(intel)

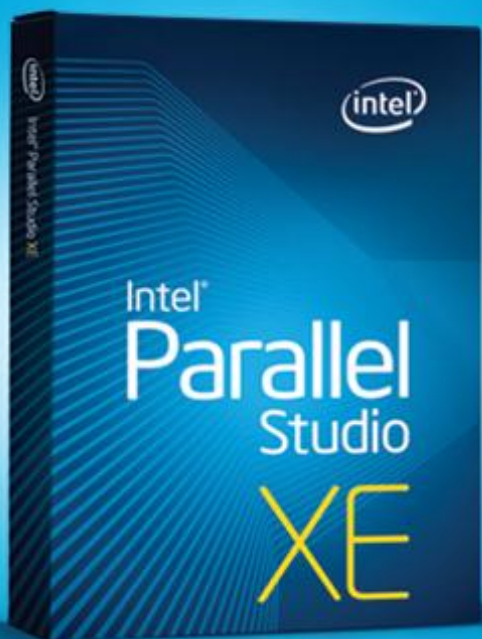# Eliminate Threading Errors and Improve Program Stability

## with Intel® Parallel Studio XE

# Eliminate Threading Errors and Improve Program Stability

(intel)

# Can running one simple tool make a difference?

Yes, in many cases. You can find errors that cause complex, intermittent bugs and improve yourcustomers confidence in the stability of your application.

This guide describes how to use the Intel® Inspector XE analysis tool to find threading errors before they happen. The following information walks you through the steps using a sample application.

# Three Easy Steps to Better Performance

### Step 1. Install and Set Up Intel® Parallel Studio XE

Estimated completion time: 15-30 minutes

1. [Download](#) an evaluation copy of Intel Parallel Studio XE.

2. Install Intel Parallel Studio XE by clicking on the **parallel_studio_xe_2011_setup.exe** (can take 15 to 30 minutes depending on your system).

### Step 2. Install and View the Adding_Parallelism Sample Application

Install the sample application:

1. Download the [Tachyon+Sample.zip](#) sample file to your local machine. This is a C++ console application created with Microsoft* Visual Studio* 2005.

2. Extract the files from the Tachyon_conf.zip file to a writable directory or share on your system, such as **My Documents\Visual Studio 20xx\Intel\samples** folder.

### Step 3. Find Threading Errors Using Intel® Inspector XE

Intel® Inspector XE is a serial and multithreading error-checking analysis tool.  It is available for both Linux* and Windows* including integration with Microsoft* Visual Studio*. It supports applications created with the C/C++, C#, .NET, and Fortran languages. Intel Inspector XE detects challenging memory leaks and corruption errors as well as threading data races and deadlock errors. This easy, comprehensive developer-productivity tool pinpoints errors and provides guidance to help ensure application reliability and quality.

NOTE: Samples are non-deterministic. Your screens may vary from the screen shots shown throughout these tutorials.

# Eliminate Threading Errors and Improve Program Stability

(intel)

## Identify, Analyze, and Resolve Threading Errors

You can use Intel Inspector XE to identify, analyze, and resolve threading errors in parallel programs by performing a series of steps in a workflow. This tutorial guides you through these workflow steps while using a sample program named tachyon_conf

### Choose a Target

1. Open the sample in Microsoft Visual Studio. Go to **File > Open > Project/Solution** and open the **tachyon_conf\vc8\ tachyon_conf.sln** solution file:

Figure 1
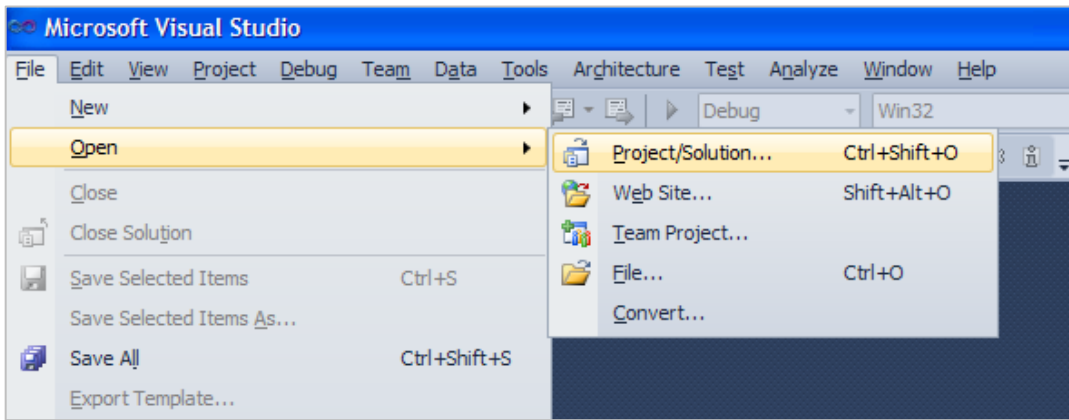


This will display the tachyon_ conf solution in the Solution Explorer pane. Figure 1

2. In the Solution Explorer pane, right-click the **find_and_fix_ threading_errors** project and select Set as Startup Project.

3. Build the application using **Build > Build Solution**. Figure 2

4. Run the application using **Debug > Start Without Debugging**. Figure 3

Figure 2



Figure 3

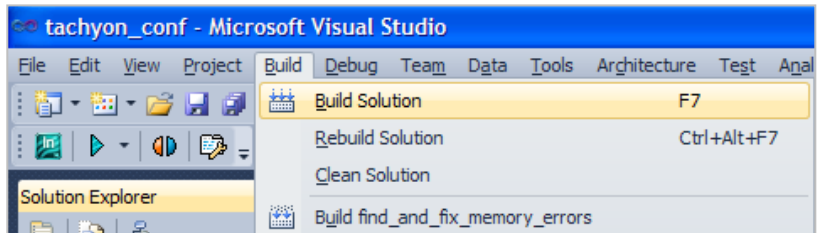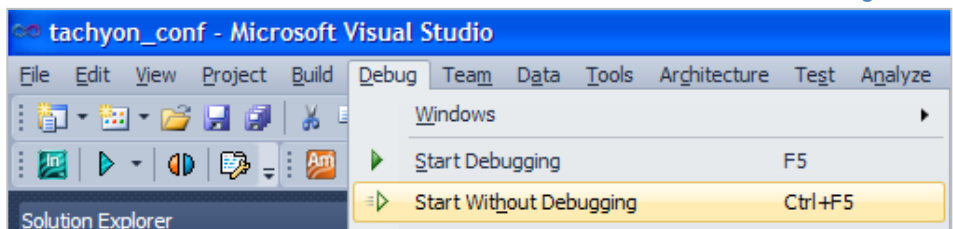# Eliminate Threading Errors and Improve Program Stability

*(intel)*

## Build the Target

Verify the Microsoft Visual Studio project is set to produce the most accurate, complete results. Then, build it to create an executable that Intel Inspector XE can check for threading errors.

You can use Intel Inspector XE on both debug and release modes of binaries containing native code; however, targets compiled/linked in debug mode using the following options produce the most accurate, complete results. Figure 4

Figure 4

| Compiler/Linker Options | Correct Setting | Impact If Not Set Correctly |
|---|---|---|
| Debug information | Enabled (/Zi or /ZI) | Missing file/line information |
| Optimization | Disabled (/Od) | Incorrect file/line information |
| Dynamic runtime library | Selected (/MD or /MDd) | False positives or missing observations |

## Build the Target

Figure 5

To verify that debug mode is configured:

1. In the Solution Explorer pane, right-click the **find_and_fix_threading_errors** project and select **Properties**.

2. Check that the Configuration drop-down list is set to **Debug, or Active(Debug).** Figure 5

3. In the left pane, choose **Configuration Properties > C/C++ > General**. Verify the Debug Information Format is set to **Program Database (/Zi) or Program Database for Edit & Continue (/ZI).** Figure 6
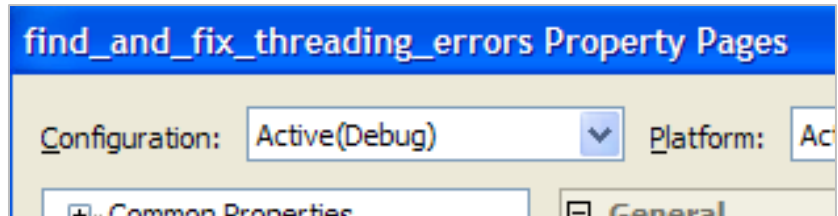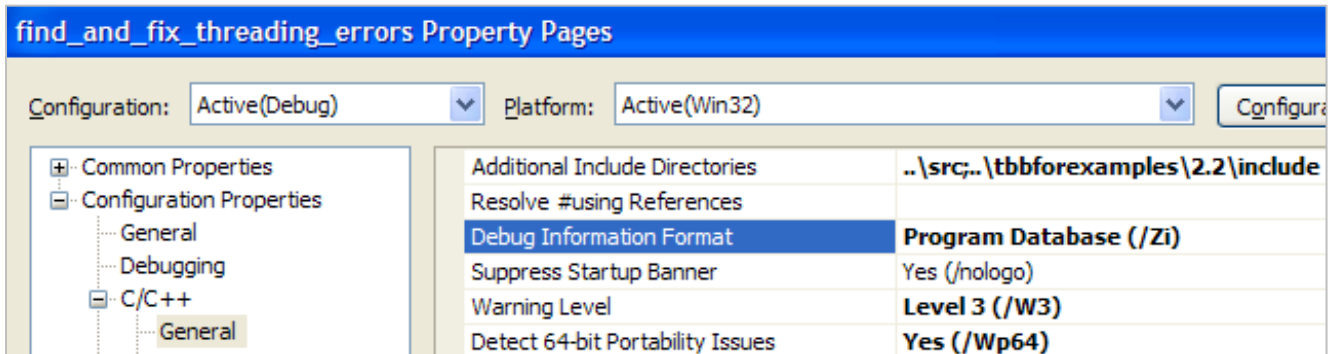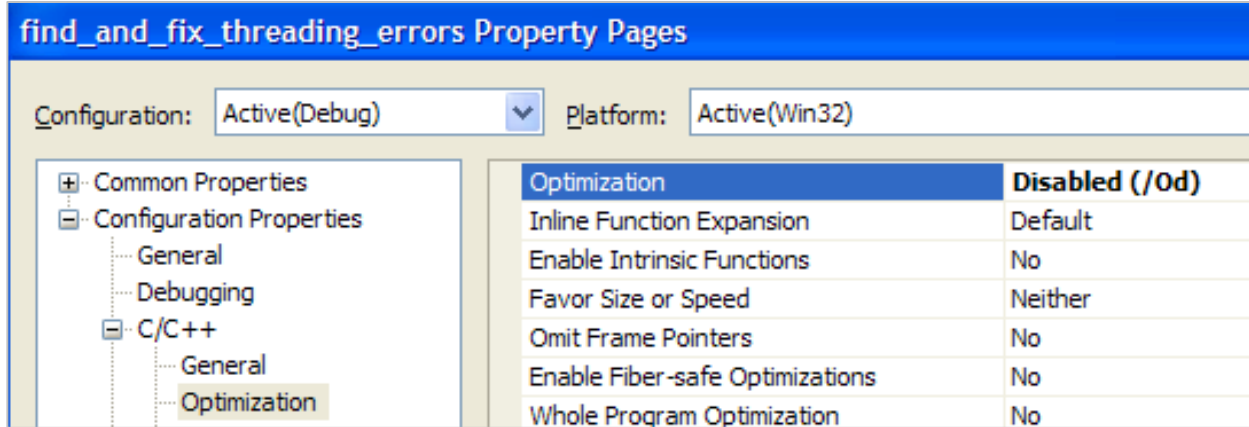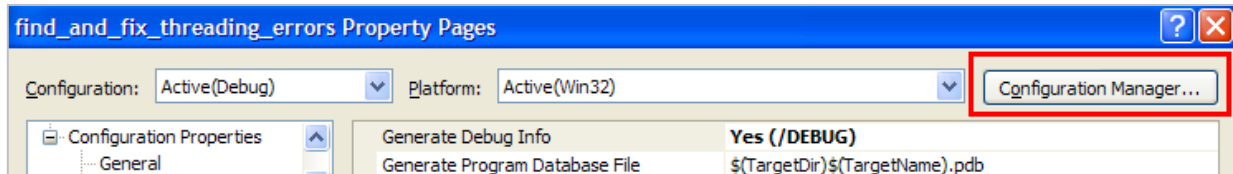


Figure 6

4. Choose **Configuration Properties > C/C++ > Optimization**. Verify the Optimization field is set to **Disabled (/Od)**. Figure 7

Figure 7



5. Choose **Configuration Properties > C/C++ > Code Generation**. Verify the Runtime Library field is set to **Multi-threaded DLL (/MD) or Multi-threaded Debug DLL (/MDd)**.
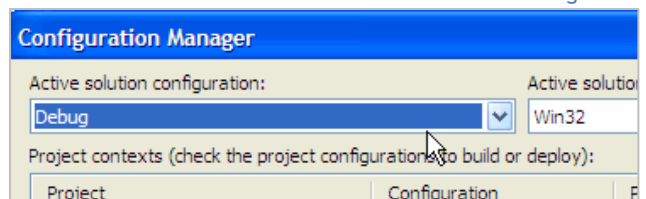
Figure 8



6. Choose **Configuration Properties > Linker > Debugging**. Verify the Generate Debug Info field is set to **Yes (/Debug)**.

*To verify the target is set to build in debug mode:*

1. In the Properties dialog box, click the **Configuration Manager button**. Figure 8

2. Verify the Active solution configuration drop-down list is set to **Debug**. Figure 9

3. Click the **Close** button to close the Configuration Manager dialog box.

4. Click the **OK** button to close the Property Pages dialog box.

Figure 9



5

*Build the Target*

1. Choose **Debug > Start Without Debugging**. When the application starts, you should see a display similar to this:

   Notice the discolored dots in the image. Figure 10

   If this application had no errors, the output would look like Figure 11.

Figure 10



Figure 11

# Eliminate Threading Errors and Improve Program Stability

*Configure Analysis*

Choose a preset configuration to influence threading error analysis scope and running time.

To configure a threading error analysis:

1. From the Microsoft Visual Studio menu, choose **Tools > Intel Inspector XE 2011 > New Analysis**... to display an Analysis Type window.

Choose the **Detect Deadlocks and Dataraces** analysis type to display a window similar to the following. Figure 12

Figure 12
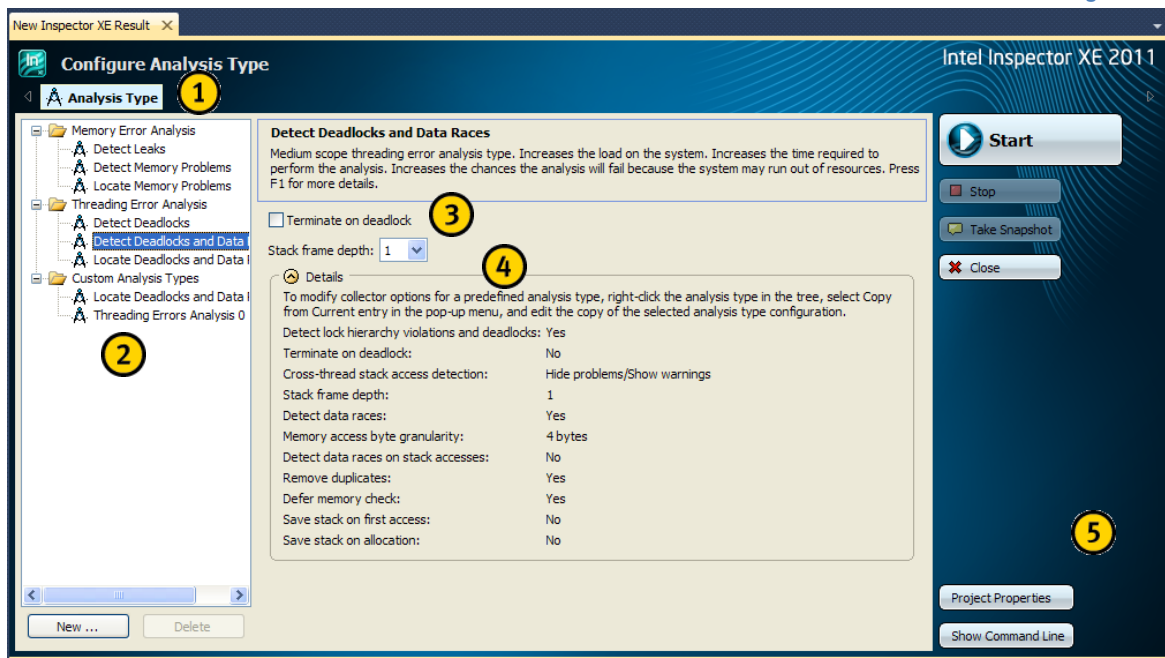


① Use the Navigation toolbar to navigate among the Intel Inspector XE windows. The buttons on the toolbar vary depending on the displayed window.

② This tutorial covers threading error analysis types, which you can use to search for these kinds of errors: data race, deadlock, lock hierarchy violation, and cross-stack thread access.

Use memory error analysis types to search for these kinds of errors: GDI resource leak, kernel resource leak, incorrect memcpy call, invalid deallocation, invalid memory access, invalid partial memory access, memory leak, mismatched allocation/deallocation, missing allocation, uninitialized memory access, and uninitialized partial memory access.

You can also use the New button to create custom analysis types from existing analysis types.

③ Use the checkbox(es) and drop-down list(s) to fine-tune some, but not all, analysis type settings. If you need to fine-tune more analysis type settings, choose another preset analysis type or create a custom analysis type.

④ The Details region shows all current analysis type settings. Try choosing a different preset analysis type or checkbox/drop-down list value to see the impact on the Details region.

⑤ Use the **Command** toolbar to control analysis runs and perform other functions. For example, use the **Project Properties** button to display the **Project Properties** dialog box, where you can change the default result directory location, set parameters to potentially speed up analysis, and perform other project configuration functions.

## Run the Analysis

Run a threading error analysis to detect threading issues that may need handling.

To run a threading error analysis:

Click the Start button to:

> Execute the **find_and_fix_threading_errors.exe** target.

> Identify threading issues that may need handling.

> Collect the result in a directory in the **tachyon_conf/vc8/My Inspector Results XE – find_and_fix_threading_errors** directory.

> Finalize the result (convert symbol information into file names and line numbers, perform duplicate elimination, and form problem sets).

During collection, Intel Inspector XE displays a Collection Log window similar to the following. Figure 13

Figure 13
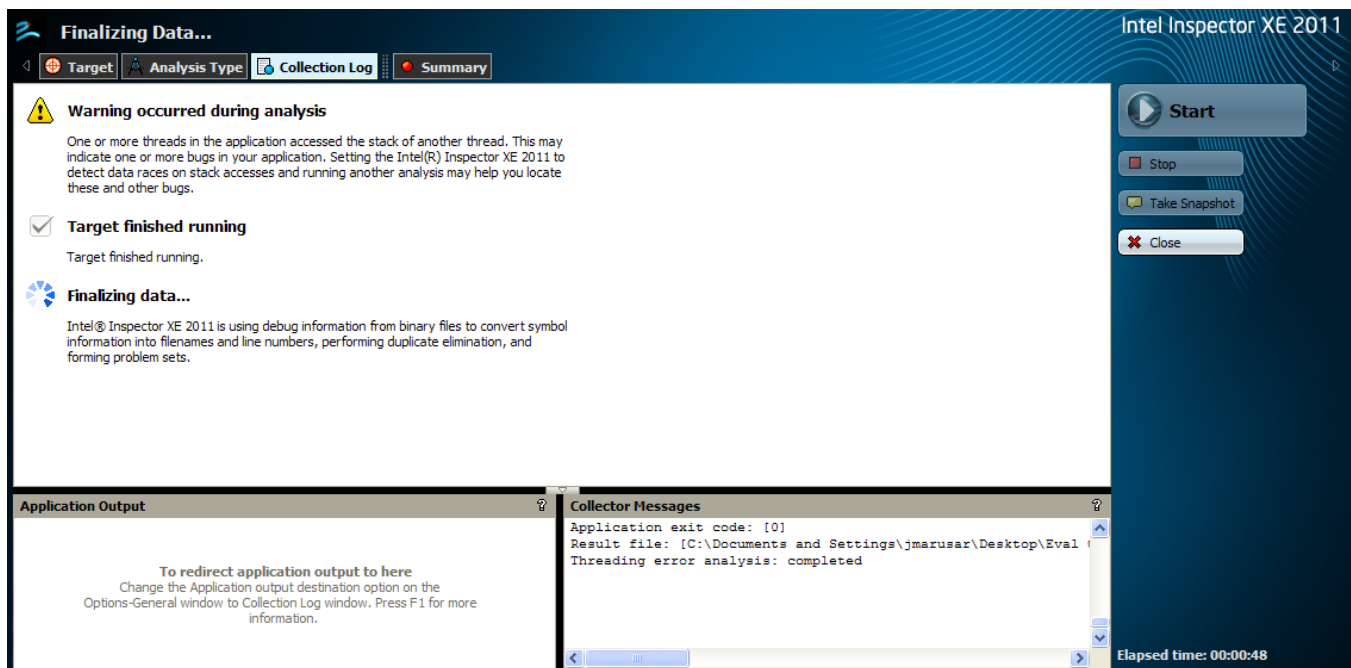
### Choose a Problem Set

Choose a problem set on the Summary window to explore a detected threading issue. Figure 14

To choose a problem set:

1. Click the data row for the Data Race problem in the **find_and_fix_threading_errors.cpp** source file.

2. Double-click the data row for the X1 Write code location to display the Sources window, which provides more visibility into the cause of the error.

3. You started exploring a Data Race problem set in the Sources window that contains one or more problems composed of Read and Write code locations in the find_and_fix_threading_errors.cpp source file. Figure 15
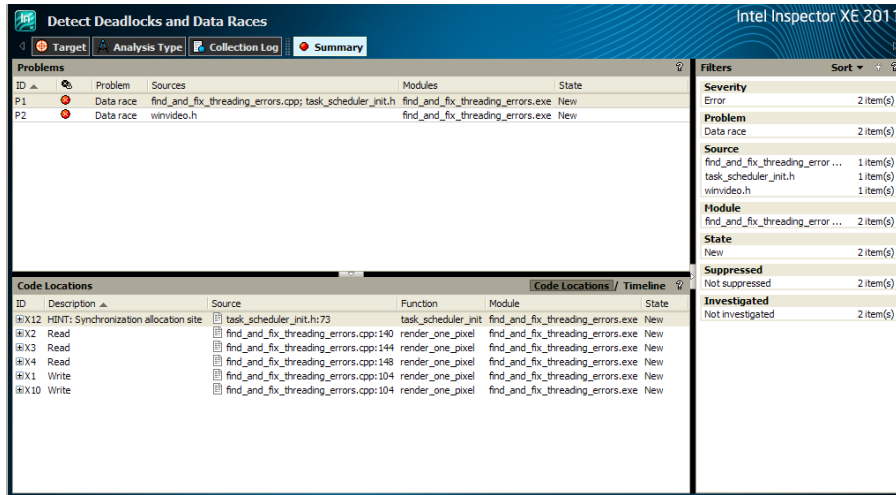
Figure 14



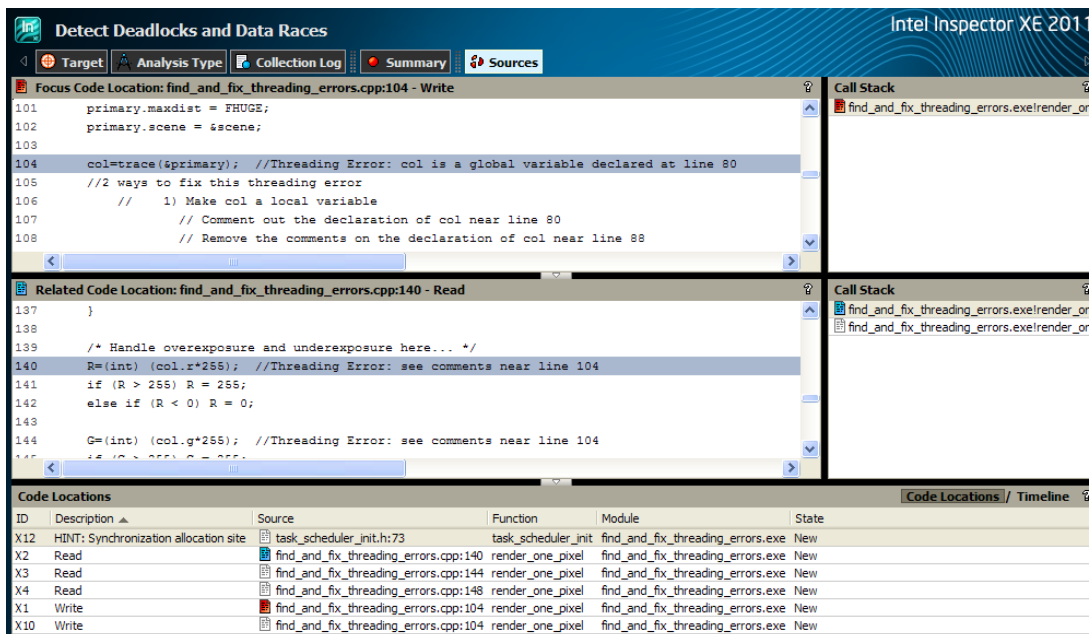Figure 15

### Interpret the Result Data

Interpret data on the Sources window to determine the cause of the detected threading issue. Figure 16

**①** Like the pane on the Summary window, the Code Locations pane shows all the code locations in one Write -> Write Data race problem and three Write -> Read Data race problems in the Data race problem set.

The Write -> Write Data race problem contains two code locations:

- The X1 Write code location represents the instruction and associated call stack of the thread responsible for a memory write.
- The X12 Write code location represents the instruction and associated call stack of the thread responsible for a concurrent memory write.
- Each Write -> Read Data race problem also contains two code locations:
- The X1 Write code location represents the instruction and associated call stack of the thread responsible for a memory write.
- The X2, X3, and X4 Read code locations represent the instructions and associated call stacks of the threads responsible for a concurrent memory read.

Notice the X1 Write code location is in all problems.

**②** The Related Code Location pane shows the source code in the find_and_fix_threading_errors.cpp source file surrounding the X2 Read code location. (Notice the icon in the pane title matches the icon on the X2 Read code location data row in the Code Locations pane.) The source code corresponding to the Read code location is highlighted.

**③** The Focus Code Location pane shows the source code in the find_and_fix_threading_errors.cpp source file surrounding the X1 Write code location. (Notice the icon in the pane title matches the icon on the X1 Write code location data row in the Code Locations pane.) The source code corresponding to the Write code location is highlighted.

**④** The Timeline pane is a graphic visualization of relationships among dynamic events in a problem

## To interpret result data:

You must first understand the following:

- A **Write -> Write Data race** problem occurs when multiple threads write to the same memory location without proper synchronization.

- A **Write -> Read Data race** problem occurs when one thread writes to, while a different thread concurrently reads from, the same memory location.

The commenting in the Focus Code Location window identifies the cause of the Data race problems: Multiple threads are concurrently accessing the global variable col. One possible correction strategy: Change the global variable to a local variable.

To access more information on interpreting and resolving problems:

1. Right-click any code location in the Code Locations pane.

2. Choose Explain Problem to display the Intel Inspector XE Help information for the Data race problem type.

You determined the cause of a Data race problem set in the find_and_fix_threading_errors.cpp source file: Multiple threads are concurrently accessing the global variable col.

## Resolve the Issue

Access the Microsoft Visual Studio editor to fix the threading issue.

**To resolve the issue:**

1. Scroll to this source code near line 80 in the Focus Code Location pane: color col;:

2. Double-click the line to open the find_and_fix_threading_errors.cpp source file in a separate tab where you can edit it with the Visual Studio* editor.

3. Comment color col; and uncomment //color col; near line 89 to localize the variable col to the function render_one_pixel , Figure 17

Figure 17



```
find_and_fix_threading_errors.cpp  ×  r000ti2     New Inspector XE Result

(Global Scope)

65   // shared but read-only so could be private too
66   static thr_parms *all_parms;
67   static scenedef scene;
68   static int startx;
69   static int stopx;
70   static int starty;
71   static int stopy;
72   static flt jitterscale;
73   static int totaly;
74
75   #include "tbb/task_scheduler_init.h"
76   #include "tbb/parallel_for.h"
77   #include "tbb/mutex.h"
78   #include "tbb/blocked_range.h"
79
80   //color col;
81
82   static color_t render_one_pixel (int x, int y, unsigned int *local_mbox, volatile unsigned int &serial,
83                                    int startx, int stopx, int starty, int stopy)
84   {
85       /* private vars moved inside loop */
86       ray primary, sample;
87       color avcol;
88     color col;
89       int R,G,B;
90       intersectstruct local_intersections;
91       int alias;
92       /* end private */
93
94       primary=camray(&scene, x, y);
95       primary.intstruct = &local_intersections;
96       primary.flags = RT_RAY_REGULAR;
97
98       serial++;
```

11

### Rebuild and Rerun the Analysis

Rebuild the target with your edited source code, and then run another threading error analysis to see if your edits resolved the threading error issue.

**To rebuild the target:**

In the Solution Explorer pane, right-click the **find_and_fix_threading_errors** project and choose **Build** from the pop-up menu.

**To rerun the same analysis type configuration as the last-run analysis:**

Choose **Tools > Intel Inspector XE 2011 > New Analysis**... and follow the steps above to execute the find_and_fix_threading_errors.exe target and display the following: Figure 18

Figure 18
Notice that the image now displays correctly



### Success

In this example, we had a bug and the program was not behaving correctly. After running Intel Inspector XE, we found the bug, and now the graphics render correctly. Often, you will be able to obtain the same results on your own application by running Intel Inspector XE right out of the box.

However, as you have seen, the time dilation can be significant; this is just the nature of the technology. In the next section, you will find tips for running large applications on Intel Inspector XE.

Intel Inspector XE also has a command-line interface that you can use to automate the testing of your application on multiple workloads and test cases by running it overnight in batch mode or as part of a regression test suite.

# Tips for Larger/Complex Applications

### Key Concept: Choosing Small, Representative Data Sets

When you run an analysis, Intel Inspector XE executes the target against a data set. Data set size has a direct impact on target execution time and analysis speed.

For example, it takes longer to process a 1000x1000 pixel image than a 100x100 pixel image. One possible reason could be that you have loops with an iteration space of 1…1000 for the larger image, but only 1…100 for the smaller image. The exact same code paths may be executed in both cases. The difference is the number of times these code paths are repeated.

You can control analysis cost, without sacrificing completeness, by removing this kind of redundancy from your target. Instead of choosing large, repetitive data sets, choose small, representative data sets. Data sets with runs in the time range of seconds are ideal. You can always create additional data sets to ensure all your code is inspected.

### Managing Memory Errors

Intel Inspector XE can also identify, analyze, and resolve memory errors, such as memory leaks and corruption errors in serial and parallel programs. These errors can manifest intermittently and may decrease application performance and reliability.
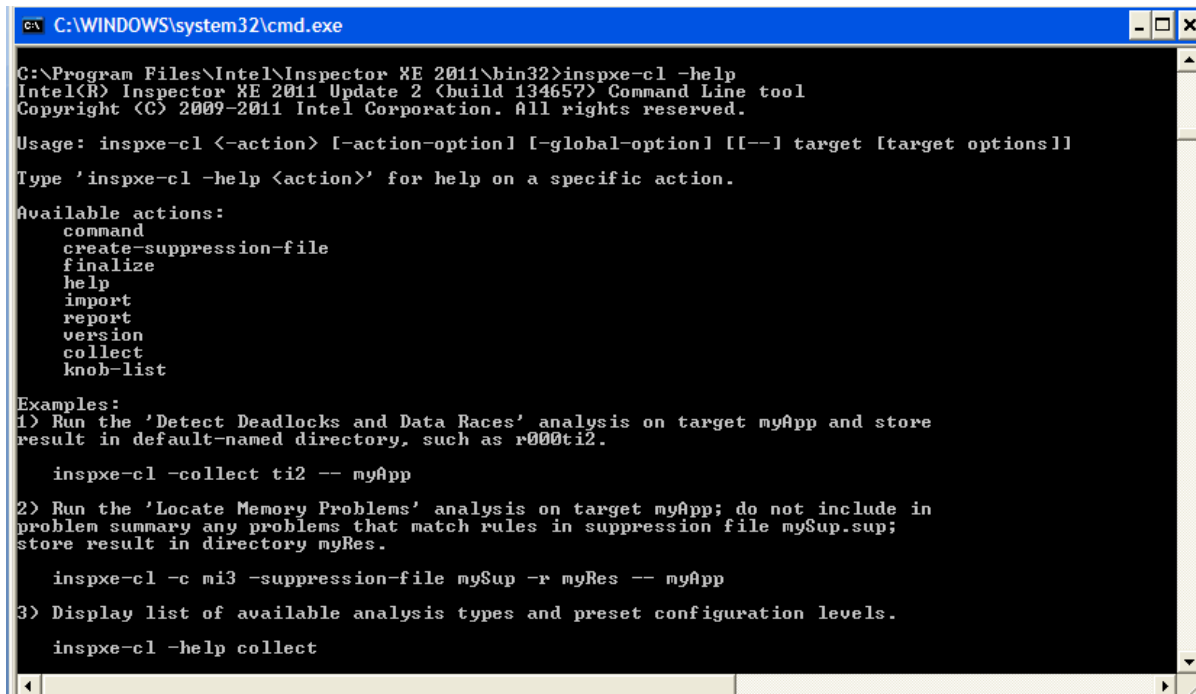
### Using the Command-line to Automate Testing

As you can see, Intel Inspector XE has to execute your code path to find errors in it. Thus, run Intel Inspector XE on multiple versions of your code, on different workloads that stress different code paths, as well as on corner cases. Furthermore, given the inherent time dilation that comes with code-inspection tools, it would be more efficient to run these tests overnight or as part of your regression testing suite and have the computer do the work for you; you just examine the results of multiple tests in the morning.

The Intel Inspector XE command-line version is called inspxe-cl, and is available by opening a command window (Start > Run, type in "cmd" and press OK) and typing in the path leading to where you installed Intel Inspector XE.  Figure 19

To get help on inspxe-cl, use the –help command line option.

```
> c:\Program Files\Intel\Inspector XE 2011\bin32\inspxe-cl -help
```

Figure 19



```
C:\Program Files\Intel\Inspector XE 2011\bin32>inspxe-cl -help
Intel(R) Inspector XE 2011 Update 2 (build 134657) Command Line tool
Copyright (C) 2009-2011 Intel Corporation. All rights reserved.

Usage: inspxe-cl <-action> [-action-option] [-global-option] [[--] target [target options]]

Type 'inspxe-cl -help <action>' for help on a specific action.

Available actions:
    command
    create-suppression-file
    finalize
    help
    import
    report
    version
    collect
    knob-list

Examples:
1> Run the 'Detect Deadlocks and Data Races' analysis on target myApp and store
result in default-named directory, such as r000ti2.

    inspxe-cl -collect ti2 -- myApp

2> Run the 'Locate Memory Problems' analysis on target myApp; do not include in
problem summary any problems that match rules in suppression file mySup.sup;
store result in directory myRes.

    inspxe-cl -c mi3 -suppression-file mySup -r myRes -- myApp

3> Display list of available analysis types and preset configuration levels.

    inspxe-cl -help collect
```

# Additional Resources

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

## Optimization Notice

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20110307