

Getting Started with the Intel® Parallel Composer

This guide tells you how to use the Intel® Parallel Composer features to introduce threads, compile, and debug the application. It uses simple code samples to show how to:

- Start Intel® Parallel Composer.
- Thread an application.
- Use Intel Libraries with a project.
- Begin debugging code using Intel® Parallel Debugger Extension.

The guide suggests common scenarios for selecting a threading technique that best fits your application performance goals and gives directions for finding more user and reference information.

NOTE: For a video demonstration on getting started with the Intel® Parallel Composer, try the [Show Me](#) Video. Show Me videos require Adobe* Flash* Player.

Contents

1	Open Samples	2
2	Start Intel® Parallel Composer	2
3	Thread Your Application	3
4	Debug Your Code with Intel® Parallel Debugger Extension.....	7
5	User and Reference Documentation	10
	Legal Information	12



1 Open Samples

Intel® Parallel Composer provides several sample applications, located at

C:\Program Files\Intel\Parallel Studio\Composer\
Samples\<locale>\C++\<solution name>.zip

1. Unzip <solution name>.zip to a directory of your choice.
2. Locate and double-click the .sln file or open the solution file from the Microsoft Visual Studio* development environment.

Discussion in the steps below uses the samples provided.

2 Start Intel® Parallel Composer

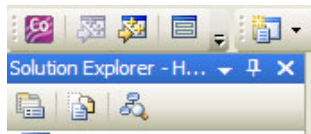
Intel® Parallel Composer integrates with:

- Microsoft Visual Studio* 2008
- Microsoft Visual Studio* 2005

To start the Composer, perform the following steps:

1. Launch Microsoft Visual Studio*.
2. In the **Solution Explorer** pane, open a .sln file from the supplied samples (for example, **par-OpenMP.sln**).

NOTE: The Composer menu commands referred to in the steps below are also available from the Composer toolbar:



3. From the Microsoft Visual Studio* main menu choose **Project > Intel Parallel Composer > Use Intel C++**.
4. Click **Yes** in the **Confirmation** dialog box. This configures the solution to use the Intel® C++ Compiler.
5. Select **Rebuild Solution** from the Visual Studio **Build** menu.

The results of the compilation are displayed in the **Output** window.



See Intel® Parallel Composer online help for more information on building applications; see [User and Reference Documentation](#) for information on accessing the online help.

3 Thread Your Application

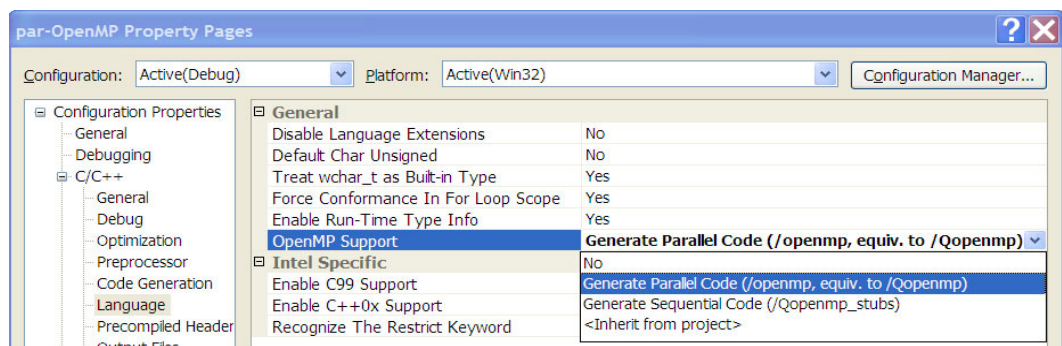
This section shows you how to parallelize your applications using the Intel® C++ Compiler and Intel Libraries. The simple code examples illustrate how to enable different parallelism techniques.

Parallelize Loops Using OpenMP*

You can thread your application by using OpenMP* directives or the Intel® C++ Compiler Language Extensions for Parallelism keywords.

Actions:

1. In Microsoft Visual Studio*, open the **par-OpenMP.sln** or **par-LangExtensions.sln** solution.
2. From the main menu choose **Project > Properties**.
3. In the **Property Pages** window, locate and select **C/C++ > Language > OpenMP Support**.
4. From the drop-down menu, select **Generate Parallel Code (/openmp, equiv. to /Qopenmp)**.



5. Click **Apply**.

Enable the OpenMP diagnostics

1. While still in Property Pages, add `/Qdiag-enable:openmp` under **Command Line > Additional Options**.



2. Click **OK**.
3. Rebuild the solution.

Results:

If the loop is successfully parallelized, you will see the message “OpenMP DEFINED LOOP WAS PARALLELIZED” in the output Window.

Sample Code Explanation:

If you examine the code in the sample, the `omp parallel for` directive specifies that the `for` loop that immediately follows is executed in parallel by multiple threads.

```
void f_sum ( size_t length, const float *a, const float *b, float *c ) {  
#pragma omp parallel for  
    for (int i=0; i<length; i++)  
        c[i] = a[i] + b[i];  
}
```

Using C++ Language Keywords

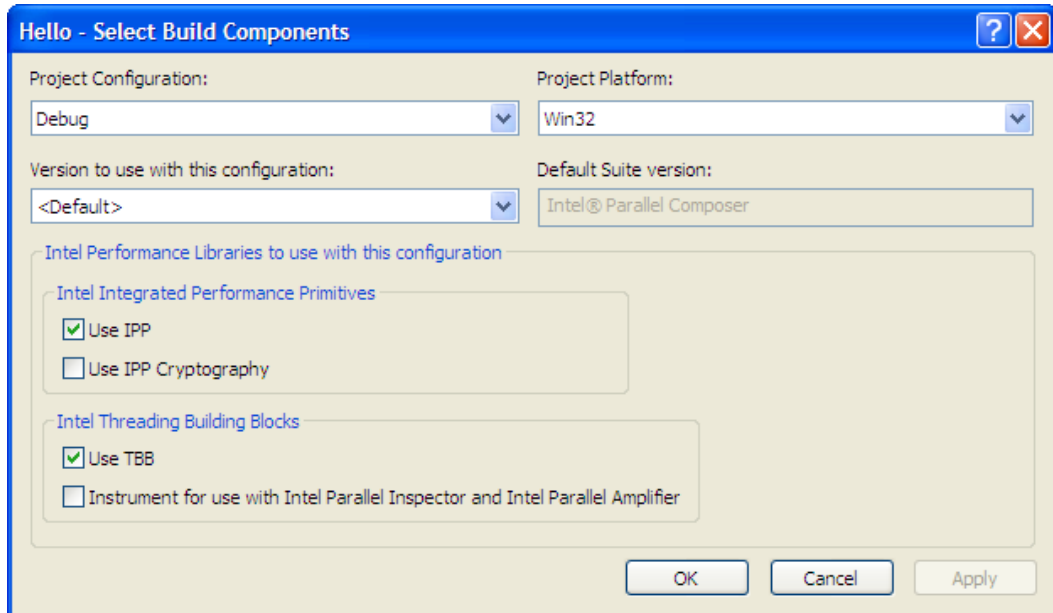
You can parallelize the `for` loop in the above code using the `__par` keyword as a statement prefix immediately before the `for` statement. The `__par` keyword is equivalent to the OpenMP directive `#pragma omp parallel for`:

```
void f_sum ( size_t length, const float *a, const float *b, float *c ) {  
    __par for (int i=0; i<length; i++)  
        c[i] = a[i] + b[i];  
}
```

Use Intel® IPP and Intel® TBB Libraries

You can thread your applications using Intel® Integrated Performance Primitives (Intel® IPP) or Intel® Threading Building Blocks (Intel® TBB).

1. Open your project in the **Solution Explorer** pane.
2. From the main menu, chose **Project > Intel Parallel Composer > Select Build Components**. The **Select Build Components** dialog box opens.



The dialog box allows you to enable one or both of the Intel Libraries.

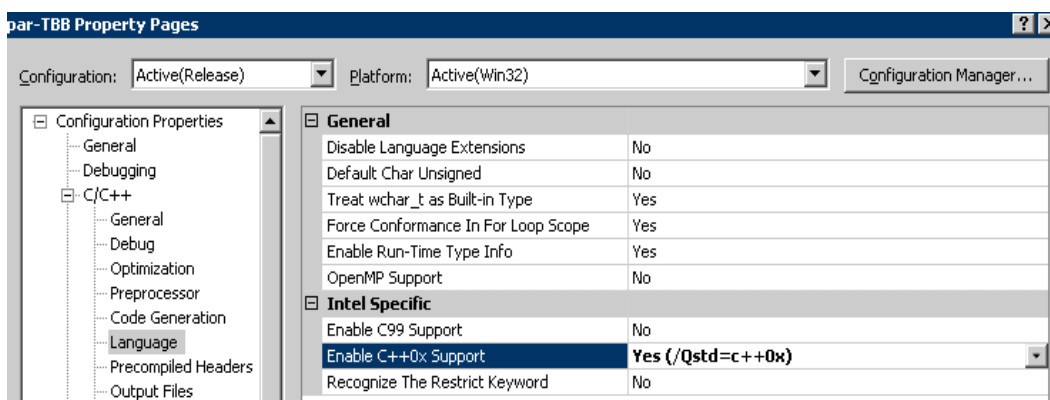
Action	Result
Check Use IPP	Enable the Common Intel® IPP libraries only.
Check Use IPP Cryptography	Enable both Cryptography Intel® IPP libraries and Common Intel® IPP libraries.
Uncheck Use IPP	Disable Intel® IPP libraries.
Check Use TBB	Enable Intel® TBB libraries.
Uncheck Use TBB	Disable Intel® TBB libraries.
Check Instrument for Use with Intel Parallel Inspector and Intel Parallel Amplifier	Add the preprocessor definition TBB_USE_THREADING_TOOLS to the project property Preprocessor > Preprocessor Definitions NOTE: The Instrument for use with Intel Parallel Inspector and Intel Parallel Amplifier is only available if the project property Code Generation > Runtime Library is set to Multi-threaded DLL (/MD) .

Once you click **OK** or **Apply**, the appropriate Intel® IPP and Intel® TBB include and lib paths for the selected architecture will be added to the project property settings.



Parallelize Loops Using Intel® TBB

1. Open the **par-TBB.sln** solution.
2. Enable Intel® TBB (as shown in the previous section).
3. From the main menu choose **Project > Properties**.
4. In the **Property Pages** window, locate and select **C/C++ > Language > Intel Specific > Enable C++0x Support**.
5. From the drop-down menu, select **Yes (/Qstd=c++0x)**.



Sample Code Explanation:

The code initializes the task scheduler for duration of the program:

```
tbb::task_scheduler_init init;
```

The iteration space here is of type `size_t`, and it goes from 0 to `length-1`. The template function `tbb::parallel_for` breaks the iteration space into chunks of iterations. The signature for `parallel_for` used in the example looks like:

```
template<typename Range, typename Body>
parallel_for( const Range& range, const Body& body,
              const auto_partitioner& );
```

The first step in parallelizing this loop is to convert the loop body into a body class, which is a function object or functor that operates on a chunk across the range:

```
[&](const tbb::blocked_range<size_t> &r) {
    for (size_t i=r.begin(); i<r.end(); i++)
        c[i] = a[i] + b[i];
}
```

[&] introduces lambda. The & inside specifies that the local variables outside the lambda should be accessed by reference inside the lambda, instead of making local



copies. See [User and Reference Documentation](#) to help locate additional information and help on the lambda expression.

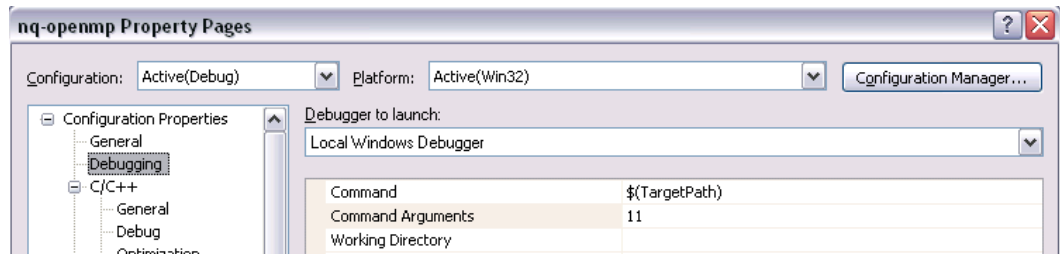
Select a Threading Method

Parallel Method	Benefits	Considerations
Intel® IPP	Highest performance, scalable for certain multimedia application domains; does not expose parallelism in your code.	Cryptography, image, audio, video, compression, and the like. Best for multimedia.
Auto-parallel	Automatic parallelization; does not expose explicit parallel syntax in your code.	Limited to well-formed loops.
New Parallel Extensions	Good for rapid prototyping, easy syntax, maps to OpenMP* 3.0 runtime.	Does not have the controls for shared/private variables like OpenMP* or Intel® TBB.
OpenMP* 3.0	Well-known standard, efficient runtime, scalable. OpenMP 3.0 support for <code>task</code> construct helps function-level parallelism; Intel support for OpenMP* is compatible with Microsoft implementation.	Pragma use is not common for some C++ developers; can be difficult for C++ objects/data; Microsoft Visual Studio* 2008 supports OpenMP* 2.5.
Intel® TBB	Natural C++ solution, efficient, scalable runtime.	Need to understand C++ Object Oriented Programming: templates and containers.

4 *Debug Your Code with Intel® Parallel Debugger Extension*

Prepare Example Code for Debugging




1. Unzip the **NQueens-ParallelStudio.zip** to a directory of your choice.
2. In Microsoft Visual Studio* open the `nq-openmp` solution and double-click **nq-openmp.cpp**.
3. Navigate to line 84 and comment out the line `#pragma omp atomic` to create a data sharing violation.
4. Ensure that the `nq-openmp` project properties show that the command argument `11` is being handed over to the program.

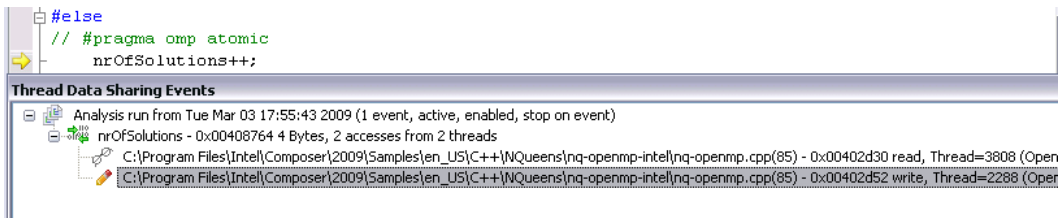


5. Ensure that the compiler command line options `/ZI`, `/debug:parallel`, and `/Qopenmp` are set on the **Property Pages** of the project:
Configuration Properties > C/C++ > General > Debug Information Format, Configuration Properties > C/C++ > Debug > Enable Parallel Debug Checks, and **Configuration Properties > C/C++ > Language > General > OpenMP Support** respectively.
6. Confirm that the linker command line option `/debug` is set:
Configuration Properties > Linker > Generate Debug Info.
7. Rebuild the project.

Identify Parallel Coding Issues

To identify **Thread Data Sharing Events** that might trigger runtime data sharing violations, do the following:

1. Select **Enable Detection** from **Debug > Intel Parallel Debugger Extension > Thread Data Sharing Detection** menu (or click ).
2. Start the debug session by selecting the **Start Debugging** from the **Debug** menu (or click ). The program execution will be halted as soon as a thread data sharing event is detected.
3. Click the **Thread Data Sharing Events** button  to see a log of events that caused the execution to be halted.






Result:

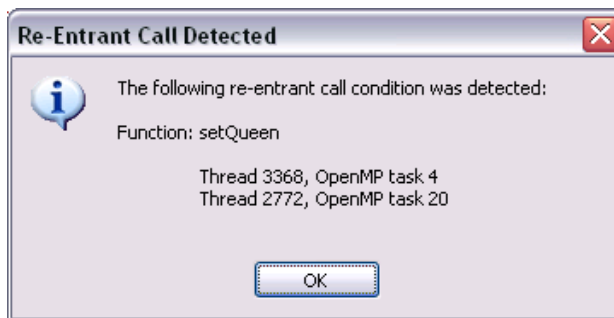
You can see from the **Thread Data Sharing Events** log that two different OpenMP* threads have been trying to access the `nrOfSolutions` variable: thread 3808 read the value and thread 2288 incremented the value. Since there are two different threads



modifying and relying on a single variable this is a potentially serious problem that causes the N-Queens algorithm to produce incorrect results. Use the `#pragma omp atomic` to make the `nrOfSolutions++` increment thread safe.

To identify **Function Re-Entrancy** events, do the following:

1. Disable the data sharing event detection by clicking on the **Enable Detection** button  so it changes to a disabled state .
2. Select **Debug > Intel Parallel Debugger Extension > Break on Re-Entrant Call** from the menu (or click .
3. In the pop-up window that is being displayed enter `{,,nq-openmp.exe)setQueen`. Simply entering `setQueen` would also suffice.
4. Select **Continue Debug**.
5. Upon detecting function re-entrancy, the **Re-entrant Call Detected** window opens with information about the thread IDs of the threads involved.



6. Select **OK**.

Result:

The debugger instruction pointer sits at the function entry point of the function where re-entrance was detected.

```
void setQueen(int queens[], int row, int col, int id) {
    for(int i=0; i<row; i++) {
        // vertical attacks
        if (queens[i]==col) {
            return;
        }
    }
}
```

OpenMP task 4 and OpenMP task 20 both entered the function `setQueen()`. Both tasks are modifying values and using data inside this function. You need to ensure that they do not overwrite each other's data. If you observed a runtime problem in the function `setQueen()`, this could be an important pointer telling you that `setQueen()` needs to be implemented in a thread-safe manner if it is not already thread-safe.

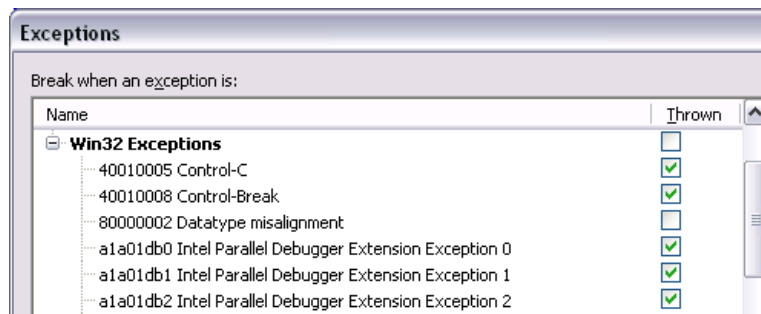


Other Debugging Considerations

If you are using Microsoft Visual Studio* 2005 you might first need to verify that the Intel® Parallel Debugger Exceptions are enabled in the Microsoft Visual Studio* Debugger.

Verify the exceptions are enabled by doing the following:

1. From the **Debug** pull-down menu select **Exceptions**.
2. Ensure that in the **Win32 Exceptions** treeview list the boxes for Intel Parallel Debugger Extension Exception are enabled.



3. Go to the **File** pull-down menu, and select **Save all**.

5 User and Reference Documentation

This guide focuses on basic features of the Intel® Parallel Composer. To explore more features and get the most out of the Intel® Parallel Composer, refer to the following documentation and resources:

Resource	Notes
<i>Intel® Parallel Composer Documentation</i>	Use this HTML page to locate additional Intel® Parallel Composer documentation for the following: Intel® C++ Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Parallel Debugger Extension To open this HTML page from the Windows* Start menu, choose Intel Parallel Studio > Parallel Studio Documentation > Composer Documentation .
Sample code	Use sample code in a zip file provided at <code><install_dir>\Samples\<locale>\C++\</code> to learn how to use various threading techniques.



<p><i>Intel® IPP Static Libraries</i></p>	<p>Knowledge Base Article describing how to download and install the Intel® Integrated Performance Primitives Static Libraries for Intel® Parallel Composer: http://software.intel.com/en-us/articles/intel-ipp-static-libraries/</p>
<p><i>Intel® Parallel Studio Resources</i></p>	<p>Intel Parallel Studio provides the most comprehensive set of tools for parallelism including Intel® Parallel Amplifier, Intel® Parallel Composer, and Intel® Parallel Inspector.</p> <p>Refer to the <i>Getting Started with the Parallel Studio</i> for an overview of all the components in the Parallel Studio. From the Windows* Start menu, choose Intel Parallel Studio >Getting Started>Parallel Studio Getting Started Guide.</p> <p>To open documentation that points to more resources for each installed Intel Parallel Studio tool, from the Windows* Start menu, choose Intel Parallel Studio > Parallel Studio Documentation > Composer Documentation.</p> <p>You can find additional information on the Intel® Parallel Studio at http://www.intel.com/software/products/</p>



Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2009, Intel Corporation. All rights reserved.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.