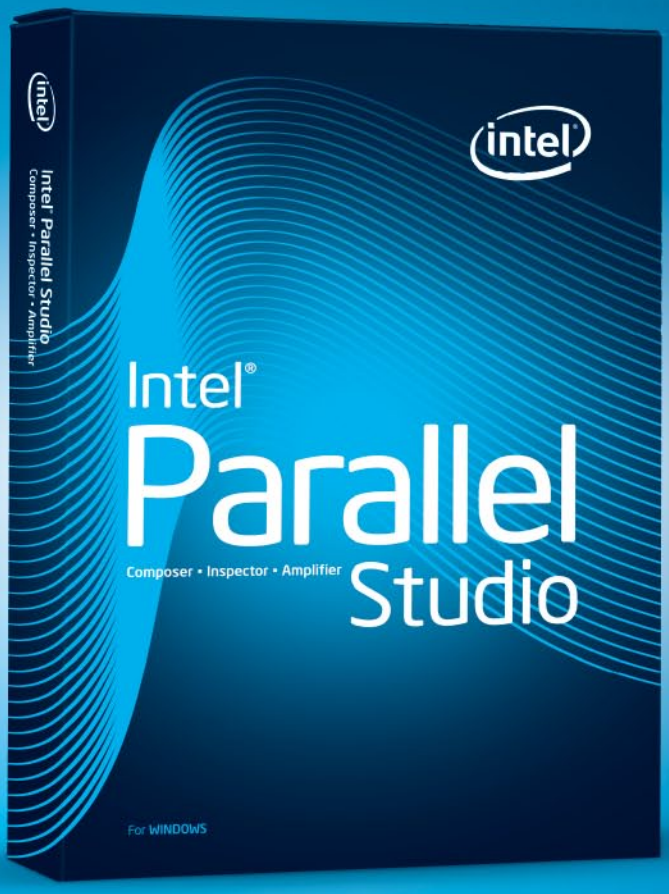
INTEL® PARALLEL STUDIO EVALUATION GUIDE

Locate a Hotspot and Optimize It



Locate a Hotspot and Optimize It

Can Recompiling Just One File Make a Difference?

Yes, in many cases it can! Often, you can get a major performance boost by recompiling a single file with the optimizing compiler in Intel® Parallel Studio. You do not always need to recompile the entire app, a truism that applies to both serial and parallel applications. **Figure 1**

Two Easy Steps to Better Performance

Step 1. Find the hotspot(s): Measure where the application is spending time

In order to tune effectively, you must optimize the parts of the applications that demand a lot of time. Tune something that is already fast, and you will see very little benefit. A "hotspot" is a place where the app is spending a lot of time. We want to find those areas and speed them up. This is easily done using a profiling tool like Intel® Parallel Amplifier. So, do not waste your time optimizing things that do not need it—find your hotspots.

OK, you have found the hotspot, now what? In some cases, it may be obvious how to make the program run faster. For example, you may find you are repeating an operation that you only need to do once. Unfortunately, in most cases the answer is less obvious. People often ask, "Can't you suggest something or do it automatically?" In many cases, we can.

Step 2. Optimize it: Recompile just the hotspot *(even just one file)*

The optimizing compiler in Intel® Parallel Composer can often improve performance just by recompiling the file(s) in which the hotspot(s) are located.

On smaller applications, you can just recompile everything and see what you get. On large applications with many modules and projects, this may be impractical. Fortunately, there is rarely a need to recompile the entire application. Recompiling one or two files may be all that is necessary, or perhaps just a single project. And, since the Intel® Compiler is binary and debug compatible with the Microsoft* compiler, you can seamlessly mix and match objects built with either tool.

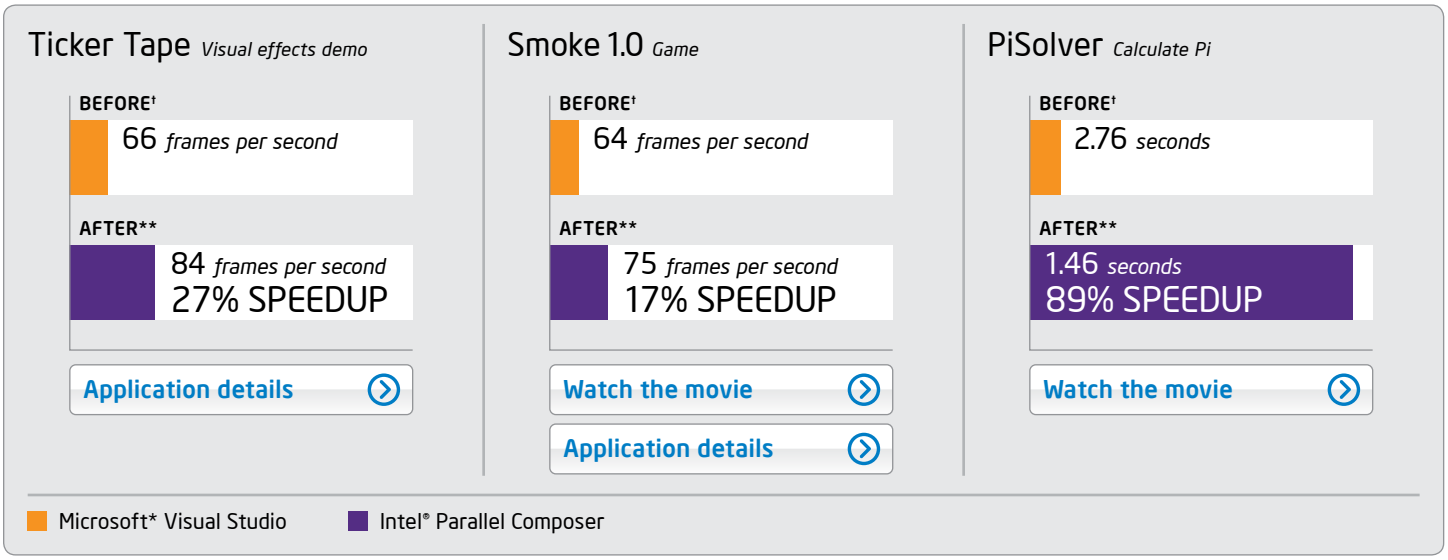


Figure 1

†Microsoft Visual Studio 2008* used for Ticker Tape and PiSolver; Microsoft Visual Studio 2005* for Smoke

**Intel® Parallel Composer, update 5

System Specifications: Ticker Tape and Smoke: Intel® Core™ i7 processor (4 cores), 3.20 GHz, 3.0 GB RAM, NVIDIA GeForce 9800 GX2; Windows* Vista Ultimate SP2; PiSolver: Intel® Core™ 2 Duo 1.2 GHz (Centrino® Pro Laptop), 2 GB RAM, Microsoft* Windows XP SP 3

Try It Yourself

Here is a simple example using the Intel C++ Compiler included in Intel Parallel Studio. You can read it here or try it yourself using the steps below and the PiSolver sample code.

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 PiSolver Sample Application

1. Download the PiSolver Sample.zip sample file to your local machine.
2. Extract the files from the PiSolver.zip file to a writable directory or share on your system.
3. Open the sample in Microsoft Visual Studio. Go to **File** > **Open** > **Project Solution:** **Figure 2**
4. Set the PiSolver sample application to **Release** mode. Select **Build** > **Configuration Manager** and then, under the

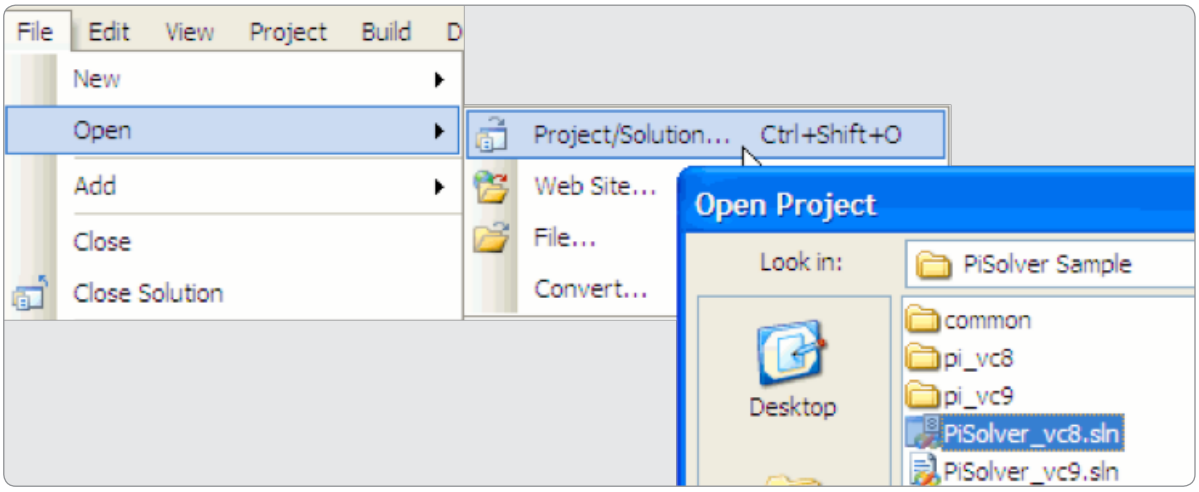


Figure 2

Locate a Hotspot and Optimize It

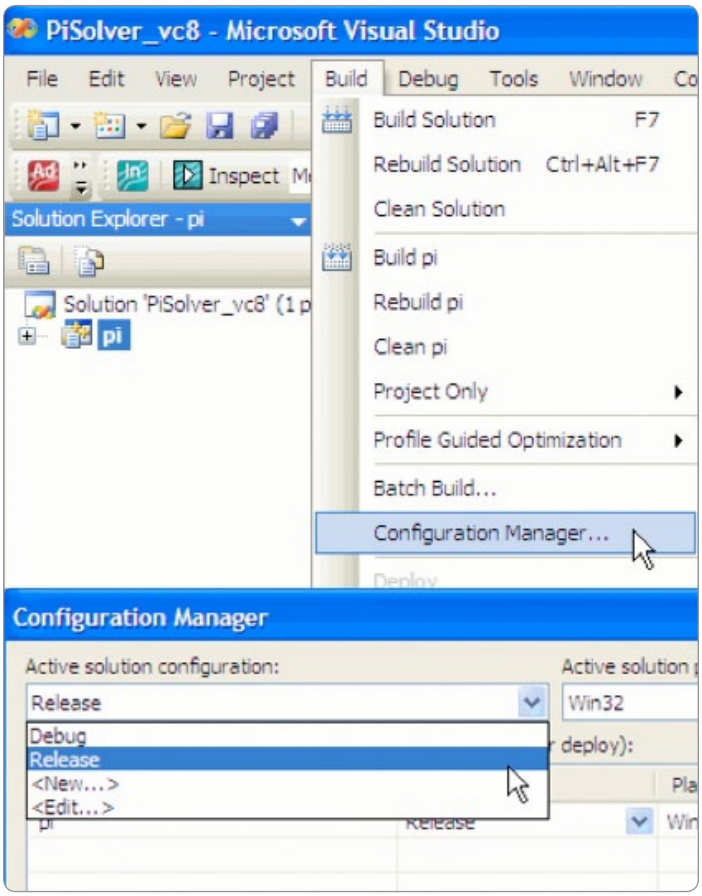


Figure 3

Active
Solution Configuration drop-down box, select the **Release** setting and close the **Configuration Manager**. **Figure 3**

5. Build the application. Go to **Build** > **Build Solution**.
6. Run the application from Microsoft Visual Studio. Go to **Debug** > **Start Without Debugging**. **Figure 4**
7. Click the **Calculate** button to compute the pi value and see the time it took in milliseconds. **Figure 5**

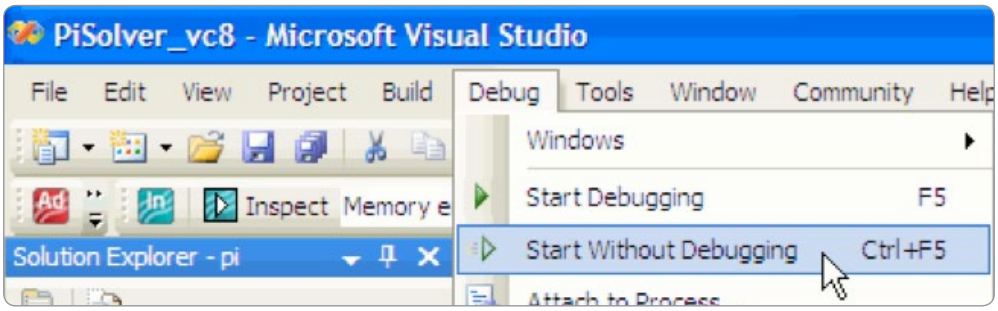


Figure 4

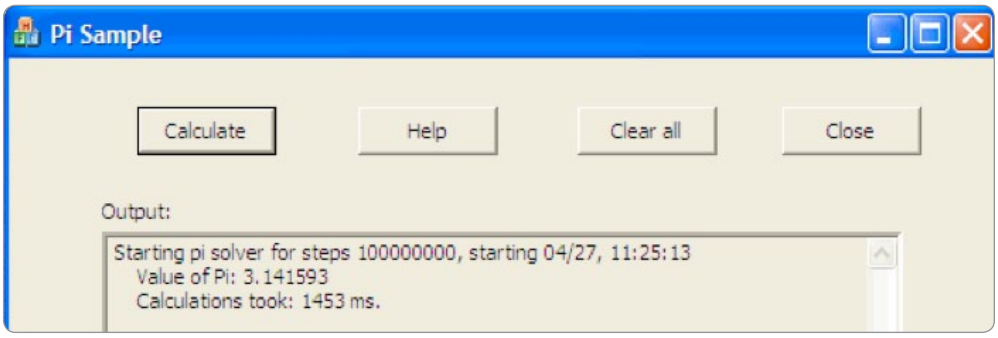


Figure 5

Step 3. Run Intel® Parallel Amplifier—Find the Hotspots

ESTIMATED COMPLETION TIME: 10 MINUTES

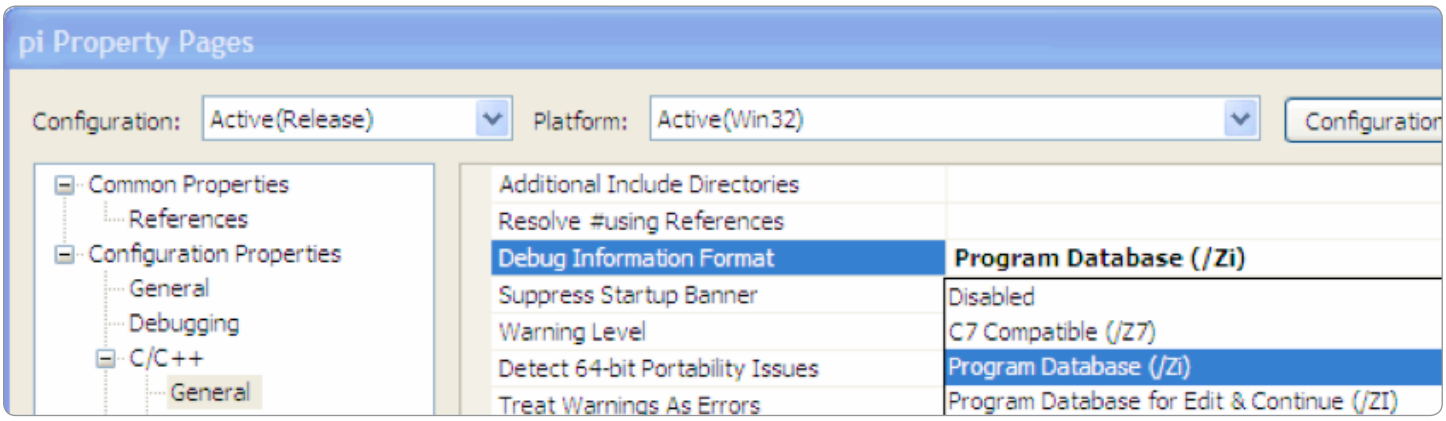
Intel® Parallel Amplifier enables you to quickly find the best place to focus your performance tuning.

Configure the Project Settings

Set the configuration to Release *(optimized)* with debug symbols. This setting enables the Intel Parallel Amplifier to provide the most useful information about the application.

1. In the Microsoft Visual Studio **Solution Explorer** window, right-click the Pi project and select **Properties**.
2. Expand the **Configuration Properties**, if it is not already expanded, and click the plus (+) sign next to **Configuration Properties**.
3. Set the general debug information property.
4. Expand **C/C++** and click **General**.
5. Under the Debug Information Format, select **Program Database (/Zi)** and click **Apply**. **Figure 6**
6. Set the linker debug information property.
7. Expand **Linker** and click **Debugging**.
8. Select **Generate Debug Info** > **Yes (/DEBUG)** and click **Apply**. **Figure 7**

Figure 6



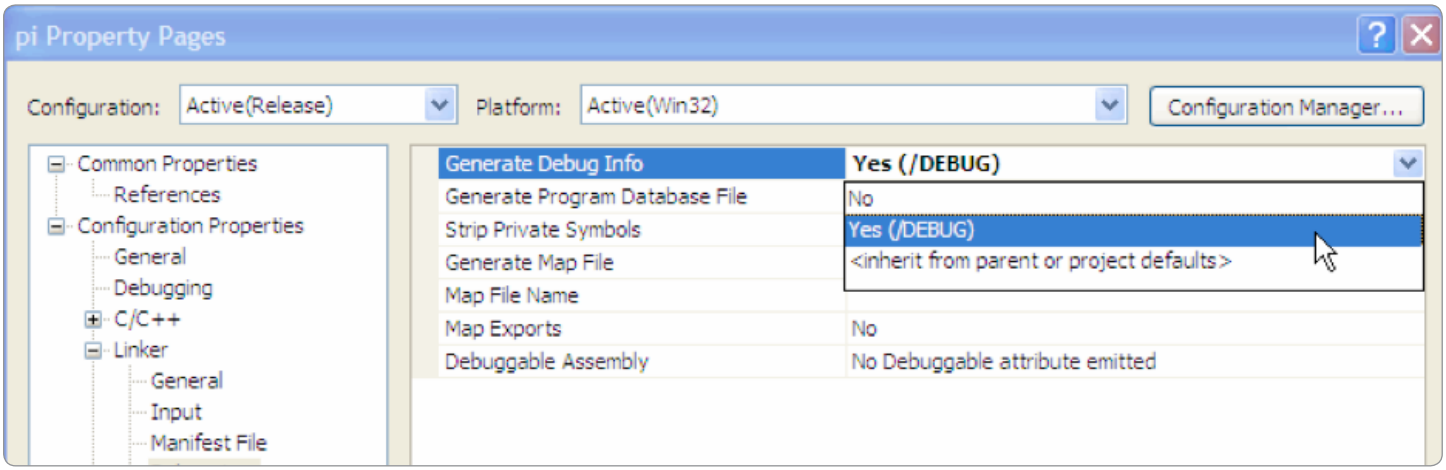


Figure 7

Locate a Hotspot and Optimize It

Run Intel Parallel Studio Amplifier

1. In the Intel Parallel Amplifier toolbar, click the drop-down button and choose **Hotspots—Where is my program spending time?** **Figure 8**
2. Click the **Profile** button to run Intel Parallel Amplifier.

   Intel Parallel Amplifier launches the Pi Sample application.
3. Click **Calculate** to start the application and get results. Record the Output results. This is your baseline measurement.
4. Click **Close** to close the application and start analysis by Intel Parallel Amplifier. You may see the Hotspot Analysis explanation text box covering the report; read and close this first.

   Intel Parallel Amplifier results show a hotspot in the **Hotspots** and the **Call Stack** panes : CalcPi(int) - pi.cpp. **Figure 9**
5. Click the plus (+) sign in front of CalcPi in the Caller Function Tree to expand the call tree for that module.
6. Double-click the hotspot for **CalcPi (piGetSolutions)** to find the source file involved with the hotspot.

   For some applications, it may be easier to see the call tree by using the **Top-Down Tree** view. Also, for larger applications, you will likely have larger function trees to expand to find the hottest functions. In the PiSolver example, your hotspot is located in the file pi.cpp.

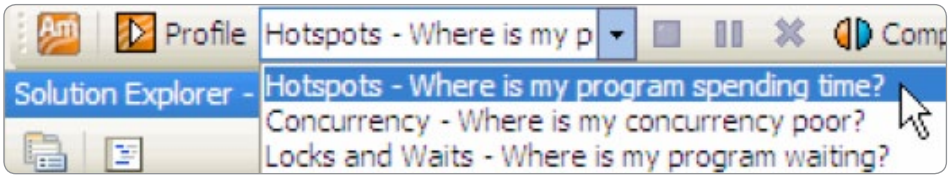
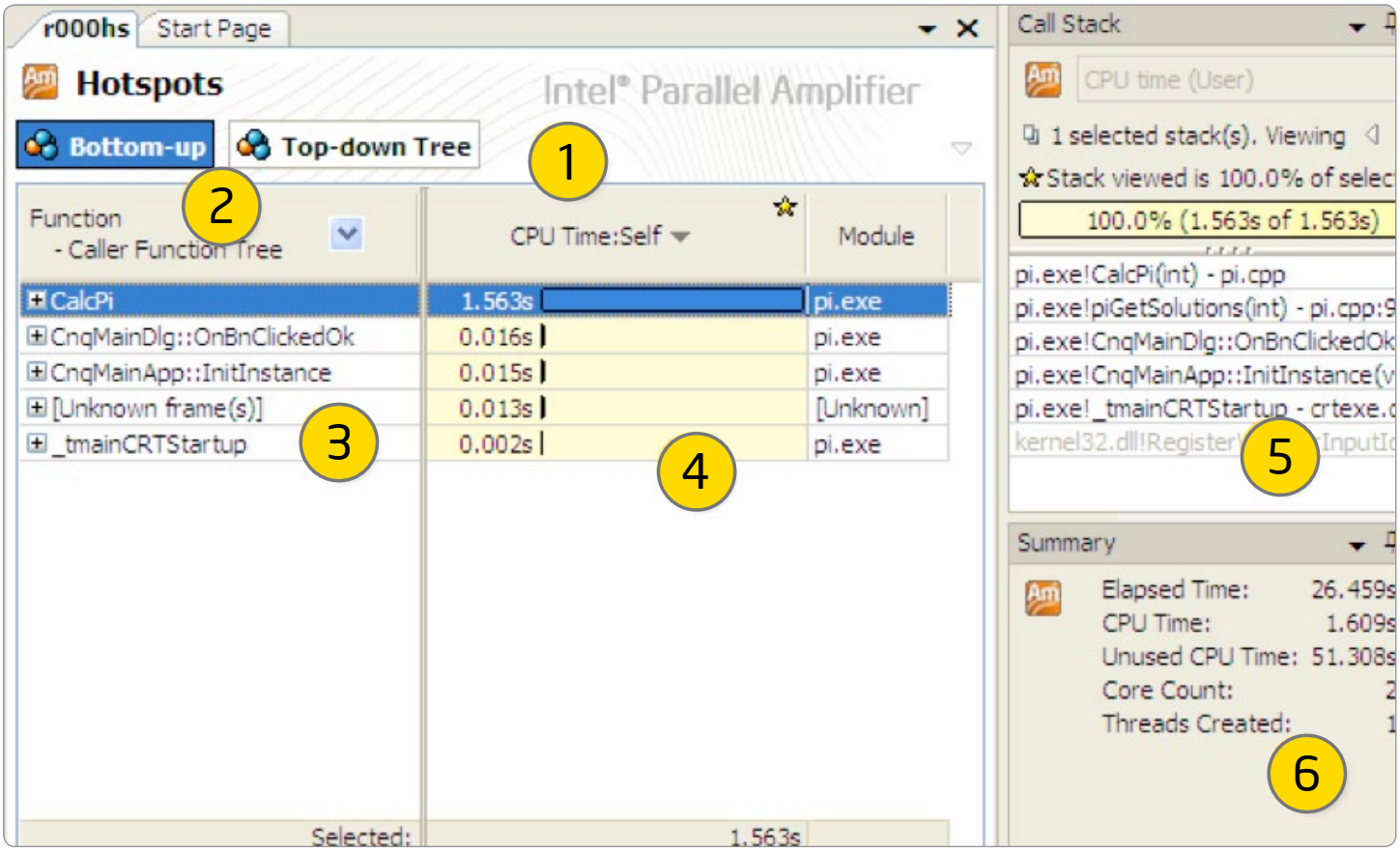


Figure 8

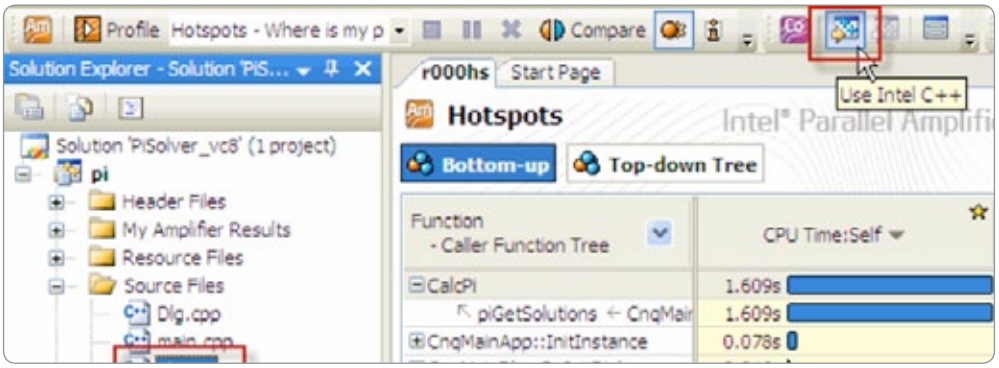
Locate a Hotspot and Optimize It

Figure 9



1. Results from hotspots analysis. Number (000) increments for each result collected.
2. Function—Bottom-up Tree is the default grouping level for hotspot data. Click the arrow button to change the grouping level.
3. Click the plus (+) sign in front of the function name to view call stacks for the selected function. Callers of the selected function are displayed, and then callers of the first caller(s), and so on.
4. CPU time is the active time taken to execute a function on a logical processor. For multiple threads, CPU time is summed up. This is the Data of Interest column for the hotspots analysis results.
5. Full stack information for the function selected in the grid. The yellow bar shows the contribution of the selected stack to the hotspot function CPU time.
6. Summary data on the analysis run: **1) Elapsed Time** is the execution time of the application from start to termination; **2) CPU Time** is the sum of CPU time for all threads; **3) Unused CPU Time** is the total time for each core when it was either waiting or not utilized by the application; **4) Core Count** is the logical CPU count for your machine; **5) Threads Created** is the number of threads created by your system during the application run.

Figure 10



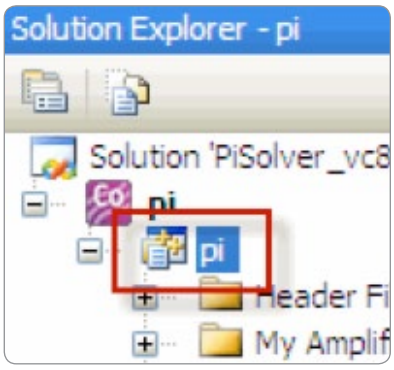


Figure 11

Note: *You can check the box entitled* ***Do not clean project(s)*** *for large projects that may take a long time to compile. However, this is not necessary for the* ***PiSolver*** *example.*

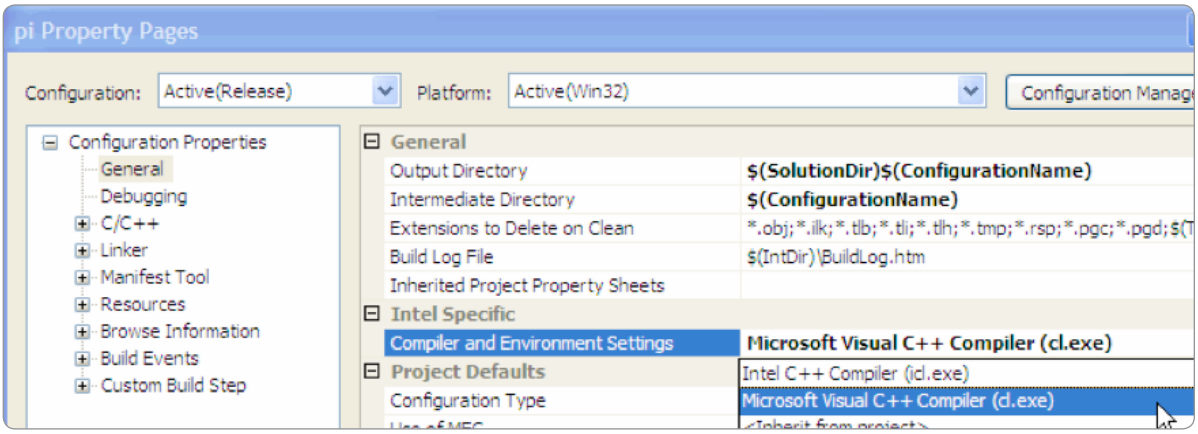


Figure 12

Step 4. Compile with the Intel C++ Compiler in Intel Parallel Composer

ESTIMATED COMPLETION TIME: 10 MINUTES

Compile the file with the hotspot using Intel C++ Compiler, which is provided by Intel Parallel Composer.

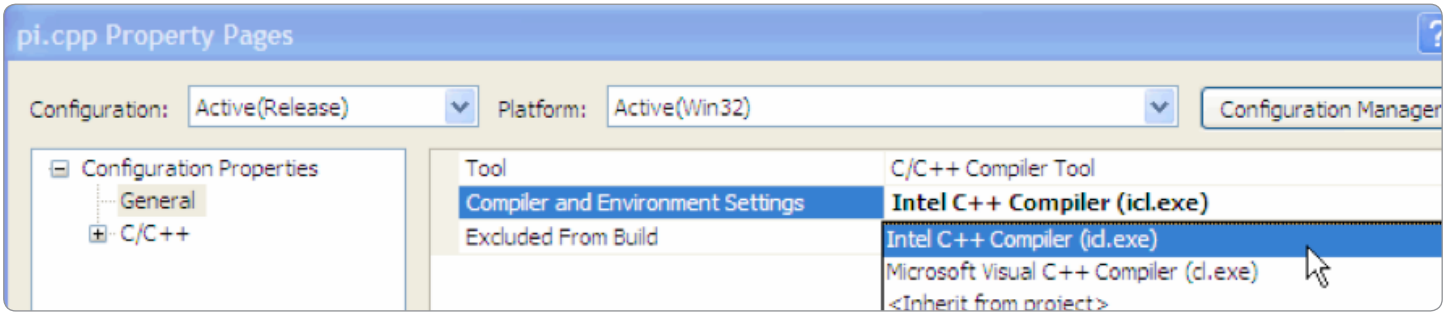
1. In the **Solution Explorer, Sources** folder, select **pi.cpp** and click the **Use Intel C++** button. **Figure 10**

2. In the Confirmation dialog box, click **OK**.

   You created a new project configuration under the Pi project. This configuration uses the Intel C++ Compiler instead of the default Microsoft Visual Studio Compiler. **Figure 11**

Locate a Hotspot and Optimize It

Figure 13



Now, you will change the settings to use the Intel C++ Compiler only for selected files and the Microsoft Visual Studio default compiler for the rest.

1. Change the project configuration to use the Microsoft C++ Compiler by selecting **Project > Properties** and then, under the **Configuration Properties > General** view, change the Compiler and Environment Settings to use the Microsoft Visual C++ Compiler (cl.exe). **Figure 12**
2. Click **Continue** in the Confirmation box. Then, click **Apply** and **OK**.

   The project is now configured as an Intel Parallel Composer project that uses the Microsoft C++ Compiler.
3. Right-click the pi.cpp file and select **Properties**, and then under the **Configuration Properties > General** view, change the Compiler and Environment Settings to use the Intel C++ Compiler (icl.exe). **Figure 13**
4. Click **OK** and **Apply**.
5. Build the project: Click the Pi project in the **Solution Explorer** Pane and go to **Build > Build Pi**.

   You will see in the **Output** pane that pi.cpp is compiled with the Intel compiler, while the others are built with the Microsoft Visual C++ Compiler.
6. Run the PiSolver application again with **Debug > Start without Debugging** and click **Calculate** in the application box.

   You should see a significant speedup in seconds compared to what you experienced compiling pi.cpp with the Microsoft Compiler.

Success

This example demonstrates how simple it is to identify and optimize a hotspot in your application and improve performance. Use Intel Parallel Amplifier together with Intel Parallel Composer to easily achieve performance improvements.

Speedup for this example, using an Intel® Core™2 Duo processor:

- Calculation time before optimization: **1453ms**
- Calculation time after optimization: **875ms**

Locate a Hotspot and Optimize It

Key Terms and Concept

Key Terms

CPU time: The CPU time is the amount of time a thread spends executing on a logical processor. For multiple threads, the CPU time of the threads is summed. The application CPU time is the sum of the CPU time of all the threads that run the application.

Target: A target is an executable file that you analyze using Intel Parallel Amplifier.

Key Concept

Hotspot analysis: Hotspot analysis helps you understand the application flow and identify sections of code that take a long time to execute (i.e., hotspots). This is where you want to focus your tuning efforts as it will have the biggest impact on overall application performance.

Intel Parallel Amplifier creates a list of functions in your application ordered by the amount of time spent in a function. It also detects the call stacks for each of these functions so you can see how the hot functions are called. It uses a low-overhead (about 5 percent), statistical-sampling algorithm that gets you the information you need without a significant slowing of application execution.

Summary

Speeding up your application may be as easy as recompiling a single file using the Intel C++ Compiler. The trick is picking the source file that contains the performance hotspot. Intel Parallel Amplifier finds the hotspot so you can focus your optimization efforts where they will be most effective.

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:

- Optimize an Existing Program by Introducing Parallelism
- Eliminate Memory Errors and Improve Program Stability

0510/BLA/CMD/PDF 323865-001US