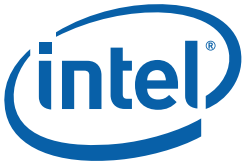


Getting Started with the Intel(R) C++ Compiler Professional Edition 11.1 for Windows* OS

This guide shows you how to start Intel® C++ Compiler Professional Edition, use Intel® Libraries with your project, and 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.

Contents

1 Start Intel® C++ Compiler Professional Edition

The Intel® C++ Compiler Professional Edition 11.1 for Windows* OS integrates into the following versions of the Microsoft Visual Studio* IDE:

- Microsoft Visual Studio 2008*.
- Microsoft Visual Studio 2005*.
- Microsoft Visual Studio .NET 2003*.

To start the Intel® Compiler Professional Edition from Microsoft Visual Studio* IDE, perform the following steps:

1. Launch Microsoft Visual Studio*.
2. Open or create a Visual Studio solution in the **Solution Explorer** pane.
3. From the **Project** menu, or project context menu, select **Intel C++ Compiler Pro > Use Intel C++**.
4. Click **Yes** in the **Confirmation** dialog box. This configures the solution to use the Intel® C++ Compiler (you may configure the solution to use the Intel® C++ compiler by clicking  on the toolbar too).
5. Select **Rebuild Solution** from the Visual Studio **Build** menu.

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

See Intel® C++ Compiler Professional Edition online help for more information on building applications. See User and Reference Documentation to help locate additional information and help.

Start the Compiler from the Command Line

To start the compiler from the Windows command line, use the Windows program menu to select:

Intel(R) Software Development Tools >
Intel(R) C++ Compiler Professional 11.1.xxx >
Build Environment for...

This method opens a Windows command prompt and sets the environment variables for the Intel compiler. To compile a C++ language source file (for example, `my_source_file.cpp`), use this command:

```
icl my_source_file.cpp
```

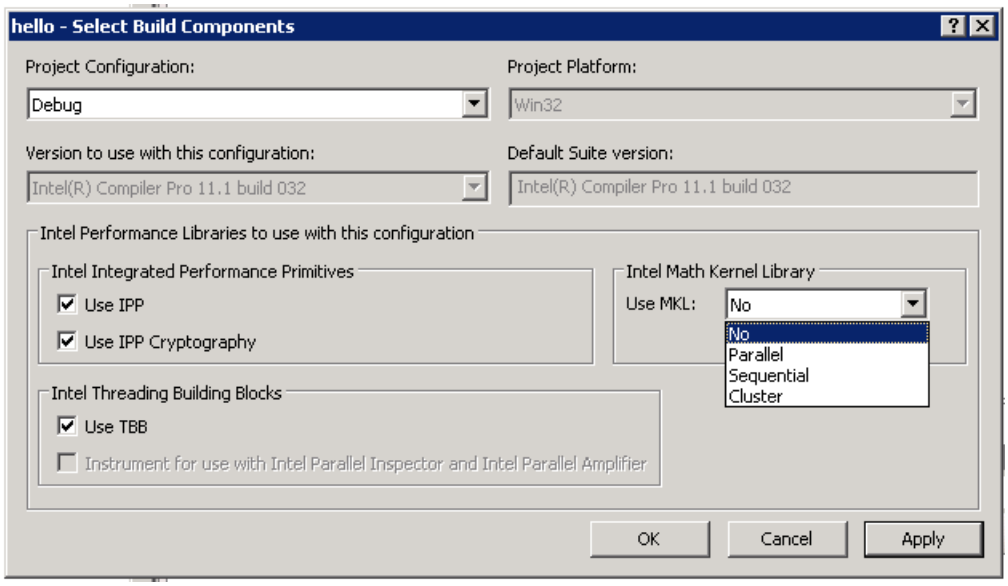
Following successful compilation, an executable named `my_source_file.exe` is created in the current directory.

2 Use Intel® Performance Libraries

To use the Intel® C++ project with the Intel® Integrated Performance Primitives, Intel® Threading Building Blocks, or Intel® Math Kernel Library, do the following:

1. Open your project in the Solution Explorer.
2. On the menu bar, select **Project > Intel C++ Compiler Pro > Select Build Components** (or click  on the toolbar).

The **Select Build Components** dialog box opens:



The dialog box allows you to enable Intel Performance 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 **Use MKL**:	Enable Intel® MKL libraries.
Select **Parallel**	Use parallel Intel® MKL libraries.
Select **Sequential**	Use sequential Intel® MKL libraries.
Select **Cluster**	Use cluster Intel® MKL libraries.
Select **No**	Disable all Intel® MKL libraries.

Once you click **OK** or **Apply** in the **Select Build Components** dialog box, the appropriate Intel® IPP, Intel® TBB, or Intel® MKL `include` paths and `link` libraries for the selected architecture will be added to the project property settings.

If your target platform is IA-64 or Intel® 64, an additional, **Use ILP64 interfaces**, switch box appears. If switched on, corresponding `ilp` MKL libraries will be added to the linker command line and `MKL_ILP64` preprocessor definition will be added to the compiler command line, otherwise `lp` MKL libraries will be used by default.

NOTE: See the *Intel® MKL User's Guide* for explanation of the difference between LP and ILP64 versions of the library.

3 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 `task` 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.
Intel® MKL	Highly optimized, extensively threaded math routines for science, engineering, and financial applications that require maximum performance; does not expose parallelism in your code.	Extensively threaded math functions: BLAS, LAPACK, Sparse Solvers, Fast Fourier Transforms, Vector Math, Vector Statistics.

4 Debug Your Code with Intel® Parallel Debugger Extension

Open Samples

Intel® C++ Compiler Professional Edition provides several sample applications, located at:

`<install-dir>\Samples\<locale>\C++\<solution name>.zip`

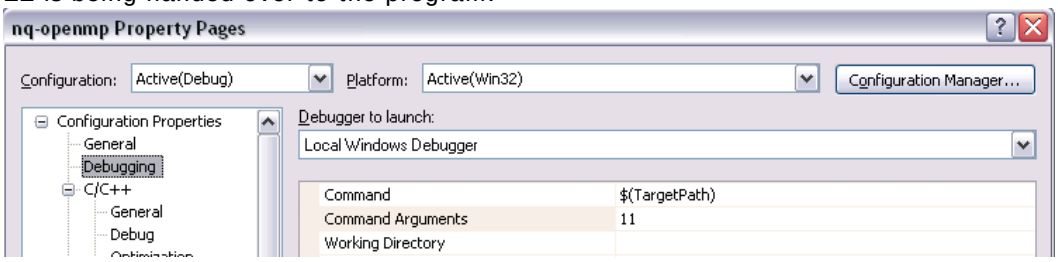
1. Unzip `<solution name>.zip` to a directory of your choice.
2. Locate and double-click the `.sln` file or open the solution file from the Microsoft Visual Studio* development environment.

Discussion in the steps below uses the **NQueens.zip** samples provided.

NOTE: If, during product installation, you accepted the default installation location, `<install-dir>` is `C:\Program Files\Intel\Compiler\11.1\<build_number>`.

Prepare Example Code for Debugging

1. Unzip the `NQueens.zip` to a directory of your choice.
2. Open sample projects in Microsoft Visual Studio* by double-clicking the `NQueens.sln`.
3. In `nq-openmp-intel` project, double-click `nq-openmp.cpp`.
4. Comment out the line `#pragma omp atomic` to create a data sharing violation.
5. Ensure that the `nq-openmp` project properties show that the command argument `11` is being handed over to the program.



6. 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**

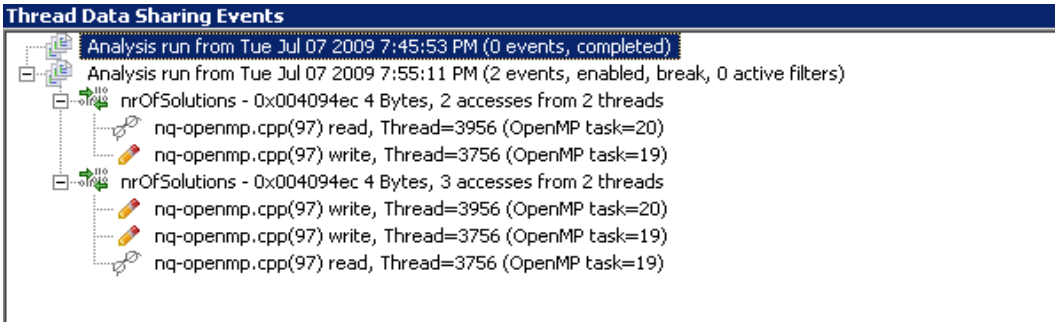
Debug Checks; and **Configuration Properties > C/C++ > Language > General > OpenMP Support** respectively.

7. Confirm that the linker command line option /DEBUG is set: **Configuration Properties > Linker > Debugging > Generate Debug Info**.
8. 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 3956 read the value and thread 3756 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 20 and OpenMP task 19 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.

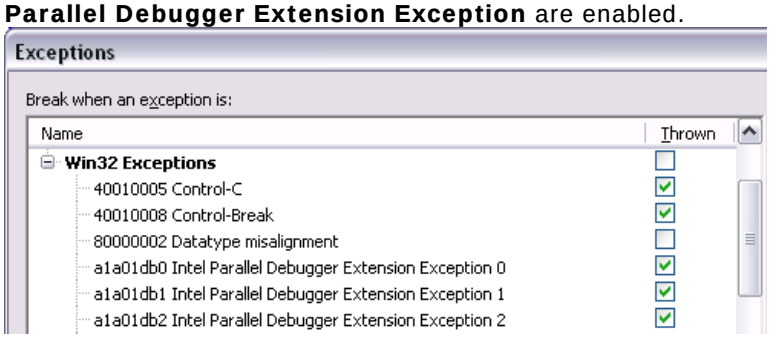
Other Debugging Considerations

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** tree-view 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 Intel® C++ Compiler Professional features. To explore more features, check the following resources.

Resources	Notes
Intel® C++ Compiler Professional Edition Documentation	Use this HTML page to locate additional Intel® C++ Compiler Professional Edition 11.1 documentation for the following: • Intel® C++ Compiler. • Intel® Threading Building Blocks. • Intel® Math Kernel Library. • Intel® Integrated Performance Primitives. • Intel® Parallel Debugger Extension. To open this HTML page, from the Windows* Start menu, choose **Intel® Software Development Tools > Intel® C++ Compiler Professional 11.1.xxx > Documentation**.
Sample code	Use sample code in a zip file provided at `<install_dir>\Samples\<locale>\C++\` to learn how to use various threading techniques. `samples.htm` provides an overview of the C++ samples.
Integrated User and Reference Help	Open the *Integrated User and Reference Help* from Microsoft Visual Studio* Help menu. Select **Help > Intel C++ Compiler Pro > Intel C++ Compiler Help**. To enable Intel C++ quick start help topic, click  on the toolbar. Filter the contents and index information: select **Intel** from the **Filtered by** drop-down list to view only the Intel® C++ Compiler Pro documentation.
Product Information and Technical Support	For general product information or information on support for Intel software products, visit the Intel web site: http://www.intel.com/software/products/. At this site, you will find comprehensive product information, including: • Links to each product, where you will find technical information such as white papers and articles.

	• Links to user forums. • Links to news and events. To find technical support information, to register your product, or to contact Intel, please visit: http://www.intel.com/software/products/support. For additional information, see the Technical Support section of your Release Notes. See the registration web site for licensing and registration guidance: https://registrationcenter.intel.com/.

Legal Information