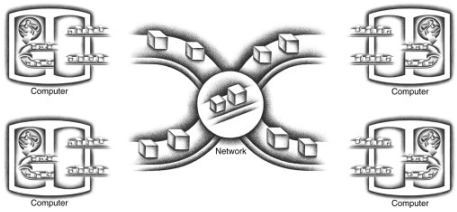
6.1 Introduction

> "I swing big, with everything I've got. I hit big or I miss big. I like to live as big as I can.
> *Babe Ruth, American baseball player*

Figure 6.1.1: Multiprocessor or cluster organization.



> "Over the Mountains Of the Moon, Down the Valley of the Shadow, Ride, boldly ride the shade replied—If you seek for El Dorado!
> *Edgar Allan Poe, "El Dorado," stanza 4, 1849*

Computer architects have long sought the "The City of Gold" (El Dorado) of computer design: to create powerful computers simply by connecting many existing smaller ones. This golden vision is the fountainhead of *multiprocessors*. Ideally, customers order as many processors as they can afford and receive a commensurate amount of performance. Thus, multiprocessor software must be designed to work with a variable number of processors. As mentioned in COD Chapter 1 (Computer Abstractions and Technology), energy has become the overriding issue for both microprocessors and datacenters. Replacing large inefficient processors with many smaller, efficient processors can deliver better performance per joule both in the large and in the small, if software can efficiently use them. Thus, improved energy efficiency joins scalable performance in the case for multiprocessors.

Multiprocessor: A computer system with at least two processors. This computer is in contrast to a uniprocessor, which has one, and is increasingly hard to find today.

Since multiprocessor software should scale, some designs support operation in the presence of broken hardware; that is, if a single processor fails in a multiprocessor with *n* processors, these systems would continue to provide service with *n* - 1 processors. Hence, multiprocessors can also improve availability (see COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)).

High performance can mean greater throughput for independent tasks, called *task-level parallelism* or *process-level parallelism*. These tasks are independent single-threaded applications, and they are an important and popular use of multiple processors. This approach contrasts with running a single job on multiple processors. We use the term *parallel processing program* to refer to a single program that runs on multiple processors simultaneously.

Task-level parallelism or ***process-level parallelism***: Utilizing multiple processors by running independent programs simultaneously.

Parallel processing program: A single program that runs on multiple processors simultaneously.

There have long been scientific problems that have needed much faster computers, and this class of problems has been used to justify many novel parallel computers over the decades. Some of these problems can be handled simply today, using a cluster composed of microprocessors housed in many independent servers (see COD Section 6.7 (Clusters, warehouse scale computers, and other message-passing multiprocessors)). In addition, clusters can serve equally demanding applications outside the sciences, such as search engines, Web servers, email servers, and databases.

Cluster: A set of computers connected over a local area network that function as a single large multiprocessor.

As described in COD Chapter 1 (Computer Abstractions and Technology), multiprocessors have been shoved into the spotlight because the energy problem means that future increases in performance will primarily come from explicit hardware parallelism rather than much higher clock rates or vastly improved CPI. As we said in COD Chapter 1 (Computer Abstractions and Technology), they are called *multicore microprocessors* instead of multiprocessor microprocessors, presumably to avoid redundancy in naming. Hence, processors are often called *cores* in a multicore chip. The number of cores is expected to increase with **Moore's Law**. These multicores are almost always *Shared Memory Processors (SMPs)*, as they usually share a single physical address space. We'll see SMPs more in COD Section 6.5 (Multicore and other shared memory multiprocessors).

Multicore microprocessor: A microprocessor containing multiple processors ("cores") in a single integrated circuit. Virtually all microprocessors today in desktops and servers are multicore.

Shared memory multiprocessor (***SMP***): A parallel processor with a single physical address space.

The state of technology today means that programmers who care about performance must become parallel programmers, for sequential code now means slow code.

The tall challenge facing the industry is to create hardware and software that will make it easy to write correct parallel processing programs that will execute efficiently in performance and energy as the number of cores per chip scales.

This abrupt shift in microprocessor design caught many off guard, so there is a great deal of confusion about the terminology and what it means. The figure below tries to clarify the terms serial, parallel, sequential, and concurrent. The columns of this figure represent the software, which is either inherently sequential or concurrent. The rows of the figure represent the hardware, which is either serial or parallel. For example, the programmers of compilers think of them as sequential programs: the steps include parsing, code generation, optimization,

and so on. In contrast, the programmers of operating systems normally think of them as concurrent programs: cooperating processes handling I/O events due to independent jobs running on a computer.

Figure 6.1.2: Hardware/software categorization and examples of application perspective on concurrency versus hardware perspective on parallelism (COD Figure 6.1).

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

The point of these two axes of the figure above is that concurrent software can run on serial hardware, such as operating systems for the Intel Pentium 4 uniprocessor, or on parallel hardware, such as an OS on the more recent Intel Core i7. The same is true for sequential software. For example, the MATLAB programmer writes a matrix multiply thinking about it sequentially, but it could run serially on the Pentium 4 or in parallel on the Intel Core i7.

You might guess that the only challenge of the parallel revolution is figuring out how to make naturally sequential software have high performance on parallel hardware, but it is also to make concurrent programs have high performance on multiprocessors as the number of processors increases. With this distinction made, in the rest of this chapter we will use *parallel processing program* or *parallel software* to mean either sequential or concurrent software running on parallel hardware. The next section of this chapter describes why it is hard to create efficient parallel processing programs.

Before proceeding further down the path to parallelism, don't forget our initial incursions from the earlier chapters:

- COD Chapter 2 (Instructions: Language of the Computer), COD Section 2.11: Parallelism and instructions: Synchronization
- COD Chapter 3 (Arithmetic for Computers), COD Section 3.6: Parallelism and computer arithmetic: Subword parallelism
- COD Chapter 4 (The Processor), COD Section 4.10: Parallelism via instructions
- COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy), COD Section 5.10: Parallelism and memory hierarchy: Cache coherence

PARTICIPATION ACTIVITY 6.1.1: Check yourself: Multiprocessors.

1) To benefit from a multiprocessor, an application must be concurrent.
   - ○ True
   - ○ False

Provide feedback on this section