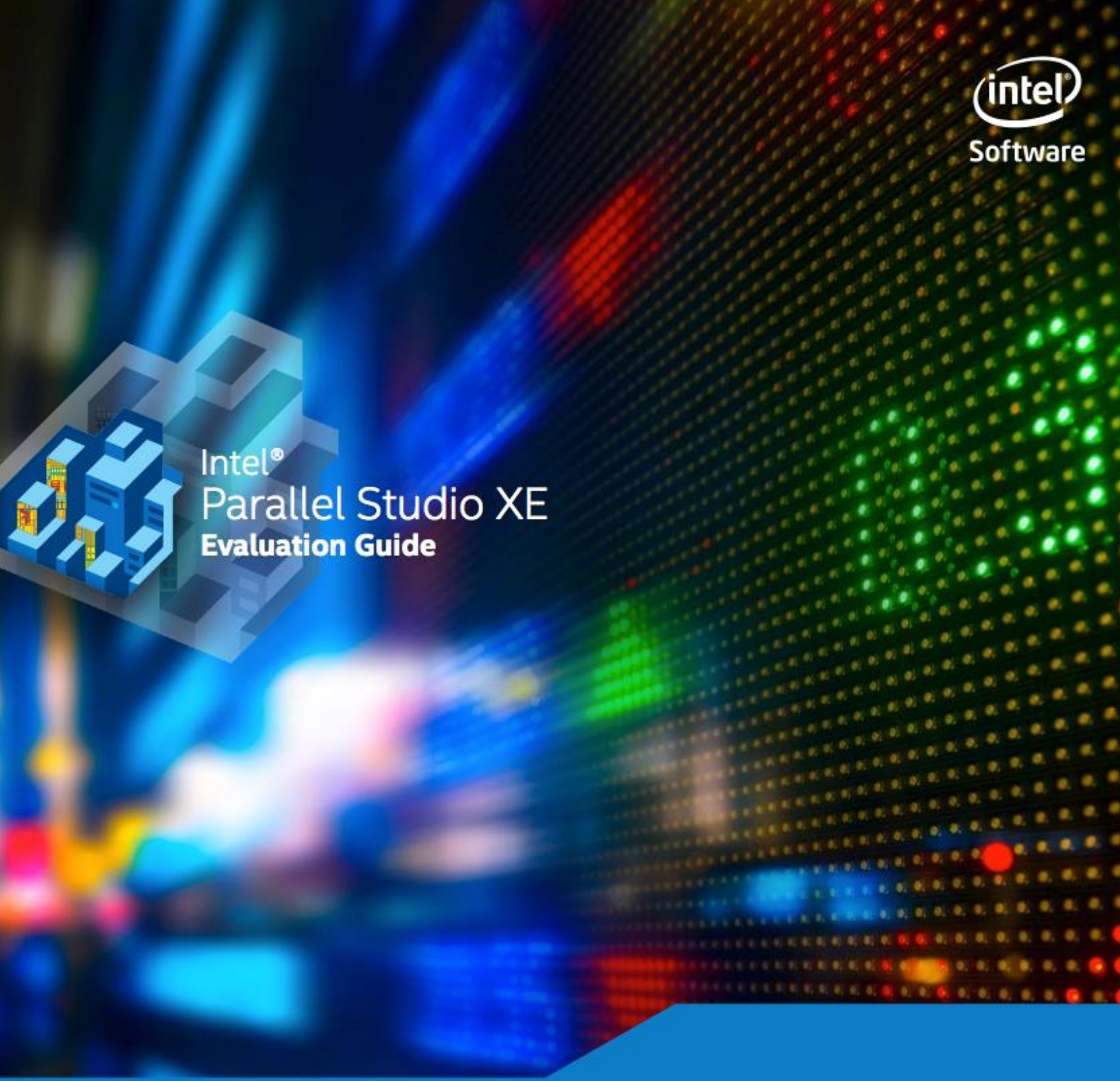


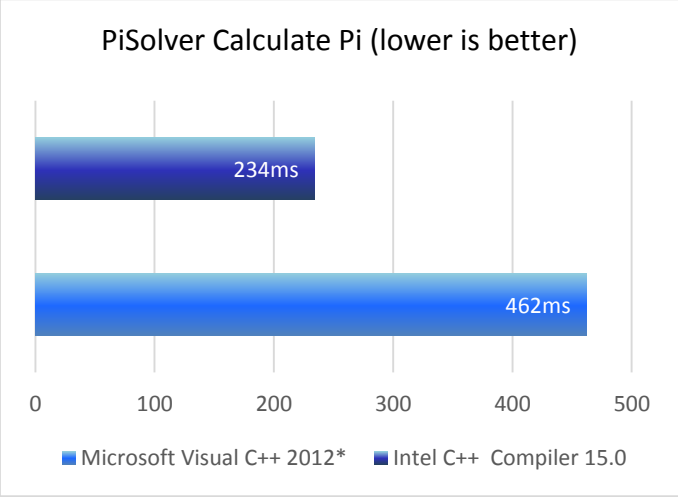

# Get an Easy Performance Boost Even with Unthreaded Applications

## Introduction

This guide will illustrate how you use Intel® Parallel Studio XE to find the "hotspots" (areas that are taking a lot of time) in your application and then recompiling those parts to improve overall performance of the application.

## 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.




System Specifications: Intel® Core™ i5-3550 processor, 3.3 GHz, 4 cores, 4GB RAM, Microsoft Windows* Server 2008 R2 Enterprise x64, service pack 1

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation. Optimization Notice: at end of document

## 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 application 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 faster. 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 Intel® C++ Compiler 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

### Install and Set Up Intel Parallel Studio XE

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

## Step 2. Install and Run the Sample Application

1. Download the PiSolver+Sample.zip file to your local machine.
2. This is an MFC dialog-based program created with Microsoft Visual Studio*. A solution file is provided for Microsoft Visual Studio 2010*, 2012 and 2013**. Internally, it calls a C function to solve for pi and display the result in the GUI.
3. Extract the files from the PiSolver+Sample.zip file to a writable directory or share on your system, such as in <My Documents>\Intel Parallel Studio XE\samples.

### Build the sample:

- Build the PiSolver sample application in “Release” mode with the default Microsoft Visual C++ Compiler inside Microsoft Visual Studio.
- 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

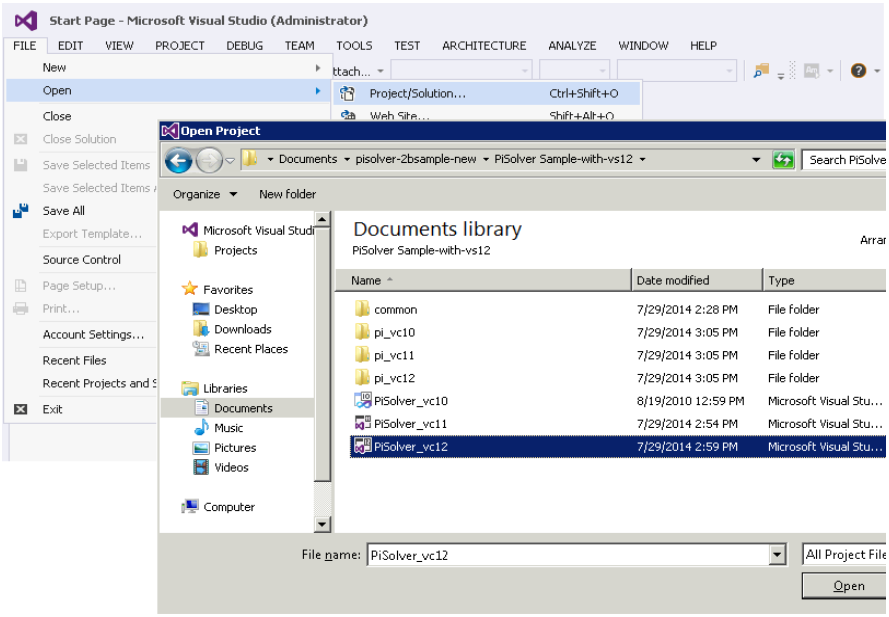



*Figure 1*

- Build the solution with Microsoft Visual C++ using the Release (optimized) configuration settings. Select the Configuration drop down list and then, select the Release setting. Figure 2

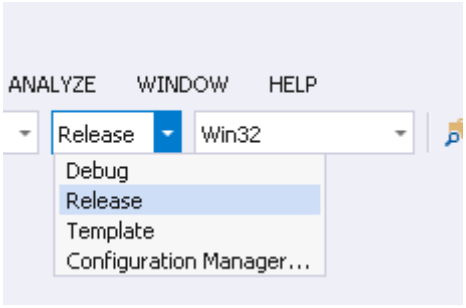



*Figure 2*

- Build the solution using Build > Build Solution. Figure 3

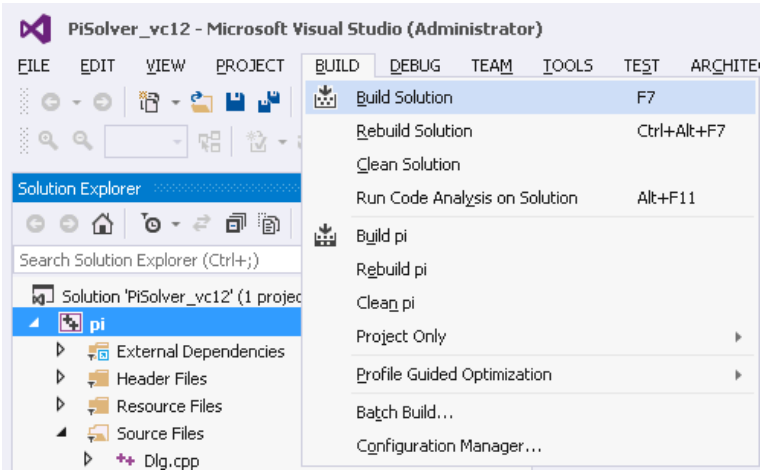



*Figure 3*

- Run the application from within Microsoft Visual Studio with Debug > Start Without Debugging. Figure 4

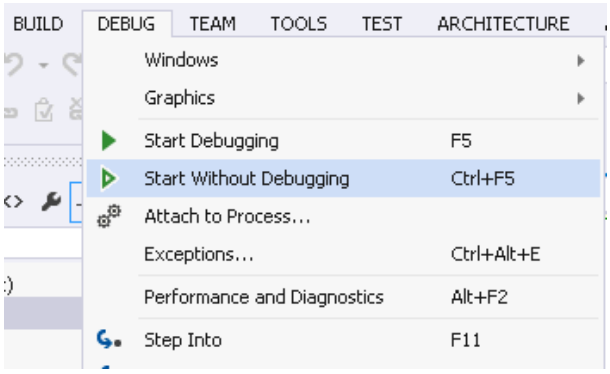



*Figure 4*

- Click on the Calculate button to compute the pi value and see the time it took in milliseconds. Figure 5

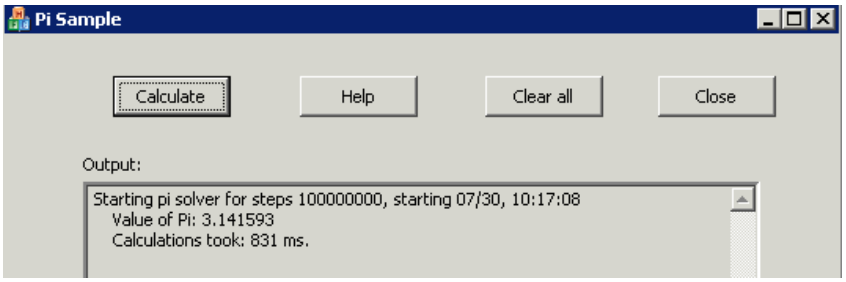



*Figure 5*

## Step 3. Run Intel® VTune Amplifier XE: 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.
   - Highlight the **pi** project with a right-click on **pi** in the Solution Explorer window.
   - Select Project > Properties to open the pi Property Pages dialog box.
   - Expand the Configuration Properties by clicking on the triangle symbol, if it is not already expanded.
   - Expand C/C++ and then click on General.
   - Under the Debug Information Format, select Program Database (/Zi) and click on Apply. Figure 6

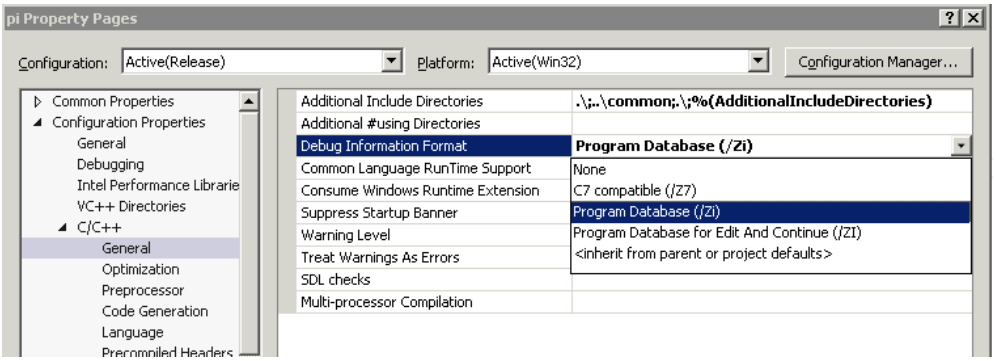



*Figure 6*

- Now, expand the Linker properties, click on Debugging, and select Generate Debug Info > Yes (/DEBUG). Click on Apply and OK. Figure 7

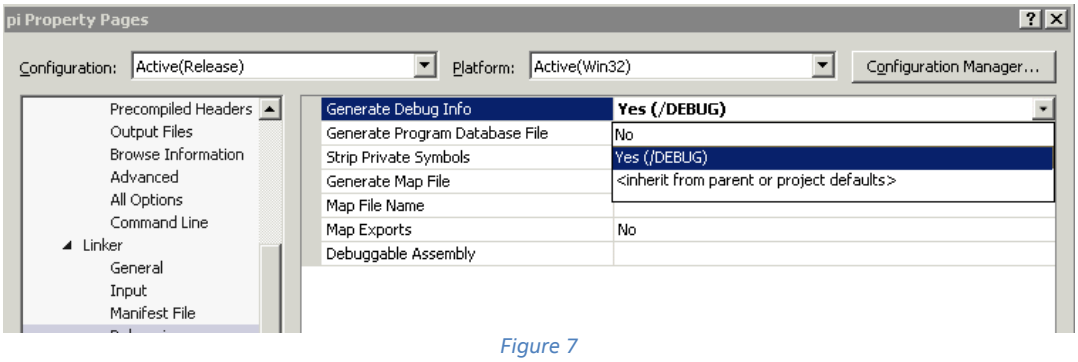



*Figure 7*

2. Select "New Analysis" from the Intel VTune Amplifier XE toolbar Figure 8 and select '**Hotspots'**. Figure 9

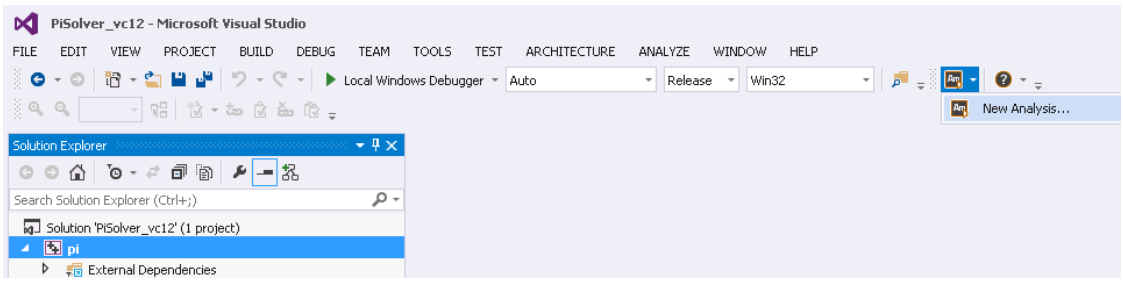



*Figure 8*

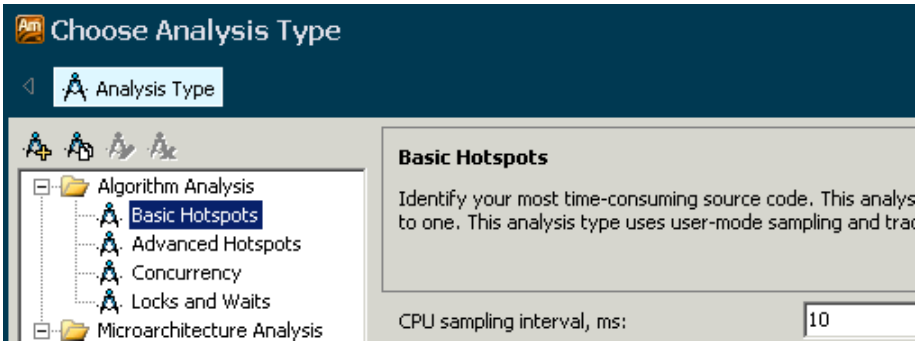



*Figure 9*

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 Summary report similar to the one in Figure 10. (You may see the Hotspot Analysis explanation text box covering the report; read and close this box, first.)

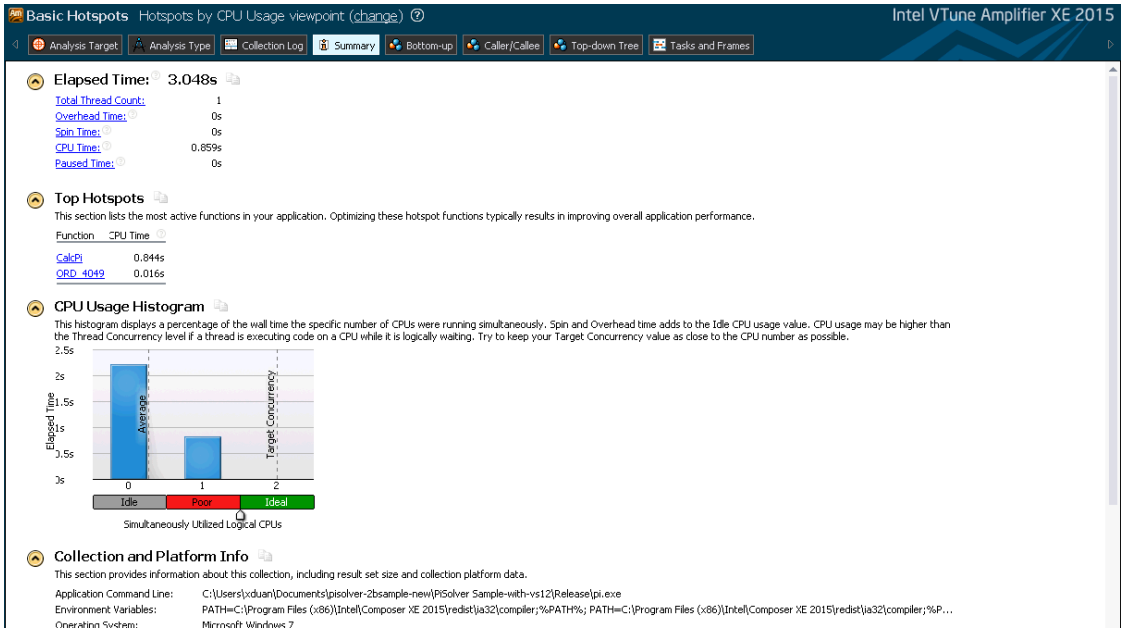



*Figure 10*

5. Click on the “Bottom-up” label to view all the hotspots. Figure 11

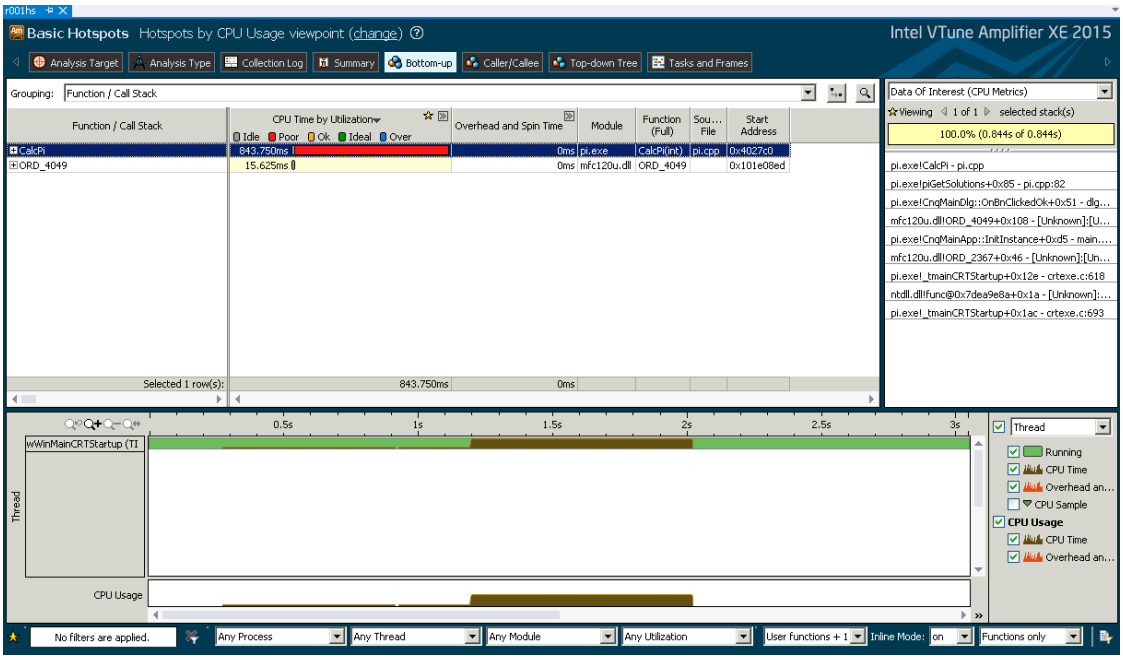



*Figure 11*

1. Results from hotspots analysis. Number (000) increments for each result collected.
2. Function / Call stack is the default grouping level for hotspot data. Click on the arrow button to change the grouping.
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 for each level expanded.
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

6. 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 function CalcPi to view the source code. Note the name of the source file. Figure 12

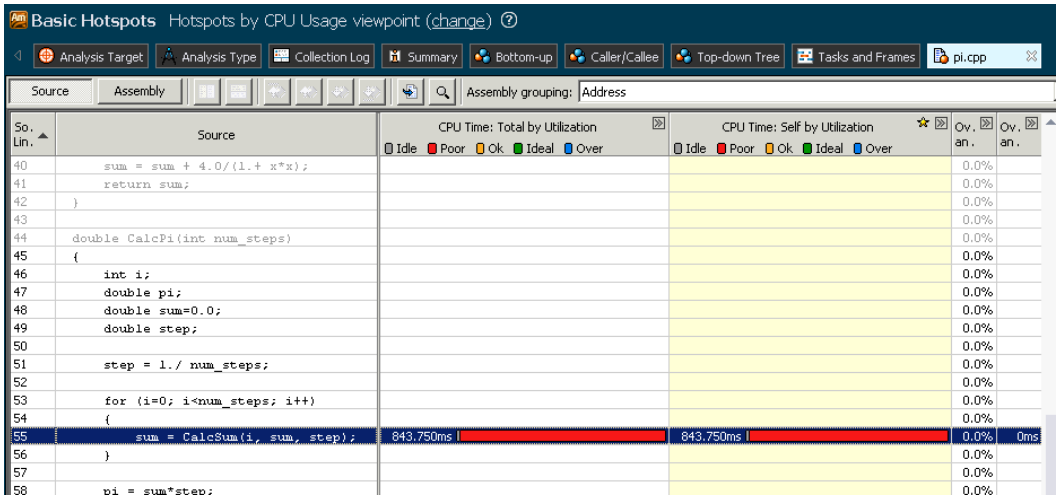



*Figure 12*

7. 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.

## Step 4. Compile with Intel® C++ Compiler

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 Compiler XE 2015 > Use Intel C++
4. The Intel C++ Compiler project confirmation box will open. Click on OK.
5. Change the project configuration to use the Microsoft C++ Compiler.

### Microsoft Visual Studio 2010 users:

- Go to Project > Properties and then, under the Configuration Properties > C/C++ > General [Intel C++] view, change Use Visual C++ Compiler to Yes. Figure 13

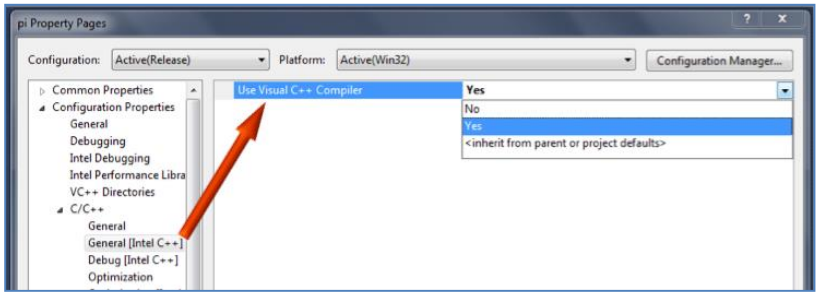



*Figure 13*

- Click on Apply and OK. Now, you are using the Microsoft C++ compiler in the project.

### Microsoft Visual Studio 2012 or 2013 users:

- Go to Project > Properties and then, under the Configuration Properties > General view, change Platform Toolset to Visual Studio 2012 (v110) or Visual Studio 2013 (v120). Figure 14

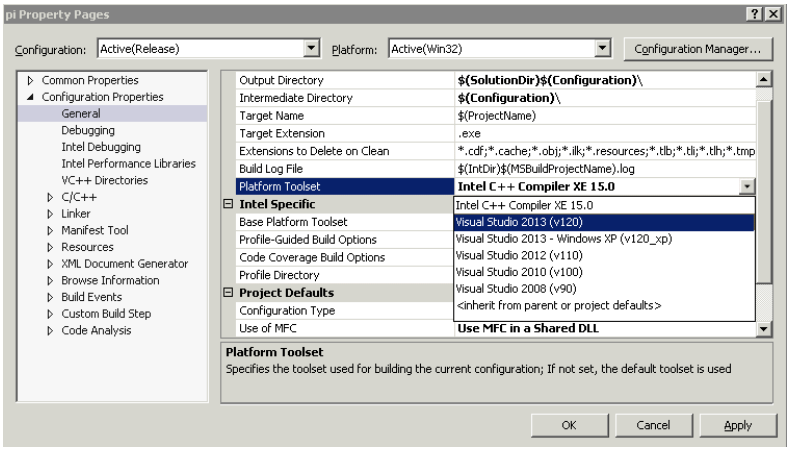



*Figure 14*

6. Click on Apply and OK. Now, you are using the Microsoft C++ compiler in the project.Set the Intel C++ Compiler for the **pi.cpp** file.

**Microsoft Visual Studio 2010 users:**

Expand the **pi** project in the Solution Explorer and then expand the Source Files. 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 15

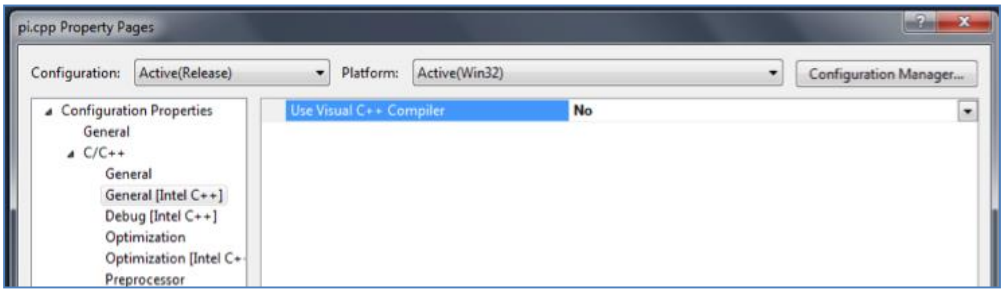



*Figure 15*

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

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

1. 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 other files are built with the Microsoft compiler.
2. 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.

**Microsoft Visual Studio 2012 or 2013 users:**

Expand the **pi** project in the Solution Explorer and then expand the Source Files. Right click on the pi.cpp file and select Intel Compiler XE 2015 -> Use Intel C++. Figure 16

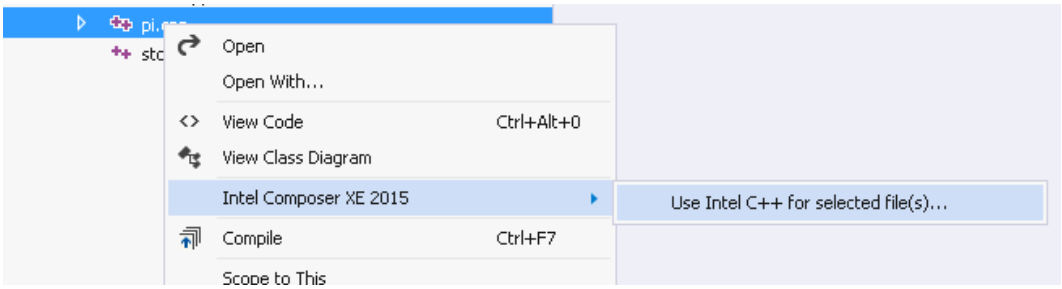



*Figure 16*

Click **OK**.

Note: Visual Studio 2012 and 2013 also has the feature that allows you to use Ctrl + Left-Click to select multiple files to Build with the Intel C++ Compiler.

1. 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 other files are built with the Microsoft compiler.
2. 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 102 percent faster just by recompiling using Intel C++ Compiler XE.

| Application | PiSolver |
|---|---|
| Before[1] | 0.462 seconds |
| After[2] | 0.234seconds |
| Speedup | 1.97x |

System Specifications: Intel® Core™ i5-3550 processor, 3.3 GHz, 4 cores, 4GB RAM, Microsoft Windows* Server 2008 R2 Enterprise x64, service pack

In this example, we improved performance just by recompiling with Intel C++ Compiler 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 Parallel Studio XE for C++ includes libraries and programming models to make it easier to write efficient code that takes advantage of multicore and vectorization.

- Intel® Integrated Performance Primitives - includes functions for multimedia, data processing, and communications applications. Intel IPP offers thousands of optimized functions covering frequently used fundamental algorithms,
- Intel® Math Kernel Library – math routines for applications that require maximum performance. Core math functions include BLAS, LAPACK, ScaLAPACK1, sparse solvers, fast Fourier transforms, vector math, and more.
- Intel® Threading Building Blocks - award-winning C++ template library for creating reliable, portable, and scalable parallel applications.
- Intel® Cilk™ Plus - an extension to C and C++ that provides a simple yet surprisingly powerful model for parallel programming, while runtime and template libraries offer a well-tuned environment for building parallel applications

### 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 Hotspot analysis. After pressing Start, the application will be re-run. Press Calculate, again, and then Close after the calculation completes.
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 17

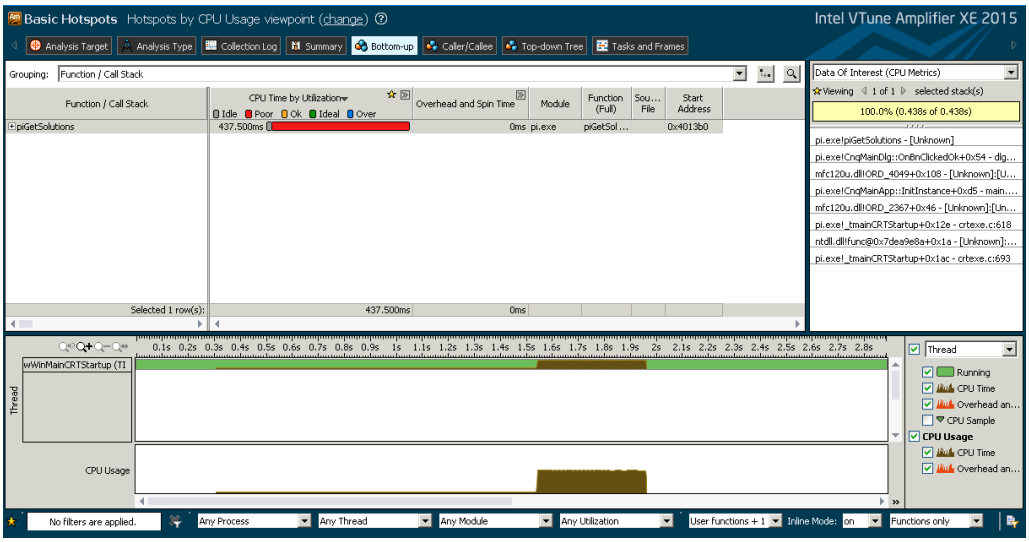



*Figure 17*

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 changes occurred. To do this, select the two result files in the solution explorer, right click and select Compare. You will see a comparison like the one shown in Figure 18. The detailed report shows the changes function by function and how the time in CalcPi has been drastically reduced.

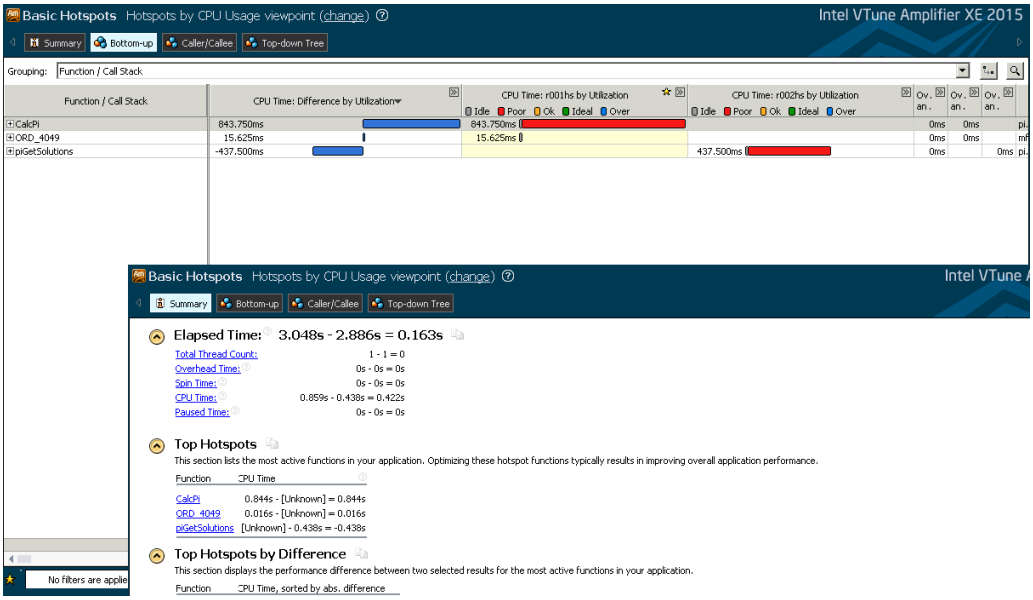



*Figure 18*

## 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 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 in 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++ Compiler 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" 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 19

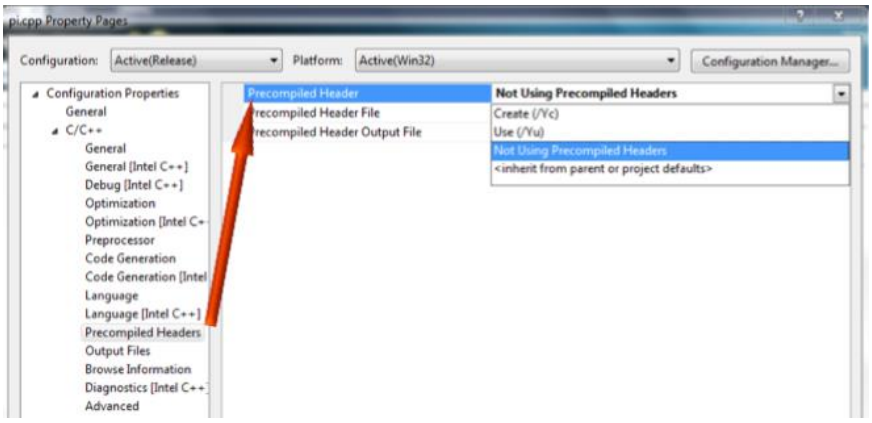



*Figure 99*

## 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.

## 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.

## 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

## 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*)<br>Linux* (GNU)<br>OS X*[4] (XCode*) | Windows (Visual Studio)<br>Linux (GNU) | Windows (Visual Studio)<br>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



boost-performance-studioxe-evalguide/Rev-082014