Tutorial: Find Where to Add Parallelism with Intel® Advisor

Intel® Advisor 2013 for Linux* OS

Fortran Sample Application Code

Document number: 327861-004US

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

Legal Information

Contents

Legal Information

Overview


Discover where to add parallelism by using Intel® Advisor 2013 and the `nqueens` Fortran sample application program. The major steps are to profile your application and add Intel Advisor annotations, predict your program's parallel behavior, and add parallel code to the selected parallel regions.

About This Tutorial	This tutorial demonstrates the first part of a workflow you can ultimately apply to your own applications. You will: • Build a serial application to produce a target executable. This target will be analyzed by Intel Advisor tools. • Profile the application to locate possible places where you might add parallelism. • Edit application code to add annotations that mark possible parallel (multithreaded) code regions. • Measure the predicted parallel performance of the annotated target by using the Suitability tool analysis. • Detect predicted parallel sharing problems of the annotated target by using the Correctness tool analysis. • Examine the summary data from the Suitability and Correctness tools to help you decide whether to modify or accept the proposed parallel sites and tasks. • Fix the predicted data sharing problems detected by the Correctness tool. Run both tools again to check the impact of fixing sharing problems. • View parallel code from a selected high-level parallel framework.
Estimated Duration	25-30 minutes.
Learning Objectives	After you complete this tutorial, you should be able to: • Understand the major steps in the Intel Advisor workflow. • Become familiar with the build requirements for the three Intel Advisor tools. • Know how to run the Survey tool to profile your application and view its report to identify the time-consuming loops and functions. • Select a possible parallel code region (site) and its tasks. Insert Intel Advisor annotations to mark these possible parallel code regions. • Run the Suitability tool to measure your annotated program's predicted parallel performance for each parallel site and its task(s). • Run the Correctness tool to predict your annotated program's likely parallel data sharing problems, such as data races. • View the Summary window to examine your program's predicted parallel performance and data sharing problems. • Become familiar with the process of modifying your code to fix predicted sharing problems by adding synchronization. • Locate resources to help you replace Intel Advisor annotations with parallel framework code.
More Resources	The concepts and procedures in this tutorial apply regardless of programming language or parallel framework selected; however, a similar tutorial using a sample application in the C/C++ language is also installed and may be available at http://software.intel.com/en-us/articles/intel-software-product-tutorials/. This site also offers tutorials for all the Intel® Parallel Studio XE products and a printable version (PDF) of tutorials. In addition, you can find more resources on the Intel® Developer Zone at http://software.intel.com/en-us/articles/intel-parallel-studio-xe/.

Next Step

Profile and Add Annotations

1 Navigation Quick Start

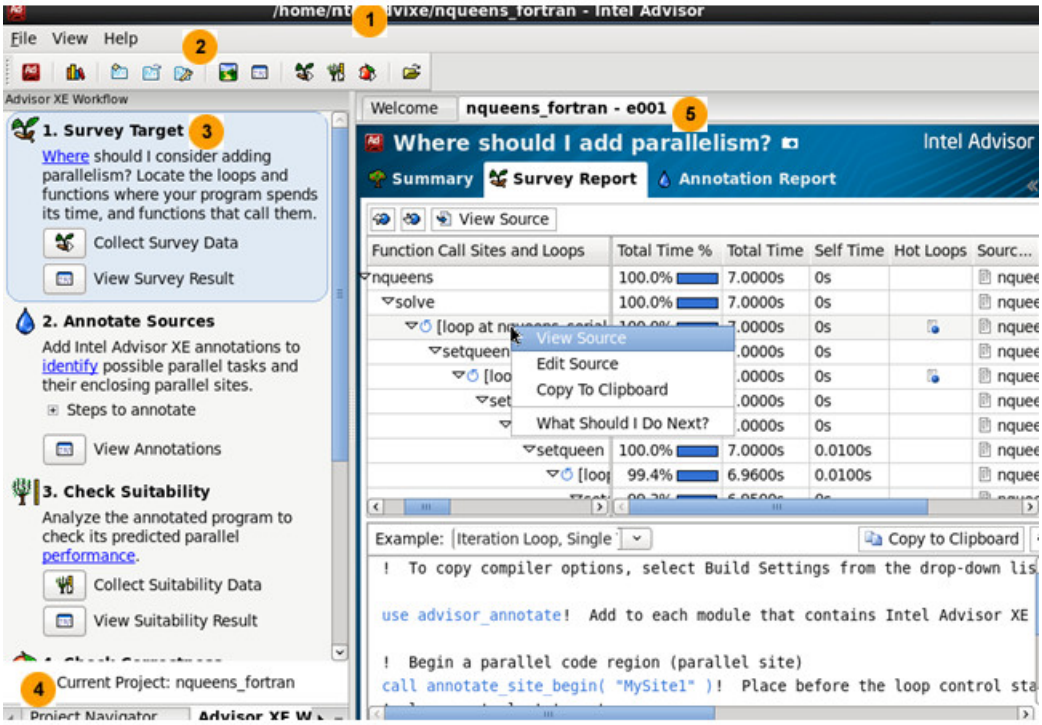
Intel Advisor provides tools that help you decide where to add parallelism to your application. Use Intel Advisor for applications:

- Created with the C/C++ or Fortran programming languages.
- Containing serial code that could possibly be replaced with parallel (multithreaded) code.

Intel Advisor Access

1. Open a terminal session.
2. Use an `ls` command to locate the Intel Advisor installation directory. The default installation directory is `/opt/intel/advisor_xe_2013/`.
3. Type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` to set up your `bash` shell environment (specify `.../advixe-vars.csh` if using a different shell).
4. Type `advixe-gui` to launch the Intel Advisor GUI.

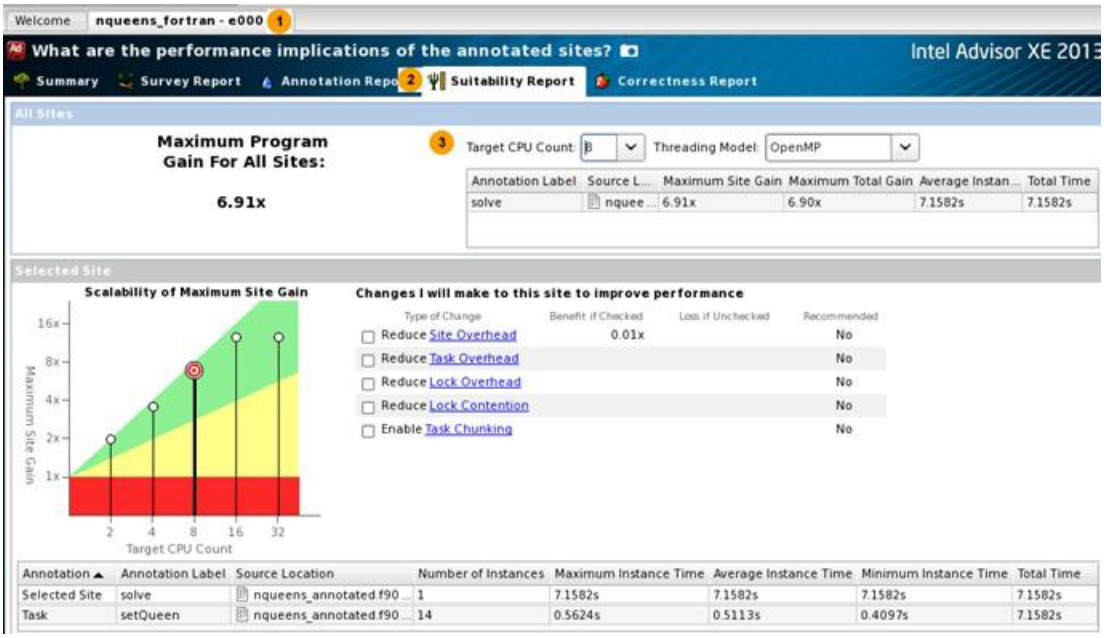
Intel Advisor GUI



The menu, toolbar, **Advisor XE Workflow**, and **Project Navigator** offer different ways to perform many of the same functions.

1. After you open a project, its name appears in the title bar. Use the menus to create and open projects; set various options; and open the Intel Advisor Getting Started page and the Help.
2. Use the toolbar to open the Intel Advisor *Getting Started* page, create or open projects, specify project properties, create and open results, open or close the **Project Navigator**, and view a summary report.
3. Use the **Advisor XE Workflow** to guide you through the recommended workflow. It helps you launch Intel Advisor tools and view tool reports.
4. Click the **Project Navigator** button to display the **Project Navigator** tab. Use the **Project Navigator**:
   - Tree diagram to see a hierarchical view of your projects and results based on the directory where the opened project resides.
   - Context menus to perform functions available from the menu and toolbar. You can also delete or rename a selected project or result, close all opened results, and copy various directory paths to the system clipboard.
5. Use the result tab to view the data collected by Intel Advisor tools. There is one active result for each project. You can also create read-only result snapshots. To display a context menu, right-click the displayed data or code in the various result windows.

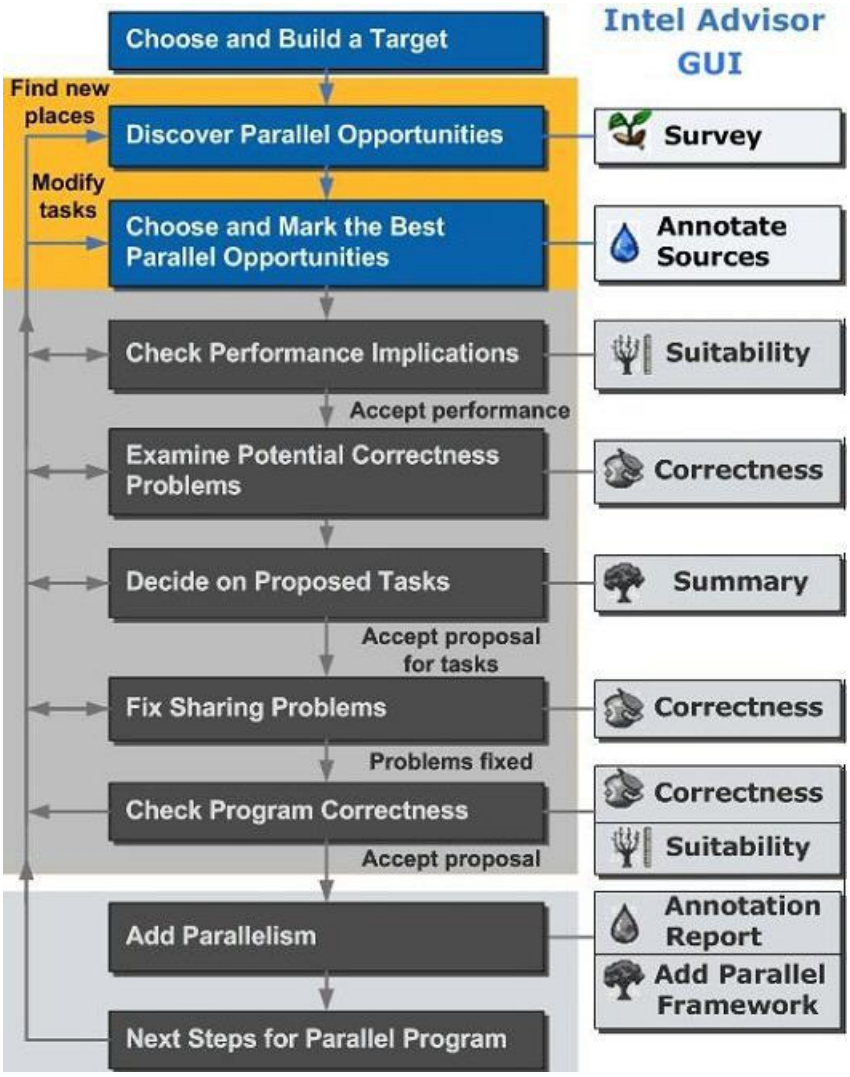
Intel Advisor Result Tab



1. The result name appears in the tab. It consists of the specified project name (such as **nqueens_fortran)**, followed by an identifier in the format e*nnn* (shown as **e000**). For read-only result snapshots, the words **(read-only)** appear after the result name that you previously specified (see Decide on the Proposed Tasks).
2. Click buttons on the navigation toolbar to change window views.
3. The displayed result shows the selected window view, presenting the finalized data collected by the Intel Advisor tool analysis.

2 Profile and Add Annotations

This is the initial step to use the parallel design capabilities provided by the Intel® Advisor. The following diagram shows the first few steps in the Intel Advisor workflow, where you use the Survey tool to discover possible places to add parallelism and mark those serial code regions with Intel Advisor annotations. Two additional parts of this tutorial describe the rest of the workflow steps.



Step 1: Unpack Sample and Prepare Target Program for Survey tool analysis	In the Intel Advisor GUI: Build Application and Create New Project

Step 2: Discover Parallel Opportunities -Profile Using the Survey Tool	• Display the Advisor XE Workflow • Run the Survey Tool
Step 3: Choose and Mark the Best Parallel Opportunities	• Display sources in the Survey Source window • Find where to add Intel Advisor parallel site and task annotations • Add parallel site and task annotations

By inserting Intel Advisor annotations, the Intel Advisor Suitability and Correctness tools can predict your application's parallel behavior.

Intel® Advisor GUI: Build Application and Create New Project

To create an application the Intel Advisor can profile:

- Get software tools and unpack the sample.
- Verify optimal compiler/linker options.
- Build the target executable in release mode.
- Run and test the target executable.
- Open the Intel Advisor GUI.
- Create a project and specify its properties.

Get Software Tools and Unpack the Sample

You need the following tools to try tutorial steps yourself using the `nqueens` Fortran sample application:

- Intel Advisor, including sample applications
- `.tgz` file extraction utility
- Supported compiler (see *Release Notes* for more information). Intel Advisor samples require the Intel® Fortran (ifort) compiler included in Intel® Composer XE.
- Source code editor

Acquire Intel Advisor

If you do not already have access to the Intel Advisor, you can download an evaluation copy from http://software.intel.com/en-us/articles/intel-software-evaluation-center/.

Install and Set Up Intel Advisor Sample Applications

1. Copy the `nqueens_Fortran.tgz` file from the `samples/<locale>/Fortran/` directory to a writable directory or share on your system. The default installation path is `/opt/intel/advisor_xe_2013/`).
2. Extract the sample from the `.tgz` file.
3. Ensure you have set the VISUAL or EDITOR environment variable to your editor.

Verify Optimal Compiler/Linker Options

You can use the Intel Advisor Survey tool to profile your application.

Applications compiled/linked for a release build using the following options produce the most accurate, complete results.

Compiler/Linker Settings	Release Build Options
Compiler: Debug information	`-g`
Compiler: Optimization	`-O2` and `-fno-inline-functions`

Compiler/Linker Settings	Release Build Options
Linker Options	`-g` and `-ldl`

Build the Target Executable

1. In a terminal session, locate the Intel Advisor installation directory root on your system. The default installation location is `/opt/intel/advisor_xe_2013`.
2. Type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or equivalent path) to set up your `bash` shell environment. With a different shell, source the `advixe-vars.csh` script.
3. If you need to set up your compiler environment, do so now.
4. Verify that the environment variable is set. If it is not set, define it by using an `export` command: `export ADVISOR_XE_2013_DIR=/opt/intel/advisor_xe_2013`.
5. Change directory to the `nqueens/` directory.
6. For the Survey tool, type `make 1_nqueens_serial` to build a release version of the `nqueens` Fortran sample application.
7. For the Suitability tool, type `make 2_nqueens_annotated` to build a release version of the annotated `nqueens` sample application. If you get compiler errors, make sure the *ADVISOR_XE_2013_DIR* environment variable is defined.
8. For the Correctness tool, type `make 2_nqueens_annotated_debug` to build a debug version of the annotated `nqueens` sample application. If you get compiler errors, make sure the *ADVISOR_XE_2013_DIR* environment variable is defined.

NOTE
This first part of the tutorial uses the `1_nqueens_serial` executable. Later parts use different files, such as `2_nqueens_annotated` and `2_nqueens_annotated_debug`.

Verify the Target Executable Runs Outside the Intel Advisor

1. In the same terminal session, change directory (`cd`) to the `nqueens` directory that you created when you extracted the `nqueens_Fortran.tgz` file.
2. Type `./1_nqueens_serial` to execute the sample application.
3. Check for output similar to the following:

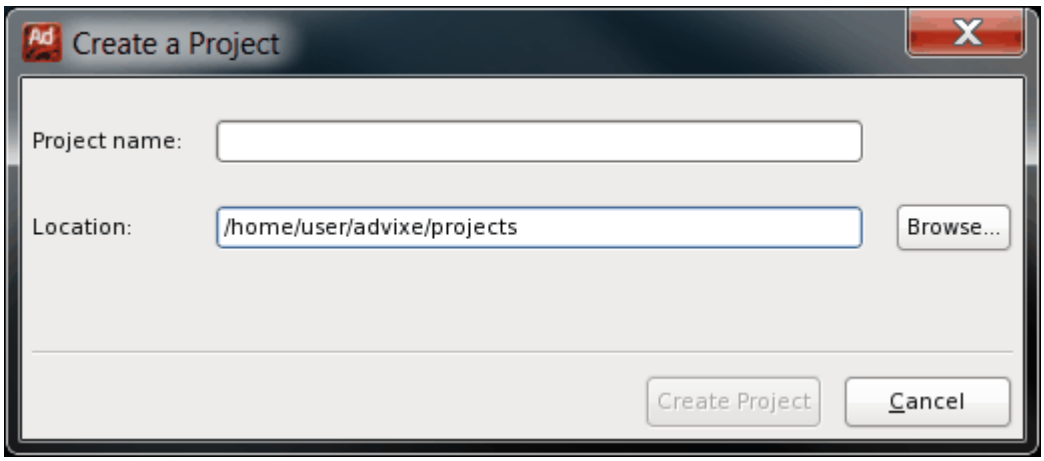
```
 Usage: 1_nqueens_serial boardSize
 Using default size of 14
Starting serial solver for size 14
Number of solutions: 365596
Calculations took 7160ms.
Correct Result!
```

Open the Intel Advisor GUI

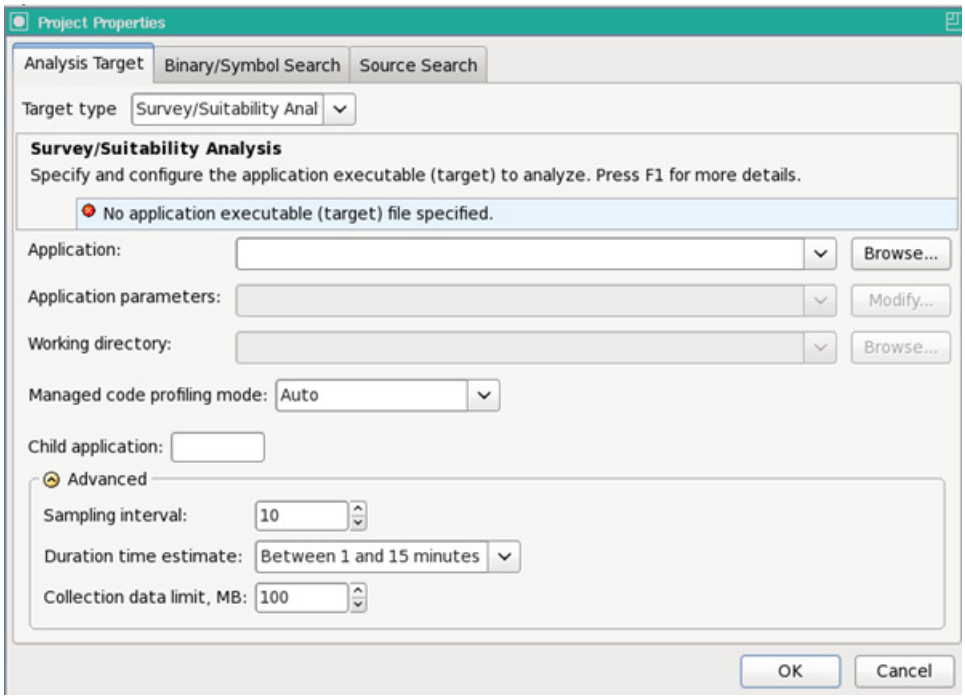
1. If you have not already done so, type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or `.csh`) to set up your command-line environment.
2. Type `advixe-gui` to open the Intel Advisor GUI.

Create a New Project and Specify its Properties

1. Choose **File** > **New** > **Project...** (or click New Project... in the Welcome page) to display a dialog box similar to the following:



2. Type `nqueens_fortran` in the **Project name** field. Then click the **Create Project** button to create a `config.advixeproj` file in the `$HOME/intel/advixe/projects/nqueens_fortran/` directory (default location) and display a dialog box similar to the following:



3. For the selected **Target type** of **Survey/Suitability Analysis**, click the **Browse...** button next to the **Application** field and select the `nqueens_fortran/1_nqueens_serial` executable file. Click **Open**. Notice that the Intel Advisor autofills the project **Working directory** field for you.
4. Click the **Binary/Symbol Search** tab. In this tab:
   - Locate the line labeled **Add new search location**.
   - Click the ... button on the right of that line.
   - Navigate to and select the `nqueens` directory.
   - Click **Open** to add the line.
   - Later when you use your own application, you may need to repeat the above steps to include other directories. You can move the current line up, down, and delete lines using the buttons on the lower-right.

- Do not select (check) the **Search recursively** box (lower-left corner).

5. Click the **Source Search** tab. In this tab:
   - Locate the line labeled **Add new search location**.
   - Click the  button on the right of that line.
   - Navigate to and select the `nqueens` directory.
   - Click **Open** to add the line.
   - With your own application, repeat the above if you need to include other local directories. You can also use wildcard characters (`*.f90`) or specific file names to be included or excluded. You can move the current line up, down, and delete lines using the buttons on the lower-right.
   - Do not check the **Search recursively** box (lower-left corner).
6. When the directories are complete, click the **OK** button. Notice that the project name appears in the title bar.

You have created a project and specified its project properties. At any time, to modify project properties for the currently opened project, click the  icon in the Intel Advisor toolbar.

Key Terms

target

Next Step

Discover Parallel Opportunities

Discover Parallel Opportunities

 To locate 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:

- Display the Advisor XE Workflow.
- Run the Survey tool.

Display the Advisor XE Workflow

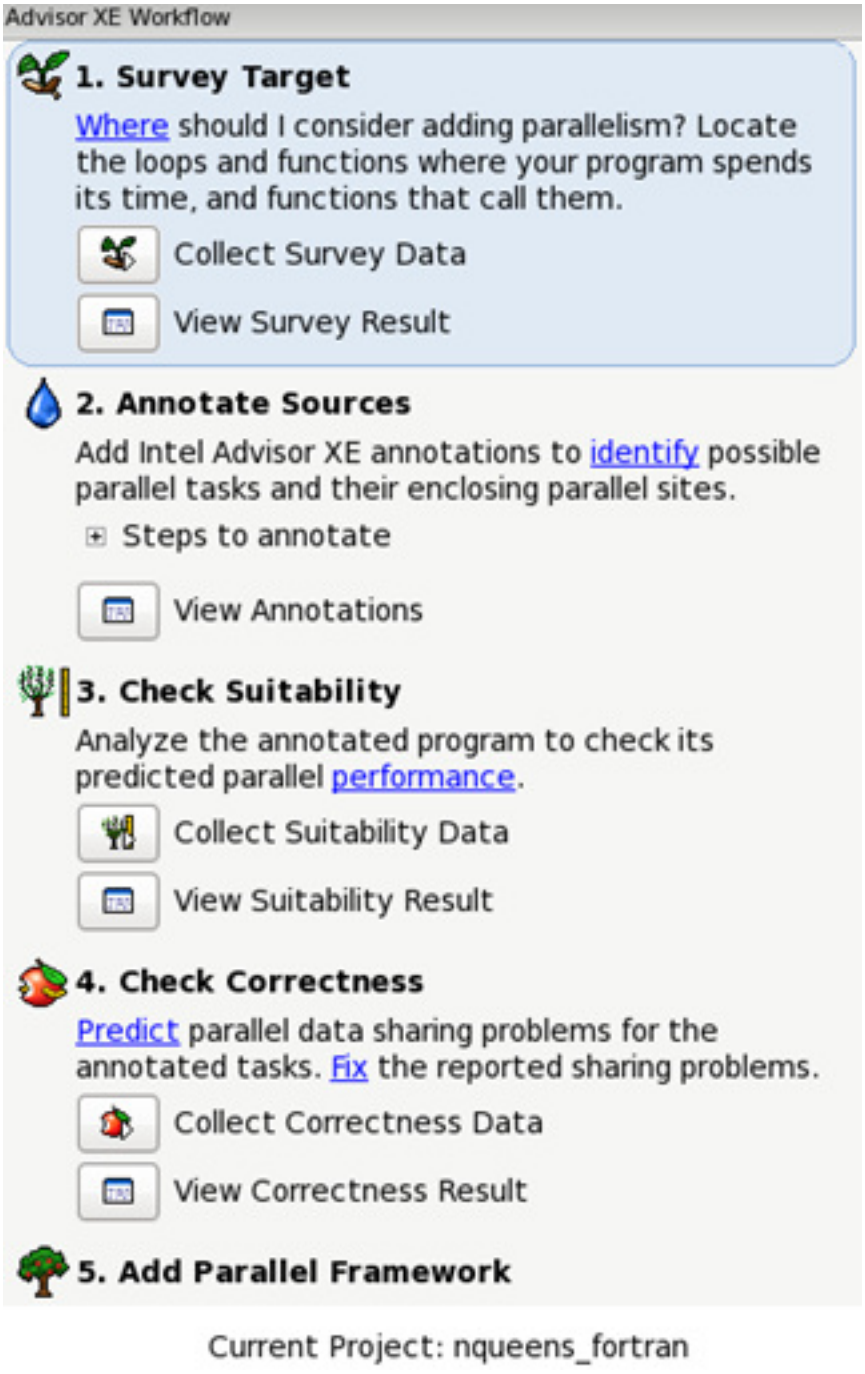
The **Advisor XE Workflow** helps guide you through the steps in the workflow.

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

- In the toolbar, click the  icon.
- In the Intel Advisor GUI, click **File** > **View** > **Open Advisor XE Workflow**

Each step in the **Advisor XE Workflow** contains instructions, which include relevant links to the Intel Advisor help. To view the steps below the **2. Annotate Sources** and **5. Add Parallel Framework** instructions, click the  icon.

In the **Advisor XE Workflow**, follow the steps from top-to-bottom.



The **Advisor XE Workflow** tab presents a high-level version of the full Intel Advisor workflow shown in the Intel Advisor help.

Run the Survey Tool

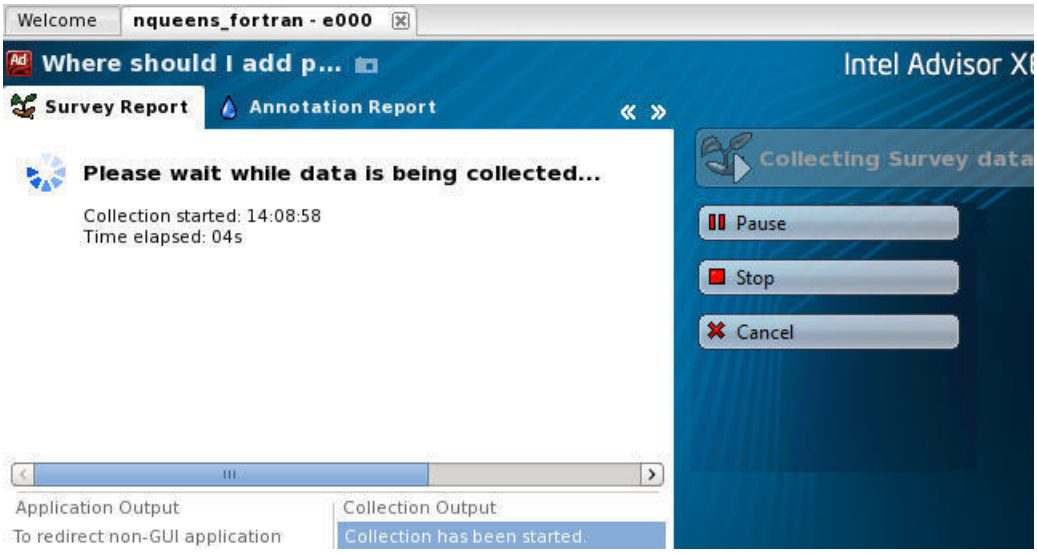
To run the Survey tool, do one of the following:

- In the **Advisor XE Workflow**, click the **Collect Survey Data** button below **1. Survey Target**.
- In the **Survey Report** window, click the **Collect Survey Data** or the **Start Paused** button on the side command toolbar. To hide or show the command toolbar, click the » or « button in the upper-right of the Survey Report.

- In the Intel Advisor GUI, choose **File** > **New** > **Start Survey Analysis**.

> **NOTE**
> This tutorial explains how to use Intel Advisor tools from the Intel Advisor graphical user interface (GUI). You can also use the Intel Advisor command-line interface (`advixe-cl` command) to run a tool analysis.

While the Survey tool runs your program, or if the selected project contains no Survey data, the **Survey Report** command toolbar appears. For example, immediately after you click the **Collect Survey Data** button, it appears on the right of the result tab:

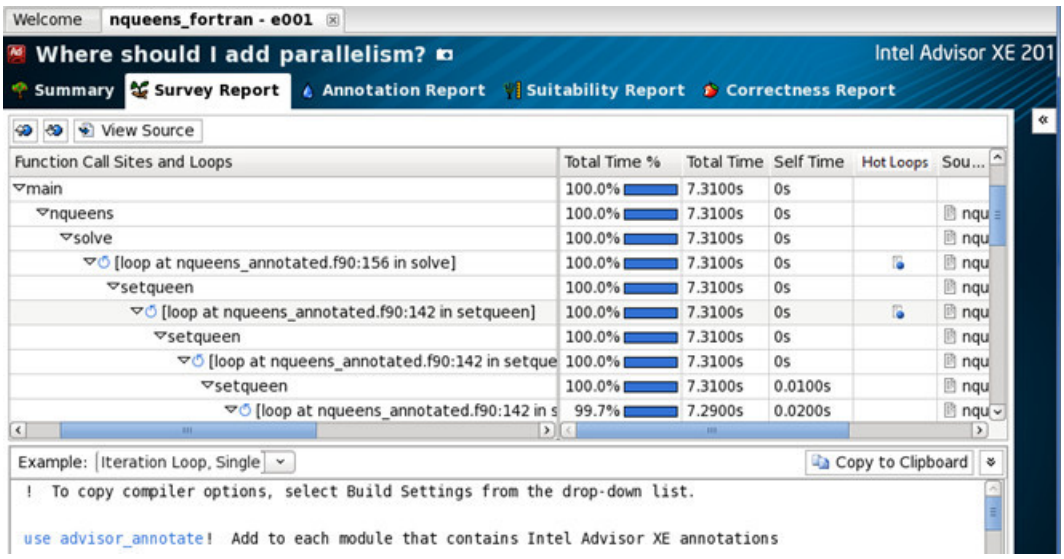


Command output appears while the Survey tool runs your program to collect data and analyze its run-time performance characteristics. After the Survey tool successfully completes its analysis, the **Survey Report** appears.

To minimize the analysis scope to specific places of interest in a large application, see the Intel Advisor help topic: Discovering Parallel Opportunities > Minimizing Data Collection, Result Size, and Execution Time (Survey).

To redirect application non-GUI output to the **Application Output** pane (shown above) instead of a command window, use the **Options** dialog box.

The collected Survey data appears in the **Survey Report** and **Summary** windows:



When the **Survey Report** window first appears, an informational help tip appears. To close it, click the icon in its upper-right corner. To hide or show the command toolbar, click the or button in the upper-right of the Survey Report.

The result tab displays data collected by the various Intel Advisor tools for the selected project. The result name **nqueens_fortran - e001** appears near the top of the result tab shown above.

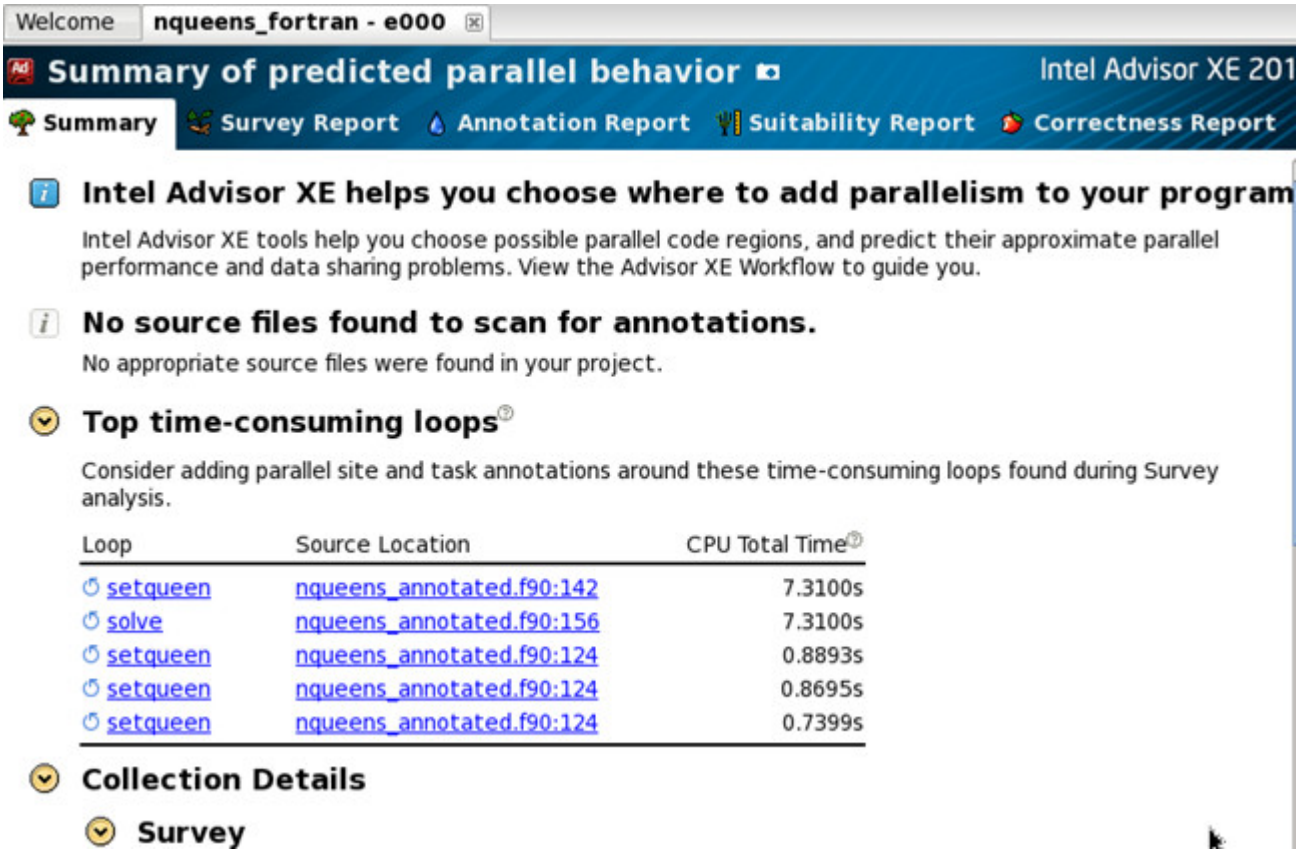
The **Survey Report** window provides a top-down view of an expanded call graph with loops. The subroutine `setQueen` calls itself recursively, so it appears multiple times in the **Function Call Sites and Loops** column, along with its loops (icon). The command toolbar shows the **Start Paused** button below **Collect Survey Data**.

Loops are often the most time-consuming (hot) parts of your program. The Survey Report identifies the most time-consuming loops with the icon in the **Hot Loops** column. Such loops are candidates for parallel tasks. The top time-consuming (hot) loop is within the `solve` subroutine, which calls the `setQueen` subroutine.

You can also look at the values in the **Total Time** and the **Self Time** columns, to see that the recursive `setQueen` subroutine 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` subroutine.

The top of the **Survey Report** window provides a small toolbar to help you quickly locate the next or previous hot loop, and view sources in the **Survey Source** window.

In the result tab, click the **Summary** (left-most) button to display the **Summary** window:



To help you quickly identify possible loops where you might add parallelism, the top time-consuming loops and their CPU time appear under **Top time-consuming loops**. These loops are also identified in the **Hot Loops** column in the **Survey Report**.

Click the link under the **Source Location** column to view the time-consuming loop code, which is located within the `solve` subroutine.

As you run (and re-run) Intel Advisor tools, more data gets added to this dashboard-like **Summary** window. It provides easy access to detailed data in the report windows, your sources, and collection details. For example, if you click the link for the `solve` subroutine, the **Survey Report** window appears.

The **Summary** window also displays a count of annotations found and source files searched. In this case, the sources do not yet contain Intel Advisor source annotations.

Key Terms

hotspot

Next Step

Choose and Mark the Best Parallel Opportunities

Choose and Mark the Best Parallel Opportunities

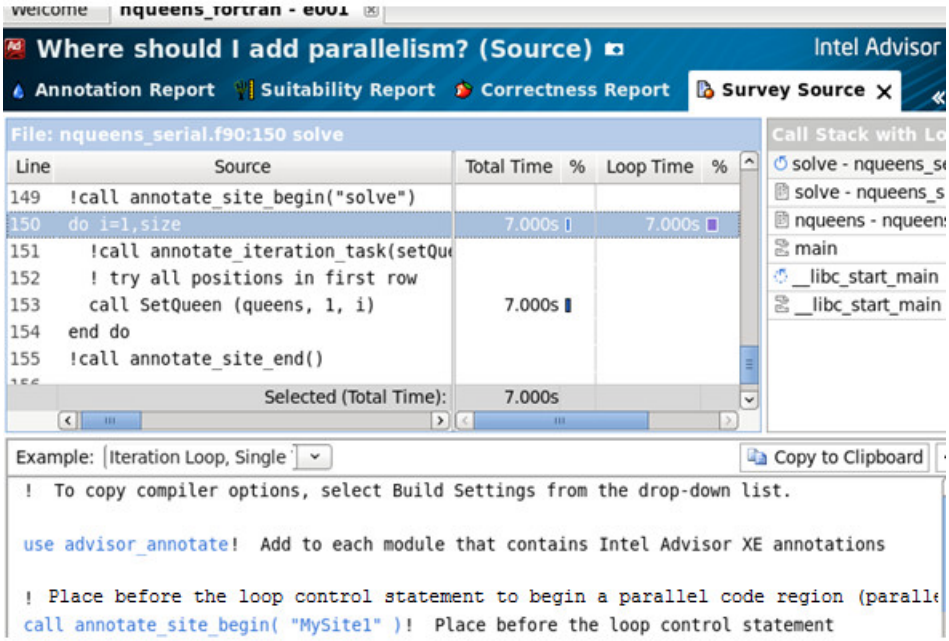
Using the **Survey** tool data, choose places in your program to add parallelism and mark those places by inserting Intel Advisor annotations. To do this:

- Display sources in the Survey Source window.

- Find where to add Intel 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 subroutine `solve` (the first **Hot Loop**) to display the **Survey Source** window:



The recursive subroutine call to `setQueen` 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 measured time executing this statement or in functions invoked from this statement.
- The **Loop Time** column shows the sum of the **Total Time** for all the code in this loop. It is displayed for one statement in the loop, such as the loop header.

Find Where to Add Intel 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 subroutine `setQueen`. In this case, `main` accepts command-line arguments, initializes an array, and calls the subroutine `solve`. The `solve` subroutine calls the `setQueen` subroutine, which calls itself recursively.

Either use the **Survey Source** window (double-click or right-click a line and select **Edit Source**) or the editor to open the source file `nqueens_serial.f90`. Get familiar with the code execution paths and data use. For example, you need to understand code paths to ensure that any annotations you add will be executed.

The `nqueens` sample includes lines that begin with `!ADVISOR COMMENT`. In `nqueens_serial.f90`, notice that:

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

- The `use` statement references the Intel Advisor annotations definitions (module file). This is needed because annotations are present in this source file. With your own Fortran application modules that contain annotations, you need to insert this line:

  ```
  use advisor_annotate
  ```

- For your convenience, the annotations are inserted as comment lines in this version of this project's source file. For example, this is a site-begin annotation, placed just before the loop:

  ```
    !call annotate_site_begin("solve")
  ```

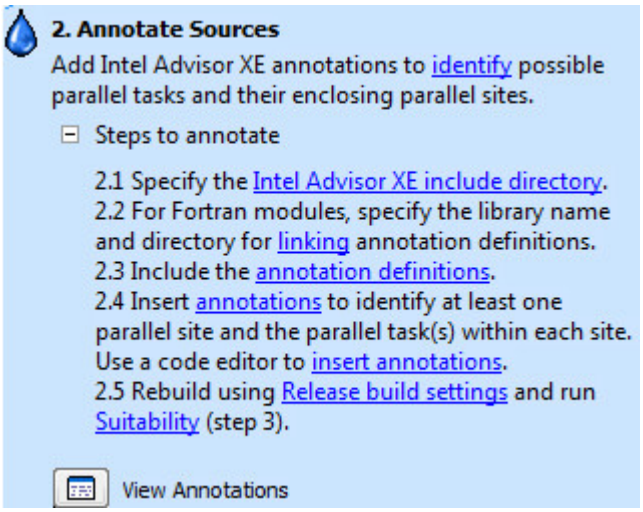
  These annotations are also present in the next source file (`nqueens_annotated.f90`).

```
subroutine solve (queens)
  implicit none
  integer, intent(inout) :: queens(:)
  integer :: i

  !ADVISOR COMMENT: When surveying this is the top function below the main function.
  !ADVISOR COMMENT: Uncomment the three annotations below to model parallelizing the
  !ADVISOR COMMENT: Don't forget to uncomment the "use advisor_annotate" at the top

  !call annotate_site_begin("solve")
  do i=1,size
    !call annotate_iteration_task("setQueen")
    ! try all positions in first row
    call SetQueen (queens, 1, i)
  end do
  !call annotate_site_end()
end subroutine solve
```

In your own program, choosing where to add task annotations may require some experimentation. If your parallel site has nested loops and the computation time used by the innermost loop is small, consider adding task annotations around the next outermost loop.

Adding Parallel Site and Task Annotations to Your Program

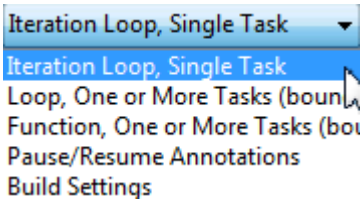
When adding annotations to your own program, remember to include the annotations definitions, such as the `advisor_annotate` module for Fortran programs. For help completing this step, access the **Advisor XE Workflow** tab and click the ⊞ button below **2. Annotate Sources** to display instructions:



At the bottom of the **Survey Report** or **Survey Source** windows, use the annotation assistant pane to copy the annotation example code that you can paste into your editor:

- On the right side, select **Iteration Loop, Single Task** from the drop-down list.
- View the displayed example annotation code.
- Click the **Copy to Clipboard** button to copy the displayed text into the paste buffer.
- Paste the code snippet into an intermediate editing window or directly into your editor. To launch the code editor with that source file opened to the corresponding location, either double-click a source line in the **Survey Source** window or right click and choose **Edit Source**. Within the Intel Advisor GUI, click **File > Options > Editor** to choose the Linux* editor displayed for each source language.
- Use the code editor to change the placeholder site and task name parameters to meaningful ones. For example, change the site name from `"MySite1"` to `"solve"` before you save the file.

For other task code structures, select the appropriate type from the list:



For example, if loop(s) within the parallel site (proposed parallel code region) contain multiple tasks or a task does not include the entire loop body, select **Loop, One or More Tasks**. For multiple function calls that each might be tasks, select **Function, One or More Tasks**.

You can also copy language-specific build settings to paste into your application's build script by selecting **Build Settings** and clicking the **Copy to Clipboard** button.

Key Terms

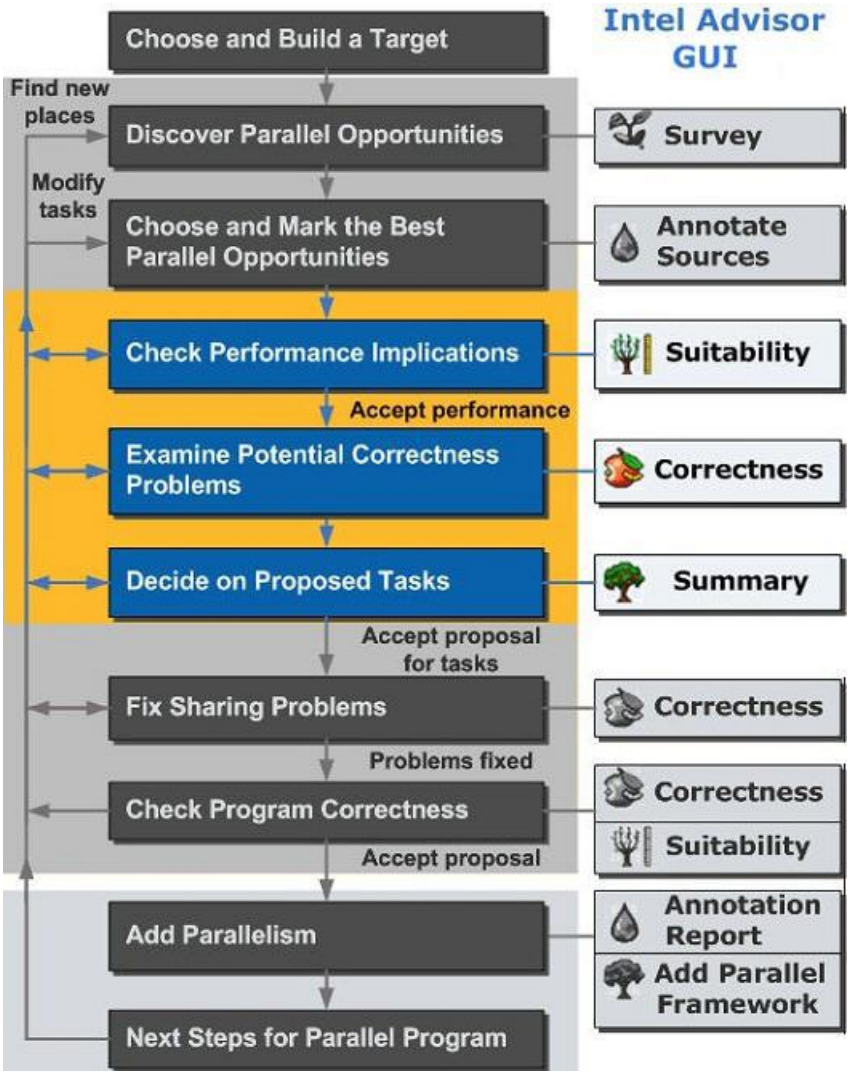
annotation, parallel site and parallel task

Next Step

Predict Parallel Behavior

3 Predict Parallel Behavior

This is the second step to use the parallel design capabilities provided by the Intel® Advisor. The following workflow shows how to predict the parallel behavior of an annotated program.



Step 1: Unpack Sample and Prepare for Suitability tool analysis	In the Intel Advisor GUI: Build Application and Open the Project
Step 2: Predict Parallel Performance Using the Suitability Tool	Run the Suitability tool to Check Performance Implications

Step 3: Prepare for Correctness tool analysis	In the Intel Advisor GUI: Build Application and Open the Project
Step 4: Predict Data Sharing Problems Using the Correctness Tool	Run the Correctness tool to Examine Potential Correctness Problems
Step 5: Decide on the Tasks	Use the Summary window to Decide on the Proposed Tasks

Intel® Advisor GUI: Build Application and Open the Project

To create an application the Intel® Advisor tools can use to predict parallel performance and possible parallel data sharing problems, you need to:

- Get software tools and unpack the sample.
- Verify optimal compiler/linker options.
- Build the target executable.
- Run and test the target executable.
- Open the Intel Advisor GUI.
- Modify project properties.

Get Software Tools and Unpack the Sample

If you completed this step in the previous part of this tutorial, skip to Verify optimal compiler/linker options.

You need the following tools to try tutorial steps yourself using the `nqueens` sample application:

- Intel Advisor, including sample applications
- `.tgz` file extraction utility
- Supported compiler (see *Release Notes* for more information). Intel Advisor samples require the Intel® Fortran (ifort) compiler included in Intel® Composer XE.
- Source code editor

Acquire Intel Advisor XE

If you do not already have access to the Intel Advisor XE, you can download an evaluation copy from http://software.intel.com/en-us/articles/intel-software-evaluation-center/.

Install and Set Up Intel Advisor Sample Applications

1. Copy the `nqueens_Fortran.tgz` file from the `samples/<locale>/Fortran/` directory to a writable directory or share on your system. The default installation path is `/opt/intel/advisor_xe_2013/`).
2. Extract the sample from the `.tgz` file.
3. Ensure you have set the VISUAL or EDITOR environment variable to your editor.

Verify Optimal Compiler/Linker Options

You can use the Intel Advisor Suitability and Correctness tools to predict parallel performance and possible parallel data sharing problems.

Applications compiled/linked using the following options produce the most accurate, complete results.

Compiler/Linker Settings	Release Build for Suitability	Debug Build for Correctness
Compiler: Debug information	`-g`	`-g`

Compiler/Linker Settings	Release Build for Suitability	Debug Build for Correctness
Compiler: Optimization	`-O2` and `-fno-inline-functions`	`-O0`
Compiler: Code Generation		`-threads -shared-intel`
Compiler: search an additional include directory	`-I${ADVISOR_XE_2013_DIR}/include`	`-I${ADVISOR_XE_2013_DIR}/include`
Linker Options	`-g` and `-ldl`	`-g` and `-ldl` The linker should search for unresolved references in a multithreaded, shared object run-time library. If needed, specify `-threads` or `-Bdynamic`.

Build the Target Executable

1. In a terminal session, locate the Intel Advisor installation directory root on your system. The default installation location is `/opt/intel/advisor_xe_2013`.
2. Type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or equivalent path) to set up your `bash` shell environment. With a different shell, source the `advixe-vars.csh` script.
3. If you need to set up your compiler environment, do so now.
4. Verify that the *ADVISOR_XE_2013_DIR* environment variable is set - for example, type: `env | grep ADVISOR_XE_2013_DIR`. If it is not set, define it by using an `export` command: `export ADVISOR_XE_2013_DIR=/opt/intel/advisor_xe_2013`.
5. Change directory to the `nqueens/` directory .
6. Type:
   - `make 2_nqueens_annotated` to build a release version of the annotated `nqueens` sample application for Suitability (or Survey) tool analysis. If you get compiler errors, make sure the *ADVISOR_XE_2013_DIR* environment variable is defined (use an `export` command).
   - `make 2_nqueens_annotated_debug` to build a debug version of the annotated `nqueens` sample application for Correctness tool analysis. If you get compiler errors, make sure the *ADVISOR_XE_2013_DIR* environment variable is defined.

Verify the Target Executable Runs Outside the Intel Advisor XE

1. In the same terminal session, change directory (`cd`) to the nqueens directory that you created when you extracted the `nqueens_Fortran.tgz` file.
2. Type `./2_nqueens_annotated` to execute the sample application.
3. Check for output similar to the following:

```
 Usage: 2_nqueens_annotated[_debug] boardSize
 Using default size of 14
Starting serial recursive solver for size 14
Number of solutions: 365596
Calculations took 7159ms.
Correct Result!
```

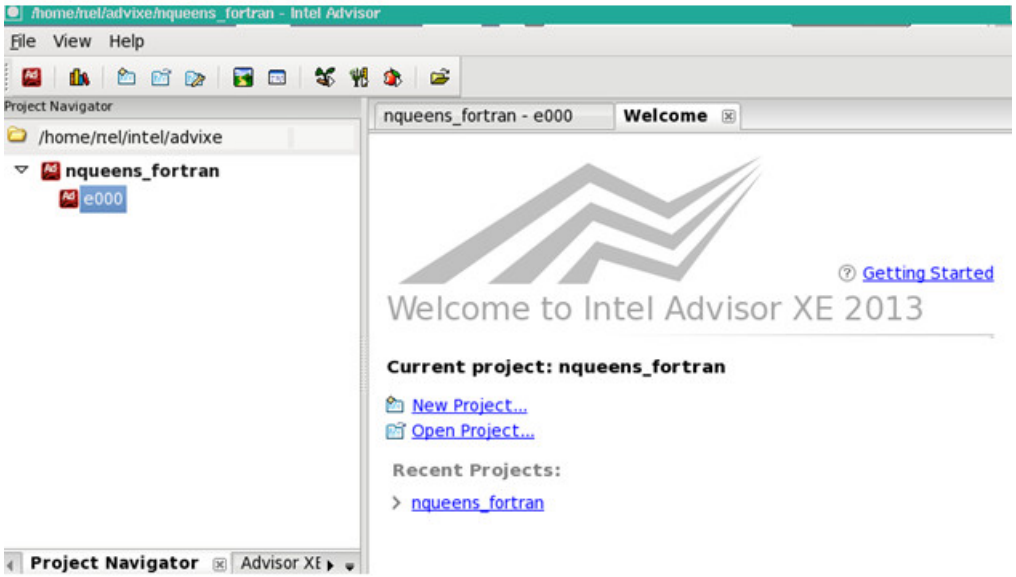
   Type `./2_nqueens_annotated_debug` to run the debug build version, which takes longer to execute than a release build.

TIP
Keep the command prompt window open.

Open the Intel Advisor GUI

1. If you have not already done so, type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or equivalent path) to set up your command-line environment.
2. Type `advixe-gui` to open the Intel Advisor GUI.

   The previously opened project should be re-opened and its name should appear in the title bar and in the Project Navigator:

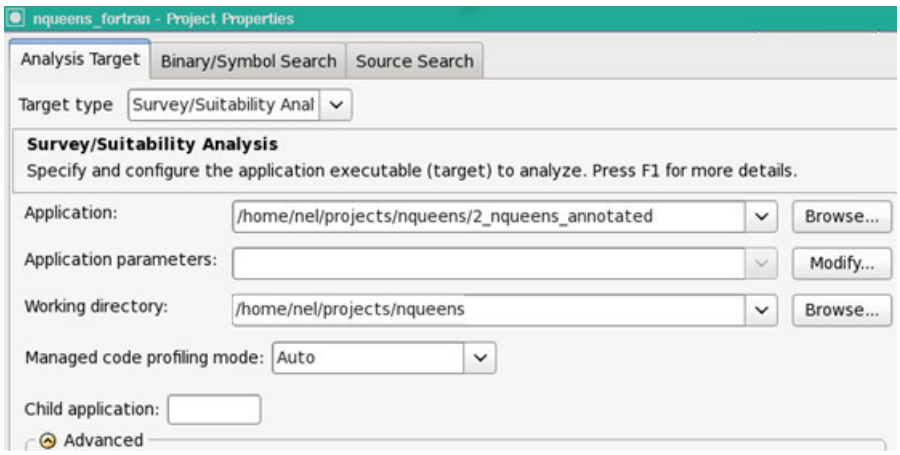


If the project was closed, either click the `nqueens_fortran` link below Recent Projects: in the **Welcome** page, or choose **File** > **Recent Projects...** and select the recently created project name.

NOTE
If you did not complete the first part of this tutorial, you need to *create* a project, not *open* one. Select **File** > **New** > **Project...** and specify the project name as nqueens_fortran. Use **Project Properties...** (next step) to specify the target executable and other properties.

Modify Project Properties

1. Verify that the project is open by viewing its name in the title bar.
2. Click **File** > **Project Properties...** or click the toolbar  icon to display a dialog box similar to the following (shown with Suitability release build values):



3. For the selected **Target type** of Survey/Suitability, click the **Browse...** button next to the **Application** field and select the `.../nqueens/2_nqueens_annotated` executable file.
4. For the selected **Target type** of Correctness:
   - Click the **Browse...** button next to the **Application** field and select the `nqueens/2_nqueens_annotated_debug` file.
   - To the right of **Application parameters**, type a value of **8**. This reduces the board size to minimize execution time.
5. Click the **Binary/Symbol Search** tab. In this tab:
   - Click the line labeled **Add new search location**.
   - Click the ... button on the right of that line.
   - Navigate to and select the `nqueens` directory.
   - Click **Open** to add the line.
   - With your own application, repeat the above steps if you need to include other directories. You can move the current line up, down, and delete lines using the buttons on the lower-right.
6. Click the **Source Search** tab. In this tab:
   - Locate the line labeled **Add new search location**.
   - Click the ... button on the right of that line.
   - Navigate to and select the `nqueens` directory.
   - Click **Open** to add the line.
   - With your own application, repeat the above steps if you need to include other local directories. You can also use wildcard characters (`*.f90`) or specific file names to be included or excluded. You can move the current line up, down, and delete lines using the buttons on the lower-right.
7. When all directories are complete, click **OK**.

Key Terms

target

Next Step

Check Performance Implications

Check Performance Implications

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

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

Build a Target for the Suitability Tool

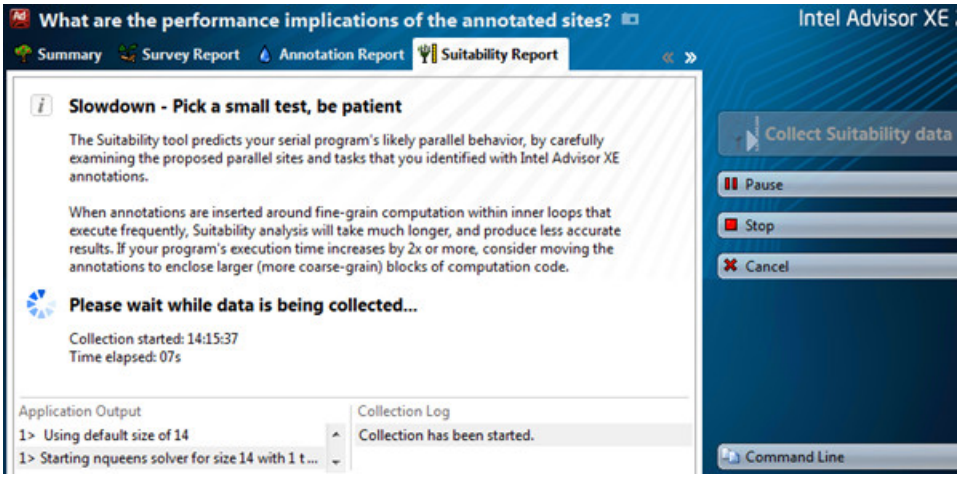
This step was completed previously, using a release build that includes debug information and moderate optimization. Before proceeding, make sure that a release build is selected. See the Intel Advisor GUI.

Run the Suitability Tool

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

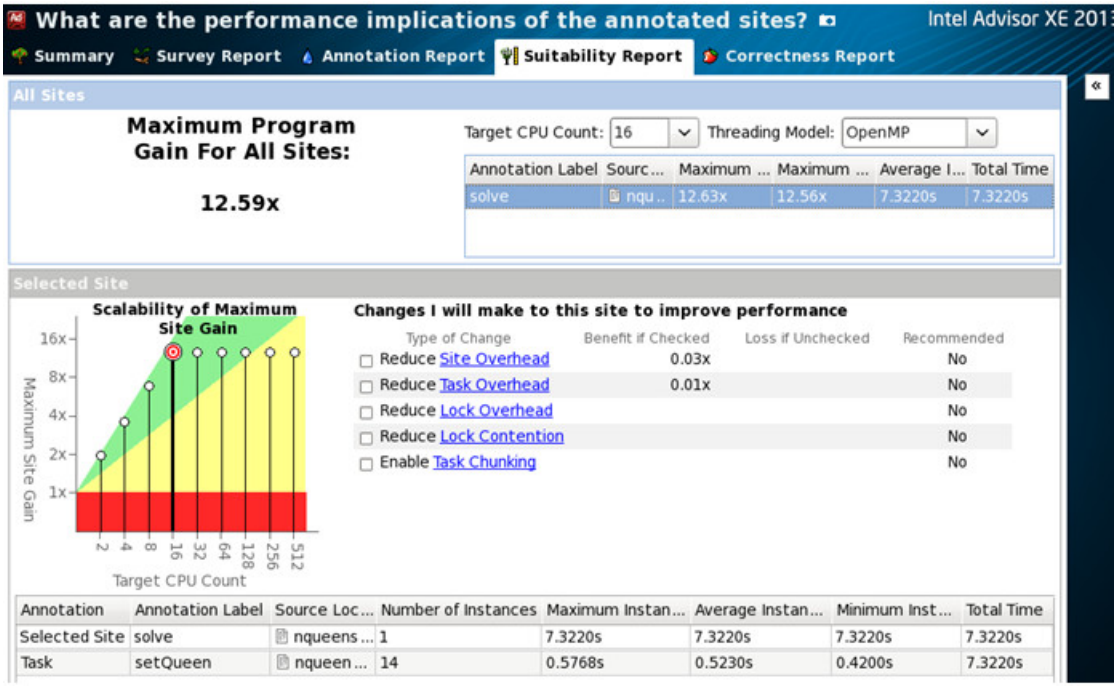
- Click the **Collect Suitability Data** or the **Start Paused** button on the side command toolbar. To hide or show the command toolbar, click the » or « button in the upper-right of the Suitability Report.
- Click the **Collect Suitability Data** button in the **Advisor XE Workflow** tab (below **3. Check Suitability**).
- In the Intel Advisor GUI, choose **File** > **New** > **Start Suitability Analysis**.

While the Suitability tool runs your program, or if the selected project contains no Suitability data, the **Suitability Report** command toolbar appears. For example, immediately after you click the **Collect Suitability Data** button:



Command output appears while the Suitability tool runs your program to predict its parallel performance characteristics. You can redirect application non-GUI output to the **Application Output** pane (shown above) instead of the command window using the **Options** dialog box.

After the Suitability tool finalizes the data, data appears in the result tab in the **Suitability Report** window:



In the upper part of the window, the **All Sites** pane shows there was one parallel site annotation pair executed in source file `nqueens_annotated.f90` (second column) with the label `solve` (first column). The column **Maximum Site Gain** indicates the approximate predicted gain for the single parallel site of **12.63x** based on the modeling parameters.

On your system, use the drop-down list to set the **Target CPU Count** to 16.

NOTE
The values displayed on your screen for **Maximum Site Gain** and **Maximum Program Gain For All Sites:** will be different than the numbers shown in this tutorial.

The value after **Maximum Program Gain For All Sites:** indicates the possible maximum performance for all sites. In this case, the value **12.59x** for the **Target CPU Number** of **16** indicates a good run-time gain.

NOTE
Depending on the vertical screen 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 **12.63x** for the original **Target CPU Number** of **16**. 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.

Select the **Threading Model** that you intend to use when you add parallelism. This selection does slightly alter the calculations based on the relative overhead expected. For this Fortran sample, select the **Threading Model** as **OpenMP**

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.

Viewing Task Characteristics

The **Selected Site** pane shows information about the tasks and locks for the selected parallel site. For example, use this information with your own program to help you decide where to add task annotations, especially if it contains nested loops. Under the **Number of Instances** column, the instances of the **Task** were **14**, which is the default data set board size for this sample with a release build.

Look at the **Average Instance Time** value for the task in the **Selected Site** pane. The **Average Instance Time** should be large enough to overcome the overhead of starting and ending a task. The **Enable Task Chunking** in the lower pane is useful when the task's **Average Instance Time** is lower than task start/stop overhead time and should be considered when values are less than 0.01 second, and always used with lower values, such as 0.00001 second. If your Fortran program has small tasks, you can use the task chunking feature with the OpenMP* parallel framework.

Similarly, if a task is an innermost loop whose computation time is small and that loop occurs within a nested loop, consider adding task annotations around the next outermost loop.

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 on the Y axis. In this case, notice that because the data set size is 14, the addition of more cores does not help speed-up beyond 14 cores, because there are only 14 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. The circles for this site appear in the green shaded area and indicate good results. If the minimum-maximum range appears in the yellow-shaded area, you should investigate how the results can be improved.

To the right of the graph is a table under **Changes I will make to this site to improve performance** that lists items related to parallel overhead, task chunking, and lock contention. Under the **Recommended** column, if you see the word **Yes**, there is a benefit for that item. In this case, you should select the check box to indicate that you agree to take the appropriate action later when you change annotations into parallel framework code. The graph will change to reflect the modified modeling parameters.

Using the Suitability Source Window and an Editor

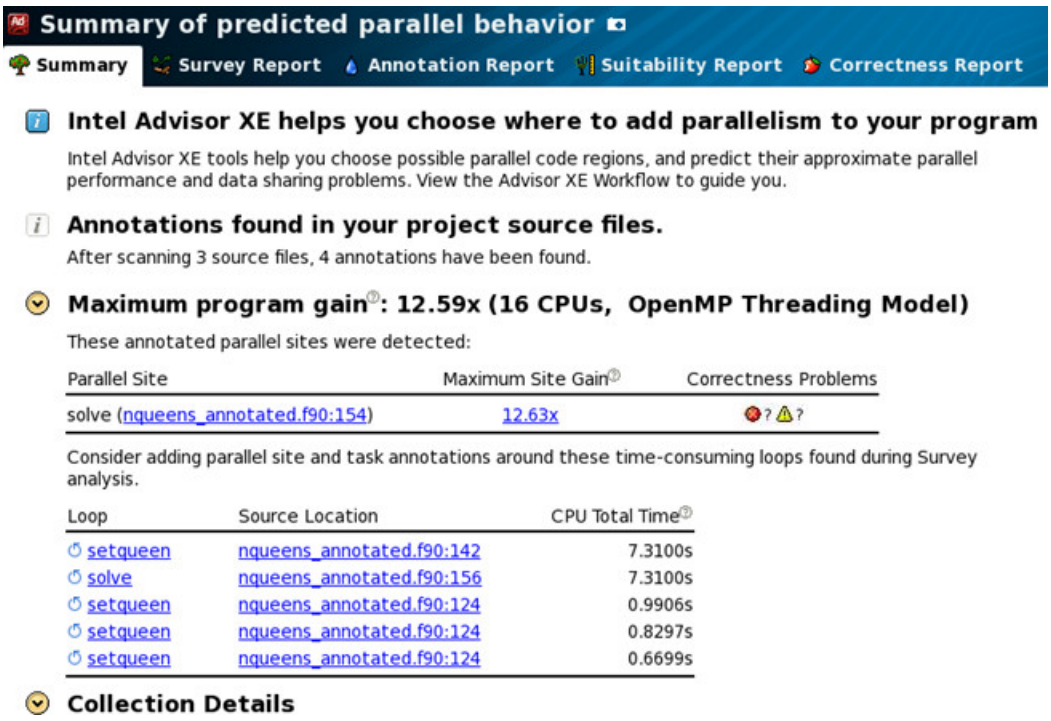
To view the sources associated with a task or lock, double-click (or right-click and select **View Source**) a line to display the **Suitability Source** window.

After you determine a correct location, double-click a source line in the **Suitability Source** window to launch the code editor with that source file opened to the corresponding location. Within the Intel Advisor GUI, click **File > Options > Editor** to choose the Linux* editor displayed for each source language.

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

Viewing the Summary Window

To view the **Summary** window, click the **Summary** button in the result tab:



The **Summary** window provides a dashboard-like summary of data collected by Intel Advisor tools. It provides easy access to detailed data in the report windows, your sources, and collection details. It also lists a count of the source files and annotations found. For example, under **Potential program gain**, in the column **Maximum Site Gain**, click the number (shown above as 12.63x) to display the **Suitability Report** window.

In the example **Summary** window shown above, because the Correctness tool has not been run for this project, there is no data (indicated by **?**) shown under the **Correctness Problems** column for the **solve** parallel site. As you run additional Intel Advisor tools, more data gets added to the **Summary** window.

Key Terms

parallel site, task

Next Step

Examine Potential Correctness Problems

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 build.
- Provide a reduced but representative data set.
- Run the Correctness tool.
- View the Correctness Report and Correctness Source windows.

Build a Target for the Correctness Tool

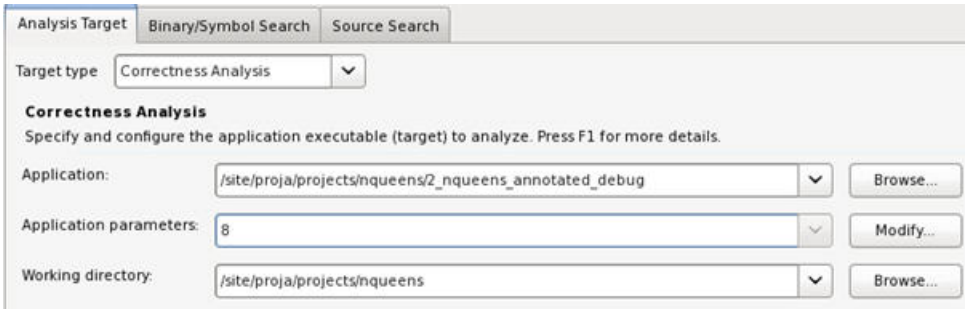
This step was completed previously, using a debug build that includes debug information and no optimization. See the Intel Advisor GUI.

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, you should reduce the data set by using the project's properties or by modifying the source code. 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 is 14, but you can specify a value of **8**:

When using the Intel Advisor GUI, select Project Properties and specify:

- **Target type** of **Correctness Analysis**
- **Application** of **.../nqueens/2_nqueens_annotated_debug**
- **Application parameters** of **8**



On the command line, add the number 8 after the command name that runs the program. For example, type:

```
./2_nqueens_annotated_debug 8
```

Run the Correctness Tool

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

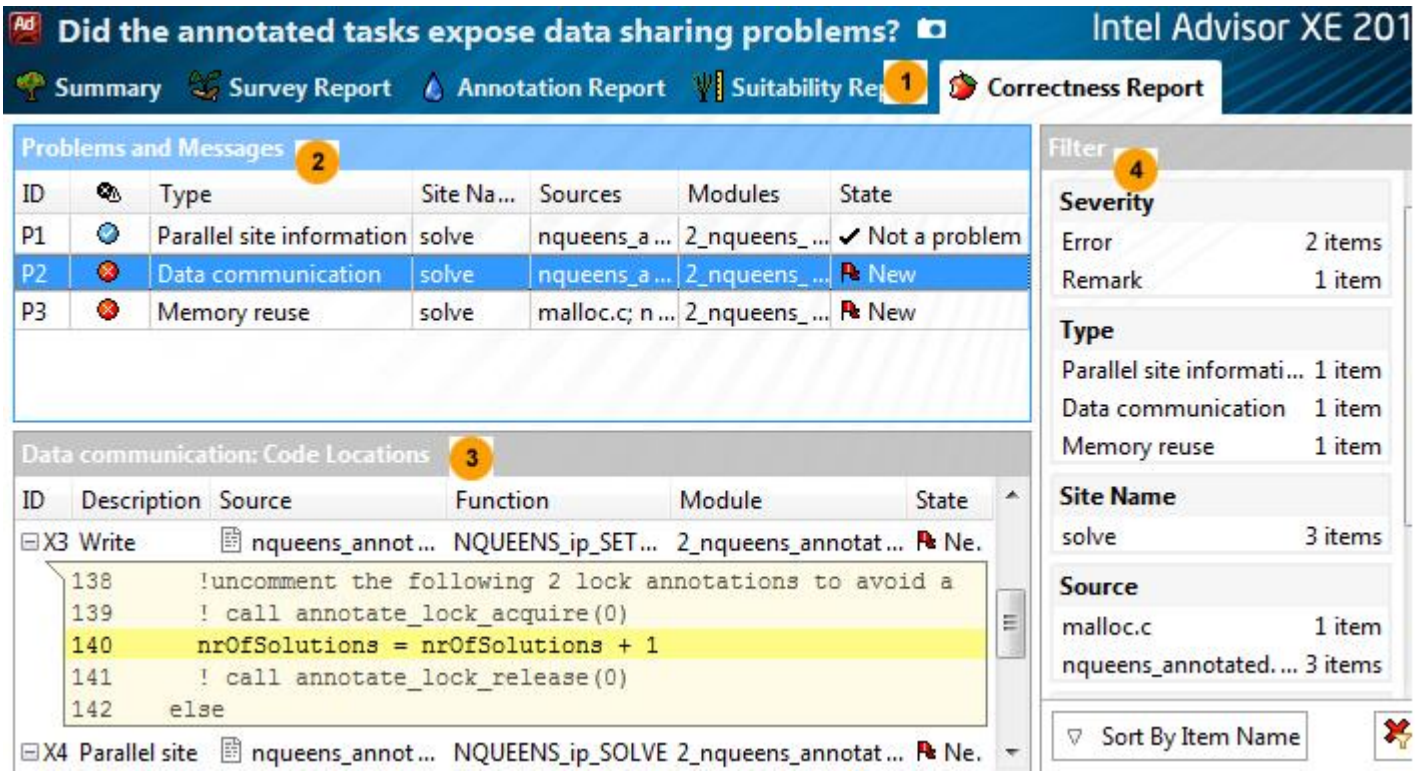
- Click the **Collect Correctness Data** button on the side command toolbar. To hide or show the command toolbar, click the » or « buttons in the upper-right of the Correctness Report.
- Click the **Collect Correctness Data** button in **Advisor XE Workflow** tab (below **4. Check Correctness**).
- In the Intel Advisor GUI, choose **File** > **New** > **Start Correctness Analysis**.

The Correctness tool runs with your program to collect data and analyze its run-time characteristics. A command window appears and displays text similar to the following when the `nqueens` program completes (the time used will be different on your system):

```
Starting serial recursive solver for size 8
Number of solutions: 92
Calculations took 528ms.
Correct Result!
```

The **Correctness Report** window's command toolbar appears while data is being collected or if there is no **Correctness Report** data for the project.

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



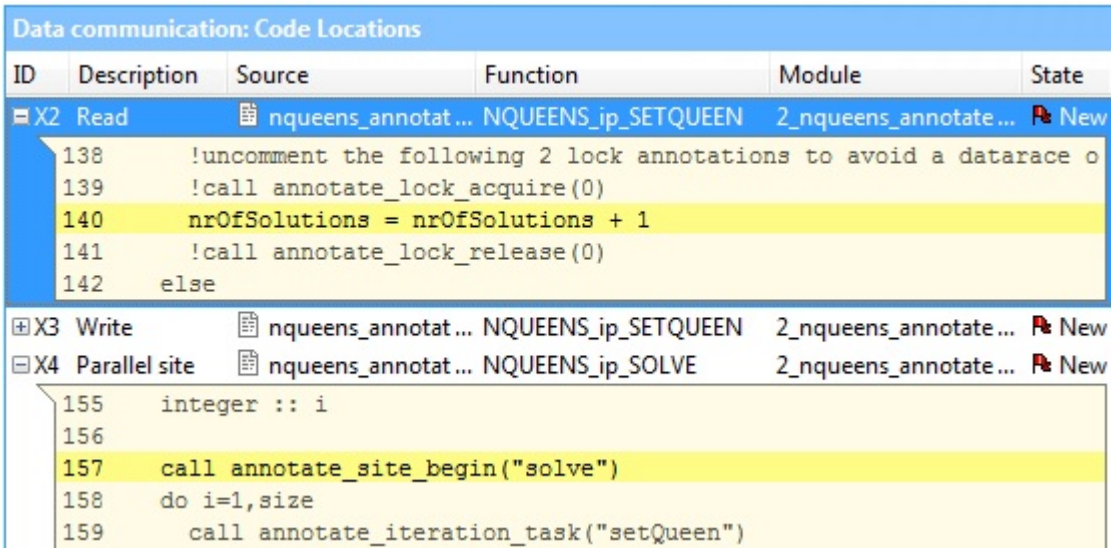
1	The **Correctness Report** window is the starting point for viewing potential parallel problems.
2	The **Problems and Messages** pane shows the observed problems. In this case, there are two problems and one informational message, whose names are listed under the **Type** column.
3	For the currently selected Problem or Message, details appear in the **Code Locations** pane (lower-left corner). In this case, details appear for the **Data Communication** 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 Report and Correctness Source Output

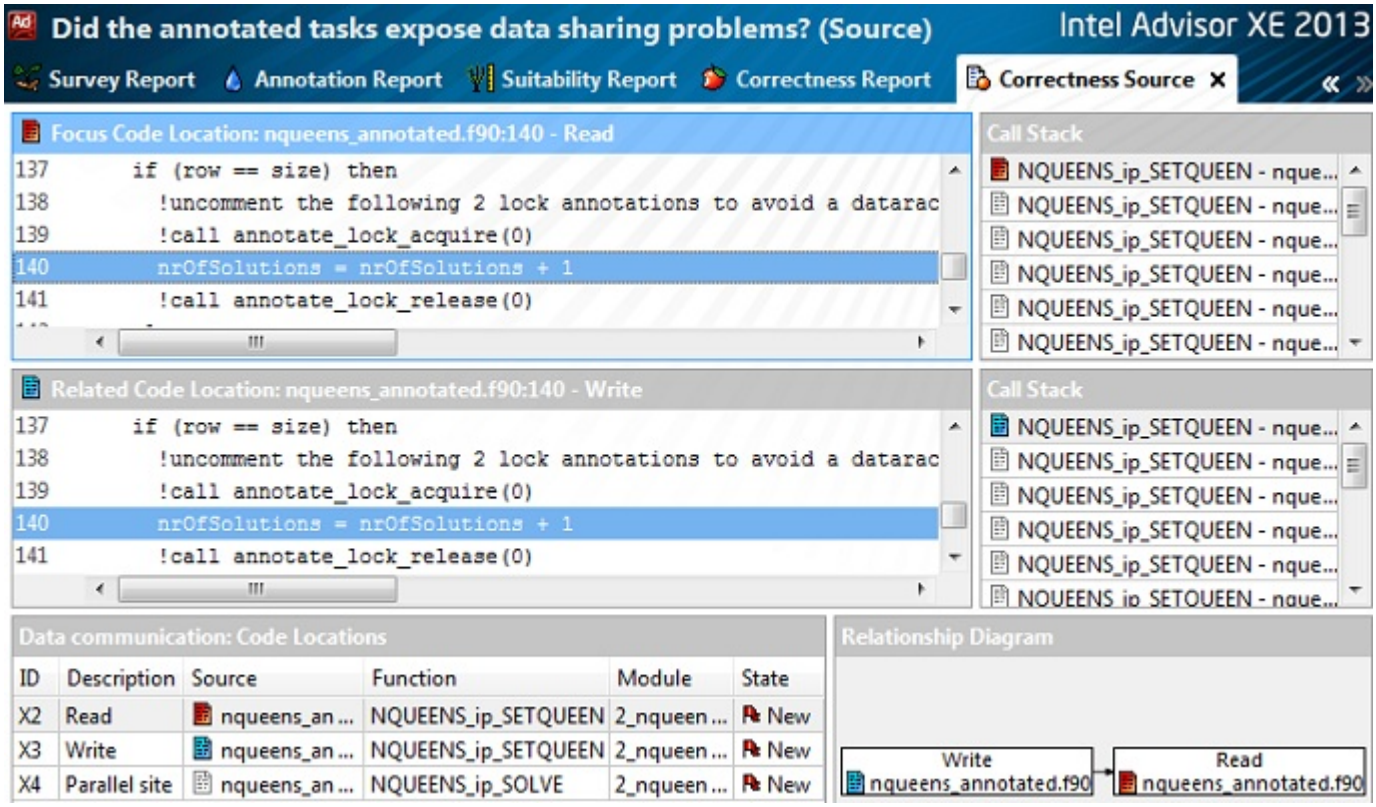
The **Problems and Messages** pane lists the data sharing problems found. The column with the icon lists the severity of each problem, such as error, warning, or informational remark message. Click a line under the **Problems and Messages** column and the code locations associated with that problem or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

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

- Click the or icon next to one of the **Code Locations** to view or hide a few lines of its source code snippet, such as to hide **X3**:



- Double-click one of the **Code Locations** (or right-click and select **View Source**), such as **X4**. This opens the **Correctness Source** window, which lets you view source for the **Focus Code Location** (red icon), **Related Code Location** (blue icon), their call stacks, a list of all **Code Locations**, and a diagram of Code Location **Relationships**.



Key Terms

data race

Next Step

Decide on the Proposed Tasks

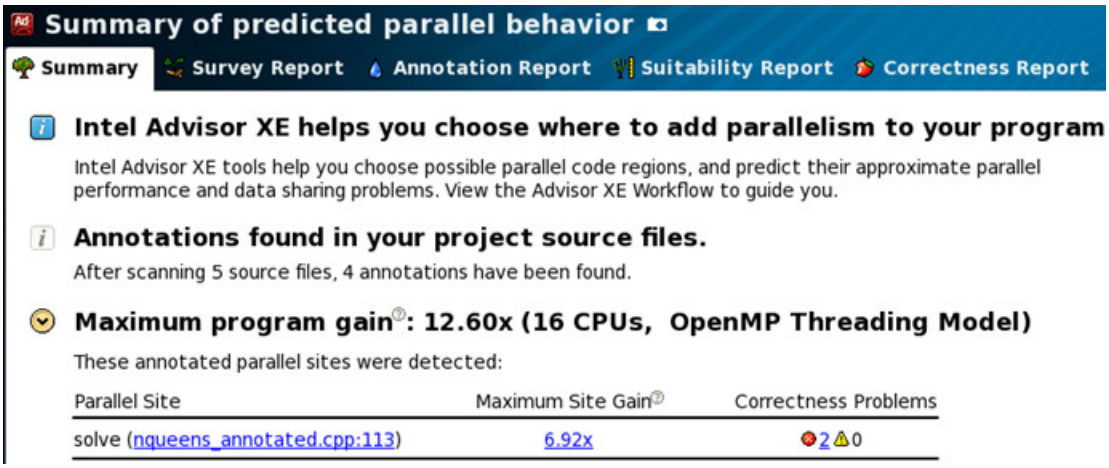
Decide on the Proposed Tasks

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

You can create a read-only result snapshot after you collect data from one or more Intel Advisor tools, perhaps after you have decided on the tasks for the current parallel site or to capture your current analysis state.

View the Summary Window

After you run the Suitability and Correctness tools, the **Summary** window 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 parallel site (`solve`) has a good **Maximum Site Gain** based on 16 CPUs, and it has only 2 Correctness (data sharing) problems and no warnings. Because this site provides a good performance benefit and a small number of sharing problems, keep your proposed site and task code regions.

As you fix sharing problems, you should 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 synchronization.

Creating a Read-only Result Snapshot

Only the active result for a project can be modified by collecting new data. To create a snapshot of the active result and save its data in a read-only result:

- Display the active result (not a read-only result snapshot).
- Click the button next to the window caption. The **Create a Result Snapshot** dialog box appears.
- Type the **Result name** of the Intel Advisor read-only result. Provide a unique name, perhaps by adding an identifying suffix within the result name.
- Click **OK**. With a large result, you may need to wait as the read-only result gets created.

You can visually compare a read-only result with the current active result or other read-only results. The read-only result appears in the Intel Advisor GUI project navigator. In the result tab, the words **(read-only)** appear after the name of a read-only result.

Key Terms

parallel site, task

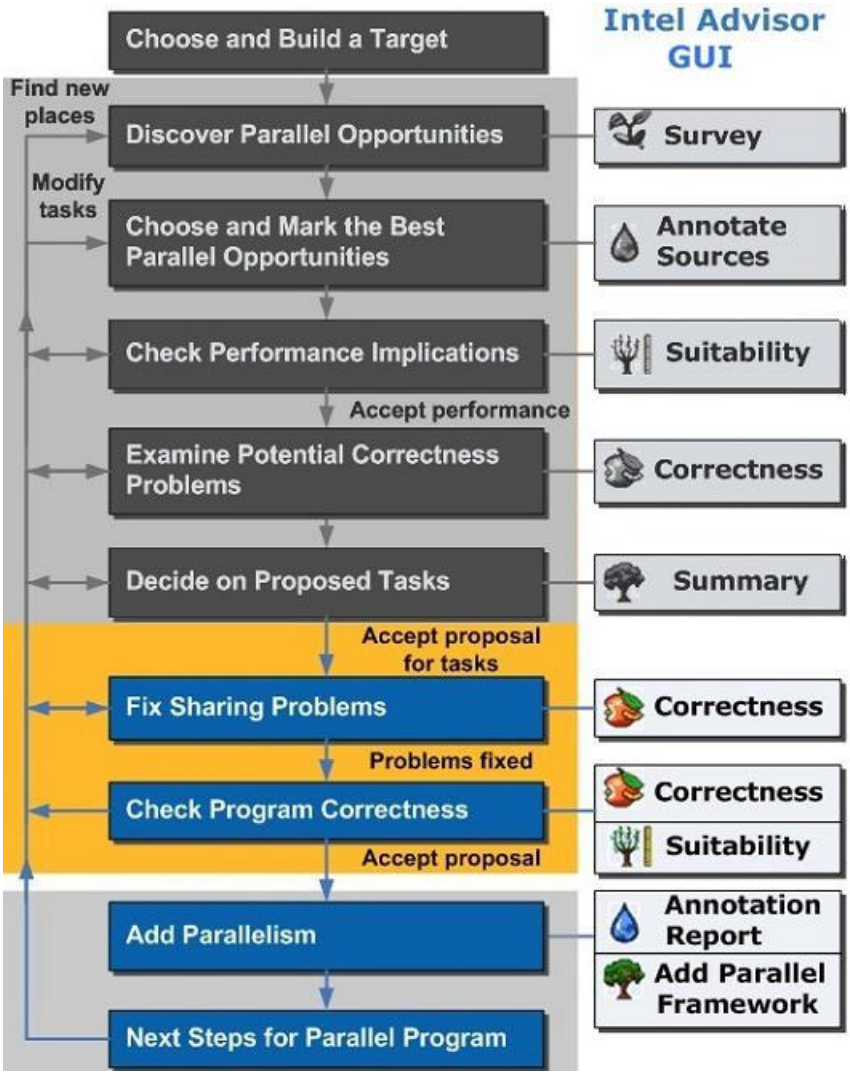
Next Step

Fix Sharing Problems and Add Parallelism

Fix Sharing Problems and Add Parallelism

4

This is the third (final) step to use the parallel design capabilities provided by the Intel® Advisor. The following diagram shows the last few steps in the workflow, where you fix data sharing problems and add parallelism to an annotated program.



Step 1: Target Program for Suitability and Correctness Tool Analysis	Continue to use the same project and target application from the previous part of this tutorial, including the **Correctness Report**.
Step 2: Fix Sharing Problems	View the **Correctness Report** to understand how to Fix Sharing Problems.

Step 3: Check Program Correctness	Rebuild your program and run the Correctness tool again to make sure the corrected sharing problems have been fixed to Check Program Correctness.
Step 4: Add Parallelism	Select a high-level parallel framework. For the proposed parallel code regions, replace Intel Advisor annotations with parallel framework code to Add Parallelism.
Step 5: Next Steps for the Parallel Program	After successfully adding parallelism to relevant parts of your program, consider using other Intel software suite tools to check and tune your parallel program, as the Next Steps for the Parallel Program.

Intel® Advisor GUI: Build the Application and Open the Project


To use Intel Advisor to help you fix sharing problems and add parallelism, you need to:

- Get software tools (completed in previous parts of this tutorial)
- Verify optimal compiler/linker options (done in a previous part of this tutorial).
- Open the Intel Advisor GUI.
- Open an existing project to view its result.

NOTE
This part of this tutorial assumes you have completed the previous part that predicts parallel behavior of your serial, annotated application. This includes installing Intel Advisor and running the Suitability and Correctness tools.

Verify Optimal Compiler/Linker Options

As explained in the previously, you can use these Intel Advisor tools:

- Suitability tool to predict your serial application's parallel performance.
- Correctness tool to predict your serial application's parallel data sharing problems.

Applications compiled/linked using the specified options produce the most accurate, complete results. If you completed the previous part of this tutorial, you will have the required build settings, such as:

- A release build with debug information and moderate optimization for the Suitability and Survey tools.
- A debug build with debug information and no optimization for the Correctness tool.

Open the Intel Advisor GUI

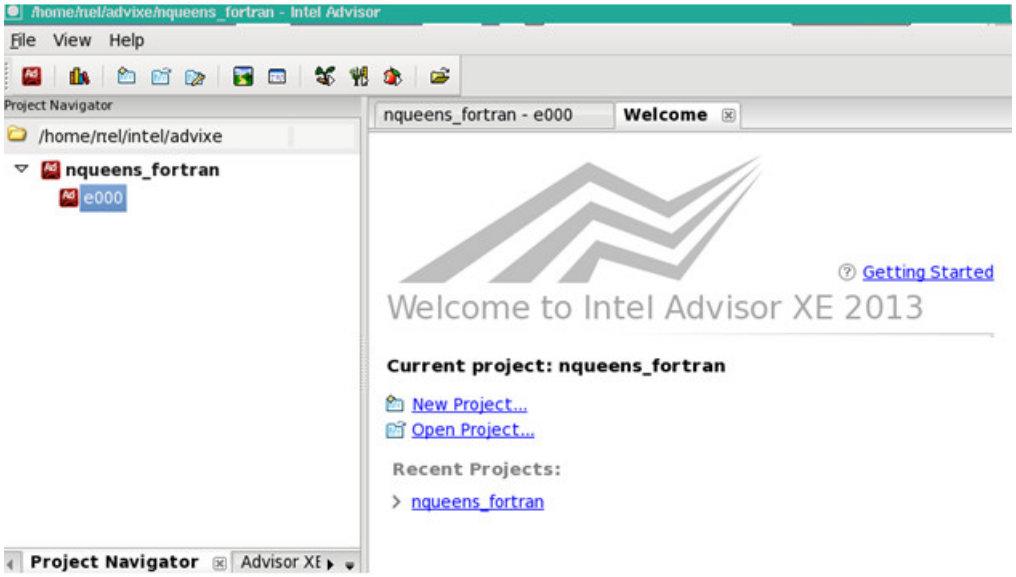
1. If you have not already done so, type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or equivalent path or `.csh` script file) to set up your command-line environment.
2. Type `advixe-gui` to open the Intel Advisor GUI.

NOTE
After you set up your command line environment, you can also use the Intel Advisor command line interface, `advixe-cl`, to collect data analysis or create reports by using the command line or scripts. For help, type: `advixe-cl --help`.

Open the Project

In a previous part of this tutorial, you created a project and built the annotated serial version of the `nqueens_Fortran` sample. You may need to look at the previously collected results while completing this part.

1. The previously opened project should be re-opened and its name should appear in the title bar and in the Project Navigator. If it was closed, either click the `nqueens_fortran` link below Recent Projects: in the **Welcome** page, or choose **File** > **Recent Projects...** and select the recently created project name.
2. The following screen appears showing the opened project's name in the title bar (your screen may differ slightly) and the **Welcome** page:



3. To open a previously collected result, click **File** > **Recent Results...** and select the recently created result name **...nqueens_Fortran/e000**, or right click the result (such as **e000**) and select **Open Result** in the **Project Navigator**. The result will appear. For example, click the **Correctness Report** button to view the Correctness Report data needed to fix sharing problems.
4. To view and modify your open project's properties, choose **File** > **Project Properties...** or click the  icon on the toolbar.

Key Terms

parallel framework

Next Step

Fix Sharing Problems

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` subroutine.
- **Data communication** problem type. This is located in the `SetQueen` subroutine.

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 code location** and **Related code locations**.

To open the 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 Code Location** pane. You can view the **Related Code Locations** or simply double-click the source code to launch the editor with that file opened at the same position. Within the Intel Advisor GUI, click **File > Options > Editor** to choose the Linux* editor displayed for each source language (or specify the VISUAL or EDITOR environment variable).
3. In each case, scroll up to locate the subroutine containing the source code:
   - The **Memory reuse** problem originates in the `solve` subroutine.
   - The **Data communication** problem is located in the `SetQueen` subroutine.
4. Read the source 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 the result tab for that project.

Fix the Memory Reuse Sharing Problem

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

```
! Main solver routine
subroutine solve (queens)
  implicit none
  integer, intent(inout) :: queens(:)
  integer :: i

  call annotate_site_begin("solve")
  do i=1,size
    call annotate_iteration_task("setQueen")
    ! try all positions in first row
    call SetQueen (queens, 1, i)
  end do
  call annotate_site_end()
end subroutine solve
```

The call to `SetQueen` is located within the parallel site. Examine the `SetQueen` subroutine code and you will notice that the current version of the `setQueen` subroutine uses the same array passed from the `solve` subroutine:

```
recursive subroutine setQueen (queens, row, col)
  implicit none
  integer, intent(inout) :: queens(:)
```

```
  integer, intent(in) :: row, col
  integer :: i
  integer, volatile :: j

  !In order to avoid a data race on the "queens" array create a local copy.
  !Uncomment the statements using the "lcl_queens" array and comment the
  !statements using the "queens" array.
  !integer :: lcl_queens(ubound(queens,dim=1))

  ! Make copy of queens array
  !lcl_queens = queens

  do i=1,row-1
    ! vertical attacks
    !if (lcl_queens(i) == col) return
    if (queens(i) == col) return
    ! diagonal attacks
    !if (abs(lcl_queens(i)-col) == (row-i)) return
    if (abs(queens(i)-col) == (row-i)) return
    end do

  ! column is ok, set the queen
  !lcl_queens(row) = col
  queens(row) = col

  if (row == size) then
    !uncomment the following 2 lock annotations to avoid a datarace on the nrOfSolutions variable.
    !call annotate_lock_acquire(0)
    nrOfSolutions = nrOfSolutions + 1
    !call annotate_lock_release(0)
  else
    ! try to fill next row
    do j=1,size
      !call setQueen (lcl_queens, row+1, j)
      call setQueen (queens, row+1, j)
    end do
  end if
end subroutine SetQueen
```

This placement causes the site code and each task to share the same array! This is called incidental or accidental data sharing:

```
!In order to avoid a data race on the "queens" array create a local copy.
!Uncomment the statements using the "lcl_queens" array and comment the
!statements using the "queens" array.
!integer :: lcl_queens(ubound(queens,dim=1))

! Make copy of queens array
!lcl_queens = queens
```

Instead of sharing this `queens` array, each task should have its own private copy of this array. You need to comment out the lines that use the `queens` array and uncomment lines that use the `lcl_queens` array as well as the lines above that declare and assign it:

```
recursive subroutine setQueen (queens, row, col)
  implicit none
  integer, intent(inout) :: queens(:)
  integer, intent(in) :: row, col
  integer :: i
```

```
  integer, volatile :: j

  !In order to avoid a data race on the "queens" array create a local copy.
  !Uncomment the statements using the "lcl_queens" array and comment the
  !statements using the "queens" array.
  integer :: lcl_queens(ubound(queens,dim=1))

  ! Make copy of queens array
  lcl_queens = queens

  do i=1,row-1
    ! vertical attacks
    if (lcl_queens(i) == col) return
    !if (queens(i) == col) return
    ! diagonal attacks
    if (abs(lcl_queens(i)-col) == (row-i)) return
    !if (abs(queens(i)-col) == (row-i)) return
    end do

  ! column is ok, set the queen
  lcl_queens(row) = col
  !queens(row) = col

  if (row == size) then
    !uncomment the following 2 lock annotations to avoid a datarace on the nrOfSolutions variable.
    !call annotate_lock_acquire(0)
    nrOfSolutions = nrOfSolutions + 1
    !call annotate_lock_release(0)
  else
    ! try to fill next row
    do j=1,size
      call setQueen (lcl_queens, row+1, j)
      !call setQueen (queens, row+1, j)
    end do
  end if
end subroutine SetQueen
```

By assigning a private copy to each task using the `lcl_queens` array, the Memory Reuse problem will be fixed.

Fix the Data Communication Sharing Problem

To fix the **Data communication** problem located in the `SetQueen` subroutine, examine that subroutine's code:

```
...
  ! column is ok, set the queen
  lcl_queens(row) = col
  !queens(row) = col

  if (row == size) then
    !uncomment the following 2 lock annotations to avoid a datarace on the nrOfSolutions variable.
    !call annotate_lock_acquire(0)
    nrOfSolutions = nrOfSolutions + 1
    !call annotate_lock_release(0)
  else
    ! try to fill next row
    do j=1,size
      call setQueen (lcl_queens, row+1, j)
      !call setQueen (queens, row+1, j)
    end do
```

```
  end if
end subroutine SetQueen
```

The line that increments the variable *nrOfSolutions* is located in the recursively called `SetQueen` subroutine, 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 `!call annotate_lock_acquire(0)` and `!call annotate_lock_release(0)` annotation lines.

After you uncomment the pair of lock annotations calls, your code should look like the following:

```
call annotate_lock_acquire(0)
nrOfSolutions = nrOfSolutions + 1
call annotate_lock_release(0)
```

Once you have modified your code to fix both data sharing problems:

1. Rebuild the target.
2. Run the Correctness tool again and verify that the problems have been fixed. That is, the only line displayed in the **Problems and Messages** pane is **Parallel site information**.
3. If they have been fixed, run the Suitability tool again using a Release build target to check the performance implications of the modified sources.

Key Terms

data race, synchronization

Next Step

Check Program Correctness

Check Program Correctness

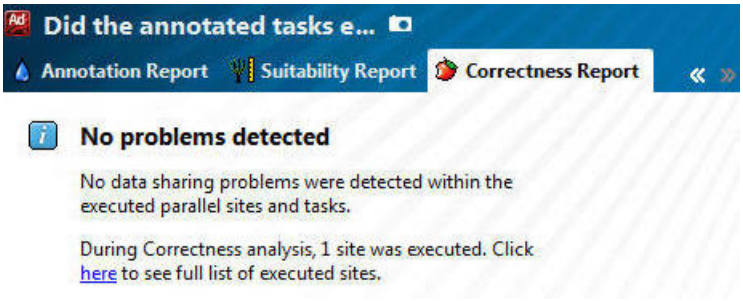
 To check the Correctness of the serial program, run the Intel Advisor Correctness tool. Do one of the following:

- Click the  **Collect Correctness Data** button in **Advisor XE Workflow** tab (below **4. Check Correctness**) or in the **Correctness Report** window's command toolbar. To hide or show the command toolbar, click the  or  buttons in the upper-right of the Correctness Report.
- Click the  icon in the Intel Advisor toolbar.

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` program completes, similar to the following (the time on your system will be different):

```
Starting serial recursive solver for size 8
Number of solutions: 92
Calculations took 522ms.
Correct Result!
```

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



In this case, there are no problems reported by the Correctness tool. If your **Correctness Report** window appears with a reported problem instead of this message, carefully check the source code changes described in the previous topic (Fix Sharing Problems), rebuild, and run the Correctness tool again.

You can click the here link to view the **Parallel site information** remark that indicates the code region actually executed by your program. The reported **Parallel site information** shows where the parallel site begins, which is marked by the annotation `call annotate_site_begin("solve")`.

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** and/or **Suitability Report** windows.

Next Step

Add Parallelism

Add Parallelism

To add parallelism to the relevant parts of your program:

- Replace annotations with parallel framework code.
- Build a parallel version of your program.
- Run the parallel 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 editor to replace Intel Advisor annotations with parallel framework code. For your convenience, you only need to view the source files after you use `make` to build the executable:

- In the source file `nqueens_omp.f90` that is built into the `3_nqueens_omp` executable file, OpenMP* code has been added. OpenMP is the high-level parallel framework supported by Intel Advisor for Fortran programs.

Build and Examine the Parallel Program

Use the `nqueens` Fortran sample to build and run one of the parallel programs, which use the OpenMP parallel framework.

1. In a terminal session, locate the Intel Advisor installation directory root on your system. The default installation location is `/opt/intel/advisor_xe_2013`.
2. Type `source /opt/intel/advisor_xe_2013/advixe-vars.sh` (or equivalent path) to set up your `bash` shell environment. With a different shell, source the `advixe-vars.csh` script.
3. If you need to set up your compiler environment, do so now.
4. Verify that the *ADVISOR_XE_2013_DIR* environment variable is set - for example, type: `env | grep ADVISOR_XE_2013_DIR`. If it is not set, define it by using an `export` command: `export ADVISOR_XE_2013_DIR=/opt/intel/advisor_xe_2013`.
5. Change directory to the `nqueens_fortran/` directory (created in a previous tutorial).
6. Type `make 3_nqueens_omp` to build a OpenMP parallel version of the `nqueens` Fortran sample application.

View the parallel source code:

- You used the OpenMP project, so open the source file `nqueens_omp.f90`. View the OpenMP `#include` file and related OpenMP code, such as the `!OMP PARALLEL DO` line in the `solve` subroutine. Also notice that the lock annotations have been replaced by an `!OMP ATOMIC` lock for the `nrOfSolutions = nrOfSolutions + 1` statement in the `setQueen` subroutine. For more information about OpenMP, see Intel Advisor help topics under Adding Parallelism to Your Program or locate the OpenMP documentation in your compiler documentation, such as the Intel® Composer XE documentation directory.

With your own program, while you add parallel framework code, view the **Annotation Report** window to help you locate the remaining Intel Advisor annotations that need to be replaced with parallel framework code. For help completing this step for your own program:

- Open the **Advisor XE Workflow** tab.
- Click the ⊞ button below **5. Add Parallel Framework**.
- View the instructions. Click the links to display topics in Intel Advisor help.

Run the Parallel Program

1. In the same terminal session, change directory (`cd`) to the `nqueens_fortran` directory that you created when you extracted the `nqueens_Fortran.tgz` file.
2. Run the sample parallel application that you built previously using one of the parallel frameworks. For example, type: `./3_nqueens_omp`.
3. Check for output similar to the following:

```
-bash-4.1$ ./3_nqueens_omp
 Usage: 3_nqueens_omp[_debug] boardSize
 Using default size of 14
Starting OpenMP solver for size 14 with 4 thread(s)
Number of solutions: 365596
Calculations took 3477ms.
Correct Result!
```

The displayed execution time to run the parallel program on a 4-core system is significantly less than the time required to run the serial version of the program, which was about 7160 ms. The execution times will be different on your system.

The difference between the serial and parallel execution time depends on multiple factors:

- The number of cores available on your system.
- How much of the original program's execution time was placed within a parallel site(s) and the characteristics of the tasks. Intel Advisor Suitability and Correctness tools use annotations to predict your serial program's parallel behavior, which lets you experiment with your sites and tasks - before you add any parallel code.
- Parallel overhead, type of locks, thread characteristics, and other factors - see Next Steps for the Parallel Program

Key Terms

parallel framework

Next Step

Next Steps for the Parallel Program

Next Steps for the Parallel Program

You have used Intel Advisor to help you add parallelism to the relevant parts of the `nqueens` sample program.

Continue with Intel® Parallel Studio XE or Intel® Cluster Studio XE Tools

As it examines a serial version of a program, Intel 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 Intel Advisor annotations, you convert the annotations to parallel framework code.

Before deploying your parallel (multithreaded) program, test it for correctness and verify its performance by using the Intel® Inspector and Intel® VTune™ Amplifier tools. In addition to Intel Advisor, the Intel® Parallel Studio XE, Intel® Cluster Studio XE, or similar product suites also include other useful tools:

- The Intel® Composer XE contains the OpenMP* high-level parallel framework used in the previous step. In addition, the Intel Composer XE includes the Intel® C++ Compiler, the Intel® Fortran Compiler, the efficient Intel® Integrated Performance Primitives Library (for C/C++ use), the efficient Intel® Math Kernel Library (for Fortran use), the Intel® Threading Building Blocks parallel framework (for C++ use), and Intel® Cilk™ Plus parallel framework (for C/C++ use).
- The Intel® Inspector uses the parallel version of your program to find correctness issues, such as shared data problems; it can also find certain problems related to memory use.
- The Intel® VTune™ Amplifier uses the parallel version of your program to discover hotspots and provide other tuning information. It can help you measure performance speed-up, identify additional hotspots, and help you adjust locks for improved parallel performance.

Key Terms

parallel framework

Next Step

Tutorial Summary

5 Tutorial Summary

 Here are some important things to remember when using Intel® Advisor to find where to add parallelism.

Prepare Target Executable for Tool Analysis

If you used the Intel Advisor GUI: You built and ensured the application runs on your system outside the Intel Advisor, and created a project to hold analysis results.

In either case, you built a release build with debug symbol information and moderate optimization for the Survey and Suitability tools, and a debug build with debug symbol information and no optimization with a limited data set for the Correctness tool.

Discover, Choose, and Mark Parallel Opportunities

You used the **Summary** window to learn which loops or functions consume significant CPU time to be possible candidates for parallelism. You also used the **Survey Report** window to examine possible areas where your application spends its time. You decided where to add Intel Advisor annotations for a parallel site and a parallel task and learned how to add them.

Check Performance Implications and Fix Potential Correctness Problems

You used the Suitability tool to predict your annotated application's approximate parallel performance. You examined the **Suitability Report** to predict the overall program gain and each parallel site's performance gain and task characteristics.

You used the Correctness tool to predict your annotated application's likely data sharing problems. You examined the **Correctness Report** to examine each data sharing problem, including related code locations and the source code in the **Correctness Source** window to understand the cause. You were able to Fix Sharing Problems by modifying (refactoring) your source code and/or using synchronization techniques to eliminate the predicted data sharing problems.

Decide on the Proposed Tasks

After you ran the Suitability and Correctness tools, you examined the **Summary** window to decide whether the proposed sites and tasks provide sufficient predicted performance (Suitability) gain as well as a set of data sharing problems (Correctness) that likely can be fixed.

Add Parallelism and Next Steps for the Parallel Program

You selected a high-level parallel framework. For the proposed parallel code regions, you replaced Intel Advisor annotations with parallel framework code. After successfully adding parallelism to relevant parts of your program, you considered using other suite tools to check and tune your parallel program.

Next step: Congratulations, you have completed the Intel Advisor tutorial!

With your own application, you can:

- Profile your program with the Survey tool to find where it spends its time.
- Select a parallel site and define its tasks by inserting Intel Advisor annotations.
- Predict the parallel behavior of your annotated program using the Suitability and Correctness tools.
- Identify and fix possible sharing problems, and run the Suitability and Correctness tools again.

- Select a parallel framework. Replace annotations with parallel code.
- Use the other tools in Intel Parallel Studio XE, Intel Cluster Studio XE, or similar studio tool suites to check and tune your parallel program.

6 Key Terms


The following terms are used throughout this tutorial.

annotation: Intel® Advisor annotations are `call` statements that identify certain information to Intel Advisor tools, such as the location of proposed parallel sites. To insert annotations into your source code, you can copy code snippets from the annotation assistant pane into your code editor.

data race: A bug that can occur after 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 parallel execution errors that are difficult to detect and reproduce. Using the Correctness tool helps you predict and fix likely data races *before* you add parallelism.

hotspot: A small code region that consumes much of the program's run time, such as a loop. Hotspots can be identified by a profiler, such as the Intel Advisor Survey tool or the Intel® VTune™ Amplifier. Hotspots are code regions that help you select candidates for adding parallelism that you mark as a *parallel site* and one or more *task(s)*.

parallel framework: A combination of libraries, language features, or other software techniques that enable code for a program to execute in parallel. Examples for C/C++ include Intel® Threading Building Blocks and Intel® Cilk™ Plus, which are both included with Intel® Composer XE. The OpenMP* parallel framework for C/C++ and Fortran code is available with multiple compilers. After you find possible places to add parallelism with Intel 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* or POSIX* Threads require that you directly create and control threads.

parallel site: A region of code that contains one or more tasks that may execute in parallel. An effective parallel site typically contains a hotspot 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 hotspot, 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 or mutex can be used to restrict access to shared data to prevent a data race.

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

task: A portion of time-consuming code and its data that can be executed in one or more parallel threads to distribute work. One or more tasks execute within a *parallel site*.