+

William Stallings Computer Organization and Architecture 10th Edition

+ Chapter 2

Performance Issues

Designing for Performance

- The cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically
- Today's laptops have the computing power of an IBM mainframe from 10 or 15 years ago
- Processors are so inexpensive that we now have microprocessors we throw away
- Desktop applications that require the great power of today's microprocessor-based systems include:
  - Image processing
  - Three-dimensional rendering
  - Speech recognition
  - Videoconferencing
  - Multimedia authoring
  - Voice and video annotation of files
  - Simulation modeling

- Businesses are relying on increasingly powerful servers to handle transaction and database processing and to support massive client/server networks that have replaced the huge mainframe computer centers of yesteryear
- Cloud service providers use massive high-performance banks of servers to satisfy high-volume, high-transaction-rate applications for a broad spectrum of clients

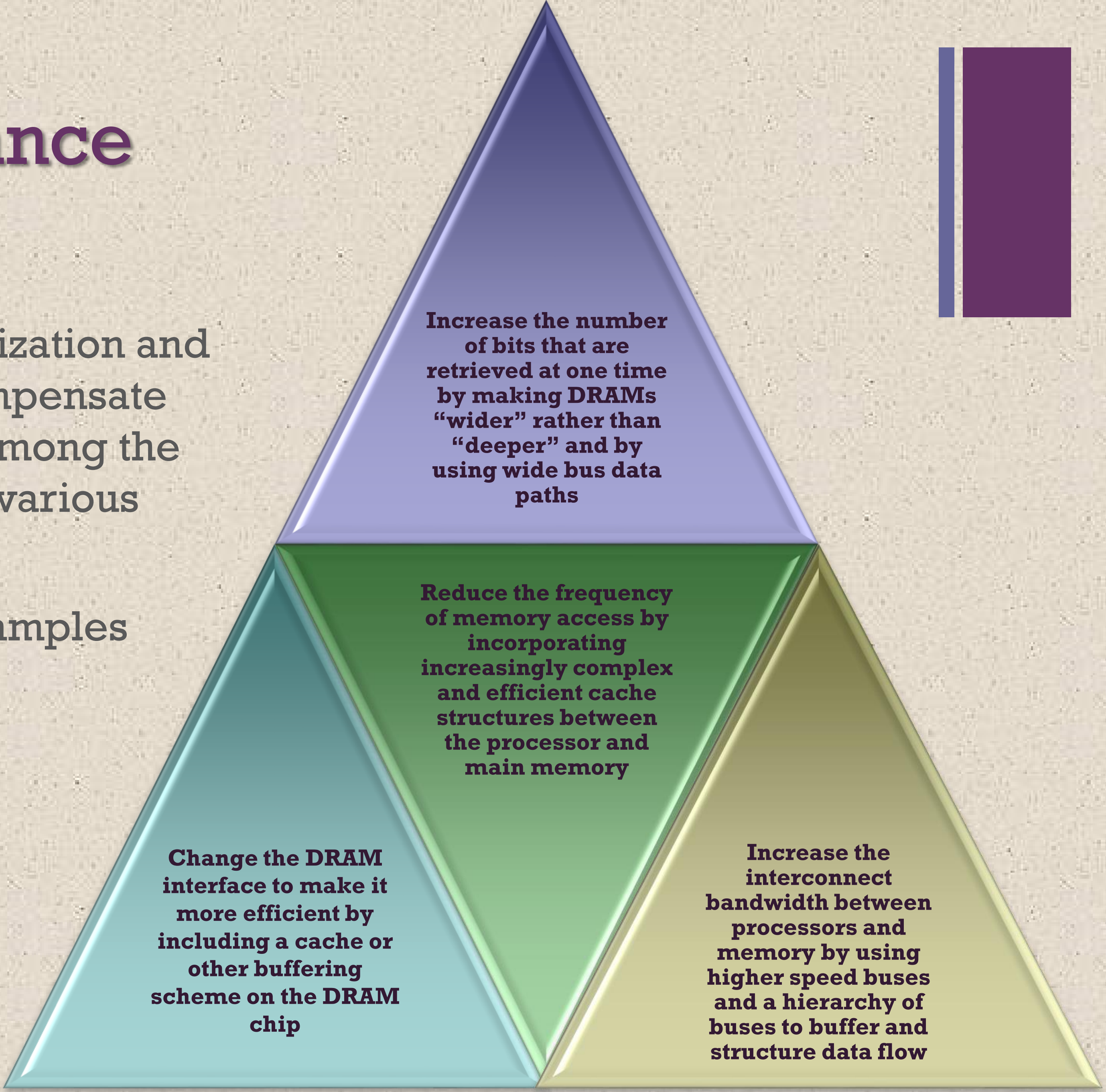
Microprocessor Speed

Techniques built into contemporary processors include:

Pipelining	• Processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously
Branch prediction	• Processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next
Superscalar execution	• This is the ability to issue more than one instruction in every processor clock cycle. (In effect, multiple parallel pipelines are used.)
Data flow analysis	• Processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
Speculative execution	• Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations, keeping execution engines as busy as possible

Performance Balance

- Adjust the organization and architecture to compensate for the mismatch among the capabilities of the various components
- Architectural examples include:



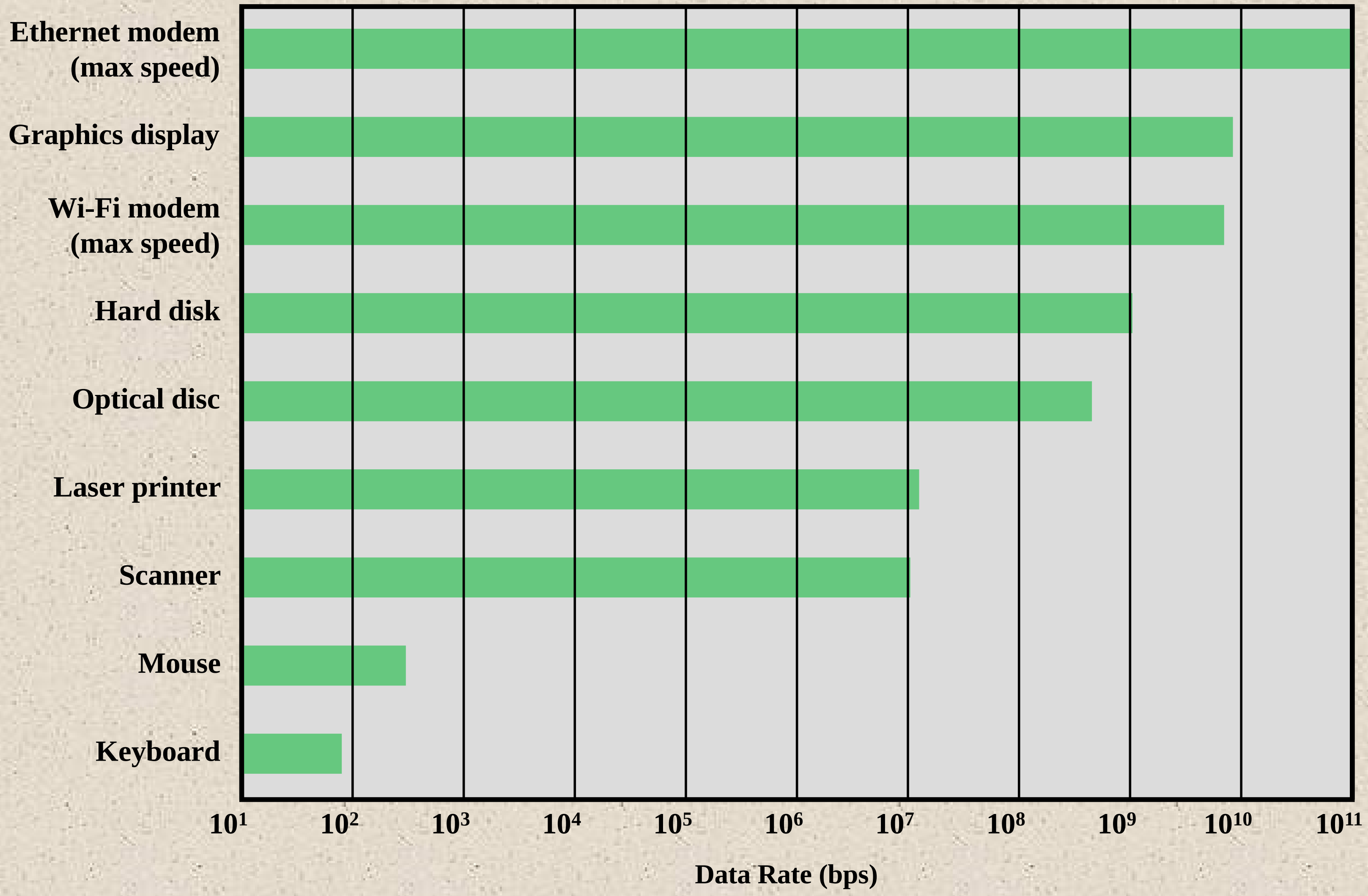
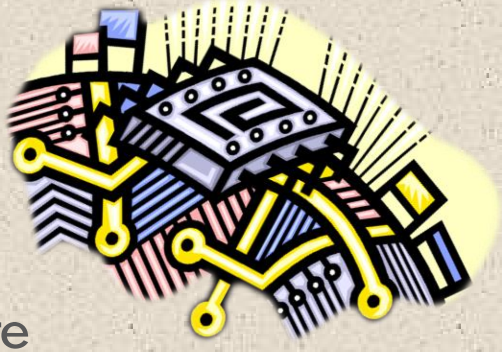


Figure 2.1 Typical I/O Device Data Rates

Improvements in Chip Organization and Architecture

- Increase hardware speed of processor
  - Fundamentally due to shrinking logic gate size
    - More gates, packed more tightly, increasing clock rate
    - Propagation time for signals reduced
- Increase size and speed of caches
  - Dedicating part of processor chip
    - Cache access times drop significantly
- Change processor organization and architecture
  - Increase effective speed of instruction execution
  - Parallelism



Problems with Clock Speed and Login Density

- Power
  - Power density increases with density of logic and clock speed
  - Dissipating heat
- RC delay
  - Speed at which electrons flow limited by resistance and capacitance of metal wires connecting them
  - Delay increases as the RC product increases
  - As components on the chip decrease in size, the wire interconnects become thinner, increasing resistance
  - Also, the wires are closer together, increasing capacitance
- Memory latency
  - Memory speeds lag processor speeds

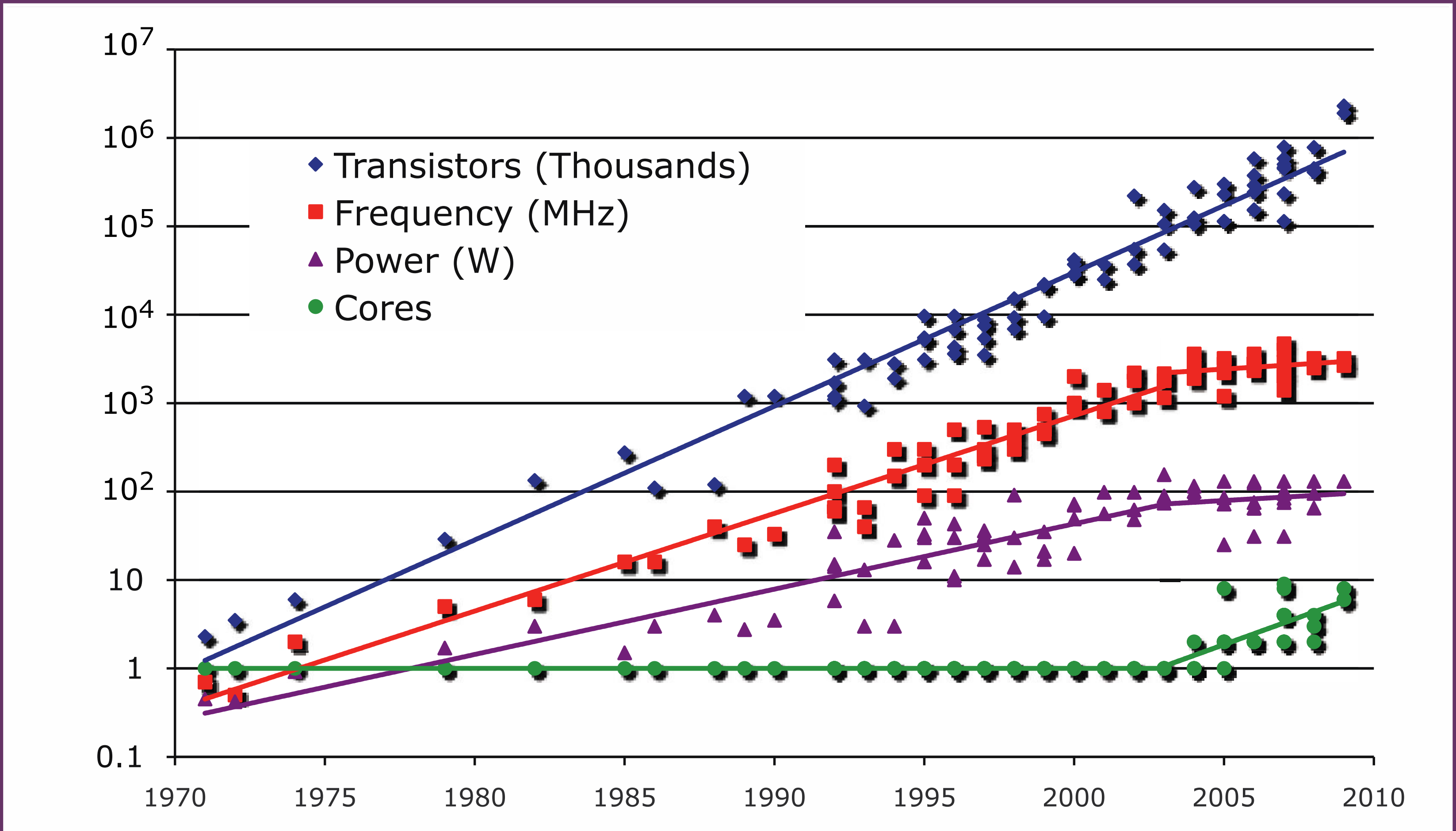
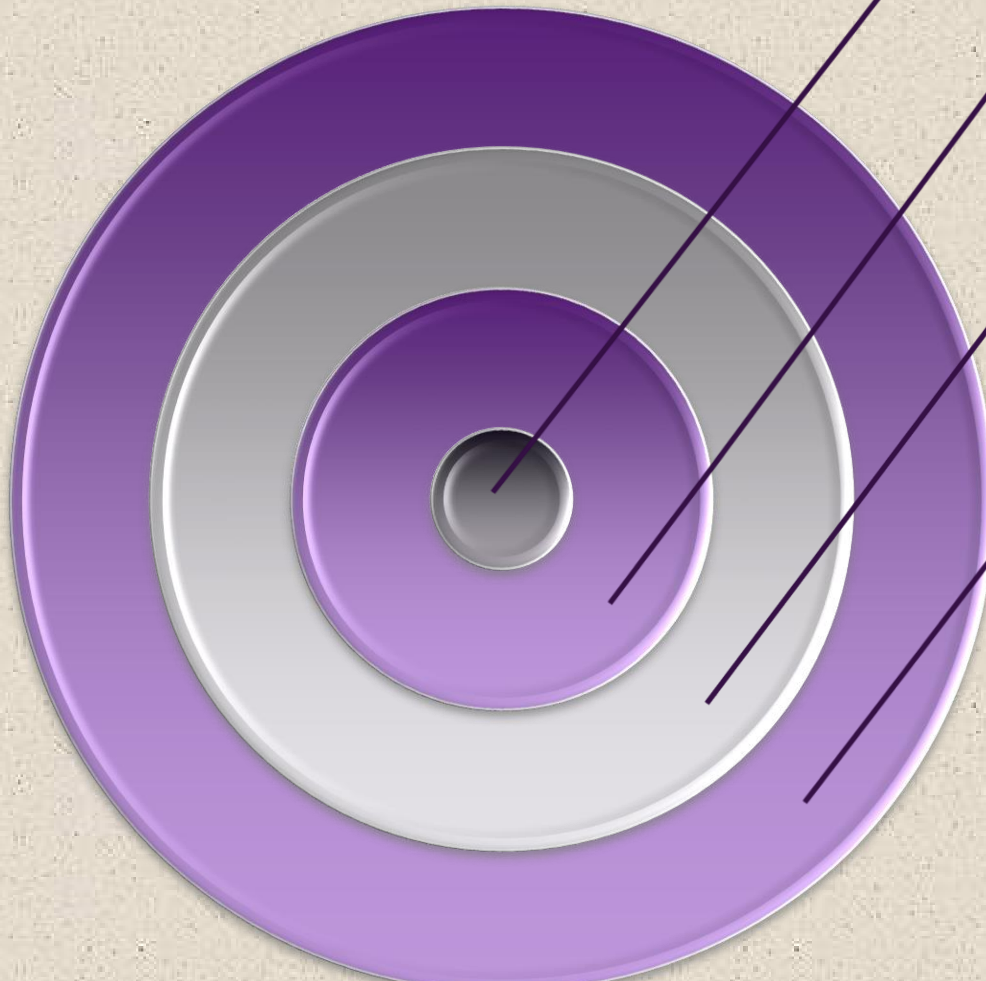


Figure 2.2 Processor Trends

Multicore



The use of multiple processors on the same chip provides the potential to increase performance without increasing the clock rate

Strategy is to use two simpler processors on the chip rather than one more complex processor

With two processors larger caches are justified

As caches became larger it made performance sense to create two and then three levels of cache on a chip

+

Many Integrated Core (MIC) Graphics Processing Unit (GPU)

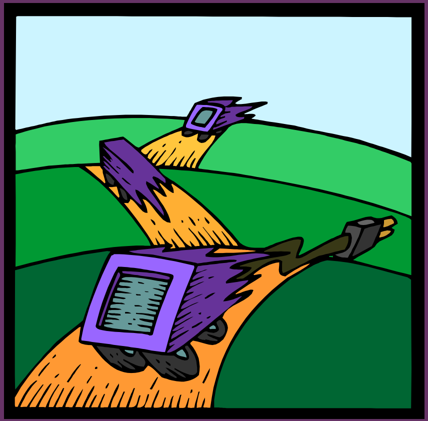
MIC

- Leap in performance as well as the challenges in developing software to exploit such a large number of cores
- The multicore and MIC strategy involves a homogeneous collection of general purpose processors on a single chip

GPU

- Core designed to perform parallel operations on graphics data
- Traditionally found on a plug-in graphics card, it is used to encode and render 2D and 3D graphics as well as process video
- Used as vector processors for a variety of applications that require repetitive computations

Amdahl's Law



- Gene Amdahl
- Deals with the potential speedup of a program using multiple processors compared to a single processor
- Illustrates the problems facing industry in the development of multi-core machines
  - Software must be adapted to a highly parallel execution environment to exploit the power of parallel processing
- Can be generalized to evaluate and design technical improvement in a computer system

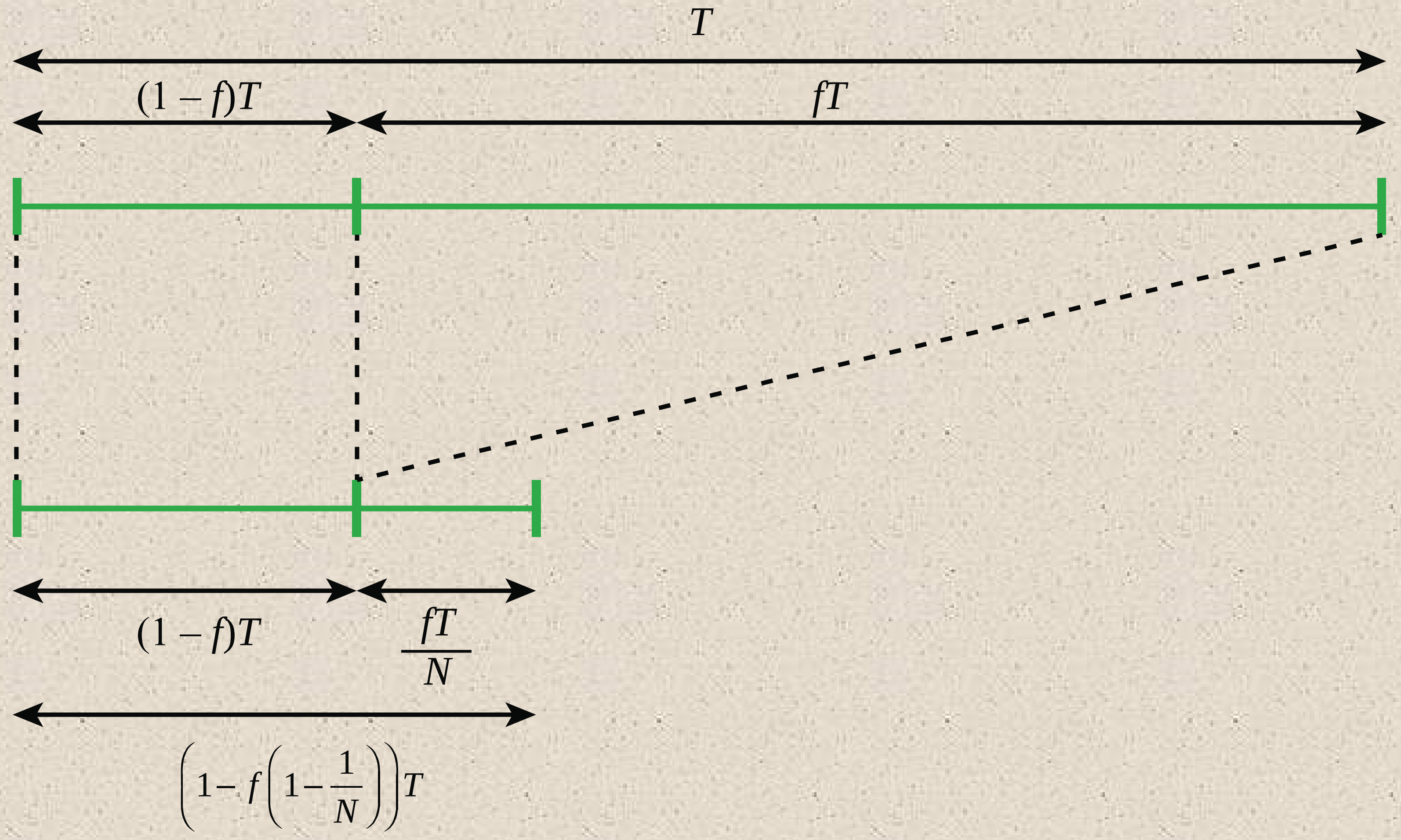


Figure 2.3 Illustration of Amdahl's Law

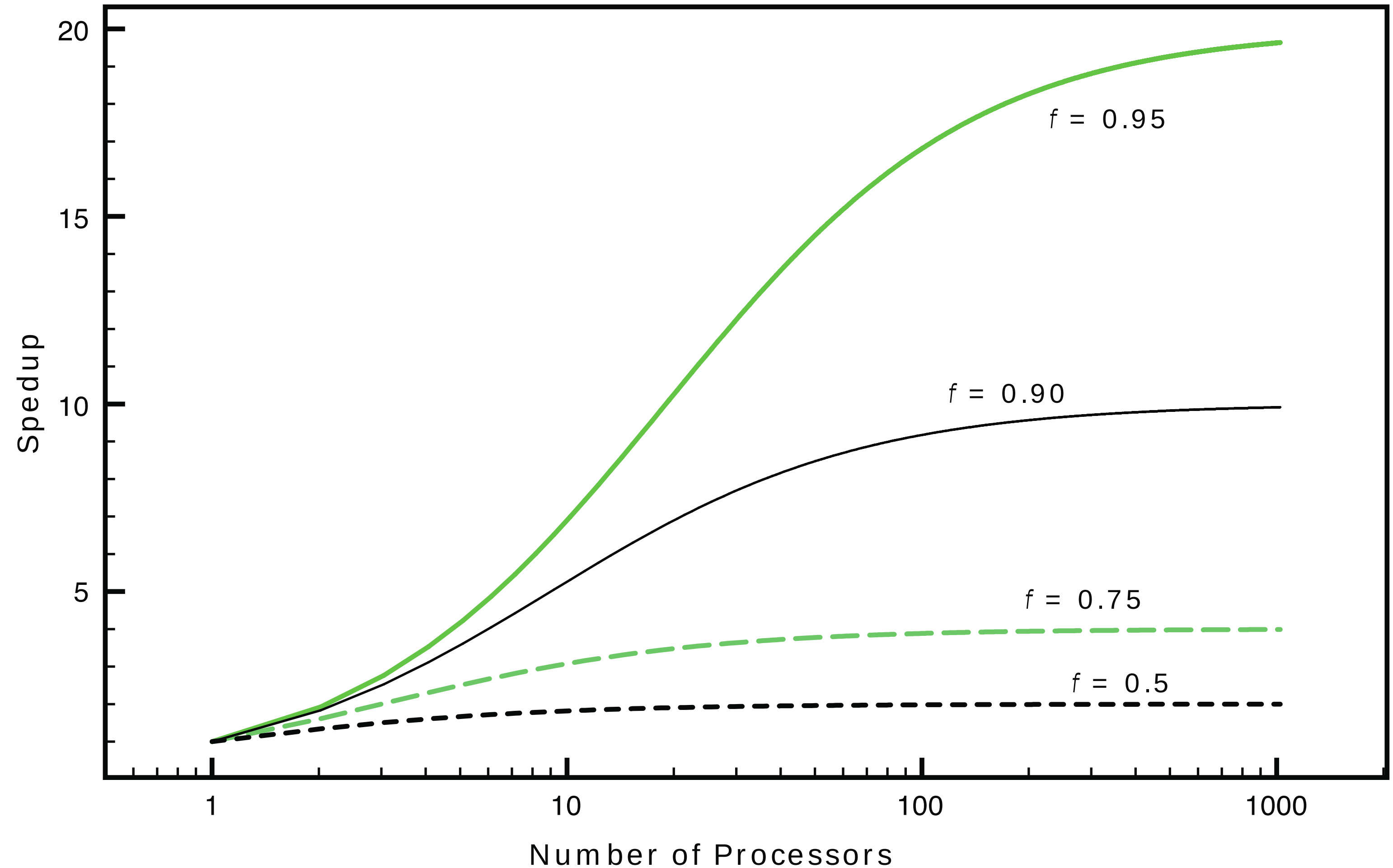
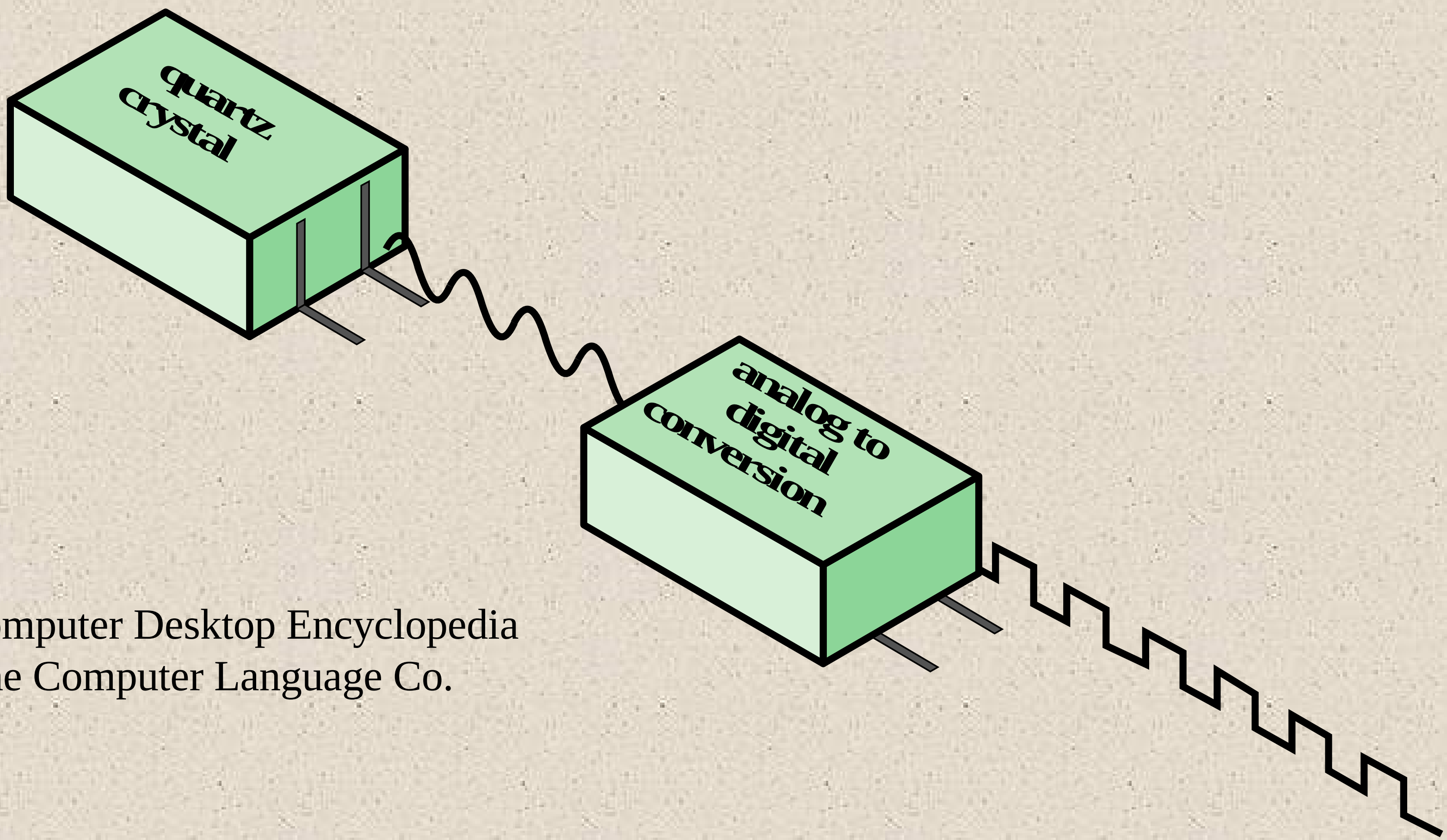


Figure 2.4 Amdahl's Law for Multiprocessors

+

Little's Law

- Fundamental and simple relation with broad applications
- Can be applied to almost any system that is statistically in steady state, and in which there is no leakage
- Queuing system
  - If server is idle an item is served immediately, otherwise an arriving item joins a queue
  - There can be a single queue for a single server or for multiple servers, or multiple queues with one being for each of multiple servers
- Average number of items in a queuing system equals the average rate at which items arrive multiplied by the time that an item spends in the system
  - Relationship requires very few assumptions
  - Because of its simplicity and generality it is extremely useful



From Computer Desktop Encyclopedia
1998, The Computer Language Co.

Figure 2.5 System Clock

	I_c	p	m	k	t
Instruction set architecture	X	X			
Compiler technology	X	X	X		
Processor implementation		X			X
Cache and memory hierarchy				X	X

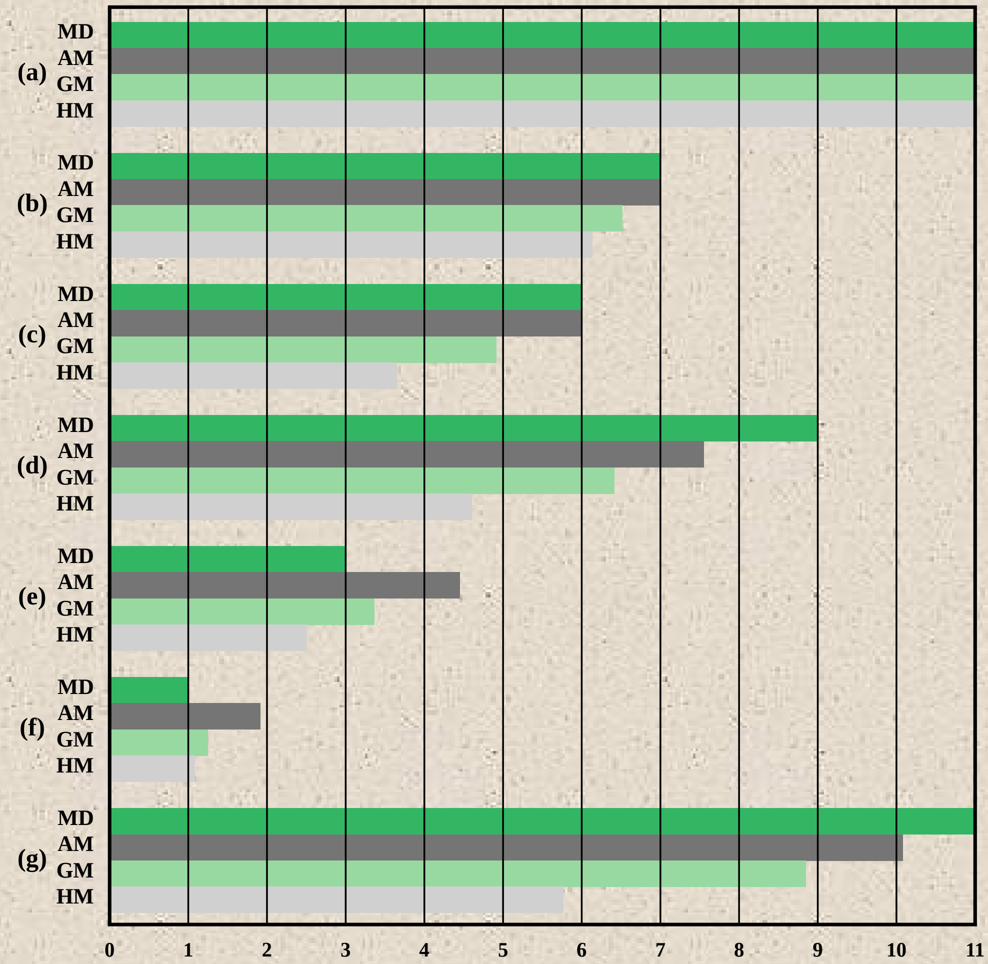
Table 2.1 Performance Factors and System Attributes

Calculating the Mean

The use of benchmarks to compare systems involves calculating the mean value of a set of data points related to execution time

The three common formulas used for calculating a mean are:

- Arithmetic
- Geometric
- Harmonic



(a) Constant (11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11)
(b) Clustered around a central value (3, 5, 6, 6, 7, 7, 7, 8, 8, 9, 1 1)
(c) Uniform distribution (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1 1)
(d) Large-number bias (1, 4, 4, 7, 7, 9, 9, 10, 10, 1 1, 11)
(e) Small-number bias(1, 1, 2, 2, 3, 3, 5, 5, 8, 8, 1 1)
(f) Upper outlier (11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
(g) Lower outlier (1, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11)

MD = median
AM = arithmetic mean
GM = geometric mean
HM = harmonic mean

Figure 2.6 Comparison of Means on Various Data Sets
(each set has a maximum data point value of 1 1)

- An Arithmetic Mean (AM) is an appropriate measure if the sum of all the measurements is a meaningful and interesting value

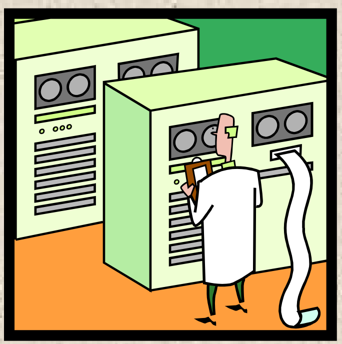
- The AM is a good candidate for comparing the execution time performance of several systems

For example, suppose we were interested in using a system for large-scale simulation studies and wanted to evaluate several alternative products. On each system we could run the simulation multiple times with different input values for each run, and then take the average execution time across all runs. The use of multiple runs with different inputs should ensure that the results are not heavily biased by some unusual feature of a given input set. The AM of all the runs is a good measure of the system's performance on simulations, and a good number to use for system comparison.

Arithmetic Mean

+

- The AM used for a time-based variable, such as program execution time, has the important property that it is directly proportional to the total time
  - If the total time doubles, the mean value doubles



	Computer A time (secs)	Computer B time (secs)	Computer C time (secs)	Computer A rate (MFLOPS)	Computer B rate (MFLOPS)	Computer C rate (MFLOPS)
Program 1 (10^8 FP ops)	2.0	1.0	0.75	50	100	133.33
Program 2 (10^8 FP ops)	0.75	2.0	4.0	133.33	50	25
Total execution time	2.75	3.0	4.75			
Arithmetic mean of times	1.38	1.5	2.38			
Inverse of total execution time (1/sec)	0.36	0.33	0.21			
Arithmetic mean of rates				91.67	75.00	79.17
Harmonic mean of rates				72.72	66.67	42.11

Table 2.2

A Comparison of Arithmetic and Harmonic Means for Rates

Table 2.3 A Comparison of Arithmetic and Geometric Means for Normalized Results

(a) Results normalized to Computer A

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (1.0)	1.0 (0.5)	0.75 (0.38)
Program 2	0.75 (1.0)	2.0 (2.67)	4.0 (5.33)
Total execution time	2.75	3.0	4.75
Arithmetic mean of normalized times	1.00	1.58	2.85
Geometric mean of normalized times	1.00	1.15	1.41

(b) Results normalized to Computer B

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (2.0)	1.0 (1.0)	0.75 (0.75)
Program 2	0.75 (0.38)	2.0 (1.0)	4.0 (2.0)
Total execution time	2.75	3.0	4.75
Arithmetic mean of normalized times	1.19	1.00	1.38
Geometric mean of normalized times	0.87	1.00	1.22

Table 2.4 Another Comparison of Arithmetic and Geometric Means for Normalized Results

(a) Results normalized to Computer A

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (1.0)	1.0 (0.5)	0.20 (0.1)
Program 2	0.4 (1.0)	2.0 (5.0)	4.0 (10)
Total execution time	2.4	3.00	4.2
Arithmetic mean of normalized times	1.00	2.75	5.05
Geometric mean of normalized times	1.00	1.58	1.00

(b) Results normalized to Computer B

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (2.0)	1.0 (1.0)	0.20 (0.2)
Program 2	0.4 (0.2)	2.0 (1.0)	4.0 (2)
Total execution time	2.4	3.00	4.2
Arithmetic mean of normalized times	1.10	1.00	1.10
Geometric mean of normalized times	0.63	1.00	0.63

Benchmark Principles

- Desirable characteristics of a benchmark program:

  1. It is written in a high-level language, making it portable across different machines
  2. It is representative of a particular kind of programming domain or paradigm, such as systems programming, numerical programming, or commercial programming
  3. It can be measured easily
  4. It has wide distribution

+

System Performance Evaluation Corporation (SPEC)

- Benchmark suite
  - A collection of programs, defined in a high-level language
  - Together attempt to provide a representative test of a computer in a particular application or system programming area
- SPEC
  - An industry consortium
  - Defines and maintains the best known collection of benchmark suites aimed at evaluating computer systems
  - Performance measurements are widely used for comparison and research purposes

+

SPEC CPU2006

- Best known SPEC benchmark suite
- Industry standard suite for processor intensive applications
- Appropriate for measuring performance for applications that spend most of their time doing computation rather than I/O
- Consists of 17 floating point programs written in C, C++, and Fortran and 12 integer programs written in C and C++
- Suite contains over 3 million lines of code
- Fifth generation of processor intensive suites from SPEC

Benchmark	Reference time (hours)	Instr count (billion)	Language	Application Area	Brief Description
400.perlbench	2.71	2,378	C	Programming Language	PERL programming language interpreter, applied to a set of three programs.
401.bzip2	2.68	2,472	C	Compression	General-purpose data compression with most work done in memory, rather than doing I/O.
403.gcc	2.24	1,064	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron.
429.mcf	2.53	327	C	Combinatorial Optimization	Vehicle scheduling algorithm.
445.gobmk	2.91	1,603	C	Artificial Intelligence	Plays the game of Go, a simply described but deeply complex game.
456.hmmer	2.59	3,363	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models.
458.sjeng	3.36	2,383	C	Artificial Intelligence	A highly ranked chess program that also plays several chess variants.
462.libquantum	5.76	3,555	C	Physics / Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
464.h264ref	6.15	3,731	C	Video Compression	H.264/AVC (Advanced Video Coding) Video compression.
471.omnetpp	1.74	687	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
473.astar	1.95	1,200	C++	Path-finding Algorithms	Pathfinding library for 2D maps.
483.xalancbmk	1.92	1,184	C++	XML Processing	A modified version of Xalan-C++, which transforms XML documents to other document types.

Table 2.5

SPEC CPU2006 Integer Benchmarks

(Table can be found on page 69 in the textbook.)

Benchmark	Reference time (hours)	Instr count (billion)	Language	Application Area	Brief Description
410.bwaves	3.78	1,176	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow.
416.gamess	5.44	5,189	Fortran	Quantum Chemistry	Quantum chemical computations.
433.milc	2.55	937	C	Physics / Quantum Chromodynamics	Simulates behavior of quarks and gluons
434.zeusmp	2.53	1,566	Fortran	Physics / CFD	Computational fluid dynamics simulation of astrophysical phenomena.
435.gromacs	1.98	1,958	C, Fortran	Biochemistry / Molecular Dynamics	Simulate Newtonian equations of motion for hundreds to millions of particles.
436.cactusADM	3.32	1,376	C, Fortran	Physics / General Relativity	Solves the Einstein evolution equations.
437.leslie3d	2.61	1,273	Fortran	Fluid Dynamics	Model fuel injection flows.
444.namd	2.23	2,483	C++	Biology / Molecular Dynamics	Simulates large biomolecular systems.
447.dealII	3.18	2,323	C++	Finite Element Analysis	Program library targeted at adaptive finite elements and error estimation.
450.soplex	2.32	703	C++	Linear Programming, Optimization	Test cases include railroad planning and military airlift models.
453.povray	1.48	940	C++	Image Ray-tracing	3D Image rendering.
454.calculix	2.29	3,04`	C, Fortran	Structural Mechanics	Finite element code for linear and nonlinear 3D structural applications.
459.GemsFDTD	2.95	1,320	Fortran	Computational Electromagnetics	Solves the Maxwell equations in 3D.
465.tonto	2.73	2,392	Fortran	Quantum Chemistry	Quantum chemistry package, adapted for crystallographic tasks.
470.lbm	3.82	1,500	C	Fluid Dynamics	Simulates incompressible fluids in 3D.
481.wrf	3.10	1,684	C, Fortran	Weather	Weather forecasting model
482.sphinx3	5.41	2,472	C	Speech recognition	Speech recognition software.

Table 2.6

SPEC CPU2006 Floating-Point Benchmarks

(Table can be found on page 70 in the textbook.)

Terms Used in SPEC Documentation

- Benchmark
  - A program written in a high-level language that can be compiled and executed on any computer that implements the compiler
- System under test
  - This is the system to be evaluated
- Reference machine
  - This is a system used by SPEC to establish a baseline performance for all benchmarks
    - Each benchmark is run and measured on this machine to establish a reference time for that benchmark
- Base metric
  - These are required for all reported results and have strict guidelines for compilation
- Peak metric
  - This enables users to attempt to optimize system performance by optimizing the compiler output
- Speed metric
  - This is simply a measurement of the time it takes to execute a compiled benchmark
    - Used for comparing the ability of a computer to complete single tasks
- Rate metric
  - This is a measurement of how many tasks a computer can accomplish in a certain amount of time
    - This is called a throughput, capacity, or rate measure
    - Allows the system under test to execute simultaneous tasks to take advantage of multiple processors

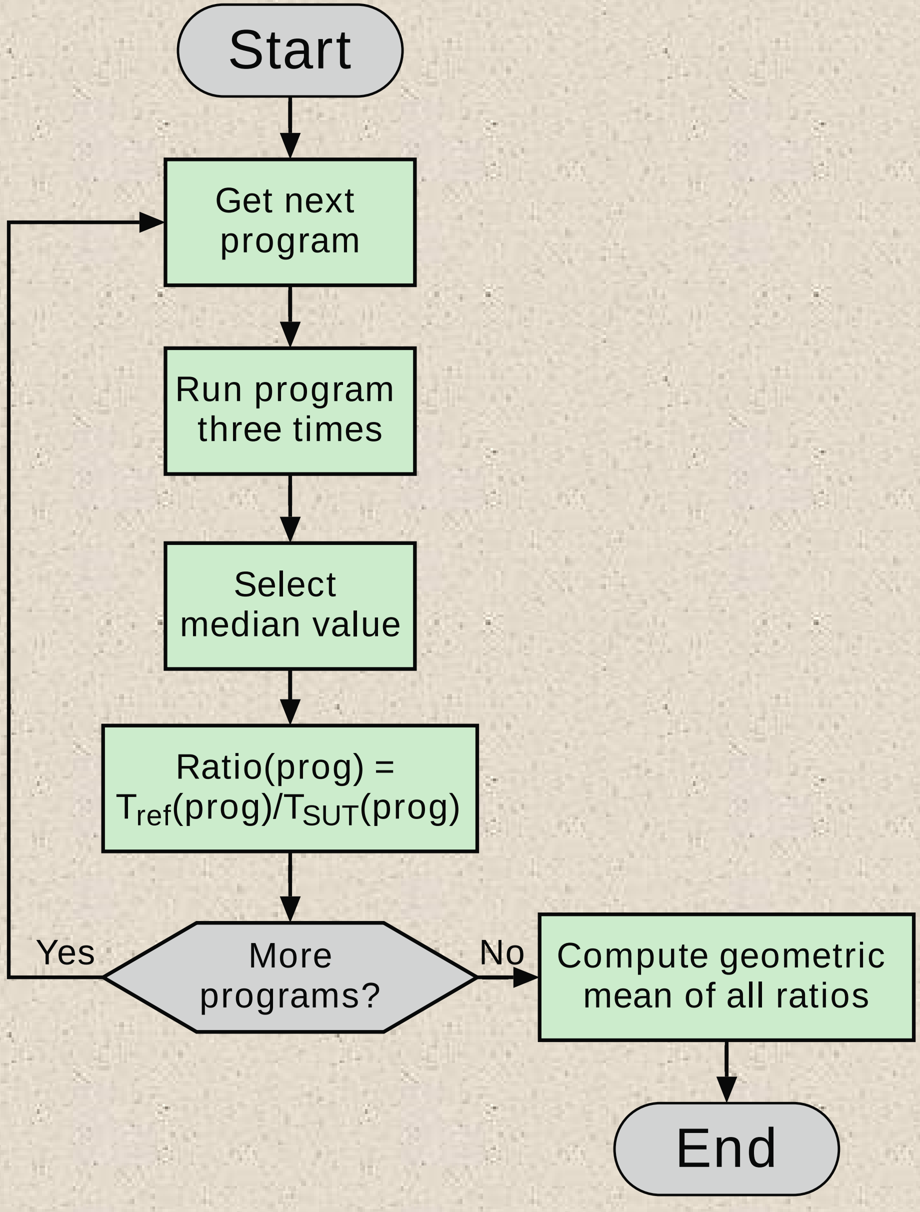


Figure 2.7 SPEC Evaluation Flowchart

Table 2.7 Some SPEC CINT2006 Results

(a) Sun Blade 1000

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio
400.perlbench	3077	3076	3080	9770	3.18
401.bzip2	3260	3263	3260	9650	2.96
403.gcc	2711	2701	2702	8050	2.98
429.mcf	2356	2331	2301	9120	3.91
445.gobmk	3319	3310	3308	10490	3.17
456.hmmer	2586	2587	2601	9330	3.61
458.sjeng	3452	3449	3449	12100	3.51
462.libquantum	10318	10319	10273	20720	2.01
464.h264ref	5246	5290	5259	22130	4.21
471.omnetpp	2565	2572	2582	6250	2.43
473.astar	2522	2554	2565	7020	2.75
483.xalancbmk	2014	2018	2018	6900	3.42

(b) Sun Blade X6250

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio	Rate
400.perlbench	497	497	497	9770	19.66	78.63
401.bzip2	613	614	613	9650	15.74	62.97
403.gcc	529	529	529	8050	15.22	60.87
429.mcf	472	472	473	9120	19.32	77.29
445.gobmk	637	637	637	10490	16.47	65.87
456.hmmer	446	446	446	9330	20.92	83.68
458.sjeng	631	632	630	12100	19.18	76.70
462.libquantum	614	614	614	20720	33.75	134.98
464.h264ref	830	830	830	22130	26.66	106.65
471.omnetpp	619	620	619	6250	10.10	40.39
473.astar	580	580	580	7020	12.10	48.41
483.xalancbmk	422	422	422	6900	16.35	65.40

+ Summary

Chapter 2

- Designing for performance
  - Microprocessor speed
  - Performance balance
  - Improvements in chip organization and architecture
- Multicore
- MICs
- GPGPUs
- Amdahl's Law
- Little's Law

Performance Issues

- Basic measures of computer performance
  - Clock speed
  - Instruction execution rate
- Calculating the mean
  - Arithmetic mean
  - Harmonic mean
  - Geometric mean
- Benchmark principles
- SPEC benchmarks