Intel® Parallel Composer 2011 Getting Started Tutorials

Document Number: 323647-001US

World Wide Web: http://developer.intel.com

Contents

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) 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 go to: http://www.intel.com/design/literature.htm.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to
http://www.intel.com/products/processor%5Fnumber/ for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Core Inside, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, skoool, the skoool logo, Sound Mark, The Journey 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.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

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

Introducing the Intel® Parallel Composer 2011

This guide shows you how to start the Intel® Parallel Composer 2011, use Intel® Libraries with your project, and begin debugging code using the Intel® Parallel Debugger Extension. The Intel Parallel Composer 2011 is a comprehensive set of software development tools that includes the following components:

- Intel® C++ Compiler
- Intel® Integrated Performance Primitives
- Intel® Threading Building Blocks
- Intel® Parallel Debugger Extension

Check http://software.intel.com/en-us/articles/intel-software-product-tutorials/ for the following:

- Printable version (PDF) of all Composer tutorials
- ShowMe videos of each Composer tutorial

Prerequisites

You need the following tools, skills, and knowledge to effectively use these tutorials.

NOTE. Although the instructions and screen captures in these tutorials refer to the Visual Studio* 2005 integrated development environment (IDE), you can use these tutorials with later versions of Visual Studio.

Required Tools

You need the following tools to use these tutorials:

- Microsoft Visual Studio 2005 or later.
- Intel® Parallel Composer 2011.
- Intel® C++ Composer XE 2011.
- Sample code included with Intel® Parallel Studio 2011.
- Sample code included with the Intel® C++ Composer XE 2011.

NOTE.

- Samples are non-deterministic. Your results may vary from the examples shown throughout these tutorials.
- Samples are designed only to illustrate features and do not represent best practices for creating multithreaded code.

Required Skills and Knowledge

These tutorials are designed for developers with a basic understanding of Microsoft Visual Studio, including how to:

- open a project/solution.
- access the **Document Explorer**.
- display the **Solution Explorer**.
- compile and link a project.
- ensure a project compiled successfully.

Getting Started with the Intel® C++ Parallel Composer 2011

The Intel® Parallel Composer 2011 integrates into the following versions of the Microsoft Visual Studio* Integrated Development Environment (IDE):

- Microsoft Visual Studio 2010*
- Microsoft Visual Studio 2008*
- Microsoft Visual Studio 2005*

To start the Intel® Parallel Composer 2011 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, select **Intel Parallel Composer 2011 > Use Intel C++**.
4. Click **OK** in the **Confirmation** dialog box. This configures the solution to use the Intel® C++ Compiler. (For Visual Studio 2008 or Visual Studio 2005 you may configure the solution to use the Intel® C++ Compiler by clicking on the toolbar icon  . For Visual Studio 2010, you can use **Project > Properties General > Platform Toolset** to select the Intel C++ Compiler. This method is equivalent to using the Use Intel C++ menu item except you can make the selection in individual build configurations.)
5. Select **Rebuild Solution** from the Visual Studio **Build** menu.

The results of the compilation display in the Output window.

Setting Compiler Options

1. Select **Project > Properties**. The Property Pages for your solution display.
2. Locate **C/C++** in the list and expand the heading.
3. Step through the available properties to select your configuration.

You can also start the Intel® C++ Compiler from the command line. These tutorials assume you will be using the IDE. Refer to the User and Reference Guide for more information.

Tutorial: Threading Your Application with Intel® Cilk™ Plus

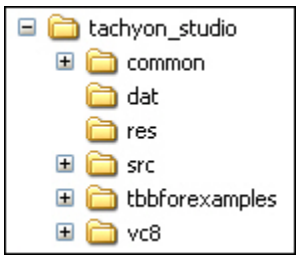
1

Learning Objectives

In this tutorial, we will be building different parallel implementations of the same function with both the Microsoft Visual C++* Compiler and Intel® Parallel Composer 2011. When executed, the application will display the execution time required to render the object in the window title. This time is an indication of the speedup obtained with parallel implementations compared to a baseline established with a serial implementation in the first step.

Threading Your Application with Intel® Cilk™ Plus

Tachyon is a ray-tracer application, rendering objects described in data files. The Tachyon program is located in the product Samples directory: `\Parallel Studio 2011\Samples\<locale>\tachyon_studio.zip`. Expand the archive to `\Tachyon`

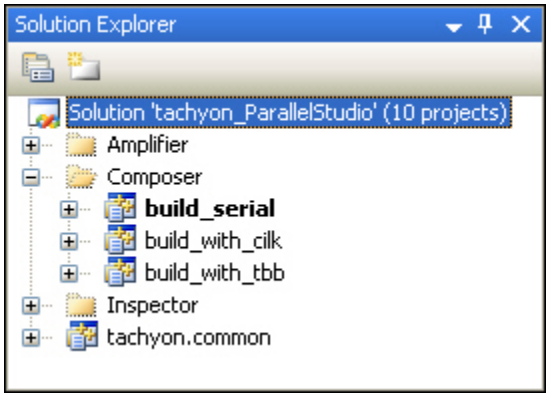


By default we use `balls.dat` as the input file. Data files are stored in the directory `\tachyon_studio\dat\`. Originally, Tachyon was an application with parallelism implemented in function `pthread_create()`(source file `\tachyon_studio\src\Windows\pthread.cpp`) with explicit threads: one for the rendering, and the other for calculations. In this tutorial we implement parallelization on the calculation thread with Intel® Cilk™ Plus. Parallelization is implemented only for one function `draw_task()`, which you can find in the source file `build_serial.cpp`, in project `build_serial`.

Open the Microsoft Visual Studio* Solution `\tachyon_studio\vc8\tachyon_ParallelStudio.sln`. It includes these projects:

- `build_serial`
- `build_with_cilk`
- `build_with_tbb`

- tachyon.common



NOTE. The build_serial, build_with_tbb, and build_with_cilk projects are dependent on the project tachyon.common. The build_with_tbb and build_with_cilk projects use Intel® TBB and Intel Cilk Plus respectively.

Follow the steps below to build the serial and Intel Cilk Plus approaches to Tachyon.

Workflow Steps

In the following, we will be building different parallel implementations of the same function with both the Microsoft Visual C++ Compiler and the Intel® C++ Compiler. When executed, the application will display the execution time required to render the object in the window title. This time is an indication of the speedup obtained with parallel implementations compared to a baseline established with a serial implementation in the first step.

Building the Serial Project

1. Set the build_serial project as the StartUp project (**Project > Set as StartUp Project**).
2. Set the configuration to Release mode: **Build > Configuration Manager > Active solution configuration: > Release**, then build the build_serial project.
3. Execute the application tachyon_compiler.exe with **Debug > Start without Debugging**. Take a note of the time in seconds displayed in the window title. This time to render the image is the baseline for parallelization with the Microsoft Visual C++ Compiler.
4. For projects build_serial and "tachyon.common" change compiler to Intel Parallel Composer (**Project > Intel Parallel Composer 2011 > Use Intel C++ ...**).
5. Rebuild build_serial in Release mode (now with Intel Compiler).
6. Execute the application. Note the time to render the image as the baseline for parallelization with the Intel C++ Compiler.

Building with Intel® Cilk™ Plus

1. Set the build_with_cilk project as the StartUp project.

2. For project `build_with_cilk` change compiler to the Intel C++ Compiler (**Project > Intel Parallel Composer 2011 > Use Intel C++ ...**).
3. For the project `build_with_cilk` make sure Intel Cilk Plus for Intel® C++ Compiler additional include directory is set (**Project > Properties > Configuration Properties > C/C++ > General > Additional Include Directories = C:\Program Files\Intel\Parallel Studio 2011\Composer\compiler\include\cilk**).
4. Open source file `build_with_cilk.cpp` in the project `build_with_cilk`.
5. Uncomment Intel Cilk Plus header files.
6. Uncomment routine `draw_task` related to Intel Cilk Plus implementation.
7. Comment out the serial `draw_task()` function
8. Build `build_with_cilk` in Release mode.
9. Execute the application.
10. Measure performance compared with the serial version for Intel Parallel Composer.

Platform and Other Details

The solution for this example was created in Microsoft Visual Studio 2005. If you open the `tachyon_compiler.sln` solution in Microsoft Visual Studio 2008, then it will be converted to a Microsoft Visual Studio 2008 solution.

For Platform Win32

- The executable file for all implementations is `tachyon_compiler.exe` in the `\tachyon_studio\vc8\Release\` directory.
- Object files are stored in `\tachyon_studio\vc8\tachyon_compiler\Release\` directory.

For Platform x64

- The executable file for all implementations is `tachyon_compiler.exe` in the `\tachyon_studio\vc8\x64\Release\` directory.
- Object files are stored in `\tachyon_studio\vc8\x64\tachyon_compiler\Release\` directory.

In addition to the serial and Intel Cilk Plus implementations covered in this tutorial, there is also an implementation of Tachyon using Intel® Threading Building Blocks, `build_with_tbb`. Please search for "TODO" in the source code for steps to modify the source for parallelization with Intel Threading Building Blocks.

Tutorial: Using Intel® Integrated Performance Primitives 2

Learning Objectives

This tutorial shows how to use Intel® Integrated Performance Primitives (Intel® IPP) to add vectorization to an application, how to use some of the Intel IPP APIs, and provides an example of how to add threading to an Intel IPP application using Intel® Threading Building Blocks (Intel® TBB) and Intel® Cilk™ Plus.

Estimated completion time: two hours.

After you complete this tutorial, you should be able to do the following:

- Add Intel IPP and Intel TBB to your Microsoft* Visual Studio* project.
- Set the number of Intel IPP threads programmatically.
- Use Intel IPP to copy the contents of one buffer to another.
- Use Intel IPP to compare each of the elements of one buffer to a constant.
- Use Intel IPP to perform a binary `AND` operation on the contents of two buffers.
- Use Intel TBB to add threading to the `Game of Life` application.
- Use Intel Cilk Plus to add threading to the `Game of Life` application.

Key Terms and Concepts

Key Terms

region of interest: Most Intel® Integrated Performance Primitives (Intel® IPP) image processing functions can operate not only on entire images but also on image areas. The image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

channel: The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The image can be either pixel-order or planar. For pixel-order images, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

Key Concept: Adding Intel IPP and Intel TBB to your project

There is an easy way to add Intel IPP and Intel® Threading Building Blocks (Intel® TBB) compile paths and link libraries to your project. Intel IPP and Intel TBB functions can be embedded in a C++ DLL, which is called from the .NET language.

Key Concept: Using Intel IPP functions

Intel IPP image processing functions typically operate on regions of interest and image data in buffers. There are many Intel IPP image processing functions for manipulating buffers. This tutorial discusses several Intel IPP functions.

Key Concept: Threading an application that uses Intel IPP

Intel TBB or Intel® Cilk™ Plus can be used to thread an application using Intel IPP. Intel IPP functions are thread safe so they can be called by multiple threads without causing synchronization bugs, as long as the data that is passed to them is protected.

The Game of Life Samples

The **Game of Life** samples are a collection of Microsoft* Visual Studio* 2005 project, solution, and Visual Basic* .NET and C++ source files. The **Game of Life** samples are located in the product Samples directory: `\Parallel Studio 2011\Samples\en_US\Life.zip`.

The algorithm can be implemented using different approaches. For example, you can optimize for speed using vectorization, or threading, or both. There are several different samples available. The samples illustrate different approaches to optimizing the **Game of Life** algorithm using Intel® Integrated Performance Primitives (Intel® IPP), Intel® Threading Building Blocks (Intel® TBB), and the Intel® C++ Compiler with Intel® Cilk™ Plus.

The **Game of Life** algorithm was first introduced by British mathematician John Conway in 1970 and is an example of a cellular automaton. For more information on the **Game of Life** algorithm, search the web.

Workflow Steps

In each of the following sections, step-by-step instructions are provided on how to convert the **Game of Life** application from a serial application to a parallel application. The serial project provides baseline performance.

There are a number of technologies to take advantage of vector instructions in the processor and to thread an application; this example uses Intel® Integrated Performance Primitives (Intel® IPP) to provide vectorization, and Intel® Threading Building Blocks (Intel® TBB) or Intel® Cilk™ Plus to thread the application.

To help you see the code lines implementing vectorization and threading, the source code is annotated with `# defines` stating the action to take in the source code.

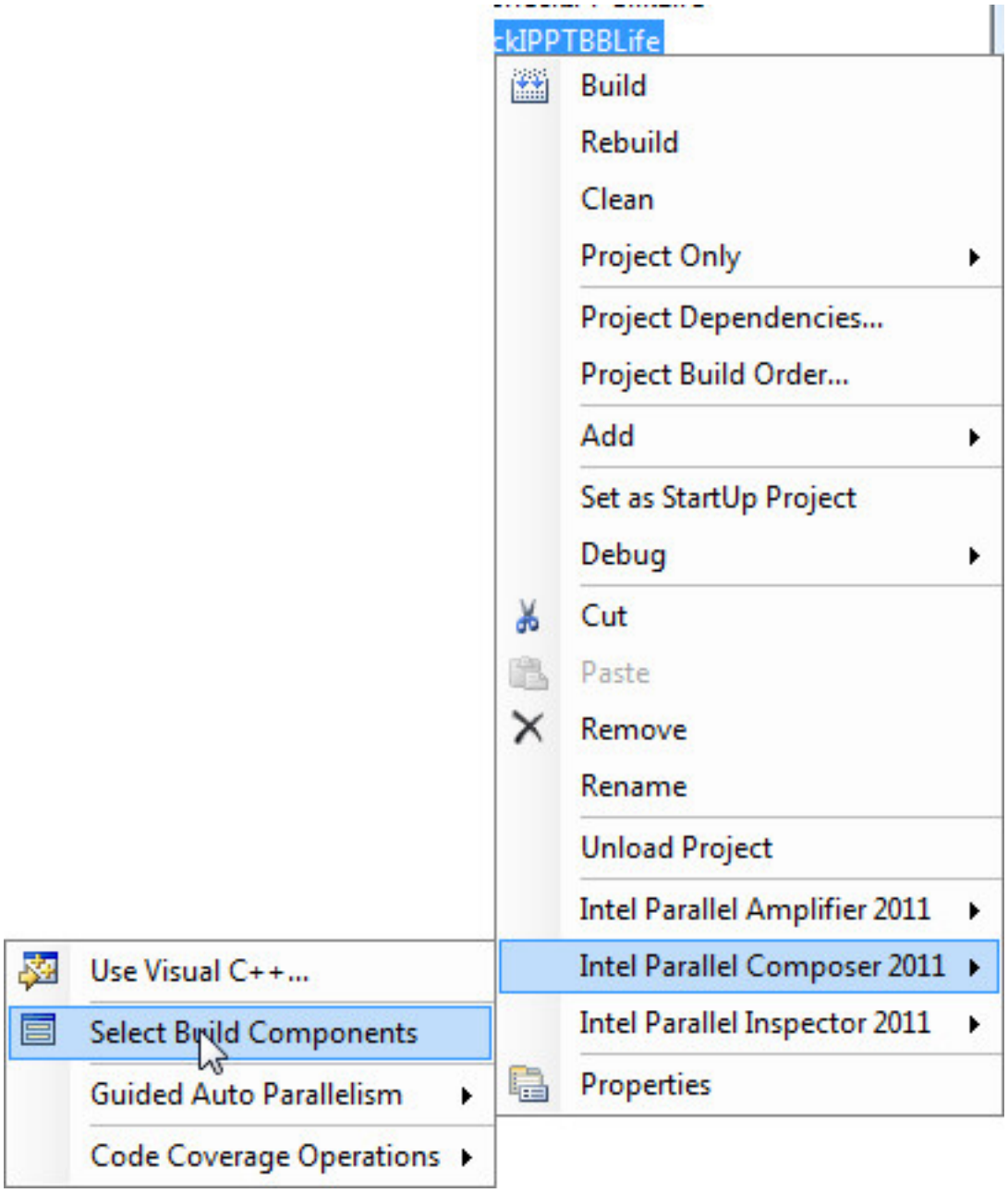
The following are the workflow steps:

- Integrating Intel IPP into your project
- Using Intel IPP functions
- Threading your application

Integrating Intel® IPP into Your Project

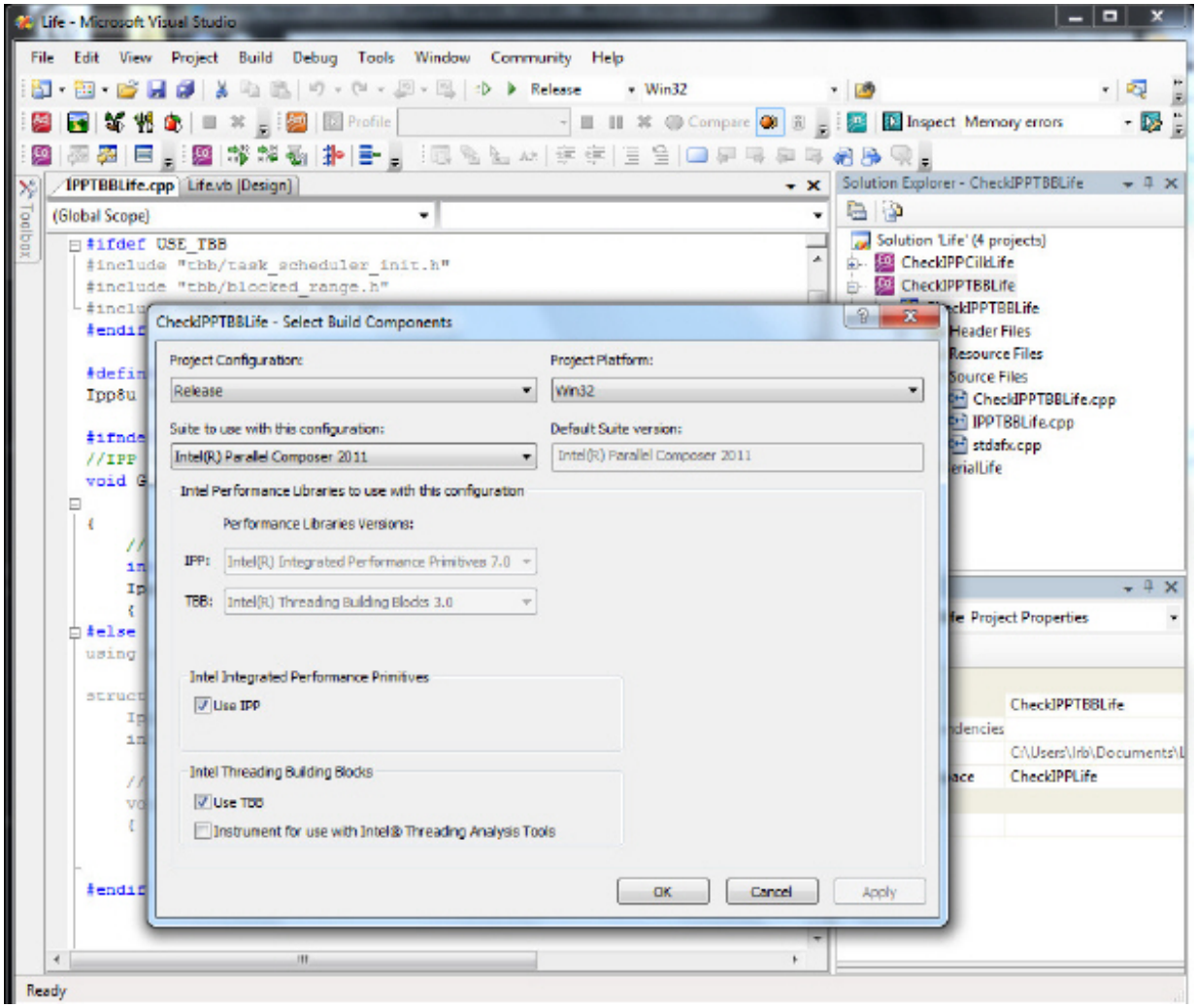
The paths to Intel® Integrated Performance Primitives (Intel® IPP) and Intel® Threading Building Blocks (Intel® TBB) include files and the names and paths of the libraries to link to. These files must be available in the project. You can use the Intel Parallel Composer to make these files available in the project.

To display the dialog box and add Intel IPP or Intel TBB to a project, right click on a project name in the Solution Explorer. Select **Intel Parallel Composer 2011** and then **Select Build Components** menu item.



When the **Select Build Components** dialog box appears, click on the check boxes for **IPP** and/or **TBB** to add the relevant paths and linkage information to your project.

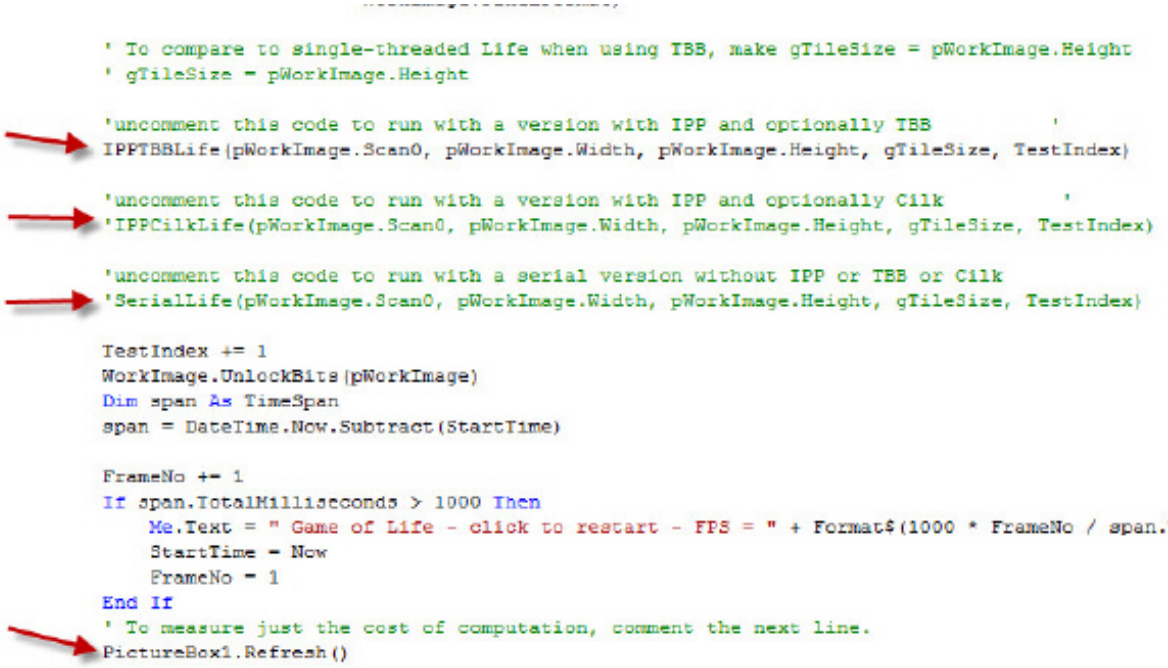
For example, right click on the **CheckIPPTBBLife** project name; select the **Intel Parallel Composer 2011**, and then the **Select Build Components** menu item. Note that Intel IPP and Intel TBB have already been added to the project since the check boxes are checked.



Using Intel® IPP Functions

To see the correct output, in the `Life.vb` check the following:

1. The `IPPTBBLife` function located around line 48 is uncommented
2. The functions `IPPCilkLife` and `SerialLife` are commented out
3. The function `PictureBox1.Refresh()` is uncommented



```
' To compare to single-threaded Life when using TBB, make gTileSize = pWorkImage.Height
' gTileSize = pWorkImage.Height

'uncomment this code to run with a version with IPP and optionally TBB          '
IPPTBBLife(pWorkImage.Scan0, pWorkImage.Width, pWorkImage.Height, gTileSize, TestIndex)

'uncomment this code to run with a version with IPP and optionally Cilk          '
'IPPCilkLife(pWorkImage.Scan0, pWorkImage.Width, pWorkImage.Height, gTileSize, TestIndex)

'uncomment this code to run with a serial version without IPP or TBB or Cilk
'SerialLife(pWorkImage.Scan0, pWorkImage.Width, pWorkImage.Height, gTileSize, TestIndex)

TestIndex += 1
WorkImage.UnlockBits(pWorkImage)
Dim span As TimeSpan
span = DateTime.Now.Subtract(StartTime)

FrameNo += 1
If span.TotalMilliseconds > 1000 Then
    Me.Text = " Game of Life - click to restart - FPS = " + Format$(1000 * FrameNo / span.Tota
    StartTime = Now
    FrameNo = 1
End If
' To measure just the cost of computation, comment the next line.
PictureBox1.Refresh()
```

The `Game of Life` algorithm specifies that a living cell must have two or three neighbors to stay alive. The Intel® Integrated Performance Primitives (Intel® IPP) function `ippiCompareC_8u_C1R` performs this test on a buffer of cell data instead of one cell at a time. Intel IPP takes advantage of the Intel® Streaming SIMD Extensions (Intel® SSE) when possible, vectorizing and speeding up the operation.

The function `ippiCompareC_8u_C1R` takes a buffer of 8 bit unsigned image data and compares pixels of the source image ROI to a given *value* specified and writes the results to a one-channel image of `Ipp8u` data. If the values, in this case 2, are equal, then the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0. The image ROI is set to be the whole buffer. This buffer is used to store which cells have two neighbors.

In the file `IPPTBBLife.cpp` comment out the second function `ippiCompareC_8u_C1R` that appears approximately at line 62 and then run the program. You will notice that the visual cellular activity decreases significantly.

```
        // only cells with 2 or 3 neighbors can live
        ippiCompareC_8u_C1R(pTmp2, Stride, 3, pTmp3, Stride, roi, ippCmpEq);
//      ippiCompareC_8u_C1R(pTmp2, Stride, 2, pTmp2, Stride, roi, ippCmpEq); //lesson
        ippiOr_8u_C1R(pTmp3, Stride, pTmp2, Stride, pTmp4, Stride, roi); // pTmpImage4 is both 2 & 3
```

Later on in the code we use the Intel IPP function `ippiAnd_8u_C1IR` to compare the current living cells with the information about which have two or three neighbors.

The `ippiAnd_8u_C1IR` function performs a bitwise `AND` operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. This function is used to keep alive (set to 1) the pixels that were alive and had two or three neighbors. Again the image ROI is set to be the whole buffer. Where possible, Intel IPP uses the Intel Streaming SIMD Extensions, vectorizing and speeding up the operation.

In the file `IPPTBBLife.cpp` comment out the `ippiAnd_8u_C1IR` function that appears approximately at line 69 and then run the program. You will notice that the visual cellular activity does not progress.

```
        // pTmpImage4 is currently living cells that also had 2 or 3 neighbors
//      ippiAnd_8u_C1IR(pNext, Stride, pTmp4, Stride, roi); //lesson
        // invert the map of living cells
```

The `ippiDup_8u_C1C3R` function copies a one-channel (gray scale) image to each channel of the three-channel (color) image. In this case this function copies the new Life cell state to the output buffer so it can be displayed.

In the file `IPPTBBLife.cpp` comment out the `ippiDup_8u_C1C3R` function that appears approximately at line 76 and then run the program. You will notice that the output window is blank.

```
        ippiOr_8u_C1R(pTmp4, Stride, pTmp3, Stride, pNext, Stride, roi);
    //  ippiDup_8u_C1C3R(pNext, Stride, pDst, Channels * Stride, roi); //lesson
    }
};
```

Threading Your Application

Intel® Integrated Performance Primitives (Intel® IPP) functions are thread safe. It means that you can thread your application and call Intel IPP functions without synchronization problems if the same functions are called by different threads. Intel® Threading Building Blocks (Intel® TBB) and Intel® Cilk™ Plus are two threading technologies that you can use to take advantage of multi-core architectures.

Application without Threading

1. To run serially the **Game of Life** application, comment out the `#define USE_TBB` statement that appears approximately around line 5 in the file `IPPTBBLife.cpp`. This will cause .

```
//comment out this line to use only IPP
//#define USE_TBB
```

2. Launch the Windows `Task Manager` and click on the **Performance** tab. See the **CPU Usage** section. This reading includes the CPU cycles for running the application and displaying the image. To get the CPU usage numbers for just the **Game of Life** application, without the cycles consumed for the display, comment out the call to `PictureBox1.Refresh()` around line 68 in `Life.vb`.

```
' To measure just the cost of computation, comment the next line.
'PictureBox1.Refresh()
```

3. Run the application and note the CPU Usage percentage.

Threading the Application with Intel TBB

1. In the file `IPPTBBLife.cpp` uncomment the `#define USE_TBB` statement that appears approximately at line 5. This will thread the **Game of Life** application using Intel TBB.
2. Run the application and note the **CPU Usage** percentage.

The **CPU Usage** percentage should be significantly higher than in the serial case for a multi-core system, because Intel TBB will split the work across multiple processors. A higher **CPU Usage** percentage indicates more work is being done per unit of time, which means that application performance is better.

Threading the Application with Intel Cilk Plus

Do the following to see how Intel Cilk Plus can be used to thread the **Game of Life** application.

1. Uncomment the `IPPCilkLife` function that appears approximately at line 51 in `Life.vb`
2. Comment out the functions `IPPTBBLife` and `SerialLife`
3. Comment out the `#define USE_CILK` statement that appears approximately at line 5 in the file `IPPClkLife.cpp`. This will cause the **Game of Life** application to run serially.

```
//comment out this line to use only IPP
//#define USE_CILK
```

4. Launch the Windows `Task Manager`, click on the **Performance** tab, run the application and note the **CPU Usage** percentage.
5. Uncomment the `#define USE_CILK` statement that appears approximately at line 5 in the file `IPPTBBLife.cpp`. This will thread the **Game of Life** application using Intel Cilk Plus.
6. Run the application and note the **CPU Usage** percentage.

The **CPU Usage** percentage should be significantly higher than in the serial case for a multi-core system, because Intel Cilk Plus will split the work across multiple processors. A higher **CPU Usage** percentage indicates more work is being done per unit of time, which means that application performance is better.

Game of Life - Parallelization Approaches

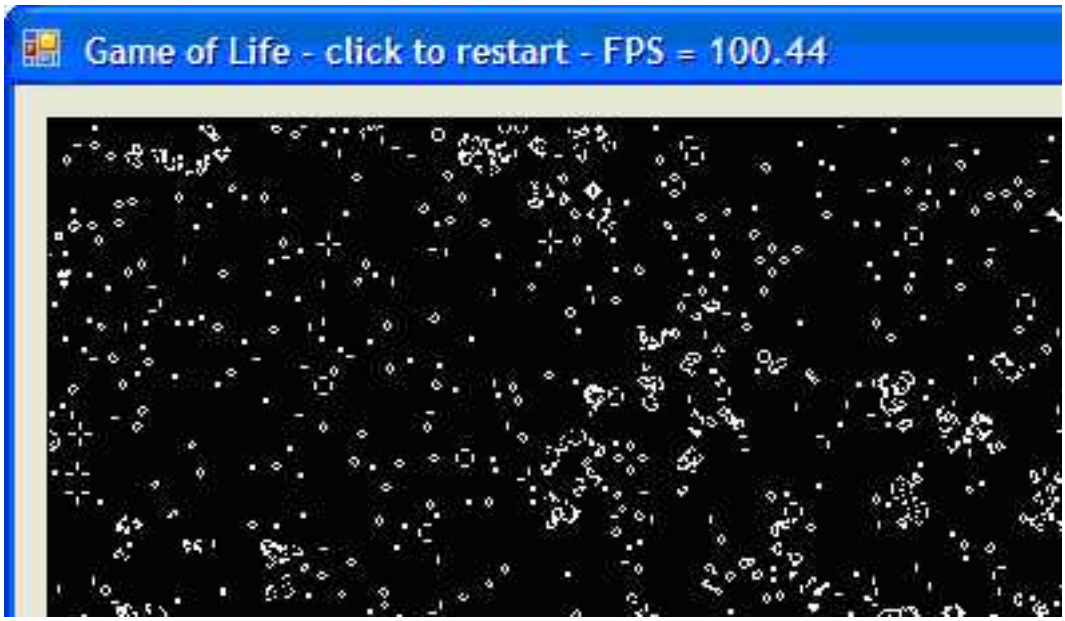
This section provides step-by-step instructions on how to convert the **Game of Life** application from a serial application to a parallel application using different approaches: Intel® Integrated Performance Primitives (Intel® IPP) and Intel® Threading Building Blocks (Intel® TBB) or Intel® Cilk™ Plus. The serial project provides baseline performance.

Follow the steps below to build the serial and threaded **Game of Life** applications.

Serial Project

1. From the directory folder, double-click on the solution file `Life.sln` to open it in Microsoft Visual Studio*.
2. Set **CheckSerialLife** as the **StartUp** project.
3. View the code for `Life.vb` in the **Life** project.
4. Comment the call to `IPPTBBLife` and `IPPCilkLife`.
5. Uncomment the call to `SerialLife` at approximately line 54.
6. Uncomment the call to `PictureBox1.Refresh()` at approximately line 68.

7. From the **Build >...** menu select **Release** to build a release project.
8. Run the `Life.exe` application.
9. The display window looks similar to this:



10. Start **Task Manager** and record the CPU usage. This reading includes the CPU cycles for running the application and displaying the image.

To get the CPU usage numbers for just the **Game of Life** application, without the cycles consumed for the display, perform the following steps:

1. Comment out the call to `PictureBox1.Refresh()`.
2. From the **Build >...** menu select **Release** to build a release project.
3. Run the `Life.exe` application.
4. Start **Task Manager** and record the CPU usage. This provides a baseline for CPU usage.

Vectorized and Threaded with Intel TBB

Create vectorized version:

1. Set **CheckIPPTBBLife** as the **StartUp** project.
2. View the code for `Life.vb` in the **Life** project.
3. Uncomment the call to `IPPTBBLife` at approximately line 48.
4. Uncomment the call to `PictureBox1.Refresh()`.
5. Comment the call to `SerialLife` and `IPPCilkLife`.
6. Open the file `IPPTBBLife.cpp`.
7. Comment out the define for `USE_TBB`.

8. From the **Build >...** menu select **Release** to build a release project.
9. Run the `Life.exe` application.
10. Start **Task Manager** and record the CPU usage. This reading includes the CPU cycles for running the application and displaying the image.

To get the CPU usage numbers for just the **Game of Life** application, without the cycles consumed for the display, perform the following steps:

1. Comment out the call to `PictureBox1.Refresh()`.
2. From the **Build >...** menu select **Release** to build a release project.
3. Run the `Life.exe` application.
4. Start **Task Manager** and record the CPU usage. This provides a baseline for CPU usage.

Is there better CPU usage than in the serial version? Intel IPP provides good vectorization. It means it takes advantage of the Intel® Streaming SIMD extensions, which increases CPU usage.

Parallelize the vectorized version:

1. Open the file `IPPTBBLife.cpp`.
2. Uncomment the define for `USE_TBB`.
3. From the **Build >...** menu select **Release** to build a release project.
4. Run the `Life.exe` application.
5. Record the frames per second and compare to the Intel IPP only version. This provides a baseline for IPP plus TBB performance.

To get the CPU usage numbers for just the **Game of Life** application, without the cycles consumed for the display, perform the following steps:

1. Comment out the call to `PictureBox1.Refresh()`.
2. From the **Build >...** menu select **Release** to build a release project.
3. Run the `Life.exe` application.
4. Start **Task Manager** and record the CPU usage. This provides a baseline for Intel IPP plus Intel TBB CPU usage.

Is there better CPU usage than in either the serial or Intel IPP only versions? Intel TBB provides effective threading so that all cores are used on the machine, which increases CPU usage.

Vectorized and Threaded with Intel® Cilk™ Plus

Create vectorized and treaded version:

1. Set **CheckCilkLife** as the **StartUp** project.
2. View the code for `Life.vb` in the **Life** project.
3. Uncomment the call to `IPPCilkLife` at approximately line 51.
4. Uncomment the call to `PictureBox1.Refresh()`.
5. Comment the call to `SerialLife` and `IPPTBBLife`.

6. Open the file `IPPCilkLife.cpp`.
7. Uncomment the define for `USE_CILK`.
8. From the **Build >...** menu select **Release** to build a release project.
9. Run the `Life.exe` application.
10. Start **Task Manager** and record the CPU usage. This reading includes the CPU cycles for running the application and displaying the image.

To get the CPU usage numbers for just the **Game of Life** application, without the cycles consumed for the display, perform the following steps:

1. Comment out the call to `PictureBox1.Refresh()`.
2. From the **Build >...** menu select **Release** to build a release project.
3. Run the `Life.exe` application.
4. Start **Task Manager** and record the CPU usage. This provides a baseline for CPU usage.

Is there better CPU usage than in the serial version? Intel IPP plus Intel® Cilk™ Plus provides good vectorization. This means it takes advantage of Intel Streaming SIMD extensions, which increases CPU usage. Intel Cilk Plus provides threading so that more cores are used on the machine, which increases CPU usage.

Summary

Congratulations, you have completed the tutorial. Here are a few things you have learned:

- You learned how to add Intel® Integrated Performance Primitives (Intel® IPP) and Intel® Threading Building Blocks (Intel® TBB) or Intel® Cilk™ Plus support to your Microsoft Visual Studio* project.
- You learned how to use Intel IPP to compare each of the elements of one buffer to a constant, to perform a binary AND of the contents of one buffer and another buffer, and to copy the contents of one buffer to another.
- You know two technologies for threading an application that can be used with Intel IPP.
- You understand the difference between using Intel TBB and Intel Cilk Plus.
- You learned how to record the CPU usage.
- You viewed how Intel IPP helps speed up application code.