

Intel® Threading Building Blocks (Intel® TBB) 2.1

In-Depth

Contents

Intel® Threading Building Blocks (Intel® TBB) 2.1	3
Features	3
New in this Release	5
Technical Support	5
Which Intel TBB license is right for your needs?	5

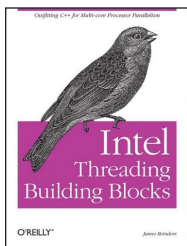
Intel® Threading Building Blocks (Intel® TBB) 2.1

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. Use Intel TBB to implement task-based parallel applications and enhance developer productivity for scalable software on multicore platforms. Intel TBB is the most efficient way to implement parallel applications and unleash multicore platform performance compared with other threading methods like native threads and thread wrappers.

Productivity: Improves developer productivity by using task-based abstractions that make it easier to get scalable and reliable parallel applications with less lines of code (see figure 2). Task-based algorithms, containers and synchronization primitives simplify parallel application development.

Future proof applications: Application performance automatically improves as processor core count increases by using abstract tasks. Sophisticated task scheduler dynamically maps tasks to threads to balance the load among available cores, preserve locality and maximize parallel performance.

Portability: Expand customer base by using a production ready, open solution for parallelism that is available on a broad range of platforms. Available as a commercial and open source project, Intel TBB is coded in C++ and available on a multitude of platforms to provide a cross-platform solution for parallelism. Intel TBB is available as a standalone product or with the Intel® Compiler Professional Editions



Order the Intel® TBB Book at: http://www.amazon.com/Intel-Threading-Building-Blocks-Parallelism/dp/0596514808/ref=pd_bbs_sr_1?ie=UTF8&s=books&qid=1215536190&sr=1-1

Features

This section graphically highlights the key benefits of Intel TBB. For a more comprehensive description of the features, click on the Features & Benefits button below to visit the open source site.

Intel TBB offers comprehensive, abstracted templates, containers and classes for parallelism. Version 2.1 expands usage models and improves performance and usability. Figure 1 highlights the major functional groups within Intel TBB 2.1. Improved and new capabilities are highlighted in yellow. Be sure to go to the New in This Release section for a more detailed description of the new capabilities within Intel TBB 2.1.

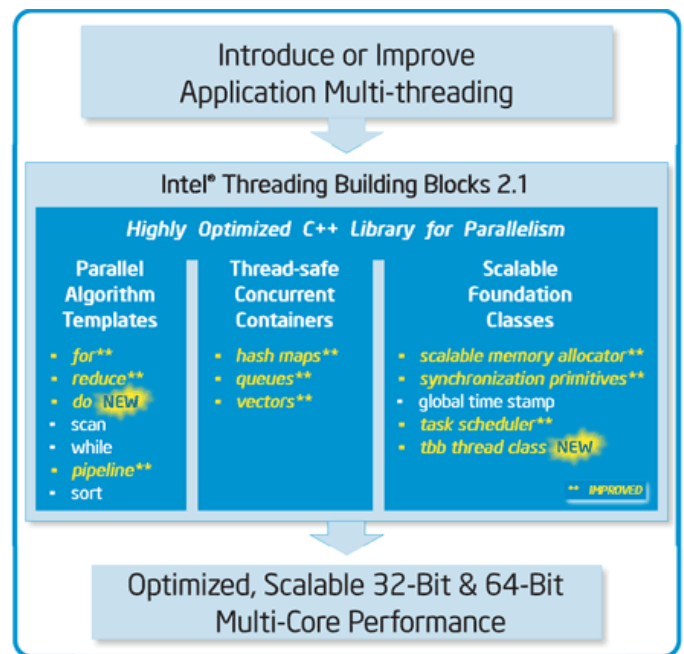


Figure 1: Intel TBB offers comprehensive, abstracted templates, containers and classes for parallelism. Version 2.1 expands usage models and improves performance and usability.

Intel TBB lets developers focus on adding value to their application instead of thread management. Figure 2 highlights a dramatically simpler multi-thread implementation with Intel TBB versus native threads. Intel TBB utilizes robust functions that reduce threading errors like deadlock and race conditions.

76% less code*! Focus on your app, not thread management

Windows* Threads	Intel® Threading Building Blocks
<p>Thread Setup and Initialization</p> <pre> CRITICAL_SECTION MyMutex, MyMutex2, MyMutex3; int get_num_cpus(void) { SYSTEM_INFO si; GetSystemInfo(&si); return (int)si.dwNumberOfProcessors; } int nthreads = get_num_cpus(); HANDLE *threads = (HANDLE *) malloc (nthreads * sizeof (HANDLE)); InitializeCriticalSection (&MyMutex); InitializeCriticalSection (&MyMutex2); InitializeCriticalSection (&MyMutex3); for (int i = 0; i < nthreads; i++) { DWORD id; @thread[i] = CreateThread (NULL, 0, parallel_thread, i, 0, &id); } for (int i = 0; i < nthreads; i++) { WaitForSingleObject (@thread[i], INFINITE); } </pre>	<p>2D Ray Tracing Application</p> <p>Thread Setup and Initialization</p> <pre> #include "tbb/task_scheduler_init.h" #include "tbb/spin_mutex.h" tbb::task_scheduler_init init; tbb::spin_mutex MyMutex, MyMutex2; </pre>
<p>Parallel Task Scheduling and Execution</p> <pre> const int MINPATCH = 150; const int DIVFACTOR = 2; typedef struct work_queue_entry_s { patch pch; struct work_queue_entry_s *next; } work_queue_entry_t; work_queue_entry_t *work_queue_head = NULL; work_queue_entry_t *work_queue_tail = NULL; void generate_work (patch *pch) { int startx, stopx, starty, stopy; int xx,yy; startx=pch->startx; stopx= pch->stopx; starty=pch->starty; stopy= pch->stopy; if((stopx-startx) >= MINPATCH) { int kpatchsize = (stopx-startx)/DIVFACTOR + 1; int jpatchsize = (stopy-starty)/DIVFACTOR + 1; for (j=0; j<jpatchsize; j++) for (k=0; k<kpatchsize; k++) { patch pch; pch.startx = startx + k; pch.starty = starty + j; pch.stopx = MIN(startx+kpatchsize-1, stopx); pch.stopy = MIN(starty+jpatchsize-1, stopy); generate_work (&pch); } } else { /* just trace this patch */ work_queue_entry_t *q = (work_queue_entry_t *) malloc (sizeof work_queue_entry_t); q->pch.startx = startx; q->pch.stopx = stopx; q->pch.starty = starty; q->pch.stopy = stopy; q->next = NULL; if (work_queue_head == NULL) work_queue_head = q; else { work_queue_tail->next = q; } work_queue_tail = q; } } void generate_worklist (void) { patch pch; pch.startx = startx; pch.stopx = stopx; pch.starty = starty; pch.stopy = stopy; generate_work (&pch); } bool schedule_thread_work (patch &pch) { EnterCriticalSection (&MyMutex); work_queue_entry_t *q = work_queue_head; if (q != NULL) { pch = *q->pch; work_queue_head = work_queue_head->next; } LeaveCriticalSection (&MyMutex); return (q != NULL); } generate_worklist (); void parallel_thread (void *arg) { patch pch; while (schedule_thread_work (pch)) { for (int y = pch.starty; y <= pch.stopy; y++) for (int x=pch.startx; x<=pch.stopx; x++) { render_one_pixel (x, y); } if (scene.displaymode == RT_DISPLAY_ENABLED) { EnterCriticalSection (&MyMutex2); for (int y = pch.starty; y <= pch.stopy; y++) { GraphicsDrawRow(startx-1, y-1, totalx, (unsigned char *) &global_buffer[(y-starty)*totalx+(pch.startx-startx)*3]); } LeaveCriticalSection (&MyMutex3); } } } </pre>	<p>Parallel Task Scheduling and Execution</p> <pre> #include "tbb/parallel_for.h" #include "tbb/blocked_range2d.h" class parallel_task { public: void operator() (const tbb::blocked_range2d<int> &r) const { for (int y = r.rows().begin(); y != r.rows().end(); ++y) { for (int x = r.cols().begin(); x != r.cols().end(); x++) { render_one_pixel (x, y); } } if (scene.displaymode == RT_DISPLAY_ENABLED) { tbb::spin_mutex::scoped_lock lock (MyMutex2); for (int y = r.rows().begin(); y != r.rows().end(); ++y) { GraphicsDrawRow(startx-1, y-1, totalx, (unsigned char *) &global_buffer[(y-starty)*totalx*3]); } } } }; parallel_for (tbb::blocked_range2d<int> (starty, stopy + 1, grain_size, startx, stopx + 1, grain_size), parallel_task ()); </pre> <p>This example includes software developed by John E. Stone.</p> <p>* Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations.</p>

Figure 2: Side-by-side comparison of equivalent Windows* thread functionality that requires significantly more code to make a 2D ray tracing program, Tacheon, correctly threaded. Linux* and Mac OS* X developers can expect similar results.

Intel® Threading Building Blocks (Intel® TBB) 2.1

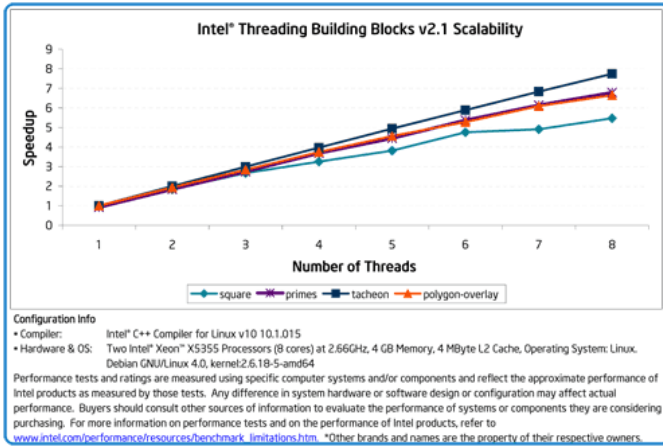


Figure 3. Excellent scalability and improved performance using Intel TBB versus a serial implementation. Linux and Mac OS X developers can expect similar results.

New in this Release

Intel TBB 2.1 offers considerable improvements over version 2.0 through pivotal functionality, performance, and usability enhancements.

Do more: Intel TBB 2.1 enables new use cases by allowing developers to create threads that do not interfere with working tasks. It now has the ability to handle blocking tasks that maximizes processor utilization while handling all the mapping of tasks and `tbb_threads` to raw threads. Now Intel TBB can be applied to cross-platform applications that need the benefit of threading in both computational and interfacing components. Use Intel TBB 2.1 to work with GUIs, AI, I/O operations, and network events without blocking ongoing computation being done by other active tasks.

Do it faster: Intel TBB 2.1 has significantly improved performance on workloads which benefit from static scheduling. Intel TBB has an easy to use affinity mechanism that combines the performance advantage of static scheduling while offering the flexibility of dynamic scheduling. Now the Intel TBB work-stealing task scheduler more efficiently prioritizes work to reduce unnecessary task stealing. Use Intel TBB to abstract from thread maintenance and be assured of great performance independent of the workload on which your algorithm works.

Do it easier: Intel TBB 2.1 makes it easier to use in Microsoft Visual Studio* by offering a compile configuration plug-in. Now Intel TBB makes it simple to configure different compilation variables for multiple development phases including debug and production.

For a more comprehensive description of Intel TBB 2.1 features, click on the New in This Release button below to visit the open source site.

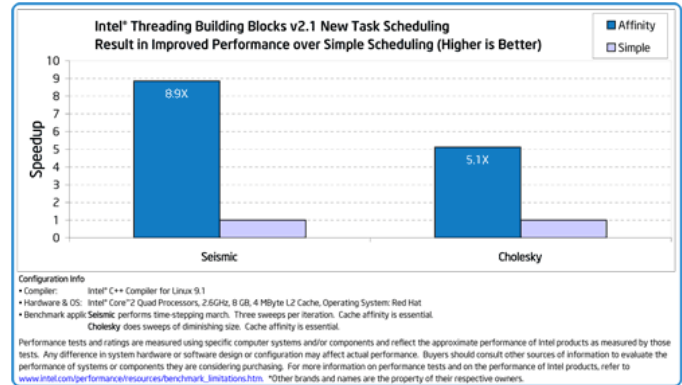


Figure 4. Benchmarks highlight how the auto and task affinity scheduling available in version 2.1 can significantly improve performance over simple task scheduling that was the only choice in the previous version. Developers get performance benefits with minimal or no code changes because of the continued improvements in the library.

Technical Support

With the purchase of Intel TBB, you will receive one year of technical support and product updates from Intel® Premier Support, our interactive issue management and communication web site. This premium support service allows you to submit questions, download product updates, and access technical notes, application notes, and other documentation. For more information, visit the Intel Registration Center at: <http://www.intel.com/software/products/registrationcenter>

Which Intel TBB license is right for your needs?

Intel TBB is available commercially as a binary distribution, and in open source in both source and binary forms. If you need commercial support services you should purchase either a standalone commercial license or take advantage of the considerable value in purchasing the Intel® Compiler Professional Edition. If your legal counsel is comfortable with your use of software under the Intel TBB open source license and you do not require commercial support services, please download the latest versions of open source Intel TBB (<http://threadingbuildingblocks.org>). If you require commercial support services for platforms not supported by Intel TBB please contact us.

When built from source, Intel TBB is intended to be highly portable and so supports a wide variety of operating systems and platforms. Binary distributions, including commercial distributions, are validated and officially supported for the hardware, software, operating systems and compilers listed here.

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

0209/BLA/CMD/PDF 321517-001

