
# Design Parallel Performance with Less Risk and More Impact

# Introduction

This guide helps you experiment with adding parallelism to your serial applications using the Intel® Advisor XE. You get hands-on experience with our sample code in a 50-minute exercise that shows you the power of the Intel Advisor XE. You can then explore other Intel® Parallel Studio XE components on your own. The final section is packed with resources to help you in the process of threading.

# What Is Intel Advisor XE?

Intel Advisor XE is a thread prototyping tool for software architects. It lets you evaluate the performance implications and the refactoring cost of parallelizing a code region before committing resources to that endeavor. Because you only mark where the parallelism could be, your application stays serial. Thus, any changes made to ease the transition to parallel code can be verified with your existing test system.

# Try It Yourself

## Install and Set Up Intel Parallel Studio XE

1. Download and install an evaluation copy of the Intel Parallel Studio XE.

## Install and Open the tachyon_Advisor Sample Application

Tachyon is a 2-D ray-tracing/rendering program. After computing and displaying the image and the elapsed time (about 10 seconds), it waits another five seconds so you can view the window before it closes.

The source, tachyon_serial.cpp, contains three functions. thread_trace() initializes the static variables before having parallel_thread() draw the picture, while render_one_pixel() determines the color to emit. Loops in parallel_thread() and render_one_pixel() need close attention.

This document walks you through using the Intel Advisor XE to explore parallelizing these functions.

1. Download the tachyon_Advisor.zip sample file to your machine. This is a C++ console application created with Microsoft Visual Studio* 2010.

   Intel Advisor XE samples can be used with Visual Studio 2010, 2012, and 2013.
2. Extract the files from the tachyon_Advisor.zip file to a writable directory or share on your system, such as a C:\Work\Samples folder.
3. Load the solution into Visual Studio by selecting **File>Open>Project/Solution...** and navigate to the tachyon_Advisor.sln file to see something similar to Figure 1.

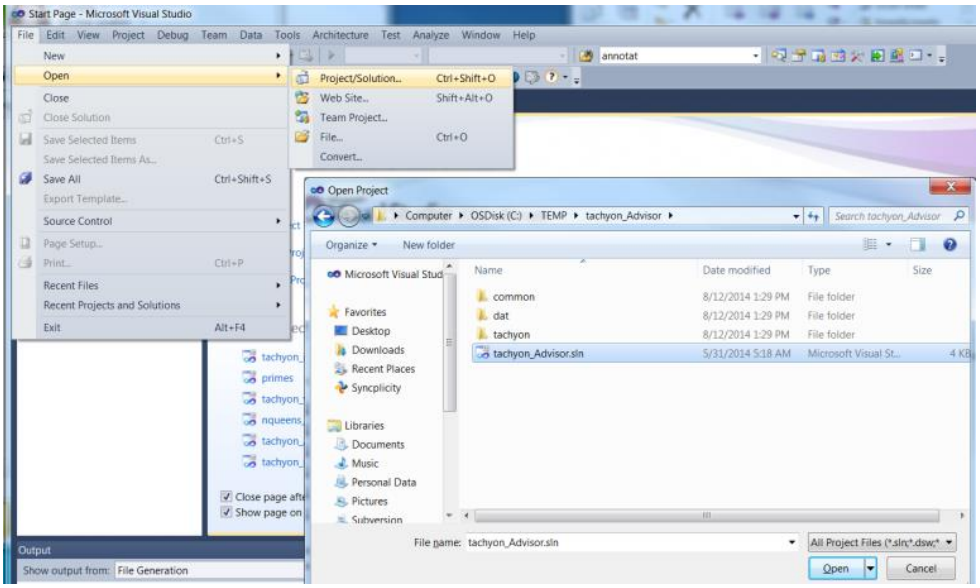



*Figure 1*

The tachyon_Advisor solution contains several projects. In this guide, you will modify only the 1_tachyon_serial and tachyon.common projects shown in Figure 2. The others (2_tachyon_annotated, 3_tachyon_cilk, 3_tachyon_omp, and 3_tachyon_tbb) are not used in this guide.

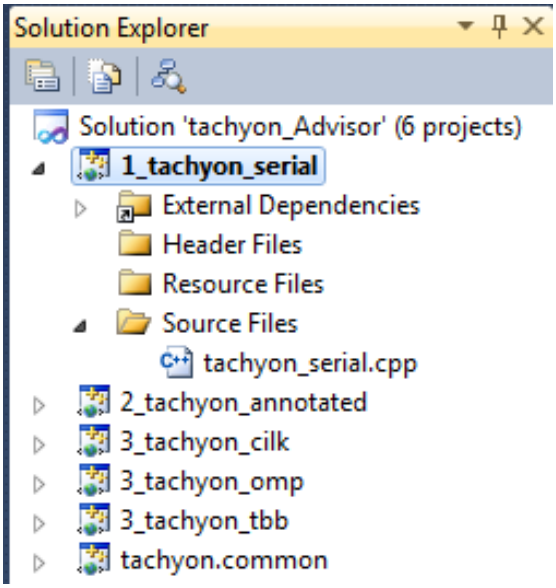



*Figure 2*

1_tachyon_serial is configured to reference the Intel Advisor XE default installation include directory. To view the settings, right-click the project in the Solution Explorer and choose Properties. The relevant configuration file is highlighted in Figure 3.

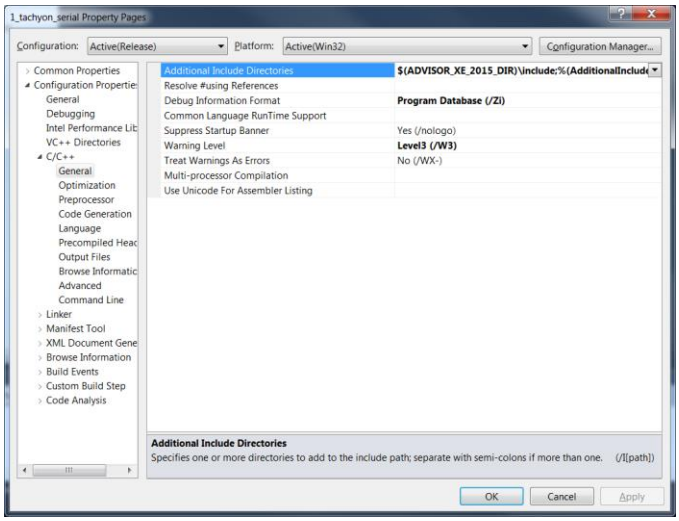



*Figure 3*

### Activate Intel Advisor XE

You can activate the Intel Advisor XE through its toolbar, the Visual Studio tools menu, or the Solution Explorer project context menu.

Click the Workflow icon (first from the left) in the Intel Advisor XE toolbar shown in Figure 4 to open the Intel Advisor XE Workflow shown in Figure 5. This workflow is available to guide you through the methodology of experimenting with parallelism.

*Figure 4*

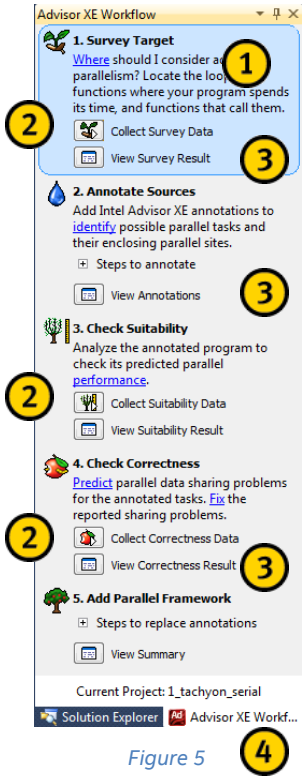



*Figure 5*

1. Current step (highlighted)
2. Shortcuts to activate the Intel Advisor XE tools
3. Shortcuts to open the Intel Advisor XE window and view the tools reports
4. Current start-up project

## Survey Target: Identify Call Sites and Loops That May Benefit From Parallelism

The first workflow step: Run the Intel Advisor XE Survey tool to identify call sites and loops that consume most of your program's time. This focuses your attention on the hot call trees and loops as potential locations to experiment with parallelization.

1. With 1_tachyon_serial project defined as the solution Start-Up Project, build the Release configuration.
2. Click the Collect Survey Data icon under **1. Survey Target** in the Intel Advisor XE Workflow.

   The Survey tool identifies two loops in parallel_thread( ) that consume significant CPU time. Figure 6

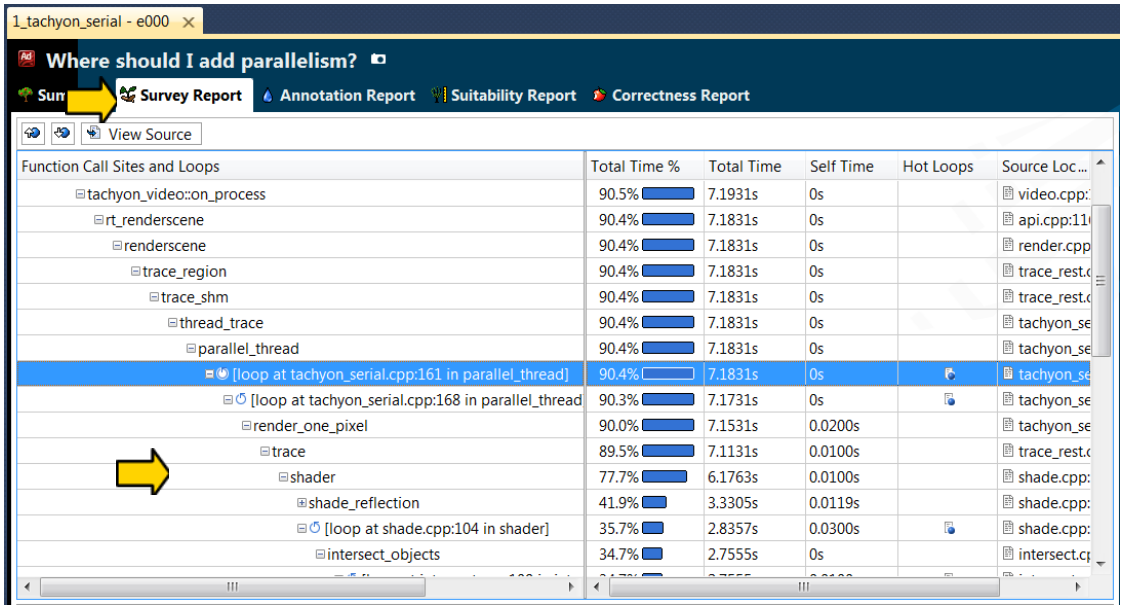



*Figure 6*

3. Double-click the row for the outer loop to see details in the Survey Source view.

   We can see these nested loops account for a significant percentage of program execution time. After evaluating the data accesses within these loops and the amount of time spent, we can identify the body of the outer loop as a good candidate for experimentation: each of the picture's horizontal lines will be rendered in parallel. Figure 7

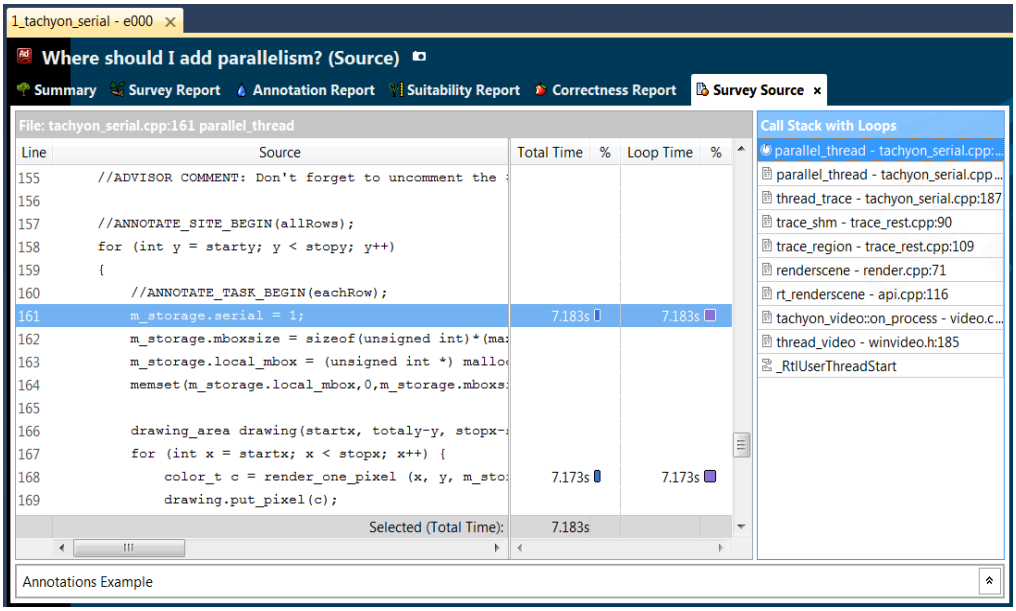



*Figure 7*

### For Additional Assistance

- Check the help tip at the top of the Workflow pane.
- Review the documentation references in the Workflow.
- Double-click the Intel Advisor XE icon at the top of the My Advisor Results window for an overview of the current display. Figure 8

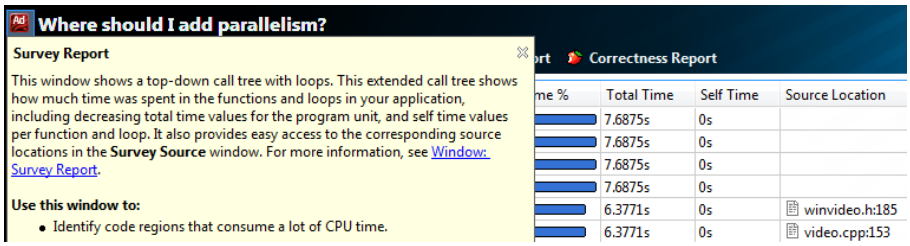



*Figure 8*

- Right-click a row or column of the My Advisor Results window and select **What Should I Do Next?** from the context menu. Figure 9

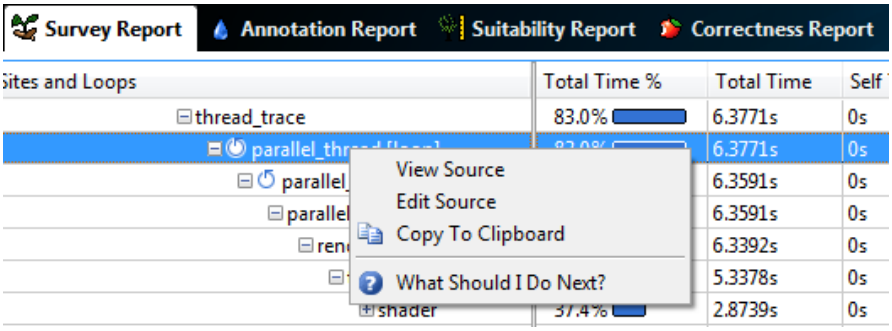



*Figure 9*

### To Navigate to an Editable Code Window

1. Double-click a row in an Intel Advisor XE window.
2. Right-click to activate the context menu, and then select **Edit Source**.

## Annotate Sources: Insert Intel Advisor XE Annotations to Model Parallelism

Now that we have a good idea where to create our parallel task (the body of the outer loop), we can use the Intel Advisor XE to model the performance of the application as if it were parallelized. Remember that our application remains serial while we experiment with parallelism.

To define our parallel experiment for Intel Advisor XE, we continue with: **2. Annotate Sources** in the Intel Advisor XE Workflow.

Expand the **Steps to annotate** section in the workflow to see a summary of how to annotate the sources. Figure 10

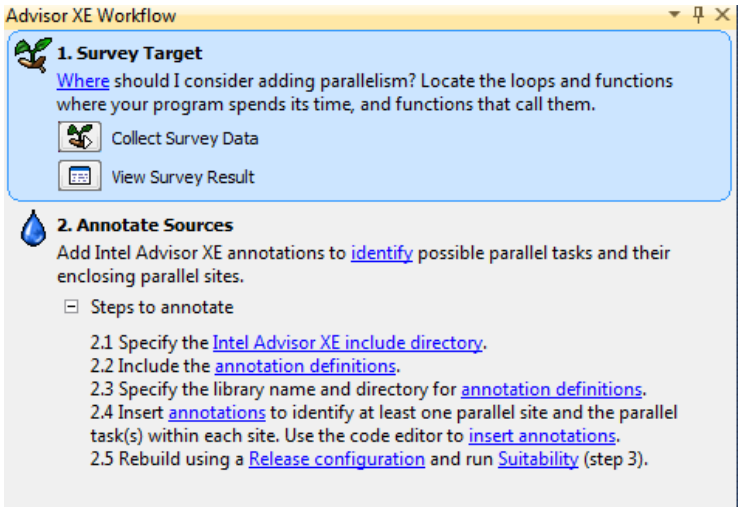



*Figure 10*

## Include the Header File advisor-annotate.h.

We will reference the advisor-annotate.h header file from the Intel Advisor XE installation directory. Since the project include path property is already modified, we only need to add the include statement to the source file.

From the editor window, navigate to where we want to include the header file, right-click to open the editor context menu, and select **Intel Advisor XE > Insert Annotation Definitions Reference**. Figure 11

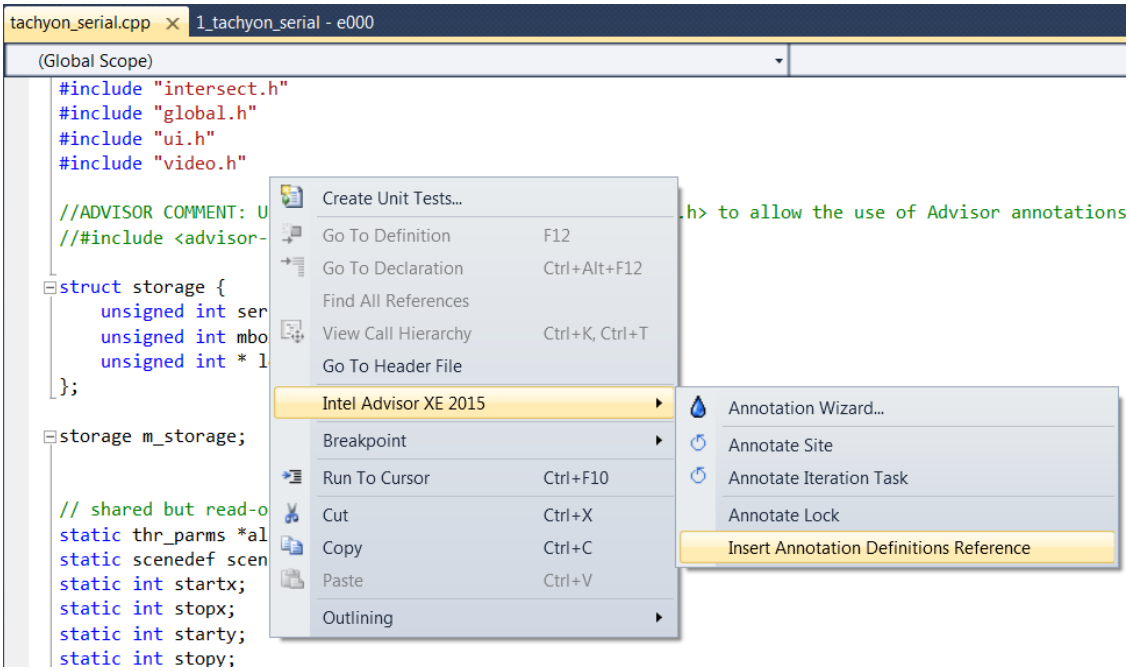



*Figure 11*

## Define the Parallel Task by Inserting a Task Annotation

From within the editor window, select the body of the outer loop, `for (int y = starty; y < stopy; y++)`, right-click to open the editor pop-up menu, and select **Intel Advisor XE > Annotation Wizard**. Figure 12

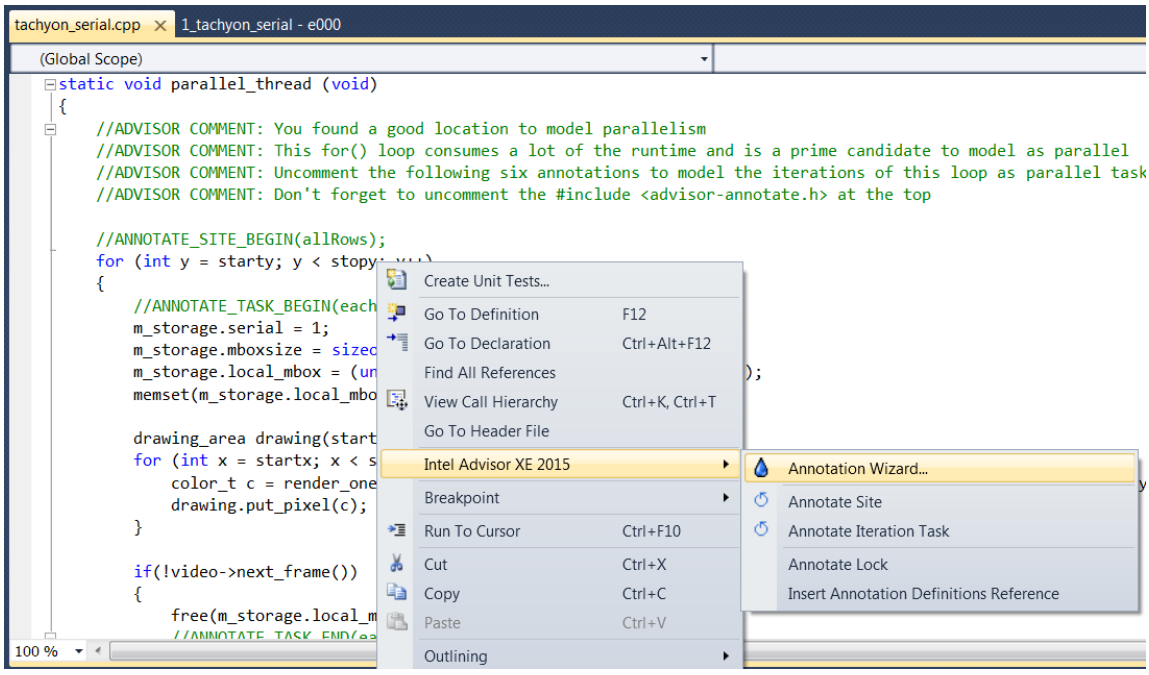



*Figure 12*

Using the Annotation Wizard, insert the Iteration Task annotation with the label *MyTask1*. This annotation tells the Intel Advisor XE analysis tools that the iterations of this loop may run concurrently in this parallelized model. Figure 13

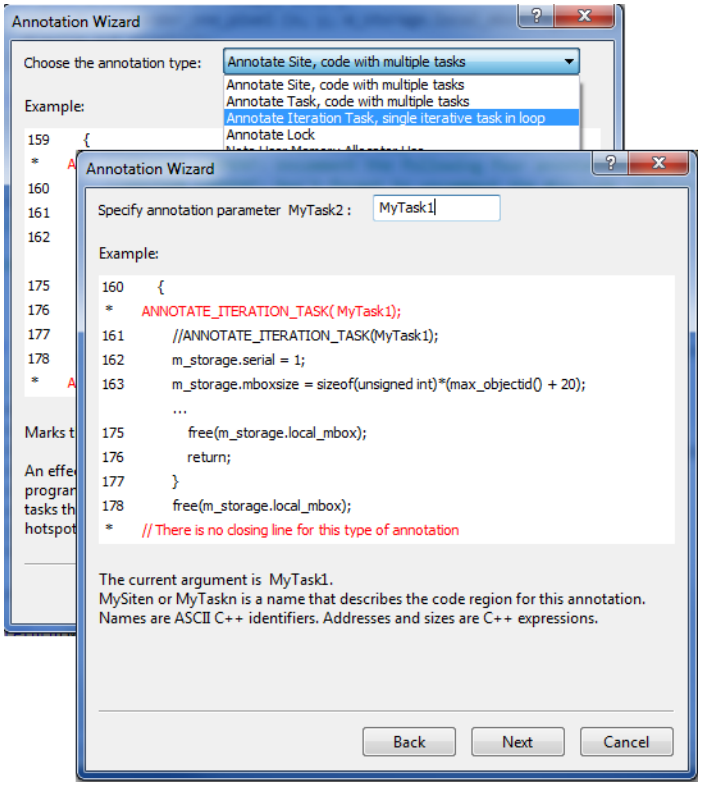



*Figure 13*

## Surround the Task With a Parallel Site Annotation

A parallel site is a region of code that contains one or more tasks that may execute in parallel.

Again, select the entire loop, right-click, and select **Intel Advisor XE > Annotate Site**. This annotation uses the label generated by the Intel Advisor XE. Figure 14

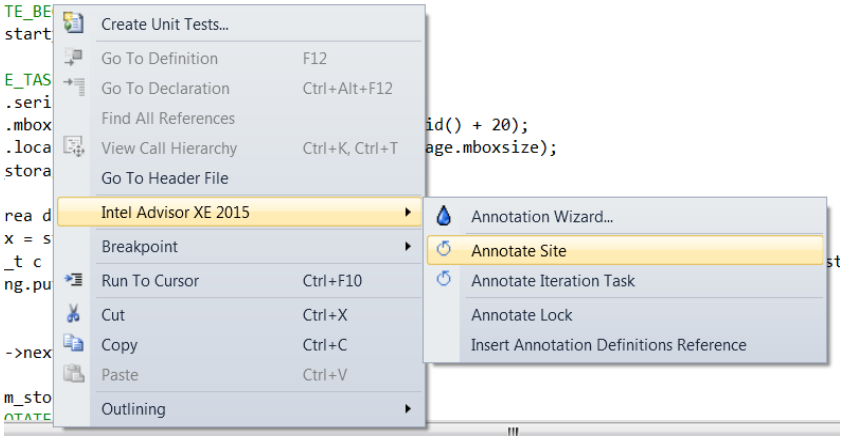



*Figure 14*

The source file had existing annotations commented out. The newly added annotations should be similar to these annotations. Figure 15

```
158   ANNOTATE_SITE_BEGIN( MySite2 );
159   //ANNOTATE_SITE_BEGIN(allRows);
160   for (int y = starty; y < stopy; y++)
161   {
162     ANNOTATE_ITERATION_TASK( MyTask1);
163     //ANNOTATE_ITERATION_TASK(MyTask1);
164     m_storage.serial = 1;
165     m_storage.mboxsize = sizeof(unsigned int)*(max_objectid() + 20);
166     m_storage.local_mbox = (unsigned int *) malloc(m_storage.mboxsize);
167     memset(m_storage.local_mbox,0,m_storage.mboxsize);
168
169     drawing_area drawing(startx, totaly-y, stopx-startx, 1);
170     for (int x = startx; x < stopx; x++) {
171       color_t c = render_one_pixel (x, y, m_storage.local_mbox, m_stora
172       drawing.put_pixel(c);
173     }
174
175     if(!video->next_frame())
176     {
177       free(m_storage.local_mbox);
178       return;
179     }
180     free(m_storage.local_mbox);
181   }
182   //ANNOTATE_SITE_END(allRows);
183   ANNOTATE_SITE_END();
184
185 }
```

*Figure 15*

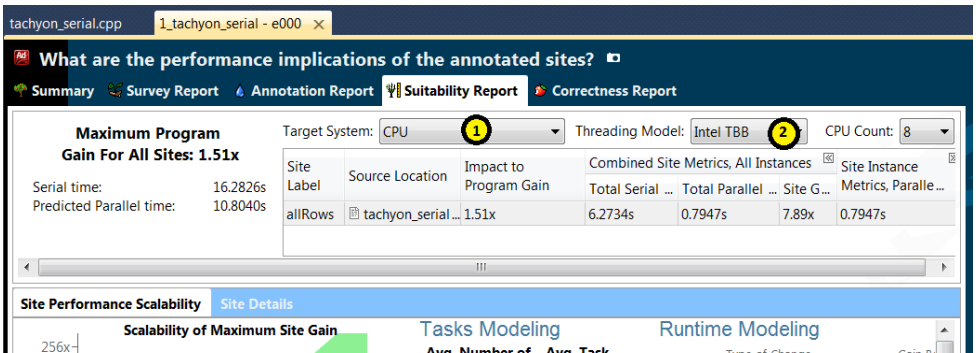
## Build the Project

Save your edits and rebuild the project Release configuration.

### Check Suitability: What Are the Performance Implications?

The Intel Advisor XE Suitability tool helps you evaluate the performance of your parallel experiment by displaying the performance projection for the parallel site and how the site's performance impacts the entire program.

Select the Collect Suitability Data icon under **3. Check Suitability** in the Intel Advisor XE Workflow to analyze the program. Figure 16




1. Model an alternative CPU/Coprocessor.
2. Model a specific threading paradigm.

*Figure 16*

3. Scalability of the selected site
4. Model changes in the number or duration of loop iterations..
5. Model reducing site overhead.

## Check Correctness: What Are the Potential Data-Sharing Problems?

The Intel Advisor XE Correctness tool helps you identify data issues like data races in your parallel experiment. A data race, which is a bug that can occur from adding parallelism to parts of your program, happens when multiple tasks read and write data at a shared memory location without coordinating those read and write operations.

### Run the Correctness Tool

1. Switch to the Debug configuration and set the Command Argument in the **Project Properties > Debugging** menu to ..\dat\3spheres.dat if it is not already set. This decreases the complexity of the input so that the Correctness analysis runs faster.
2. Build the Debug configuration.
3. Select the Collect Correctness Data icon under **4. Check Correctness** in the Intel Advisor XE Workflow to start the Correctness tool. Figure 17

The Correctness tool predicts that our parallel experiment will have data-sharing problems that need to be resolved if we want to parallelize this application. Figure 18

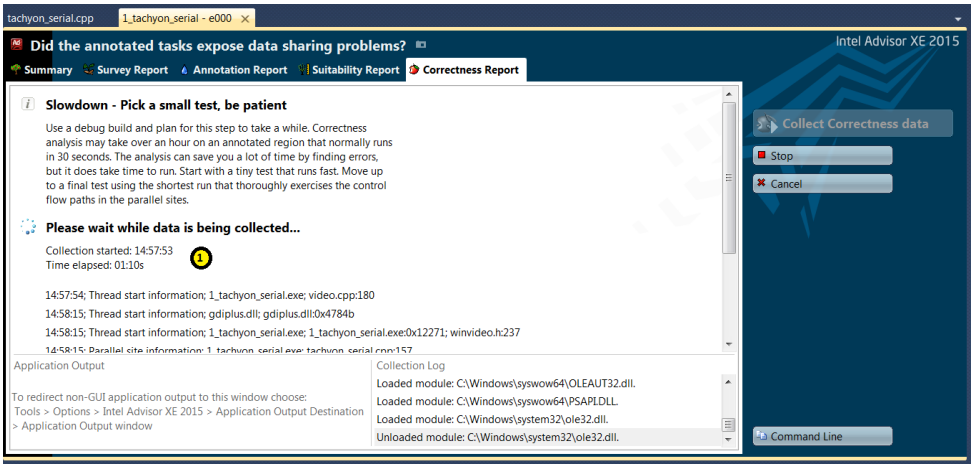
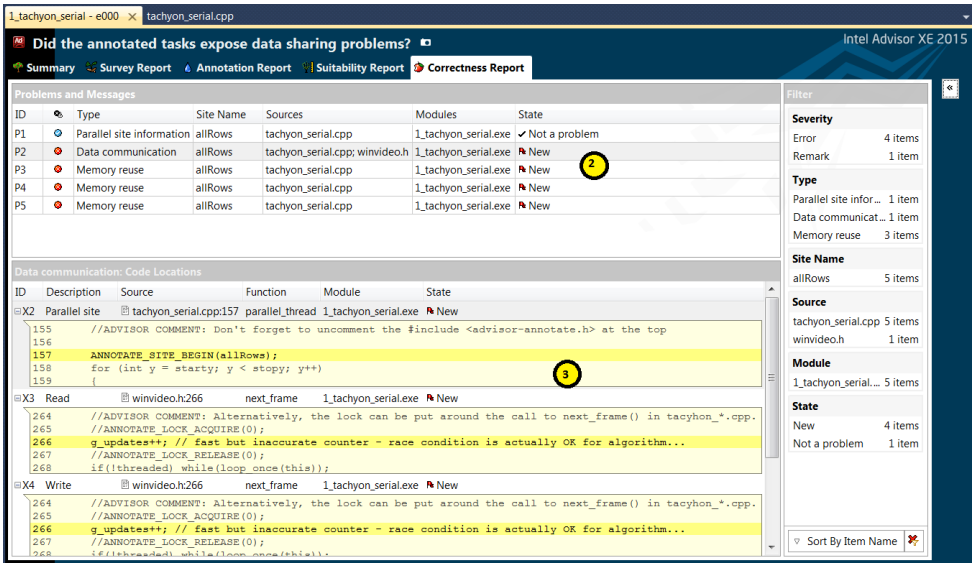



*Figure 17 (above) and 18 (below)*

1. Data collection progress
2. Problems and messages
3. Individual observations corresponding to the problems and messages




## See the Big Picture

When you are experimenting with several parallel models (several sites and tasks), it is a good idea to see the big picture. Use the Summary report to see all the annotations encountered when the start-up project executes, and their corresponding performance projections and data-sharing issues.

Select Summary in the Intel Advisor XE window to display Figure 19.

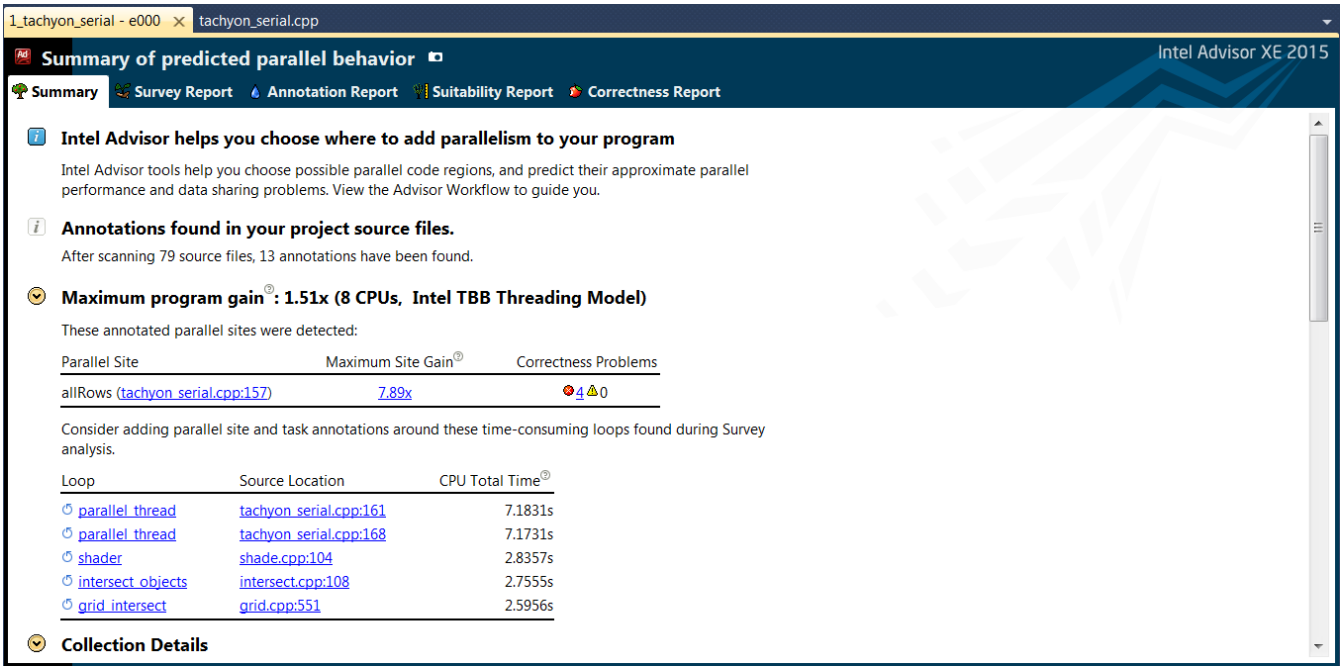



*Figure 19*

## Investigate and Fix the Data-Sharing Problems

Double-click each problem row to see the corresponding Correctness Source view.

Starting with the memory reuse problem reported in the third row (P3), we see what is illustrated in Figure 20.

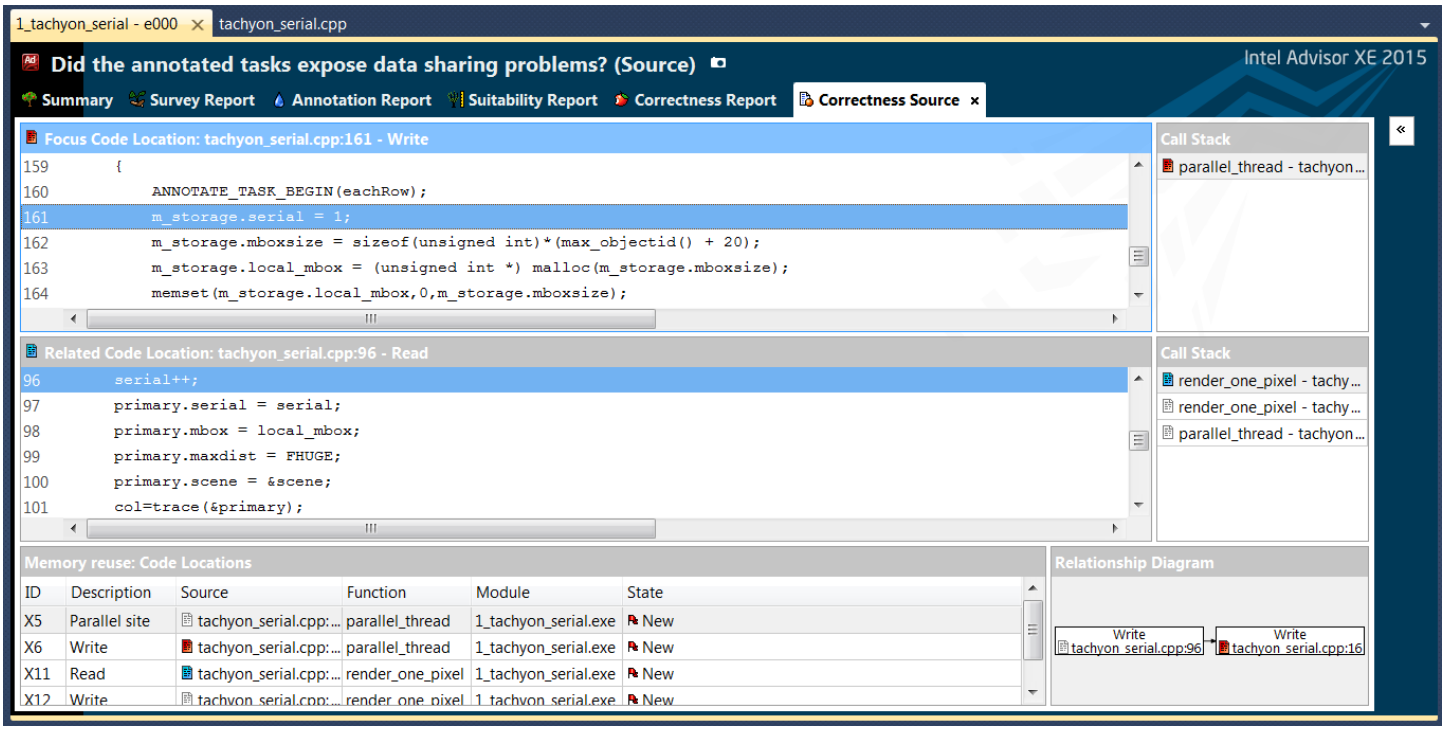



*Figure 20*

After comparing this with the other memory-reuse observations (P4 and P5), we can determine that these three problems are related and describe a case of incidental sharing of m_storage. For the parallel version to work correctly, each task (loop iteration) needs its own copy of m_storage. This is possible if we move the declaration of m_storage to within the task (loop).

Navigate to the editor window (right-click the line in the Observation window and select **Edit Source**) and move the declaration of m_storage Figure 21 to inside the task. Figure 22.

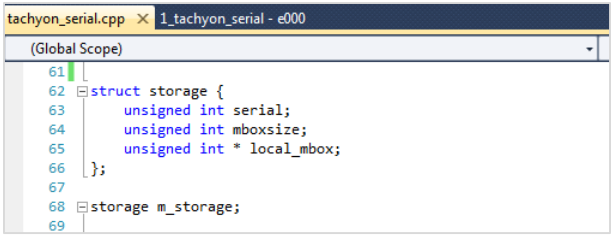



*Figure 21*

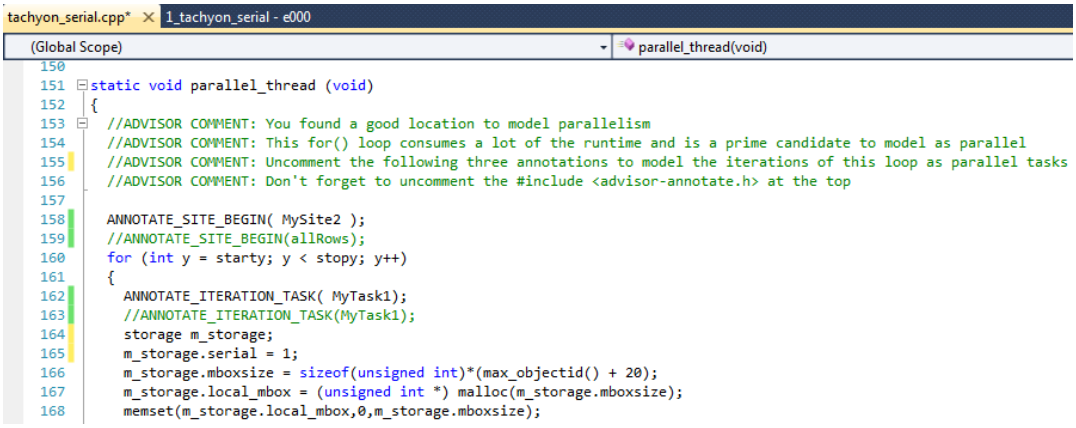



*Figure 22*

The final predicted issue is a data communication observation (P2). Upon investigation, we can see that there is a global variable, g_updates, that must be updated consistently between iterations—this problem cannot be resolved by defining and incrementing local copies. Figure 23

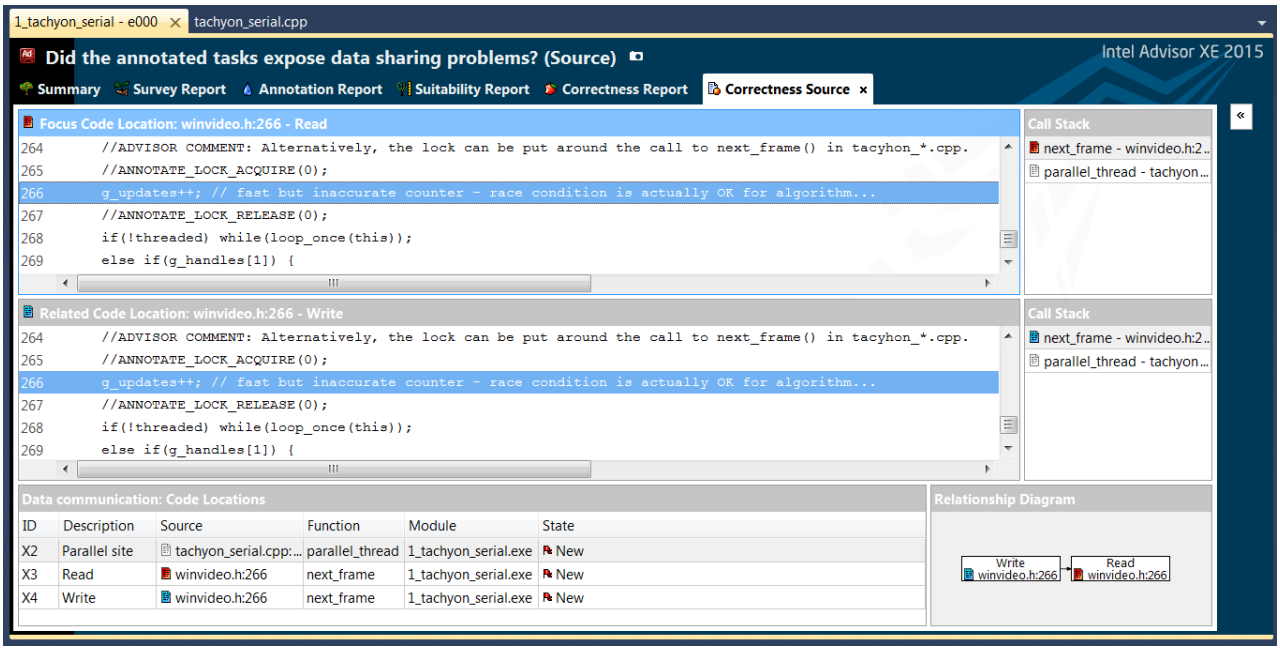



*Figure 23*

The simple fix is to insert a lock annotation pair around the update of g_updates.

1. Navigate to the editable source window by double-clicking the g_updates++ line.
2. Scroll near the top of the winvideo.h file.
3. Uncomment the line to include the Intel Advisor XE annotation definition header file.
4. Select the g_updates++ source line, right-click, and select **Intel Advisor XE > Annotate Lock**.
5. Replace the <address> argument for the lock annotations with 0. Figure 24 and 25

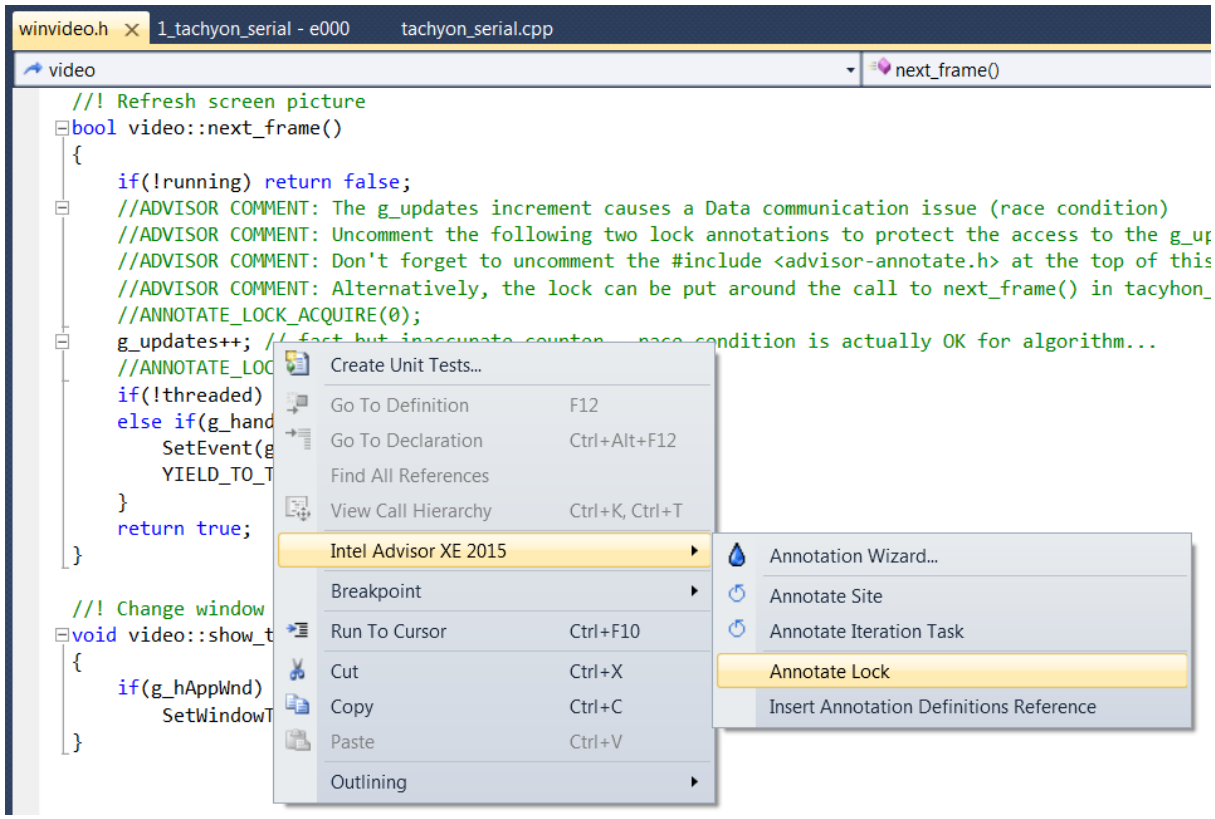



*Figure 24*

```
256
257  //! Refresh screen picture
258 bool video::next_frame()
259 {
260     if(!running) return false;
261   //ADVISOR COMMENT: The g_updates increment causes a Data communication issue (race condition)
262     //ADVISOR COMMENT: Uncomment the following two lock annotations to protect the access to the
263   //ADVISOR COMMENT: Don't forget to uncomment the #include <advisor-annotate.h> at the top of th
264   //ADVISOR COMMENT: Alternatively, the lock can be put around the call to next_frame() in tacyhc
265   //ANNOTATE_LOCK_ACQUIRE(0);
266   ANNOTATE_LOCK_ACQUIRE(0);
267   g_updates++; // fast but inaccurate counter - race condition is actually OK for algorithm...
268   ANNOTATE_LOCK_RELEASE(0);
269   //ANNOTATE_LOCK_RELEASE(0);
270     if(!threaded) while(loop_once(this));
271     else if(g_handles[1]) {
272         SetEvent(g_handles[1]);
273         YIELD_TO_THREAD();
274     }
275     return true;
276 }
```

*Figure 25*

### Check the Effects of Modifying the Sources

With the Correctness data observations resolved, we need to rebuild and rerun both the Suitability and Correctness tools to evaluate the impact of the source changes, and to verify that we have not introduced new issues. The result:

- The Correctness tool does not identify any additional issues.
- The Suitability tool reassures us that the lock of g_updates should have minimal impact. Figure 26

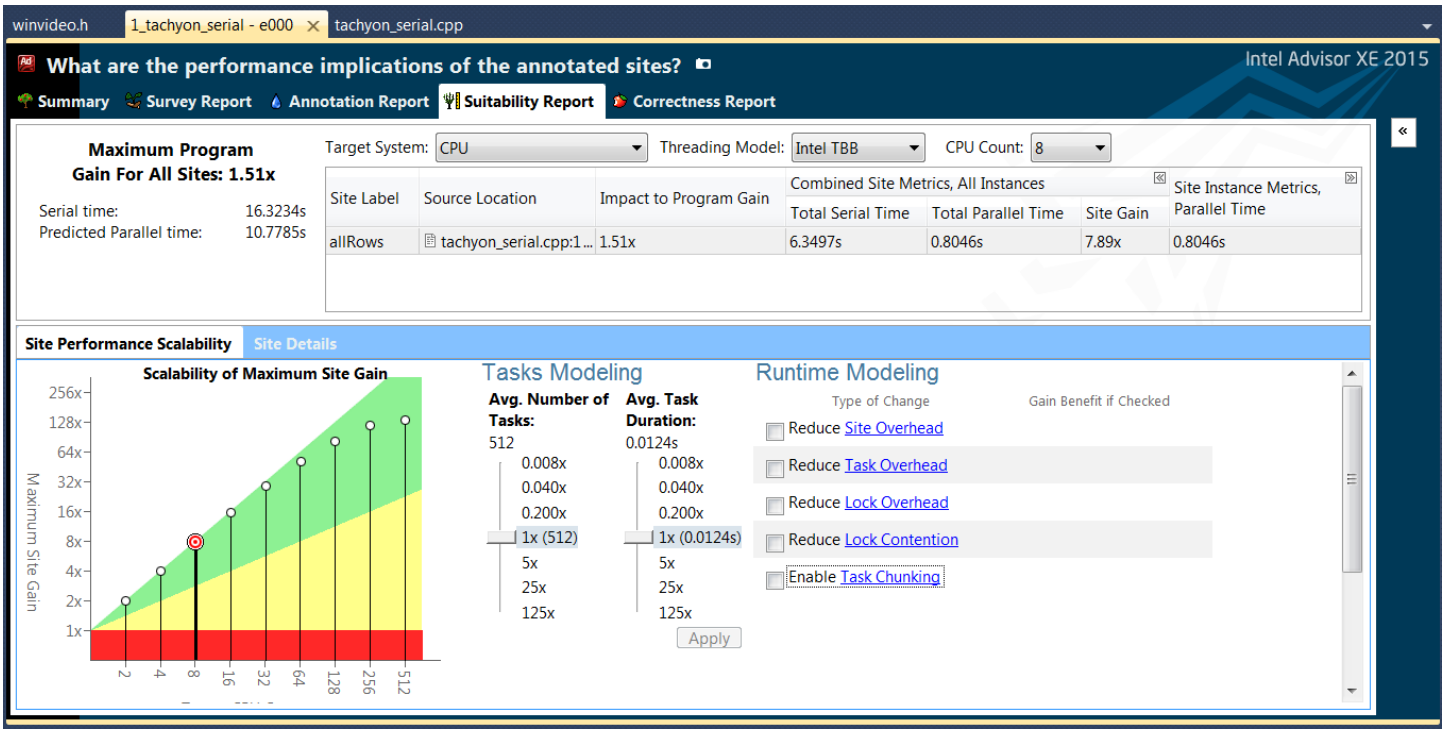



*Figure 26*

## Replace Annotations With a Parallel Framework

The final step is to transition the Intel Advisor XE annotations to a parallel framework. This is Intel Advisor XE Workflow step **5. Add Parallel Framework**. Expand the **Steps to replace annotations** section to display the information in Figure 27.

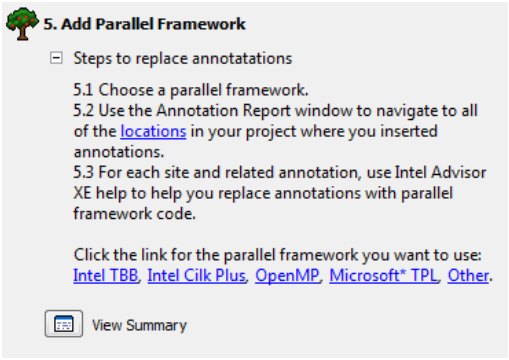



*Figure 27*

Intel Advisor XE provides documentation to assist with the transition of annotations to the parallel framework of your choice. Use the Summary view to quickly navigate to each annotated site by clicking the link in the **Potential program gain** section.

## Build the Parallel Version and Verify Speedup

We can build and run the resulting executable on a multi-core system after replacing the Intel Advisor XE annotations with the appropriate parallel framework, such as Intel® Threading Building Blocks (Intel® TBB). To quickly see the results of parallelizing tachyon using either Intel® TBB, Intel® Cilk Plus, or OpenMP* you can use the 3_tachyon_cilk, 3_tachyon_tbb, or 3_tachyon_omp project.

Figure 28 displays running tachyon on a four-core system: The rendering shows four threads rendering separate parts of the image in parallel.

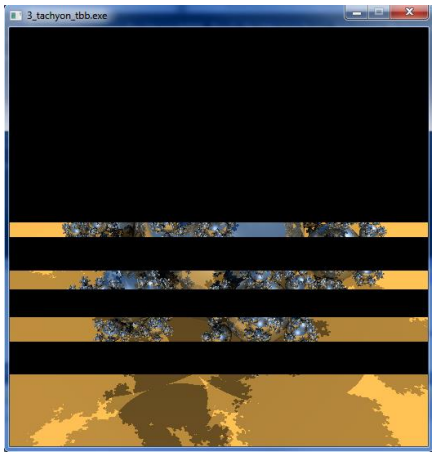



*Figure 28*

## Success

This example demonstrates how you can use the Intel Advisor XE to add parallelism to your existing application. Intel Advisor XE helps you evaluate the performance benefits of parallelism, while identifying potential data races and other problems in the serial code where it is easy to make changes.

Learn more about parallelism and Intel Parallel Studio XE from our experts. We are here to help developers write correct, high-performing code that will take advantage of both today's and tomorrow's processing power.

## Additional Resources

Intel Learning Lab – Technical videos, whitepapers, webinar replays and more.

Intel Parallel Studio XE product page – How to videos, getting started guides, documentation, product details, support and more.

Evaluation Guide Portal – Additional evaluation guides that show how to use various powerful capabilities.

Intel® Software Network Forums – A community for developers.

Intel® Software Products Knowledge Base – Access to information about products and licensing,

Download a free 30 day evaluation

## Purchase Options: Language Specific Suites

Intel® Parallel Studio XE comes in three editions based on your development needs. Single language (C++ or Fortran) versions are available in the Composer and Professional editions.

- **Composer Edition** includes compilers, performance libraries, and parallel models made to build fast parallel code.
- **Professional Edition** includes everything in the Composer edition. It adds performance profiler, threading design/prototyping, and memory & thread debugger to design, build, debug and tune fast parallel code.
- **Cluster Edition** includes everything in the Professional edition. It adds a MPI cluster communications library, along with MPI error checking and tuning to design, build, debug and tune fast parallel code that includes MPI.

| | Intel® Parallel Studio XE Composer Edition[1] | Intel® Parallel Studio XE Professional Edition[1] | Intel® Parallel Studio XE Cluster Edition |
|---|---|---|---|
| Intel® C++ Compiler | √ | √ | √ |
| Intel® Fortran Compiler | √ | √ | √ |
| Intel® Threading Building Blocks (C++ only) | √ | √ | √ |
| Intel® Integrated Performance Primitives (C++ only) | √ | √ | √ |
| Intel® Math Kernel Library | √ | √ | √ |
| Intel® Cilk™ Plus (C++ only) | √ | √ | √ |
| Intel® OpenMP* | √ | √ | √ |
| Rogue Wave IMSL* Library[2] (Fortran only) | Bundled and Add-on | Add-on | Add-on |
| Intel® Advisor XE | | √ | √ |
| Intel® Inspector XE | | √ | √ |
| Intel® VTune™ Amplifier XE[3] | | √ | √ |
| Intel® MPI Library[3] | | | √ |
| Intel® Trace Analyzer and Collector | | | √ |
| Operating System (Development Environment) | Windows* (Visual Studio*)<br>Linux* (GNU)<br>OS X*[4] (XCode*) | Windows (Visual Studio)<br>Linux (GNU) | Windows (Visual Studio)<br>Linux (GNU) |

Notes:
1. Available with a single language (C++ or Fortran) or both languages.
2. Available as an add-on to any Windows Fortran* suite or bundled with a version of the Composer Edition.
3. Available bundled in a suite or standalone
4. Available as single language suites on OS X.

Learn more about Intel Parallel Studio XE

- Click or enter the link below: http://intel.ly/parallel-studio-xe
- Or scan the QR code on the left

Download a free 30-day evaluation

- Click or enter the link below: http://intel.ly/sw-tools-eval
- Click on 'Product Suites' link

## Notices



### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804



design-parallel-performance_studioxe-evalguide/Rev-081714