June 2007

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

マルチコア世代の最高の C++ & Fortran 開発ソリューション

Phil De La Zerda
グローバル・ビジネス・デベロップメント・ディレクター
Intel Corporation

公開が禁止された情報が含まれています。
本資料に含まれるインテル® コンパイラー 10.0
についての情報は、6 月 5 日まで
公開が禁止されています。

マルチコア・プロセッサーがもたらす変革

これまでは … より高速なソフトウェアは高速なプロセッサーによってもたらされた

これからは、性能はマルチコア・プロセッサーによってもらされる

ソフトウェア並列化によりマルチコアの性能を最大限に

並列化によるパフォーマンスの向上

高品質なコードを素早く作成できるように支援

並列プログラミングへの転換を推

ツールの提供

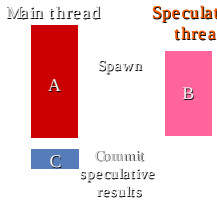
Intel Software Products

構築、
スレッド化、
デバッグ ＆
チューニング

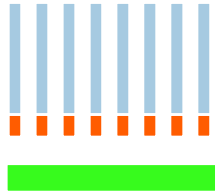
http://http://www.intel.co.jp/jp/software/products/

将来の技術研究

トランザクション・メモリー



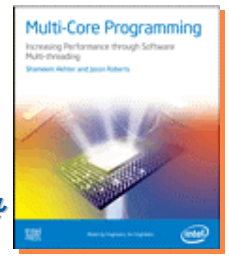
スペキュレーティブ マルチスレッディング



未来の専門家を育成

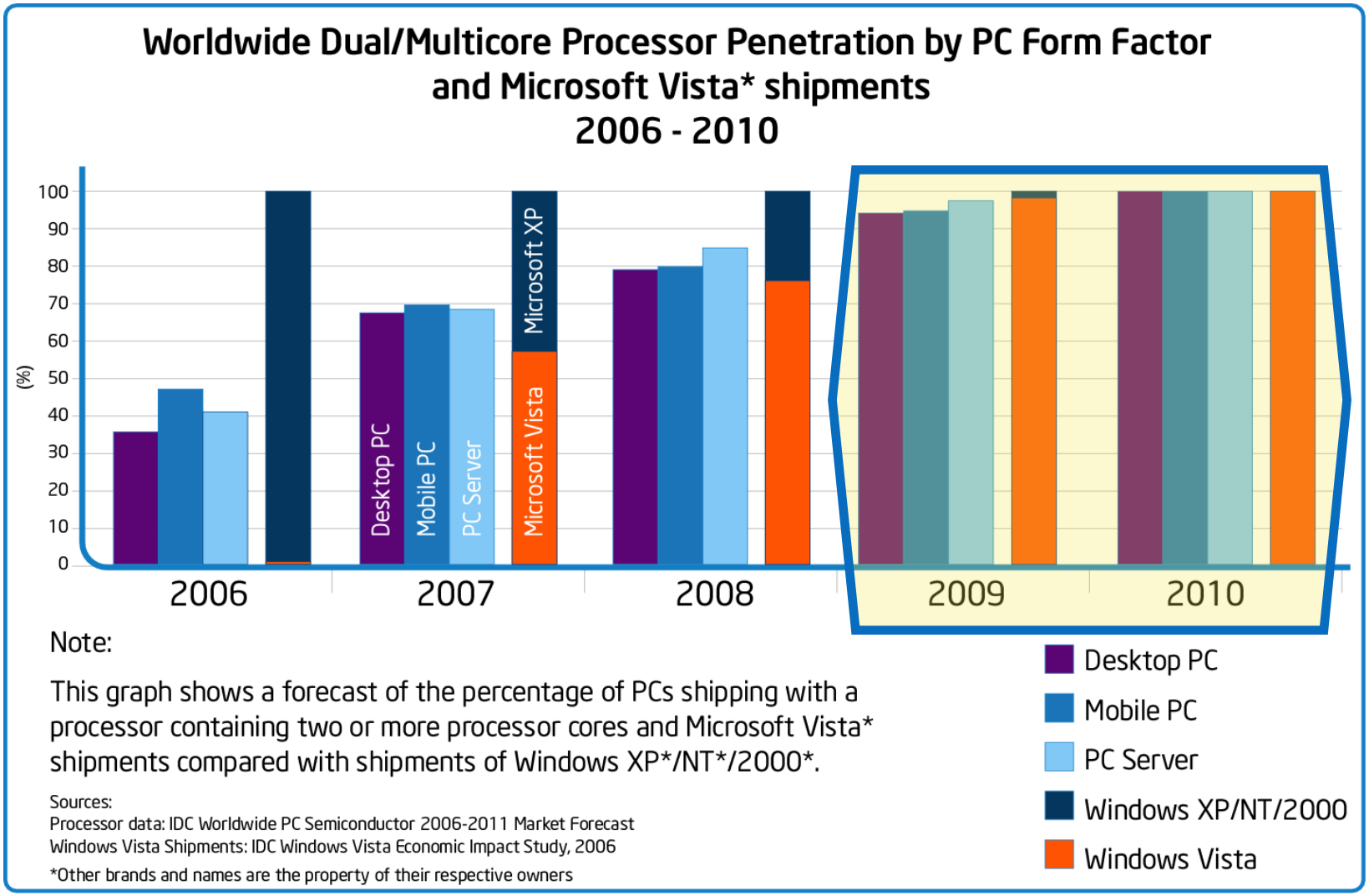
高等教育支援

- *45 の大学で並列プログラミング・コースの開設を支援*
- *7500 人の学生が履修*
- *2007 年の目標:400以上の大学で開設*



重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

世界的な動向と予測



マルチコア処理は新たな標準

並列性の活用は今後パフォーマンスを
実現するための最良の方法

ソフトウェア開発者の挑戦:

“THINK PARALLEL (さもなければ滅びる)”

並列化プログラミングには優れたツールが不可欠

アプリケーションと
システムの動作を可視化

高度に最適化可能な
コンパイラーはスケーラブルな
解決策を提供

潜在的な
プログラミング上の
問題を検出

パフォーマンスと
スケーラビリティーを
チューニング

設計上の解析

並列化の実装

正当性の検証

最適化と
チューニング

マルチコア世代の最高の C++ & Fortran 開発ソリューション

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

並列化はすべてのアプリケーションで必要

支援ツールだけでなく...

"ソリューション" も必要

問題箇所の分析・特定

スレッド化作業

正当性の検証

パフォーマンス・チューニング

クラスター・プログラミング
(作成、チューニング、デバッグ)

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

並列化のステップ 1-2-3

ライブラリーから始め、できるだけ多くのライブラリーを使用
- OpenMP* を推奨
- ライブラリーと OpenMP をサポート

インテル® スレッディング・ビルディング・ブロック (推奨)
- ステップ 1 から 3 へ飛ばないようにする
- インテルによるエキサイティングな新製品
- 業界の活性化を推進

最後のステップ: MPI の使用、手動によるスレッド化
- MPI および手動でのスレッド化をサポート
- (初期からスレッド化を採用しているプログラマー向けに手動での実装と、クラスター・プログラマー向けに MPI を使用した実装をサポート)

並列化のステップ 1-2-3

ライブラリーから始め、できるだけ多くのライブラリーを使用
- OpenMP* を推奨
- ライブラリーと OpenMP をサポート

インテル® スレッディング・ビルディング・ブロック (推奨)
- ステップ 1 から 3 へ飛ばないようにする
- インテルによるエキサイティングな新製品
- 業界の活性化を推進

最後のステップ: MPI の使用、手動によるスレッド化
- MPI および手動でのスレッド化をサポート
- (初期からスレッド化を採用しているプログラマー向けに手動での実装と、クラスター・プログラマー向けに MPI を使用した実装をサポート)

#2 – インテル® スレッディング・ビルディング・ブロック

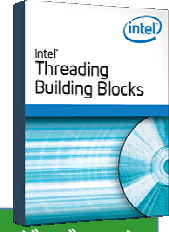
C++ テンプレート・ベースのランタイム・ライブラリー

- マルチスレッド・アプリケーションを簡単に作成
- 高度なプログラミング概念
- スケーラビリティーを重視
- あらかじめビルドしテストされたデータ構造 & アルゴリズム

作業量の少ない方と多い方、どちらを選びますか?

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

わずかなコードの追加で並列化を実現
例: 2 次元レイ・トレーシング・アプリケーション



Windows* スレッド

スレッドセットアップと初期化

```
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 *) alloca (nthreads * sizeof (HANDLE));
InitializeCriticalSection (&MyMutex);
InitializeCriticalSection (&MyMutex2);
InitializeCriticalSection (&MyMutex3);
for (int i = 0; i < nthreads; i++) {
   DWORD id;
   &threads[i] = CreateThread (NULL, 0, parallel_thread, i, 0, &id);}
for (int i = 0; i < nthreads; i++) {
   WaitForSingleObject (&threads[i], INFINITE);
}
```

並列タスクのスケジューリングと実行

```
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* pchin)
{  int startx, stopx, starty, stopy;
   int xs,ys;
   startx=pchin->startx; stopx= pchin->stopx;
   starty=pchin->starty; stopy= pchin->stopy;
   if(((stopx-startx) >= MINPATCH) || ((stopy-starty) >= MINPATCH)) {
     int xpatchsize = (stopx-startx)/DIVFACTOR + 1;
     int ypatchsize = (stopy-starty)/DIVFACTOR + 1;
     for (ys=starty; ys<=stopy; ys+=ypatchsize)
     for (xs=startx; xs<=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 (&pch);}
} else {
     /* just trace this patch */
     work_queue_entry_t *q = (work_queue_entry_t *) malloc (sizeof
(work_queue_entry_t));
     q->pch.starty = starty; q->pch.stopy = stopy;
     q->pch.startx = startx; q->pch.stopx = stopx;
     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 (&MyMutex3);
   work_queue_entry_t *q = work_queue_head;
   if (q != NULL) {
     pch = q->pch;
     work_queue_head = work_queue_head->next;
   }
   LeaveCriticalSection (&MyMutex3);
   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 (&MyMutex3);
       for (int y = pch.starty; y <= pch.stopy; y++) {
         GraphicsDrawRow(pch.startx-1, y-1, pch.stopx-pch.startx+1,
(unsigned char *) &global_buffer[((y-starty)*totalx+(pch.startx-startx))*3]);
       }
       LeaveCriticalSection (&MyMutex3);
     }
   }
}
```

このサンプルには、John E. Stone 氏開発のソフトウェアが含まれています。

インテル® スレッディング・ビルディング・ブロック

スレッドセットアップと初期化

```
#include "tbb/task_scheduler_init.h"
#include "tbb/spin_mutex.h"
tbb::task_scheduler_init init;
tbb::spin_mutex MyMutex, MyMutex2;
```

並列タスクのスケジューリングと実行

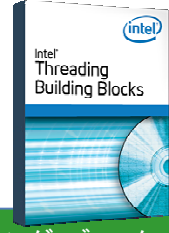
```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range2d.h"
class parallel_task {
public:
   void operator() (const tbb::blocked_range2d<int> &r) const {
     for (int y = r.rows().begin(); y != r.rows().end(); ++y) {
       for (int x = r.cols().begin(); x != r.cols().end(); x++) {
         render_one_pixel (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_task () {}
};
parallel_for (tbb::blocked_range2d<int> (starty, stopy + 1,
grain_size, startx, stopx + 1, grain_size), parallel_task ());
```

スレッドをどう制御するかに注目するのではなく、行うべき処理に注目

インテル® スレッディング・ビルディング・ブロックは、クロスプラットフォームで利用可能な共通 API によって Windows*、Linux* および Mac OS* でのプラットフォーム間でのポータビリティーを提示します。このコード比較は、2D レイ・トレーシング・プログラム (Tacheon) に正しくスレッド化するために必要とされる追加コードを示します。これにより、アプリケーションが現在と将来のマルチコア・ハードウェアを利用することを可能にします。

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

わずかなコードの追加で並列化を実現
例: 2 次元レイ・トレーシング・アプリケーション



Windows スレッド

スレッドセットアップと初期化

```
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 *) alloca (nthreads * sizeof (HANDLE));
InitializeCriticalSection (&MyMutex);
InitializeCriticalSection (&MyMutex2);
InitializeCriticalSection (&MyMutex3);
for (int i = 0; i < nthreads; i++) {
   DWORD id;
   &threads[i] = CreateThread (NULL, 0, parallel_thread, i, 0, &id);}
for (int i = 0; i < nthreads; i++) {
   WaitForSingleObject (&threads[i], INFINITE);
}
```

並列タスクのスケジューリングと実行

```
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* pchin)
{  int startx, stopx, starty, stopy;
   int xs,ys;
   startx=pchin->startx; stopx= pchin->stopx;
   starty=pchin->starty; stopy= pchin->stopy;
   if(((stopx-startx) >= MINPATCH) || ((stopy-starty) >= MINPATCH)) {
     int xpatchsize = (stopx-startx)/DIVFACTOR + 1;
     int ypatchsize = (stopy-starty)/DIVFACTOR + 1;
     for (ys=starty; ys<=stopy; ys+=ypatchsize)
     for (xs=startx; xs<=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 (&pch);}
} else {
     /* just trace this patch */
     work_queue_entry_t *q = (work_queue_entry_t *) malloc (sizeof
(work_queue_entry_t));
     q->pch.starty = starty; q->pch.stopy = stopy;
     q->pch.startx = startx; q->pch.stopx = stopx;
     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 (&MyMutex3);
   work_queue_entry_t *q = work_queue_head;
   if (q != NULL) {
     pch = q->pch;
     work_queue_head = work_queue_head->next;
   }
   LeaveCriticalSection (&MyMutex3);
   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 (&MyMutex3);
       for (int y = pch.starty; y <= pch.stopy; y++) {
         GraphicsDrawRow(pch.startx-1, y-1, pch.stopx-pch.startx+1,
(unsigned char *) &global_buffer[((y-starty)*totalx+(pch.startx-startx))*3]);
       }
       LeaveCriticalSection (&MyMutex3);
     }
   }
}
```

この例は、John E. Stone 氏によって開発されたソフトウェアを使用しています。

インテル® スレッディング・ビルディング・ブロック

スレッドセットアップと初期化

```
#include "tbb/task_scheduler_init.h"
#include "tbb/spin_mutex.h"
tbb::task_scheduler_init init;
tbb::spin_mutex MyMutex, MyMutex2;
```

並列タスクのスケジューリングと実行

```
#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_task () {}
};
parallel_for (tbb::blocked_range2d<int> (starty, stopy + 1,
grain_size, startx, stopx + 1, grain_size), parallel_task ());
```

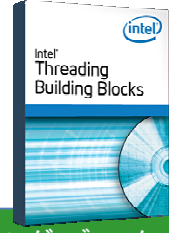
スレッドをどう制御するかに注目するのではなく、行うべき処理に注目

インテル® スレッディング・ビルディング・ブロックは、クロスプラットフォームで利用可能な共通 API によって Windows*、Linux* および Mac OS* でのプラットフォーム間でのポータビリティーを提示します。このコード比較は、2D 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必要な追加コードを示しています。これにより、アプリケーションが現在と将来のマルチコア・ハードウェアを利用することを可能にします。

マルチコア世代の最高の C++ & Fortran 開発ソリューション

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

わずかなコードの追加で並列化を実現
例: 2 次元レイ・トレーシング・アプリケーション



Windows スレッド

スレッドのセットアップと初期化

```
CR...ION MyMutex, MyMutex2, MyMutex3;
int ...id) {

   return (...sors;}
int nthreads = get...
HANDLE *threads = (HA...(nthreads * sizeof (HANDLE));
InitializeCriticalSection (&My...
InitializeCriticalSection (&MyM...
InitializeCriticalSection (&MyMu...
for (int i = 0; i < nthreads; i++) {
   DWORD id;
   &threads[i] = CreateThread (NULL, ...
for (int i = 0; i < nthreads; i++) {
   WaitForSingleObject (&threads[i], INFINI...
}
```

並列タスクのスケジューリングと実行

```
const int MINPATCH = 150;
const int DIVFACTOR = 2;
typedef struct work_queue_entry_s {
   patch pch;
   struct work_queue_entry_s *
} work_queue_entry_t;
work_queue_entry_t *work... = NULL;
work_queue_entry_t *wo... = NULL;
void generate_work (pa...
{  int startx, stopx, sta...
   int xs,ys;
   startx=pchin->st... pchin->stopx;
   starty=pchin->s... = pchin->stopy;
   if(((stopx-start...ATCH) || ((stopy-starty) >= MINPATCH)) {
     int xpatchsiz...startx)/DIVFACTOR + 1;
     int ypatchsi...starty)/DIVFACTOR + 1;
     for (ys=st...topy; ys+=ypatchsize)
     for (xs=s...stopx; xs+=xpatchsize) {
       patch p...
       pch.st...
       pch.s...
       pch...N(xs+xpatchsize-1,stopx);
       pch...IN(ys+ypatchsize-1,stopy);
       ge...k (&pch);}
} else
   /...this patch */
   ...entry_t *q = (work_queue_entry_t *) malloc (sizeof
(work_qu...entry_t));
   q->pch.starty = starty; q->pch.stopy = stopy;
   q->pch.startx = startx; q->pch.stopx = stopx;
   q->next = NULL;
   if (work_queue_head == NULL) {
     work_queue_head = q;
   } else {
     work_queue_tail->next = q...
   }
   work_queue_tail = q;
  }
}
void generate_w...
{
   ...starty;
   ...stopy = stopy;
   generate_work (&pch);
}
bool schedule_thread_work (patch &pch)
{
   EnterCriticalSection (&MyMutex3);
   work_queue_entry_t *q = work_queue_head;
   if (q != NULL) {
     pch = q->pch;
     ...queue_head = work_queue_head->next;
     ...on (&MyMutex3);
   }
   generate_w...

void parallel_thread (void...
{
   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 (&MyMutex3);
       for (int y = pch.starty; y <= pch.stopy; y++) {
         GraphicsDrawRow(pch.startx-1, y-1, pch.stopx-pch.startx+1,
(unsigned char *) &global_buffer[((y-starty)*totalx+(pch.startx-startx))*3]);
       }
       LeaveCriticalSection (&MyMutex3);
     }
   }
}
```

この例は、John E. Stone 氏によって開発されたソフトウェアを使用しています。

インテル® スレッディング・ビルディング・ブロック

スレッドのセットアップと初期化

```
#include "tbb/task_scheduler_init.h"
#include "tbb/spin_mutex.h"
tbb::task_scheduler_init init;
tbb::spin_mutex MyMutex, MyMutex2;
```

並列タスクのスケジューリングと実行

```
#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_task () {}
};
parallel_for (tbb::blocked_range2d<int> (starty, stopy + 1,
grain_size, startx, stopx + 1, grain_size), parallel_task ());
```

スレッドをどう制御するかに注目するのではなく、行うべき処理に注目

インテル® スレッディング・ビルディング・ブロックは、クロスプラットフォームで利用可能な共通 API によって Windows*、Linux* および Mac OS* でのプラットフォーム間でのポータビリティーを提示します。このコード比較は、2D レイ・トレーシング・プログラム (Tacheon) に正しくスレッド化するために必要とされる追加コードを示します。これにより、アプリケーションが現在と将来のマルチコア・ハードウェアを利用することを可能にします。

マルチコア世代の最高の C++ & Fortran 開発ソリューション

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。

わずかなコードの追加で並列化を実現
例: 2 次元レイ・トレーシング・アプリケーション

Windows スレッド

インテル® スレッディング・ビルディング・ブロック

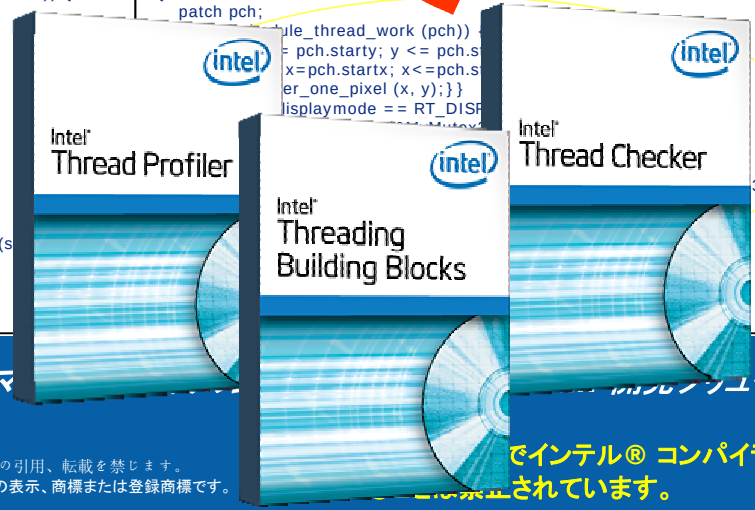
この例は、John E. Stone 氏によって開発されたソフトウェアを使用しています。

明示的なスレッドの管理は不必要

開発と保守に費やす時間は貴重

スレッドをどう制御するかに注目するのではなく、行うべき処理に注目

インテル® スレッディング・ビルディング・ブロックは、クロスプラットフォームで利用可能な共通 API によって Windows*、Linux* および Mac OS* でのプラットフォーム間でのポータビリティーを提示します。このコード比較は、2D レイ・トレーシング・プログラム (Tacheon) に正しくスレッド化するために必要とされる追加コードを示します。これにより、アプリケーションが現在と将来のマルチコア・ハードウェアを利用することを可能にします。



並列化のステップ 1-2-3

ライブラリーから始め、できるだけ多くのライブラリーを使用
- OpenMP* を推奨
- ライブラリーと OpenMP をサポート

インテル® スレッディング・ビルディング・ブロック (推奨)
- ステップ 1 から 3 へ飛ばないようにする
- インテルによるエキサイティングな新製品
- 業界の活性化を推進

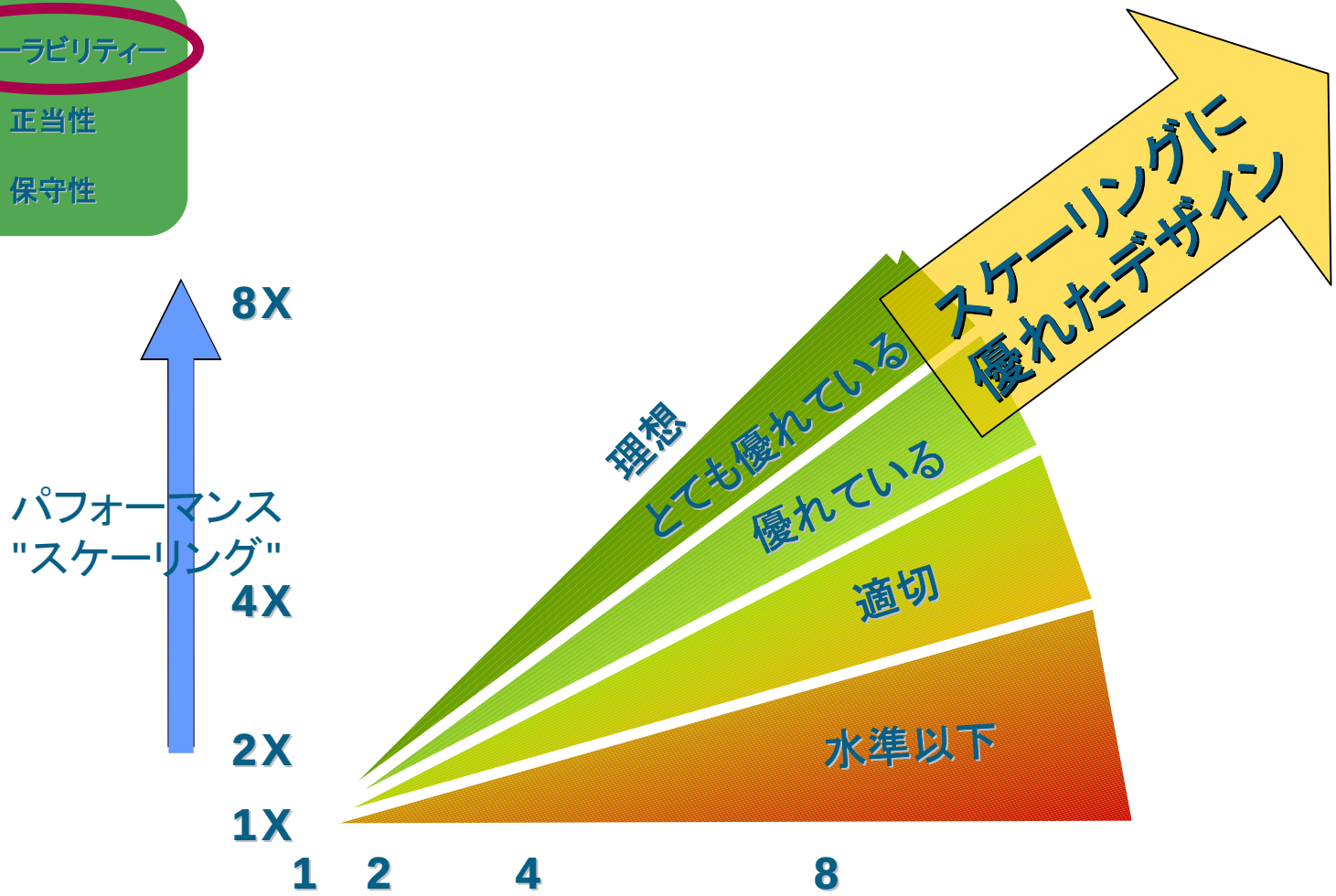
最後のステップ: MPI の使用、手動によるスレッド化
- MPI および手動でのスレッド化をサポート
- (初期からスレッド化を採用しているプログラマー向けに手動での実装と、クラスター・プログラマー向けに MPI を使用した実装をサポート)

スケーラビリティー: 競争力に優れたソフトウェア

スケーラビリティー

正当性

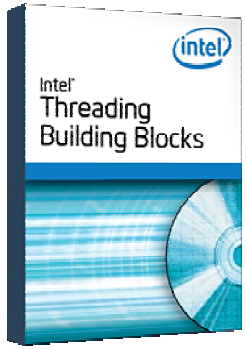
保守性



優れたスケーリングにはインテル® ソフトウェア開発ツール

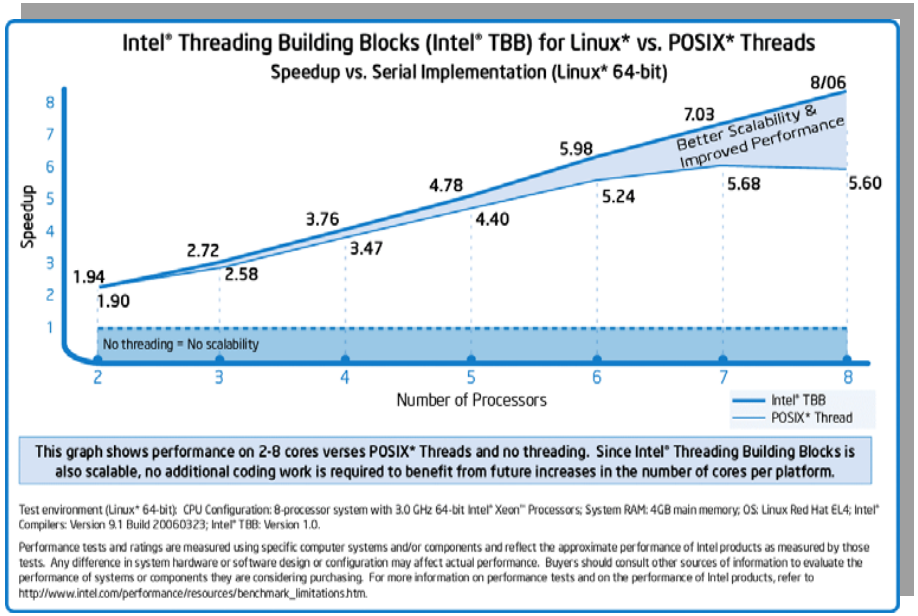
スケーラビリティーの導入:
ライブラリー、OpenMP、MPI

スケーラビリティー
&
ハイパフォーマンス

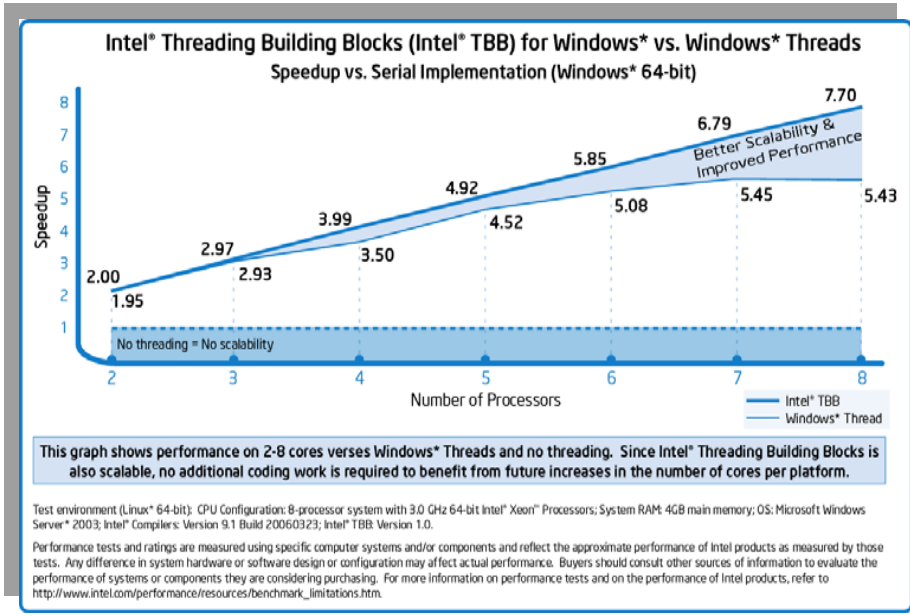


インテル® スレッディング・ビルディング・ブロック

Linux*



Windows*



スレッドをどう制御するかに注目するのではなく、行うべき処理に注目できるように、
インテル® スレッディング・ビルディング・ブロックにスレッドの*向上*をまかせ、
*優れた*スケーラビリティーとパフォーマンスを得ることができました。

重要: 6 月 5 日までインテル® コンパイラー 10.0 について公開することは禁止されています。