Intel® Parallel Studio

Product Brief

Intel® Parallel Studio

Parallelism for Your Development Lifecycle

Intel® Parallel Studio brings comprehensive parallelism to C/C++ Microsoft Visual Studio* application development. Parallel Studio was created in direct response to the concerns of software industry leaders and developers. From the way the products work together to support the development lifecycle to their unique feature sets, parallelism is now easier and more viable than ever before. The tools are designed so those new to parallelism can learn as they go, and experienced parallel programmers can work more efficiently and with more confidence. Parallel Studio is interoperable with common parallel programming libraries and API standards, such as Intel® Threading Building Blocks (Intel® TBB) and OpenMP*, and provides an immediate opportunity to realize the benefits of multicore platforms.

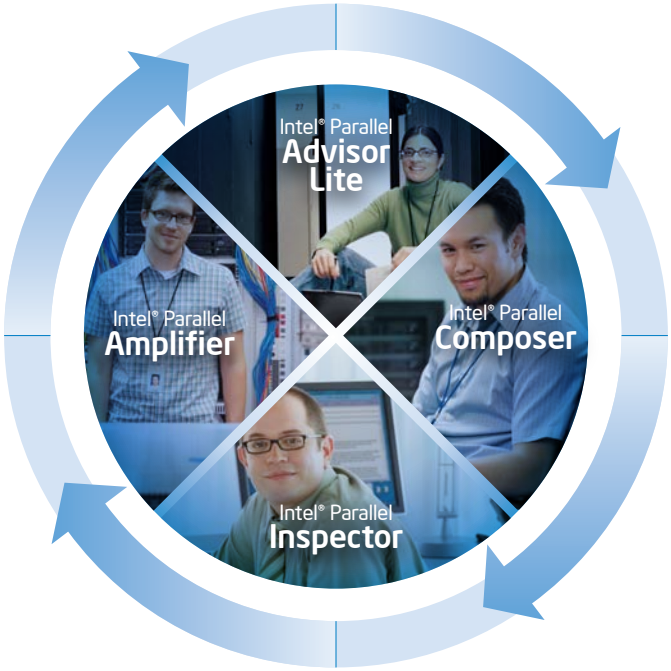
"Intel® Parallel Studio makes the new Envivio 4Caster Series Transcoder's development faster and more efficient. The tools included in Intel Parallel Studio, such as Intel® Parallel Inspector, Intel® Parallel Amplifier, and Intel® Parallel Composer (which consists of the Intel® C++ Compiler, Intel® IPP, and Intel® TBB) shortens our overall software development time by increasing the code's reliability and its performance in a multicore multithreaded environment. At the qualification stage, the number of dysfunctions is reduced due to a safer implementation, and the bug tracking becomes easier too. Intel Parallel Studio globally speeds up our software products' time-to-market."

Eric Rosier
V.P. Engineering
Envivio

Intel® Parallel Studio Tools

Intel® Parallel Studio Workflow

The workflow diagram below depicts a typical usage model across all of the tools in Parallel Studio. If you are just starting to add parallelism to your application, finding hotpsots would be a great first step. If you have already added some parallelism or if your application has been optimized, you could start by verifying error free code or by tuning.



- Introduce threads, compile, and debug with **Intel® Parallel Composer**
- Find threading and memory errors with **Intel® Parallel Inspector**
- Tune with **Intel® Parallel Amplifier**
- Identify areas that can benefit from parallelism with **Intel® Parallel Advisor Lite**

To fully utilize the power of Intel® multicore processors and achieve maximum application performance on multicore architectures, you must effectively use threads to partition software workloads. When adding threads to your code to create an efficient parallel application, you will typically encounter the following questions:

a. What programming model and specific threading techniques are most appropriate for your application?

b. How do you detect and fix threading and memory errors, which are hard to reproduce because the threaded software runs in a non-deterministic manner, where the execution sequence depends on the run?

c. How can you actually boost performance of your threaded application on multicore processors and make the performance scale with additional cores?

Intel Parallel Studio Addresses the Issues Listed Above.

The following list shows the Intel Parallel Studio tools and provides a brief description of how they address the above issues.

Co	Intel® Parallel Composer	Intel® C++ compiler, Intel® Threading Building Blocks, Intel® Integrated Performance Primitives, and Intel® Parallel Debugger Extension	Addresses issues A and B
In	Intel® Parallel Inspector	A multithreading tool to detect challenging threading and memory errors	Addresses issue B
Am	Intel® Parallel Amplifier	A performance analysis and tuning tool for parallel applications to optimize performance on multiple cores	Addresses issue C
	In addition, a product enhancement is available at whatif.intel.com.		
Ad	Intel® Parallel Advisor Lite	Identifies candidate functions for parallelizing, and advises on protecting or sharing data.	Identifies areas that can most benefit from parallelism

Intel® Parallel Composer

A comprehensive set of Intel® C++ compilers, libraries, and debugging capabilities for developers bringing parallelism to their Windows*-based client applications. It integrates with Microsoft Visual Studio* 2005 and 2008, is compatible with Visual C++*, and supports the way developers work, protecting IDE investments while delivering an unprecedented breadth of parallelism development capabilities, including parallel debugging. Parallel Composer is a stand-alone product or can be purchased as part of Intel® Parallel Studio, which includes Intel® Parallel Inspector to analyze threading and memory errors, and Intel® Parallel Amplifier for performance analysis of parallel applications. Parallel Composer also includes:

- Intel C++ Compilers for 32-bit processors and a cross-compiler to create 64-bit applications on 32-bit systems
- Intel Threading Building Blocks (Intel TBB), an award winning C++ template library that abstracts threads to tasks to create reliable, portable, and scalable parallel applications. It can also be used with Visual C++.
- Intel® Integrated Performance Primitives (Intel® IPP), an extensive library of multicore-ready, highly optimized software functions for multimedia, data processing, and communications applications. Intel IPP includes both hand-optimized primitive-level functions and high-level threaded solutions such as codecs. It can be used for both Visual C++ and .NET development.
- Intel® Parallel Debugger Extension, which integrates with the Microsoft Visual Studio debugger

Intel® Parallel Inspector

Combines threading and memory error checking into one powerful error checking tool. It helps increase the reliability, security, and accuracy of C/C++ applications from within Microsoft Visual Studio. Intel Parallel Inspector uses dynamic instrumentation that requires no special test builds or compilers, so it's easier to test code more often.

- Find memory and threading errors with one easy-to-use tool
- Help ensure that shipped applications run error-free on customer systems
- Give both experts and novices greater insight into parallel code behavior
- Find latent bugs within the increasing complexity of parallel programs
- Reduce support costs and increase productivity

Intel® Parallel Amplifier

Makes it simple to quickly find multicore performance bottlenecks without needing to know the processor architecture or assembly code. Intel Parallel Amplifier takes away the guesswork and analyzes performance behavior in Windows* applications, providing quick access to scaling information for faster and improved decision making.

Fine-tune for optimal performance, ensuring cores are fully exploited and new capabilities are supported.

- Make significant performance gains that impact customer satisfaction
- Increase application headroom for richer feature sets and next-gen innovation
- Find performance problems quickly and easily
- Scale applications for multicore

Intel® Parallel Advisor Lite

In conjunction with the Intel Parallel Studio toolset, Intel Parallel Advisor Lite (a technology preview prototype) is available at http://whatif.intel.com.

For many developers, especially those just starting parallelism, the first step is to identify candidate loops or sections in an application that can benefit from parallelism. These are typically the most time-consuming algorithms in applications and often the hardest to find. Once found, these hotspots only address part of the process in parallel application development. Identifying data objects that need to be made private or shared is an important step to achieving optimal results. Intel Parallel Advisor Lite enables you to quickly and easily find hotspots and recommend objects to parallelize, saving valuable development time and resources.

After installing Intel Parallel Studio, download and try out Intel Parallel Advisor Lite at: http://whatif.intel.com.

Intel Parallel Composer

Intel C++ Compiler: Microsoft Visual Studio Integration, Microsoft Visual C++* Compatibility, and Support for Numerous Parallel Programming APIs (Application Programming Interfaces)

All features in Intel Parallel Studio are seamlessly integrated into Microsoft Visual Studio 2005 and 2008.

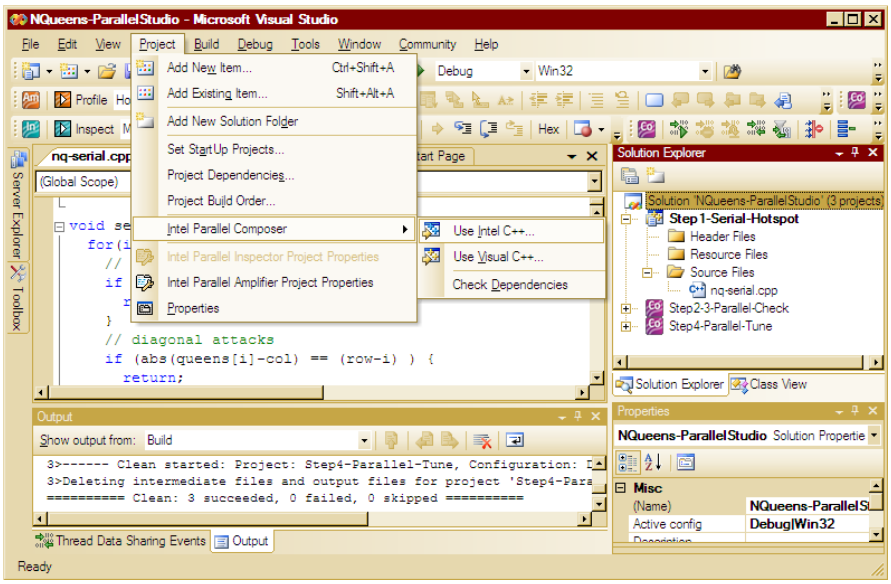


Figure 2. Intel® Parallel Composer integrates into Visual Studio*. The solution on display shows how to switch to the Intel® C++* compiler. You can easily switch to Visual C++* from the Project menu or by right-clicking over the solution or project name.

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is an award-winning C++ template library that abstracts threads to tasks to create reliable, portable, and scalable parallel applications. Included in Parallel Composer, Intel TBB can be used with the Intel C++ Compiler or with Microsoft Visual C++.

Intel TBB solves *three key challenges* for parallel programming

- **Productivity**: Simplifies the implementation of parallelism
- **Correctness**: Helps eliminate parallel synchronization issues
- **Maintenance**: Aids in the creation of applications ready for tomorrow, not just today

Advantages of using Intel TBB:

- **Future-proof applications**: As the number of cores (and threads) increase, application speedup increases using Intel TBB's sophisticated task scheduler
- **Portability**: Implement parallelism once to execute threaded code on multiple platforms.
- **Interoperability**: Commitment to work with a variety of threading methods, hardware, and operating systems
- **Active open source community**: Intel TBB is also available in an open source version. www.threadingbuildingblocks.org is an active site with forums, blogs, code samples, and much more.

Intel TBB offers comprehensive, abstracted templates, containers, and classes for parallelism.

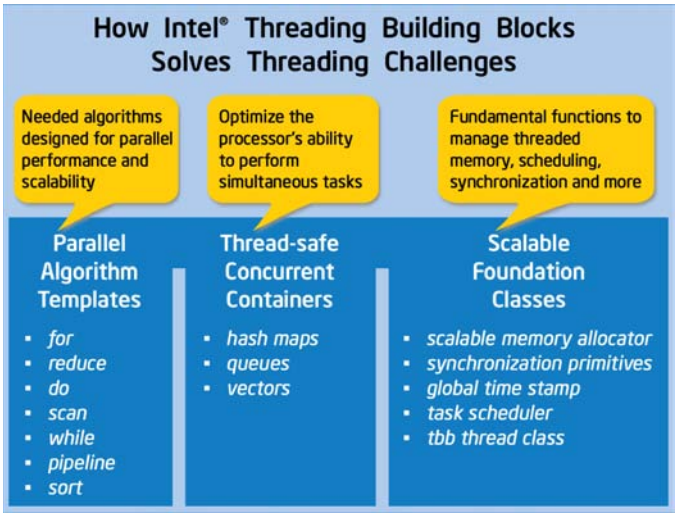


Figure 3: Major function groups within Intel® TBB

Intel TBB can be used to solve a wide variety of threading tasks. Figure 4 presents three common parallelism problems addressed by Intel TBB.

Problem	Solution
How to add parallelism easily	**Intel TBB parallel_for command** ▪ Straightforward replacement of for/next loops to get advantages of parallelism ▪ Load-balanced parallel execution of fixed number of independent loop iterations
Management of threads to get best scalability	**Intel TBB Task Scheduler** ▪ Manages thread pool and hides complexity of native threads ▪ Designed to address common performance issues of parallel programming - Oversubscription: One scheduler thread per hardware thread - High overhead: Programmer specifies tasks, not threads - Load imbalance: Work-stealing balances load
Memory allocation is a bottleneck in concurrent environment	Intel TBB provides tested, tuned, and scalable memory allocator based on per-thread memory management algorithm ▪ As an allocator argument to STL template classes ▪ As a replacement for malloc/realloc/free calls (C programs) ▪ As a replacement for global new and delete operators (C++ programs)

Figure 4: Intel® TBB addresses three major parallelism challenges

Support for Lambda Functions

The Intel Compiler is the first C++ compiler to implement lambda functions in support of the working draft of the next C++ standard C++0x. A lambda construct is almost the same as a function object in C++ or a function pointer in C. Together with closures, they represent a powerful concept because they combine code with a scope. For example, if you have a C++ application that uses iterators for loops, Intel TBB, with lambda support, will help implement templated loop patterns.

The source code in Figure 5 below is an example of a function object created by a lambda expression. Tighter C++ and Intel TBB integration allows the simplification of the functor operator() concept by using lambda functions and closures to pass code as parameters.

```
void ParallelApplyFoo(float a[], size_t n ) {
  parallel_for( blocked_range<size_t>( 0, n ),
    [=](const blocked_range<size_t>& range) {
    for( int i= range.begin(); i!=range.end();
++i )
      Foo(a[i]);
    },
    auto_partitioner() );
}
```

Figure 5: Source code example of a lambda function

OpenMP*

OpenMP is an industry standard for portable multithreaded application development. It is effective at fine-grain (loop-level) and coarse-grain (function-level) threading. OpenMP provides an easy and powerful way to convert serial applications into parallel applications, enabling potentially big performance gains from parallel execution on multicore and symmetric multiprocessor systems.

When an application that has been written and built using OpenMP is run on a system with just one processor, the results are the same as unmodified source code. Stated differently, the results you get are the same as you would get from unmodified, serial-execution code. This makes it easier for you to make incremental code changes, while maintaining serial consistency. Because only directives are inserted into the code, it is possible to make incremental code changes and still maintain a common code-base for your software as it runs on systems that still have only one processor.

OpenMP is a single source code solution that supports multiple platforms and operating systems. There is also no need to "hard-code" the number of cores into your application because the OpenMP runtime chooses the right number for you.

OpenMP 3.0 Task Queuing

Sometimes programs with irregular patterns of dynamic data or complicated control structures, like recursion, are hard to parallelize efficiently. The work queuing model in OpenMP 3.0 allows you to exploit irregular parallelism, beyond that possible with OpenMP 2.0 or 2.5.

The task pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. When a task pragma is encountered, the code inside the task block is conceptually queued into the queue associated with the task. To preserve sequential semantics, there is an implicit barrier at the completion of the task. The developer is responsible for ensuring that no dependencies exist or that dependencies are appropriately synchronized, either between the task blocks, or between code in a task block and code in the task block outside of the task blocks. An example is presented below in Figure 6.

```
#pragma omp parallel
#pragma omp single
{
  for(int i=0; i<size; i++) {
    // try all positions in first row
    // create separate array for each recursion
    // started here
#pragma omp task
    setQueen(new int[size], 0, i);
    }
}
```

Figure 6. Example of OpenMP 3.0* task queuing

In the example in Figure 6 above, we need only one task queue. Therefore we need to set up the queue by invoking only one thread (omp single). The setQueen calls are independent of each other and therefore they fit nicely into the task concept. You might want to also read about the Intel Parallel Debugger Extension, which makes it easy to inspect the state of tasks, teams, locks, barriers, or taskwaits in your OpenMP program in dedicated windows.

Simple Concurrency Functions

Parallel Composer offers four new keywords to help make parallel programming with OpenMP easier: __taskcomplete, __task, __par, and __critical. In order for your application to benefit from the parallelism made possible by these keywords, you specify the /Qopenmp compiler option and then recompile, which links in the appropriate runtime support libraries, which manage the actual degree of parallelism. These new keywords use the OpenMP 3.0 runtime library to deliver the parallelism, but free you from actually expressing it with OpenMP pragma and directive syntax. This keeps your code more naturally written in C or C++.

The keywords mentioned above are used as statement prefixes. For example, we can parallelize the function, solve(), using __par. Assuming that there is no overlap among the arguments, the solve() function is modified with the addition of the __par keyword. With no change to the way the function is called, the computation is parallelized. An example is presented in Figure 7.

```
void solve() {
    __par for(int i=0; i<size; i++) {
    // try all positions in first row
    // create separate array for each
recursion
    // started here
    setQueen(new int[size], 0, i);
    }
}
```

Figure 7. Example of __par, one of four simple concurrency functions, new in the Intel® C++ Compiler in Intel® Parallel Studio

Intel Integrated Performance Primitives (Intel IPP)

Parallel Composer includes the Intel Integrated Performance Primitives. Intel IPP is an extensive library of multicore-ready, highly optimized software functions for multimedia, data processing, and communications applications. It offers thousands of optimized functions covering frequently used fundamental algorithms in video coding, signal processing, audio coding, image processing, speech coding, JPEG coding, speech recognition, computer vision, data compression, image color conversion, cryptography, string processing/regular expressions, and vector/matrix mathematics.

Intel IPP includes both hand-optimized primitive level functions and high-level threaded samples, such as codecs, and can be used for both Visual C++ and .NET development. All of these functions and samples are fully thread-safe, and many are internally threaded, to help you get the most out of today's multicore processors and scale to future manycore processors.

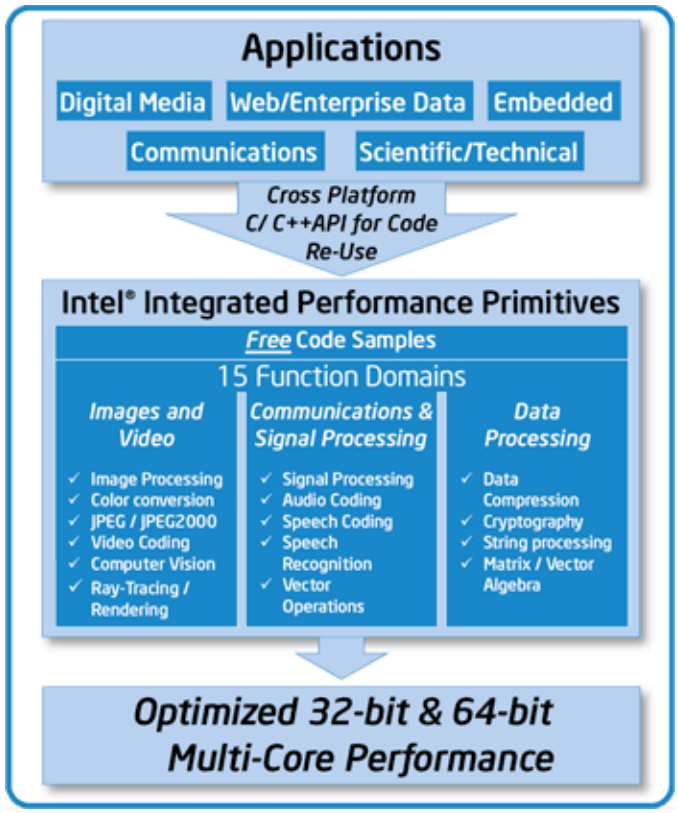


Figure 8. Intel® Integrated Performance Primitives is included in Intel® Parallel Composer, a part of Intel® Parallel Studio, and features threaded and thread-safe library functions over a wide variety of domains

Intel IPP Performance

Depending on the application and workload, Intel IPP functions can perform many times faster than the equivalent compiled C code. In the image resize example below, the same operation that required 338 microseconds to execute in compiled C++ code required only 111 microseconds when Intel IPP image processing functions were used. That is a 300 percent performance improvement.

In this image resizing example, Intel® IPP code ran 3x **faster** than compiled C++ code

Figure 9. In this image resizing example (from 256 x 256 bits to 460 x 332 bits), the Intel® IPP-powered application ran in 111 msec vs. 338 msec for compiled C++ code (system configuration: Intel® Xeon®, 2.9 GHz, 2 processors, 4 cores/processor, 2 threads/processor).

Using Intel IPP in Visual Studio

It's easy to add Intel IPP support to a Microsoft Visual Studio project. Parallel Composer includes menus and dialog boxes to add Intel IPP library names and paths to a Visual Studio project. Simply click on the project name in the Solution Explorer, opt for the Intel Build Components Selection menu item, and use the Build Components dialog to add Intel IPP. Then just add Intel IPP code to your project including the header and functional code. You'll notice that the Build Selection dialog automatically adds the library names to the linker for IPP and adds a path to the Intel IPP libraries.

In addition to C++ projects, Intel IPP can also be used in C# projects using the included wrapper classes to support calls from C# to Intel IPP functions in the string processing, image processing, signal processing, color conversion, cryptography, data compression, JPEG, matrix, and vector math domains.

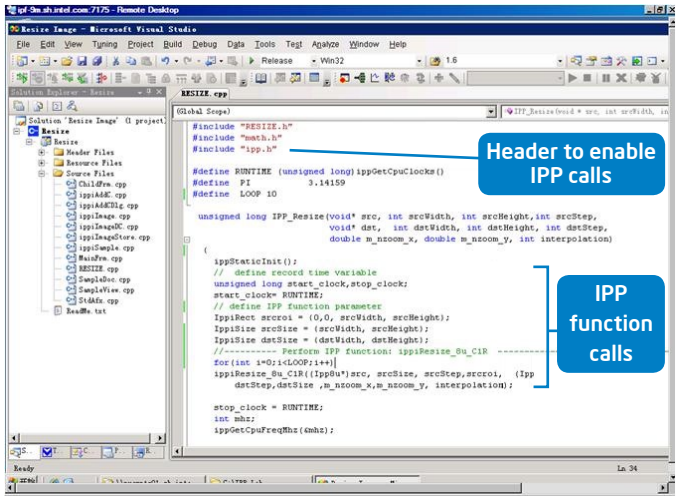


Figure 10: It's easy to incorporate Intel® IPP library calls into your Visual Studio* code

Optimize Embarrassingly Parallel Loops

Algorithms that display data parallelism with iteration independence lend themselves to loops that exhibit "embarrassingly parallel" code. Parallel Composer supports three techniques to maximize the performance of such loops with minimal effort: auto-vectorization, use of Intel-optimized valarray containers, and auto-parallelization. Parallel Composer can automatically detect loops that lend themselves to auto-vectorization. This includes explicit *for* loops with static or dynamic arrays, vector and valarray containers, or user-defined C++ classes with explicit loops. As a special case, implicit valarray loops can either be auto-vectorized or directed to invoke optimized Intel Integrated Performance Primitives (IPP) library primitives. Auto-vectorization and use of optimized valarray headers optimize the performance of your application to take full advantage of processors that support the Streaming SIMD Extensions.

In a moment, we'll look at how to enable Intel optimized valarray headers. But first, let's look at Figure 11 which shows an example of an explicit valarray, vector loops, and an implicit valarray loop.

```
valarray<float> vf(size), vfr(size);
vector<float> vecf(size), vecfr(size);


//log function, vector, explicit loop
for (int j = 0; j < size-1; j++) {
  vecfr[j]=log(vecf[j]);

}

//log function, valarray, explicit loop
for (int j = 0; j < size-1; j++) {
  vfr[j]=log(vf[j]);
}

//log function, valarray, implicit loop
vfr=log(vf);
```

Figure 11: The source code above shows examples of explicit valarray, vector loops, and an implicit valarray loop

To use optimized valarray headers, you need to specify the use of Intel Integrated Performance Primitives as a Build Component Selection and set a command line option. To do this, first load your project into Visual Studio and bring up the project properties pop-up window. In the "Additional Options" box, simply add "/Quse-intel-optimized-headers" and click "OK." Figure 12 presents a picture of how to do this.

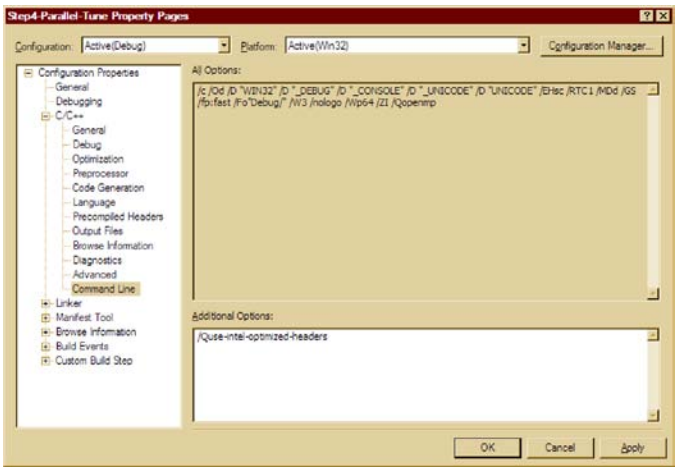


Figure 12: Adding the command to use optimized header files to a command line in Visual C++*

Next, from the Project menu, select Intel Parallel Composer, then Select Build Components. In the resulting pop-up box, check "Use IPP" click "OK." Figure 13 presents a picture of this. With this done, you can rebuild your application and check it for performance and behavior as you would when you make any change to your application.

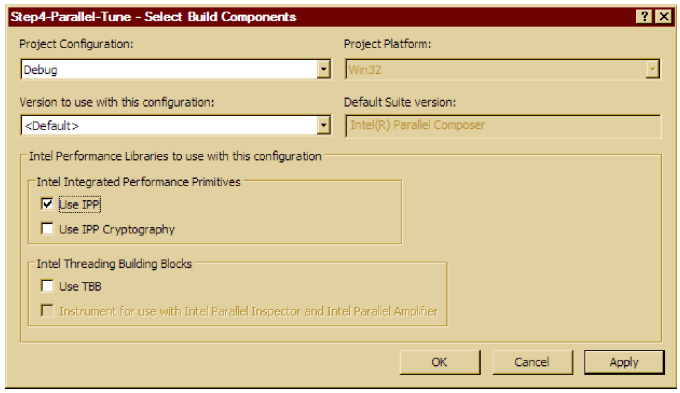


Figure 13: Directing Visual Studio* to use Intel® IPP

Auto-parallelization improves application performance by finding parallel loops capable of being executed safely in parallel and automatically generating multithreaded code, allowing you to take advantage of multicore processors. It relieves the user from having to deal with the low-level details of iteration partitioning, data sharing, thread scheduling, and synchronizations.

Auto-parallelization complements auto-vectorization and use of optimized valarray headers, giving you optimal performance on multicore systems that support SSE. For more information on multithreaded application support, see the user guide (http://software.intel.com/en-us/intel-parallel-composer/, then click the documentation link).

Intel Parallel Debugger Extension

Intel Parallel Composer includes the Intel Parallel Debugger Extension which, after installation, can be accessed through the Visual Studio Debug pull-down menu (see Figure 14 below).

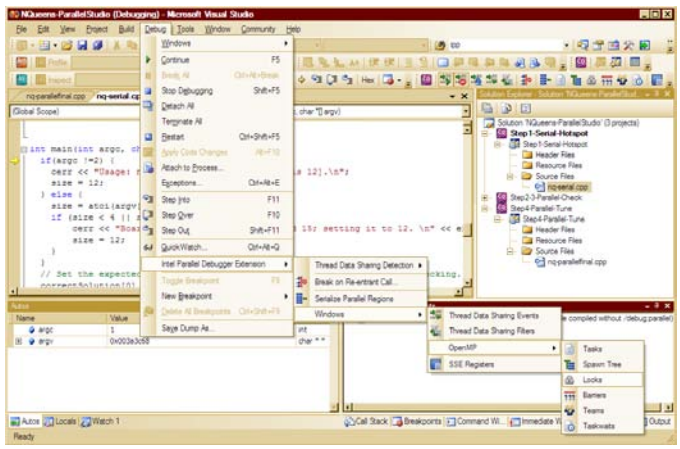


Figure 14: Intel® Parallel Debugger Extension is accessible from the Debug pull-down menu in Microsoft Visual Studio*

The Intel Parallel Debugger Extension provides you with additional insight and access to shared data and data dependencies in your parallel application. This facilitates faster development cycles and early detection of potential data access conflicts that can lead to serious runtime issues. After installing the Parallel Composer and starting Visual Studio, you can use the Intel Parallel Debugger Extension whenever your applications are taking advantage of Single Instruction Multiple Data (SIMD) execution and get additional insight into the execution flow and possible runtime conflicts if your parallelized application uses OpenMP threading.

To take advantage of the advanced features of the Intel Parallel Debugger Extension, such as shared data event detection, function re-entrancy detection, and OpenMP awareness including serialized execution of parallelized code, compile your code with the Intel Compiler using the /debug:parallel option for debug info instrumentation.

For more information, check out the "Intel® Parallel Debugger Extension" white paper at http://software.intel.com/en-us/articles/parallel-debugger-extension/. This paper goes into many more details and benefits that the Debugger Extension can bring to you, and how to best take advantage of them.

Intel Parallel Inspector

Find threading and memory errors

After you have added parallelism to your code, compiled and debugged, you can investigate the existence of potential memory and threading errors using Intel Parallel Inspector. Intel Parallel Inspector can be used with applications that contain programming interfaces mentioned earlier, such as Intel® TBB, Intel® IPP and OpenMP.

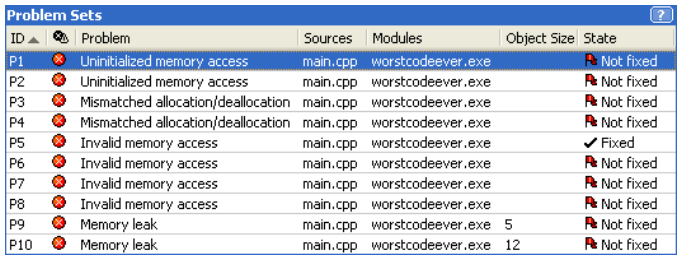


Figure 15. Quickly finds memory errors including leaks and corruptions in single and multithreaded applications. This decreases support costs by finding memory errors before an application ships.

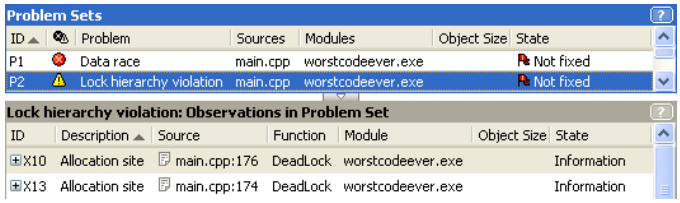


Figure 16. Accurately pinpoints latent threading errors including deadlocks and data races. This helps reduce stalls and crashes due to threading errors not found by debuggers and other tools.

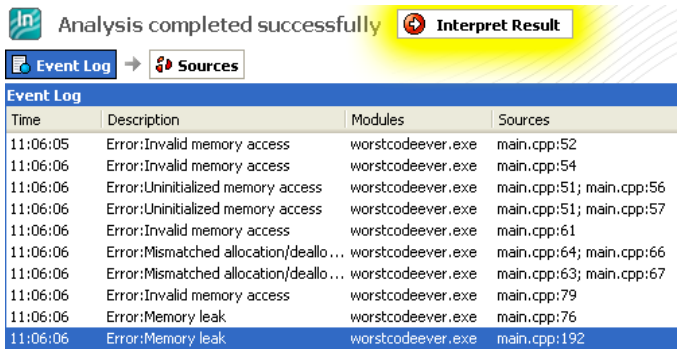


Figure 17. Click Interpret Result button to intuitively guide the developer by grouping related issues together. When you fix one problem, Parallel Inspector shows you all of the related locations where the same fix needs to be applied.

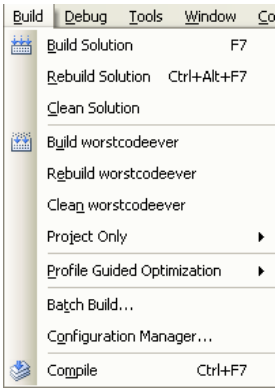


Figure 18. Parallel Inspector works on standard debug builds and even binaries. No special compilers, add-ins or special builds needed.

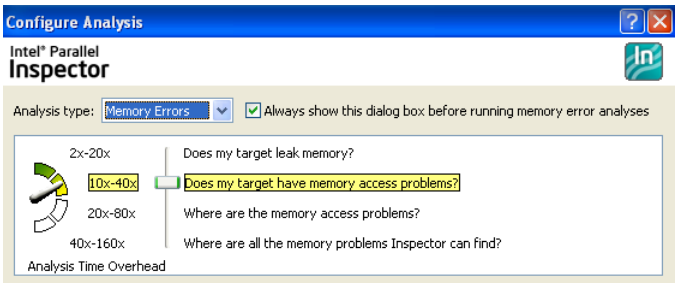


Figure 19. Simple analysis configuration enables developers to control the depth of analysis vs. collection time.

- L1 analysis finds memory leaks and deadlocks.
- L2 analysis identifies the existence of a problem.
- L3 analysis provides root cause information to fix problems.
- L4 provides the most comprehensive level of problem identification and detail.

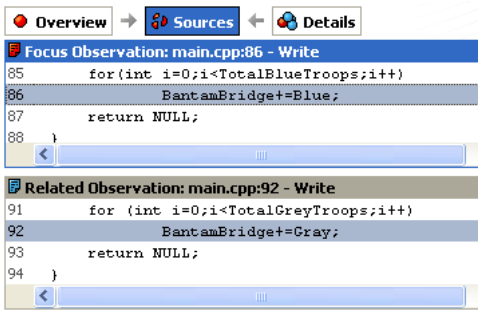


Figure 20. Click on an identified problem to reveal source code to go directly to the offending code to make changes quickly.

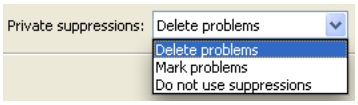


Figure 21. Result suppression allows you to mark or delete identified issues that are not relevant.

Figure 22. Before-and-after view of Intel® Parallel Inspector (incorrect memory accesses and leaks fixed)

Intel Parallel Amplifier

After you have checked for errors, you can quickly find multicore performance bottlenecks using Intel Parallel Amplifier. Intel Parallel Amplifier can be used with applications that contain programming interfaces mentioned earlier, such as Intel® TBB, Intel® IPP and OpenMP.

Get the Best Performance Out of Multicore

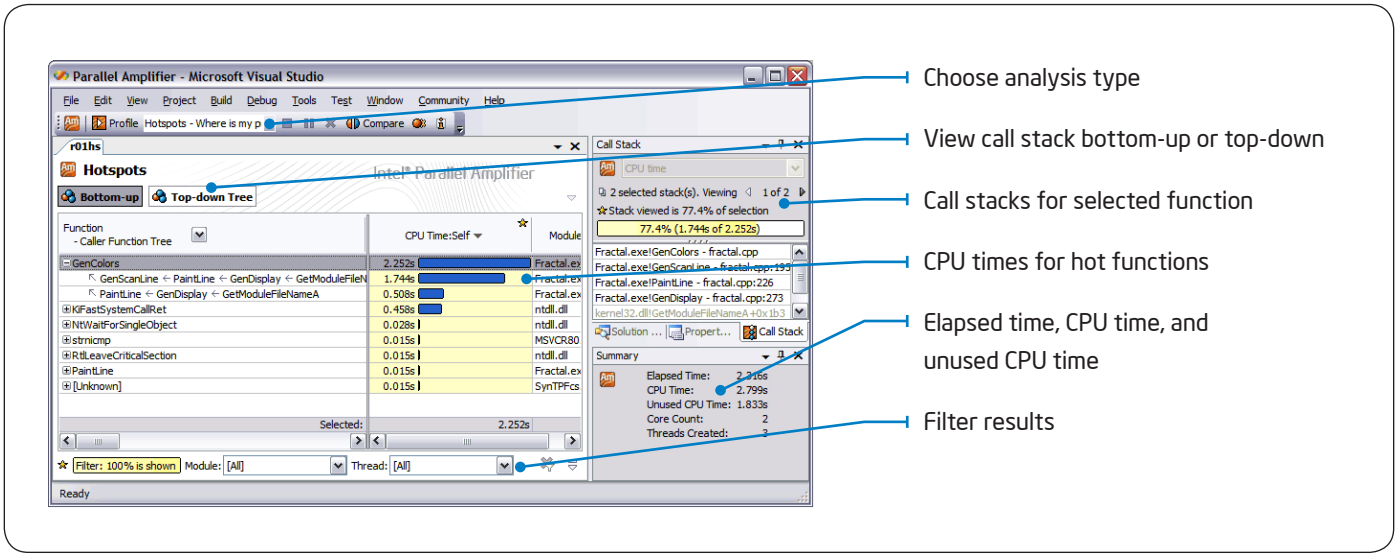


Figure 23. Hotspot analysis: Where is my application spending time?

Find the functions in your application that consume the most time. This is where you'll want to tune or add parallelism to make your program faster. Intel Parallel Amplifier also shows the stack, so you know how the function is being called. For functions with multiple calling sequences, this lets you see if one of the call stacks is hotter than the others.

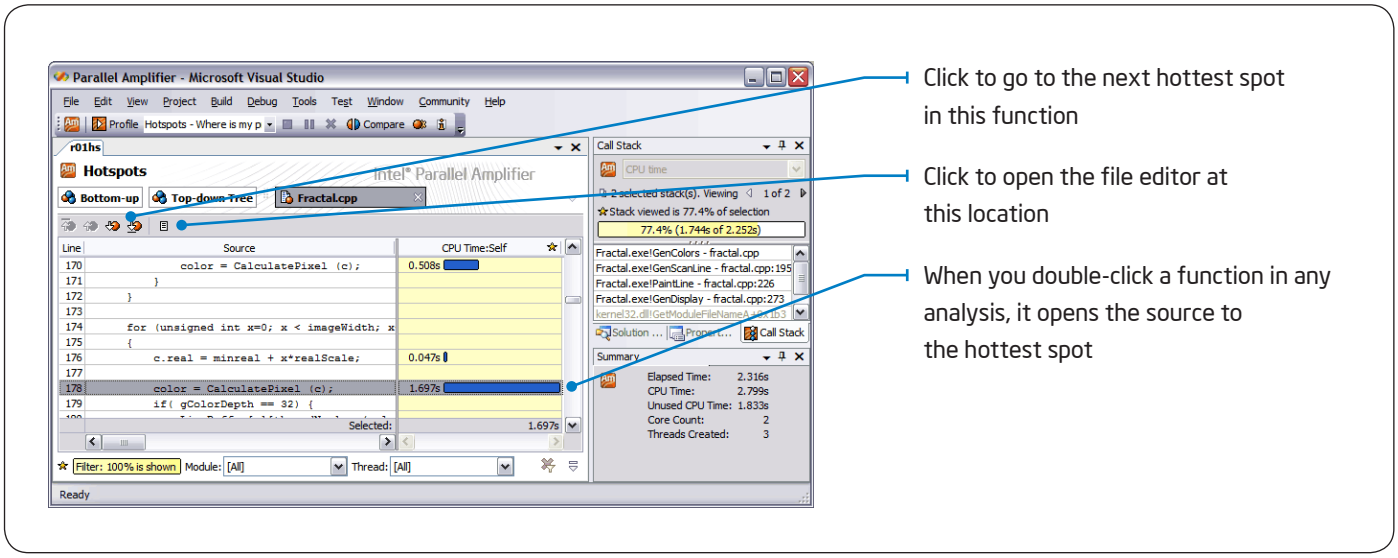


Figure 24. Source view: See the results on your source

Source view shows you the exact location on your source. Just double-click on the function names in any of the analysis views to see the source.

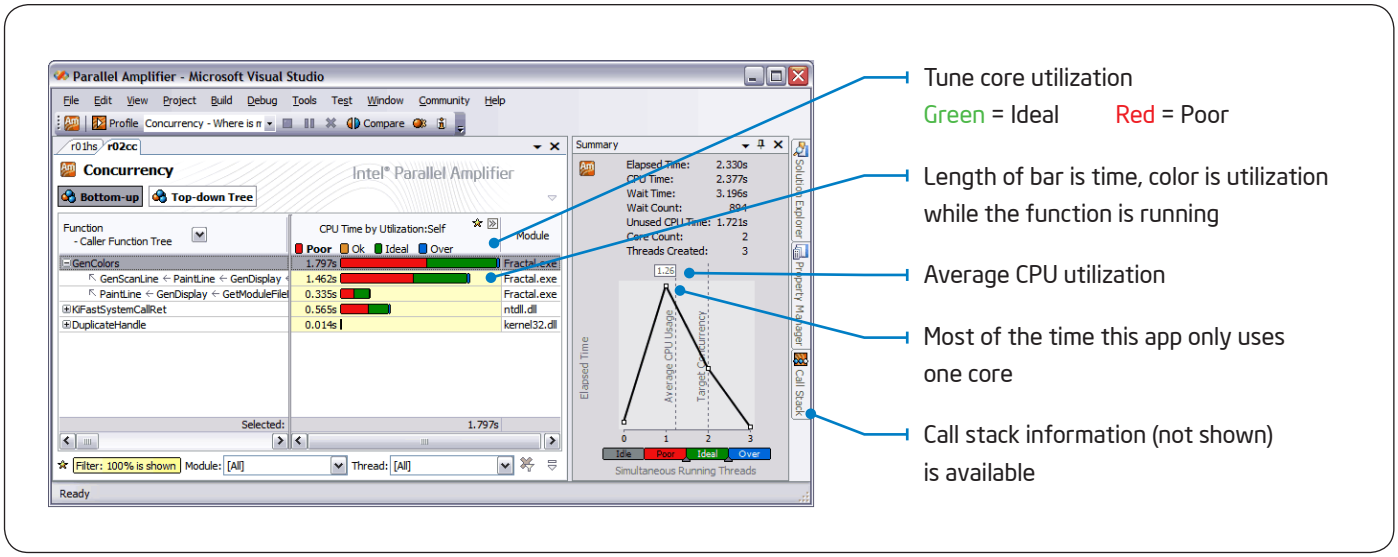


Figure 25. Concurrency analysis: When are cores idle?

Like hotspot analysis, concurrency analysis finds the functions where you are spending the most time. But it also shows you how well you are utilizing multiple cores. Color indicates the core utilization while the function is running. A green bar means all the cores are working. A red bar means cores are underutilized. When there is red, add parallelism and get all the cores working for you. This helps you ensure application performance scales as more cores are added.

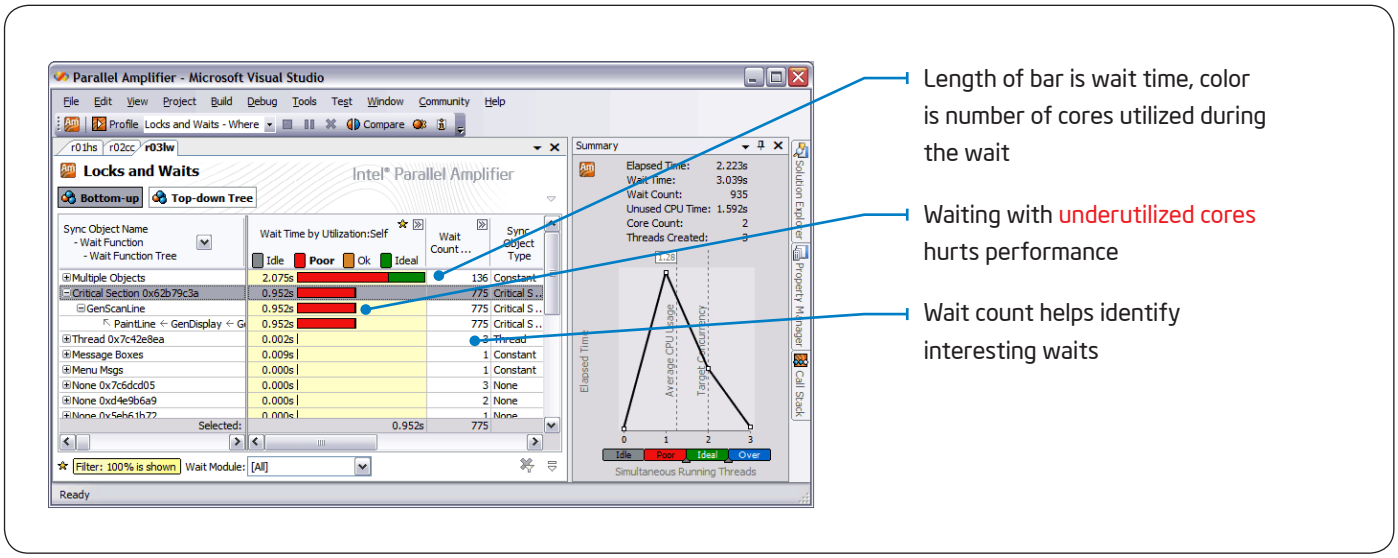


Figure 26. Locks and waits analysis: Where are the bad waits?

Waiting too long on a lock is a common source of performance problems. It's not bad to wait while all the cores are busy (green). It is bad to wait when there are unused cores available (red).

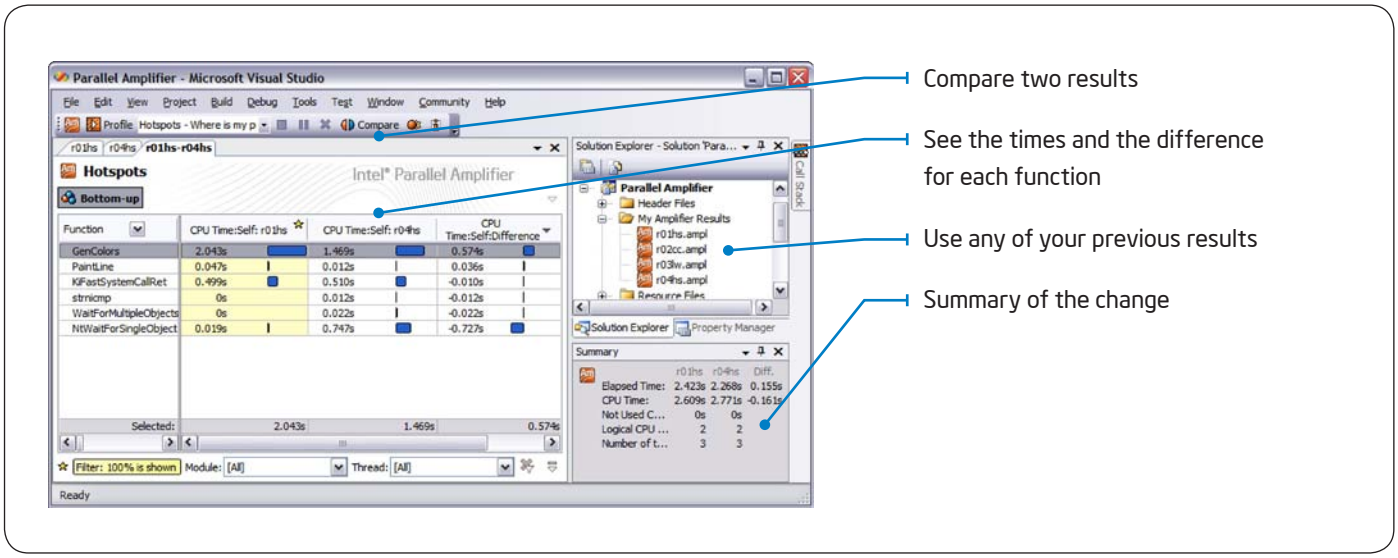


Figure 27. Compare results: Quickly see what changed.

This gives you a fast way to check progress when tuning and also makes a handy regression analysis.

Benefits

Designed for today's applications and tomorrow's software innovators. Everything you need to design and build parallel programs in Microsoft Visual Studio*

- Fully integrated in Visual Studio*
- Supports latest specifications of OpenMP*
- Prepares legacy serial and new parallel applications to take advantage of multicore and be ready to "forward scale" for manycore
  - Preserve investments in source code and development environments
  - Take advantage of the rapidly growing installed base of multicore systems
- Intel® Parallel Studio tools are available for each stage of the development lifecycle, from design assistance to coding and debugging to verifying error-free code to tuning
- Includes built-in guidance and recommendations, access to threading libraries containing thousands of code options
- Simplifies threading and provides capabilities to reduce bugs and system performance issues, helping you to bring better, feature-rich products to market faster
- Designed for architects and developers, leveraging decades of Intel experience providing software development products to help implement technical applications, databases, compute intensive applications, and threaded applications for high-performance computing (HPC)

System Requirements

- Microsoft Visual Studio* 2005 or 2008 (except the Express Edition)
- For the latest system requirements, go to: www.intel.com/software/products/systemrequirements/

Support

Intel Parallel Studio products include access to community forums and a knowledge base for all your technical support needs, including technical notes, application notes, documentation, and all product updates.

For more information, go to
http://software.intel.com/sites/support/

Download a Trial Version Today

Evaluation copy available at:
www.intel.com/software/products/ParallelStudio/

0209/BLA/CMD/PDF 321552-001