インテル® スレッディング・ビルディング・ブロック

インテル株式会社
ソフトウェア＆ソリューションズ統括部
ソフトウェア製品部

内容

- はじめに
- インテル® スレッディング・ビルディング・ブロック概要
- インテル® TBB によるタスク・ベースの並列プログラミング
- スレッド化の手法
  - インテル® TBB 、 OpenMP* 、明示的なスレッディング
- まとめ

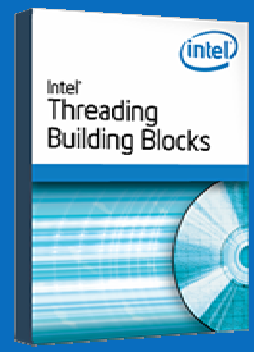
はじめに

- パフォーマンスと正当性の問題からスレッド・プログラミングは難しい
- OpenMP* と OS API を使用したスレッド・プログラミングは主流であるが、共に利点と欠点がある
- インテル® スレッディング・ビルディング・ブロックはまったく新しい並列プログラミング・モデル：チューニングされた並列アルゴリズムとデータ・コンテナをパッケージ化した C++ ライブラリー
- インテル® TBB は特殊なコンパイラーのサポートを必要とせず、さまざまな既存の開発環境で容易に利用可能
- インテル® TBB はスケーラビリティーとパフォーマンスに優れたタスクベースの並列プログラミングの方法論を提唱
- インテル® TBB はスレッド化のパフォーマンスとマルチコア・プラットフォームの価値を高めることを目的とする

インテル® TBB：現在のデュアルコア・プロセッサー向けに
スケーラブルなコードを開発することで、将来コアの増加にも対応

インテル® スレッディング・ビルディング・ブロック

スケーラブルで高速なスレッド



- **概要**
  - C++ のテンプレート・ベースのランタイム・ライブラリーを使用して容易に高性能スレッド・アプリケーションを記述
- **使用方法**
  - 実装：マルチコア・プラットフォームの能力を引き出すマルチスレッド・コードを簡単に実装
  - パフォーマンス：パフォーマンスとスケーラビリティーに優れた共通アルゴリズムを使用
  - 品質：共通の慣用句とコンテナによるパッケージされたライブラリーを採用しバグの混入軽減
  - 設計：タスクとスケーラブルな構造により高度な機能の実装にフォーカス
- **サポート環境**
  - インテル、Microsoft* および GNU* コンパイラー
  - API – OpenMP*、Windows* スレッド、 POSIX* スレッド
  - ツールのサポート – インテル® スレッド・チェッカー、インテル® スレッド・プロファイラー
- **プラットフォーム**

	32 bit (インテル® Core™ 2 Duo, Xeon®, Pentium® プロセッサー...)	Intel 64 (インテル® Core™ 2 Duo, Xeon®, Pentium® プロセッサー ...)	IA-64 (インテル® Itanium® 2 プロセッサー)
Linux*	✓	✓	✓
Mac OS*	✓		
Windows*	✓	✓	

インテル® スレッディング・ビルディング・ブロック概要
C++ のための並列プログラミング・モデル

一般的な並列アルゴリズム
parallel_for
parallel_while
parallel_reduce
pipeline
parallel_sort
parallel_scan

同時実行データ・コンテナ
concurrent_hash_map
concurrent_queue
concurrent_vector

タスク・スケジューラー

低レベル同期プリミティブ
atomic
spin_mutex
queuing_mutex
spin_rw_mutex
mutex

メモリー・アロケーション
cache_aligned_allocator
scalable_allocator

タイミング
tick_count

インテル® TBB はスケーラブルな並列プログラミングのためのクロス・プラットフォーム・フレームワーク

インテル® スレッディング・ビルディング・ブロックによるタスク・ベースの並列プログラミング

- なぜ HPC 分野では OS API によるスレッドが利用されないのか？
  - オーバー・サブスクリプションとアンダー・サブスクリプションの回避が困難
  - OS スケジューラーはスレッドのサービスにオーバーヘッドを要する：スレッドと同期プリミティブはカーネル・オブジェクト
  - OS スケジューラーは公正でプリエンプティブであり、効果的なメモリーやキャッシュの利用効率は期待できない
- 論理タスクとは何か？
  - ユーザーレベルの軽量なタスク制御；インテル® TBB は論理プロセッサーへタスクを割り当てる
  - タスクはプリエンプトされない；インテル® TBB は非公正なスケジューラーであり、キャッシュ中のデータを最も最近利用したタスクを割り当てる
  - インテル® TBB スケジューラーのタスクスチール技術は、オーバー/アンダー・サブスクリプションからプログラムを保護する
- タスク・ベースのアプローチが適応できない場合
  - タスクが多くの I/O や同期をブロックしている場合、インテル® TBB のタスク・スケジューラーは効率的に動作しない

インテル® スレッディング・ビルディング・ブロックのタスク・スケジューラー

- タスク・スケジューラーは高度な並列パターンを持つエンジンであり、スケーラビリティーに優れた検証済みでチューニングされたパッケージから成る
  - *parallel_for*: ループ間で依存性が無いループを負荷分散し並列処理
  - *parallel_reduce*: リダクションを伴う独立したループを負荷分散し並列処理 (例:配列エレメントの合計を求める)
  - *parallel_while*: 不明もしくは動的な領域変更を伴う独立したループを負荷分散し並列処理 (例:リンクされたリストの要素に関数を適用)
  - *parallel_scan*: 並列プリフィックスを計算するテンプレート関数
  - *pipeline*: データの流れのパイプライン・パターン
  - *parallel_sort*: 並列ソート処理を行う

例：ParallelFor を使用した単純なループの並列化

条件：独立したループと明確で固定された範囲

```
const int N = 100000;

void change_array(float array, int M) {
    for (int i = 0; i < M; i++){
        array[i] *= 2;
    }
}

int main (){
    float A[N];
    initialize_array(A);
    change_array(A, N);
    return 0;
}
```

ParallelFor を使用した単純なループの並列化
ステップ 1: TaskScheduler の初期化

```
#include <tbb/TaskSchedulerInit.h>
using namespace ThreadingBuildingBlocks;

const int N = 100000;

void change_array(float array, int M) {
    for (int i = 0; i < M; i++){
        array[i] *= 2;
    }
}

int main (){
    float A[N];
    TaskSchedulerInit init;
    initialize_array(A);
    change_array(A, N);
    return 0;
}
```

アクセスする ThreadingBuildingBlocks ネームスペースを使用する

各スレッドはアルゴリズムを使用する前に、TaskSchedulerInit を作成する

ParallelFor を使用した単純なループの並列化

ステップ 2: ParallelFor テンプレートを使用する

```
#include <tbb/TaskSchedulerInit.h>
#include <tbb/ParallelFor.h>
#include <tbb/BlockedRange.h>
using namespace ThreadingBuildingBlocks;

const int N = 100000;
const int IdealGrainSize = 10000;

void change_array(float *array, int M) {
 ParallelFor (BlockedRange<int>(0, M, IdealGrainSize),
                ChangeArrayBody(array));
}

int main (){
    float A[N];
    TaskSchedulerInit init;
    initialize_array(A);
    change_array(A, N);
    return 0;
}
```

並列化のアルゴリズム

Range オブジェクト。同時実行の必要に応じライブラリーのスケジューラーによって”分割”される

Range に適用するループの本体を記述するユーザー定義の関数オブジェクト

ParallelFor を使用した単純なループの並列化

ステップ 3: ChangeArrayBody クラスを定義

```
...

class ChangeArrayBody {
    float *array;
public:
    ChangeArrayBody (float *a): array(a) {}
    void operator()( const BlockedRange<int>& r ) const{
        for (int i=r.begin(); i!=r.end(); i++ ){
            array[i] *= 2;
        }
    }
};

void change_array(float *array, int M) {
 ParallelFor (BlockedRange<counter>(0, M, IdealGrainSize),
                  ChangeArrayBody(array));
}

...
```

operator() がループの本体を range r へ割り当てる

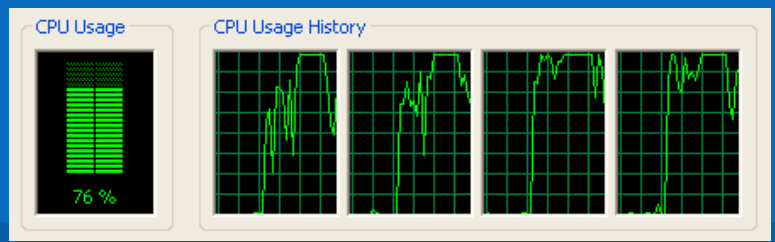
r.begin() から r.end() の範囲を処理

タスクスケジューラが range を分割し必要に応じて本体を複製する

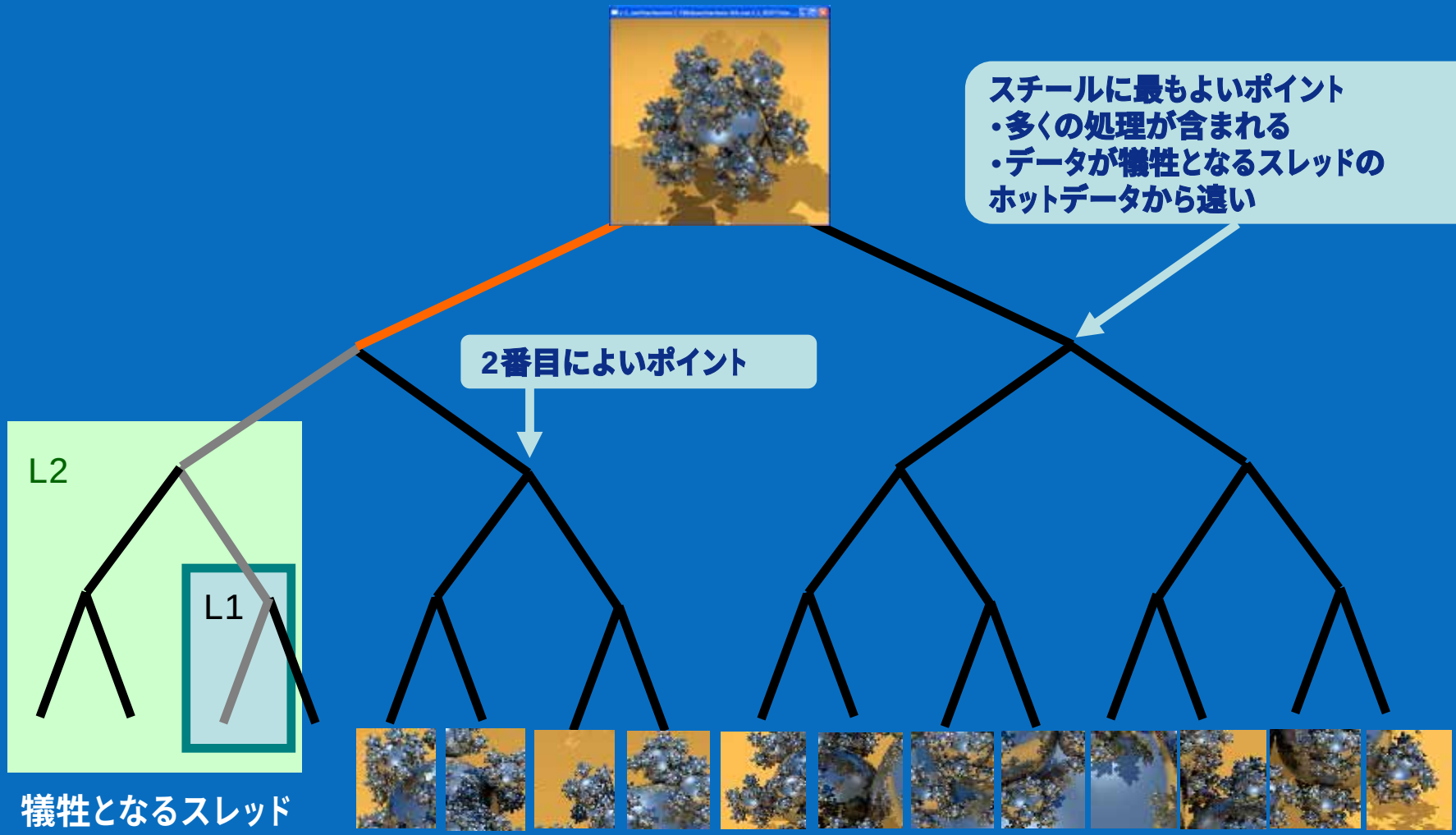
インテル® スレッディング・ビルディング・ブロックのタスク・スケジューラー

- Tacheon はレイ・トレーシングを行い、写真精度のイメージを生成する
- インテル® TBB は並列化に parallel_for の 2D 分解を使用

```
parallel_for body {
    for (x, y) { // 2Dパッチ処理
        render_one_pixel (x, y, … );
    }
    spin_mutex::scoped_lock (io_lock);
    for (y) {
        GraphicsDrawRow (y, …);
    }
}
```



インテル® スレッディング・ビルディング・ブロックの タスク・スケジューラー ：深度優先実行とタスク・スチール



Windows* で Win32 API を使用した場合とのスケーリング比
ベンチマーク：2D Ray Tracing アプリケーション

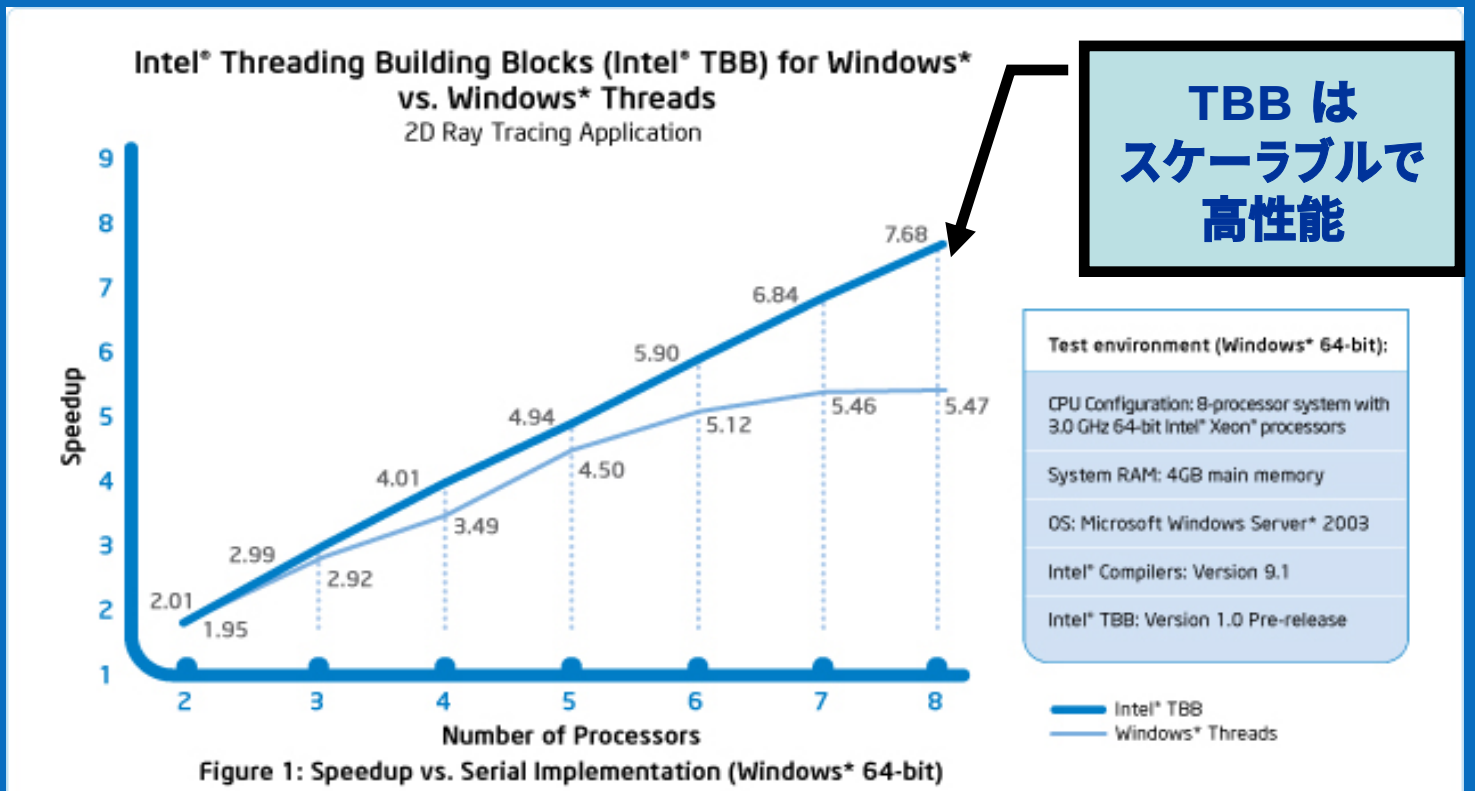


Figure 1: Speedup vs. Serial Implementation (Windows* 64-bit)

TBB は スケーラブルで 高性能

Figure 1 and **Figure 2** show performance on 2-8 cores versus Windows* Threads and POSIX* Threads. Since Intel® Threading Building Blocks is also scalable, no additional coding work is required to benefit from future increases in the number of cores per platform.

開発者はどのようにスレッドを管理するかではなく、どうデータを使用するかに注力する。スレッドの管理はインテル® スレッディング・ビルディング・ブロックに任せることで、優れたスケーラビリティーと性能を確保。

マルチスレッド・アプリケーション開発の容易性

並列化プログラミングでは新たなツールが不可欠

並列化プログラミングにおける3つの課題	インテル® スレッディング・ビルディング・ブロック	OpenMP*	Windows * スレッド	POSIX スレッド
容易なプログラミングと保守	✓			
タスクレベル	✓	✓		
クロスプラットフォーム/複数のOS	✓	✓		✓
C++ 開発サポート	✓		✓	✓
スケーラブルなランタイム・ライブラリー	✓			
現状の開発環境で利用可能	✓		✓	✓
正当性 – 並列処理における同期の問題を解決	✓	✓		
問題の解決 – 事前テストと評価	✓	✓		
利用可能なツール – インテル® スレッド・チェッカー	✓	✓	✓	✓
スケーラビリティー – 将来にわたって利用可能	✓	✓		
パフォーマンスへの要求を満たす	✓	✓		
利用可能なツール –インテル® スレッド・プロファイラー	✓	✓	✓	✓

Fills a strong need

インテル® スレッディング・ビルディング・ブロック vs. 明示的なスレッディング

- 並列プログラミングのパターン
  - **明示的スレッド**：実装の繰り返しが必須
  - **インテル® TBB**：検証済でチューニングされた並列アルゴリズムのパッケージ
- リソースの利用効率
  - **明示的スレッド**：OS のスケジューラーに依存；スレッドの数を正しく制御するのが困難；マニュアルで負荷分散
  - **インテル® TBB**：タスク・スケジューラーが自動的に負荷分散して並列化；オーバー/アンダー・サブスクリプションの回避が容易、性能特性に優れた設計
- 並列化の設計
  - **明示的スレッド**：異なる OS への移植作業が必要；CPU 数の変更に際し、コードの変更が必要となる；スレッドの制御を細かく意識しなければいけない
  - **インテル® TBB**：IA プラットフォーム間での移植は容易；最小限の労力による一度の実装で CPU 数の変更にスケール

インテル® TBB はスケーラビリティーとパフォーマンスを求めて設計された

インテル® スレッディング・ビルディング・ブロックのユニークな機能

- **一般的なアルゴリズム**
  parallel_for, parallel_while, parallel_reduce, parallel_scan, parallel_sort, pipeline
  - 高レベルの一般的な並列アルゴリズムの実装；広範囲のユーザー定義データ・タイプで適用可能
  - 並列処理におけるスケーラブルなデータの分割
- **同時実行データ・コンテナ**
  concurrent_hash_map, concurrent_queue, concurrent_vector
  - STL タイプと同様；きめ細かいロックあり/なしのアルゴリズムを基にする
- **タスク・スケジューラー**
  - C++ のインファーフェースでカスタムな並列モデルを実装可能；入れ子になった並列処理に対応
- **ユーザー・レベルの同期**
  spin_mustex, spin_rw_mutex, queuing_mutex, queuing_rw_mutex
  - ユーザー・レベルのオブジェクトと例外を意識した領域のロック・パターンによる完全な同期プリミティブ
- **メモリー・アロケーター**
  cache_aligned_allocator, scalable_allocator

インテル® TBB は並列処理を設計するための包括的なフレームワーク

製品価格 – インテル® スレッディング・ビルディング・ブロック
開発者ごとのシングルユーザー・ライセンスの場合

インテル® スレッディング・ビルディング・ブロック v1.0 Windows* **版**
商用 43,200円　**アカデミック**　18,500円

インテル® スレッディング・ビルディング・ブロック v1.0 Linux* **版**
商用 43,200円　**アカデミック**　18,500円

インテル® スレッディング・ビルディング・ブロック v1.0 Mac OS X* **版**
商用 43,200円　**アカデミック**　18,500円

Platform Support

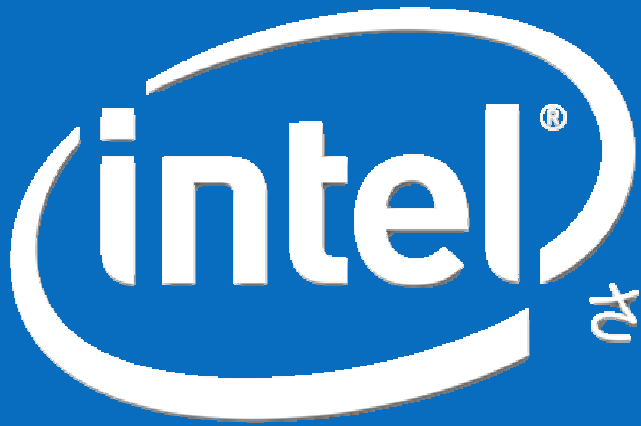
	32 bit (Intel® Core™ 2 Duo, Xeon®, Pentium® processors...)	Intel 64 (Intel® Core™ 2 Duo, Xeon®, Pentium® processors...)	IA-64 (Intel® Itanium® 2 processors)
Linux*	✓	✓	✓
Mac OS*	✓		
Windows*	✓	✓	

2006年8月29日発表、
9月6日パッケージ出荷開始

まとめ

- インテル® スレッディング・ビルディング・ブロックは、C++ 開発者が既存のデュアルコア・プロセッサー向けにスケーラブルなコードを開発することを可能にし、そのコードは将来のコア増加においても優れたスケーラビリティーを可能にする
- インテル® スレッディング・ビルディング・ブロックを利用した設計は簡潔で、競争力のあるパフォーマンスとスケーラビリティーを提供
- インテル® スレッディング・ビルディング・ブロックによる高性能スレッドの開発は OS API を利用したスレッドより容易;インテル® スレッディング・ビルディング・ブロックはインテル® スレッド化ツールに対応
- インテル® スレッディング・ビルディング・ブロックは OpenMP* を補足する;インテル® スレッディング・ビルディング・ブロックは C++ 環境に対応し明示的なスレッドに代わる優れたツール
- インテル® スレッディング・ビルディング・ブロックはパフォーマンスとスケーラビリティーに優れる

インテル® TBB は、開発者が将来のマルチコア・プラットフォームを有効活用することを可能にする – スケーラブルなプログラミングのためのポータビリティーに優れたフレームワーク


intel
さあ、その先へ。™