

# インテル® スレッディング・ビルディング・ ブロック (インテル® TBB) 2.2

詳細

## 目次

インテル® スレディング・ビルディング・ブロック (インテル® TBB) 2.2 .....	3
機能と利点 .....	3
本リリースのポイント .....	5
テクニカルサポート .....	6
インテル® TBB ライセンスについて .....	6

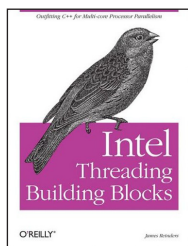
## インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 2.2

インテル® スレッディング・ビルディング・ブロック (インテル® TBB) は、受賞歴のある C++ テンプレート・ライブラリーで、スレッドをタスクに抽象化し、安定性を備えた、移植性とスケーラビリティに優れた並列アプリケーションを作成します。インテル® TBB を使用して、タスクベースの並列アプリケーションを実装することにより、マルチコア・プラットフォームにおけるスケーラブルなソフトウェアを開発する際の生産性を強化できます。ネイティブスレッドやスレッドラッパーなどの他のスレッディング手法の中で、インテル® TBB は並列アプリケーションを実装し、マルチコア・プラットフォームの性能を引き出す最も効果的な手法を提供します。

**生産性:** 少ないコード量でスケーラブルかつ安定性に優れた並列アプリケーションを開発することが可能なタスクベースの抽象化を使用することにより、開発の生産性を向上させます。タスクベースのアルゴリズム、コンテナ、同期プリミティブにより、並列アプリケーションの開発を単純化します。

**将来保証のアプリケーション:** 抽象化タスクを使用することにより、プロセッサのコア数が増えると自動でアプリケーションのパフォーマンスが向上します。洗練されたタスク・スケジューラーが利用可能なコアの負荷のバランスをとりながら動的にタスクをスレッドにマップし、局所性を維持し、並列パフォーマンスを最大限に引き出します。

**移植性:** あらゆるプラットフォーム上で利用可能なプロダクション・レディーの並列化オープンソース・ソリューションを使用することにより、顧客基盤を拡大します。商用版およびオープンソース・プロジェクトとして提供されているインテル® TBB は、C++ でコーディングされ、数多くのプラットフォームで利用でき、並列化のクロスプラットフォーム・ソリューションを提供します。インテル® TBB は、単体製品およびオープンソースとして、またインテル® コンパイラー・プロフェッショナル・エディションおよびインテル® Parallel Studio に同梱された、より完全なコスト効率の高いソリューションです。



インテル® TBB の書籍は以下の Web ページからご購入いただけます。  
<http://www.amazon.co.jp/gp/product/4873113555/>

## 機能と利点

インテル® TBB では、包括的な、並列化のための抽象化されたテンプレート、コンテナ、およびクラスが用意されています。バージョン 2.2 では、使用モデルを広げ、パフォーマンスとユーザビリティが強化されています。図 1 は、インテル® TBB 2.2 の主な機能グループを示しています。インテル® TBB 2.2 の新機能についての詳細は、<http://threadingbuildingblocks.org> (英語) の 2.2 の新機能のセクションを参照してください。

インテル® スレッディング・ビルディング・ブロック 2.2		
<b>高レベルな抽象化</b>		
<b>並列アルゴリズム・テンプレート</b> <ul style="list-style-type: none"> <li>parallel_for</li> <li>parallel_for_each</li> <li>parallel_do</li> <li>parallel_while</li> <li>parallel_invoke</li> <li>parallel_reduce</li> <li>parallel_scan</li> <li>parallel_sort</li> <li>pipeline</li> </ul>	<b>スレッドセーフな並列コンテナ</b> <ul style="list-style-type: none"> <li>concurrent_vector</li> <li>concurrent_queue</li> <li>concurrent_bounded_queue</li> <li>concurrent_hash_map</li> </ul>	
<b>スケラブル・メモリーとタスク管理</b>		
<b>メモリー割り当て</b> <ul style="list-style-type: none"> <li>tbb_allocator</li> <li>cache_aligned_allocator</li> <li>zero_allocator</li> <li>aligned_space</li> </ul>	<b>タスク・スケジューリング</b> <ul style="list-style-type: none"> <li>task_scheduler_init</li> <li>task</li> <li>task_handle</li> <li>task_group</li> <li>structured_task_group</li> <li>tbb_thread</li> </ul>	<b>スレッド・ローカル・ストレージ</b> <ul style="list-style-type: none"> <li>combinable&lt;T&gt;</li> <li>enumerable_thread_specific</li> </ul>
<b>低レベルのプリミティブ</b>		
<b>同期プリミティブ</b> <ul style="list-style-type: none"> <li>atomic&lt;T&gt;</li> <li>mutex</li> <li>queuing_mutex</li> <li>recursive_mutex</li> <li>spin_mutex</li> </ul>	<b>タイミング・プリミティブ</b> <ul style="list-style-type: none"> <li>tick_count</li> </ul>	

図 1: インテル® TBB では、包括的な、並列化のための抽象化されたテンプレート、コンテナ、およびクラスが用意されています。バージョン 2.2 では、使用モデルを広げ、パフォーマンスとユーザビリティが強化されています。

インテル® TBB により、開発者はスレッドの管理ではなく、アプリケーションの価値を高めることに専念できます。図 2 は、ネイティブスレッドと比較して、インテル® TBB によるマルチスレッドの実装がいかに簡単かを示しています。インテル® TBB は、デッドロックや競合状態などのスレッド化エラーを減少させる強固な機能を利用します。

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

Windows* Threads	Intel* Threading Building Blocks
<p><b>Thread Setup and Initialization</b></p> <pre> CRITICAL_SECTION MyMutex, MyMutex2, MyMutex3; int get_num_cpus (void) {     SYSTEM_INFO si;     GetSystemInfo(&amp;si);     return (int)si.dwNumberOfProcessors; } int nthreads = get_num_cpus (); HANDLE *threads = (HANDLE *) malloc (nthreads * sizeof (HANDLE)); InitializeCriticalSection (&amp;MyMutex); InitializeCriticalSection (&amp;MyMutex2); InitializeCriticalSection (&amp;MyMutex3); for (int i = 0; i &lt; nthreads; i++) {     DWORD id;     @threads[i] = CreateThread (NULL, 0, parallel_thread, i, 0, &amp;id); } for (int i = 0; i &lt; nthreads; i++) {     WaitForSingleObject (@threads[i], INFINITE); }                 </pre> <p><b>Parallel Task Scheduling and Execution</b></p> <pre> const int NMPATCH = 150; const int DMFACTOR = 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 xs, ys;     startx = pchin-&gt;startx; stopx = pchin-&gt;stopx;     starty = pchin-&gt;starty; stopy = pchin-&gt;stopy;     if ((stopx-startx) &gt;= NMPATCH) {         int xpatchsize = (stopx-startx)/DMFACTOR + 1;         int ypatchsize = (stopy-starty)/DMFACTOR + 1;         for (ys=starty; ys&lt;=stopy; ys+=ypatchsize) {             for (xs=startx; xs&lt;=stopx; xs+=xpatchsize) {                 patch pch;                 pch.startx = xs;                 pch.starty = ys;                 pch.stopx = MIN(xs+xpatchsize-1, stopx);                 pch.stopy = MIN(ys+ypatchsize-1, stopy);                 generate_work (&amp;pch);             }         }     } else {         /* just trace this patch */         work_queue_entry_t *q = (work_queue_entry_t *) malloc (sizeof         (work_queue_entry_t));         q-&gt;pch.startx = startx; q-&gt;pch.starty = starty;         q-&gt;pch.stopx = stopx; q-&gt;pch.stopy = stopy;         q-&gt;next = NULL;         if (work_queue_head == NULL) {             work_queue_head = q;         } else {             work_queue_tail-&gt;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 (&amp;pch); } bool schedule_thread_work (patch &amp;pch) {     EnterCriticalSection (&amp;MyMutex3);     work_queue_entry_t *q = work_queue_head;     if (q != NULL) {         pch = q-&gt;pch;         work_queue_head = work_queue_head-&gt;next;     }     LeaveCriticalSection (&amp;MyMutex3);     return (q != NULL); } generate_worklist ();  void parallel_thread (void *arg) {     patch pch;     while (schedule_thread_work (pch)) {         for (int y = pch.starty; y &lt;= pch.stopy; y++) {             for (int x = pch.startx; x &lt;= pch.stopx; x++) {                 render_one_pixel (x, y);             }             if (scene.displaymode == RT_DISPLAY_ENABLED) {                 EnterCriticalSection (&amp;MyMutex3);                 for (int y = pch.starty; y &lt;= pch.stopy; y++) {                     GraphicsDrawRow(pch.startx-1, y-1, pch.stopx-pch.startx+1,                     (unsigned char *) &amp;global_buffer[(y-starty)*totalx+(pch.startx-startx)]);                 }                 LeaveCriticalSection (&amp;MyMutex3);             }         }     } }                 </pre>	<p style="text-align: center; font-weight: bold; color: #0070C0;">2D Ray Tracing Application</p> <p><b>Thread Setup and Initialization</b></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><b>Parallel Task Scheduling and Execution</b></p> <pre> #include "tbb/parallel_for.h" #include "tbb/blocked_range2d.h" class parallel_task { public:     void operator() (const tbb::blocked_range2d&lt;int&gt; &amp;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 *) &amp;global_buffer[(y-starty)*totalx]);             }         }     } }; parallel_for (tbb::blocked_range2d&lt;int&gt; (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>

図 2: 等価な Windows スレッド機能との比較 - 2 次元レイ・トレーシング・プログラム、Tacheon を正しくスレッド化するために、Windows スレッド機能ではより多くのコードが必要になることがわかります。Linux\* や Mac OS\* X における開発でも同様の結果が予想されます。

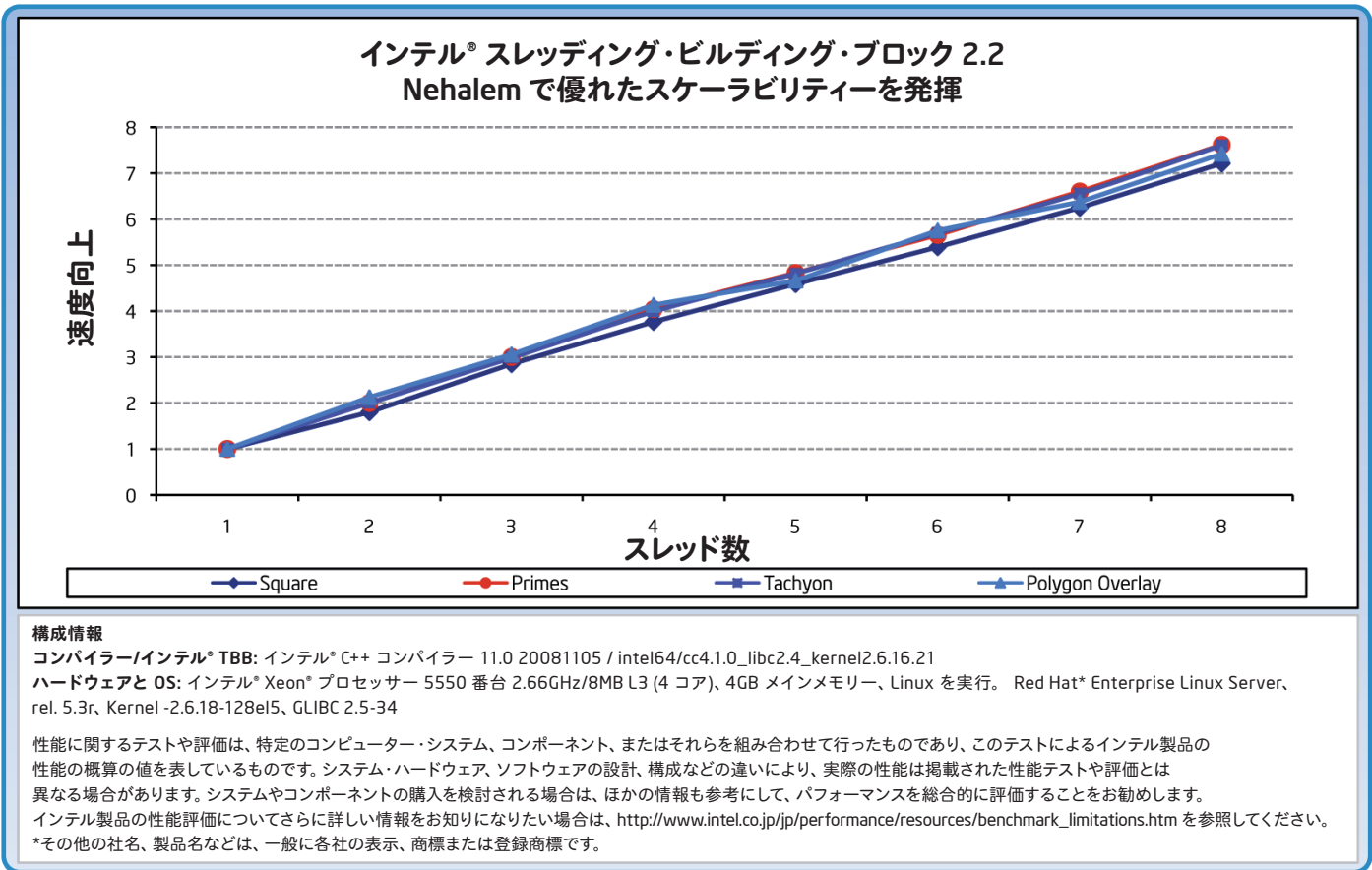


図 3: インテル® TBB と直列化実装 - インテル® TBB を使用することにより、優れたスケーラビリティとパフォーマンスの向上が得られることがわかります。Linux や Mac OS X における開発でも同様の結果が予想されます。

## 本リリースのポイント

インテル® TBB 2.2 では、バージョン 2.1 から機能性、パフォーマンス、ユーザビリティが強化されています。

### パフォーマンスの向上:

- スケラブル・メモリー・アロケーターのパフォーマンス向上
- タスク・スケジューラーの再設計により優れたパフォーマンスとスケーラビリティを提供
- アフィニティー分配のパフォーマンス向上
- ループテンプレートで simple\_partitioner に代わって auto\_partitioner がデフォルト設定

### スケラブル・メモリー・アロケーターの新機能:

- OS アロケーターからスケラブル・メモリー・アロケーターへの自動置換をサポート (Microsoft® Windows® および Linux OS)

### タスク・スケジューラーの向上:

- タスク・スケジューラーの自動初期化サポート (task\_scheduler\_init はオプションになりました)
- タスクグループのサポート

### 並列アルゴリズムの新しい機能と向上した機能:

- 新しい parallel\_invoke アルゴリズムと parallel\_for\_each アルゴリズム
- パイプラインにおける新しいスレッド・バインド・フィルター
- 一般的なループに使用する parallel\_for の簡単なインターフェイス
- ラムダ式サポートの拡張。インテル® C++ コンパイラー 11.0 以降などのラムダ対応コンパイラーの使用時にコードの読み取りや保守がより簡単になりました。

### データコンテナの新しい機能と向上した機能:

- 新しい enumerable\_thread\_specific クラスと combinable クラスでクロスプラットフォームのスレッド・ローカル・ストレージとそのアルゴリズムをサポート
- バインドされていない非ブロッキング・インターフェイスの concurrent\_queue と新しいブロッキング・インターフェイスの concurrent\_bounded\_queue
- concurrent\_hash\_map の簡略化されたインターフェイス
- concurrent\_vector の向上したインターフェイス

## テクニカルサポート

インテル® ソフトウェア開発製品をご購入いただくと 1 年間のサポートサービスが提供されます。このサポートには、インテル® プレミアサポートへのアクセスと製品アップデートが含まれます。インテル® プレミアサポートでは、オンラインでユーザー登録をするだけで、専門家によるテクニカルサポートや製品アップデート、サンプルコード、各種技術ドキュメントなどを入手できます。

### インテル® TBB ライセンスについて

インテル® TBB は、商用 (バイナリー配布) およびオープンソース (ソース形式とバイナリー形式の両方) で利用可能です。商用サポートサービスが必要な場合は、商用の単体ライセンスまたは、インテル® Parallel Studio、インテル® コンパイラー・プロフェッショナル・エディションのいずれかを購入してください。インテル® TBB のオープンソース・ライセンスの利用に法務上の問題がなく、商用のサポートサービスを利用する必要がない場合は、最新のインテル® TBB オープンソース (<http://threadingbuildingblocks.org> (英語)) をダウンロードしてください。また、TBB の商用ソースコードの変更や配布が必要な場合は、インテルの担当者まで詳細をお問い合わせください。

ソースからビルドする場合、インテル® TBB は高度な移植性を意図しているため、多種多様なオペレーティング・システムとプラットフォームをサポートします。商用配布を含むバイナリー配布は、各種ハードウェア、ソフトウェア、オペレーティング・システムで動作が確認され、正式にサポートされています。

