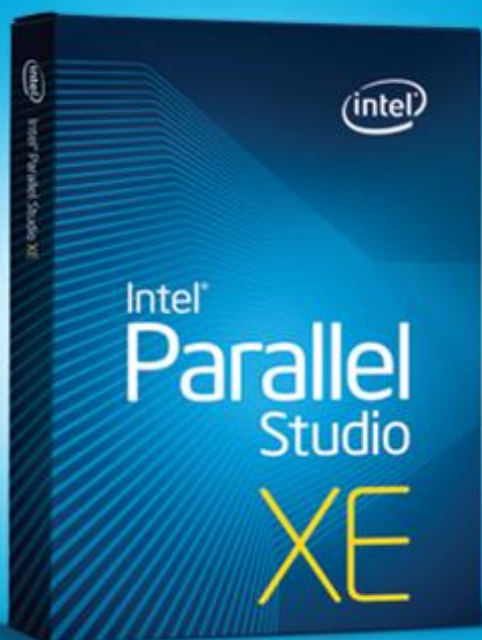




Get an Easy Performance Boost Even with Unthreaded Apps

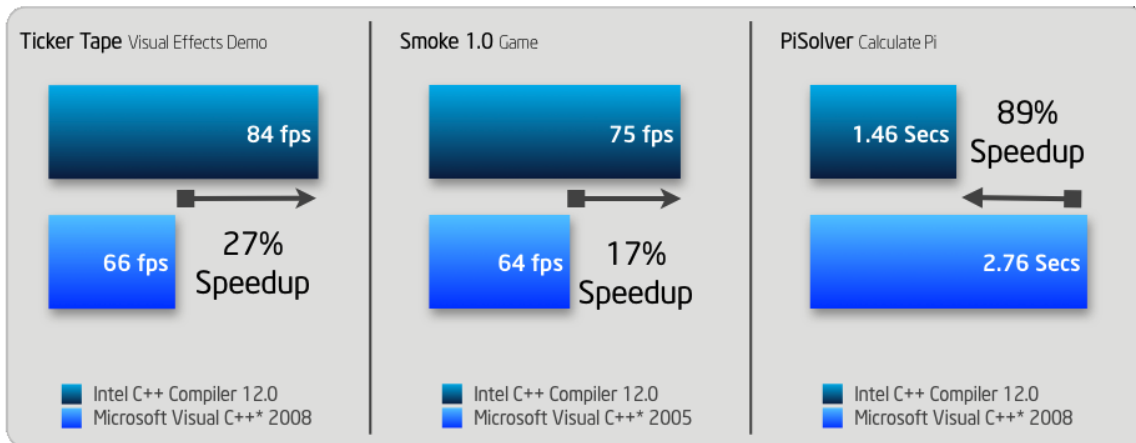
with Intel® Parallel Studio XE for Windows*





Can recompiling just one file make a difference?

Yes, in many cases it can! Often, you can achieve a major performance boost by recompiling a single file with the optimizing compiler in Intel® Parallel Studio XE. You don't always need to recompile the entire app! This holds true for both serial and parallel applications.



[application details >](#)

[Watch the movie >](#)

[Watch the movie >](#)

[application details >](#)

[details below](#)

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

** Intel® C++ Composer XE, update 1

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™ i7 processor (4 cores), 1.6 GHz, 4 GB RAM, Microsoft Windows® 7

Two Easy Steps For Better Performance

1. Find the hotspots: Measure where the application is spending time

In order to tune effectively, you 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 application is spending a lot of time. We want to find those areas and make them run fast. This is easily done using a profiling tool like Intel® VTune™ Amplifier XE. 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?" Fortunately, in many cases, we can.

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

The optimizing compiler in Intel® C++ Composer XE 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.



Try It Yourself

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 Run the Sample Application

1. Download the [PiSolver Sample.zip](#) file to your local machine.

This is an MFC dialog-based program created with Microsoft Visual Studio 2005*. A solution file is also provided for Microsoft Visual Studio 2008* and Microsoft Visual Studio 2010*. Internally, it calls a C function to solve for pi and display the result in the GUI.

2. Extract the files from the [PiSolver.zip](#) file to a writable directory or share on your system, such as in a `My Documents\Visual Studio 20xx\Intel\samples` folder.

Build the sample:

1. Build the PiSolver sample application in "Release" mode with the default Microsoft Visual C++ Compiler inside Microsoft Visual Studio.
 - a. In Microsoft Visual Studio, go to **File > Open > Project/Solution** and navigate to the `PiSolver.sln` file in the zip file directory from which you extracted it. [Figure 1](#)
 - b. Build the solution with Microsoft Visual C++ using the Release (optimized) configuration settings. Select **Build > Configuration Manager** and then, under the **Active solution configuration** drop-down box, select the **Release** setting and close the **Configuration Manager**. [Figure 2](#)
 - c. Build the solution using **Build > Build Solution**. [Figure 3](#)

Figure 1

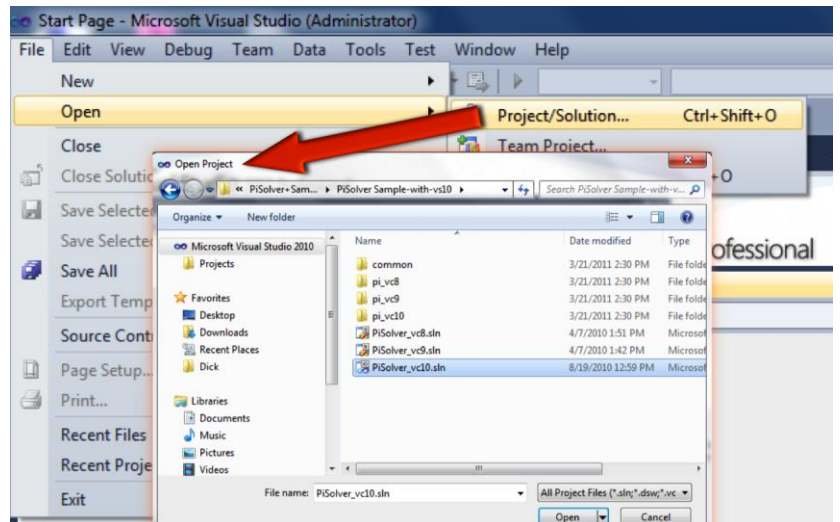


Figure 2

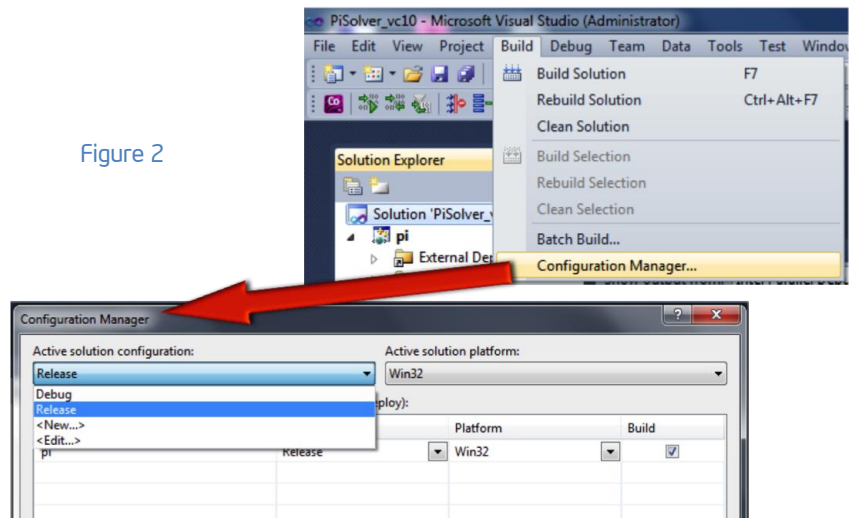
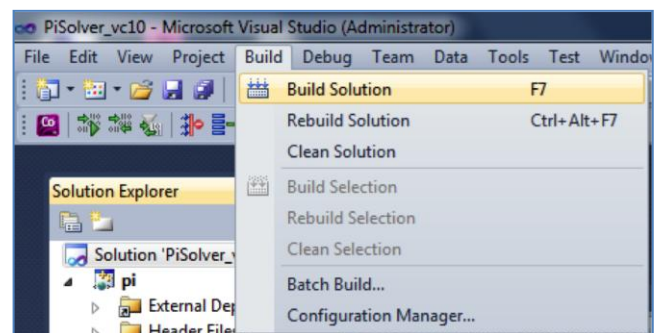


Figure 3





d. Run the application from within Microsoft Visual Studio with **Debug > Start Without Debugging**. [Figure 4](#)

e. Click on the Calculate button to compute the pi value and see the time it took in milliseconds. [Figure 5](#)

Step 3. Run Intel VTune Amplifier XE 2011: Find the Hotspots

1. Make sure that debug symbols are being generated, even in the Release (optimized) configuration. This is necessary to make sure that Intel® VTune™ Amplifier XE will provide the most useful information about the application.
 - a. Highlight the Pi project with a right-click on Pi in the Solution Explorer window.
 - b. Select **Project > Properties** to open the Pi Property Pages dialog box.
 - c. Select Configuration Properties by clicking on the triangle symbol, if it is not already expanded.
 - d. Expand C/C++ and then click on General.
 - e. Under the Debug Information Format, select **Program Database (/ZI)** and click on **Apply**. [Figure 6](#)
 - f. Now, expand the Linker properties, click on Debugging, and select **Generate Debug Info > Yes (/DEBUG)**. Click on **Apply** and **OK**. [Figure 7](#)

Figure 4

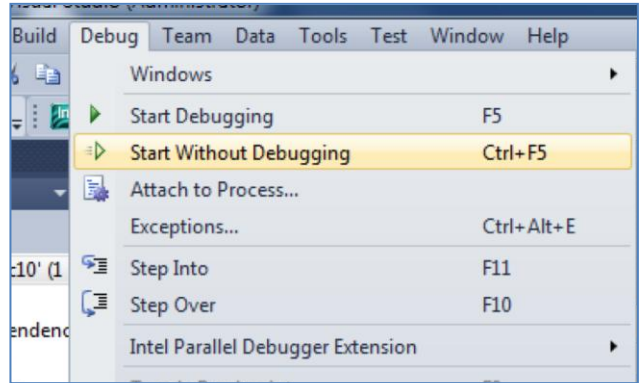


Figure 5

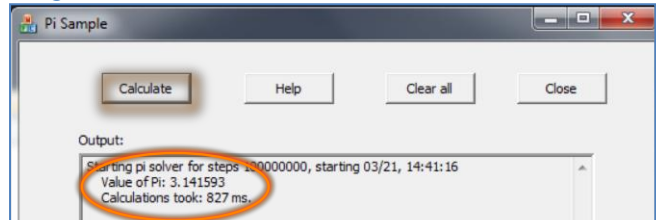


Figure 6

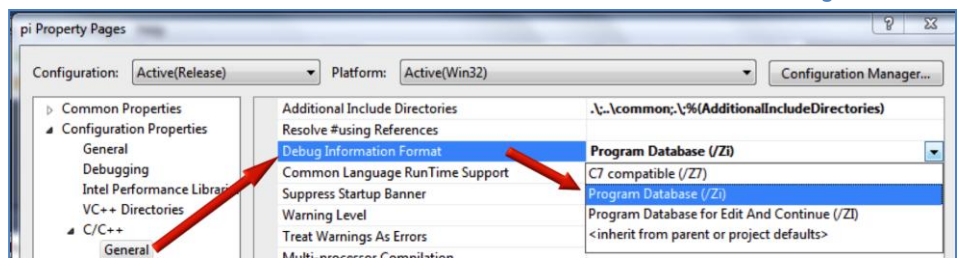
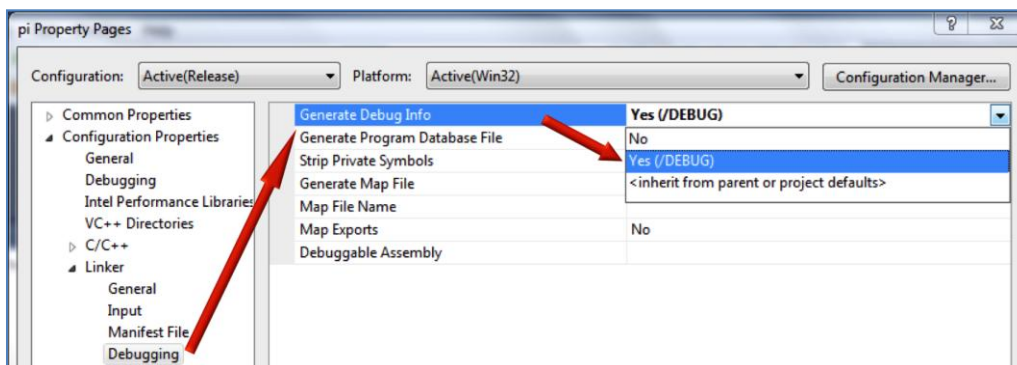


Figure 7





Get an Easy Performance Boost with Intel® Parallel Studio XE



2. Select "New Analysis" from the Intel VTune Amplifier XE toolbar and select 'Hotspots'. [Figure 8](#)
3. Click on the **Start** button. This will launch the PiSolver application.
4. In the PiSolver application, click on the Calculate button to run the calculation and then click on the Close button after you see the value and time in the dialog box. At this point, Intel VTune Amplifier XE will finish collecting the data and display a hotspot report similar to the one in [Figure 9](#). (You may see the Hotspot Analysis explanation text box covering the report; read and close this first.)
5. Click on the plus (+) sign in front of CalcPi in the Bottom-up function list to expand the call stack for that function. Then, double-click on the call stack for CalcPi with the most time (piGetSolutions) to view the source code calling the hotspot function, CalcPi.
6. 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

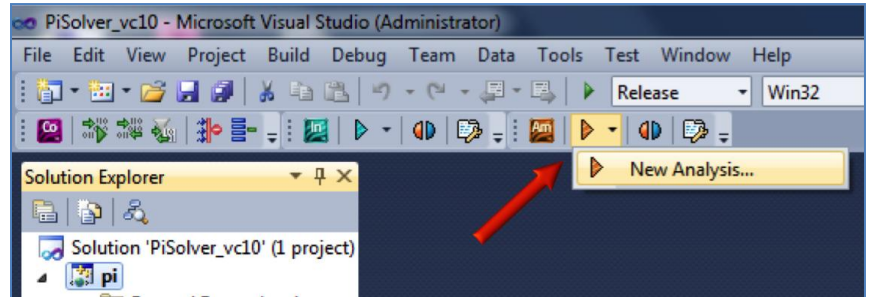
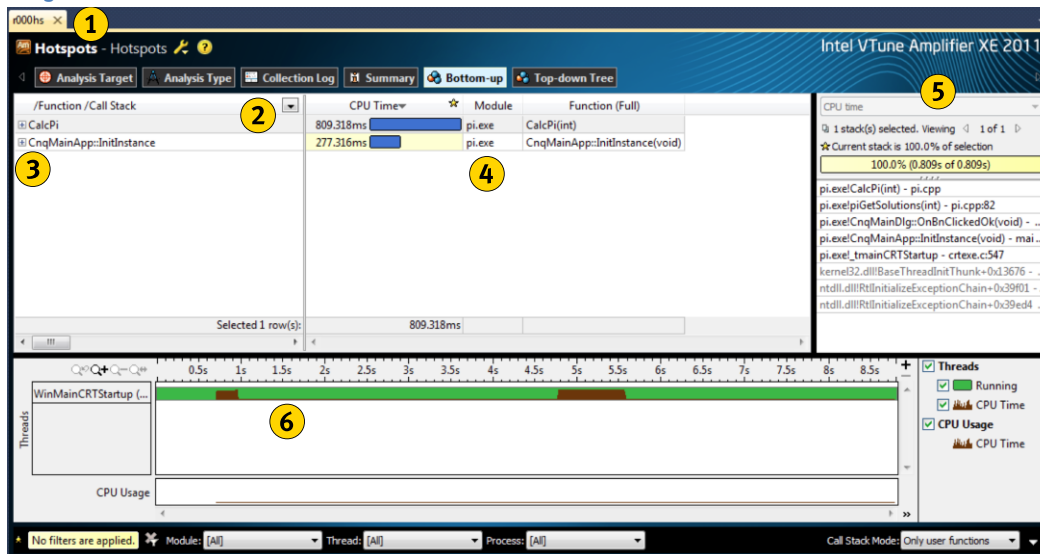


Figure 9



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 on the arrow button to change the grouping level.
3. Click on the plus (+) sign in front of the function name to view call stacks for the selected function. Callers of the selected function are displayed, followed by 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 hotspot 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. Timeline view shows CPU activity across threads over time



Step 4. Compile with Intel® C++ Compiler in Intel® C++ Composer XE

1. Look in the Solution Explorer pane of Microsoft Visual Studio and find the project in which the hotspot file(s) are located. In the PiSolver example, pi.cpp is found in the Pi project.
2. Click on Pi in the Solution Explorer pane to highlight the Pi project.
3. Select **Project > Intel C++ Composer XE 2011 > Use Intel C**
4. The Intel C++ Compiler project confirmation box will open. Click on OK.

Microsoft Visual Studio 2005/Microsoft Visual Studio 2008
 Note: For large projects that may take a long time to compile, you can check the box for "Do not clean project(s)." In the PiSolver example, this is not necessary.

You have created a new project configuration that uses the Intel C++ Compiler instead of the Microsoft compiler, but now we will change the settings to use the Intel C++ Compiler only for selected files and Microsoft's compiler for the rest.

5. Change the project configuration to use the Microsoft C++ Compiler.

Microsoft Visual Studio 2005/Microsoft Visual Studio 2008 users:

- a. Go to **Project > Properties** and then, under the **Configuration Properties > General view**, change **Compiler and Environment Settings** to use the Microsoft Visual C++ Compiler (cl.exe). **Figure 10**
- b. Say Continue to the Confirmation box. Then, click on Apply and OK. Now, you are using the Microsoft C++ compiler in the Intel Parallel Composer XE project.

Microsoft Visual Studio 2010 users:

- a. Go to **Project > Properties** and then, under the **Configuration Properties > C/C++ > General [Intel C++] view**, change Use Visual C++ Compiler to Yes. **Figure 11**
- b. Click on Apply and OK. Now, you are using the Microsoft C++ compiler in the Intel C++ Composer XE project.

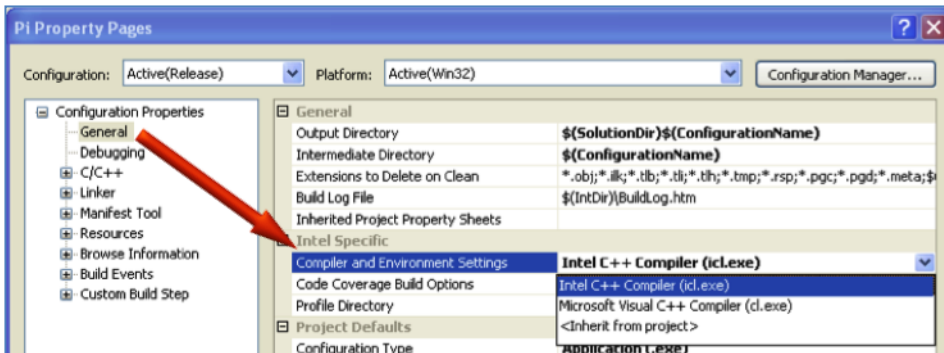
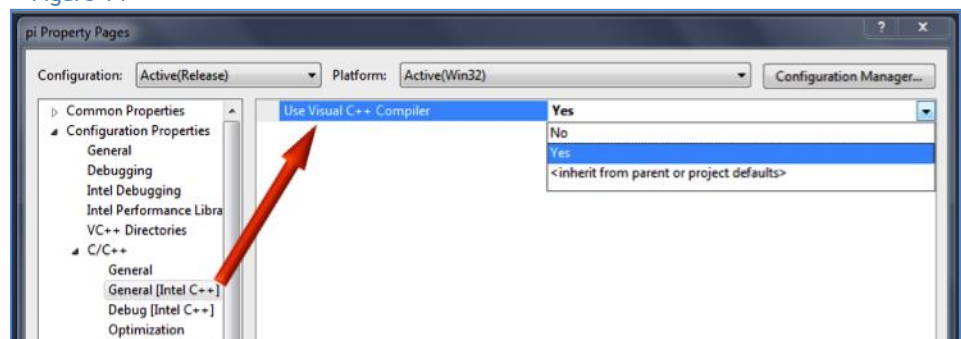


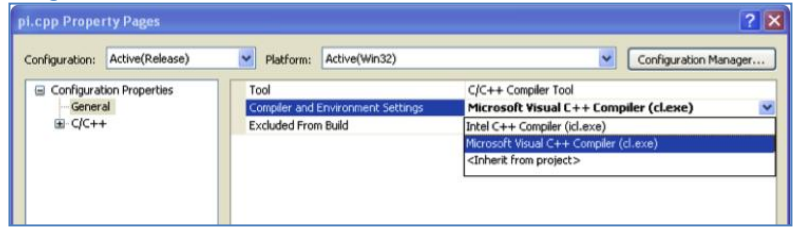
Figure 11





6. Set the Intel C++ Compiler for the `pi.cpp` file.
 - a. Microsoft Visual Studio 2005/Microsoft Visual Studio 2008 users: Right click on the `pi.cpp` file and select **Properties**, and then under the **Configuration Properties > General view**, change **Compiler and Environment Settings** to use the Intel C++ Compiler (`icl.exe`). [Figure 12](#)

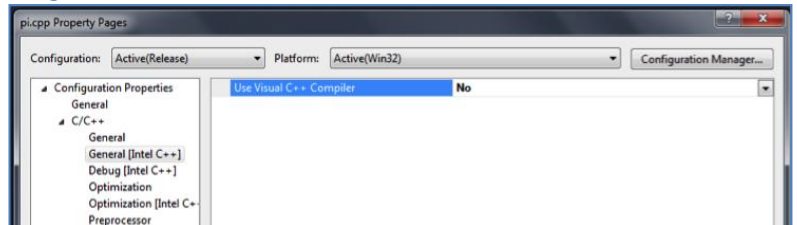
Figure 12



Click on **Apply** and **OK**.

- b. Microsoft Visual Studio 2010 users: Right click on the `pi.cpp` file and select **Properties**, and then, under the **Configuration Properties > C/C++ > General [Intel C++] view**, change **Use Visual C++ Compiler** to **No**. [Figure 13](#)

Figure 13



Click **Apply** and **OK**.

Note: Visual Studio 2010 also has the feature that you can use **Ctrl + Left-Click** and select multiple files to build with Intel C++ Compiler.

7. Build the project by clicking the **Pi** project to highlight it, and then use **Build**. You will see in the Output pane that `pi.cpp` gets compiled with the Intel Compiler, and the others are built with the Microsoft compiler.
8. Run the PiSolver application again with **Debug > Start Without Debugging** and click on **Calculate** in the application box. You should see a significant speedup in seconds versus what you experienced compiling `pi.cpp` with the Microsoft compiler.

Success!

On our test system, PiSolver ran 96 percent faster just by recompiling using Intel C++ Composer XE.

Application	PiSolver
Before ¹	0.827 seconds
After ²	0.421 seconds
Speedup	96%

System Specifications: Intel® Core™ i7 processor (4 cores), 1.6 GHz, 4 GB RAM, Microsoft® Windows 7
¹Microsoft Visual Studio® 2010
²Intel® C++ Composer XE 2011, update 1

In this example, we improved performance just by recompiling with Intel C++ Composer XE's optimizing compiler. Often, this is enough to get a significant performance gain, even with non-threaded applications.

In other cases, Intel VTune Amplifier XE may show that you are spending a lot of time in a slow library function. If you find yourself in this situation, an easy way to speed up your app is to replace the slow function with a fast one.

Fortunately, in addition to an optimizing compiler, Intel C++ Composer XE 2011 includes Intel® Integrated Performance Primitives (Intel® IPP). Intel IPP is an extensive library of multicore-ready, highly optimized software functions for digital media and data-processing applications. Intel IPP offers thousands of optimized functions covering frequently used fundamental algorithms.



Step 5. Use Intel VTune Amplifier XE and Compare Results

1. Click on the **New Analysis** button on the Intel VTune Amplifier XE toolbar again and select **Quick Hotspot Analysis**.
2. Look at the overall time of the application in the Summary tab; you should see that the CPU time is reduced. Also, look at the call tree. piGetSolutions now stands out on its own as other functions are taking more relative time. In a larger application, after reducing the application time for one hotspot, you might uncover another hotspot that you should optimize. [Figure 14](#)
3. Another way to compare results is to use Intel VTune Amplifier XE's "Compare Results" function. This allows you to do a side-by-side comparison of previous runs of the application to see what the changes have been. To do this, click on the **Compare results** button on the Intel VTune Amplifier XE toolbar. You will see a comparison like the one shown in [Figure 15](#). The detailed report shows the changes function by function, showing how the time in CalcPi has been drastically reduced.

Figure 14

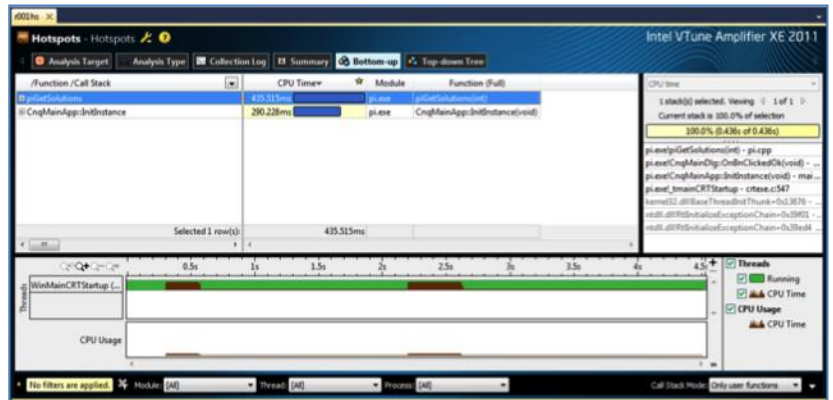
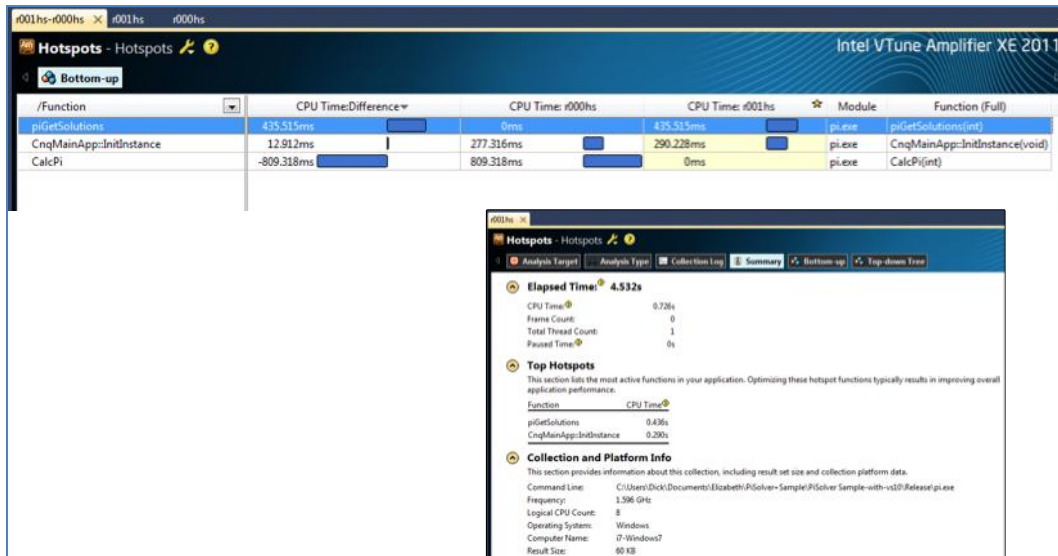


Figure 15





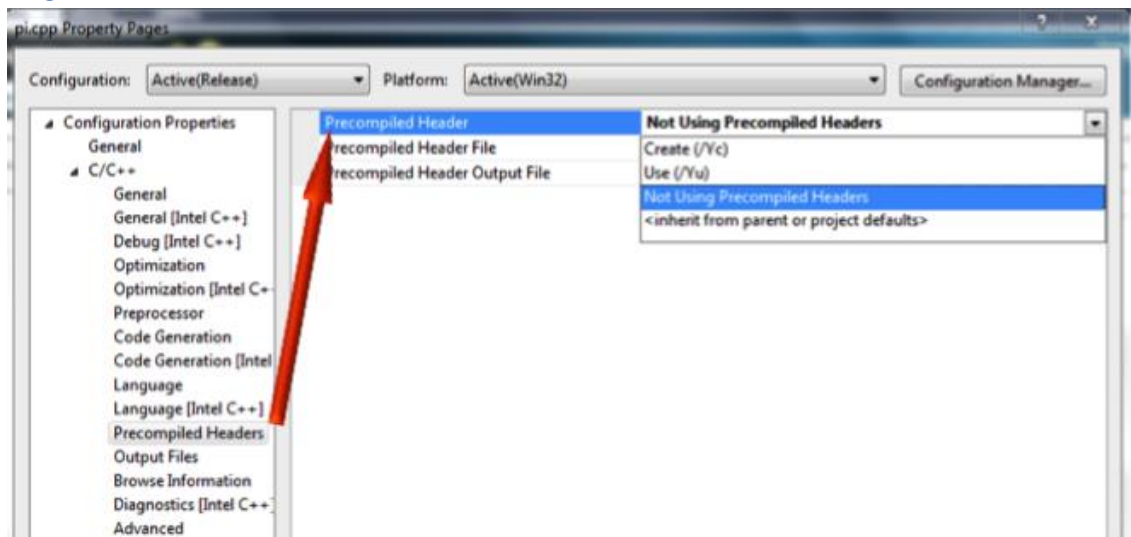
Tips for Larger, More Complex Applications

The PiSolver sample application is small, but it demonstrates how to quickly find and recompile a hotspot. In larger applications with multiple projects, there may be many more functions showing significant time taken in the hotspot profile. In such cases, it might be easier (and perhaps more fruitful for performance) to simply rebuild the whole project in which the hotspot is found, rather than address just the one (or two) files. Below are some tips for rebuilding the hotspots:

- > Consider Whole-Program Optimization (Link-time Code Generation (/GL) in Microsoft Visual Studio; Inter-procedural Optimization (/Qipo) in Intel Compilers). This optimization can greatly improve application performance for some applications through cross-file inlining and other cross-file/function optimizations. It is enabled by default when you create a "Release" configuration in both compilers. However, it requires that the compiler that performed the optimization must also perform the link step. Thus, when you recompile only one file with the Intel Compiler, as we did in the PiSolver sample, you might not get the full range of benefits of whole-program optimization from the Intel Compiler. This is exactly what we did for the Smoke application results seen the table on page 1; Click on the video link. Smoke is a very large application with lots of projects, and the hotspot project was fairly small, so it was a very fast rebuild showing a very good performance increase

- > For larger applications that have many projects inside a solution, like the Smoke example, it may be easier to rebuild the whole project in which the hotspot file(s) are located. This is easily done by just switching to the Intel C++ Composer XE configuration for the whole project and rebuilding the project instead. In this case, just skip steps 5 and 6 in the "Compile with Intel® C++ Compiler in Intel® C++ Composer XE" instructions above.
- > Another thing to watch for is that in many applications, precompiled headers (i.e., preprocessed .h files that are saved for future use) are used to speed up compilation. Precompiled headers built by the Microsoft compiler are not usable with the Intel Compilers, so they must be either rebuilt or not used. If you are using the Intel Compiler for a whole project, this is not an issue—the precompiled headers will be built with the Intel Compiler. If, however, you are only rebuilding a single file, you may need to do the following for the files you are compiling with the Intel Compiler:
 - > Right click on the file you want to compile with the Intel Compiler, say pi.cpp, and select Properties. In the Property Page box, select Configuration Properties > C/C++ > Precompiled Headers > Create/ Use Precompiled Header > Not Using Precompiled Headers. [Figure 16](#)

Figure 16





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 VTune Amplifier XE.

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 effort because it will have the biggest impact on overall application performance.

Intel VTune Amplifier XE 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 VTune Amplifier XE finds the hotspot so you can focus your optimization efforts where they will be most effective.

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.

[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