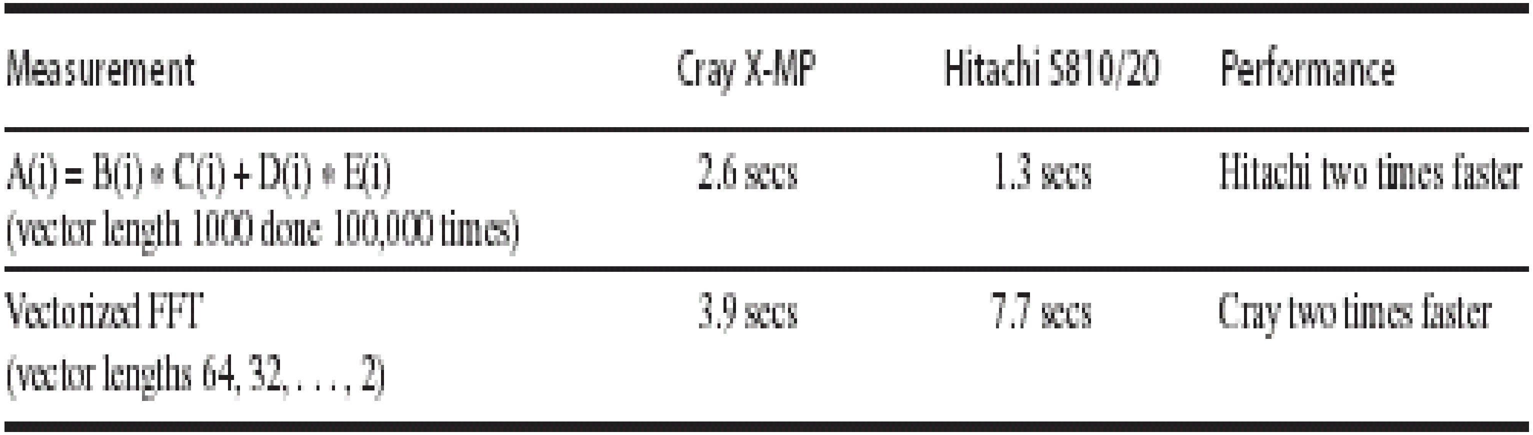
Fallacy 4

- The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so sisks practically never fail

Fallacy 5

- Peak performance tracks observed performance
- The gap between peak performance and observed performance is large and can vary significantly by benchmark
- Peak performance will be close to observed performance unless the workload consists of small programs

Fallacy 5 (Cont'd)



Measurement	Cray X-MP	Hitachi S810/20	Performance
A(i) = B(i) * C(i) + D(i) * E(i) (vector length 1000 done 100,000 times)	2.6 secs	1.3 secs	Hitachi two times faster
Vectorized FFT (vector lengths 64, 32, . . . , 2)	3.9 secs	7.7 secs	Cray two times faster

Pitfall 3

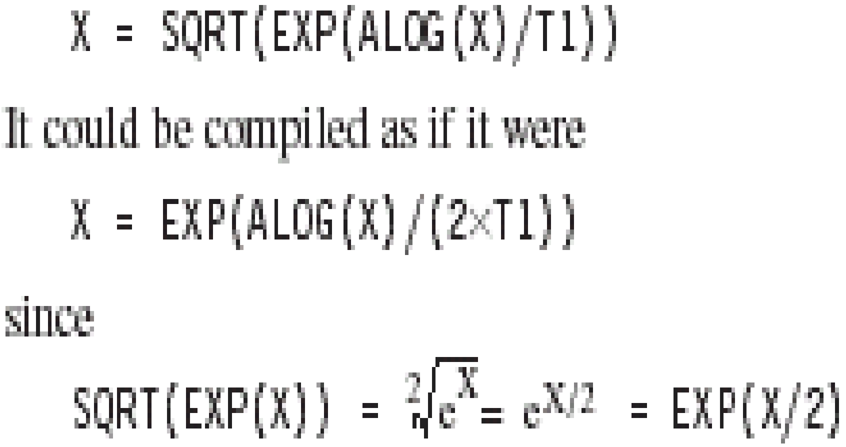
- Fault detection can lower availability

Fallacy 6

- Synthetic benchmarks predict performance for real programs
- Whetstone and Dhrystone, artificial programs may not reflect program behavior for factors not measured
- Compiler and hardware optimizations can artificially inflate performance of these benchmarks but not of real programs
- They don’t reward optimizations of behaviors that occur in real programs

Fallacy 6 - Examples

- Optimizing compilers can discard 25% of the Dhrystone code
- Most Whetstone floating-point loops execute small numbers of times or include calls inside the loop
- Compilers can optimize a key piece of the Whetstone loop, even though this is very unlikely to occur in real programs, e.g.



```
X = SQRT(EXP(ALOG(X)/T1))
```

It could be compiled as if it were

```
X = EXP(ALOG(X)/(2×T1))
```

since

$$\mathrm{SQRT(EXP(X))} = \sqrt[2]{e^{X}} = e^{X/2} = \mathrm{EXP(X/2)}$$

Fallacy 7

- MIPS is an accurate measure for comparing performance among computers

$$\text{Execution time} = \frac{\text{Instruction count}}{\text{MIPS} \times 10^6}$$

- MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction sets
- MIPS varies between programs on the same computer
- Most importantly, MIPS can vary inversely to performance

Fallacy 7 (Cont'd)

- The classic example of the last case is the MIPS rating of a machine with optional floating-point hardware
- Relative MIPSs are more likely to track relative performance differences??