

# **Operating Systems and Computer Hardware**

**Prof. Dr. Hasan Hüseyin BALIK  
(2<sup>nd</sup> Week)**

# Outline

## 1. Overview

- Basic Concepts and Computer Evolution
- Performance Issues



## + 1.2 Performance Issues

## 1.2 Outline


- Designing for Performance
- Multicore, MICs, and GPGPUs
- Two Laws that Provide Insight: Ahmdahl's Law and Little's Law
- Basic Measures of Computer Performance
- Calculating the Mean
- Benchmarks and SPEC

# 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
- Workstation systems now support highly sophisticated engineering and scientific applications and have the capacity to support image and video applications.
- 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

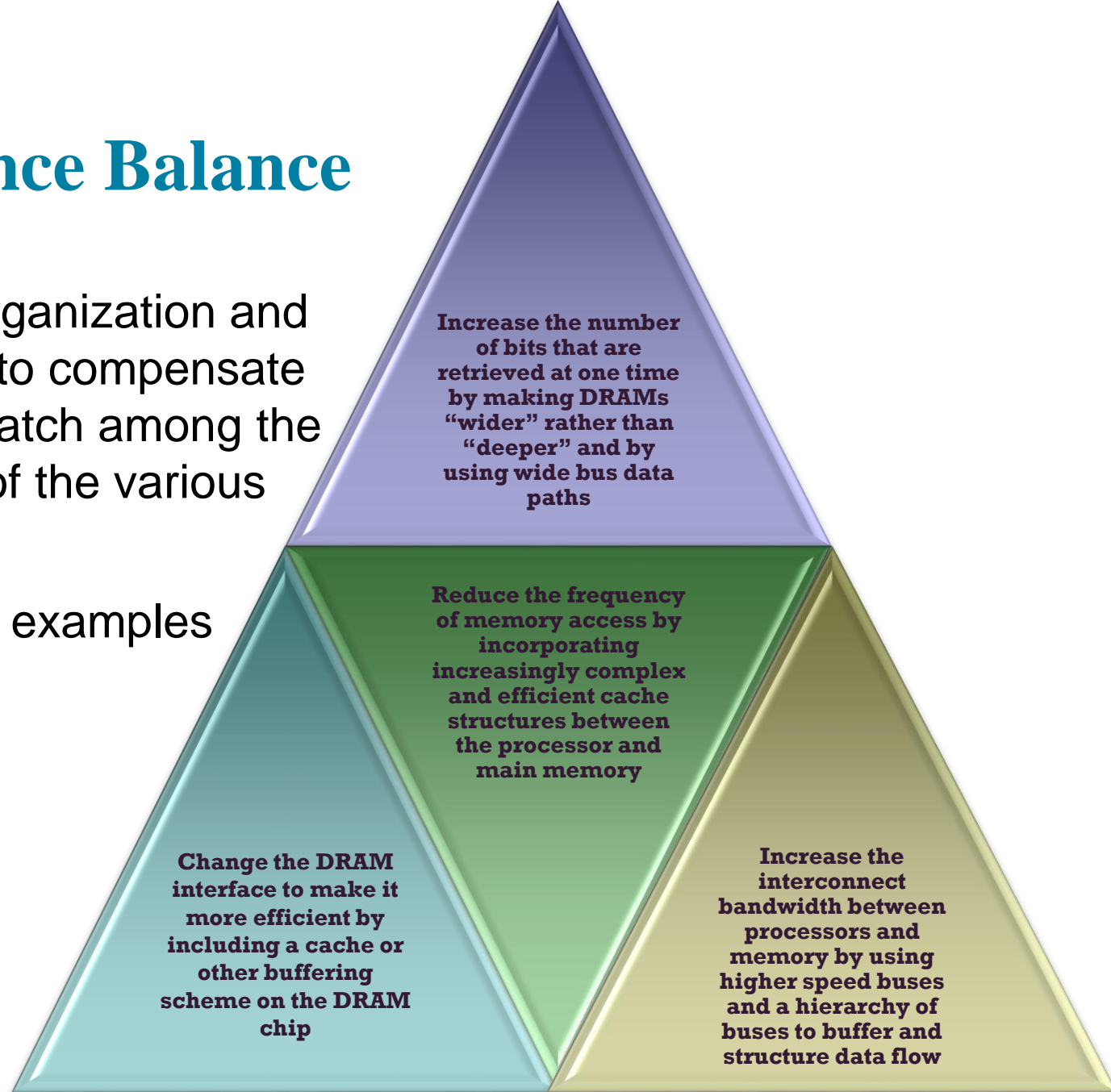
Techniques built into contemporary processors include:

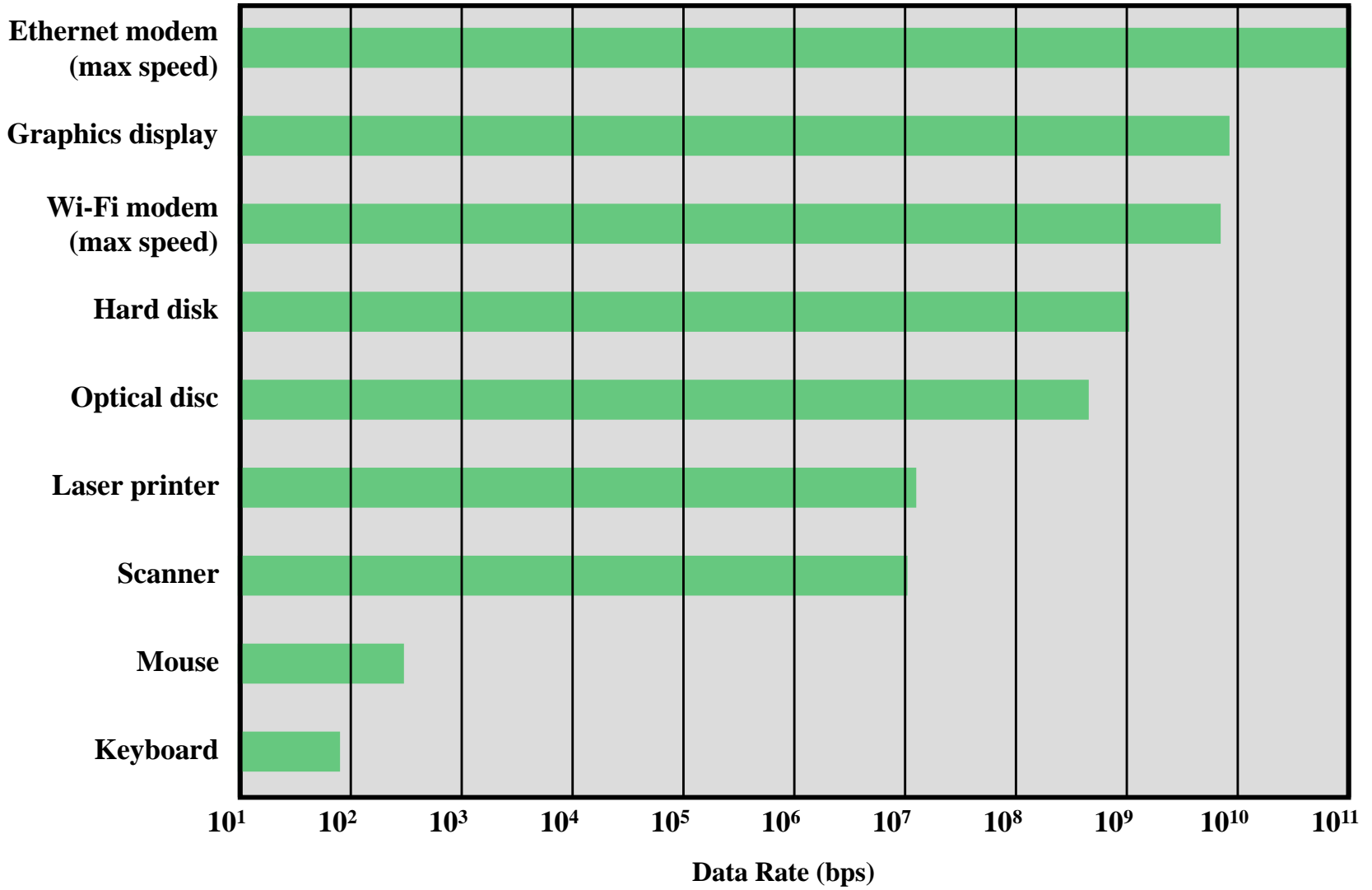


<b>Pipelining</b>	<ul style="list-style-type: none"><li>• Processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously</li></ul>
<b>Branch prediction</b>	<ul style="list-style-type: none"><li>• Processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next</li></ul>
<b>Superscalar execution</b>	<ul style="list-style-type: none"><li>• This is the ability to issue more than one instruction in every processor clock cycle. (In effect, multiple parallel pipelines are used.)</li></ul>
<b>Data flow analysis</b>	<ul style="list-style-type: none"><li>• Processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions</li></ul>
<b>Speculative execution</b>	<ul style="list-style-type: none"><li>• 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</li></ul>

# 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 Logic 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 and throughput
  - Memory access speed (latency) and transfer speed (throughput) lag processor speeds

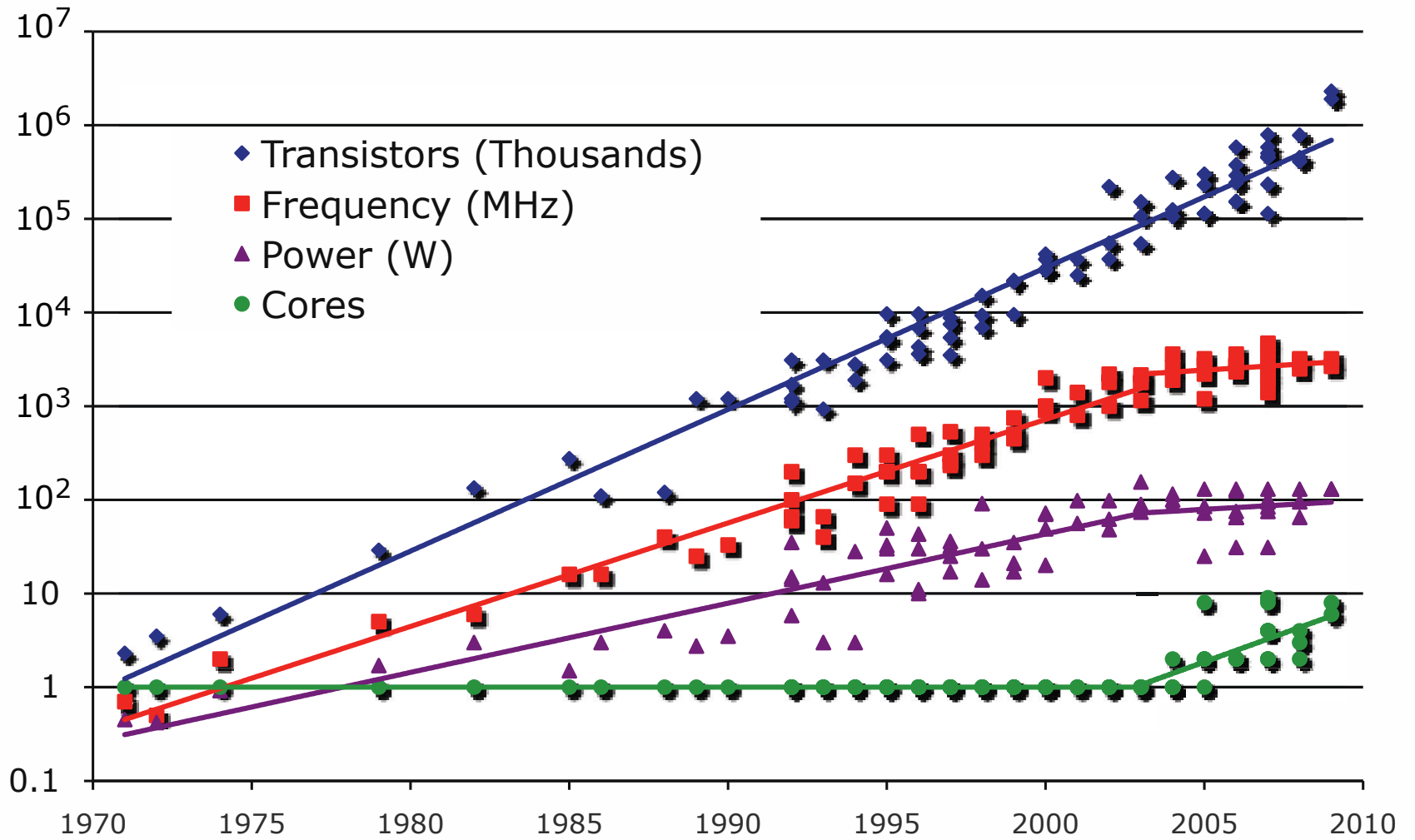


Figure 2.2 Processor Trends

# 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

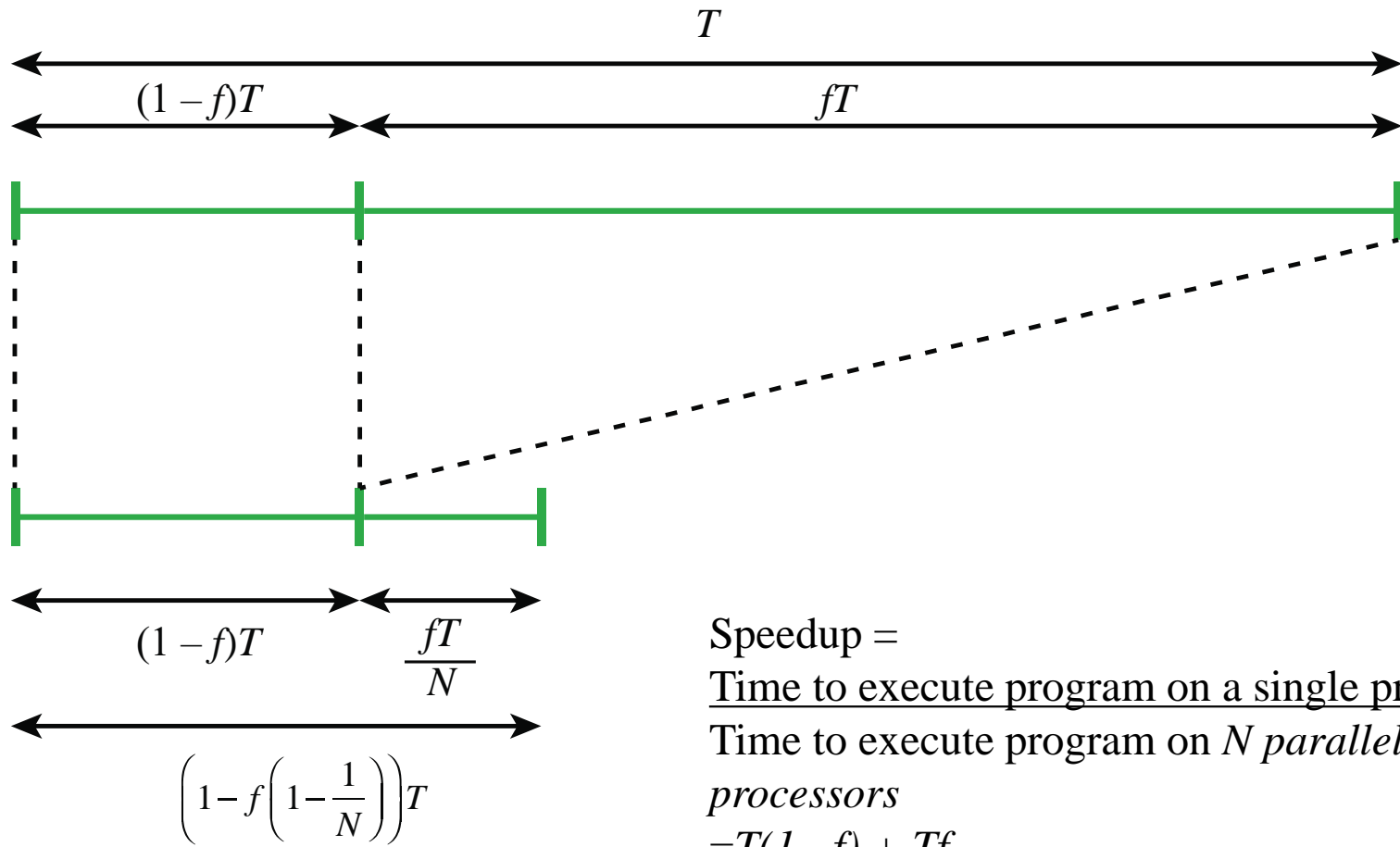
- Leap in performance as well as the challenges in developing software to exploit such a large number of cores
- The multicore and MIC strategy involves a homogeneous collection of general purpose processors on a single chip

## 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

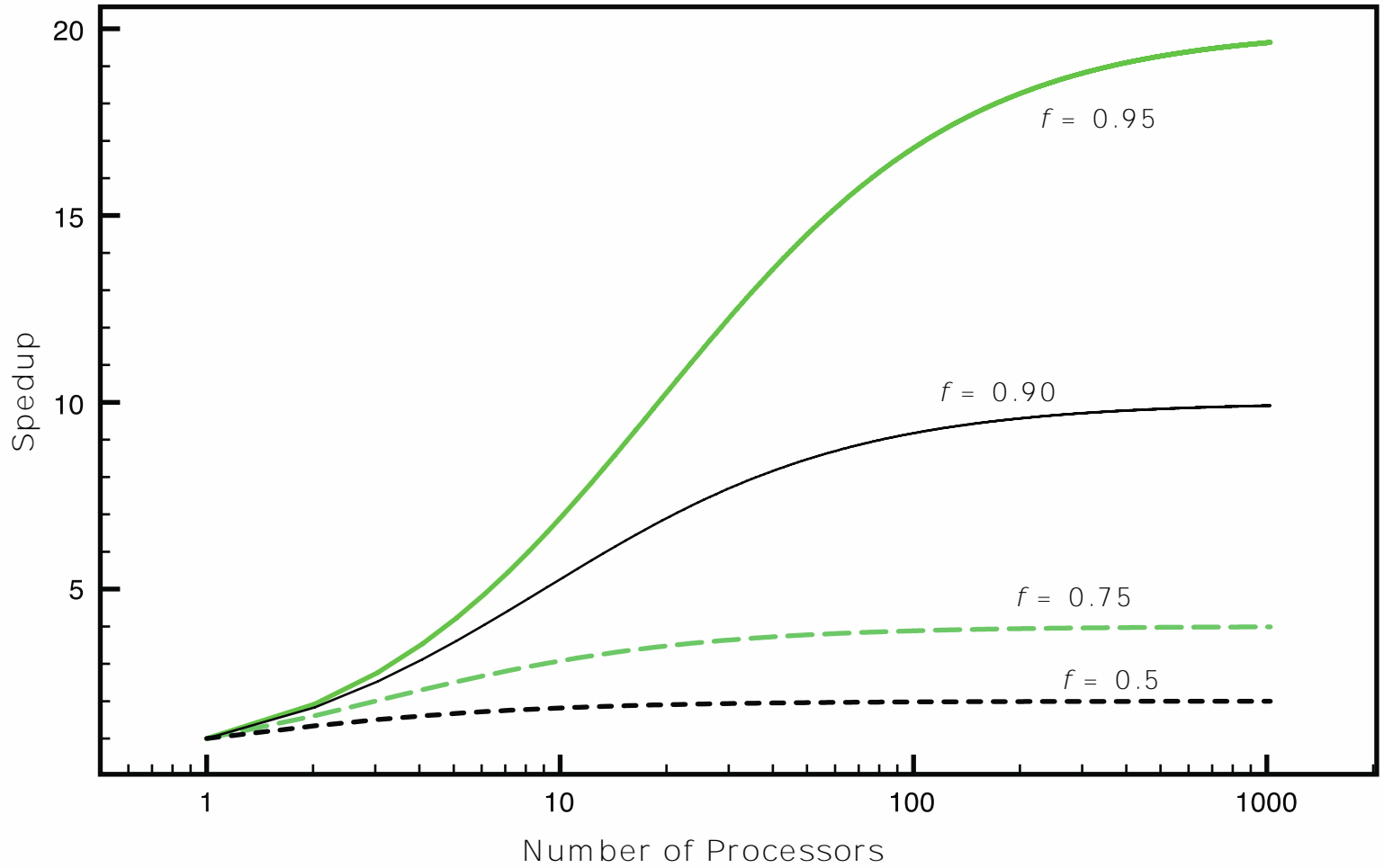
- Amdahl's law was first proposed by Gene Amdahl in 1967
- Deals with the potential speedup of a program using multiple processors compared to a single processor
- Illustrates the problems facing industry in the development of multi-core machines
  - Software must be adapted to a highly parallel execution environment to exploit the power of parallel processing
- Can be generalized to evaluate and design technical improvement in a computer system



$$\begin{aligned} \text{Speedup} &= \frac{\text{Time to execute program on a single processor}}{\text{Time to execute program on } N \text{ parallel processors}} \\ &= \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} \end{aligned}$$

**Figure 2.3 Illustration of Amdahl's Law**

$T$  is the total execution time of the program using a single processor  
 $f$  is a fraction of the execution time involves code that is infinitely parallelizable with no scheduling overhead

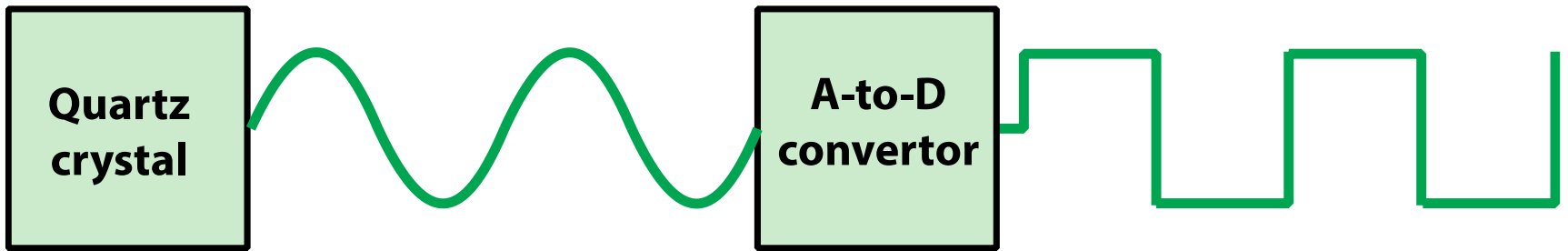


**Figure 2.4 Amdahl's Law** for Multiprocessors



# 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



**Figure 2.5 System Clock**

# Performance Factors and System Attributes

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

$I_c$  : Instruction Count

$p$  : The number of processor cycles needed to decode and execute the instruction

$m$ : The number of memory references needed

$k$ : the ratio between memory cycle time and processor cycle time

$\tau$  : cycle time (1/f)

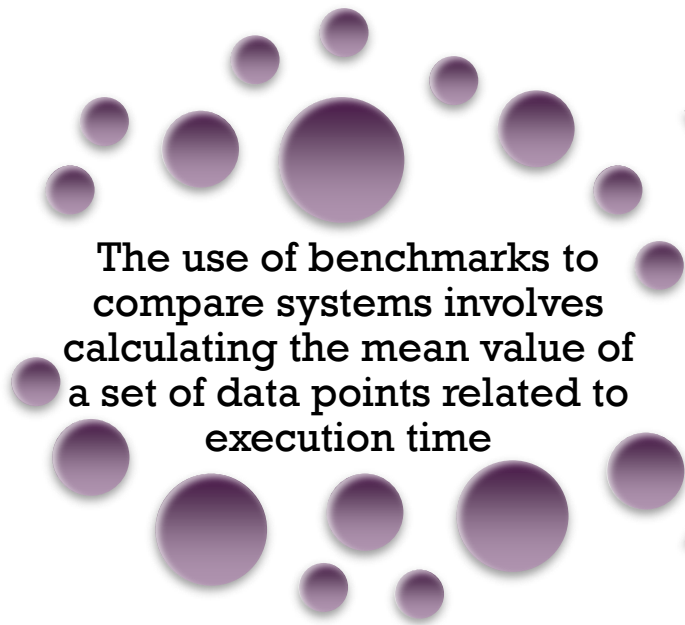
CPI: Clock cycle per instruction

T: The time needed to execute a given programme

$$CPI = \frac{\sum_{i=1}^n (CPI_i \times I_i)}{I_c}$$

$$T = I_c \times CPI \times \tau$$

# Calculating the Mean

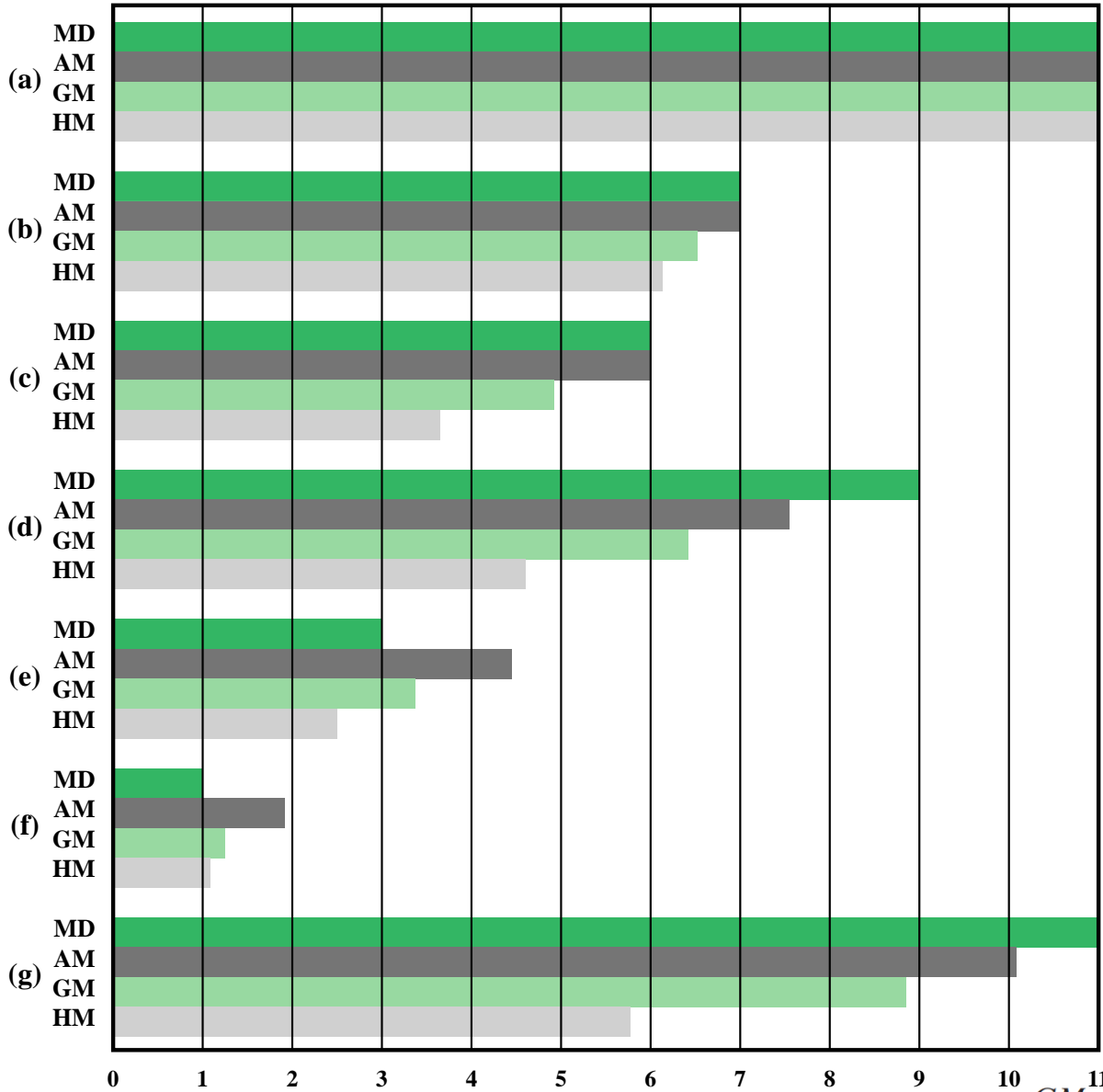


The use of benchmarks to compare systems involves calculating the mean value of a set of data points related to execution time



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

$$AM = \frac{x_1 + \dots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$GM = \sqrt[n]{x_1 \times \dots \times x_n} = \left( \prod_{i=1}^n x_i \right)^{1/n} = e^{\left( \frac{1}{n} \sum_{i=1}^n \ln(x_i) \right)}$$

$$HM = \frac{n}{\left( \frac{1}{x_1} \right) + \dots + \left( \frac{1}{x_n} \right)} = \frac{n}{\sum_{i=1}^n \left( \frac{1}{x_i} \right)} \quad x_i > 0$$

# Arithmetic Mean

- 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.

- 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

# A Comparison of Arithmetic and Harmonic Means for Rates

	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 <sup>8</sup> FP ops)	2.0	1.0	0.75	50	100	133.33
Program 2 (10 <sup>8</sup> 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

# A Comparison of Arithmetic and Geometric Means for Normalized Results

**(a) Results normalized to Computer A**

	<b>Computer A time</b>	<b>Computer B time</b>	<b>Computer C time</b>
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

**(a) Results normalized to Computer B**

	<b>Computer A time</b>	<b>Computer B time</b>	<b>Computer C time</b>
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



# Another Comparison of Arithmetic and Geometric Means for Normalized Results

**(a) Results normalized to Computer A**

	<b>Computer A time</b>	<b>Computer B time</b>	<b>Computer C time</b>
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 (10.0)
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

**(a) Results normalized to Computer B**

	<b>Computer A time</b>	<b>Computer B time</b>	<b>Computer C time</b>
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 (2.0)
Total execution time	2.4	3.0	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 CPU2017

- 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 20 integer benchmarks and 23 floating-point benchmarks written in C, C++, and Fortran
- For all of the integer benchmarks and most of the floating-point benchmarks, there are both rate and speed benchmark programs
- The suite contains over 11 million lines of code

Rate	Speed	Language	Kloc	Application Area
500.perlbench_r	600.perlbench_s	C	363	Perl interpreter
502.gcc_r	602.gcc_s	C	1304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	AI: alpha-beta tree search (chess)
541.leela_r	641.leela_s	C++	21	AI: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	AI: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

(A)

**SPEC  
CPU2017  
Benchmarks**

Kloc = line count (including comments/whitespace) for source files used in a build/1000

Rate	Speed	Language	Kloc	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics; relativity
508.namd_r		C++, C	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging; optical tomography with finite elements
511.povray_r		C++	170	Ray tracing
519.ibm_r	619.ibm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++	1577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling.

(B)

**SPEC  
CPU2017  
Benchmarks**

Kloc = line count (including comments/whitespace) for source files used in a build/1000

Benchmark	Base		Peak	
	Seconds	Rate	Seconds	Rate
500.perlbench_r	1141	1070	933	1310
502.gcc_r	1303	835	1276	852
505.mcf_r	1433	866	1378	901
520.omnetpp_r	1664	606	1634	617
523.xalancbmk_r	722	1120	713	1140
525.x264_r	655	2053	661	2030
531.deepsjeng_r	604	1460	597	1470
541.leela_r	892	1410	896	1420
548.exchange2_r	833	2420	770	2610
557.xz_r	870	953	863	961

**SPEC  
CPU 2017  
Integer  
Benchmarks  
for HP  
Integrity  
Superdome X**

**(a) Rate Result  
(768 copies)**

Benchmark	Base		Peak	
	Seconds	Ratio	Seconds	Ratio
600.perlbench_s	358	4.96	295	6.01
602.gcc_s	546	7.29	535	7.45
605.mcf_s	866	5.45	700	6.75
620.omnetpp_s	276	5.90	247	6.61
623.xalancbmk_s	188	7.52	179	7.91
625.x264_s	283	6.23	271	6.51
631.deepsjeng_s	407	3.52	343	4.18
641.leela_s	469	3.63	439	3.88
648.exchange2_s	329	8.93	299	9.82
657.xz_s	2164	2.86	2119	2.92

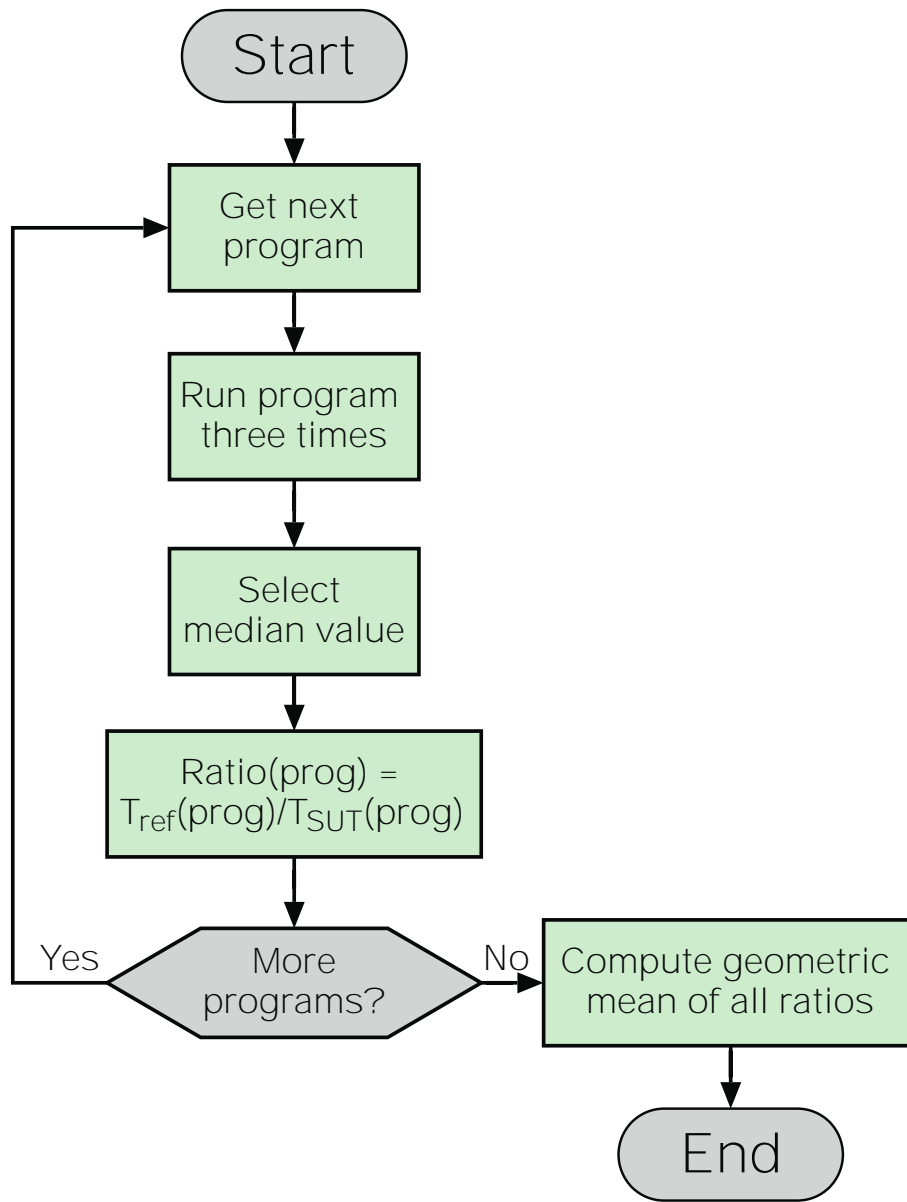
**SPEC  
CPU 2017  
Integer  
Benchmarks  
for HP  
Integrity  
Superdome X**

**(b) Speed  
Result  
(384 threads)**



# 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**

Benchmark	Seconds	Energy (kJ)	Average Power (W)	Maximum Power (W)
600.perlbench_s	1774	1920	1080	1090
602.gcc_s	3981	4330	1090	1110
605.mcf_s	4721	5150	1090	1120
620.omnetpp_s	1630	1770	1090	1090
623.xalancbmk_s	1417	1540	1090	1090
625.x264_s	1764	1920	1090	1100
631.deepsjeng_s	1432	1560	1090	1130
641.leela_s	1706	1850	1090	1090
648.exchange2_s	2939	3200	1080	1090
657.xz_s	6182	6730	1090	1140

**SPECSpeed  
2017\_int\_base  
Benchmark  
Results for  
Reference  
Machine (1  
thread)**