Planning for Parallel Optimization

Optimizing your application for multi-core technology is fast becoming a requirement: multi-core computers have become mainstream, making up 83% of PC shipments in 2010. And the number of cores is increasing, with 60% of shipments projected to have 4 or more cores in 2012, according to IDC (Worldwide PC Processor Forecasting 2009-2013). Parallel optimization can result in big performance improvements, but you will need a plan of action that is well suited to your application. Here are some tips to help you get started.

Redesign or Tune

The first choice to make is whether you will start from scratch with a parallel design, or tune existing code. If you already have a serial application that is functioning correctly, you may use that as your starting point and look for ways to introduce parallelism.

Before making any changes to existing code, be sure to measure the performance of your current software to establish a baseline. Then as you make changes, repeat the measurement so that you can tell if your changes are actually resulting in improved performance.

How you measure the performance of your application will depend on what your application is designed to do. To be able to repeat the performance measurements, start by identifying a workload that you will measure. A workload is a task or set of tasks that you identify for your application to do. The goal is to define a repeatable workload, then take measurements while the workload executes. Once you know the amount of work accomplished during a given amount of time, you can repeat the workload later and see if your application is accomplishing more work within the same period of time, or else accomplishing the same amount of work in a shorter period of time. This will give you a direct measurement of the performance improvements that you achieve as you tune your application. You may also measure the progress of your tuning efforts by using tools that measure the concurrency levels of your application, such as the Intel® Concurrency Checker.

Decompose Functions or Data

If you start with an existing serial application, the structure of the application will determine whether to employ functional decomposition or data decomposition (or both): If your application has functions or tasks that are independent of each other, then they may be run in parallel. If your application has functions that operate on large amounts of data and it is possible to break up the data into smaller units that can be processed independently, then you may employ data decomposition.

The nature of your application will also determine the granularity of parallelism that is optimal. Granularity refers to how often the tasks that make up your application need to communicate with each other. The less often communication is needed, the more coarse-grained parallelism can be used, and the more your application can benefit from parallelism since it will require less communication overhead.

Identify Hotspots and Bottlenecks

It is important to identify where the biggest problems are before you start to make changes. Tools to identify hotspots or bottlenecks in your code will guide you to apply your efforts to the areas that can yield the most improvement. Hotspots are places where the processor spends a lot of time, so they may be good areas to
target for optimization if the code is inefficient. However, it may be that a hotspot is already efficient, and the reason that the processor spends a lot of time there is because a lot of work is being accomplished. A hotspot is a bottleneck when the extra processor time is due to inefficiency. If you determine that a bottleneck is parallelizable, it is an ideal place to apply optimization effort.

Choose a Methodology

Once you know what areas of your code need improvement, you have different options for proceeding with parallel optimization. According to the Intel Parallel Computing Survey (Evans Data Corp., April 2011), the three most popular parallel programming techniques used by developers are multi-threading, shared memory model, and message passing.

- **Multi-threading** causes multiple threads (tasks to be completed by a CPU) to exist within a single process, sharing memory and other resources, resulting in faster operation on multi-core systems. A drawback of multi-threading is that it introduces non-determinism: you may not be able to predict the order in which processing occurs, which could lead to errors.
- **Shared memory model**, in which a single memory space is used by multiple processors, offers a unified address space that can be simple to work with, but requires care to avoid race conditions when there is dependency between events.
- **Message passing** involves communication between processes, and may require more work to implement, but it avoids race conditions through synchronization.

Measure Progress and Check for Errors

Each time you incorporate more parallelism into your application, it is a good idea to re-measure your workload to see if you are making progress toward improving the performance of your application. It is also important at this point to make sure you have not introduced any defects by employing tools that can help you check for threading errors, such as Intel® Parallel Studio. Once you determine that your most recent parallel optimization step was a success, you may iterate the process to seek even more parallelism and performance improvement, or stop when you have achieved your goals.

For additional information about parallel optimization, please visit the Intel® Software Network Parallel Programming Community where you may access tools and resources to help you optimize your application for multi-core technology.

Illustration

About the Author

Diana Byrne is a Multi-core Product Manager in the Software and Services Group at Intel, where she has worked since 2004. She holds master's degrees in Mathematics, Computer Science, and Management of Technology.