Getting Started with TBB on Windows

by Dave Vanden Bout

As a technical consultant with my own small company, I cannot count on job security from government intervention—just not enough votes there. In order to keep money coming in, I have to rely on reinvention—by learning new techniques for solving problems.

Introduction

One new area I am exploring is parallel programming. Parallel programming has been around since the 1960s with the four-processor Illiac IV (and maybe longer), but cheap silicon and networking now give millions of people access to parallel computers containing anywhere from a handful to thousands of processors.

I see three main areas of parallel programming that are entering the mainstream, differentiated by their distance from the CPU chip and the amount of parallelism they offer:

1. "**Cloud computing**," where thousands of computers cooperate through the Internet to compute a result. Google's proprietary MapReduce* framework is the standard bearer for this, but there are open source alternatives like Hadoop* and QtConcurrent*.
2. **Graphic processing units (GPUs)**, which use hundreds of processing engines to offload general computational work from the CPU. Nvidia's CUDA Toolkit* and AMD/ATI's CTM* are competing frameworks that support each company's particular chipset.
3. **Multicore CPUs** that divide their workload between two to eight identical computing cores. Intel and AMD provide the multicore CPUs that most people are using today.

In this article, I will concentrate on programming multicore CPUs. There are a growing number of libraries/frameworks for programming multicore architectures (Data Distribution Service, Message Passing Interface, and Concurrency and Coordination Runtime), but I chose Threading Building Blocks (TBB) for the following reasons:

- It is open source, so I can see how it works.
- It is free, so there is no economic barrier to its expansion.
- There is a commercial version, so there may be consulting work to be had.
- It is backed by Intel, so there is some technical and marketing muscle behind it.
- It is on version 2.0, so it has survived the infant mortality phase.

Of course, other frameworks can make many of these same claims. But I had to make a choice and get started, and not sit around waiting for a clear winner. Even if I make the wrong choice, hopefully most of the techniques I learn will be transferable to other frameworks.

In the sections that follow, I will describe how I set up the tools to use TBB on Windows* and used them to create a simple parallel program.

Setting Up TBB

The first thing I did was to download the TBB documentation and the Commercial Aligned Release. I unpacked the source archive (tbb20_014oss_src.tar.gz) into the directory C:\llpanorama\TBB (it is not particular about the directory where it is installed). Then I unpacked the Windows binaries (tbb20_014oss_win.tar.gz) and copied the ia32 subdirectory into the top level of the source directory. (There is also a directory for 64-bit Windows, em64t, if you are using that.)

To build TBB and create applications, I downloaded and installed the free Visual C++ 2005 Express Edition* (including the Graphical IDE). Since building TBB requires it, I also downloaded and installed the Microsoft Assembler*.

By default, the Express Edition only supports .NET applications. Since I wanted to build Win32 applications (rebuilding TBB also requires this), I installed the Microsoft Platform SDK for Windows Server 2003 R2*. [Note: to be able to follow the instructions in this article exactly, make sure you download the "Windows Server 2003 R2" version of the Platform SDK. I used this version because it is the version that was specifically designed for use with Visual Studio 2005*. If you install the newer "Windows SDK for Windows Server 2008 and .NET Framework 3.5," then you will have to modify the SDK-related instructions below accordingly.]

So that the Express Edition can find the SDK, I modified the VCProjectEngine.Dll.Express.Config file located in the \vc\vcpackages subdirectory of the Express Edition to include the executable, include, and library directories for my SDK installation. Here is my edited VCProjectEngine.Dll.Express.Config file (long lines have been broken into multiple lines to fit on this page):

```
<?xml version="1.0" encoding="utf-8"?>

<VCPlatformConfigurationFile

      Version="8.00"

      >

    <Platform

              Name="VCProjectEngine.dll"

              Identifier="Win32"

              >

              <Directories


Include="$(VCInstallDir)include;$(VCInstallDir)
PlatformSDK\include;

$(FrameworkSDKDir)include;
```

```
C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Include"



Library="$(VCInstallDir)lib;$(VCInstallDir)
PlatformSDK\lib;

$(FrameworkSDKDir)lib;$(VSInstallDir);$(VSInstall
Dir)lib;

C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Lib"



Path="$(VCInstallDir)bin;$(VCInstallDir)
PlatformSDK\bin;

$(VSInstallDir)Common7\Tools\bin;$(VSInstallDir)
Common7\tools;

$(VSInstallDir)Common7\ide;$(ProgramFiles)\HTML
Help Workshop;

$(FrameworkSDKDir)bin;$(FrameworkDir)$(FrameworkV
ersion);$(VSInstallDir);

C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Bin;$(PATH)"



                      Reference="$(FrameworkDir)$(
FrameworkVersion)"

                      Source="$(VCInstallDir)crt\
src"

               />

       </Platform>

</VCPlatformConfigurationFile>
```

As the "Using Visual C++ 2005 Express with the Microsoft Platform SDK" page states, as an alternative to editing the VCProjectEngine.Dll.Express.Config file you can specify these paths by launching Visual C++ Express Edition, selecting Options, and editing the "Project and Solutions VC++ Directories" settings.

Then I changed the corewin_express.vsprops file in C:\Program Files\Microsoft Visual Studio 8\VC\VCProjectDefaults so that the line

```
AdditionalDependencies="kernel32.lib"
```

becomes

```
AdditionalDependencies="kernel32.lib user32.lib
gdi32.lib winspool.lib

comdlg32.lib advapi32.lib shell32.lib ole32.lib
oleaut32.lib uuid.lib"
```

To enable the Win32 Windows Application, type in the Win32 Application Wizard of the Express Edition, I commented out lines 441-444 in the the AppSettings.htm file located in %ProgramFiles%\Microsoft Visual Studio 8\VC\VCWizards\AppWiz\Generic\Application\html\1033\:

```
// WIN_APP.disabled = true;

// WIN_APP_LABEL.disabled = true;

// DLL_APP.disabled = true;

// DLL_APP_LABEL.disabled = true;
```

Finally, before starting the Express Edition, I deleted the vccomponents.dat file located in the %USERPROFILE%\Local Settings\Application Data\Microsoft\VCExpress\8.0 directory.

In addition to using the Express Edition GUI, I wanted to use the compiler from the command line. I modified the PATH, INCLUDE, and LIB variables in the C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\vsvars32.bat file to include the required Platform SDK paths, as shown below:

```
@set PATH=C:\Program Files\Microsoft Visual
Studio 8\Common7\IDE;

C:\Program Files\Microsoft Visual Studio 8\VC\
BIN;

C:\Program Files\Microsoft Visual Studio 8\
Common7\Tools;

C:\Program Files\Microsoft Visual Studio 8\SDK\
v2.0\bin;

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;

C:\Program Files\Microsoft Visual Studio 8\VC\
VCPackages;

C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Bin;

%PATH%

@set INCLUDE=C:\Program Files\Microsoft Visual
Studio 8\VC\INCLUDE;

C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Include;

%INCLUDE%
```

```
@set LIB=C:\Program Files\Microsoft Visual
Studio 8\VC\LIB;

C:\Program Files\Microsoft Visual Studio 8\SDK\
v2.0\lib;

C:\Program Files\Microsoft Platform SDK for
Windows Server 2003 R2\Lib;

%LIB%
```

I also installed MinGW* make (version 3.80) on my system so I could run the TBB makefile. I renamed the executable to *make.exe* and added the mingw32\bin path to my system PATH environment variable, so I could just enter "make" at a command prompt to launch my builds.

After all that was installed and patched together, I opened a Visual Studio 2005 command prompt window, went to the C:\llpanorama\TBB\tbb20_014oss_src directory, and issued the commands:

```
C:> vsvars32.bat

C:> make
```

The first command initialized all the environment variables for the command-line version of the compiler. The second started the build process for the TBB libraries, which compiled and ran a suite of test programs. This completed without any errors, so I figured I had set up everything correctly.

Next, I read the Getting Started Guide for TBB. First I created a new system environment variable, TBB20_INSTALL_DIR, and set it to the location where I installed TBB (C:\llpanorama\TBB\tbb20_014oss_src in my case). I also added my ia32\vc8\bin directory to my system PATH so that all the TBB DLLs would be found as needed.

The Getting Started Guide describes how to build the sub_string_finder example application. Rather than type all that in, I just jumped to the directory for this example at ...\tbb20_014oss_src\examples\GettingStarted\sub_string_finder\vc8. Double-clicking the sub_string_finder.sln file brought up the Express Edition window. I built the sub_string_finder project and ran it successfully.

My First TBB Program

For my initial attempt, I wanted to concentrate on the mechanics of creating a TBB application with the Express Edition and not get too bogged down in the details of the algorithm. So I wrote a simple program that just multiplies two vectors item by item and places the products into another, equal-sized vector. I started by cranking up Express Edition and clicked on the New Project button (you can also use File -> New -> "Project..." from the menu). In the New Project window, I selected Win32 as the project type and Win32 Console Application as the template. I gave the project the creative name of "example1" and set its location to the C:\llpanorama\TBB\examples directory. After clicking OK in the New Project window, and then clicking Finish in the Win32 Application Wizard window, a window opened with a simple code skeleton. I made additions to the skeleton and arrived at the following program, which I will explain below.

```
// example1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include "tbb/task_scheduler_init.h"
#include "tbb/parallel_for.h"
#include "tbb/blocked_range.h"
#include "tbb/tick_count.h"
using namespace tbb;
using namespace std;

class vector_mult{
    double *const v1, *const v2;     // multiply these vectors
    double *v3;                      // put the result here
public:
    // constructor copies the arguments into local storage
    vector_mult(double *vec1, double *vec2, double *vec3)
19         : v1(vec1), v2(vec2), v3(vec3) { }
20     // overload () so it does a vector multiply
21     void operator() (const blocked_range<size_t> &r) const {
22         for(size_t i=r.begin(); i!=r.end(); i++)
23             v3[i] = v1[i] * v2[i];
24     }
25 };
26
27 const size_t vec_size = 1000000;
28
29 int _tmain(int argc, _TCHAR* argv[])
30 {
31     // allocate storage for vectors
32     double *v1, *v2, *v3;
33     v1 = (double*) malloc(vec_size*sizeof(double));
34     v2 = (double*) malloc(vec_size*sizeof(double));
35     v3 = (double*) malloc(vec_size*sizeof(double));
36
37     // init and multiply vectors serially and time the operation
38     for(size_t i=0; i<vec_size; i++) { v1[i] = v2[i] = v3[i] = i; }
39     tick_count serial_start = tick_count::now();
40     for(size_t i=0; i<vec_size; i++) { v3[i] = v1[i] * v2[i]; }
41     tick_count serial_end = tick_count::now();
42
43     // reinit and multiply the vectors in parallel and time the operation
44     for(size_t i=0; i<vec_size; i++) { v1[i] = v2[i] = v3[i] = i; }
45     task_scheduler_init init;
46     tick_count parallel_start = tick_count::now();
47     parallel_for( blocked_range<size_t>(0,vec_size,1000),
                         vector_mult(v1,v2,v3) );
48     tick_count parallel_end = tick_count::now();
49
50     // print some of the result vector to make sure it is correct
51     for(size_t i=0; i<10; i++) { cout << v3[i] << endl; }
52
53     // print out the speedup delivered by parallelism
54     cout << "Speedup from parallelism is "
55         << (serial_end - serial_start).seconds()
56         / (parallel_end - parallel_start).seconds() << endl;
57     return 0;
58 }
```

The program starts by allocating some memory for vectors v1, v2, and v3 (lines 32-35). The size of these vectors is set by a constant defined on line 27.

I wanted to multiply the vectors serially at first to get a baseline for how long that took. I initialized the vectors on line 38 so that each element stored its index. The current time is stored on line 39, after which a loop is started on line 40 where each element in v1 and v2 is multiplied and stored in the corresponding location in v3. Finally, the completion time for this loop is stored on line 41. The starting and ending times are used to calculate the time it takes to multiply the vectors serially.

Next the same calculation is performed in parallel. The vectors are reinitialized on line 44. (I did this to erase the product values in v3 so I could be sure any correct values came from the parallel threads and not the preceding serial calculation.) A task scheduler object is then created on line 45. This object manages the scheduling of tasks onto physical threads (cores of the CPU, essentially). You can specify the number of threads when the scheduler object is instantiated, but it is usually best to let it determine the number of threads automatically, in case your application is run on a different type of CPU.

The starting and ending times are recorded on lines 46 and 48 for the parallel loop on line 47. The parallel_for subroutine does the actual creation of tasks that are scheduled for execution on the threads. It takes two arguments. The first is an object that stores the range of the vectors and a grainsize that specifies what size the vectors should be cut into. The second argument is an object whose constructor stores the pointers to the v1, v2, and v3 vectors as private variables (see lines 18-19). What parallel_for does is partition the range of the vectors into non overlapping pieces no larger than the specified grainsize, and then create multiple copies of the vector_mult object and pass one of the subrange pieces to each one. Then the vector_mult objects are scheduled and run in parallel on the physical threads.

When one of the vector_mult objects runs, its () operator is called and passed the subrange that specifies the sections of the vectors it should multiply (see lines 21-24). The loop on line 22 gets the beginning and ending indices of the subrange and just multiplies the vector elements that lie between them. For example, the range beginning and ending might be 51000 and 52000, so the vector_mult object would do the following calculations:

```
v3[51000] = v1[51000] * v2[51000];

v3[51001] = v1[51001] * v2[51001];

...

v3[51999] = v1[51999] * v2[51999];
```

So each vector_mult object computes a small section of the final product vector. The cores just get copies of the vector_mult object and subranges in no particular order and compute the partial results. Since the subranges are non overlapping, there is no danger that the objects are overwriting each other's results. And parallel_for makes sure the entire range is covered so a complete product vector is delivered.

After the parallel_for completes, a few of the product values are printed on line 51 just to make sure the program is doing something correctly. Then the amount of speed-up provided by using parallelism is printed on lines 54-56.

Once the program was written, I had to get it compiled and linked. This required the setting of a few project properties. I selected the Project -> "example1 Properties…" menu item to open the Property Pages window. Then I selected All Configurations from the Configurations drop-down menu and I set the following properties so the compiler and linker could find the TBB header files and libraries:

```
C/C++?General:

   Additional Include Directories =
$(TBB20_INSTALL_DIR)\include

Linker?General:

   Additional Library Directories =
$(TBB20_INSTALL_DIR)\ia32\vc8\lib
```

Then I selected the Debug configuration and set the following properties:

```
Linker?Input:

   Additional Dependencies = kernel32.lib
$(NoInherit) tbb_debug.lib

Build Events?Post-Build Event:

   CommandLine = copy "$(TBB20_INSTALL_DIR)\
ia32\vc8\bin\tbb_debug.dll" "$(OutDir)"
```

These property settings link the debug version of my simple program with the debug version of the TBB library and copy the debug version of the TBB dynamic link library to the directory where the executable file for my application can find it. I then selected the Release configuration and set the corresponding properties using the non-debug versions of the library:

```
Linker?Input:

   Additional Dependencies = kernel32.lib
$(NoInherit) tbb.lib

Build Events?Post-Build Event:

   CommandLine = copy "$(TBB20_INSTALL_DIR)\
ia32\vc8\bin\tbb.dll" "$(OutDir)"
```

Once these properties were all set up, I built the Debug and Release versions by selecting the configuration and then selecting Build -> "Build example1" from the menu bar. Once a version was built, I could run it using the Debug -> Start Without Debugging menu item. I ran both versions ten times and got the following speedups:

Speedup	Debug	Release
	0.48	1.05
	0.48	1.08
	0.50	0.90
	0.47	0.95
	0.49	1.11
	0.50	1.07
	0.50	1.11
	0.50	1.13
	0.49	0.93
	0.50	1.10

In the Debug version, the parallel code runs about half the speed of the serial code because all the diagnostic features of the TBB library are enabled. These features watch out for many of the stupid things I might do while developing code (like setting the number of physical threads to a negative number). I would expect this to have a major impact on the speedup.

However, the speedup in the Release version is really disappointing! The average speedup is just 1.04. In fact, the parallel code is slower than the serial code in 30 percent of the tests. Why is that!? My feeling is that a simple multiplication of vectors does not provide enough computational mass to outweigh the overhead of starting the threads, but I have not run any experiments to investigate this.

As Alexey Kukanov illustrates in his "Why a simple test can get parallel slowdown" blog post, sometimes multithreading a simple problem leads to almost no speedup, or even to a performance slowdown. See my "Let's try this again..." blog post for a subsequent TBB experiment where I applied a more difficult problem to TBB and saw a significant speedup.

Conclusion

At this point, by following my example, you should be able to:

- Set up a development environment for building TBB-based applications.
- Successfully recompile the TBB libraries.
- Use a few TBB constructs like the task scheduler, blocked ranges, and parallel_for.
- Build a program from scratch that uses TBB.

And the best part is, it will not cost you a penny to do any of this.

Obviously, I have only scratched the surface of what TBB can do. As I continue to work with TBB, I will be guided by the following questions:

- What other constructs does TBB provide, and what parallel programming styles do they support?
- What characteristics of a program make it suitable for parallelization?
- If a program does not run well after it is coded for parallel execution, how do I find the bottlenecks?

Resources

- TBB documentation (http://www.threadingbuildingblocks.org/file.php?fid=81)
- TBB Commercial Aligned Release (http://www.threadingbuildingblocks.org/file.php?fid=78)
- Microsoft Visual C++ 2005 Express Edition (http://msdn.microsoft.com/en-us/vstudio/aa700736.aspx)
- Microsoft Assembler (http://www.microsoft.com/downloads/details.aspx?FamilyID=7a1c9da0-0510-44a2-b042-7ef370530c64&displaylang=en)
- Microsoft Platform SDK (http://msdn.microsoft.com/en-us/vstudio/aa700755.aspx)
- MinGW make (version 3.80) (http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=23918)

References

ThreadingBuildingBlocks.org, the official Threading Building Blocks open source site.
"Why a simple test can get parallel slowdown" by Alexey Kukanov, blog post describing how multithreading a simple problem can lead to a performance slow down.
"Let's try this again..." by Dave Vanden Bout, blog post about a Julia set TBB application that shows a 1.8 speedup on a dual-core Athlon* processor.