Parallelization and Threading Enhancements

Availability: Framework 4—Some Functionality in 3.5 with Parallel Extensions CTP

Until recently, CPU manufactures regularly released faster and faster processors. Speed increases, however, have all but ground to a halt due to various issues such as signal noise, power consumption, heat dissipation, and non-CPU bottlenecks.

No doubt these issues will be resolved in the future, but in the meantime manufacturers are instead concentrating on producing processors with multiple cores. Multicore processors can process sections of code in parallel, resulting in some calculations being performed quicker and thus increasing application performance. To take full advantage of multicore machines, however, code has to be designed to be run in parallel.

A number of years ago, Microsoft foresaw the importance that multicore processors would come to play and started developing the parallel extensions. In .NET 4.0, Microsoft built on this earlier work and integrated it into the core framework, enabling developers to parallelize their code in an easy and consistent way. Because this is the first mainstream release, it’s probably wise to expect to see some minor tweaks and API changes in the future.

Although the parallelization enhancements make writing code to run in parallel much easier, don’t underestimate the increasing complexity that parallelizing an application can bring. Parallelization shares many of the same issues you might have experienced when creating multithreaded applications. You must take care when developing parallel applications to isolate code that can be parallelized.

Parallelization Overview

Some of the parallelization enhancements might look familiar to a few readers because they were released previously as part of the parallel extensions. .NET 4.0 builds on this work but brings the extensions into the core CLR within mscorlib.dll.

The Microsoft parallel extensions and enhancements can be divided into five main areas:

- Task Parallel Library (TPL)) and Concurrency and Coordination Runtime (CCR)
- Parallel LINQ (PLINQ)
- New debugging and profiling tools
• Coordination data structures
• Parallel Pattern Library (PPL)—C++ only; not covered

Important Concepts

Parallelism and threading are confusing and there are a few questions many developers have (see the following questions).

Why Do I Need These Enhancements?

Can’t you just create lots of separate threads? Well, you can, but there are a couple of issues with this approach. First, creating a thread is a resource-intensive process, so (depending on the type of work you do) it might be not be the most efficient and quickest way to complete a task. Creating too many threads, for example, can slow task completion because each thread is never given time to complete as the operating system rapidly switches between them. And what happens if someone loads up two instances of your application?

To avoid these issues, .NET implements a thread pool that has a bunch of threads up and running, ready to do your bidding. The thread pool also can impose a limit on the number of threads created, preventing thread starvation issues.

However the thread pool isn’t so great at letting you know when work has been completed or cancelling running threads. The thread pool also doesn’t have any information about the context in which the work is created, which means it can’t schedule it as efficiently as it could have done. Enter the new parallelization functionality that provides additional cancellation and scheduling, and offers an intuitive way of programming.

Note that the parallelization functionality works on top of .NET’s thread pool instead of replacing it. See Chapter 4 for details about improvements made to the thread pool in this release.

Concurrent!= Parallel

If your application is multithreaded is it running in parallel? Probably not—applications running on a single CPU machine can appear to run in parallel because the operating system allocates time with the CPU to each thread and then rapidly switches between them (known as time slicing). Threads might not ever be actually running at the same time (although they could be), whereas in a parallelized application work is actually being conducted at the same time (Figure 5-1). Processing work at the same time can introduce some complications in your application regarding access to resources.

Daniel Moth (from the Parallel computing team at Microsoft) puts it succinctly when he says the following (http://www.danielmoth.com/Blog/2008/11/threadingconcurrency-vs-parallelism.html):

“On a single core you can use threads and you can have concurrency, but to achieve parallelism on a multi-core box you have to identify in your code the exploitable concurrency: the portions of your code that can truly run at the same time.”