Eliminate Memory Errors to Improve Program Stability

Introduction

This guide will illustrate how Intel® Parallel Studio XE memory checking capabilities can find crucial memory defects early in the development cycle. It provides detailed insights into application memory behavior to improve application stability. The powerful memory checker makes it easier to find latent errors on the executed code path. It also finds intermittent and non-deterministic errors, even if the error-causing timing scenario does not happen.

Can running one simple tool make a difference?

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

This guide describes how to use the Intel® Inspector XE analysis tool to minimize code defects, while maximizing code reliability and lowering development costs. The following information walks you through the steps for using a sample application.

Three Easy Steps to Better Performance

Install and Set Up Intel Parallel Studio XE

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

2. Install the Adding Parallelism Sample Application

1. Download the tachyon_insp_xe.zip sample file to your local machine. This is a C++ console application created with Microsoft Visual Studio* 2010.
2. Extract the files from the tachyon_insp_xe.zip file to a writable directory or share on your system, such as My Documents\Visual Studio 20xx\Intel\samples folder.

3. Find Memory Errors with Intel Inspector XE

Intel Inspector XE is a serial and multithreading error-checking analysis tool. It is available for both Linux* and Windows* operating systems and plugs into Visual Studio software. 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 this guide.

Identify, Analyze, and Resolve Memory Errors

You can use the Intel Inspector XE to identify, analyze, and resolve memory errors in serial or parallel programs by performing a series of steps in a workflow. This document guides you through these workflow steps while using a sample program named tachyon_insp_xe.

Choose a Target

1. Open the sample in Visual Studio software. Go to File > Open > Project/Solution and open the tachyon_insp_xe\vc10\ tachyon_insp_xe.sln solution file to display the tachyon_ insp_xe solution in the Solution Explorer. Figure 1

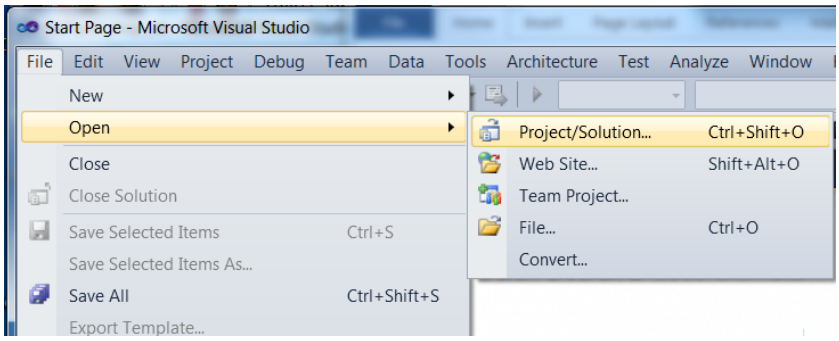

Figure 1

2. In the Solution Explorer, right-click the find_and_fix_ memory_errors project and select Set as Startup Project.
3. Build the application using Build > Build Solution. Figure 2

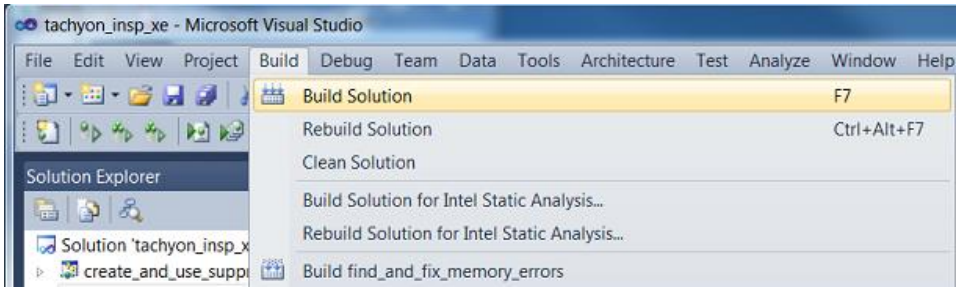

Figure 2

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

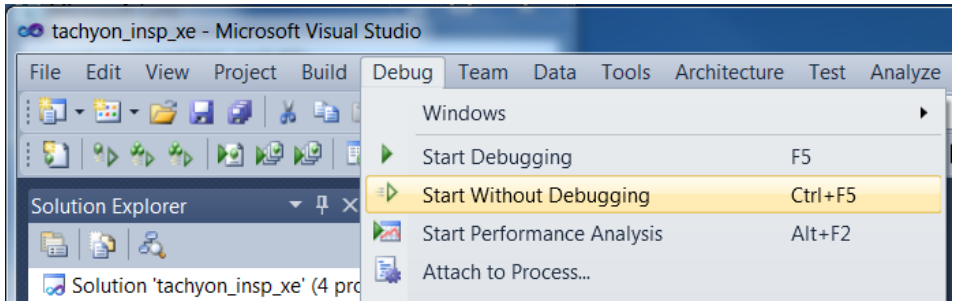

Figure 3

Build the Target

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

You can use the 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 results. Table 1

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 code locations
Basic runtime error checks	Disabled (do not use /RTC; Default option in Visual Studio software)	False positives

To verify settings:

1. In the **Solution Explorer**, right-click the find_and_fix_memory_errors project and select Properties.
2. Check that the Configuration drop-down list is set to Debug, or Active(Debug). Figure 4

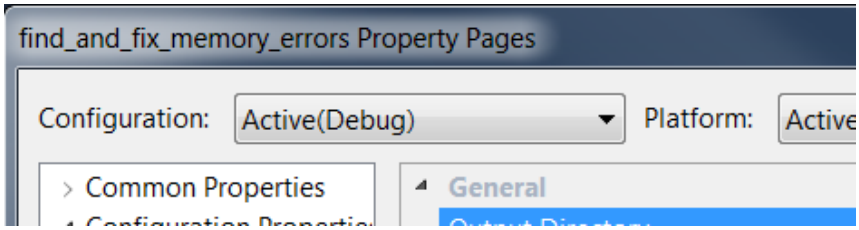

Figure 4

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 5

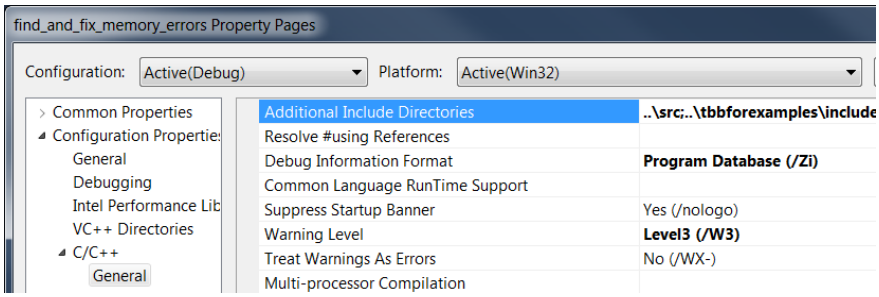

Figure 5

4. Choose Configuration Properties > C/C++ > Optimization. Verify the Optimization field is set to Disabled (/Od). Figure 6

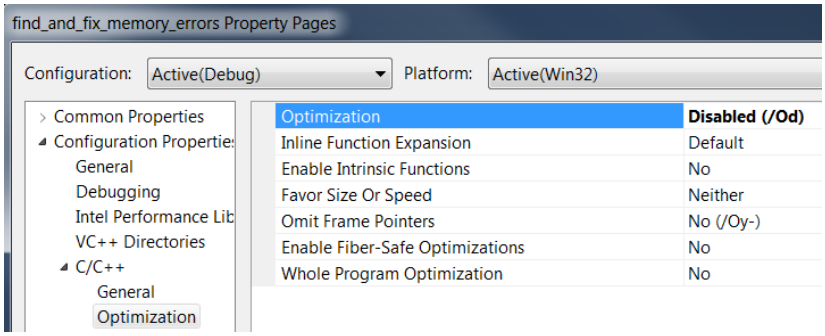

Figure 6

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) and Basic Runtime Checks is set to Default.
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 7

Figure 7

2. Verify the Active solution configuration drop-down list is set to Debug. Figure 8

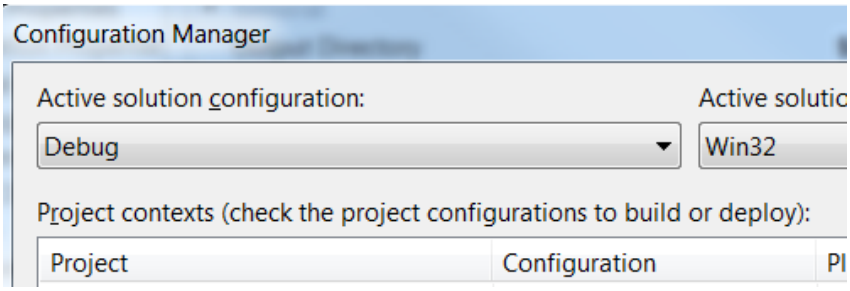

Figure 8

3. Click the Close button to close the Configuration Manager dialog box.
4. Click the OK button to close the Property Pages dialog box.

Run the Target

Choose Debug > Start Without Debugging. When the application starts, you should see a display similar to Figure 9.

As you can see, the image is not rendered fully, correctly, and consistently.

Figure 9

If this application had no errors, the output would look like Figure 10.

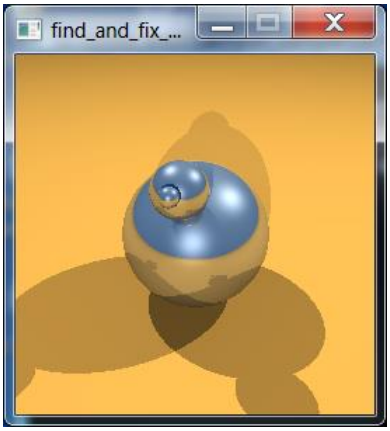

Figure 10

Configure Analysis

Choose a preset configuration to influence memory error analysis scope and running time:

1. From the Visual Studio menu, choose Tools > Intel Inspector XE 2015 > New Analysis... to display an Analysis Type window.
2. Choose Memory Error Analysis from the dropdown menu to display a window similar to the following. Figure 11

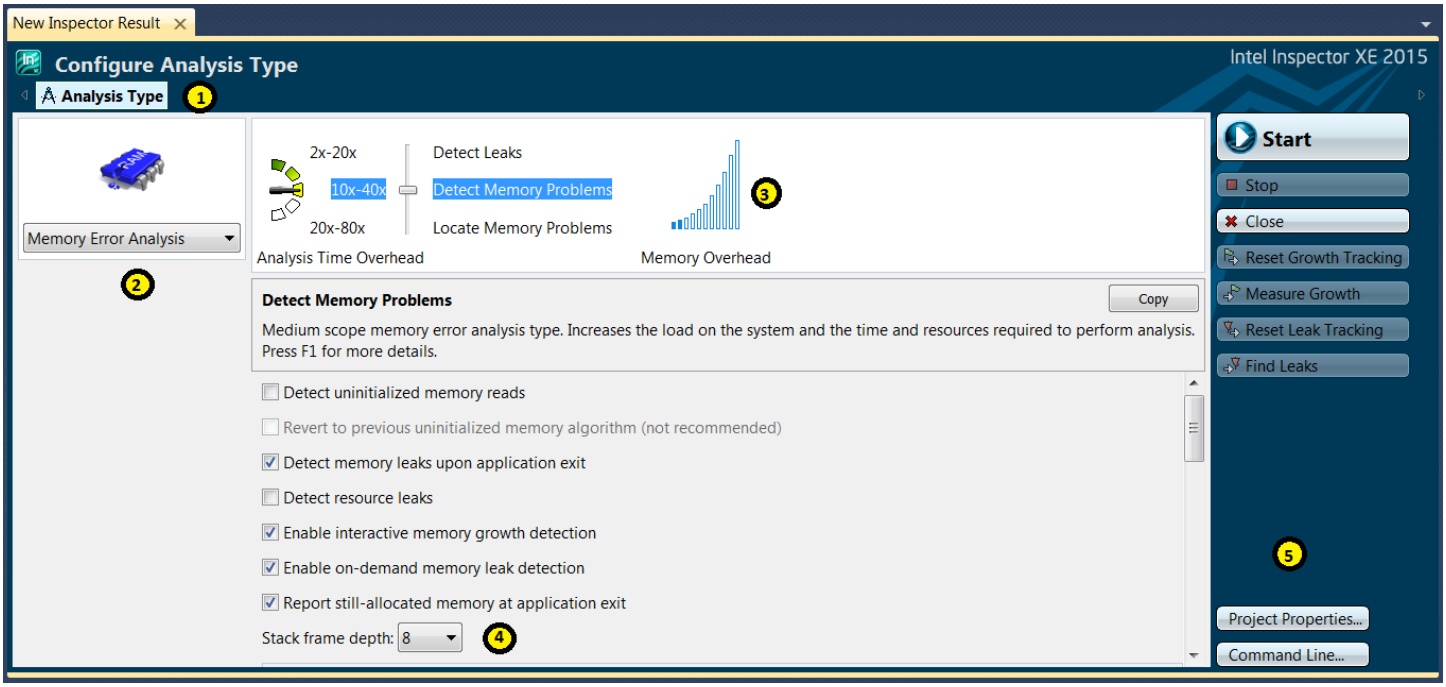

Figure 11

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. The analysis dropdown allows you to choose whether to run memory or threading error analysis or create custom analysis types from existing analysis types.

   This guide covers memory error analysis types, which you can use to search for these kinds of errors: GDI resource leak, incorrect memcpy call, invalid deallocation, invalid memory access, invalid partial memory access, kernel resource leak, memory growth, memory leak, memory not deallocated, mismatched allocation/deallocation, missing allocation, uninitialized memory access, and uninitialized partial memory access.

   Use threading error analysis types to search for these kinds of errors: Data race, deadlock, lock hierarchy violation, and cross-thread stack access.
3. The Analysis Type Overhead display shows time and memory overhead for available preset analysis types.
4. 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.
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 memory error analysis to detect memory issues that may need handling by clicking the Start button to:

- Execute the find_and_fix_memory_errors.exe target.
- Identify memory issues that may need handling.
- Collect the result in a directory in the tachyon_insp_xe/vc10/My Inspector XE Results– find_and_fix_memory_errors directory.
- Finalize the result (convert symbol information into file names and line numbers, perform duplicate elimination, and form problem sets).

During collection, the Intel Inspector XE displays a Collection Log window similar to the following. Figure 12

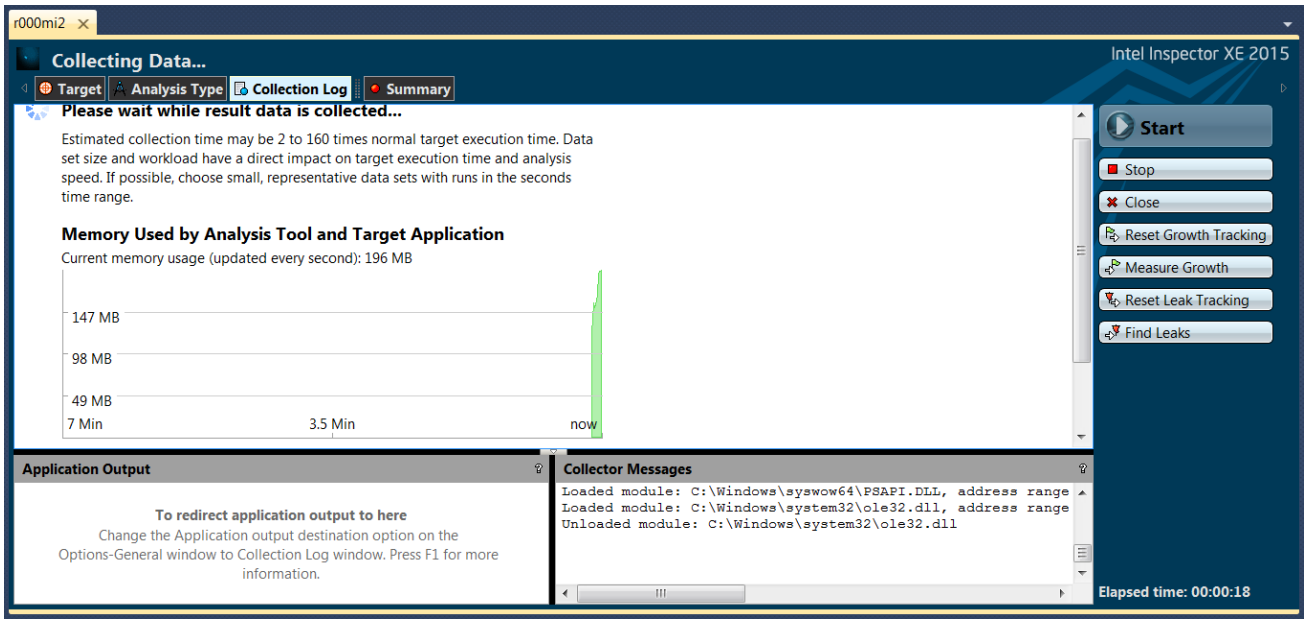

Figure 12

Choose a Problem

The Summary window is the starting point for exploring detected memory issues. Figure 13

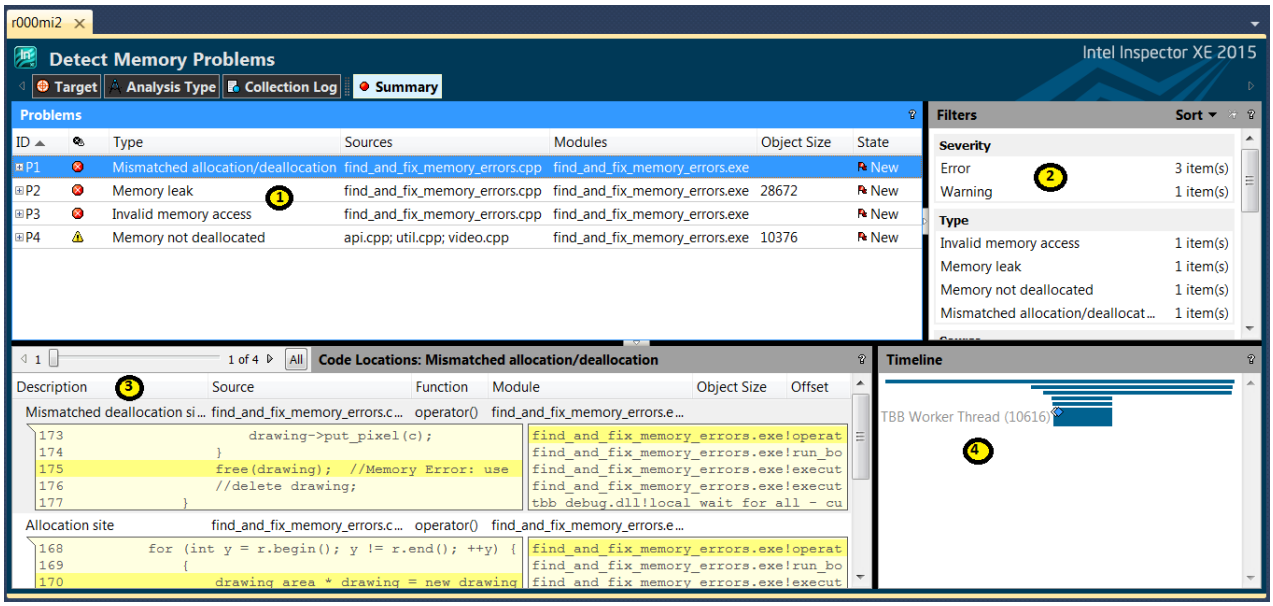

Figure 13

1. The Problems pane shows information about issues found by the tool.

   Related problems are grouped together. Click on the + button on the left hand side of each line to see individual issues in the set.

2. The Filters Pane indicates information you can use to narrow the displayed problem sets. Click on the value you want to see problem sets limited to.

3. The Code Locations pane shows all the code locations in the selected Mismatched allocation/deallocation problem set.

   The Allocation site code location represents the location and associated call stack from which the memory block was allocated. The Mismatched deallocation site code location represents the location and associated call stack attempting the deallocation.

   Sliding the slider bar in this pane allows you to see code locations from different instances of this problem.

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

To choose a memory issue:

1. Click the Sources column header in the Problems pane to sort problem sets by source file location, and, if necessary, scroll to the top of the pane to display a window to find the problem sets in the find_and_fix_memory_error.cpp file.
2. Double-click the data row for the Mismatched allocation/deallocation problem in the find_and_fix_memory_errors.cpp source file to display the Sources window, which provides more visibility into the cause of the error. Figure 14

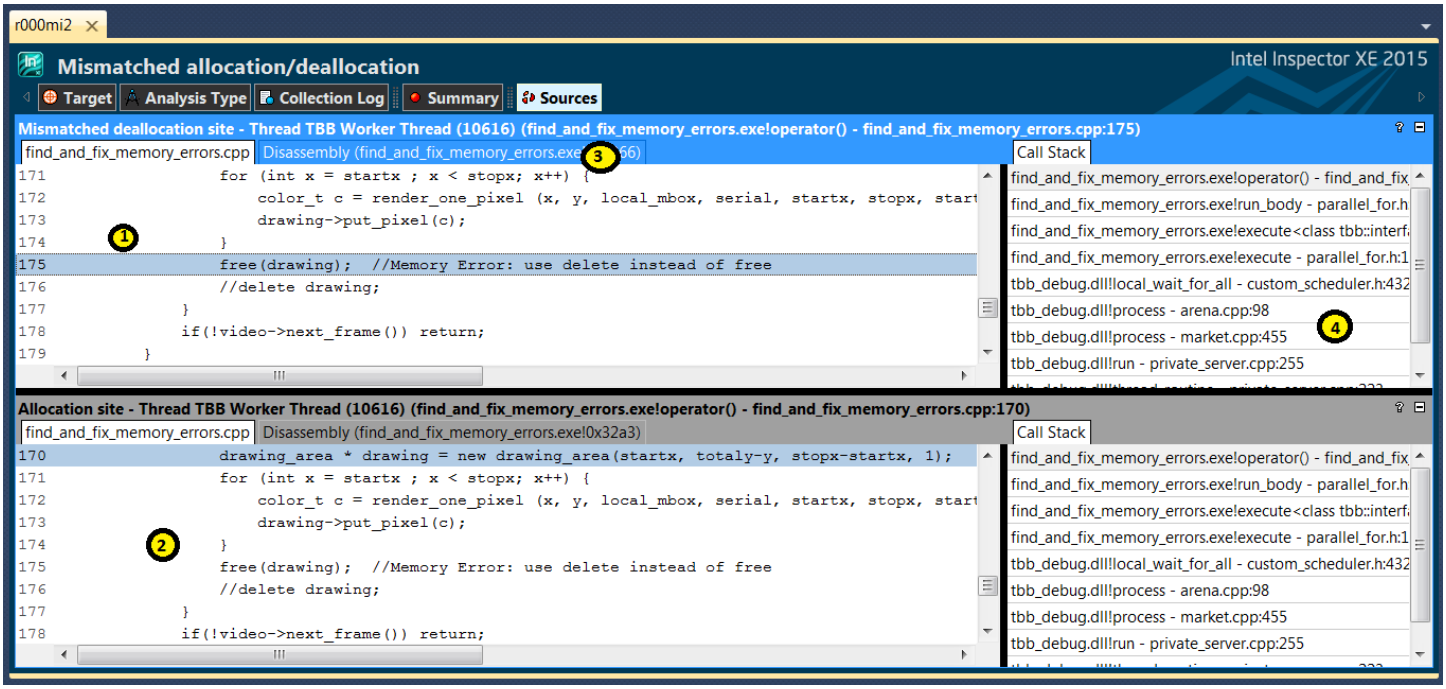

Figure 14

1. In the Sources view you see all the code locations in the Mismatched allocation/deallocation problem in the Mismatched allocation/deallocation problem set.

   The Mismatched deallocation site code location represents the location and associated call stack attempting the deallocation. The source code corresponding to the Mismatched deallocation site code location is highlighted.

2. The Allocation site code location represents the location and associated call stack from which the memory block was allocated. The source code corresponding to the Allocation site code location is highlighted.

3. Use the Disassembly Tab to switch between viewing source code and disassembly.

4. The Call Stack pane provides access to various locations in the call stack.

   Near the top of the call stack there is a small +/- button you can use to expand/collapse a given code location.

Interpret the Result Data

Look at the code in the Sources window. The code in the Allocation site contains a new allocator, while the code in the Mismatched deallocation site contains a free() deallocator.

A Mismatched allocation/deallocation problem occurs when you attempt a deallocation with a function that is not the logical reflection of the allocator. In the C++ programming language, the following are matched reflections:

- new and delete
- new[] and delete[]
- malloc() and free()

Only the matching deallocation technique is uniquely aware of all the memory allocation techniques and internal data storage used by the allocation technique. Using the wrong deallocation technique will almost certainly corrupt memory reclamation, its sole job.

NOTE: A Mismatched allocation/deallocation problem does not always cause an application crash; however, if it does cause a crash, the crash may occur later at a seemingly unrelated location.

You determined the cause of the Mismatched allocation/deallocation problem set in the find_and_fix_memory_errors.cpp source file: new and free are not matching techniques.

Resolve the Issue

Use the Visual Studio editor to fix the memory issue:

1. Double-click the highlighted code in the Mismatched deallocation site code location on the Sources window to open the find_and_ fix_memory_errors.cpp source file in a separate tab. From there, you can edit it with the Visual Studio editor: Figure 15

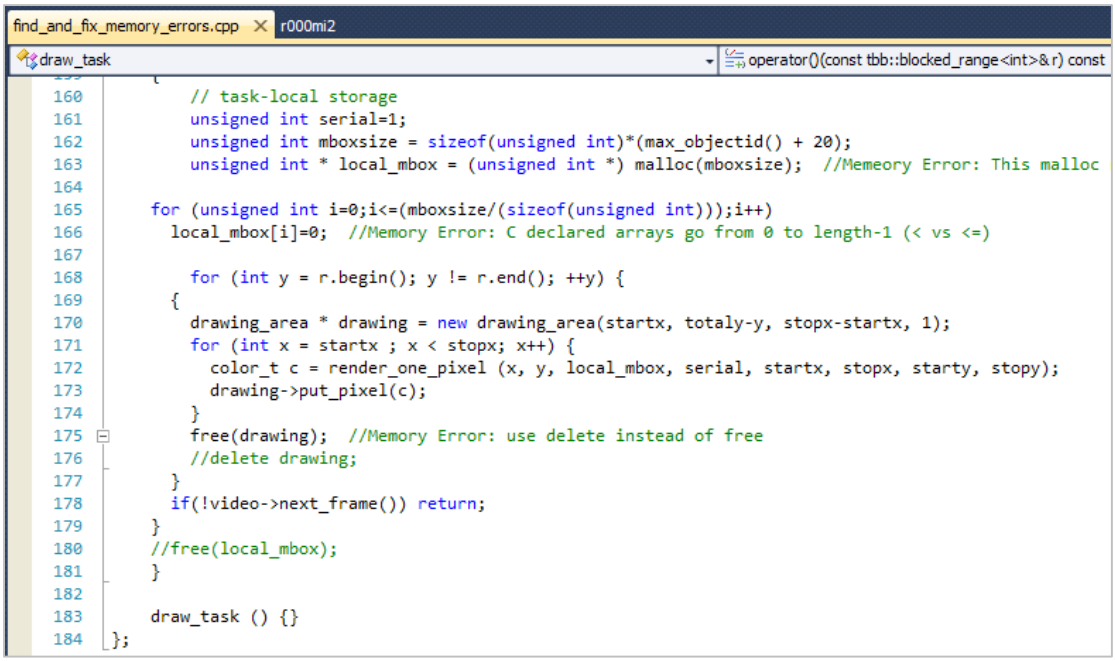

Figure 15

2. Comment free(drawing); and uncomment // delete drawing;:

Rebuild and Rerun the Analysis

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

To rebuild the target: In the Solution Explorer pane, right-click the find_and_fix_memory_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 2015 > New Analysis... and follow the steps above to execute the find_and_fix_memory_errors.exe target and display the following: Figure 16

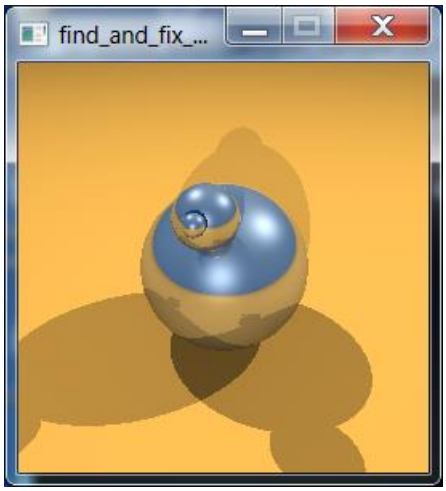

Figure 16

Success

In this example, we had a bug and the program was not behaving correctly. In addition, the graphics were not displayed consistently during rendering. After running the Intel Inspector XE, we found the bug, and now the graphics render consistently. Often, you will be able to obtain the same results on your own application by running the 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 the Intel Inspector XE.

If you have multithreaded your program to take advantage of the new multi-core processors from Intel, you will be excited to learn that Intel Inspector XE also detects threading errors such as latent data races and deadlocks.

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

Choose Small, Representative Data Sets

When you run an analysis, the 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.

Manage Threading Errors

Intel Inspector XE can also identify, analyze, and resolve threading errors, such as latent data races and deadlocks in parallel programs. Subtle errors can manifest intermittently and non-deterministically, making them extremely hard to find, reproduce, and fix.

Use 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, you need to run the Intel Inspector XE on multiple versions of your code, on different workloads that stress different code paths, as well as on corner cases. 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 17

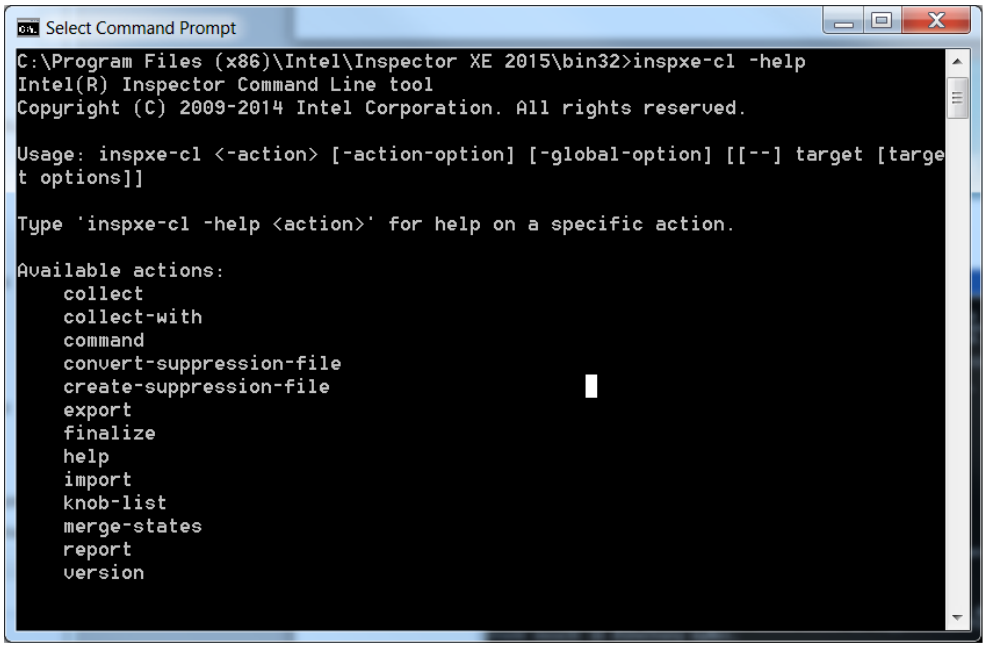

Figure 17

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, ˆ
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*) Linux* (GNU) OS X*[4] (XCode*)	Windows (Visual Studio) Linux (GNU)	Windows (Visual Studio) 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

eliminate-memory-errors_studioxe-evalguide_/Rev-082014