Intel(R) Parallel Advisor 2011 Getting Started Tutorial

Document Number: 323355-001US

World Wide Web: http://developer.intel.com

Legal Information

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to
http://www.intel.com/products/processor%5Fnumber/ for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Core Inside, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, skoool, the skoool logo, Sound Mark, The Journey Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Contents

Introducing the Intel Parallel Advisor 2011

Intel® Parallel Advisor guides you - a developer familiar with your C/C++ program source code - to prepare your code for the introduction of parallelism. By adding parallelism to certain parts of your program, you can reduce your program's execution time on multi-core systems. Intel Parallel Advisor (Advisor) guides you through a sequence of steps that help you identify which parts of your program can execute in parallel.

You use the Advisor tools within the context of Visual Studio. It helps you understand the characteristics of your program's serial code and make decisions about how to transform your serial program into a successful parallel program. Because Advisor tools analyze your running program, you need to build your program (or project) to provide a target executable for each Advisor tool. The major Advisor workflow steps include:

- You identify code regions where your program spends most of its time as possible places to add parallelism. To help you select the proposed places where you might add parallelism, use the Advisor Survey tool.
- You modify your source code with the Visual Studio code editor to add one-line Advisor annotations that mark the possible parallel places in your program. These annotations are recognized by the Advisor Suitability and Correctness tools, which predict your serial program's parallel behavior.
- You run the Suitability tool to analyze your running program to predict the approximate utilization of available cores for the selected parallel places. If the Suitability tool shows that one of the selected parallel places does not improve performance, modify or remove the corresponding annotations.
- You run the Correctness tool to analyze your running program to predict and locate data sharing problems that can occur when adding parallelism, such as data race conditions. The Correctness tool helps you examine the sources to locate the cause of the data sharing problems. After you modify your program to fix the sharing problems, run the Correctness tool again until you eliminate the potential data sharing problems. After you fix the remaining sharing problems, run the Suitability tool again.
- After you test your modified serial program, add code that actually introduces parallelism. You select a parallel framework - such as Intel® Cilk™ Plus or Intel® Threading Building Blocks (Intel® TBB) - and learn how to use it. After you add parallelism, test your program again.

Adding parallelism is typically an iterative process. After you add parallelism to one part of your program, you can repeat the Advisor workflow to locate additional places where you might add parallelism.

After building your parallel program, you can use other parts of the Intel Parallel Studio 2011 (Parallel Studio) to help you add code that enables parallelism, as well as check and tune your program.

Intel Parallel Advisor 2011 Tutorial

The Advisor tutorial shows you how to use the Advisor to find places in your program to add parallelism:

- Find Where to Add Parallelism

Check http://software.intel.com/en-us/articles/intel-software-product-tutorials for the following:

- Printable version (PDF) of all Parallel Studio tutorials

- Show Me Videos of the various Parallel Studio tools

See Also

- Prerequisites

Prerequisites

You need the following tools, skills, and knowledge to effectively use this tutorial.

NOTE. The steps and screen captures in this tutorial are based on the Visual Studio* 2005 IDE. They may differ slightly for other versions of the Visual Studio* IDE.

Required Tools

- Microsoft Visual Studio* (see the release notes for system requirements)
- Intel Parallel Advisor 2011
- Sample code and projects shipped with Intel Parallel Advisor 2011 (Advisor)
- Intel® Threading Building Blocks (Intel® TBB)
- Intel® Cilk™ Plus
- Advisor help

To install Advisor, follow the instructions in the Advisor Release Notes.

To install Intel® TBB and Intel Cilk Plus:

1. Install the Intel® Parallel Composer 2011, including Intel TBB and Intel Cilk Plus. For instructions, see the Composer Release Notes.
2. The Advisor samples work with the version of Intel TBB and Intel Cilk Plus included with Intel Parallel Composer 2011. The Advisor samples are set up to use Intel TBB version 3.0. If you do not have Intel® Parallel Composer 2011 available, see the Advisor release notes.

To install and set up the `nqueens_Advisor` sample used in this tutorial:

1. Locate the `nqueens_Advisor.zip` file in the `samples\<locale>` directory in the Parallel Studio 2011 installation folder. The default installation location is `C:\Program Files\Intel\Parallel Studio 2011\Samples`.
2. Copy the `nqueens_Advisor.zip` file to a writable directory or share on your system.
3. Use a suitable tool to open `nqueens_Advisor.zip` and extract the files in the compressed `nqueens_Advisor.zip` to a writable directory or share on your system.

To access Advisor Help, see the topic Getting Help and Support.

Required Skills and Knowledge

This tutorial and other Parallel Studio tutorials are designed for developers with the following skills and knowledge:

- At least a limited understanding of what parallel (multi-thread) execution means as well as the possible side-effects of parallel execution of parts of your program to shared data. To gain some knowledge about basic parallelism terms and concepts, read the Advisor help section Key Concepts.
- A basic understanding of the Microsoft Visual Studio* integrated development environment (IDE), including how to:
  - Open a project and solution.
  - Display the **Solution Explorer** and **Output** windows.
  - Build (compile and link) a project or solution.
  - Ensure a project or solution builds successfully.
  - View and modify the build settings, such as by using the **Configuration Manager** to specify a **Release** or **Debug** configuration build and modifying a project's **Property** settings to ensure that debug information will be present in the executable being built.
  - View Parallel Studio and Advisor help, which is available from the Visual Studio **Help** menu. With Visual Studio 2005 or 2008, you use the **Document Explorer** window. With Visual Studio 2010, you use the **Help Viewer** window.

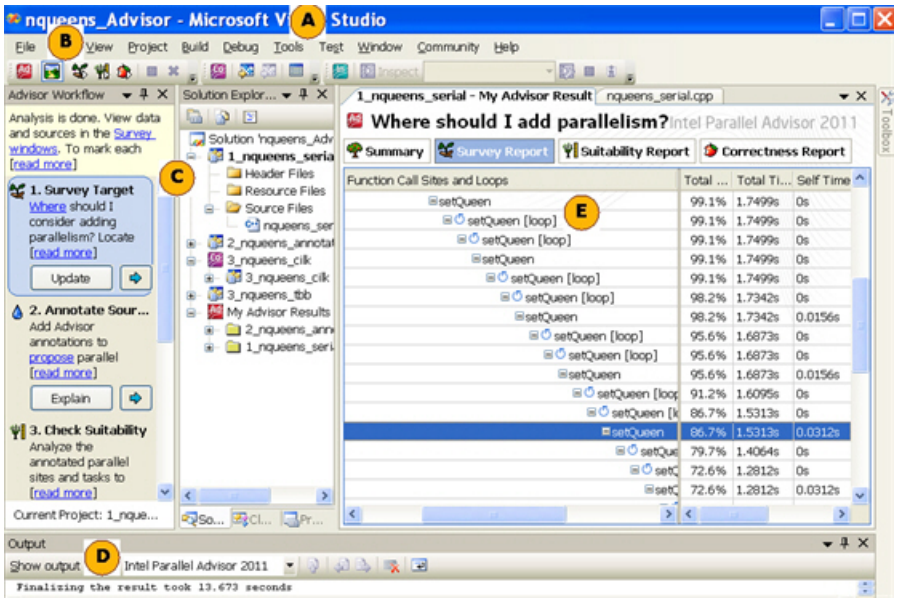
See Also

Related Information

- Navigation Quick Start
- Getting Help

Navigation Quick Start

Intel® Parallel Advisor 2011/Microsoft Visual Studio* 2005 Integration



A

From Visual Studio, use the:

- **Tools** > **Intel Parallel Advisor 2011** > **Open Advisor Workflow Window** menu item to display the **Advisor Workflow**.
- **Tools** > **Intel Parallel Advisor 2011** menu to launch Advisor tools. Advisor tools run with your program's executable (target) to collect data.

B

Use the **Advisor** toolbar to display the **Advisor Workflow** and launch Advisor tools.

C

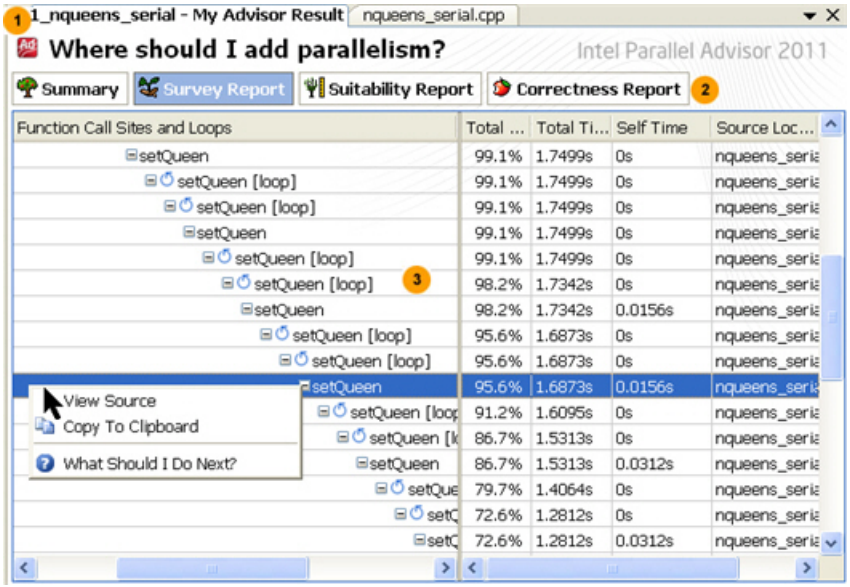
Use the:

- Visual Studio **Solution Explorer** to specify build settings, set the startup project, add files to projects, and so on.
- **Advisor Workflow** to help you launch Advisor tools as it guides you through the Advisor workflow.

D Use the **Output** window to view target execution and tool analysis output.

E Use **My Advisor Result** to view the data collected by Advisor tools. There is one **My Advisor Result** for each startup project.

My Advisor Result



1 Use the **My Advisor Result** to select the result for different Visual Studio projects. To display help tips, hover the mouse pointer over part of **My Advisor Result**.

2 Click buttons to change the window associated with one of the Advisor tools.

3 Use the windows in **My Advisor Result** to view and analyze the data collected by the Advisor tools. For more detail about a code region, display that tool's source view window. To do this, either double-click a function or loop name, or right-click it to display a pop-up menu (as shown above).

See Also

Find Where to Add Parallelism Using Intel(R) Parallel Advisor 2011

1

Learning Objectives

This tutorial shows how to use the Intel® Parallel Advisor 2011 (Advisor) to find places to add parallelism. This process includes a sequence of workflow steps that start with locating possible places to add parallelism, marking certain regions in your code by inserting one-line Advisor annotations that are recognized by two Advisor tools, and running your program with several Advisor tools to predict and mathematically model your serial program's potential parallel behavior.

The estimated completion time is 20-30 minutes.

After you complete this tutorial, you will:

- Know when and how to use Advisor tools.
- Understand how to use the **Advisor Workflow**.
- Be able to view and use output of Advisor tools in the **My Advisor Result**.
- Know why Advisor annotations are needed and how to add them to your sources using the Visual Studio code editor.

Key Terms and Concepts

Key Terms

annotation: Advisor annotations are one-line macros that identify certain information to Advisor tools, such as the location of parallel sites. You use the Visual Studio* code editor to insert annotations into your source code.

data race: A side-effect that can occur by adding parallelism to parts of your program. A data race occurs when multiple tasks read and write data at a shared memory location without coordinating those read and write operations. This can produce execution errors that are difficult to detect and reproduce.

hot spot: A small code region that consumes much of the program's run time. Hot spots can be identified by a profiler, such as the Advisor Survey tool or the Intel® Parallel Amplifier. Hot spots are code regions that help you select candidates for adding parallelism.

parallel framework: A combination of libraries, language features, or other software techniques that enable code for a program to execute in parallel. Examples include Intel® Threading Building Blocks and Intel® Cilk™ Plus, which are both included with Intel® Parallel Composer 2011. After you find possible places to add parallelism with Advisor, you identify *parallel sites* and their *tasks*, which are also used by high-level threading frameworks. In contrast, low-level threading APIs like Windows* Threads require that you directly create and control threads.

parallel code: A region of code that executes in parallel, where multiple threads are created and run at the same time during execution of that code region. Parallel code might be contained in one or more *parallel sites*. In contrast, *serial code* has a single thread of execution.

parallel site: A region of code that contains one or more tasks that execute in parallel. An effective parallel site typically contains a hot spot that consumes much of your program's CPU time. To distribute these frequently executed instructions to different *tasks* that can run at the same time, your parallel site is not usually located at the hot spot, but higher in the call graph. For example, a parallel site might be located in a function whose code eventually executes the hotspot. All tasks that were started within a site must complete before execution is allowed to proceed past the end of a site.

synchronization: Coordinating the execution of multiple threads. For example, a lock can be used to restrict access to shared data to prevent a data race.

target: An executable file. The Advisor tools run with your target executable to collect data and perform analysis about its execution characteristics.

task: A portion of code and its data that can be executed in a thread. One or more tasks are enclosed within a *parallel site*.

Key Concept: Advisor Annotations, Advisor and Visual Studio Tools

Advisor guides you through a sequence of steps in a workflow to help you add parallelism to your program. You work on the sequential (serial) version of your program, and use the Advisor tools to predict its parallel behavior. Certain Advisor tools require that you insert Advisor annotations into those parts of your program that will execute in parallel. This ability to experiment with potential parallelism helps you determine the feasibility of adding parallelism to parts of your program - before you add parallel code.

The following table summarizes the Advisor tools and how they use annotations:

Tool Name and Function	Use of Annotations
Survey: Based on where your program spends much of its time, this tool helps you identify places that could improve the program's run-time performance by executing as parallel code. It includes a source window that lets you view performance information about certain source code lines.	Does not need or look for annotations.
Visual Studio code editor: Use the code editor to insert, modify, and delete Advisor annotations in your source code. You also use the code editor to modify shared data use. After you prepare your program for parallelism and check its predicted parallel behavior, you replace annotations with parallel framework code that enables parallelism.	Helps you insert or modify Advisor annotations.

Tool Name and Function	Use of Annotations
Suitability: Displays approximate predicted performance based on the specified parallel site and task annotations. This tool also lets you mathematically model the expected performance of a theoretically ideal parallel machine, such as by specifying the number of cores. It includes a source window that lets you view performance information about source code lines.	Requires parallel site and task annotations. Recognizes most Advisor annotations.
Correctness: Displays the potential data sharing problems based on the parallel site, task, lock, and other annotations. It includes a source view that helps you locate the sharing problems in your source code. After you modify source code to fix sharing problems, run the Correctness and Suitability tools again.	Requires parallel site and task annotations. Recognizes all Advisor annotations.

As you insert Advisor annotations into your source code, you also need to add a source line that includes the `advisor-annotate.h` header file and use **Solution Explorer** to reference this header file.

Key Concept: Mixing Debug and Release Builds and Minimizing Data Sets for the Correctness Tool

Advisor assumes that Debug configuration builds are best for the **Correctness** tool and that Release configuration builds with debug symbol information are best for the **Survey** and **Suitability** tools. However, you need to do more than just change the configuration from Release to Debug before you run the **Correctness** tool. To allow reasonable execution times for the **Correctness** tool, consider one or more of the following to minimize the data set:

- Source code changes perhaps using conditional compilation, such as by using `#ifdef _DEBUG` - see the Advisor samples Heart and Newton.
- Using command-line arguments, perhaps by using project properties - see the use of the `nqueens_Advisor` sample described in Examine Potential Correctness Problems.
- Run-time selection of the workload, perhaps by asking the user to select a data set or by typing values.

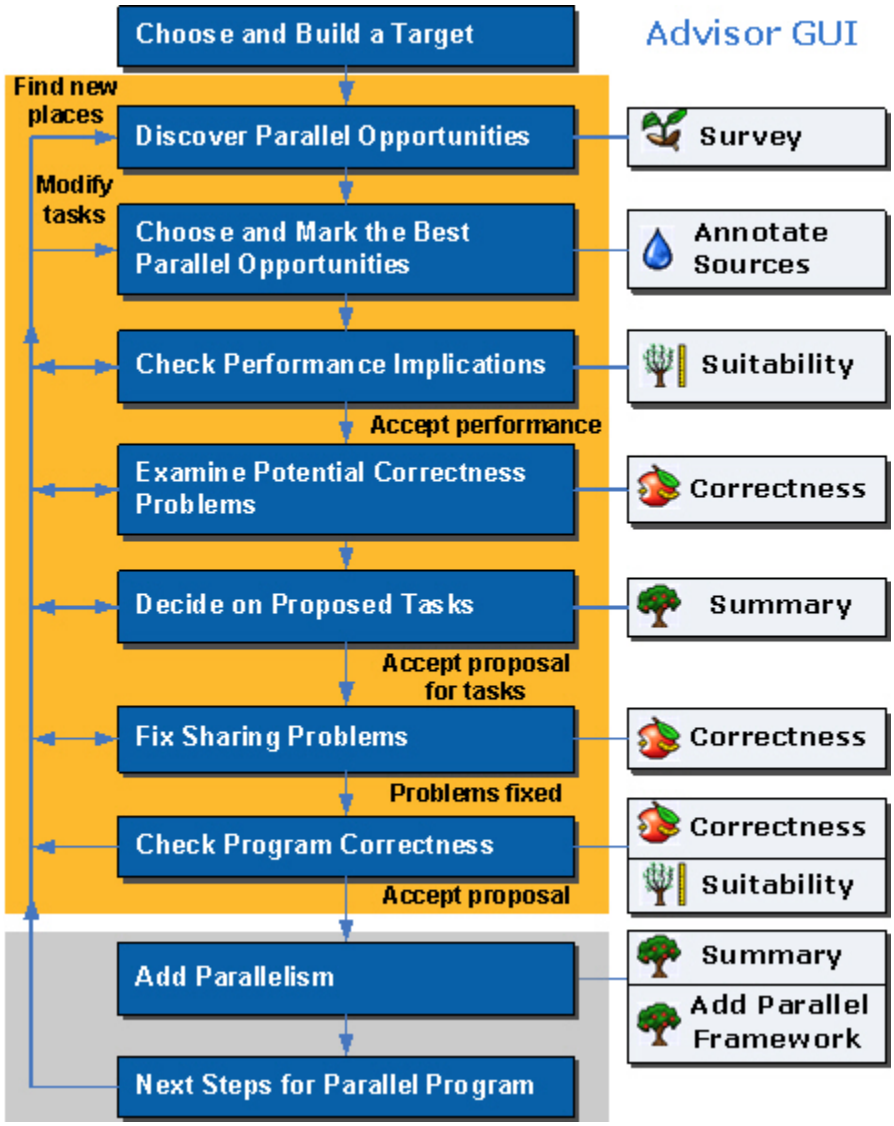
When you run the **Correctness** tool, it executes the target executable against the supplied data set. If you use a full data set, running the Correctness tool would take 50 to several hundred times longer to run your target than its normal execution time. The Correctness tool takes much longer program execution time because it needs to watch each instruction that reads or writes to memory.

Data set size and workload have a direct impact on target execution time and analysis speed. So, instead of choosing large, repetitive data sets, choose small, representative data sets that fully create threads with minimal to moderate work per thread. Your objective is to minimize run time while executing as many code paths as possible, especially in the parallel code regions. You need to minimize the redundant computation within each parallel task to the bare minimum needed for good code coverage. For more details, see the Advisor help.

Workflow Steps

The methodology used to add parallelism to a serial program has evolved from many years of experience parallelizing numerous application programs of different types. Advisor and other parts of Intel Parallel Studio provide tools that help you make informed decisions about adding parallelism and help you automate the detailed task of adding, checking, and tuning parallel parts of a program. In addition to supporting the overall parallelism methodology, the Advisor workflow includes specific steps to indicate when to use Advisor tools.

The Advisor workflow figure shown below is an expanded version of the **Advisor Workflow** tab. To display a brief explanation, hover the mouse pointer over a workflow step. Click on a step to jump to the corresponding topic for details:



In this figure, the yellow shading (near the top) represents the workflow steps where you analyze and restructure the serial program. The Advisor tools and the Advisor user interface appear to the right of the workflow steps.

The process of adding parallelism is iterative. Although the following workflow steps are sequential, you may need to return to a previous step as you learn more about your program's predicted parallel behavior. The workflow steps follow:

- Choose and Build a Target
- Discover Parallel Opportunities
- Choose and Mark the Best Parallel Opportunities
- Check Performance Implications

Choose and Build a Target

All Advisor tools analyze your running program, so you need to build a target executable for these tools.

The Survey and Suitability tools work best with a **Release** configuration build, whereas the Correctness tool requires a **Debug** configuration build.

Use the Visual Studio **Configuration Manager** to select whether a Release or Debug configuration will be built for the current project:

- Select a **Release** configuration with debug symbols to build a target for the Survey and Suitability tools.
- Select a **Debug** configuration (no optimization) to build a target for the Correctness tool.

The tools require that the target executable include debug (symbol) information. This property is specified with the Advisor samples. With your own program, you must check that the project build includes debug information in the project **Properties**.

This tutorial uses different projects in the **nqueens_Advisor** sample. Specific instructions are provided in the following sections.

Recap

A **target** is the executable that you run with Advisor tools. You need to set the startup project, specify the build Properties, and build the target as required by each Advisor tool, such as choosing either a Debug or Release configuration. In the next step, you will use one of the Advisor tools that runs a target.

Key Terms and Concepts

- Term: target
- Concept: Advisor Annotations, Advisor and Visual Studio Tools

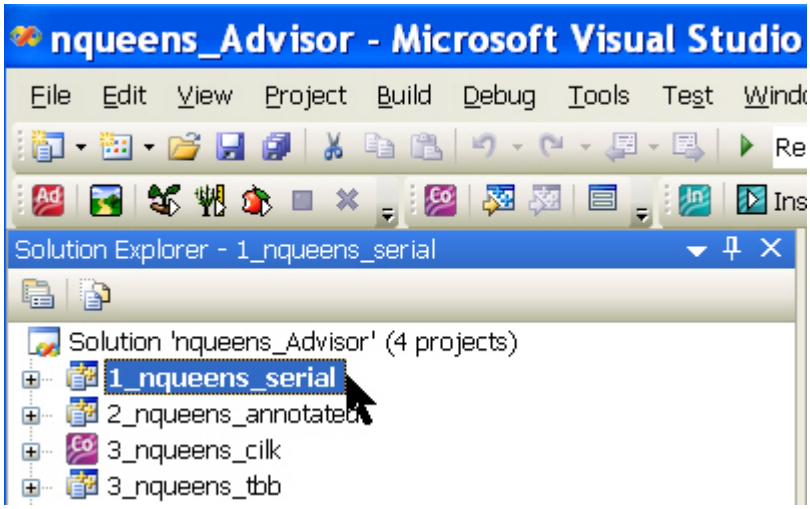
Discover Parallel Opportunities

To locate small code regions that consume a significant amount of the program's run time, use the Survey tool to profile your running program. To do this:

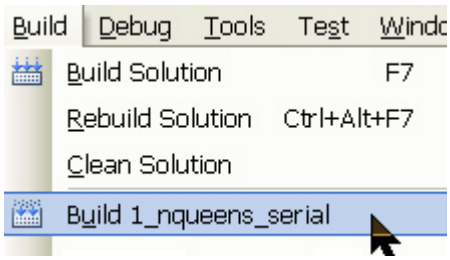
- Build a target executable for the Survey tool using a Release configuration.
- Display the Advisor Workflow.
- Run the Survey tool.

Build a Target for the Survey Tool

1. From the Microsoft Visual Studio*, choose **File** > **Open** > **Project/Solution**.
2. In the **Open Project** dialog, navigate to the directory where you extracted the `nqueens_Advisor` sample, and open the file `nqueens_Advisor.sln`. To locate and extract the `nqueens_Advisor` sample, see Prerequisites.
3. In the **Solution Explorer**, right-click the project **1_nqueens_serial**:



4. From the pop-up menu, select **Set as Startup Project**.
5. From the **Build** menu, select **Configuration Manager**.
6. Select a **Release** configuration build for this project.
7. In the **Build** menu, select **Build 1_nqueens_serial**:



8. View the **Output** window for any build errors.
9. To see the sample's command-line output, run this program by choosing **Debug** > **Start Without Debugging**. For example:

```
Usage: nqueens_serial boardSize [default is 13].
Starting nqueens (1_nqueens_serial) solver for size 13...
Number of solutions: 73712
Correct result!

Calculations took 1743ms.
Press any key to continue . . .
```

Display the Advisor Workflow Tab

The **Advisor Workflow** helps guide you through the Advisor workflow.

To display the **Advisor Workflow**, do one of the following:

- In the Advisor toolbar, click the  icon.
- In the Visual Studio **Tools** menu, choose **Intel Parallel Advisor 2011** > **Open Advisor Workflow**.

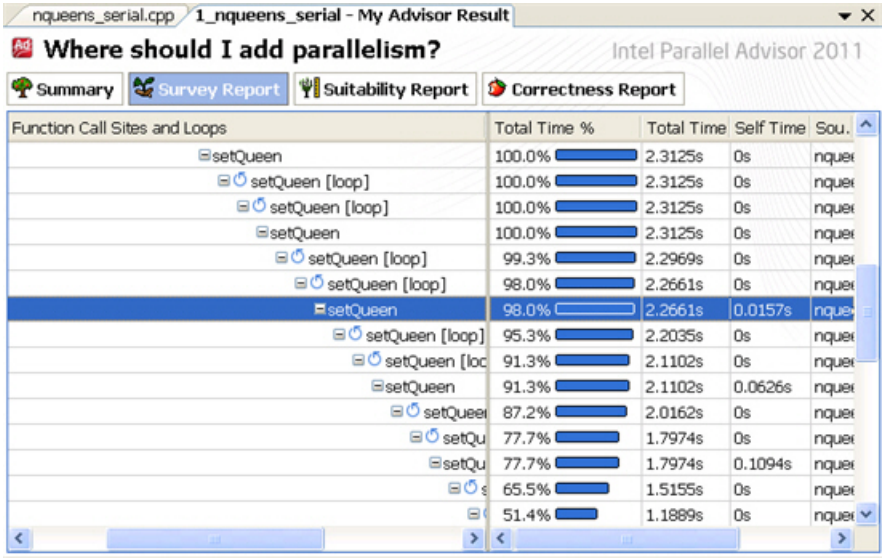
The top part of the **Advisor Workflow** contains instructions, in a light yellow background. These instructions include relevant links to the Advisor help. In the **Advisor Workflow**, follow the steps from top-to-bottom.

Run the Survey Tool

To run the Advisor Survey tool, do one of the following:

- In the **Advisor Workflow**, click the **Start** button below **1. Survey Target**.
- Click the  icon in the **Tools** > **Intel Parallel Advisor 2011** menu or the Advisor toolbar.

The Survey tool runs your program to collect data and analyze its run-time characteristics. A command window briefly appears while the program runs. After the Survey tool finalizes the data, data appears in the **My Advisor Result** in the **Survey Report** window. For example:



The **Survey Report** window provides a top-down view of an expanded call graph with loops. The function `setQueen()` recursively calls itself, so it appears multiple times in the **Function Call Sites and Loops** column, along with its loops (icon). Looking at the values in the **Total Time** and the **Self Time** columns, you see that the `setQueen()` function is where this program spends most of its time. Click the icon next to a function or loop's name to see the functions or loops it calls. You have just identified the hot part of the call graph in the sample program, the `setQueen()` function.

After you run a tool, the **Start** button becomes an **Update** button to indicate that data has previously been collected.

Recap

You built a target and ran the Survey tool to learn where your program spends its time. Now that you have found some possible places to add parallelism, your next step is to insert Advisor annotations to mark those places.

Key Terms and Concepts

- Term: hot spot

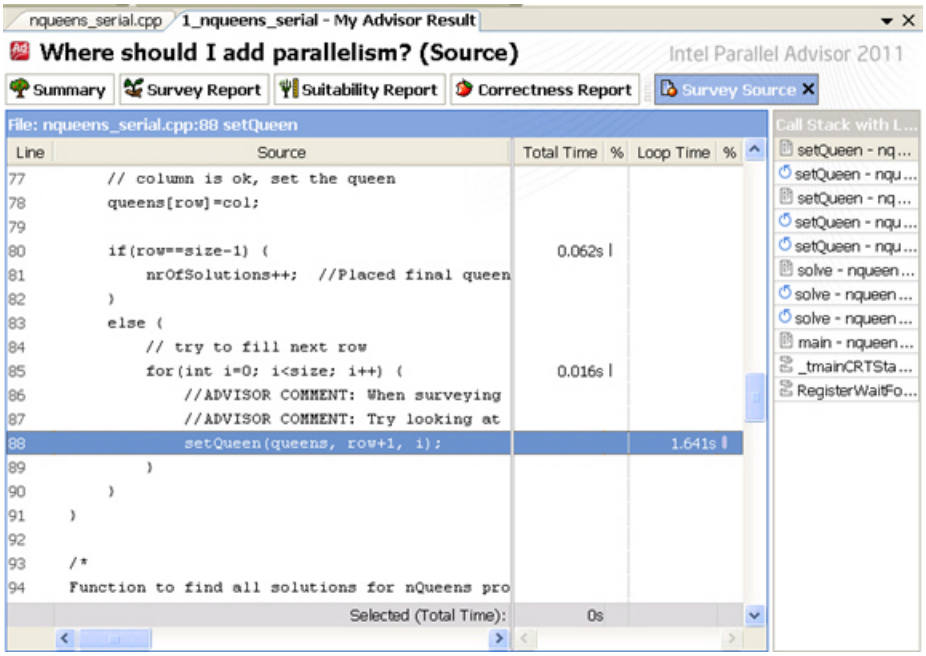
Choose and Mark the Best Parallel Opportunities

Using the **Survey** tool data, choose places in your program to add parallelism and mark those places by inserting Advisor annotations. To do this:

- Display sources in the Survey Source window.
- Find where to add Advisor parallel site and task annotations.
- Add parallel site and task annotations.

Display the Sources in the Survey Source Window

Double-click a line (or right-click a line and select **View Source**) in the **Survey Report** window for the hot function `setQueen()` to display the **Survey Source** window:



The recursive function call to `setQueens` uses nearly all of this program's CPU time. You can see the CPU time for individual source lines using the **Survey Source**:

- The Total Time column shows the predicted time executing this statement or in functions invoked from this statement.
- The Loop Time column shows a summary of the Total Time over all basic blocks in this loop. It is displayed for one statement in the loop, such as the loop header.

Find Where to Add Advisor Parallel Site and Task Annotations

You want to distribute frequently executed instructions to different tasks that can run at the same time. So rather than looking only at the function consuming all the time, you must also examine all the functions in the call tree from `main()` to the hot routine `setQueen()`. In this case, the `main()` function accepts command-line arguments, initializes an array, and calls the function `solve()`. The function `solve()` calls `setQueen()`, which recursively calls itself.

Either use the **Survey Source** window (double-click or right-click a line) or the code editor in the **Solution Explorer** to open the source file `nqueens_serial.cpp`. Get familiar with the code execution paths and data use.

The `nqueens_Advisor` sample includes lines that begin with `//ADVISOR COMMENT`. In `nqueens_serial.cpp`, notice that:

- The annotations for the parallel site include the loop in the `solve()` function (shown below). Also, the call to `setQueen()` is a task within that parallel site.

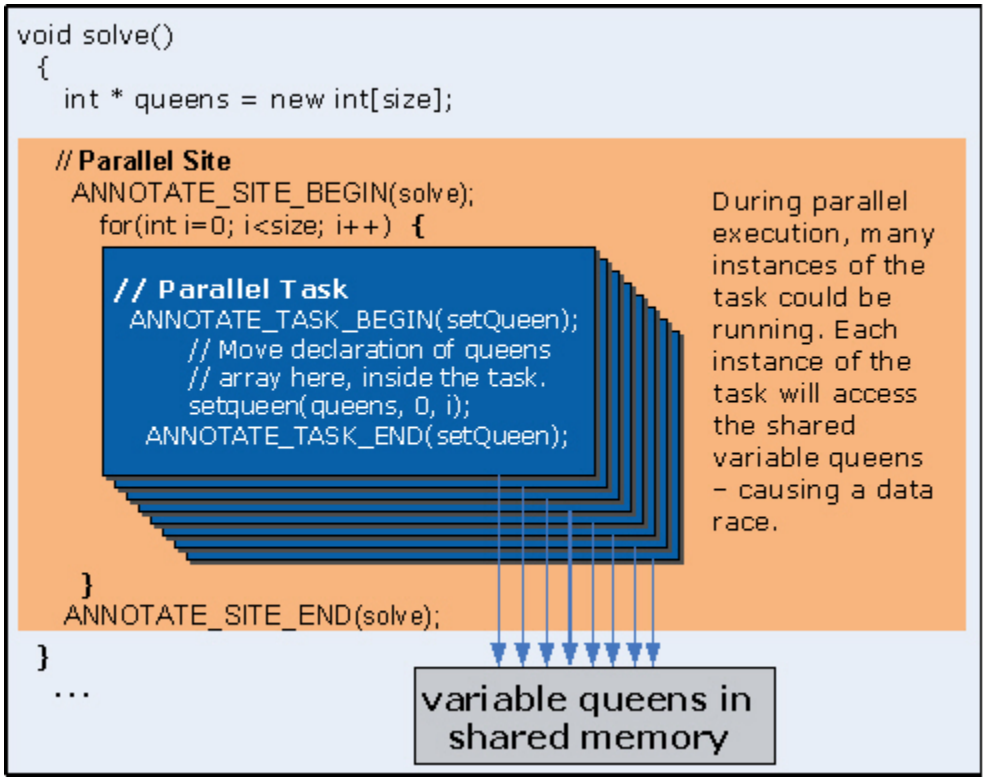
- The `#include` line (not shown below) includes the Advisor header file. This is needed because annotations are present in this source file.
- For your convenience, the annotations are commented out in this version of this project's source file. These annotations have been uncommented in the next project's source file.

```
void solve() {
 int * queens = new int[size]; //array representing queens placed on a chess board.  Index ...

 //ADVISOR COMMENT: When surveying this is the top function below main.  This for() loop is ...
 //ADVISOR COMMENT: Uncomment the four annotations below to model parallelizing the body of ...
 //ADVISOR COMMENT: Don't forget to uncomment the #include <advisor-annotate.h> at the top

 //ANNOTATE_SITE_BEGIN(solve)
  for(int i=0; i<size; i++) {
   // try all positions in first row
   //ANNOTATE_TASK_BEGIN(setQueen)
    setQueen(queens, 0, i);
   //ANNOTATE_TASK_END(setQueen)
  }
 //ANNOTATE_SITE_END(solve)
}
```

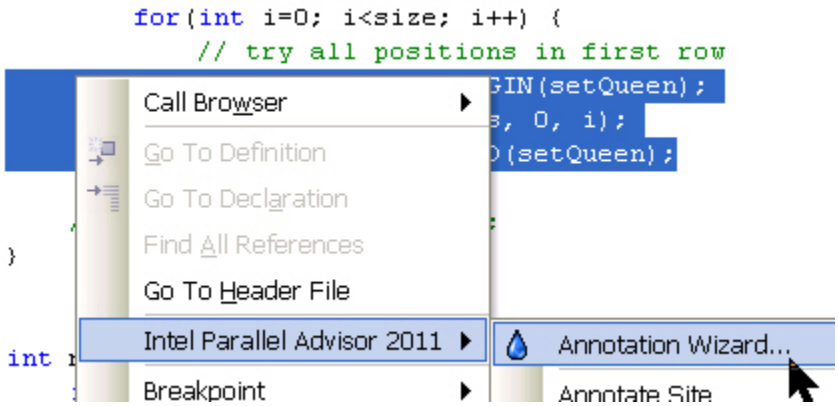
The following figure illustrates the original `nqueens_Advisor` sample code to show the task (dark blue background) and its enclosing parallel site (orange background).



Adding Parallel Site and Task Annotations to Your Program

When adding annotations to your own program, remember to include `advisor-annotate.h`. For help completing this step, click the **Explain** button below **2. Annotate Sources** in the **Advisor Workflow** tab. To insert annotations into your sources:

1. Within the Visual Studio code editor, select a code region.
2. Right-click and select the Advisor **Annotation Wizard** from the pop-up menu:



For instructions on using the **Annotation Wizard**, view the wizard's dialog boxes.

Recap

You found some possible places where you might add parallelism. The next project provides these annotations, which are used by the Advisor Suitability and Correctness tools to measure performance and predict data sharing problems for the parallel sites and tasks.

Key Terms and Concepts

- Term: annotation
- Concept: Advisor Annotations, Advisor and Visual Studio Tools

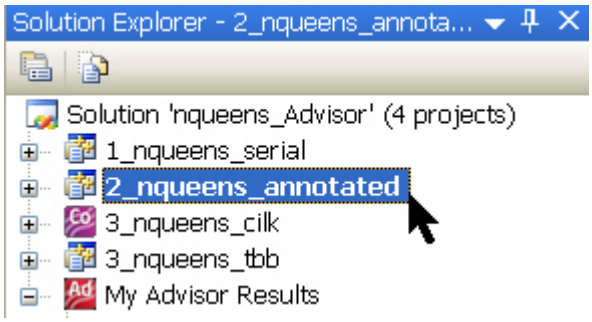
Check Performance Implications

Examine your program to measure the approximate predicted performance. You run the Suitability tool, which uses the Advisor annotations to predict your program's approximate parallel performance. To do this:

- Build a target executable for the Suitability tool using a Release configuration.
- Run the Suitability tool.
- Adjust parameters for mathematical modeling.
- View the Scalability of Maximum Site Gain graph.
- Use the Suitability Source window and the Visual Studio code editor.

Build a Target for the Suitability Tool

1. In **Solution Explorer**, right-click the project **2_nqueens_annotated**:



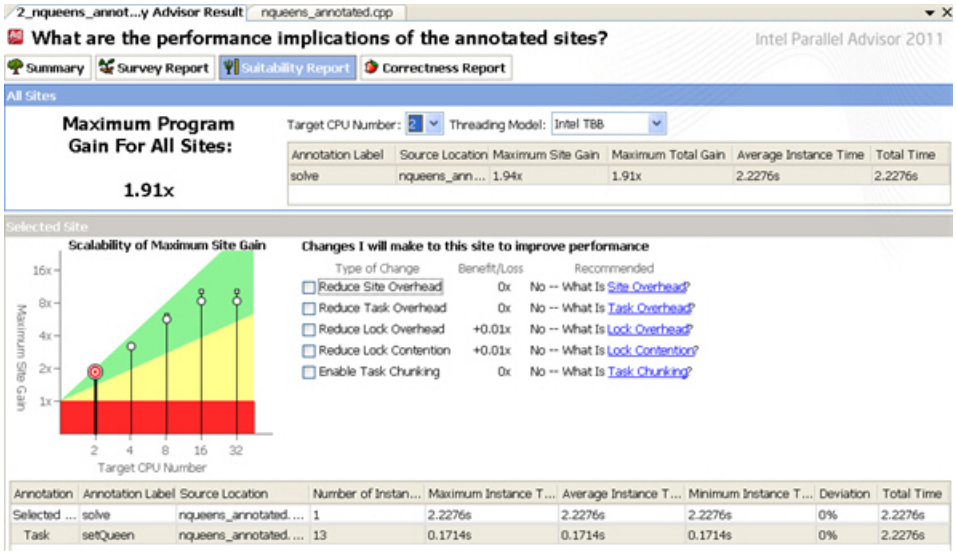
2. From the pop-up menu, select **Set as Startup Project**.
3. From the **Build** menu, select **Configuration Manager**.
4. Select a **Release** configuration build for this project.
5. In the **Build** menu, select **Build 2_nqueens_annotated**.
6. View the **Output** window for any build errors.

Run the Suitability Tool

To run the Advisor Suitability tool, do one of the following:

- Open the **Advisor Workflow** and click the **Start** button below **3. Check Suitability**.
- Click the  icon in the Visual Studio **Tools** > **Intel Parallel Advisor 2011** menu or the Advisor toolbar.

The following shows the **Suitability Report** window after the data collection and finalization completes:



To stop or cancel an Advisor collection, click the  or  buttons in the **Tools** > **Intel Parallel Advisor 2011** menu or the Advisor toolbar, or click the **Cancel** button.

The **All Sites** pane in the upper part of the window shows there is one parallel site annotation pair present in source file `nqueens_annotated.cpp` with the label `solve`. The column **Maximum Site Gain** indicates the approximate predicted gain for the single parallel site of **1.94x** based on the modeling parameters.

The value under **Maximum Program Gain for All Sites** indicates the possible maximum performance for all sites. In this case, the value **1.91x** for the original **Target CPU Number** number of **2** indicates a good run-time gain.

NOTE. Depending on the vertical size available, the **Selected Site** pane may not display both the scalability graph and the annotation grid. In this case, click the displayed link to toggle between displaying the scalability graph or the annotation grid.

Adjust Parameters for Mathematical Modeling

In the **All Sites** pane, you can change the values for the **Target CPU Number** and **Threading Model** items to see how much changing the values influences the **Maximum Site Gain** value for this site. For example, change the **Target CPU Number** from 2 to 32 and view the difference in the estimated **Maximum Site Gain** value, which is shown above as **1.994** for the original **Target CPU Number** number of **2**. Similarly, the **Selected Site** pane in the lower part of the window lists five items that can also influence the **Maximum Site Gain** value for this site.

A **Maximum Site Gain** value of less than 1.0 indicates a decrease in performance. If this occurs with your program, consider moving or removing the annotations for that site.

The **Selected Site** pane shows information about the tasks and locks for the selected parallel site. Under the **Instances** column, the instances of the **Task** were **13**, which is the default data set board size for this sample's Release configuration.

Look at the **Average Time** value for the task in the **Selected Site** pane. The **Average Time** should be large enough to overcome the overhead of starting and ending a task (typically 0.00001 second). The **Enable Chunking** in the lower pane is useful when the **Average Time** is lower than task start/stop overhead time. This feature supported by Intel® TBB and Intel® Cilk™ Plus is useful when you have small tasks.

Viewing the Scalability of Maximum Site Gain Graph

In the **Selected Site** pane, a graph summarizes the **Scalability of Maximum Site Gain** for the selected site. The number of cores appears on the X axis and the program's run-time performance gain appears in on the Y axis. The hyphens represent the plot for this site, which is in the green shaded area and indicate good results. In this case, notice that because the data set size is 13, the addition of more cores does not help speed-up beyond 13 cores, because there are only 13 tasks. Near the top of vertical lines for each CPU number, you may see a box and a circle that indicate the minimum and maximum predicted gain values. If the minimum-maximum range appears in the yellow-shaded area, you should investigate how the results can be improved.

Using the Suitability Source Window and the Visual Studio code editor

To view the sources associated with a task or lock, double-click (or right-click and select **View Source**) a line to display the **Suitability Source** window. After you determine that this is the correct location, double-click a line again to display the Visual Studio code editor, allowing you to modify the source file.

To return from **Suitability Source** to the corresponding **Suitability Report** window, click the **Suitability Report** button.

Recap

The Suitability tool measures your program's annotated parallel sites and tasks to provide you with estimated approximate run-time performance times. You can mathematically model the performance by changing certain parameters. In the next step, you predict whether proposing these parallel sites and tasks exposes data sharing problems, such as data races.

Key Terms and Concepts

- Term: parallel site
- Term: task
- Concept: Advisor Annotations, Advisor and Visual Studio Tools

Examine Potential Correctness Problems

Examine your program to predict likely data sharing problems by using the Correctness tool. Because the Correctness tool watches multiple memory locations as your program executes, you need to minimize the input data set. To do this:

- Build a target executable for the Correctness tool using a Debug configuration.
- Provide a small but representable data set .
- Run the Correctness tool.
- View the Correctness Report and Correctness Source windows.

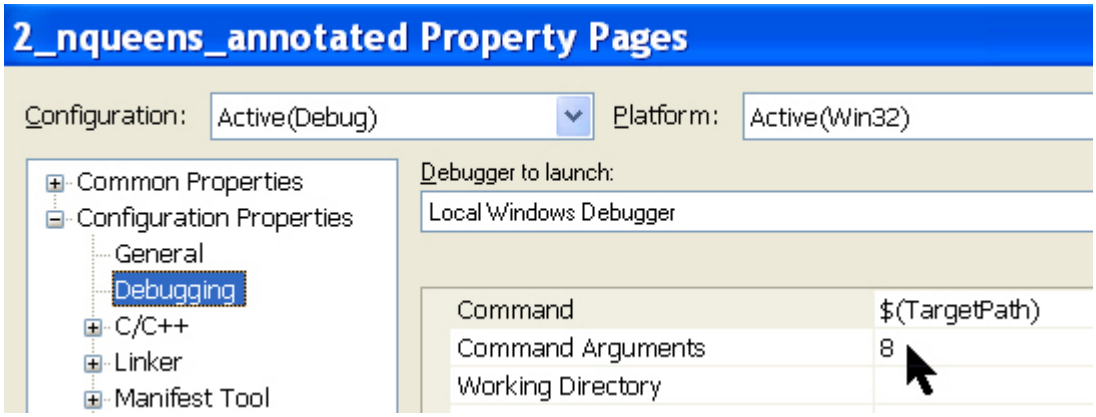
Build a Target for the Correctness Tool

1. In **Solution Explorer**, right-click the project **2_nqueens_annotated**.
2. From the pop-up menu, select **Set as Startup Project**.
3. From the **Build** menu, select **Configuration Manager**.
4. Select a **Debug** configuration build for this project.
5. In the **Build** menu, select **Build 2_nqueens_annotated**.
6. View the **Output** window for any build errors.

Reduce the Input Data Set

When you run the Correctness tool, it executes the target executable against the supplied data set. If you supplied a full data set for the Correctness tool, this would require a run time of 50 to several hundred times longer than the normal execution time of the target executable. So, for the next example, we need to reduce the data set by using the project's properties. You may have noticed that the `main()` entry point accepts command arguments and we specify the arguments without requiring a rebuild. The default board size for a Debug configuration build is 8, so you should verify the value of **8** appears below:

1. In **Solution Explorer**, right-click the project **2_nqueens_annotated**.
2. From the pop-up menu, select **Properties**.
3. In **Configuration Properties** > **Debugging** > **Command Arguments**, verify that the board size's value is **8**:



4. Click **OK**.

Run the Correctness Tool

To run the Advisor Correctness tool, do one of the following:

- In **Advisor Workflow**, click the **Start** button below **4. Check Correctness**.
- Click the  icon in the Advisor toolbar or the Visual Studio **Tools** > **Intel Parallel Advisor 2011** menu.

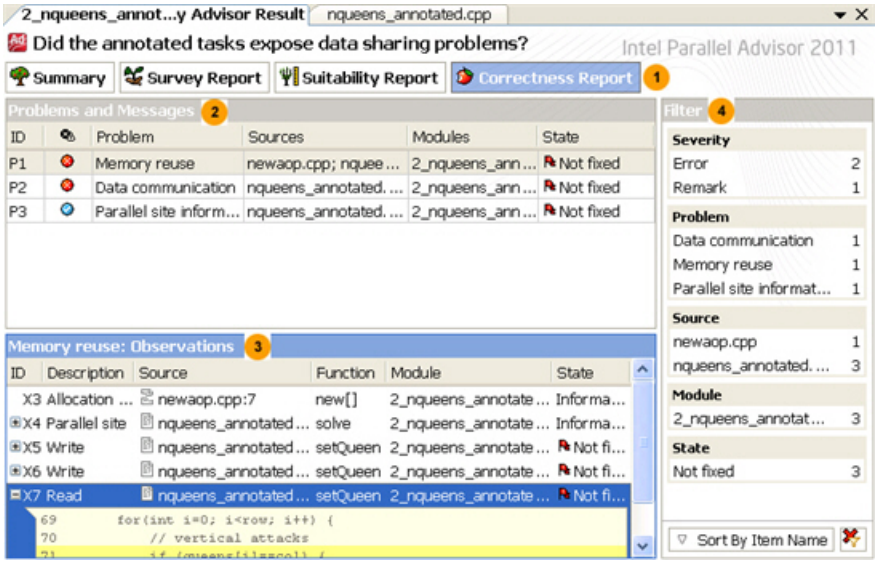
The Correctness tool runs with your program to collect data and analyze its run-time characteristics. A command window appears and displays text when the `nqueens_Advisor` program completes:

```
Starting nqueens (2_nqueens_annotated) solver for size 8...
Number of solutions: 92
Correct result!

Calculations took 1875ms.
```

To stop or cancel an Advisor collection, click the  or  buttons in the **Tools** > **Intel Parallel Advisor 2011** menu or the Advisor toolbar, or click the **Cancel** button.

After the Correctness tool finishes collecting and finalizing the data, the following appears in the **My Advisor Result** tab's **Correctness Report** window:



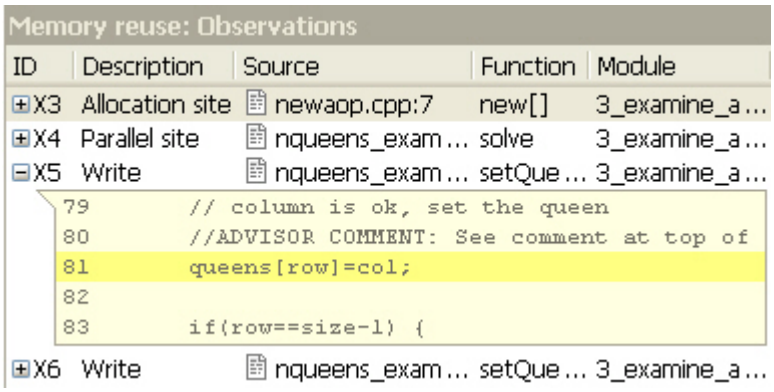
1	The **Correctness Report** window is the starting point for viewing potential problems.
2	The **Problems and Messages** pane shows the observed problems, prioritized by severity and size. In this case, there are two problems and one informational remark.
3	For the currently selected Problem, details appear in the **Observations** pane (lower-left corner). In this case, details appear for the **Memory reuse** problem.
4	The **Filter** pane provides a summary of the problems. It also lets you toggle between displaying a subset of the problems found or **all** problems.

View Correctness and Correctness Source Output

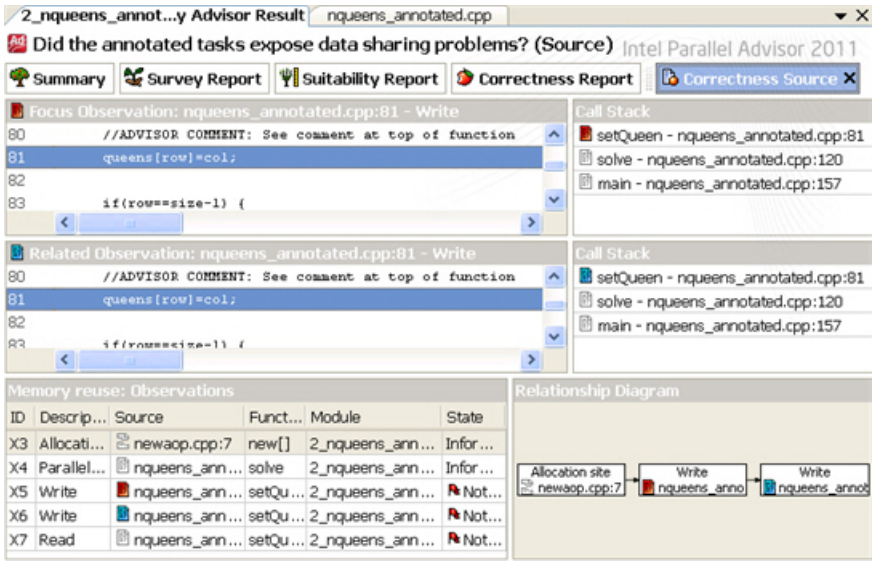
The **Problems and Messages** pane lists the data sharing problems found. The column with the icon lists the severity of each problem, such as error, warning, or informational remark. Click a line under the **Problems** column and the observations associated with that problem appear in the **Observations** pane.

To further interpret the result data, do one of the following:

- Click the icon next to one of the **Observations**, such as **X5**, to view or hide a few lines of its source code:



- Double-click one of the **Observations** (or right-click and select **View Source**), such as **X5**. This opens the **Correctness Source** window, which lets you view source for the **Focus Observation** (red icon), **Related Observation** (blue icon), their call stacks, a list of all **Observations**, and a diagram of **Observation Relationships**.



Recap

You built a target and ran the Correctness tool to check for data sharing problems. In the next step, you need to fix these sharing problems by modifying the sources and running the Correctness tool again to see whether they are fixed.

Key Terms and Concepts

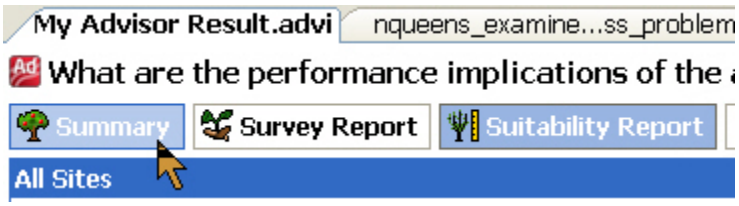
- Term: data race
- Concept: Minimizing Data Sets for the Correctness Tool

Decide on the Proposed Tasks

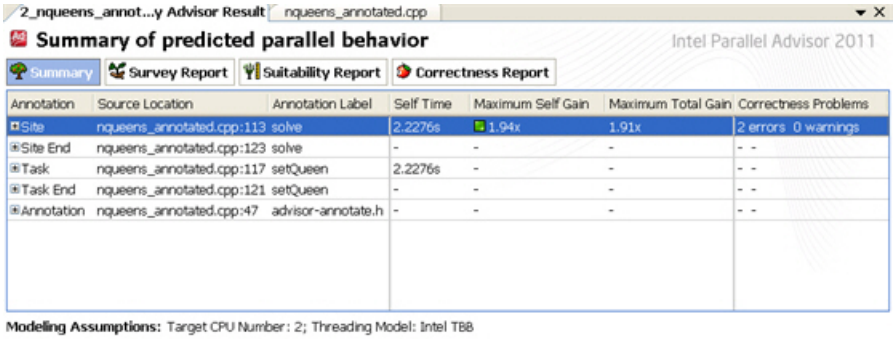
The **Summary** window displays your annotations and a high-level summary of Suitability tool performance data and Correctness tool data sharing problems side-by-side.

View the Summary Window

The **Summary** window displays the annotations in your current project. After you run the Suitability and Correctness tools, it shows the performance benefit and cost of fixing sharing problems for your sites and tasks. To display the **Summary** window, click the **Summary** button:

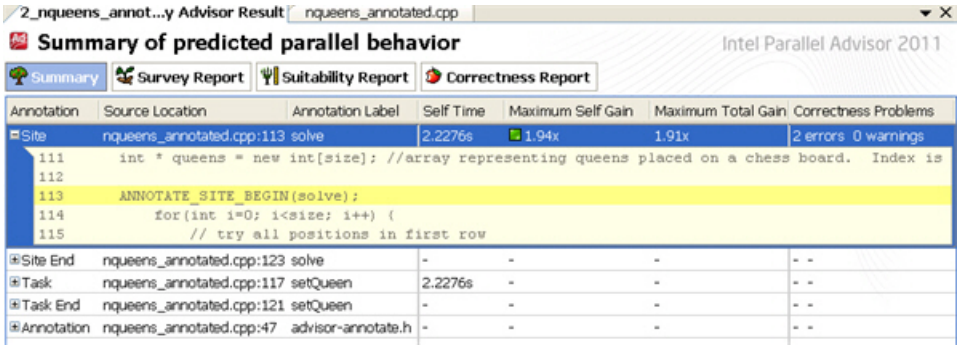


The **Summary** window appears:



The first Site has a good **Maximum Self Gain** based on the default 2 CPUs, and it has only 2 data sharing problems (and 0 warnings). Because this site provides a good performance benefit and a small number of sharing problems, keep your proposed site and task code regions.

On a specific line, click the plus icon ⊞ to display a source code snippet that shows the annotation. For example:



Like the other windows, you can click a column heading to sort values. In the screen shown above, sorting occurs using the Annotation column in ascending order.

This window also lets you drill down to see details. Depending on which column you double-click (or right-click and select **View Source**), you will see different results:

- Double-click in the first three columns (Annotation to Annotation Label) to view sources in the code editor. To return from the code editor to the **Summary** window, click **My Advisor Result** and **Summary**.
- Double-click the Self Time, Maximum Self Gain, or Maximum Total Gain columns to display the **Suitability Report** window.
- Double-click the Correctness Problem column to display the **Correctness Report** window.

As you fix sharing problems, you can run the Correctness and Suitability tools to update the data displayed in the **Summary** window. It usually takes multiple iterations to finally decide on your sites, tasks, and locks.

Recap

The **Summary** window lets you view a summary of your annotations as well as the predicted performance gain and cost of fixing sharing problems for *each* site and task. In the next step, you will fix the sharing problems.

Key Terms and Concepts

- Term: parallel site
- Term: task
- Concept: Advisor Annotations, Advisor and Visual Studio Tools

Fix Sharing Problems

View the Correctness Report and Correctness Source windows to help you fix the reported data sharing problems. To do this:

- Examine the Correctness Report and Correctness Source windows.
- Fix the Memory reuse problem.
- Fix the Data communication problem.

Examine Correctness Report and Correctness Source Windows

The Correctness tool reports two problems:

- **Memory reuse** problem type. This is located in the `solve()` function.
- **Data communication** problem type. This is located in the `setQueen()` function.

Unlike problems reported in serial programming, which often have a single cause, problems in parallel programming usually involve multiple code regions. For example, consider the case where three different code regions allocate, initialize, and access the same memory location - and each is located in two or more tasks. In the parallel version of the program, these allocate-initialize-access actions could occur at the same time. Thus, the multiple code regions have *relationships*, so the **Correctness Report** window shows a **Focus observation** and **Related observations**.

To open the Visual Studio code editor to the relevant source code:

1. In the **Problems and Messages** pane, double-click one of the two problems with a severity of Error ( ).
2. View the source code in the **Correctness Source** window. This is the **Focus Observation** pane. You can view the **Related Observations** or simply double-click the source code to display the Visual Studio code editor at the same position.
3. In each case, scroll up to locate the function containing the source code:
   - The **Memory reuse** problem originates in the `solve()` function.
   - The **Data communication** problem is located in the `setQueen()` function.
4. Notice the Advisor comments that explain the solution to both problems.
5. To return from a source view to the corresponding **Correctness Report** window, click the **Correctness Report** button in **My Advisor Results** for that project.

Fix the Memory Reuse Sharing Problem

To fix the **Memory reuse** problem located in the `solve()` function, examine that function's code:

```
void solve() {

 //ADVISOR COMMENT: Remove the following declaration of the queens array and uncomment the declaration just
 before the call to setQueen
 //ADVISOR COMMENT: This privatizes the queens array to each task and eliminates the incidental sharing
 int * queens = new int[size]; //array representing queens placed on a chess board.  Index is row position,
 value is column

 ANNOTATE_SITE_BEGIN(solve)
  for(int i=0; i<size; i++) {
   // try all positions in first row
   // create separate array for each recursion
   ANNOTATE_TASK_BEGIN(setQueen)
    //int * queens = new int[size]; //array representing queens placed on a chess board.  Index is row
position, value is column
    //ADVISOR COMMENT: This is incidental sharing because all the tasks are using the same copy of "queens"

    setQueen(queens, 0, i);
   ANNOTATE_TASK_END(setQueen)
  }
 ANNOTATE_SITE_END(solve)
}
```

The following line below the two comment lines allocates the array queens and is located outside the parallel site. This placement causes the site code and each task to share the same array! This is called incidental or accidental data sharing:

```
 //ADVISOR COMMENT: Remove the following declaration of the queens array and uncomment the declaration just
 before the call to setQueen
 //ADVISOR COMMENT: This privatizes the queens array to each task and eliminates the incidental sharing
 int * queens = new int[size]; //array representing queens placed on a chess board.  Index is row position,
 value is column
```

Instead of sharing this array, each task should have its own private copy of this array, so you need to comment out the line shown above and uncomment the same code line just below the ANNOTATE_TASK_BEGIN line:

```
 //ADVISOR COMMENT: This privatizes the queens array to each task and eliminates the incidental sharing
 // int * queens = new int[size]; //array representing queens placed on a chess board.  Index is row position,
 value is column

 ANNOTATE_SITE_BEGIN(solve)
  for(int i=0; i<size; i++) {
   // try all positions in first row
   // create separate array for each recursion
   ANNOTATE_TASK_BEGIN(setQueen)
    int * queens = new int[size]; //array representing queens placed on a chess board.  Index is row position,
 value is column
    //ADVISOR COMMENT: This is incidental sharing because all the tasks are using the same copy of "queens"

    setQueen(queens, 0, i);
   ANNOTATE_TASK_END(setQueen)
  }
 ANNOTATE_SITE_END(solve)
}
```

Fix the Data Communication Sharing Problem

To fix the **Data communication** problem located in the `setQueen()` function, examine that function's code:

```
 // column is ok, set the queen
 //ADVISOR COMMENT: See comment at top of function
 queens[row]=col;

 if(row==size-1) {


  //ADVISOR COMMENT: Uncomment the following two annotations to lock the access to nrOfSolutions and fix
the race condition

  //ANNOTATE_LOCK_ACQUIRE(0)
  //ADVISOR COMMENT: This is a race condition because multiple tasks may try and increment the variable at
 the same time
  nrOfSolutions++;  //Placed final queen, found a solution!
  //ANNOTATE_LOCK_RELEASE(0)
 }
 else {
  // try to fill next row
  for(int i=0; i<size; i++) {
   setQueen(queens, row+1, i);
  }
 }
```

The line that increments the variable `nrOfSolutions` is located in the recursively called `setQueen()` function, which is called from multiple tasks. This causes a data race, or a **Data communication** problem. The accesses to this data must be synchronized. You can uncomment the pair of `ANNOTATE_LOCK_ACQUIRE(0)` and `ANNOTATE_LOCK_RELEASE(0)` annotation lines, or you can use the following procedure to insert Advisor annotations using the Visual Studio code editor:

1. In the code editor, select the range of code lines to be included by the annotation. For example:

```
//ANNOTATE_LOCK_ACQUIRE(0);
//ADVISOR COMMENT: This is a race condition because multiple tasks may
nrOfSolutions++;  //Placed final queen, found a solution!
//ANNOTATE_LOCK_RELEASE(0);
```

2. Right click to display a pop-up menu.
3. From the pop-up menu, select **Intel Parallel Advisor 2011** > **Annotation Wizard**.
4. From the drop-down list, select **Annotate Lock**.
5. Click **Next**.
6. Accept the default annotation parameter of **0** and click **Next**.
7. View the proposed modified code and click **Finish**.
8. The resulting code below shows the two lock annotation lines inserted by using the **Annotation Wizard**:

```
//ANNOTATE_LOCK_ACQUIRE(0);
//ADVISOR COMMENT: This is a race condition because multiple tasks may
ANNOTATE_LOCK_ACQUIRE( 0 );
nrOfSolutions++;  //Placed final queen, found a solution!
ANNOTATE_LOCK_RELEASE( 0 );
//ANNOTATE_LOCK_RELEASE(0);
```

9. Because you modified the source code, rebuild the **2_nqueens_annotated** project to create a new target for the Correctness tool. For example, in the **Build** menu, select **Build 2_nqueens_annotated**. Examine the **Output** window to verify that no build errors were detected.

You have fixed both data sharing problems and rebuilt the target. Run the Correctness tool again and verify that the problems have been fixed. If they have been fixed, run the Suitability tool again to check the performance implications of the modified sources.

Recap

With the help of the Advisor Correctness tool, you examined your source code to find the causes of the reported data sharing problems. You fixed the source code, so you can now rebuild the target and run the Correctness tool to check that the errors had been fixed.

Key Terms and Concepts

- Term: data race
- Term:synchronization

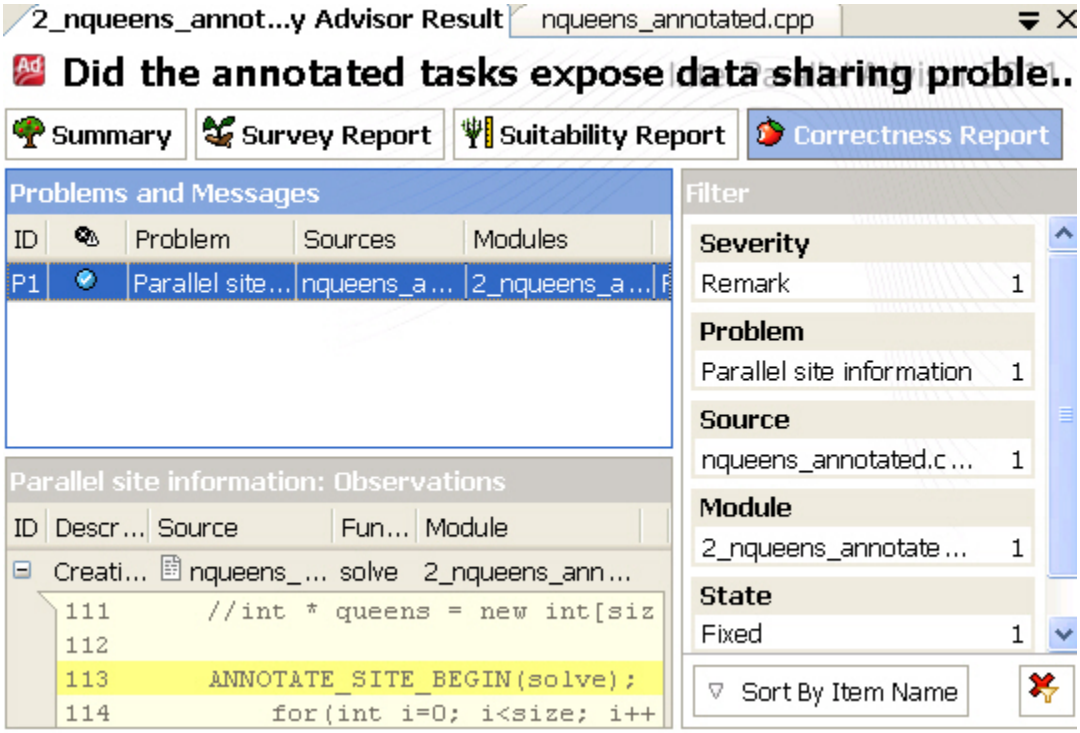
Check Program Correctness

Check Correctness of the Serial Program

To run the Advisor Correctness tool, do one of the following:

- In the **Advisor Workflow**, click the **Start** button below **4. Check Correctness**.
- Click the icon in the Visual Studio **Tools** > **Intel Parallel Advisor 2011** menu or the Advisor toolbar.

The Correctness tool runs with your program to collect data and analyze its run-time characteristics. A command window briefly appears. After the Correctness tool finishes collecting and finalizing the data, the following appears in the **My Advisor Result** tab's **Correctness Report** window:



In this case, there are no problems reported by the Correctness tool. The **Parallel site information** is a remark that indicates that this code region was actually executed by your program, and that the Correctness tool tested the parallel site's code. The reported **Parallel site information** also shows where the parallel site begins, which is marked by the annotation `ANNOTATE_SITE_BEGIN(solve);`.

If your **Correctness Report** window output shows an error under **Problems and Messages**, carefully check the source code changes described in the previous section (Fix Sharing Problems), rebuild, and run the Correctness tool again.

When modifying your application, repeat the steps of fixing the reported problems, rebuilding, and re-running the Correctness tool until no errors are reported. For example, you may encounter new problems caused by adding or modifying site or task annotations.

NOTE. If you have significantly modified your program's source code or modified any annotations, consider running the Suitability tool again to check how these changes impact your program's predicted performance and view the most recent high-level data in the **Summary** window.

Recap

You ran the Correctness tool to check that the data sharing errors had been fixed. You can now replace annotations with code that enables parallelism.

Key Terms and Concepts

- Concept: Advisor Annotations, Advisor and Visual Studio Tools

Add Parallelism

To enable parallelism to parts of your program, replace annotations with parallel framework code and build a parallel version of your program.

With your own program, before you add parallel framework code, you should complete developer/architect design and code reviews about the proposed parallel changes.

Add the Parallel Framework Code

In this step, you use the Visual Studio code editor to replace Advisor annotations with parallel framework code that enables parallelism. For your convenience, you only need to set the third sample project as your start-up project and build it:

- In the **3_nqueens_cilk** project, Intel Cilk Plus code has been added. Intel Cilk Plus is one of the high-level parallel frameworks supported by Advisor. The project is already set to use the Intel C++ Compiler.
- In the **3_nqueens_tbb** project, Intel Threading Building Blocks (Intel TBB) code has been added. Intel TBB is one of the high-level parallel frameworks supported by Advisor.

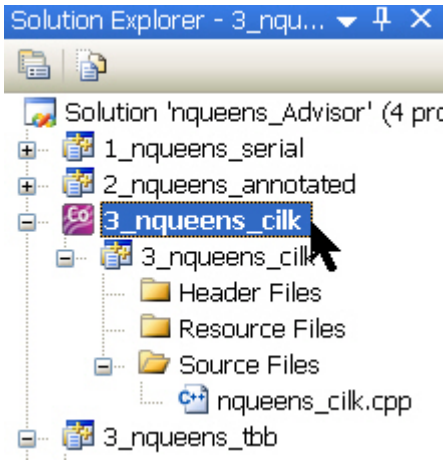
NOTE. To build and use these sample projects, you must have installed the following components provided with Intel® Parallel Composer 2011:

- The Intel® C++ Compiler, which provides support for Intel Cilk Plus.
- Intel TBB 3.0

Build Your Partially Parallel Program

Open the `nqueens_Advisor` sample, select a **Release** configuration, and build the third project as a partially parallel program:

1. In **Solution Explorer**, right-click the project **3_nqueens_cilk** or **3_nqueens_tbb**. For example:



2. From the pop-up menu, select **Set as Startup Project**.
3. From the **Build** menu, select **Configuration Manager**.
4. Select a **Release** configuration build for this project.
5. In the **Build** menu, select **Build 3_nqueens_cilk** or **Build 3_nqueens_tbb**.
6. View the **Output** window for any build errors.
7. You can optionally run this program by choosing **Debug** > **Start Without Debugging** to view its command-line output. For example:

```
Usage: nqueens_cilk boardSize [default is 13].
Starting nqueens (3_nqueens_cilk) solver for size 13...
Number of solutions: 73712
Correct result!

Calculations took 1000ms.
```

   In this case on a dual-core laptop, the serial program executed in 1743 ms (see Discover Parallel Opportunities) and the parallel version in 1000 ms (see above). The parallel execution time is significantly faster than the serial execution time. The actual execution times that will occur on your system will be different, because your system has a different processor, amount of memory, disk, and software.
8. If you used the Intel Cilk Plus project, open the source file `nqueens_cilk.cpp`. View the Intel Cilk Plus `#include` files and the Intel Cilk Plus reducer declaration, and related Intel Cilk Plus code, such as the `cilk_for` line in the `solve()` function. Locate the data race at the line where variable `nrOfSolutions` is incremented . You will notice that the lock annotations have been replaced by a declaration of the Intel Cilk Plus reducer for the `nrOfSolutions` variable. For more information about Intel Cilk Plus, locate the its documentation in Parallel Composer help.
9. If you used the Intel TBB project, open the source file `nqueens_tbb.cpp`. View the Intel TBB `#include` files and related Intel TBB code, such as the `parallel_for` line in the `solve()` function. Locate the data race at the line where variable `nrOfSolutions` is incremented. You will notice that the lock annotations have been replaced by a scoped lock. For more information about Intel TBB, locate the Intel TBB documentation in the Parallel Composer documentation directory.

NOTE. While you add parallelism, view **Summary** window to help you locate the remaining Advisor annotations that need to be replaced with parallel framework code. For help completing this step for your own program, you can click the **Explain** button and click the links to display topics in Advisor help.

Recap

You used the third project to build a parallel target executable. This project replaced the Advisor annotations with parallel framework code.

Key Terms and Concepts

- Term: parallel framework

Next Steps for the Parallel Program

Continue with Intel® Parallel Studio 2011

As it examines a serial version of a program, Advisor helps you:

- Identify areas where you should consider adding parallelism.
- Use tools that examine your running program to predict the parallel behavior of your program, such as the predicted performance of the parallel code and possible data sharing problems caused by parallel threads.

When you are satisfied with the potential parallelism identified by the Advisor annotations, you convert the annotations to parallel framework code.

Before deploying your parallel program, test it for correctness and verify its performance by using the Intel® Parallel Inspector 2011 and Intel® Parallel Amplifier 2011 tools. In addition to Advisor, the Intel Parallel Studio 2011 also includes:

- The **Intel® Parallel Composer 2011** contains Intel Threading Building Blocks used in a previous step. Intel TBB provides a library supporting parallelism in C++. Both Intel TBB and Intel Cilk Plus are high-level parallel frameworks provided with Intel Parallel Composer 2011. In addition, the Intel Parallel Composer 2011 includes the Intel® C/C++ Compiler, a parallel debugger extension, and the efficient Intel® Integrated Performance Primitives Library.
- The **Intel® Parallel Inspector 2011** uses the parallel version of your program to find correctness issues, such as shared memory problems; it can also find certain problems related to memory use.
- The **Intel® Parallel Amplifier 2011** uses the parallel version of your program to discover hot spots and provide other tuning information. It can help you measure performance speed-up, identify additional hot spots, and help you adjust locks for improved parallel performance.

Recap

You used the third project to build a parallel target executable. You can now use this parallel executable with other tools in Parallel Studio, such as checking it with the **Intel Parallel Inspector 2011** and tuning it with the **Intel Parallel Amplifier 2011**. In addition to using Intel TBB, you could also experiment with other parts of the **Intel Parallel Composer 2011**. To do this, read the related Intel Parallel Studio 2011 Getting Started Tutorials.

Key Terms and Concepts

- Term: parallel framework

Summary

Congratulations, you have completed the Advisor tutorial. Here are a few things you have learned:

- You prepared a target executable for analysis by the Advisor Survey, Suitability, and Correctness tools.
- You viewed the output of Advisor tools in the **My Advisor Result** and located relevant sources.
- You understand why you need to add Advisor site and task annotations before you use the Suitability and Correctness tools.
- You know how to add Advisor annotations to your sources.
- You learned about the **Advisor Workflow** and the basic steps needed to find places to add parallelism and use Advisor to experiment with your proposed, annotated parallel sites and tasks.
- You viewed a summary of your annotations, along with high-level data from the Suitability and Correctness tools in the **Summary** window.

More Resources

2

Getting Help

Browsing Help in the Microsoft Document Explorer

You can browse and search for topics in different ways:

- Use **Help** > **Contents** to open the Contents window and browse the Table of Contents.
- To view help for an installed parallel studio tool, select **Help > Intel Parallel Studio 2011 > Parallel Studio Help** and select the tool icon to open the title page of that tool.
- Use **Help** > **Index** to open the Index window and access an index to topics. Either type in the keyword you are looking for, or scroll through the list of keywords.
- Use **Help** > **Search** to open the Search page and search the full text of topics in the help.

Locating Intel Topics

To filter the documentation so that only the Intel documentation appears, select **Help** > **Contents** from the Visual Studio* user interface. In the **Filtered by:** drop-down list, select **Intel**.

To determine where the currently displayed topic appears in the table of contents (TOC), click the  **Sync with Table of Contents** button on the Visual Studio toolbar to highlight the topic in the Contents pane.

Where applicable, the Parallel Studio help topics provide a  **Where am I in the workflow?** button. Click the button to view the workflow with a highlight on the stage that this topic discusses.

Activating Intel Search Filters

With Microsoft Visual Studio 2005 and 2008, you can include Intel documentation in all search results by checking the **Intel** search filter box for the **Language**, **Technology**, and **Content Type** categories. You must check the **Intel** search box for all three categories to include Intel documentation in your searches. Unchecking all three **Intel** search boxes excludes Intel documentation from search results. The Intel search filters work in combination with other search options for each category.

Using Context-Sensitive Help

Context-sensitive help enables easy access to help topics on active GUI elements. The following context-sensitive help features are available on a product-specific basis:

- **F1 Help**: Press F1 to get help for an active dialog box, property page, pane, window, or for the currently selected function in the code editor.
- **What Should I Do Next? Help**: With Advisor, right click on an item in a window to display a pop-up (context) menu. Select **What Should I Do Next?** to display help for that part of the current window.

Product Website and Support

The following links provide information and support on Intel software products, including Intel® Parallel Studio:

- http://www.intel.com/software/products/

  At this site, you will find comprehensive product information, including:

  - Links to each product, where you will find technical information such as white papers and articles
  - Links to user forums
  - Links to news and events

- http://software.intel.com/en-us/articles/intel-parallel-studio/

  Intel® Software Network, Parallel Studio Support page, with links to support forums, startup help, knowledge base, and getting started video.

- http://software.intel.com/en-us/articles/tools/

  Intel® Software Development Products Knowledge Base.

- http://www.intel.com/software/products/support/

  Technical support information, to register your product, or to contact Intel.

For additional support information, see the Technical Support section of your Release Notes.

System Requirements

For detailed information on system requirements, see the Release Notes.