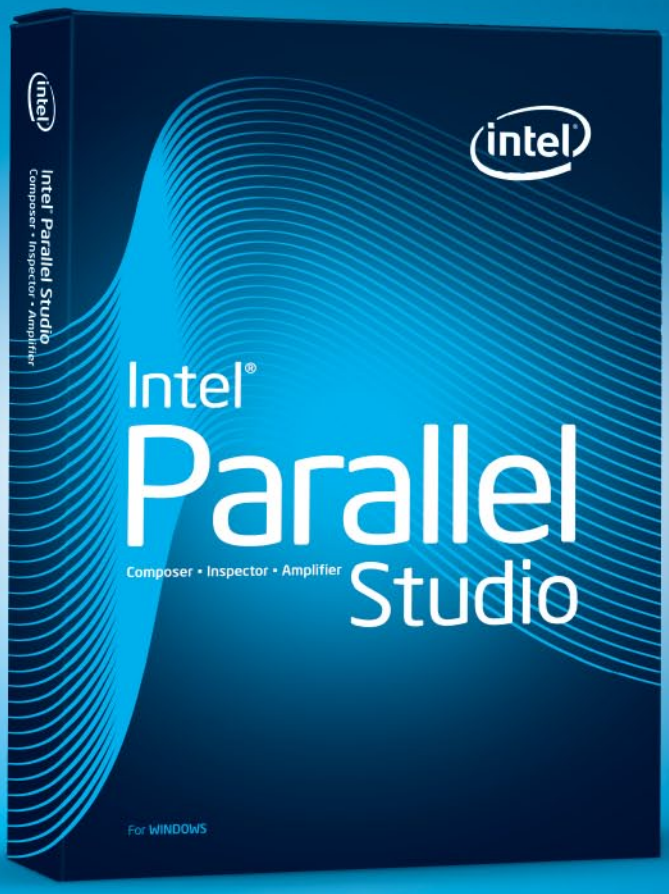
INTEL® PARALLEL STUDIO EVALUATION GUIDE

Eliminate Memory Errors and Improve Program Stability



Eliminate Memory Errors and Improve Program Stability

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® Parallel Inspector 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

Step 1. Install Intel Parallel Studio

1. Download an evaluation copy of Intel Parallel Studio.
2. Install Intel Parallel Studio by clicking on the parallel_studio_setup.exe.

Step 2. Install and View the Tachyon Sample Application

Install and open the sample application:

1. Download the Tachyon_conf.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.

Step 3. Find Memory Errors Using Intel® Parallel Inspector

Intel® Parallel Inspector is a serial and multithreading error-checking analysis tool for Microsoft* Visual Studio* C/C++ developers. Intel Parallel Inspector 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: Intel Parallel Inspector integrates into Microsoft Visual Studio 2005 and 2008. These tutorials contain instructions and screens for the Microsoft Visual Studio 2005 development environment (IDE). To use a different IDE, replace the menu items with the related menu items for your IDE.

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

Eliminate Memory Errors and Improve Program Stability

Identify, Analyze, and Resolve Memory Errors

You can use Intel Parallel Inspector to identify, analyze, and resolve memory errors in serial or 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:

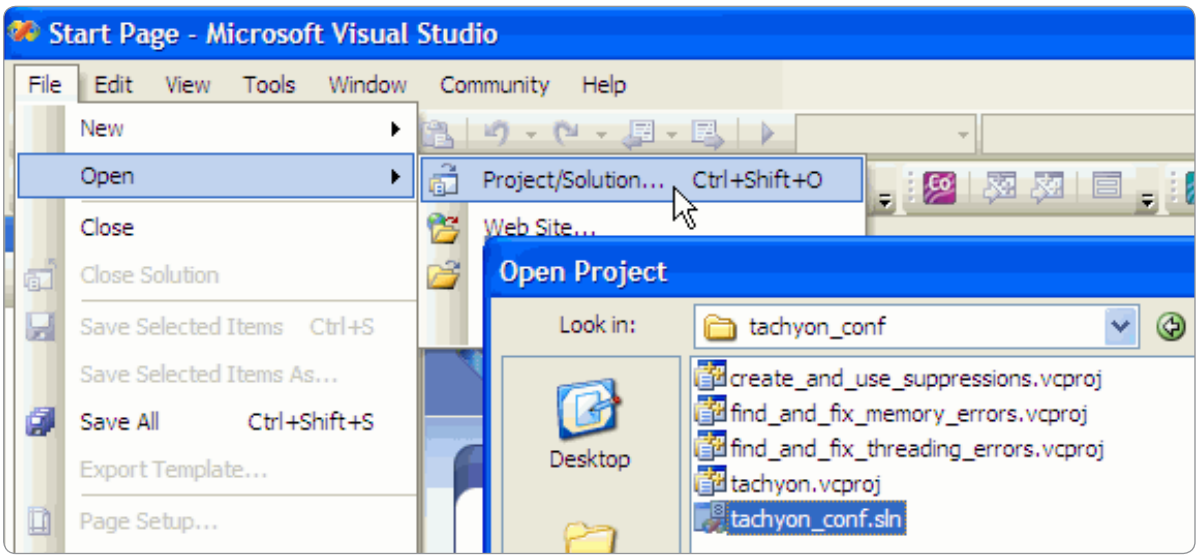


Figure 1

   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_memory_errors** project and select **Set as Startup Project**.

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

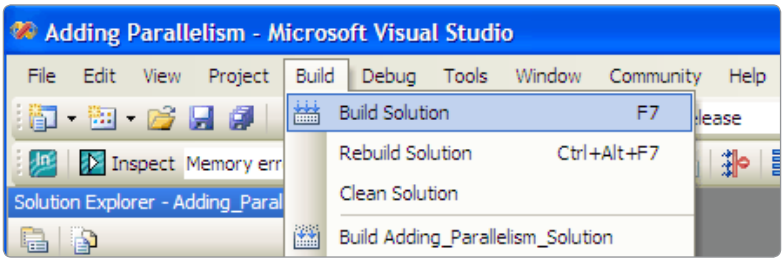


Figure 2

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

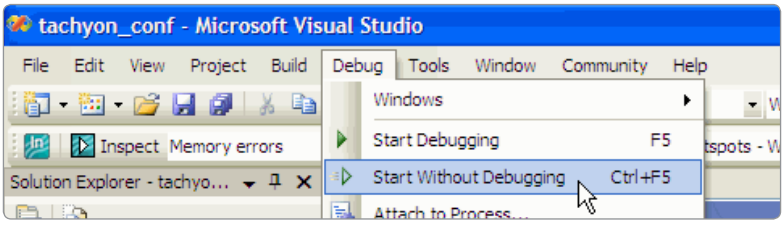


Figure 3

Eliminate Memory Errors and Improve Program Stability

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 Parallel Inspector can check for memory errors.

You can use Intel Parallel Inspector 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

To verify that debug mode is configured:

1. In the **Solution Explorer** pane, right-click the **find_and_fix_memory_errors** project and select Properties. **Figure 5**
2. Check that the **Configuration** drop-down list is set to **Debug**, or **Active(Debug)**. **Figure 6**
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)**.

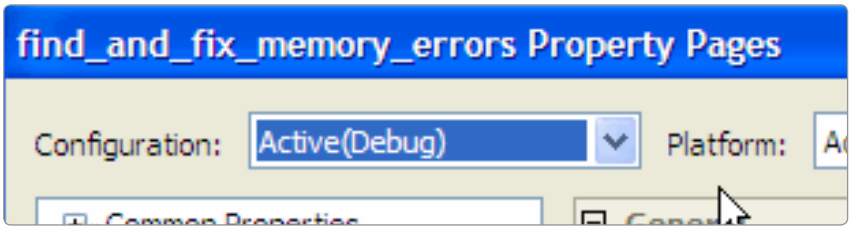


Figure 5

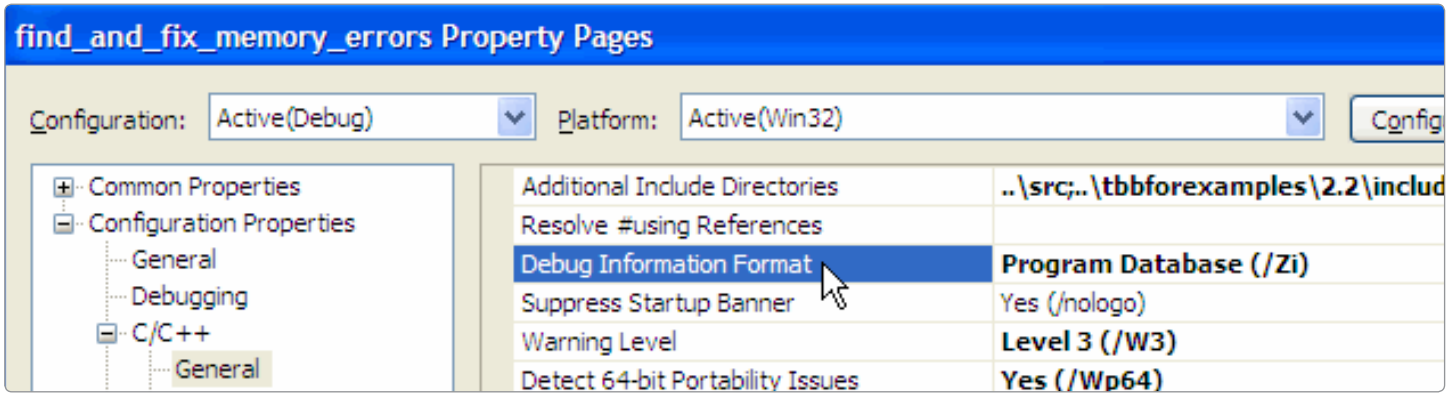


Figure 6

Eliminate Memory Errors and Improve Program Stability

Figure 7

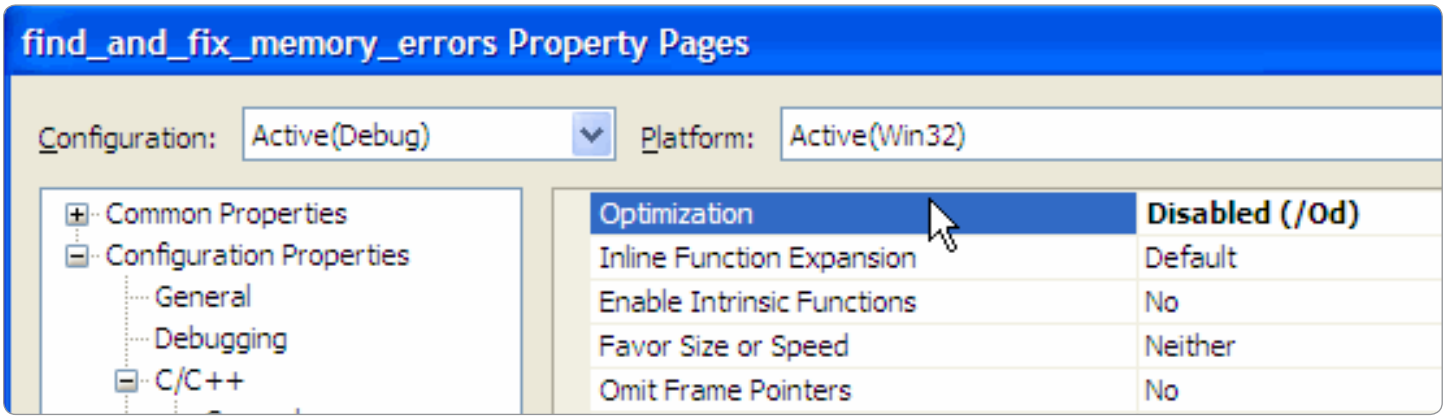
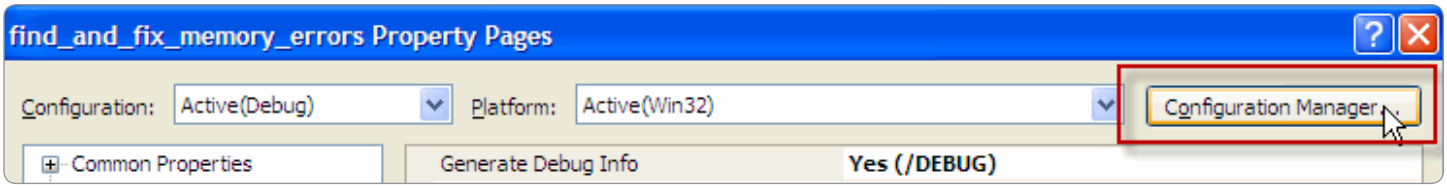


Figure 8



4. Choose **Configuration Properties** > **C/C++** > **Optimization**. Verify the **Optimization** field is set to **Disabled (/Od)**. **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)**. **Figure 9**

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

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

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

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

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

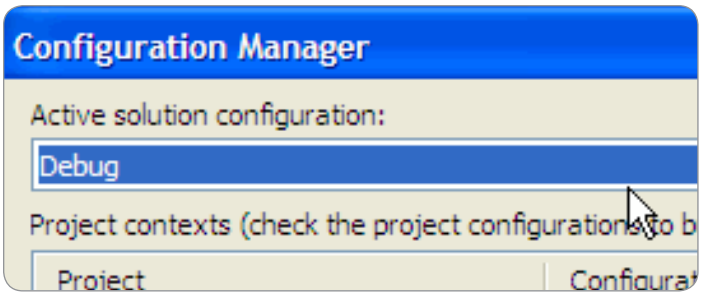


Figure 9

Build the Target

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

As you can see, the image is not rendered fully, correctly, and consistently. **Figure 10**

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

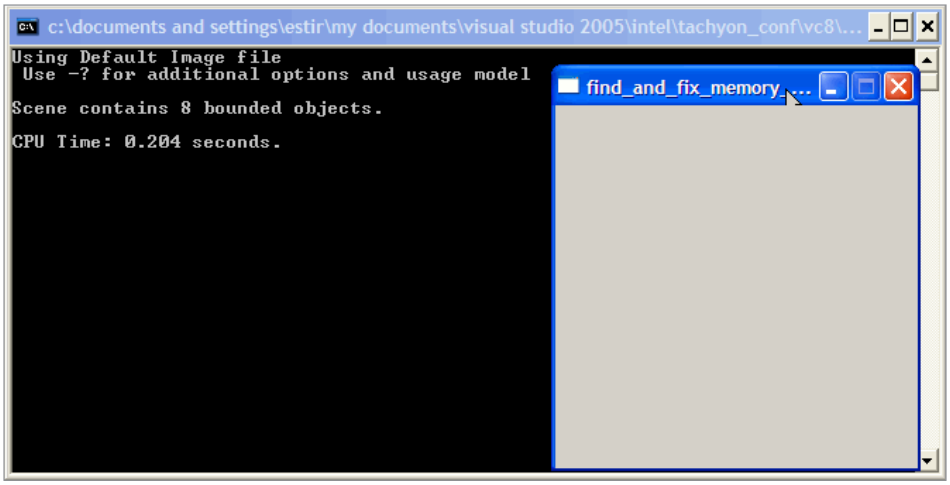


Figure 10

Figure 11

Eliminate Memory Errors and Improve Program Stability

Configure Analysis

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

To configure a memory error analysis:

1. From the Microsoft Visual Studio menu, choose **Tools > Intel Parallel Inspector > Inspect Memory Errors** to display the **Configure Analysis dialog** box.
2. Drag the configuration slider to the **Where are the memory access problems?** preset configuration. **Figure 12**

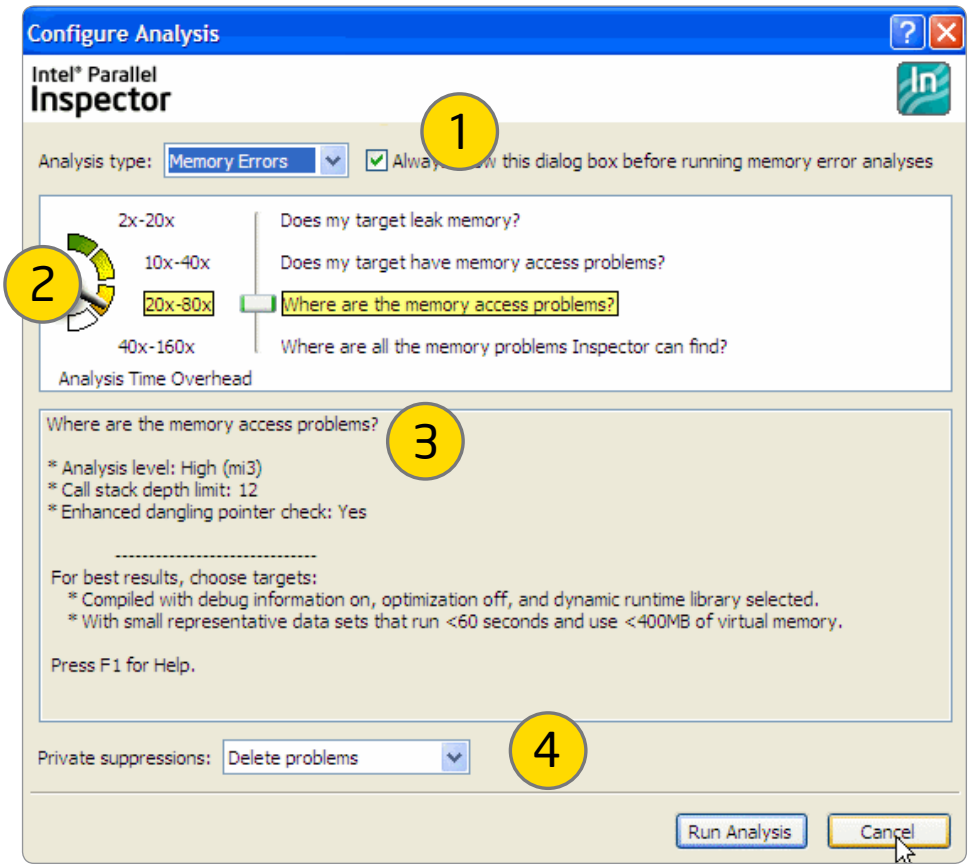


Figure 12

1. The **Analysis type** drop-down list shows Intel Parallel Inspector dynamic analysis type offerings: Memory error analysis and threading error analysis.

   This tutorial covers memory error analysis, which you can use to search for the following kinds of errors: invalid memory access, memory leak, mismatched allocation/deallocation, missing allocation, and uninitialized memory access.

2. The **Analysis Time Overhead** gauge shows the time it may take to collect a result at each preset configuration. Time is expressed in relation to normal target execution time. For example, 2x - 20x is 2 to 20 times longer than normal target execution time. If normal target execution time is 5 seconds, estimated collection time is 10 to 100 seconds.

   The configuration slider rests at the **Does my target leak memory?** preset configuration, which has the lowest overhead.

3. The configuration details region shows current configuration characteristics. Try dragging the configuration slider to see the impact on the gauge and details region.

4. *Suppressing* known issues can dramatically improve your productivity. For more details, check the Intel Parallel Inspector Help Index for *suppression rule*.

Eliminate Memory Errors and Improve Program Stability

Run the Analysis

Run a memory error analysis to detect memory issues that may need handling. **Figure 13**

To run a memory error analysis:

Click the **Run Analysis** button on the **Configure Analysis** dialog box 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_conf/vc8/My Inspector Results directory.
- Finalize the result (convert symbol information into file names and line numbers, perform duplicate elimination, and form problem sets).

During collection, Intel Parallel Inspector displays an **Event Log** window similar to the following:

To start managing result data after analysis is complete:

Click the **Interpret Result** button to display the **Overview** window.

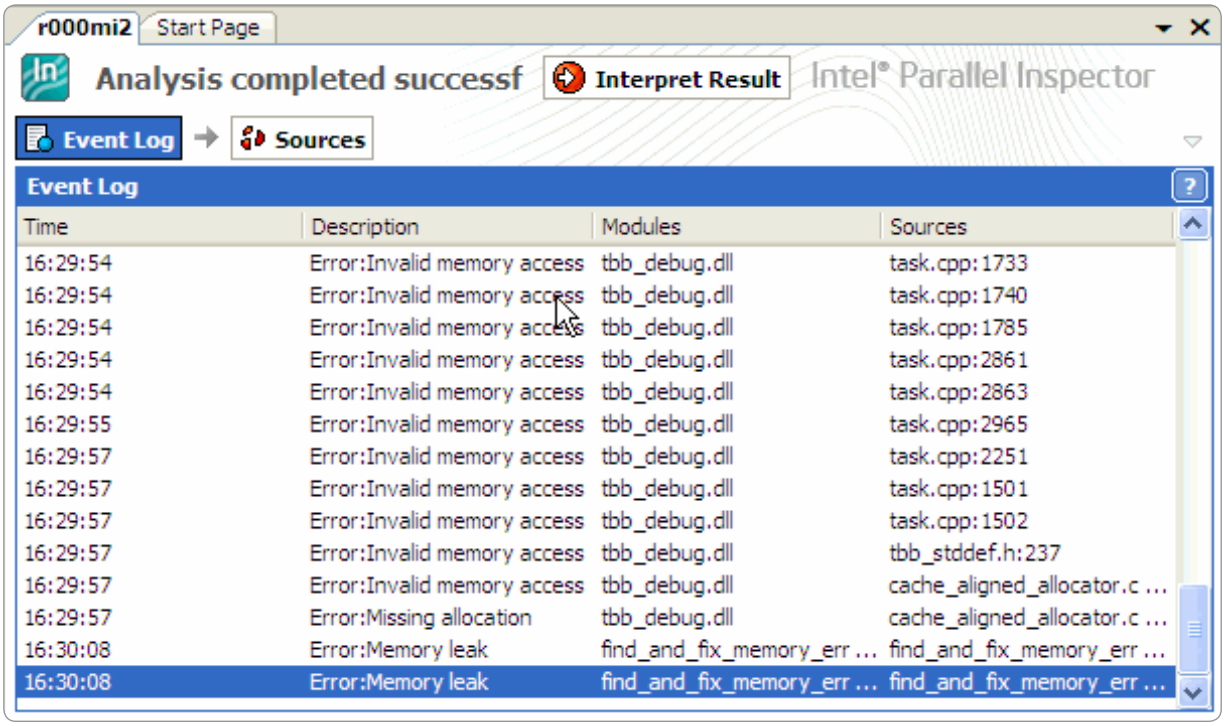


Figure 13

Choose a Problem Set

Choose a problem set on the **Overview** window to explore a detected memory issue. **Figure 14**

To choose a problem set:

1. Click the **Sources** column header in the **Problem Sets** 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 set in the **find_and_fix_memory_errors.cpp** source file to display the **Sources** window, which displays the source code for the focus observation and related observations:

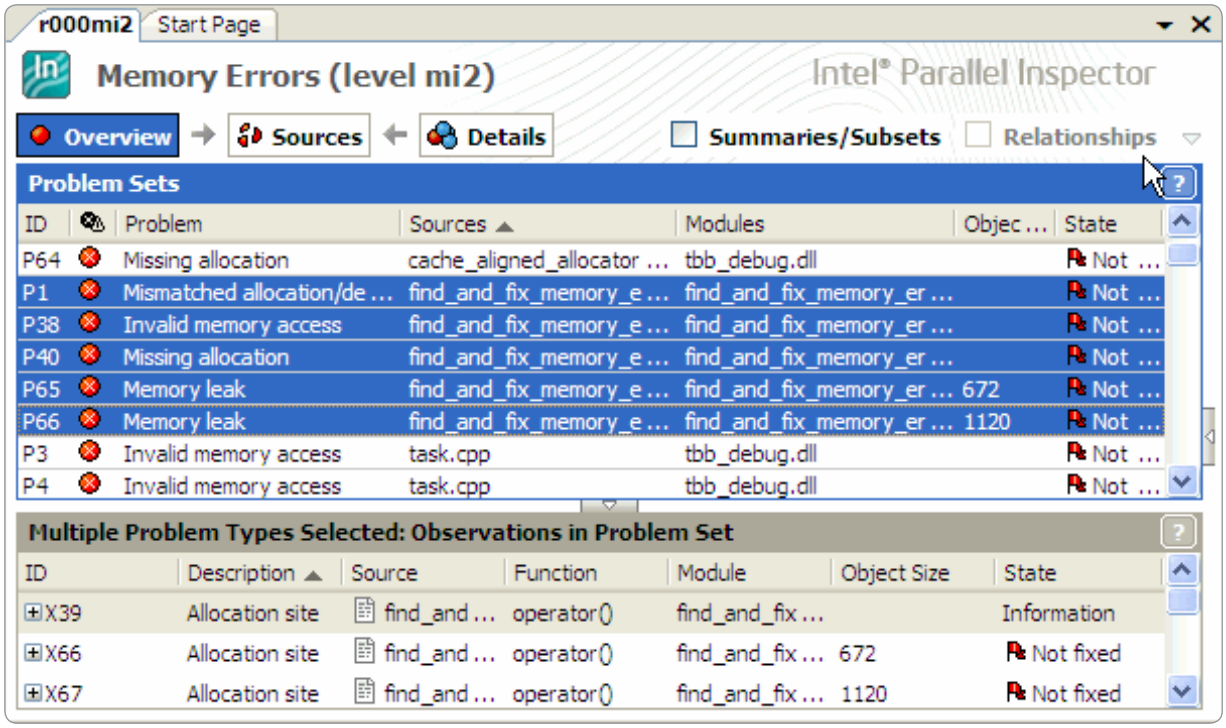


Figure 14

Eliminate Memory Errors and Improve Program Stability

You started exploring a **Mismatched allocation/deallocation** problem set that contains one **Allocation** site and one **Mismatched deallocation** site observation in the **find_and_fix_memory_errors.cpp** source file.
Figure 15

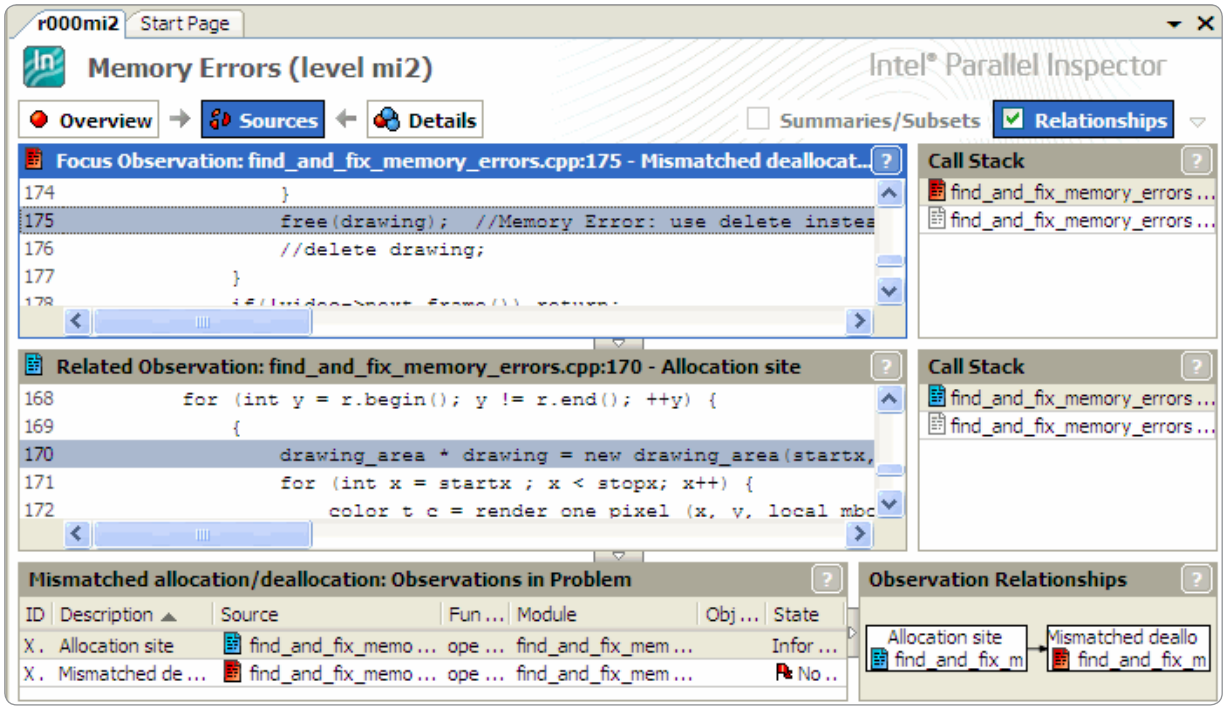


Figure 15

Eliminate Memory Errors and Improve Program Stability

Interpret the Result Data

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

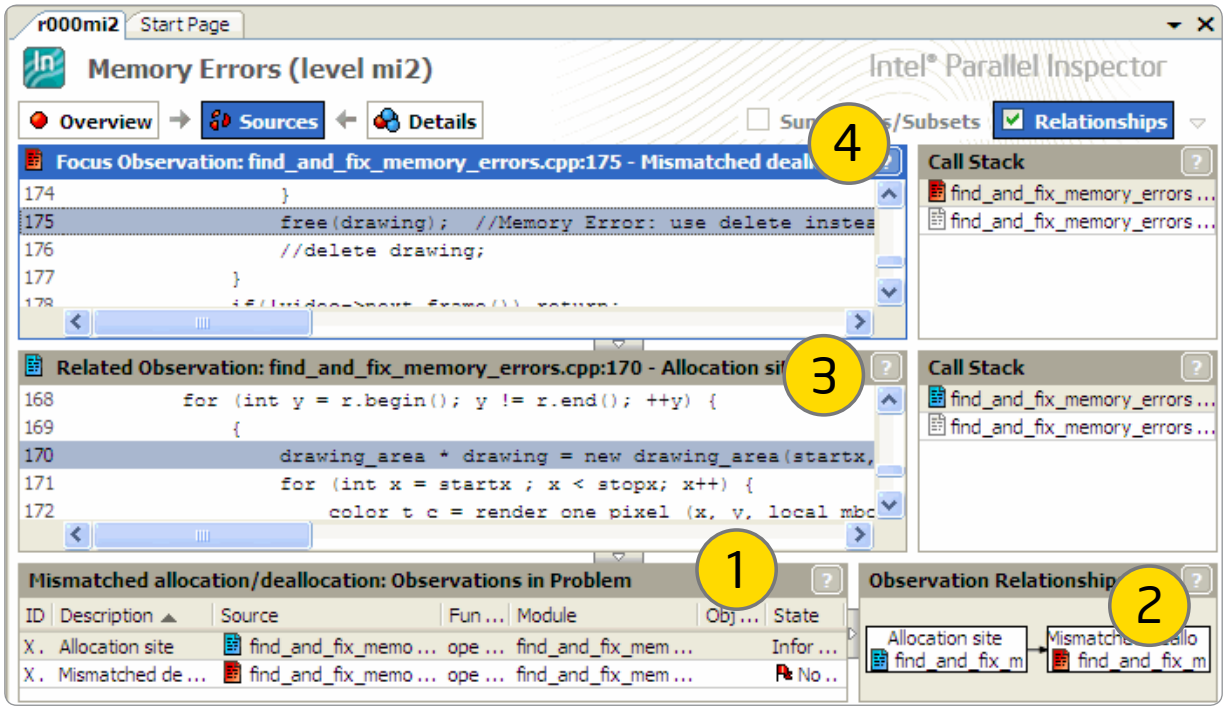


Figure 16

1. The **Observations** in **Problem Set** pane shows all the observations in all the problems in the **Mismatched allocation/deallocation** problem set.

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

2. The **Observation Relationships** pane shows the relationship between the Allocation site and **Mismatched deallocation** site observations. In relationship diagrams:

   - Each box in a diagram represents an observation in one problem in a problem set.
   - A diagram with a single box is a trivial problem with no related observation.
   - Vertically stacked boxes indicate concurrent observations.
   - Boxes arranged left to right with connecting arrows indicate a time ordering. The **Allocation site** observation occurs before the **Mismatched deallocation site** observation.
   - Boxes with connecting lines indicate association.

3. The **Related Observation Code** pane shows the source code in the find_and_fix_memory_errors.cpp source file surrounding the **Allocation site** observation. The source code corresponding to the **Allocation site** observation is highlighted.

4. The **Focus Observation Code** pane shows the source code in the find_and_fix_memory_errors.cpp source file surrounding the **Mismatched deallocation site** observation. The source code corresponding to the **Mismatched deallocation site** observation is highlighted.

Eliminate Memory Errors and Improve Program Stability

To access more information on interpreting and resolving problems:

1. Right-click the **Mismatched deallocation site** observation in the **Observations in Problem Set** pane.
2. Choose **Explain Problem** to display the Intel Parallel Inspector Help information for the **Mismatched allocation/deallocation** problem type.

To interpret result data:

Look at the code in the **Focus Observation Code** pane and the **Related Observation Code** pane.

The code in the **Allocation site** observation in the **Related Observation Code** pane contains a new allocator, while the code in the **Mismatched deallocation site** observation in the **Focus Observation Code** pane 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

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

To resolve the issue:

1. Double-click the highlighted code in the **Focus Observation** code pane 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 Microsoft Visual Studio editor: **Figure 17**
2. Comment free(drawing); and uncomment // delete drawing;:

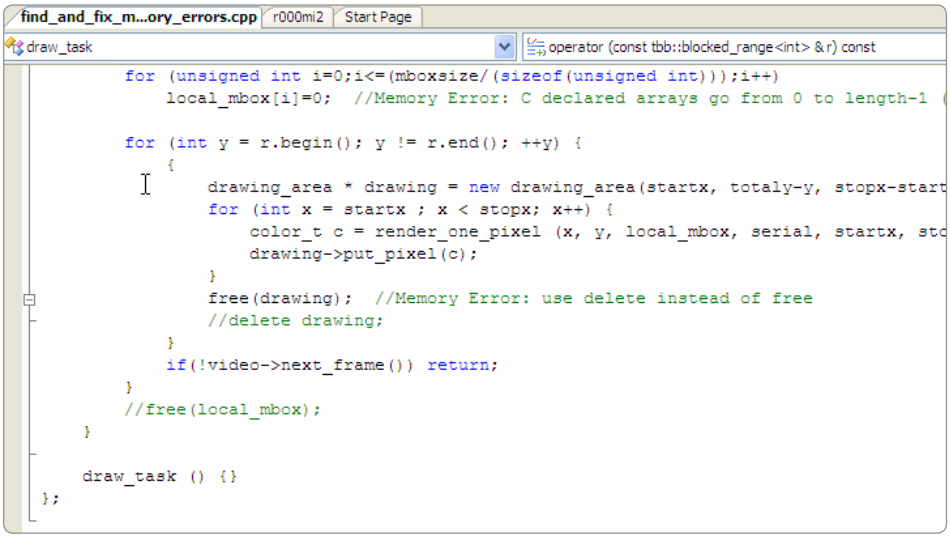


Figure 17

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 issues.

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 Parallel Inspector** > **Re-inspect, or Inspect Memory Errors** to execute the find_and_fix_memory_errors.exe target and display the following: **Figure 18**

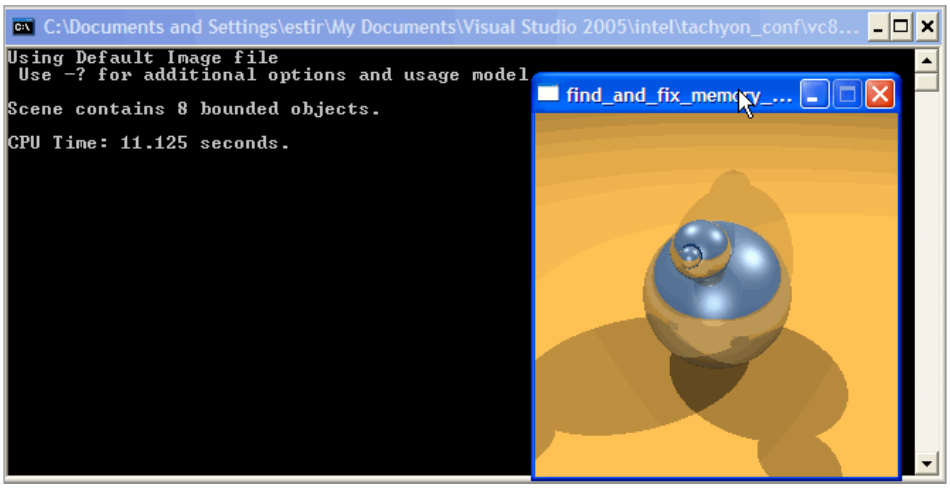


Figure 18: Notice that the image now displays correctly.

Success

In this example, we had a bug and the program was not behaving correctly. In addition, the graphics were not displayed consistently during rendering. After running Intel Parallel Inspector, 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 Intel Parallel Inspector 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 Parallel Inspector.

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

Intel Parallel Inspector 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 Parallel Inspector 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 Threading Errors

Intel Parallel Inspector 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.

Using the Command-line to Automate Testing

As you can see, Intel Parallel Inspector has to execute your code path to find errors in it. Thus, run Intel Parallel Inspector 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 Parallel Inspector command-line version is called insp-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 Parallel Inspector. **Figure 19**

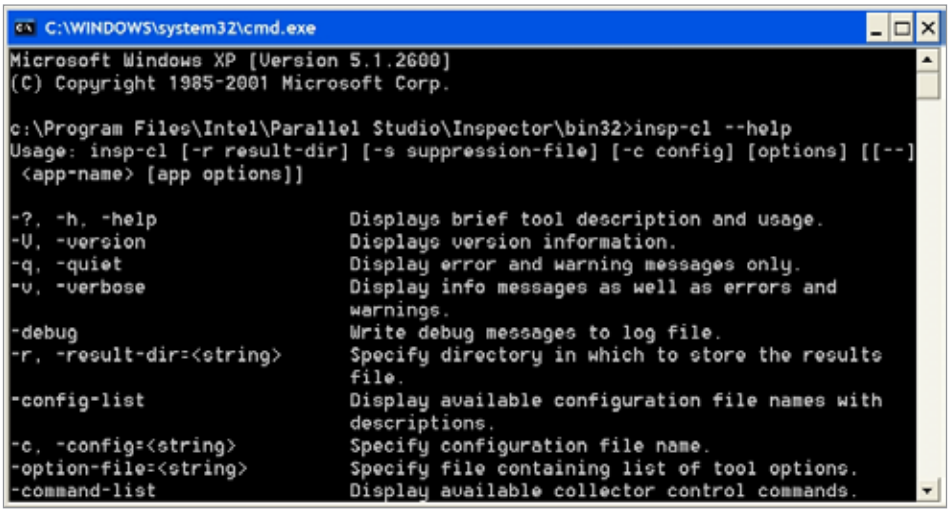


Figure 19

To get help on insp-cl, enter --help.

```
> c:\Program Files\Intel\
Parallel Studio\Inspector\
bin32\insp-cl --help
```

Eliminate Memory Errors and Improve Program Stability

The Path to Parallelism

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

Related links

- Intel® Software Network Forums
- Intel® Software Products Knowledge Base
- Intel® Software Network Blogs
- Intel® Parallel Studio Website
- Intel® Threading Building Blocks Website
- Go Parallel—Parallelism Blogs, Papers, and Videos
- Free, On-Demand Software Developer Webinars

Check out additional evaluation guides:

- Locate a Hotspot and Optimize It
- Optimize an Existing Program by Introducing Parallelism

0510/BLA/CMD/PDF 323859-001US