Computer Architecture
Basic Concepts and Performance

Multicore, Mics, and GPGPUs

Use of multiple processors on the same chip, also referred to as multiple cores or multicore, provides the potential to increase performance without increasing the clock rate.

Within a processor, the increase in performance is roughly proportional to the square root of the increase in complexity.

If the software can support the effective use of multiple processors, then doubling the number of processors almost doubles performance.

Chip manufacturers are now in the process of increasing number of cores per chip, with more than 50 cores per chip. The performance as well as the challenges in developing software to exploit such a large number of cores has led to new term: **many integrated core (MIC).**

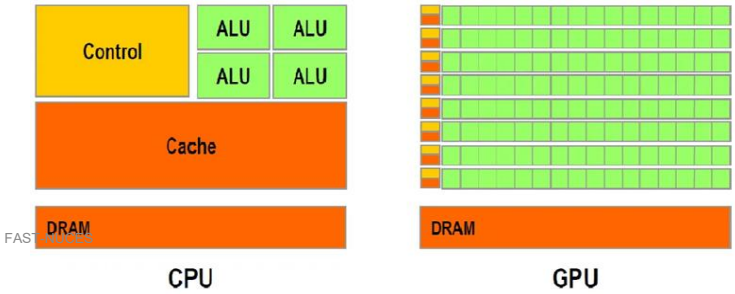
The multicore and MIC strategy involves a homogeneous collection of general-purpose processors on a single chip.

Multicore, Mics, and GPGPUs

GPU: A graphics processing unit (GPU) is a specialized electronic circuit initially designed for digital image processing and to accelerate computer graphics, being present either as a discrete video card or embedded on motherboards, mobile phones, personal computers, workstations, and game consoles.

GPUs were found to be useful for non-graphic calculations involving parallel problems due to their parallel structure. Other non-graphical uses include the training of neural networks.

GPGPU: When a broad range of applications are supported by such a processor, the term general-purpose computing on GPUs (GPGPU) is used.



Quantative Principle of Computer Design

Principle of Locality: programs tend to reuse data and instructions they have used recently.

An implication of locality is that we can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past. The principle of locality also applies to data accesses, though not as strongly as to code accesses.

Two different types of locality have been observed.

Temporal locality states that recently accessed items are likely to be accessed soon.

Spatial locality says that items whose addresses are near one another tend to be referenced close together in time.

Amdahl's Law

Amdahl's Law states that 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.

Amdahl's Law gives us a quick way to find the speedup from some enhancement,which depends on two factors:

1. The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement.

For example, if 40 seconds of the execution time of a program that takes 100 seconds in total can use an enhancement, the fraction is 40/100. This value, which we call **Fractionenhanced**, is always less than or equal to 1.

2. The improvement gained by the enhanced execution mode, that is, how much faster the task would run if the enhanced mode were used for the entire program. This value is the time of the original mode over the time of the enhanced mode.

If the enhanced mode takes, say, 4 seconds for a portion of the program, while it is 40 seconds in the original mode, the improvement is 40/4 or 10. We call this value, which is always greater than 1, **Speedupenhanced**.

Amdahl's Law

The execution time using the original computer with the enhanced mode will be the time spent using the unenhanced portion of the computer plus the time spent using the enhancement:

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

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 Suppose that we want to enhance the processor used for web serving. The new processor is 10 times faster on computation in the web serving application than the old processor. Assuming that the original processor 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?

Amdahl's Law

Example A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FSQRT) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FSQRT 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 half 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.

Little's Law

In mathematical queueing theory, Little's law states that the long-term average number **L** of customers in a stationary system is equal to the long-term average effective arrival rate **λ** multiplied by the average time **W** that a customer spends in the system.

Expressed algebraically the law is: L= λW

Using queuing theory terminology, Little's Law applies to a queuing system. The central element of the system is a server, which provides some service to items.

Items from some population of items arrive at the system to be served. If the server is idle, an item is served immediately. Otherwise, an arriving item joins a waiting line, or queue. There can be a single queue or multiples queues, one for each of multiple servers. When a server has completed serving an item, the item departs.

Little's Law

Problem#2.8: The owner of a shop observes that on average 18 customers per hour arrive and there are typically 8 customers in the shop. What is the average length of time each customer spends in the shop?

Basic Measures of Computer Performance

Clock Speed: The speed of a processor is dictated by the pulse frequency produced by the clock, measured in cycles per second, or Hertz (Hz).

Clock signals are generated by a quartz crystal, which generates a constant sine wave while power is applied. This wave is converted into a digital voltage pulse stream that is provided in a constant flow to the processor circuitry (Figure 2.5).

For example, a 1-GHz processor receives 1 billion pulses per second. The rate of pulses is known as the **clock rate or clock speed**. One increment, or pulse, of the clock is referred to as a clock cycle, or a clock tick. The time between pulses is the **cycle time**.

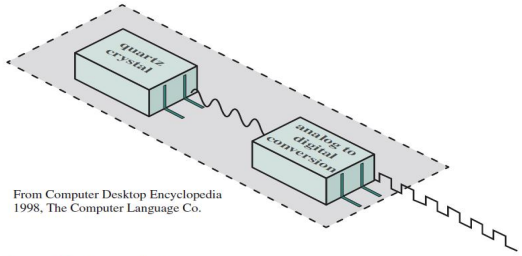


Figure 2.5 System Clock

Basic Measures of Computer Performance

When a signal is placed on a line inside the processor, it takes finite amount of time for the voltage levels to logical 1 or 0 to be available.Operations must be synchronized so that the proper electrical signal (voltage) values are available for each operation.

The execution of an instruction involves a number of discrete steps, such as fetching the instruction from memory, decoding the various portions of the instruction, loading and storing data, and performing arithmetic and logical operations. Most instructions on most processors require multiple clock cycles to complete.

Some instructions may take only a few cycles, while others require dozens. When pipelining is used, multiple instructions are being executed simultaneously. A comparison of clock speeds on different processor can not measure performance.

Basic Measures of Computer Performance

Instruction Exexution Rate: A processor is driven by a clock with a constant **frequency f** or a constant **cycle time** τ, where $\tau = 1/f$.

Define the **instruction count Ic**, for a program as the number of machine instructions executed for that program until it runs to completion or for some defined time interval.

An important parameter is the average cycles per instruction **(CPI) for a program**. If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor.

On any given processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on.

Let **CPIi be the number of cycles required for instruction type i** , and **Ii be the number of executed instructions of type i** for a given program. Then we can calculate an overall CPI as follows:

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{I_c} \qquad (2.2)$$

Basic Measures of Computer Performance

The processor time T needed to execute a given program can be expressed as:

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

We can refine this formulation by recognizing that during the execution of an instruction, part of the work is done by the processor, and part of the time a word is being transferred to or from memory.

Time to transfer depends on the memory cycle time, which may be greater than the processor cycle time. We can rewrite the preceding equation as:

$$T = I_c \times [p + (m \times k)] \times \tau$$

where ,

p is the number of processor cycles needed to decode and execute the instruction,

m is the number of memory references needed

k is the ratio between memory cycle time and processor cycle time.

Basic Measures of Computer Performance

The five performance factors in the preceding equation influenced by four system attributes, shown in Table 2.1.

Table 2.1 Performance Factors and System Attributes

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

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} \qquad (2.3)$$

Floating-point performance is expressed as millions of floating-point operations per second (MFLOPS), defined as follows:

MFLOPS rate= Number of executed floating point operations in a program

Execution Time x (10^6)

Basic Measures of Computer Performance

2.1 A benchmark program is run on a 40 MHz processor. The executed program consists of 100,000 instruction executions, with the following instruction mix and clock cycle count:

Instruction Type	Instruction Count	Cycles per Instruction
Integer arithmetic	45,000	1
Data transfer	32,000	2
Floating point	15,000	2
Control transfer	8000	2

Determine the effective *CPI*, MIPS rate, and execution time for this program.

Calculating the MEAN

In evaluating some aspect of computer system performance, it is often the case that a single number, such as execution time or memory consumed, is used to characterize performance and to compare systems.

A single number can provide only a very simplified view of a system's capability. Specially in the field of benchmarking, single numbers are typically used for performance comparison.

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

Calculating the MEAN

The three common formulas used for calculating a mean are arithmetic, geometric, and harmonic. Given a set of n real numbers $(x_1, x_2, \ldots, x_n)$, the three means are defined as follows:

Arithmetic mean

$$AM = \frac{x_1 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.4}$$

Geometric mean

$$GM = \sqrt[n]{x_1 \times \cdots \times x_n} = \left(\prod_{i=1}^{n} x_i\right)^{1/n} = e^{\left(\frac{1}{n}\sum_{i=1}^{n} \ln(x_i)\right)} \tag{2.5}$$

Harmonic mean

$$HM = \frac{n}{\left(\frac{1}{x_1}\right) + \cdots + \left(\frac{1}{x_n}\right)} = \frac{n}{\sum_{i=1}^{n}\left(\frac{1}{x_i}\right)} \qquad x_i > 0 \tag{2.6}$$

It can be shown that the following inequality holds:

$$AM \leq GM \leq HM$$

The values are equal only if $x_1 = x_2 = \ldots x_n$.

We can get a useful insight into these alternative calculations by defining the functional mean. Let $f(x)$ be a continuous monotonic function defined in the interval $0 \leq y < \infty$. The functional mean with respect to the function $f(x)$ for n positive real numbers $(x_1, x_2, \ldots, x_n)$ is defined as

Functional mean $$FM = f^{-1}\left(\frac{f(x_1) + \cdots + f(x_n)}{n}\right) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i)\right)$$

where $f^{-1}(x)$ is the inverse of $f(x)$. The mean values defined in Equations (2.1) through (2.3) are special cases of the functional mean, as follows:

- AM is the FM with respect to $f(x) = x$
- GM is the FM with respect to $f(x) = \ln x$
- HM is the FM with respect to $f(x) = 1/x$

Calculating the MEAN

EXAMPLE 2.3 Figure 2.6 illustrates the three means applied to various data sets, each of which has eleven data points and a maximum data point value of 11. The median value is also included in the chart. Perhaps what stands out the most in this figure is that the HM has a tendency to produce a misleading result when the data is skewed to larger values or when there is a small-value outlier.

we just note that the conclusions reached depend very much on the examples chosen and the way in which the objectives are stated.

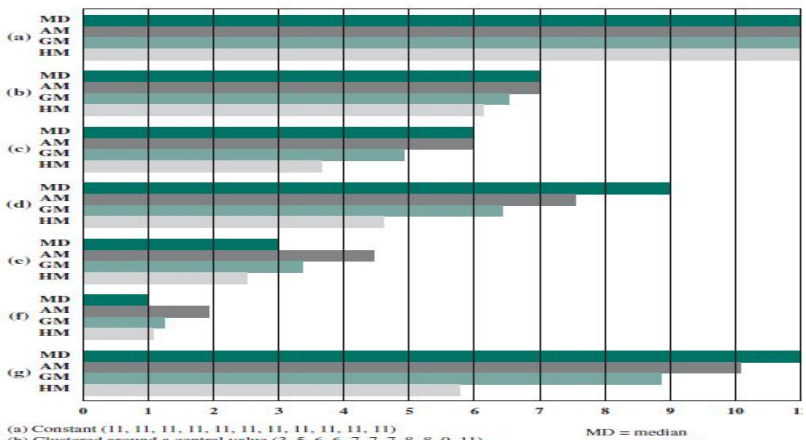


Figure 2.6 Comparison of Means on Various Data Sets (each set has a maximum data point value of 11)

Calculating the MEAN

Arithmetic Mean: AM is an appropriate measure if the sum of all the measurements is a meaningful and interesting value. 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.

AM used for a time-based variable (e.g., seconds), such as program execution time, has the important property that it is directly proportional to the total time. So, if the total time doubles, the mean value doubles

Calculating the MEAN

Harmonic Mean: A system's execution rate may be viewed as a more useful measure of the value of the system. This could be either the instruction execution rate, measured in MIPS or MFLOPS, or a program execution rate, which measures the rate at which a given type of program can be executed .

Let us look at a basic example and first examine how the AM performs. Suppose we have a set of n benchmark programs and record the execution times of each program on a given system as $t_1, t_2, \ldots, t_n$. For simplicity, let us assume that each program executes the same number of operations Z; we could weight the individual programs and calculate accordingly but this would not change the conclusion of our argument. The execution rate for each individual program is $R_i = Z/t_i$. We use the AM to calculate the average execution rate.

$$AM = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\sum_{i=1}^{n} \frac{Z}{t_i} = \frac{Z}{n}\sum_{i=1}^{n} \frac{1}{t_i}$$

We see that the AM execution rate is proportional to the sum of the inverse execution times, which is not the same as being inversely proportional to the sum of the execution times. Thus, the AM does not have the desired property.

The HM yields the following result.

$$HM = \frac{n}{\sum_{i=1}^{n}\left(\frac{1}{R_i}\right)} = \frac{n}{\sum_{i=1}^{n}\left(\frac{1}{Z/t_i}\right)} = \frac{nZ}{\sum_{i=1}^{n} t_i}$$

The HM is inversely proportional to the total execution time, which is the desired property.

Calculating the MEAN

Example: Calculate AM, total execution time, inverse of execution time and Harmonic Mean.

Table 2.2 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 (10^8 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	–	–	–

Calculating the MEAN

1. A customer or researcher may be interested not only in the overall average performance but also performance against different types of benchmark programs, such as business applications, scientific modeling, multimedia applications, and systems programs. Thus, a breakdown by type of benchmark is needed as well as a total.
2. Usually, the different programs used for evaluation are weighted differently. In Table 2.2, it is assumed that the two test programs execute the same number of operations. If that is not the case, we may want to weight accordingly. Or different programs could be weighted differently to reflect importance or priority.

Let us see what the result is if test programs are weighted proportional to the number of operations. Following the preceding notation, each program i executes Zi instructions in a time ti. Each rate is weighted by the instructions count. The weighted HM is therefore:

$$WHM = \frac{1}{\sum_{i=1}^{n}\left(\left(\frac{Z_i}{\sum_{j=1}^{n} Z_j}\right)\left(\frac{1}{R_i}\right)\right)} = \frac{n}{\sum_{i=1}^{n}\left(\left(\frac{Z_i}{\sum_{j=1}^{n} Z_j}\right)\left(\frac{t_i}{Z_i}\right)\right)} = \frac{\sum_{j=1}^{n} Z_j}{\sum_{i=1}^{n} t_i} \quad (2.7)$$

We see that the weighted HM is the quotient of the sum of the operation count divided by the sum of the execution times.

Calculating the MEAN

Geometric Mean: GM gives equal weight to all of the values in the data set.For example, suppose the set of data values to be averaged includes a few large values and more small values.

AM is dominated by the large values. A change of 10% in the largest value will have a noticeable effect, while a change in the smallest value by the same factor will have a negligible effect. A change in value by 10% of any of the data values results in the same change in the GM.

Calculating the MEAN

A second observation is that for the GM of a ratio, the GM of the ratios equals the ratio of the GMs:

$$GM = \left(\prod_{i=1}^{n} \frac{Z_i}{t_i}\right)^{1/n} = \frac{\left(\prod_{i=1}^{n} Z_i\right)^{1/n}}{\left(\prod_{i=1}^{n} t_i\right)^{1/n}} \qquad (2.8)$$

There may be cases where the AM of one data set is larger than that of another set,but the GM is smaller.

EXAMPLE 2.6 In Figure 2.6, the AM for data set d is larger than the AM for data set c, but the opposite is true for the GM.

	Data set c	Data set d
Arithmetic mean	7.00	7.55
Geometric mean	6.68	6.42

One property of the GM that has made it appealing for benchmark analysis is that it provides consistent results when measuring the relative performance of machines. This is in fact what benchmarks are primarily used for: to compare one machine with another in terms of performance metrics. The results, as we have seen, are expressed in terms of values that are normalized to a reference machine.

Calculating the MEAN

2.4 Four benchmark programs are executed on three computers with the following results:

	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Program 3	500	1000	50
Program 4	100	800	100

The table shows the execution time in seconds, with 100,000,000 instructions executed in each of the four programs. Calculate the MIPS values for each computer for each program. Then calculate the arithmetic and harmonic means assuming equal weights for the four programs, and rank the computers based on arithmetic mean and harmonic mean.