a wiz at concurrent programming. Thread monkey is a title of prestige, unlike the often derogatory connotations associated with "code monkey."

Parallelism and Concurrency: What's the Difference?

The terms "parallel" and "concurrent" have been tossed around with increasing frequency since the release of general-purpose multicore processors. Even prior to that, there has been some confusion about these terms in other areas of computation. What is the difference, or *is* there a difference, since use of these terms seems to be almost interchangeable?

A system is said to be *concurrent* if it can support two or more actions *in progress* at the same time. A system is said to be *parallel* if it can support two or more actions executing simultaneously. The key concept and difference between these definitions is the phrase "in progress."

A concurrent application will have two or more threads in progress at some time. This can mean that the application has two threads that are being swapped in and out by the operating system on a single core processor. These threads will be "in progress"—each in the midst of its execution—at the same time. In parallel execution, there must be multiple cores available within the computation platform. In that case, the two or more threads could each be assigned a separate core and would be running simultaneously.

I hope you've already deduced that "parallel" is a subset of "concurrent." That is, you can write a concurrent application that uses multiple threads or processes, but if you don't have multiple cores for execution, you won't be able to run your code in parallel. Thus, *concurrent programming* and *concurrency* encompass all programming and execution activities that involve multiple streams of execution being implemented in order to solve a single problem.

For about the last 20 years, the term *parallel programming* has been synonymous with message-passing or distributed-memory programming. With multiple compute nodes in a cluster or connected via some network, each node with one or more processors, you had a parallel platform. There is a specific programming methodology required to write applications that divide up the work and share data. The programming of applications utilizing threads has been thought of as concurrent programming, since threads are part of a shared-memory programming model that fits nicely into a single core system able to access the memory within the platform.

I will be striving to use the terms "parallel" and "concurrent" correctly throughout this book. This means that *concurrent programming* and *design of concurrent algorithms* will assume that the resulting code is able to run on a single core or multiple cores without any drastic changes. Even though the implementation model will be threads, I will talk about the parallel execution of concurrent codes, since I assume that we all have multicore processors available on which to execute those multiple threads. Also, I'll use the term "parallelization" as the process of translating applications from serial to concurrent (and the term "concurrentization" doesn't roll off the tongue quite as nicely).