1.6 Performance

Assessing the performance of computers can be quite challenging. The scale and intricacy of modern software systems, together with the wide range of performance improvement techniques employed by hardware designers, have made performance assessment much more difficult.

When trying to choose among different computers, performance is an important attribute. Accurately measuring and comparing different computers is critical to purchasers and therefore, to designers. The people selling computers know this as well. Often, salespeople would like you to see their computer in the best possible light, whether or not this light accurately reflects the needs of the purchaser's application. Hence, understanding how best to measure performance and the limitations of those measurements is important in selecting a computer.

The rest of this section describes different ways in which performance can be determined; then, we describe the metrics for measuring performance from the viewpoint of both a computer user and a designer. We also look at how these metrics are related and present the classical processor performance equation, which we will use throughout the text.

Defining performance

When we say one computer has better performance than another, what do we mean? Although this question might seem simple, an analogy with passenger airplanes shows how subtle the question of performance can be. The table below lists some typical passenger airplanes, together with their cruising speed, range, and capacity. If we wanted to know which of the planes in this table had the best performance, we would first need to define performance. For example, considering different measures of performance, we see that the plane with the highest cruising speed was the Concorde (retired from service in 2003), the plane with the longest range is the DC-8, and the plane with the largest capacity is the 747.

Table 1.6.1: The capacity, range, and speed for a number of commercial airplanes (COD Figure 1.14).

The last column shows the rate at which the airplane transports passengers, which is the capacity times the cruising speed (ignoring range and takeoff and landing times).

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers x m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Let's suppose we define performance in terms of speed. This still leaves two possible definitions. You could define the fastest plane as the one with the highest cruising speed, taking a single passenger from one point to another in the least time. If you were interested in transporting 450 passengers from one point to another, however, the 747 would clearly be the fastest, as the last column of the figure shows. Similarly, we can define computer performance in several distinct ways.

If you were running a program on two different desktop computers, you'd say that the faster one is the desktop computer that gets the job done first. If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day. As an individual computer user, you are interested in reducing *response time*—the time between the start and completion of a task—also referred to as *execution time*. Datacenter managers often care about increasing *throughput* or *bandwidth*—the total amount of work done in a given time. Hence, in most cases, we will need different performance metrics as well as different sets of applications to benchmark personal mobile devices, which are more focused on response time, versus servers, which are more focused on throughput.

Response time: Also called ***execution time***. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

Throughput: Also called ***bandwidth***. Another measure of performance, it is the number of tasks completed per unit time.

PARTICIPATION ACTIVITY 1.6.1: Execution time and throughput.

1) At Joe's Car Wash, a car enters the wash, and exits 5 minutes later (nice and clean). 5 minutes is the _____.
   - ○ execution time
   - ○ throughput

2) At Joe's Car Wash, a car enters the wash, and exits 5 minutes later. However, 1 minute after a car enters, the next car can enter. Thus, once full, the car wash outputs 1 clean car per minute. 1 car per minute is the _____.
   - ○ response time
   - ○ throughput

3) Response time is synonymous with _____.
   - ○ execution time
   - ○

throughput

4) Cars drive 60 km/h over a 1 km long bridge. A car thus requires 1 minute to cross the bridge. Cars stay separated by about 100 m, so 1 car enters and another exits the bridge every 6 seconds. The execution time is _____.
   - 1 minute
   - 6 seconds

5) Cars drive 60 km/h over a 1 km long bridge. A car thus requires 1 minute to cross the bridge. Cars stay separated by about 100 m, so 1 car enters and another exits the bridge every 6 seconds. The throughput is _____.
   - 6 cars / sec
   - 10 cars / minute

6) Increasing the speed limit for cars driving over a bridge _____ the execution time.
   - improves
   - worsens

7) Increasing the speed limit for cars driving over a bridge while keeping the same minimum separation between cars _____ the throughput.
   - improves
   - worsens

PARTICIPATION ACTIVITY 1.6.2: Example of throughput and response time.

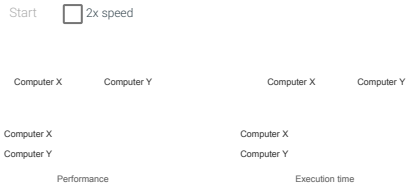
1) Replacing a processor in a computer with a faster processor has what effect?
   - Decreases response time
   - Increases throughput
   - Both (decreases response time and increases throughput)

2) Adding additional processors to a system that uses multiple processors for separate tasks -- for example, searching the web -- has what effect? Assume that before adding processors, tasks do not wait to execute (tasks do not "queue up").
   - Decreases response time
   - Increases throughput
   - Both (decreases response time and increases throughput)

3) Adding additional processors to a system that uses multiple processors for separate tasks -- for example, searching the web -- has what effect? Assume that before adding processors, tasks often must wait to execute due to another task executing (tasks "queue up").
   - Decreases response time
   - Increases throughput
   - Both (decreases response time and increases throughput)

In discussing the performance of computers, we will be primarily concerned with response time for the first few chapters. To maximize performance, we want to minimize response time or execution time for some task.

PARTICIPATION ACTIVITY 1.6.3: Performance.



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

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

$\text{Performance}_X > \text{Performance}_Y$ $\quad$ $\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$

In discussing a computer design, we often want to relate the performance of two different computers quantitatively. We will use the phrase "X is *n* times faster than Y"—or equivalently "X is *n* times as fast as Y"—to mean

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is *n* times as fast as Y, then the execution time on Y is *n* times as long as it is on X:

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

Example 1.6.1: Relative performance.

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

Answer

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

In the above example, we could also say that computer B is 1.5 times *slower than* computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 1.5$$

means that

$$\frac{\text{Performance}_A}{1.5} = \text{Performance}_B$$

For simplicity, we will normally use the terminology *as fast as* when we try to compare computers quantitatively. Because performance and execution time are reciprocals, increasing performance requires decreasing execution time. To avoid the potential confusion between the terms *increasing* and *decreasing*, we usually say "improve performance" or "improve execution time" when we mean "increase performance" and "decrease execution time."

PARTICIPATION ACTIVITY 1.6.4: Performance.

Computer A requires 10 seconds to compress a file. Computer B requires 5 seconds.

1) Which computer has higher execution time?
   - ○ A
   - ○ B

2) Which computer has higher performance?
   - ○ A
   - ○ B

3) If performance is 1 / execution time, what is A's performance?
   - ○ 0.1
   - ○ 10

4) If performance is 1 / execution time, what is B's performance?
   - ○ 0.2
   - ○ 5

5) How many times faster is B than A at compressing a file?
   - ○

2

1/2

6) To determine how many times faster Computer C is than Computer D, which is the correct calculation?

- PerfC / PerfD
- PerfD / PerfC

Measuring performance

Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest. Program *execution time* is measured in seconds per program. However, time can be defined in different ways, depending on what we count. The most straightforward definition of time is called ***wall clock time***, ***response time***, or ***elapsed time***. These terms mean the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead—everything.

Computers are often shared, however, and a processor may work on several programs simultaneously. In such cases, the system may try to optimize throughput rather than attempt to minimize the elapsed time for one program. Hence, we often want to distinguish between the elapsed time and the time over which the processor is working on our behalf. *CPU execution time* or simply *CPU time*, which recognizes this distinction, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs. (Remember, though, that the response time experienced by the user will be the elapsed time of the program, not the CPU time.) CPU time can be further divided into the CPU time spent in the program, called *user CPU time*, and the CPU time spent in the operating system performing tasks on behalf of the program, called *system CPU time*. Differentiating between system and user CPU time is difficult to do accurately, because it is often hard to assign responsibility for operating system activities to one user program rather than another and because of the functionality differences between operating systems.

CPU execution time: Also called ***CPU time***. The actual time the CPU spends computing for a specific task.

User CPU time: The CPU time spent in a program itself.

System CPU time: The CPU time spent in the operating system performing tasks on behalf of the program.

For consistency, we maintain a distinction between performance based on elapsed time and that based on CPU execution time. We will use the term ***system performance*** to refer to elapsed time on an unloaded system and ***CPU performance*** to refer to user CPU time. We will focus on CPU performance in this chapter, although our discussions of how to summarize performance can be applied to either elapsed time or CPU time measurements.

PARTICIPATION ACTIVITY 1.6.5: Measuring performance.

A task runs alone on a CPU. The task starts by running for 5 ms. The task then waits for 4 ms while the operating system runs some instructions to access disk. The CPU is then idle for 2 ms while waiting for data from disk. Finally, the task runs another 10 ms and completes.

1) The elapsed time is ____ ms.

Check Show answer

2) The user CPU time is _____ ms.

Check Show answer

3) The CPU time is _____ ms.

Check Show answer

4) The system performance is _____ ms.

Check Show answer

5) The CPU performance is _____ ms.

Check Show answer

Understanding program performance

Different applications are sensitive to different aspects of the performance of a computer system. Many applications, especially those running on servers, depend as much on I/O performance, which, in turn, relies on both hardware and software. Total elapsed time measured by a wall clock is the measurement of interest. In some application environments, the user may care about throughput, response time, or a complex combination of the two (e.g., maximum throughput with a worst-case response time). To improve the performance of a program, one must have a clear definition of what performance metric matters and then proceed to find performance bottlenecks by measuring program execution and looking for the likely bottlenecks. In the following chapters, we will describe how to search for bottlenecks and improve performance in various parts of the system.

Although as computer users we care about time, when we examine the details of a computer it's convenient to think about performance in other metrics. In particular, computer designers may want to think about a computer by using a measure that relates to how fast the hardware can perform basic functions. Almost all computers are constructed using a clock that determines when events take place in the hardware. These discrete time intervals are called *clock cycles* (or *ticks*, *clock ticks*, *clock periods*, *clocks*, *cycles*). Designers refer to the length of a *clock period* both as the time for a complete clock cycle (e.g., 250 picoseconds, or 250 ps) and as the clock rate (e.g., 4 gigahertz, or 4 GHz). ***Clock rate*** is the inverse of the clock period. In the next subsection, we will formalize the relationship between the clock cycles of the hardware designer and the seconds of the computer user.

Clock cycle: Also called ***tick***, ***clock tick***, ***clock period***, ***clock***, or ***cycle***. The time for one clock period, usually of the processor clock, which runs at a constant rate.

Clock period: The length of each clock cycle.

PARTICIPATION ACTIVITY 1.6.6: Clock cycle and clock period.

1) A particular processor has a clock rate of 1 GHz. The clock thus ticks one billion times per second.
   - ○ True
   - ○ False

2) A clock rate of 1 GHz corresponds to a period of 1 nanosecond, which is $1x10^{-9}$ seconds.
   - ○ True
   - ○ False

PARTICIPATION ACTIVITY 1.6.7: Check yourself: Throughput and response time.

Suppose we know that an application that uses both personal mobile devices and the Cloud is limited by network performance. For the following changes, determine what improvements to the application's performance is/are made, if any.

1) An extra network channel is added between the PMD and the Cloud, increasing the total network throughput and reducing the delay to obtain network access (since there are now two channels).
   - ○ Only the throughput improves.
   - ○ Both response time and throughput improve.
   - ○ Neither response time or throughput improves.

2) The networking software is improved, thereby reducing the network communication delay, but not increasing throughput.
   - ○ Only the throughput improves.
   - ○ Only the response time improves.
   - ○ Neither response time or throughput improves.

3) More memory is added to the computer.
   - ○ Both response time and throughput improve.
   - ○ Neither response time or throughput improves.

PARTICIPATION ACTIVITY 1.6.8: Check yourself: Program performance.

1) Computer C's performance is 4 times as fast as the performance of computer B, which runs a given application in 28 seconds. How long will computer C take to run that application?

   [ ] sec

   Check Show answer

CPU performance and its factors

Users and designers often examine performance using different metrics. If we could relate these different metrics, we could determine the effect of a design change on the performance as experienced by the user. Since we are confining ourselves to CPU performance at this

point, the bottom-line performance measure is CPU execution time. A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

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

Alternatively, because clock rate and clock cycle time are inverses,

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

This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle. As we will see in later chapters, the designer often faces a trade-off between the number of clock cycles needed for a program and the length of each cycle. Many techniques that decrease the number of clock cycles may also increase the clock cycle time.

Example 1.6.2: Improving performance.

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Answer

Let's first find the number of clock cycles required for the program on A:

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{seconds}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU time for B can be found using this equation:

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4\text{GHz}$$

To run the program in 6 seconds, B must have twice the clock rate of A.

PARTICIPATION ACTIVITY 1.6.9: Improving performance.

Refer to the above example.

1) What is the CPU clock cycles for computer A?
   - $2 \times 10^9 \frac{\text{cycles}}{\text{sec}}$
   - 20 x 10^9 cycles
   - 10 sec

2) Computer B's performance is improved by reducing the _____.
   - number of clock cycles required to execute the program
   - length of a clock cycle

Instruction performance

The performance equations above did not include any reference to the number of instructions needed for the program. However, since the compiler clearly generated instructions to execute, and the computer had to execute the instructions to run the program, the execution time must depend on the number of instructions in a program. One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction. Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

The term *clock cycles per instruction*, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as *CPI*. Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program. CPI provides one way of comparing two different implementations of the identical instruction set architecture, since the number of instructions executed for a program will, of course, be the same.

Clock cycles per instruction (***CPI***): Average number of clock cycles per instruction for a program or program fragment.

Example 1.6.3: Using the performance equation.

Suppose we have two implementations of the same instruction set architecture. Computer A has a

clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Answer

We know that each computer executes the same number of instructions for the program; let's call this number *I*. First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$

$$\text{CPU clock cycles}_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$\begin{aligned}\text{CPU time}_A &= \text{CPU clock cycles}_A \times \text{Clock cycle time} \\ &= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}\end{aligned}$$

Likewise, for B:

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

We can conclude that computer A is 1.2 times as fast as computer B for this program.

PARTICIPATION ACTIVITY 1.6.10: Using the performance equation.

Refer to the above example.

1) How does one know that each computer executes the same number of instructions for the program?
   - All computers use the same number of instructions for a given program.
   - Both computers use the same instruction set architecture.
   - Both computers use the same implementation.

2) Which computer has a faster clock?
   - Computer A
   - Computer B

3) Which computer requires fewer clock cycles to execute a single instruction?
   - Computer A
   - Computer B

4) If Computer A executes 1000 instructions for the program, what is the program's CPU time on Computer A?
   - 1000 instr * 2.0 cycle/instr * 250 ps/cycle = 500,000 ps.
   - 1000 instr * 1.2 cycle/instr * 500 ps/cycle = 600,000 ps.

5) If Computer A executes 1000 instructions for the program, how many instructions does Computer B execute for the program?
   - 1000
   - 1000 * 1.2 = 1200
   - 1000 * 2.0 = 2000

6) For a particular program, Computer's A and B execute 2000 instructions. A's CPU time is 2000 * 2.0 * 250 = 1,000,000 ps. B's is 2000 * 1.2 * 500 = 1,200,000 ps. How much faster is Computer A than B?
   - 1.2
   - 200,000

7) Computer A is better than Computer B.
   - Yes
   - Unclear

The classic CPU performance equation

We can now write this basic performance equation in terms of *instruction count* (the number of instructions executed by the program), CPI, and clock cycle time:

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

or, since the clock rate is the inverse of clock cycle time:

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

These formulas are particularly useful because they separate the three key factors that affect performance. We can use these formulas to compare two different implementations or to evaluate a design alternative if we know its impact on these three parameters.

Instruction count: The number of instructions executed by the program.

Example 1.6.4: Comparing code segments.

A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction counts for each instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Answer

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

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

This yields

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

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

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

PARTICIPATION ACTIVITY 1.6.11: Comparing code segments.

Refer to the above example.

1) Instruction class ___ requires the largest number of cycles per instruction.

Check Show answer

2) Code sequence 2 executes ____ instructions.

Check Show answer

3) Code sequence 2 requires ____ CPU clock cycles.

Check Show answer

4) Assume a new code sequence 3 contains the following instruction counts for each instruction class. What is code sequence 3's CPU clock cycles?

Code sequence	Instruction counts for each instruction class		
	A	B	C
3	10	4	6

Check Show answer

5) Assume a new code sequence 3 contains the following instruction counts for each instruction class. What is code sequence 3's CPI?

Code sequence	Instruction counts for each instruction class		
	A	B	C
3	10	4	6

Check Show answer

The Big Picture

The table below shows the basic measurements at different levels in the computer and what is being measured in each case. We can see how these factors are combined to yield execution time measured in seconds per program:

$$\text{Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Always bear in mind that the only complete and reliable measure of computer performance is time. For example, changing the instruction set to lower the instruction count may lead to an organization with a slower clock cycle time or higher CPI that offsets the improvement in instruction count. Similarly, because CPI depends on the type of instructions executed, the code that executes the fewest number of instructions may not be the fastest.

Table 1.6.2: The basic components of performance and how each is measured (COD Figure 1.15).

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

How can we determine the value of these factors in the performance equation? We can measure the CPU execution time by running the program, and the clock cycle time is usually published as part of the documentation for a computer. The instruction count and CPI can be more difficult to obtain. Of course, if we know the clock rate and CPU execution time, we need only one of the instruction count or the CPI to determine the other.

We can measure the instruction count by using software tools that profile the execution or by using a simulator of the architecture. Alternatively, we can use hardware counters, which are included in most processors, to record a variety of measurements, including the number of instructions executed, the average CPI, and often, the sources of performance loss. Since the instruction count depends on the architecture, but not on the exact implementation, we can measure the instruction count without knowing all the details of the implementation. The CPI, however, depends on a wide variety of design details in the computer, including both the memory system and the processor structure (as we will see in COD Chapter 4 (The Processor) and COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)), as well as on the mix of instruction types executed in an application. Thus, CPI varies by application, as well as among implementations with the same instruction set.

The above example shows the danger of using only one factor (instruction count) to assess performance. When comparing two computers, you must look at all three components, which combine to form execution time. If some of the factors are identical, like the clock rate in the above example, performance can be determined by comparing all the nonidentical factors. Since CPI varies by *instruction mix*,

both instruction count and CPI must be compared, even if clock rates are equal. Several exercises at the end of this chapter ask you to evaluate a series of computer and compiler enhancements that affect clock rate, CPI, and instruction count. In COD Section 1.10 (Fallacies and pitfalls), we'll examine a common performance measurement that does not incorporate all the terms and can thus be misleading.

Instruction mix: A measure of the dynamic frequency of instructions across one or many programs.

PARTICIPATION ACTIVITY 1.6.12: Performance components.

1) Assume CPI and clock cycle time remain constant. Reducing the instruction count will reduce the program's execution time.
   - ○ True
   - ○ False

2) For a given number of instructions, assume CPI is increased by 20%, and clock cycle time is decreased by 10%. The program execution time decreases.
   - ○ True
   - ○ False

Understanding program performance

The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware. The following table summarizes how these components affect the factors in the CPU performance equation.

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in varied ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

Elaboration

Although you might expect that the minimum CPI is 1.0, as we'll see in COD Chapter 4 (The Processor), some processors fetch and execute multiple instructions per clock cycle. To reflect that approach, some designers invert CPI to talk about ***IPC****, or* ***instructions per clock cycle****. If a processor executes on average two instructions per clock cycle, then it has an IPC of 2 and hence a CPI of 0.5.*

Elaboration

Although clock cycle time has traditionally been fixed, to save energy or temporarily boost performance, today's processors can vary their clock rates, so we would need to use the average clock rate for a program. For example, the Intel Core i7 will temporarily increase clock rate by about 10% until the chip gets too warm. Intel calls this Turbo mode.

PARTICIPATION ACTIVITY 1.6.13: Check yourself: Program execution time.

A given application written in Java runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.6 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1.

1) How fast can we expect the application to run using this new compiler?

- $\frac{15 \times 0.6}{1.1} = 8.2 \text{ sec}$
- $15 \times 0.6 \times 1.1 = 9.9 \text{ sec}$
- $\frac{15 \times 1.1}{0.6} = 27.5 \text{ sec}$

Provide feedback on this section