Getting Started with the Intel® Parallel Studio

The Intel® Parallel Studio offers a complete set of tools for every phase in the development cycle of parallel applications on multi-core systems: analysis and design, coding and debugging, verification and tuning.

All Parallel Studio functionality is integrated into the Microsoft Visual Studio* development environment:

Intel® Parallel Advisor Lite	A utility to help identify candidate functions for parallelizing and advise on protecting or sharing data.
Intel® Parallel Composer	Intel® C++ compiler, Intel® Threading Building Blocks, Intel® Integrated Performance Primitives, and Intel® Parallel Debugger Extension.
Intel® Parallel Inspector	A multithreading tool to detect challenging threading and memory errors.
Intel® Parallel Amplifier	A performance analysis and tuning tool for parallel applications.

This guide will use a sample application to show you how to get started with all these components.

Contents

1 About the Sample Application

This guide references the `NQueens-ParallelStudio` sample application found at: `C:\Program Files\Intel\Parallel Studio\Samples.` The NQueens solution includes a serial project, a parallelized project with a bug, and a correctly parallelized project.

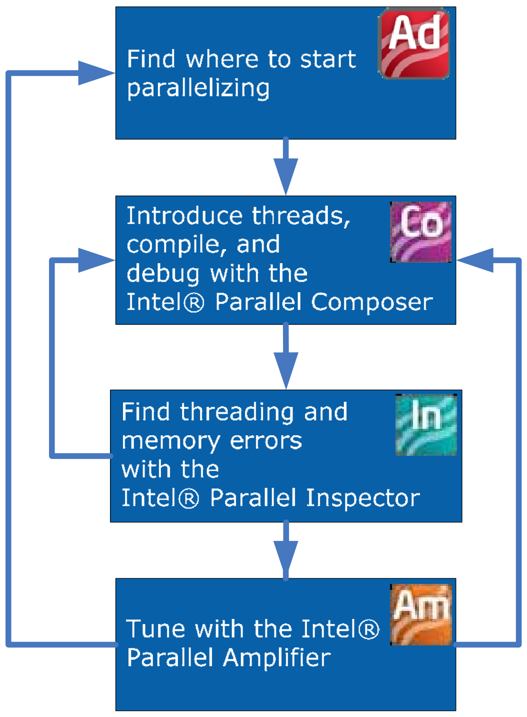
To follow along with the steps in this guide, unzip `NQueens-ParallelStudio.zip` to a local directory and load the `NQueens-ParallelStudio.sln` solution file into Microsoft Visual Studio*.

NOTE: For a graphical tour of the Intel® Parallel Studio, see the Show Me video offered at http://software.intel.com/en-us/articles/intel-parallel-studio. This video uses the NQueens sample application to walk you through each step in the workflow discussed in this document. After successful installation of the Intel® Parallel Studio, the video will also be available in the `documentation\en_US` subdirectory of the installation directory.

2 Creating Parallel Programs Using the Intel® Parallel Studio

Performance experts at Intel, experienced in parallelizing a variety of applications, have recognized four basic steps to create parallel programs: 1) Finding where to start parallelizing 2) Introducing parallelism into the application 3) Debugging the parallel program for correctness 4) Tuning the program to ensure good thread and CPU-core utilization. The tools and technology in Intel® Parallel Studio are designed to ease the time and complexity of these four steps.

Figure 1. The Intel® Parallel Studio Workflow


Find where to start parallelizing
Ad
Introduce threads, compile, and debug with the Intel® Parallel Composer
Co
Find threading and memory errors with the Intel® Parallel Inspector
In
Tune with the Intel® Parallel Amplifier
Am

Step 1. Find Where to Start Parallelizing

The first step to parallelizing an application is to identify candidate loops or sections in your application that may benefit from parallelization. These candidates are typically the obvious time-consuming algorithms in your application. Finding the hotspot, or most time-consuming function, is only a part of the process. You also need to identify the data objects that need to be made private or shared.

If you already know of potential candidates in your application that would benefit from parallelization or you already made an initial attempt at introducing parallelism, go directly to step 2 in this workflow.

Intel® Parallel Advisor Lite is a prototype utility we are testing to help you find and recommend the hotspots and objects to parallelize. After installing Parallel Studio, you can download Advisor Lite from http://whatif.intel.com. Refer to the installed documentation for Advisor Lite for more information about using it for finding parallelism opportunities.

Run the application serially first:

1. Extract the `NQueens-ParallelStudio.zip` to a directory of your choosing. If you installed to the default path, they are located at `C:\Program Files\Intel\Parallel Studio\Samples`.
2. Load the `NQueens-ParallelStudio.sln` into Microsoft Visual Studio and build it.
3. Set the `Step1-Serial-Hotspot` project as the Start-Up project by highlighting it and selecting **Project > Set as StartUp Project**.
4. Run the serial application to see its performance with **Debug > Start Without Debugging**. You will see a console window displayed with the NQueens usage information, and after a few seconds, the number of solutions found, and the time taken in milliseconds.

Result: As you read the source code, you will see that the `setQueen()`function is the hotspot, but its parent function, `solve(),` is where you need to parallelize because it contains a simple loop that drives the NQueen solution search.

Step 2. Introduce Threads, Compile and Debug

Use Intel® Parallel Composer to introduce a threading method to your application, as well as compile and debug the application. The Intel® C++ compiler, comprehensive threaded libraries, and a parallel debugger extension help you quickly create and debug threaded C/C++ applications in the Microsoft Visual Studio* development environment.

Composer provides several thread implementation methods: OpenMP* technology, Intel® Threading Building Blocks, new C++ parallel extensions, and a parallel multi-media library Intel® Integrated Performance Primitives. See "Select a Threading Technique" for more information on how to choose the best method for your application.

Try it now:

1. In the `NQueens-ParallelStudio.sln` solution, select **Step2-3-Parallel-Check** as the Start-Up Project.

2. Select the **Intel® C++ Compiler** to build the project: **Project > Intel Parallel Composer > Use Intel C++**.

3. Look at the `nq-parallelstart.cpp` file to see the OpenMP* implementation of the main driver function, `solve()`**.**

4. Build the project in **Debug** configuration.

5. Run the application using **Debug > Start Without Debugging** to see whether it runs correctly now that it is threaded.

   Result: You should see an error message of an incorrect number of solutions found. If you don't see such a message, increase the board-size (the default is 12) in the **Command Arguments** field in **Project > Properties > Debugging**.

   As with many threading errors, this application seems to work correctly on some runs with some input data, but still has a bug. Find that bug!

Refer to the Intel® Parallel Composer Getting Started Guide for detailed explanations of the threading techniques and libraries, and parallel debugging.

Step 3. Find Threading and Memory Errors

Use Intel® Parallel Inspector to find and get rid of common threading and memory errors. Inspector is a multithreading error checking tool for Microsoft Visual Studio C/C++ developers. The tool detects challenging threading and memory errors and provides guidance to help ensure application reliability.

Try it now:

1. In the `NQueens-ParallelStudio.sln` solution, use the **Step2-3-Parallel-Check** project again.

2. From the Visual Studio* main menu, choose **Tools > Intel Parallel Inspector > Inspect Threading Errors**. A pop-up window appears.

3. In the pop-up window, notice there is a selector-bar to the right of the dial. Move that down to the 2nd selection — **"Does my target have deadlocks or data races?"** and then click **Run Analysis**. Since the Inspector is analyzing the application execution, it runs a lot slower than if no analysis were underway. You will see the NQueens console window displayed during this.

4. When Inspector finishes the analysis, a pop-up will appear with two options. Choose **Interpret Results**.

5. Double-click on the **Data race** problem identified to see the source and details about it.

   Result: The `nrOfSolutions++;` statement in the `setQueens()` function should be in a critical section. This is the bug.

NOTE: Try the Inspector's Memory Checking feature to see if there are other memory leaks that could impact correctness and performance. To try this, in #2 above, use **Tools > Intel Parallel Inspector > Inspect Memory Errors** instead.

Refer to the **Intel® Parallel Inspector Getting Started Guide** for detailed explanations of the analysis and results that Inspector provides.

Step 4. Tune

Use Intel® Parallel Amplifier to tune your threaded application for multi-core performance scalability by locating unexpected serialization and other performance bottlenecks. Amplifier will help you find where your program is spending time, where your concurrency is poor, and where your program is waiting.

Try it now:

1. In the `NQueens-ParallelStudio.sln` solution, select the **Step4-Parallel-Tune** project as the Start-Up Project and build it in Release configuration. The `nqueens-parallelfinal.cpp` file contains the corrected code in the `setQueens()` function.

2. From the Visual Studio* menu, choose **Tools > Intel Parallel Amplifier > Concurrency – Where is my concurrency poor?** The NQueens console window appears once again.

   The Inspector starts "Collecting Data." When the data-collection is done, in Visual Studio, a message appears telling you "Analysis successfully completed" and the progress-bar indicates that it is "finalizing results".

3. Interpret the resulting concurrency summary chart in the lower right of the Visual Studio window to see whether all cores were fully utilized. Mouse over the small boxes in the line-diagram for some insights.

Result: This `Step4-Parallel-Tune` project is a correctly threaded solution to the NQueens problem and should show good thread/multi-core utilization. Running this application by itself (**Debug > Start Without Debugging)** should show a significant time decrease in the NQueens console window versus what you saw in the `Step1-Serial-Hotspot` project.

If all available cores were not efficiently utilized, choose **Tools > Intel Parallel Amplifier > Locks and Waits – Where is my program waiting?** to see whether there are synchronization objects, I/O, or other places causing this underutilization. This analysis and finalization of results could take a bit longer than the analysis you did above in step 2.

Refer to the **Intel® Parallel Amplifier Getting Started Guide** for explanations and examples on how best to use and interpret Amplifiers graphs and data.

3 Select a Threading Technique

Intel® Parallel Studio offers multiple threading techniques to parallelize your application. To choose the technique that best suits the code to be parallelized, review the list below, weigh the benefits of each technique against the limitations, and take into account compatibility of the techniques with one another. You can find this information at http://software.intel.com/en-us/articles/intel-parallel-composer-parallelization-guide.

OpenMP* technology	A specification defined by OpenMP.org to support shared-memory parallel programming in C/C++ and Fortran through the use of application programming interface (API) and compiler directives. For more information, see [2] under Related Publications.
Intel® C++ language extensions	The Intel® C++ compiler language extensions provide a simple and easy way to get started with OpenMP*.
Intel® Threading Building Blocks (Intel® TBB)	A template-based run-time library providing a parallel programming model for C++ code. The library uses tasks to abstract threads. TBB simplifies multi-threading for scalable, multi-core performance. For more information, see [1] under Related Publications.
Use of a threaded library	The Intel® Integrated Performance Primitives (Intel® IPP) library provides a comprehensive set of application domain-specific functions. The library adds scalable parallelism to your application through the use of functions tuned and threaded for multi-core systems.

4 User and Reference Documentation

You can access Intel® Parallel Studio documentation:

- From the Windows* **Start** menu:
  - Choose **Intel Parallel Studio > Parallel Studio Documentation**
- From the main menu in Microsoft Visual Studio*:
  - Choose **Help > Intel Parallel Studio > Parallel Studio Help**
- From within Visual Studio:
  - Press F1 from any Intel® Parallel Studio window or toolbar to display context-sensitive help

NOTE: The first time you invoke help, it might take several minutes before Visual Studio indexes and displays the help contents.

- From the knowledge base on the web:
  — http://software.intel.com/en-us/articles/intel-parallel-studio/

Parallel Studio Component Getting Started Guides:

- Intel® Parallel Amplifier Getting Started Guide
- Intel® Parallel Composer Getting Started Guide
- Intel® Parallel Inspector Getting Started Guide
- Intel® Parallel Advisor Lite Getting Started Guide

You can access the Getting Started Guides:

- From the knowledge base on the web:
  — http://software.intel.com/en-us/articles/intel-parallel-studio/

- From the Windows* **Start** menu:
  — Choose **Intel Parallel Studio > Getting Started**

Related Publications

You are strongly encouraged to read the following books for in-depth understanding of threading. Each book discusses general concepts of parallel programming by explaining a particular programming technology:

Intel® Threading Building Blocks	[1] Reinders, James. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly, July 2007 (http://oreilly.com/catalog/9780596514808/)
OpenMP* technology	[2] Chapman, Barbara, Gabriele Jost, Ruud van der Pas, and David J. Kuck (foreword). *Using OpenMP: Portable Shared Memory Parallel Programming.* MIT Press, October 2007 (http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11387)

Legal Information