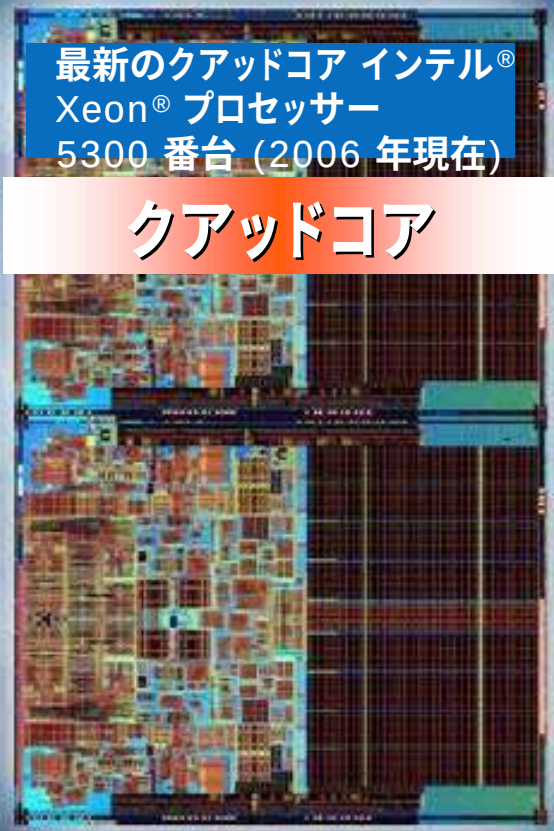
並列化はソフトウェア開発の主流となっています：皆さんは準備できていますか？

Phil De La Zerda
ディレクター
ソフトウェア開発製品部
Intel Corporation

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

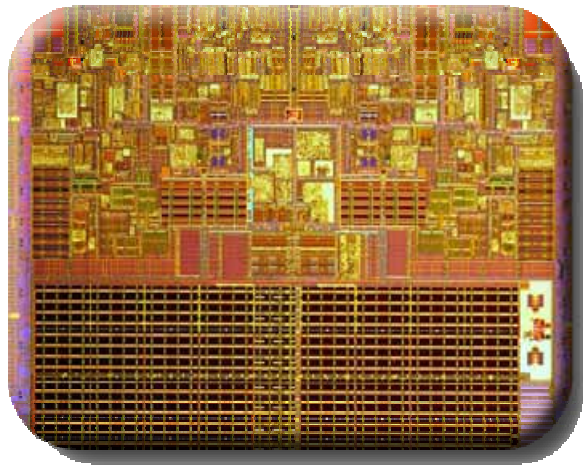
並列化によるパフォーマンスの向上

低消費電力

1 立方メートルあたりの消費電力性能

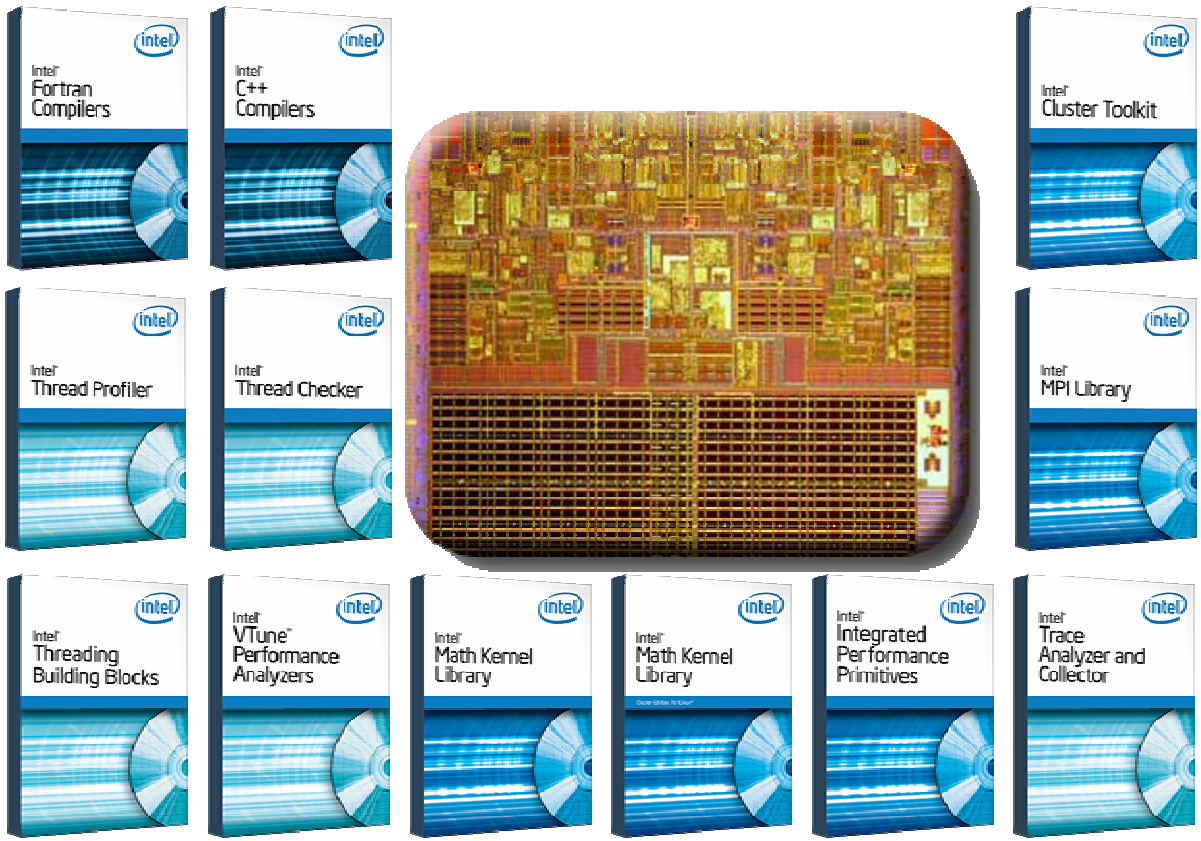
高性能

並列化によるパフォーマンスの向上



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化によるパフォーマンスの向



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化によるパフォーマンスの向

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進

ツール提供

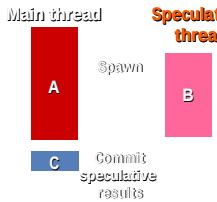
intel Software Products

構築、スレッド化、デバッグ & チューニング

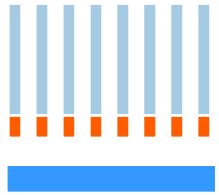
http://http://www.intel.co.jp/jp/software/products/

将来の技術研

トランザクション・メモリー



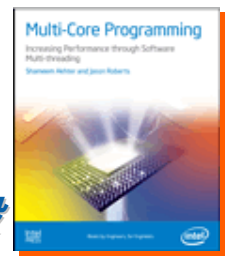
スペキュレーティブ マルチスレッディング



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*



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

新しい使用モデル、鍵は並列化

エンターテインメント

学習 & トラベル

パーソナルメディアの
作成と管理

ヘルスケア

パフォーマンス

TIPS
GIPS
MIPS
KIPS

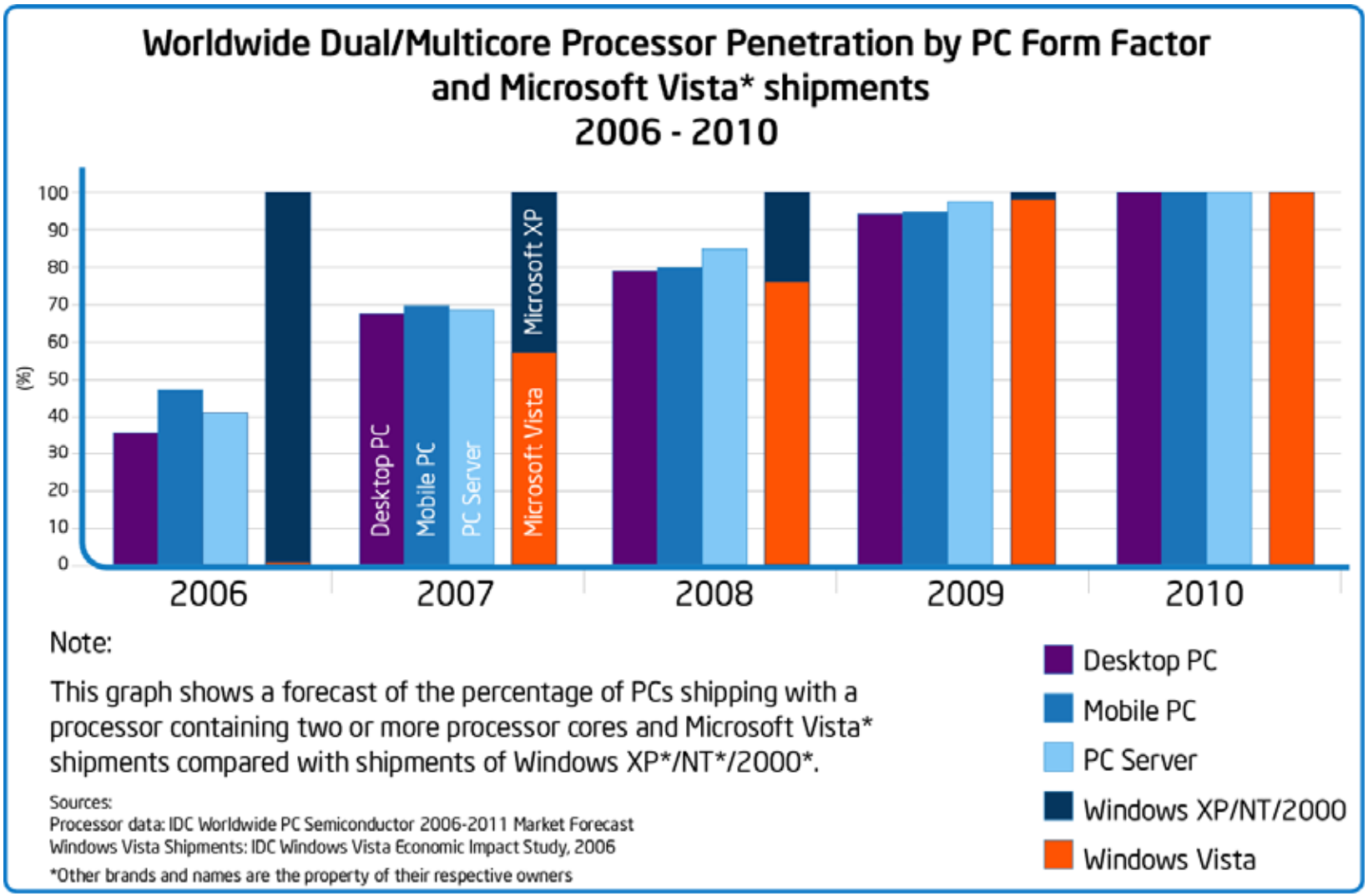
RMS
3D &
ビデオ
マルチ
メディア
テキスト
シングルコア
マルチコア
テラ・スケール

KB
MB
GB
TB

データセット・サイズ

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

世界的な動向と予測



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

キーポイント

進化するにはマルチコアは必須

- 反対論は少数派

ソフトウェア開発者にとってマルチコアはユビキタス:
利用価値があり、無視することは危険

インテルは並列化において開発者を支援する最良の製品を提供

インテルはマルチコアの将来に向けてさまざまな革新を推進

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列コンピューティングはユビキタス

- 今後数年ですべてのコンピューターは並列コンピューターになる
  - サーバー
  - ラップトップ
  - 携帯電話
- ではソフトウェアは ?
  - マイクロソフト社の Herb Sutter 氏が Dr. Dobbs ジャーナルで指摘
    - 無料のランチは終わった：ソフトウェアの基礎は同時性を求めることに変わった
  - ソフトウェアは並列性をなくして、一つの世代の CPU から次の世代の改善された H/W へ移行する際に性能を得ることができない

アプリケーションの性能は ISV の競争力の鍵を握る

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

進化するには並列化は必須

- サーバー、PC およびラップトップ・コンピューターにおける CPU は、メニーコアとなる
  - 現在はデュアルコアとクアッドコア
  - 将来はメニーコアのチップが搭載される
- この移行は物理的進化でドライブされ、避けることはできない
- マルチコアの利点を生かすには、ソフトウェアは並列化されなければいけない

並列化ソフトウェアを作成するにはチャレンジ…
ソフトウェア開発ツールを活用してください !!

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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）”

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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

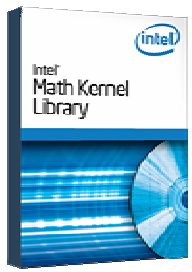
© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

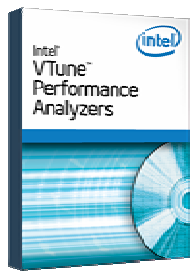
並列化支援ツールの種類

複数のプログラム
または
〝プロセス〞

何ができるか？

それぞれの実行速度を
向上させる例: Oracle データベース





© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化支援ツールの種類

何ができるか？

複数のスレッド
(共有メモリー)

スレッド化 – ライブラリー、OpenMP*、
スレッディング・ビルディング・ブロック。
パフォーマンスのチューニング。
デバッグ (正当性の検証)。
最高の性能を引き出すように各スレッドをコンパイル。

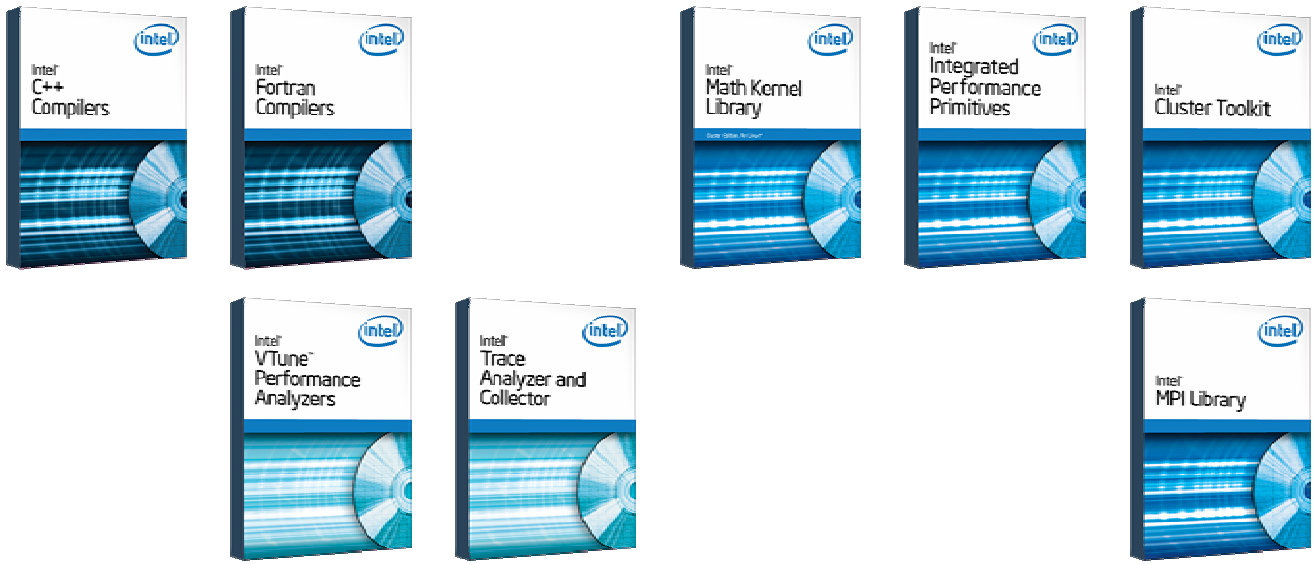
© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化支援ツールの種類

何ができるか？

クラスターノード
(ローカルメモリー)

MPI (Message Passing Interface)
(高速で移植性に優れた) 最高の MPI ライブラリー
パフォーマンスのチューニング。
デバッグ (正当性の検証)。
クラスター・マス・ライブラリー。
最高の性能を引き出すように各スレッドをコンパイル。



intel Software Products

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化支援ツールの種類

クラスター
ノード
(ローカルメモリー)

＋

ノードごとに
複数のスレッド

何ができるか？

すべて – MPI、スレッド化、
パフォーマンス、
チューニング、デバッギングほか

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

OpenMP*、クラスター OpenMP、それとも MPI?

プロセッサー / コアの数	OpenMP (業界標準)	クラスター OpenMP (インテル独自 - OpenMP / MPI のブリッジ)	MPI (業界標準)
2-4	最適 ユビキタス・ハードウェア	クラスターには適さない	クラスターには適さない
4-32			nMP と同じ 、より多くの 必要
32-64	ハードウェア・コストが高くなる	最適	最適
64-128	ハードウェア・コストが高くなる	コードによっては使用可能	最適
128 以上	高性能なマシンでは不可能	調査が必要	これ以外の選択肢はなし

インテルは、これらすべてを支援する
コンパイラー、ライブラリー、分析および検証ツールを提供

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
*その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列化のステップ 1-2-3

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

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

#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?

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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。

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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。

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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

```
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。

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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

```
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);
InitializeCriticalSec
for (int i = 0; i < n
    DWORD id;
    &threads[i] = C
for (int i = 0; i < n
    WaitForSingleO
}
```

並列タスクのスケジュ

```
const int MINPATC
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 (s
(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)
{
                                                    pch)
    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;
                                                    ]);
```

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

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

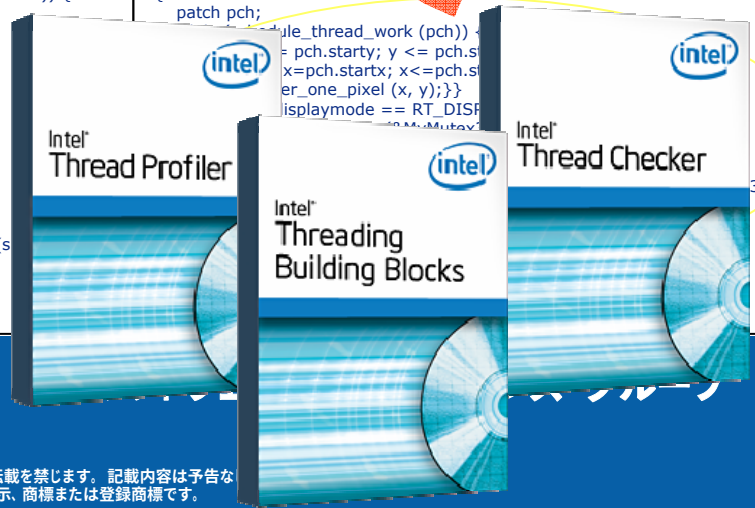
```
#include "tbb/task_scheduler_init.h"
#include "tbb/spin_mutex.h"
tbb::task_scheduler_init init;
tbb::spin_mutex MyMutex, MyMutex2;
```

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。



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告な
*その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

intel

並列化のステップ 1-2-3

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)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

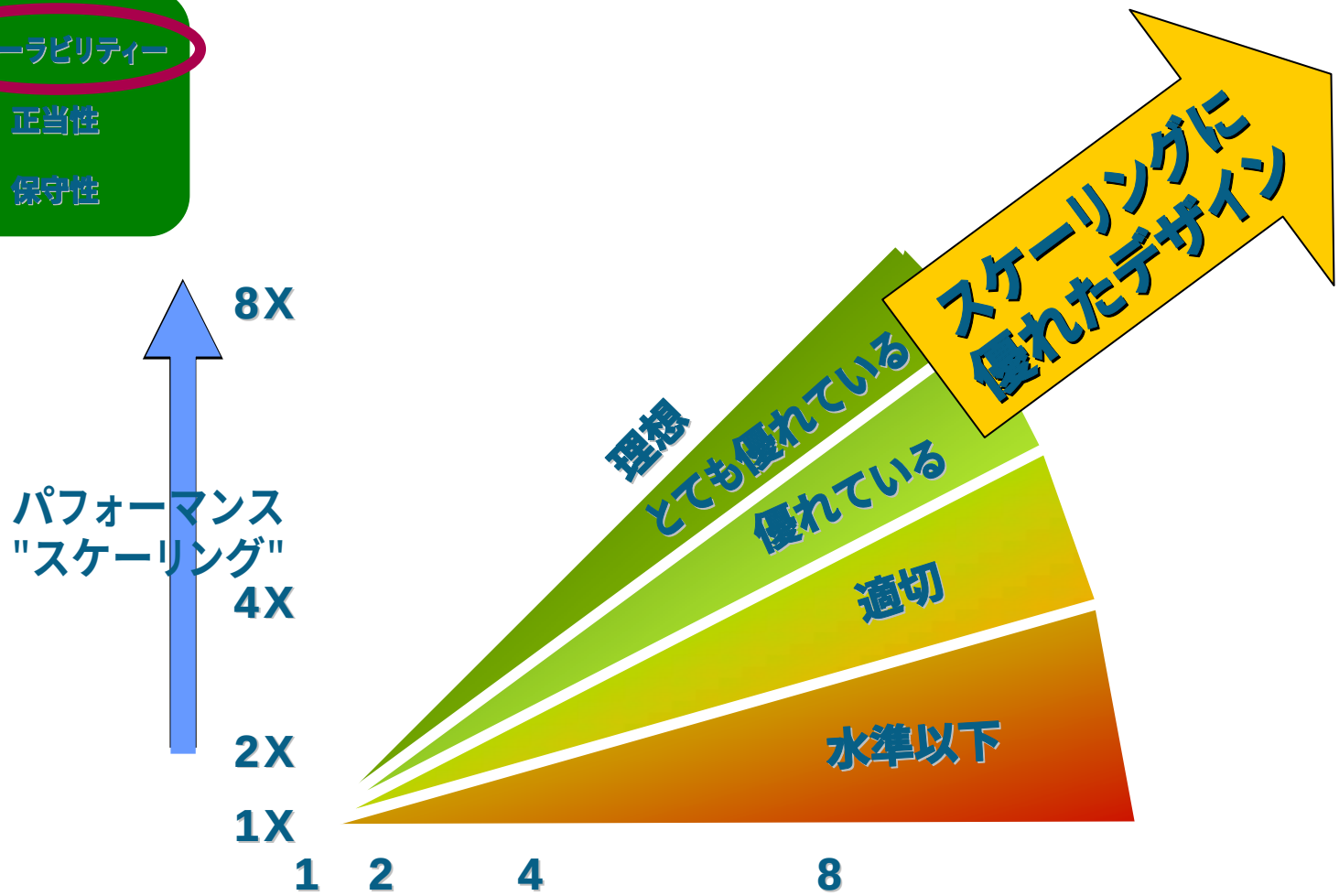
#３ – 手動によるスレッド化実装と MPI

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

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

スケーラビリティー

- スケーラビリティーを妨げる要因は多い。
- スケーリングに対する無知や不注意は*大きな問題*。

最良の解決策は低レベルでスケーラビリティーを実現するコンポーネントを使用することで、OpenMP、ライブラリーやインテル® スレッディング・ビルディング・ブロックがある。

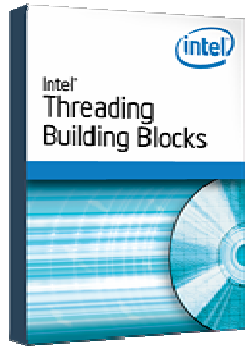
インテル® ソフトウェア開発製品ができること:

- OpenMP (コンパイラーによるサポート)
- インテル® パフォーマンス・ライブラリーはスレッドセーフ (IPP と MKL)
- インテル® スレッド・プロファイラー (ボトルネックを可視化)
- インテル® スレッディング・ビルディング・ブロック (C++ テンプレートで並列化)
- インテル® VTune™ パフォーマンス・アナライザー (高度なパフォーマンス解析ツール)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

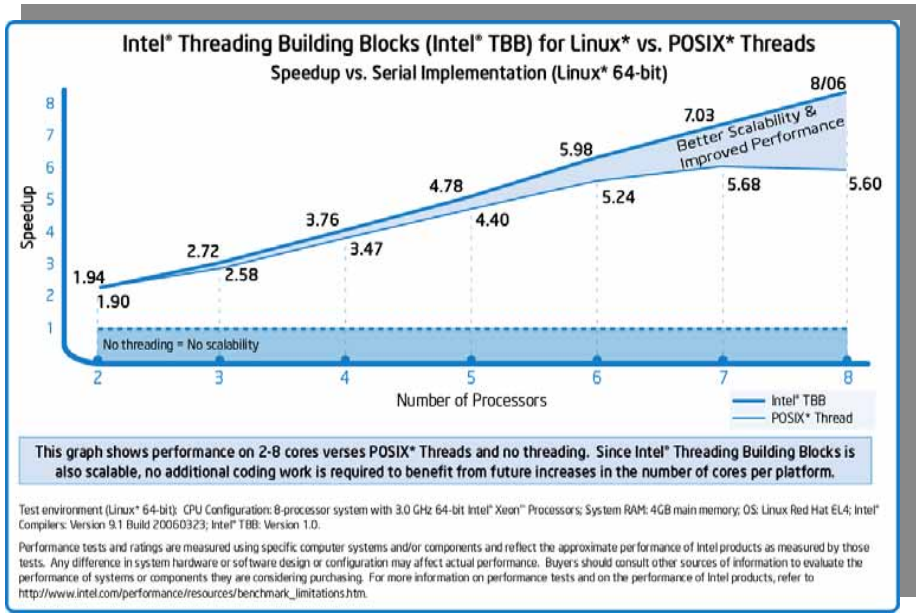
スケーラビリティーの導入:
ライブラリー、OpenMP、MPI

スケーラビリティー
&
ハイパフォーマンス

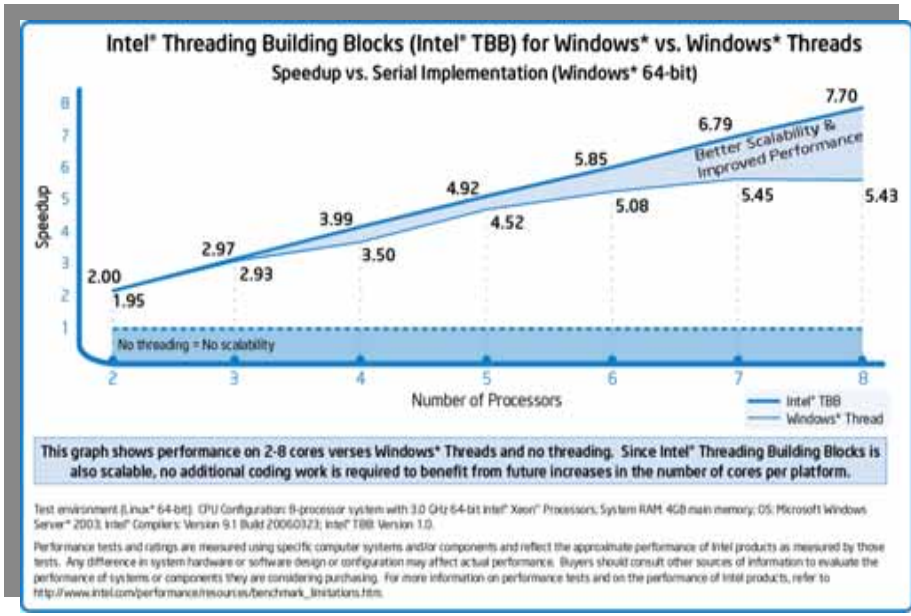


インテル® スレッディング・ビルディング・ブロック

Linux*



Windows*

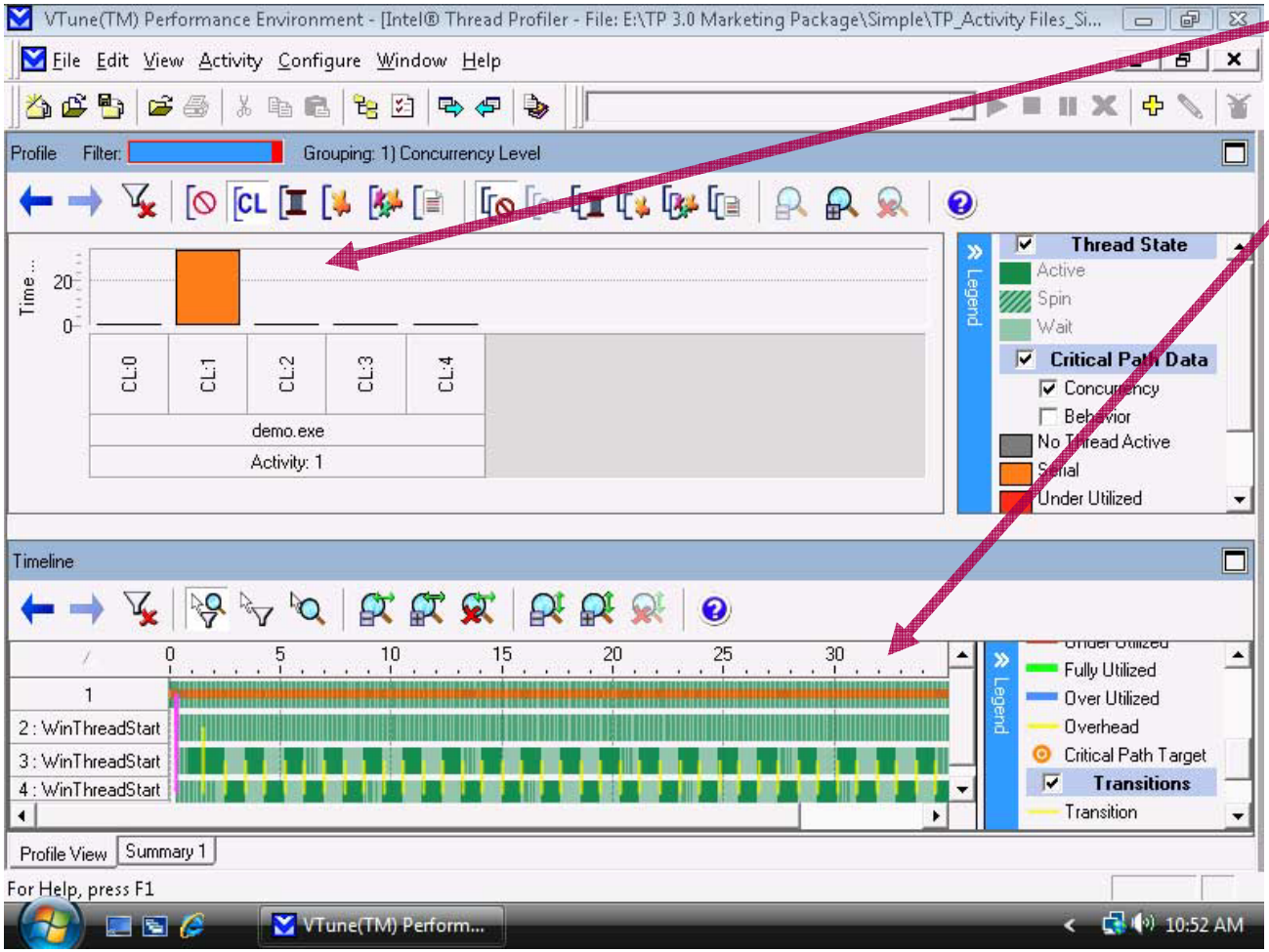


スレッドをどう制御するかに注目するのではなく、行うべき処理に注目できるように、インテル® スレッディング・ビルディング・ブロックにスレッドの*向上*をまかせ、*優れた*スケーラビリティーとパフォーマンスを得ることができました。

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® スレッド・プロファイラー

マルチスレッド・コードを最適化



- アプリケーションによって効率的に使用されているコアの数を測定
- タイムライン・ビューにより実行中のスレッドとその動作を表示
- インバランスなスレッド・ワークロードをハイライト
- 単体のソフトウェアとして、またはインテル® VTune™ パフォーマンス・アナライザー Windows 版で利用可能

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

何をする必要があるか?

Think parallel (**並列化を考える**)

- **これは最も重要な事項:**
  並列の設計で必要と思われるすべての手法は考慮し、
  並列のための直感を磨く

さらに

- **スケーラビリティー**
- **正当性**
- **並列アプリケーションの保守性**

そしてあなたがこれらに関してできること

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

次のステップ

- 今こそアプリケーションを並列化する時です
  - 並列化されていないアプリケーションはマルチコアのパワーを引き出すことはできません。
- インテルは開発者が並列化を開始するのに必要な専門知識、ツール、トレーニングを提供しています。
  - 今後のオンライン・トレーニングにご参加してください。
  - Web サイト (go-parallel.com) をご覧ください。

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® スレッド化ツールの新リリース
スレッド化を簡単に - プログラム / デバッグ / 最適化

インテル® VTune™ パフォーマンス・アナライザー 9.0

- インテル® Core™2 プロセッサーのイベントに対応
- "hotspot" ナビゲーター
- Microsoft* Vista* 対応

インテル® スレッド・プロファイラー 3.1

- 速度とユーザビリティーの向上
- Microsoft Vista 対応

インテル® スレッドチェッカー 3.1

- 速度の向上
- Microsoft* Vista* 対応

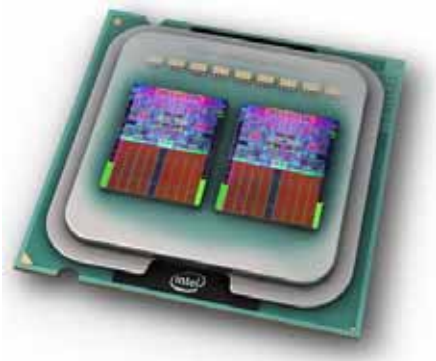
インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 1.1

- 粒度 (グレインサイズ) の自動化
- 最新の OS のサポート

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Core™2 アーキテクチャー チューニングに最適

インテル® Core™2 Duo プロセッサー
2006/07/27 発表



インテル® Core™2 Quad プロセッサー
2006/11/14 発表

インテル® Core™2 プロセッサーの新機能

- ストール・サイクル・アカウンティング
- より良いイベント
- precise イベントの追加

インテル® VTune™ パフォーマンス・アナライザー はこれらの新機能にアクセス し、より優れた解析を提供

「VTune™ アナライザーを使用したインテル® Core™2 プロセッサーのサイクルの要因分析 (Cycle Accounting) は、1 年以内にチューニング方法論のスタンダードになるだろう。」

David Levinthal 氏
シニア・テクニカル・コンサルティング・エンジニア

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

当て推量を減らして作業効率をアップ

従来のチューニング方法論
キャッシュミスの発見

1) キャッシュミスを探す

2) チューニング

3) 速度は向上したか?

問題:

- 「アウト・オブ・オーダー」のハードウェアによるキャッシュミスへの影響どれが重要か?

新しい方法論
ストール・サイクル・
アカウンティング

1) ストールを探して、原因を特定 (例: キャッシュミス)

2) チューニング

3) 速度が向上

解決策:

- 効果が見込める部分のみをチューニング

詳細は、次の Web サイトを参照してください。
http://www.devx.com/go-parallel/Link/33315

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

より優れたイベントで的確な判断

- **従来の方法論**
  すべてのキャッシュミスを検出

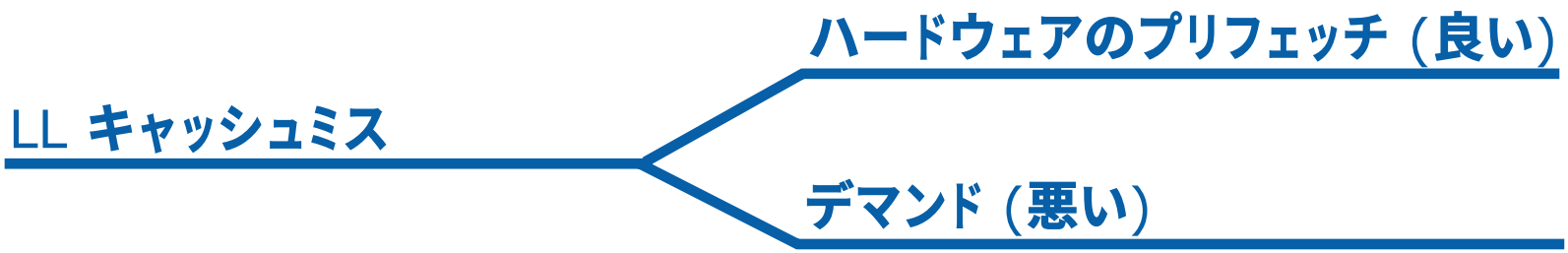
1) LL キャッシュミスを測定
2) チューニング
3) 速度は向上したか?
4) **問題**:

- LL キャッシュミスはデマンド (チューニング対象) および ハードウェアのプリフェッチ (チューニング対象外) が原因で発生する可能性がある

- ***新しい方法論***
  デマンド・キャッシュ・ミスを探す

1) デマンドに起因するキャッシュミスを測定
2) チューニング
3) 速度が向上
4) **解決策**:

- 効果が見込める部分のみをチューニング



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® VTune™ アナライザー 9.0

発見が困難なパフォーマンスのボトルネックを識別

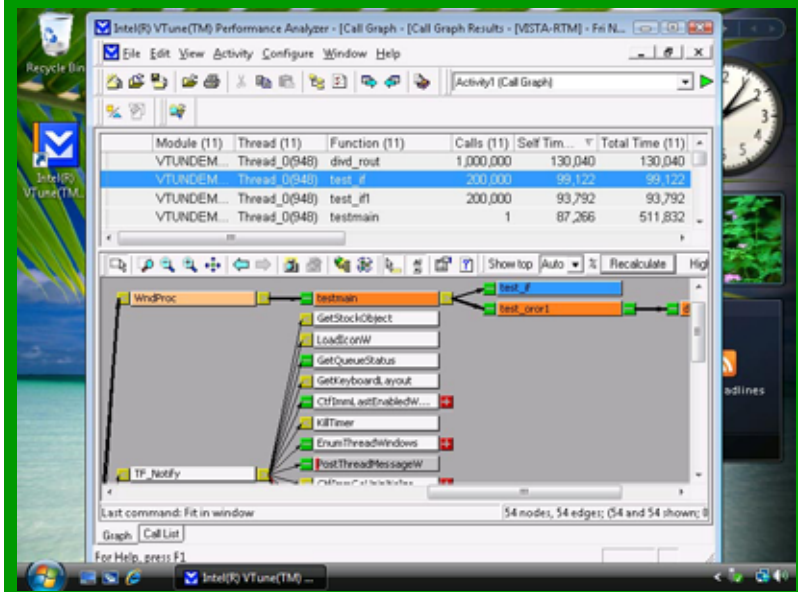
- 機能
  - プロセスまたはスレッド並列コードのチューニング
  - オーバーヘッドの少ないサンプリング
  - グラフィカルなコールグラフ
  - ソースまたはアセンブリーで結果を表示
- 新機能
  - 新しいチューニング方法論
    - Core™2 Duo プロセッサーと Core™2 Quad プロセッサーのストール・サイクル・アカウンティング
  - Windows: Microsoft Vista* 対応

Windows*	Linux*	Mac*	IA32	Intel64	IA64	Multicore
√	√		√	√	√	√

  - Linux: インテル® コンパイラーによる分析と直感的な hotspot ナビゲーター

「インテル® VTune™ パフォーマンス・アナライザーを使えば、これまで何日もかかっていた作業を 1 日以内に完了することができます。」

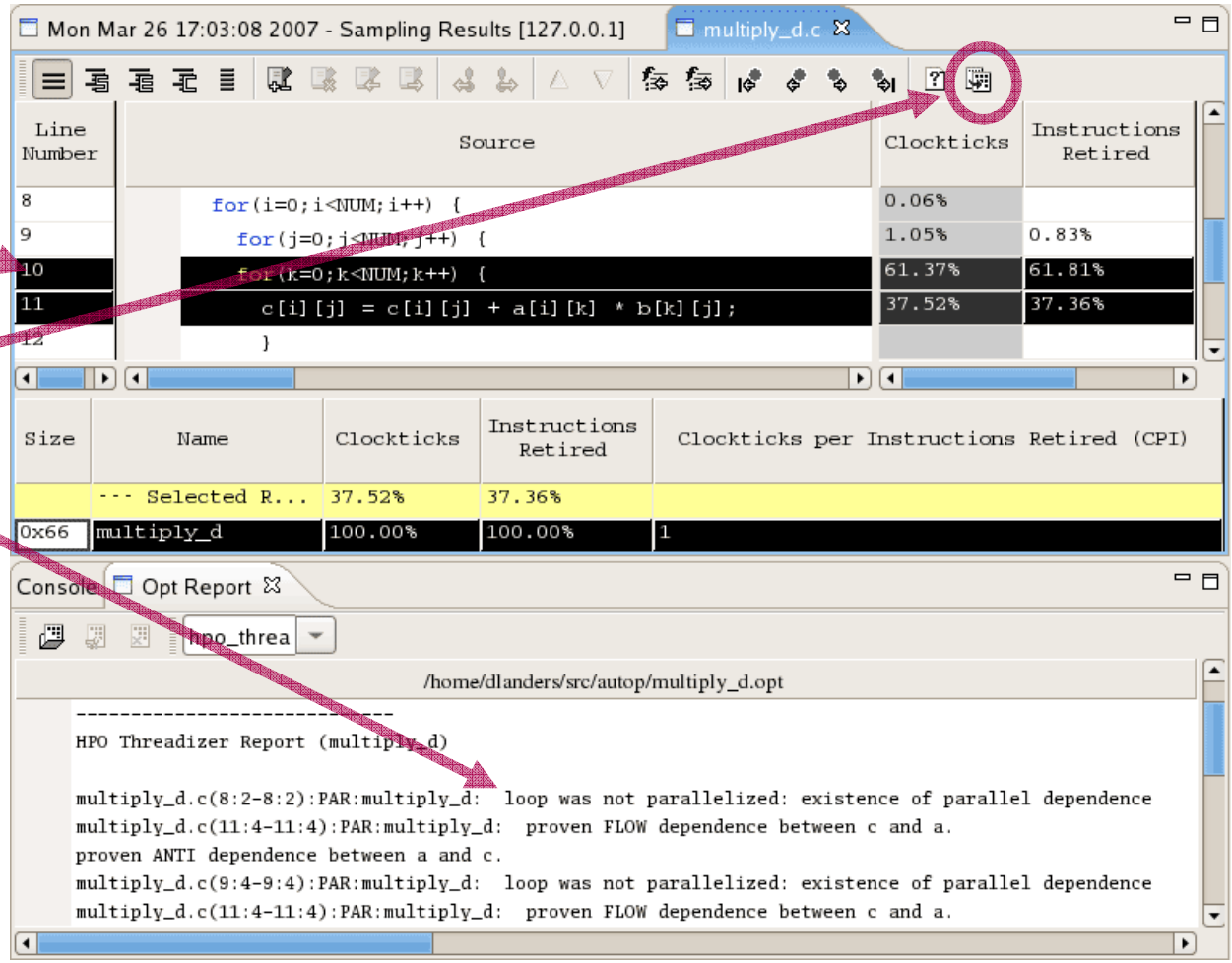
Randy Camp 氏
ソフトウェア R&D 担当副社長
MUSICMATCH Inc.



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

VTune™ アナライザー Linux* 版
コンパイラーのアドバイスを表示

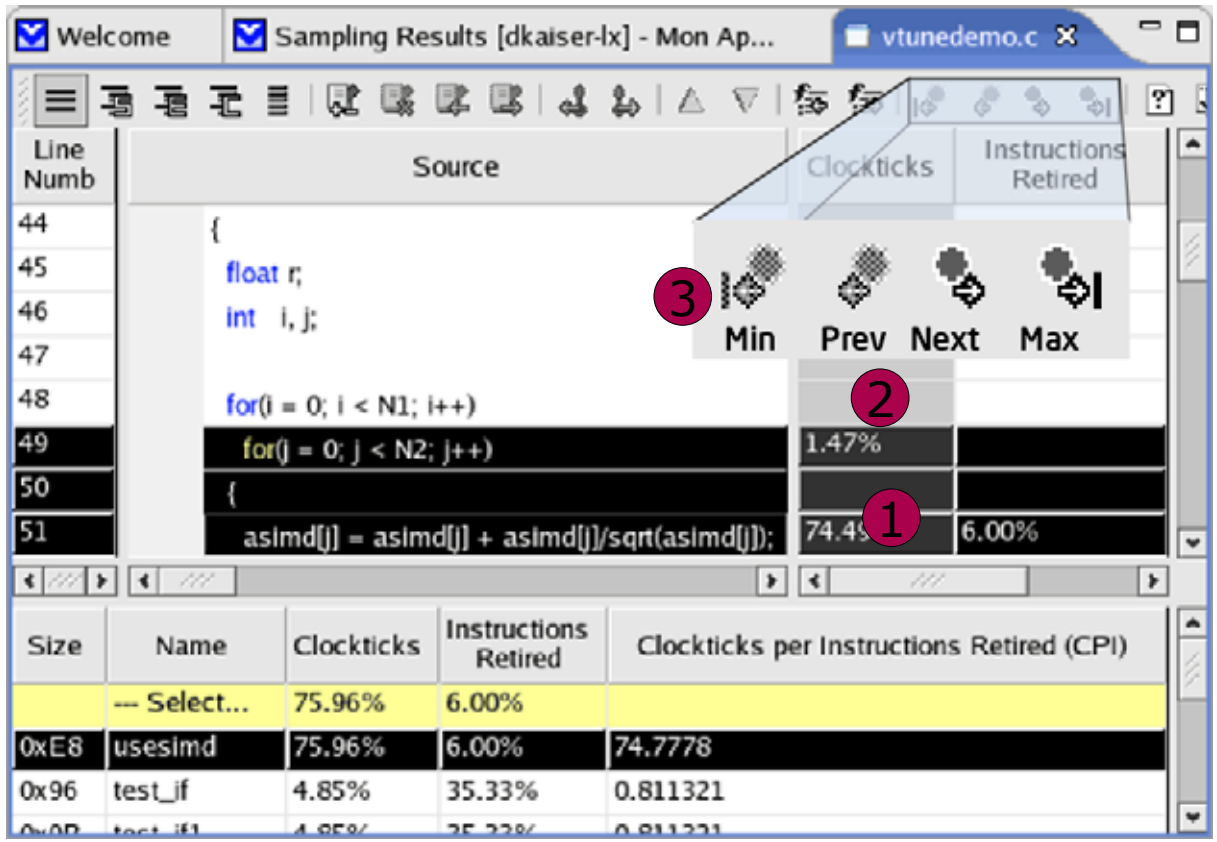
1. インテル® コンパイラーでコンパイル
2. インテル® VTune™ アナライザーを使用して、hotspot を検出
3. ソースビューでコード行を選択
4. 丸で囲まれたアイコンをクリックして、コンパイラーの最適化レポートを表示
5. レポートにより、想定される依存性が原因でコンパイラーが並列化を行わなかったことが判明
6. 依存性がないとわかっている場合は、OpenMP 文を挿入
7. より高速な並列ソフトウェアが完成
8. インテル® スレッドチェッカーでロジックを確認



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® VTune™ アナライザー Linux* 版 ワンクリック hotspot ナビゲーション (新機能)

1. ソース行の隣にあるイベントカウントで、各行の重要度を簡単に確認できます。
   - しかし、大きなソースファイルでは、最も "ホット" な hotspot をどのように見つけたら良いでしょうか? または、何千行も離れた次の重要な行にジャンプするにはどうしたら良いでしょうか?
   - イベントの列をクリックして、
2. ナビゲートするイベントを選択し、
   - [Min (最小)] アイコン、[Max (最大)] アイコン、[Next (次へ)] アイコンを
3. クリックすると、素早く hotspot を確認できます。



インテル® VTune パフォーマンス・アナライザー Linux 版

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

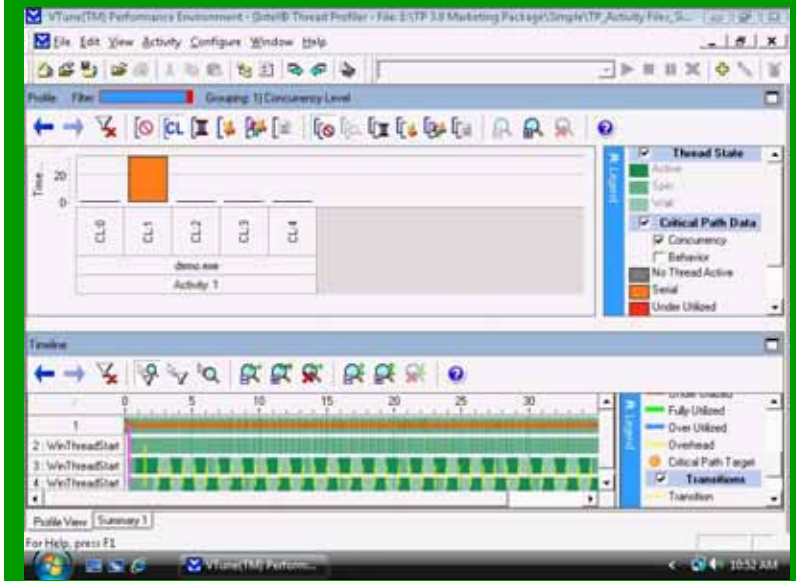
インテル® スレッド・プロファイラー 3.1
スレッド化の非効率な場所をピンポイント

- 機能
  - 並列レベルでアプリケーションを表示し、コアが十分に活用されているかを確認
  - スレッド関連のオーバーヘッドが与えているパフォーマンスの影響を特定
  - アクティブまたはインアクティブな作成スレッドを識別
  - VTune™ アナライザー Windows* 版に同梱
  - スレッディング・ビルディング・ブロック API 対応
- 新機能
  - 使用方法は簡単 - カスタム設定を呼び出すだけ
  - すぐに使用可能 - ユーザーが選択できるスタック・ウォーキング
  - Microsoft* Vista* 対応

Windows*	Linux*	Mac*	IA32	Intel64	IA64	Multicore
√	√		√	√		√

「インテル® スレッド・プロファイラーは、スレッド化コードに存在するボトルネックの分析に非常に役立ちました。インテル® スレッド・プロファイラーでは、問題のある部分が素早くピンポイントで検出され、速度低下の理由が表示されるため、コードを再構築してスレッド・パフォーマンスを向上させることができました。」§

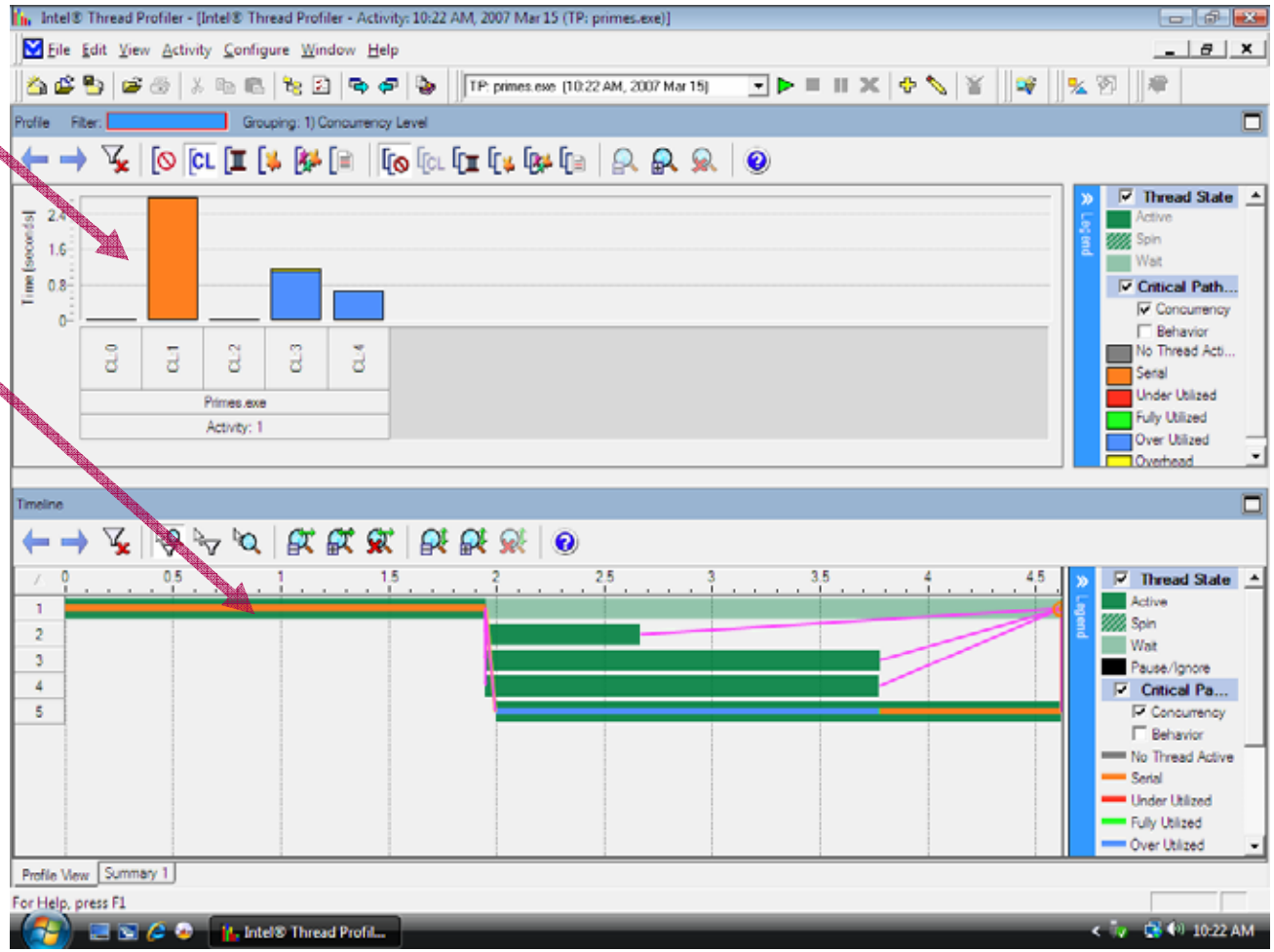
Martin Watt 氏
ソフトウェア・アーキテクト
Alias



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® スレッド・プロファイラー
マルチスレッド・コードを最適化

- アプリケーションによって効率的に使用されているコアの数を測定
- タイムライン・ビューにより実行中のスレッドとその動作を表示
- インバランスなスレッド・ワークロードをハイライト
- 単体のソフトウェアとして、またはインテル® VTune™ パフォーマンス・アナライザー



© Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の名称、製品名などは、一般に各社の表示、商標または登録商標です。

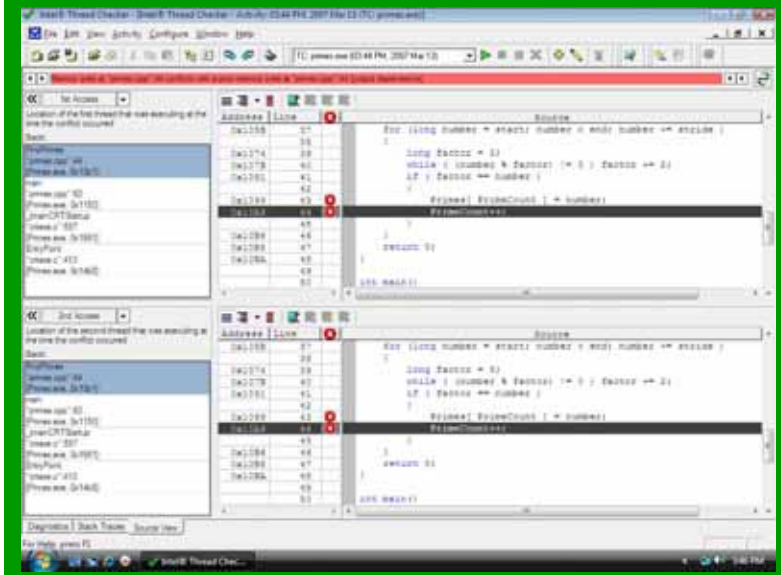
インテル® スレッドチェッカー 3.1
スレッド化のエラーを正確にピンポイント

- 機能
  - 発見の困難な競合状態やデッドロックを検出
  - ソースコード行のエラーの正確な位置を指摘
  - 回帰テスト実行用にバッチスクリプトを統合
  - Windows と Linux のコマンドライン・インターフェイス
  - 再コンパイルせずに標準的なデバッグビルドで動作
- 新機能
  - パフォーマンスの最適化による迅速な分析
  - Microsoft* Vista* 対応

Windows*	Linux*	Mac*	IA32	Intel64	IA64	Multicore
√	√		√	√		√

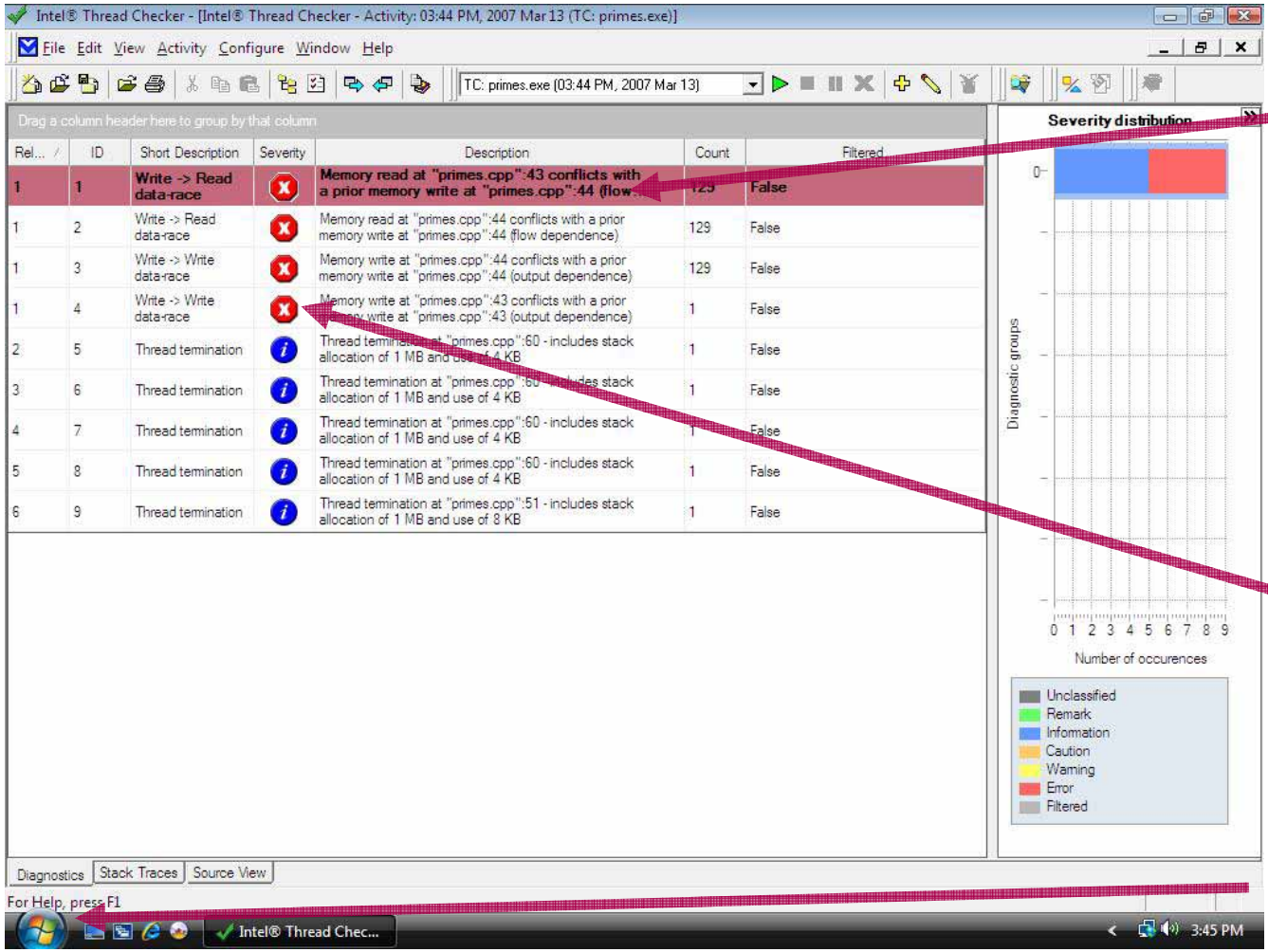
「スレッドチェッカーがなければ、これほど早く、効率的にネットワークをセットアップして実行させることはできなかったでしょう。スレッドチェッカーは本当にすばらしいツールで、この製品なしにはマルチスレッド・コードの開発は考えられません。」

Doug Service 氏、テクノロジー開発担当ディレクター
Chris Stark 氏、ソフトウェア・エンジニア
Ritual Entertainment



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® スレッドチェッカー
正しいマルチスレッド・コード



並列ビューとタイムライン・ビューを同時に表示

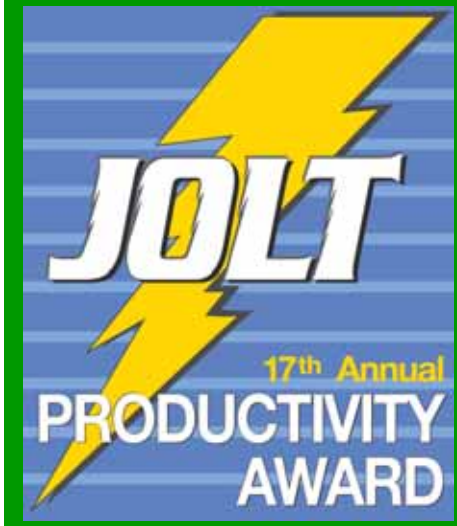
- 特許取得済みの高度なエラー検出エンジンにより、デッドロックやデータレースなどの潜在的な問題を検出して、ソースコード行、コールスタック、およびメモリー参照にマッピング

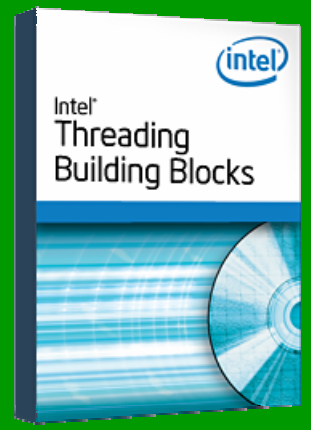
  ダブルクリックしてソースにドリルダウン
- 用途によりインストルメントするモジュールを提案
- マルチスレッド・アプリケーションが効率的に診断できるように、わかりやすい警告を表示。最も重大で潜在的なエラーはハイライト
- Microsoft Vista* および Visual Studio 2005* に対応

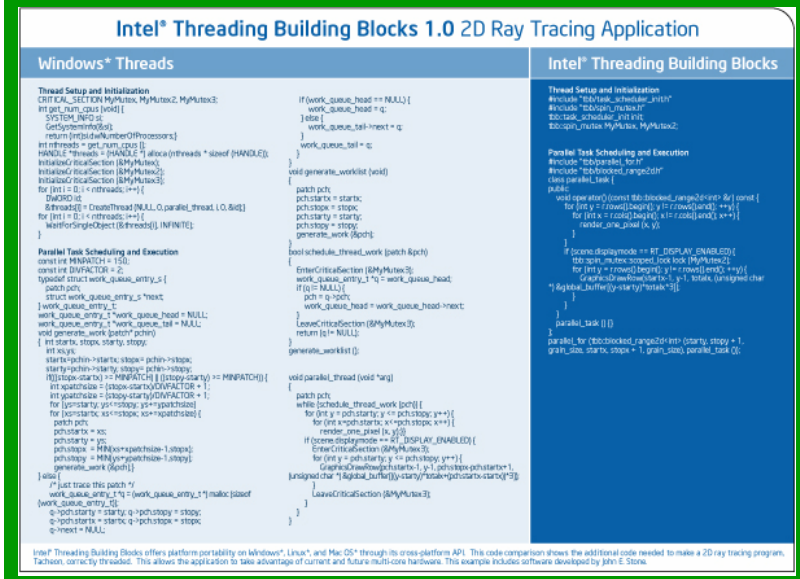
© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® スレッディング・ビルディング・ブロック 1.1
C++ で並列化

- **機能**
  - スレッドではなく、馴染みのあるタスク処理を使用する C++ ランタイム・ライブラリー
  - ハイレベルな抽象化により、パフォーマンスを損なうことなくわずかなコードの追加でスレッド化
  - 利用できるコア数に応じて適切にパフォーマンスを向上
  - スレッド・ライブラリー API は Linux、Windows、Mac OS 間でのプラットフォーム間のポータビリティーを提供
  - すべての C++ コンパイラーで動作 (例: Microsoft、GNU、インテル)
- **新機能**
  - より効率的な並列アルゴリズムのための自動粒度計算
  - Microsoft* Vista* 対応
  - Mac OS X のネイティブ 64 ビットのフルサポート







等価な Windows スレッド機能と並べて比較すると、インテル® スレッディング・ビルディング・ブロックがスレッド化の実装を大幅に単純化していることがわかる

Windows*	Linux*	Mac*	IA32	Intel64	IA64	Multicore
√	√	√	√	√	√	√

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

まとめ

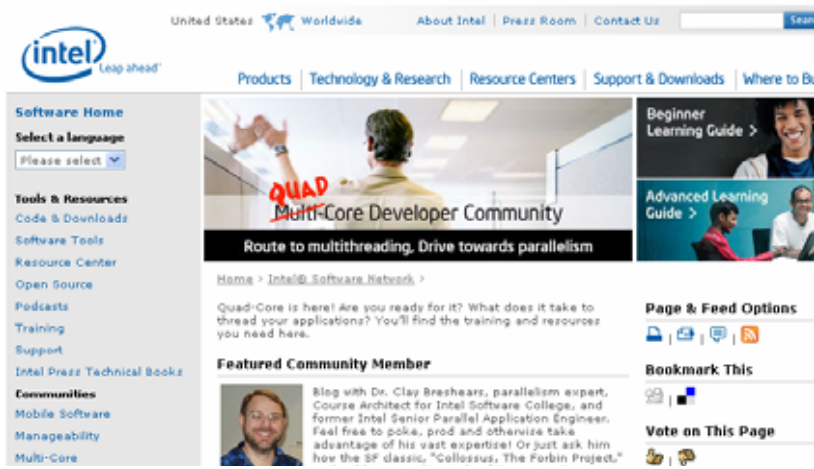
- インテル® ソフトウェア開発製品で最新のオペレーティング・システムとマルチコア・プロセッサー対応のスレッド化を*先駆け*
- インテル® VTune™ パフォーマンス・アナライザー 9.0
  - インテル® Core™2 プロセッサーのイベントサポートと "hotspot" ナビゲーター
- インテル® スレッド・プロファイラー 3.1
  - 速度とユーザビリティーの向上
- インテル® スレッドチェッカー 3.1
  - 速度の向上
- インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 1.1
  - 粒度の自動化
- インテル® ソフトウェア製品の評価版

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

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

開発者向けスレッド化リソース
他の開発者やインテルの専門家とのコラボレーション

- マルチコア・デベロッパー・センター
- 開発者ツール
- 評価版ソフトウェア (無償)
- スレッド化フォーラム
- コード & ダウンロード
- ポッドキャストと Web セミナー
- オンライン・トレーニング
- 専門家によるブログ
- ナレッジベース
- 技術資料とホワイトペーパー



www.intel.co.jp/jp/software/mcdeveloper/311534.htm

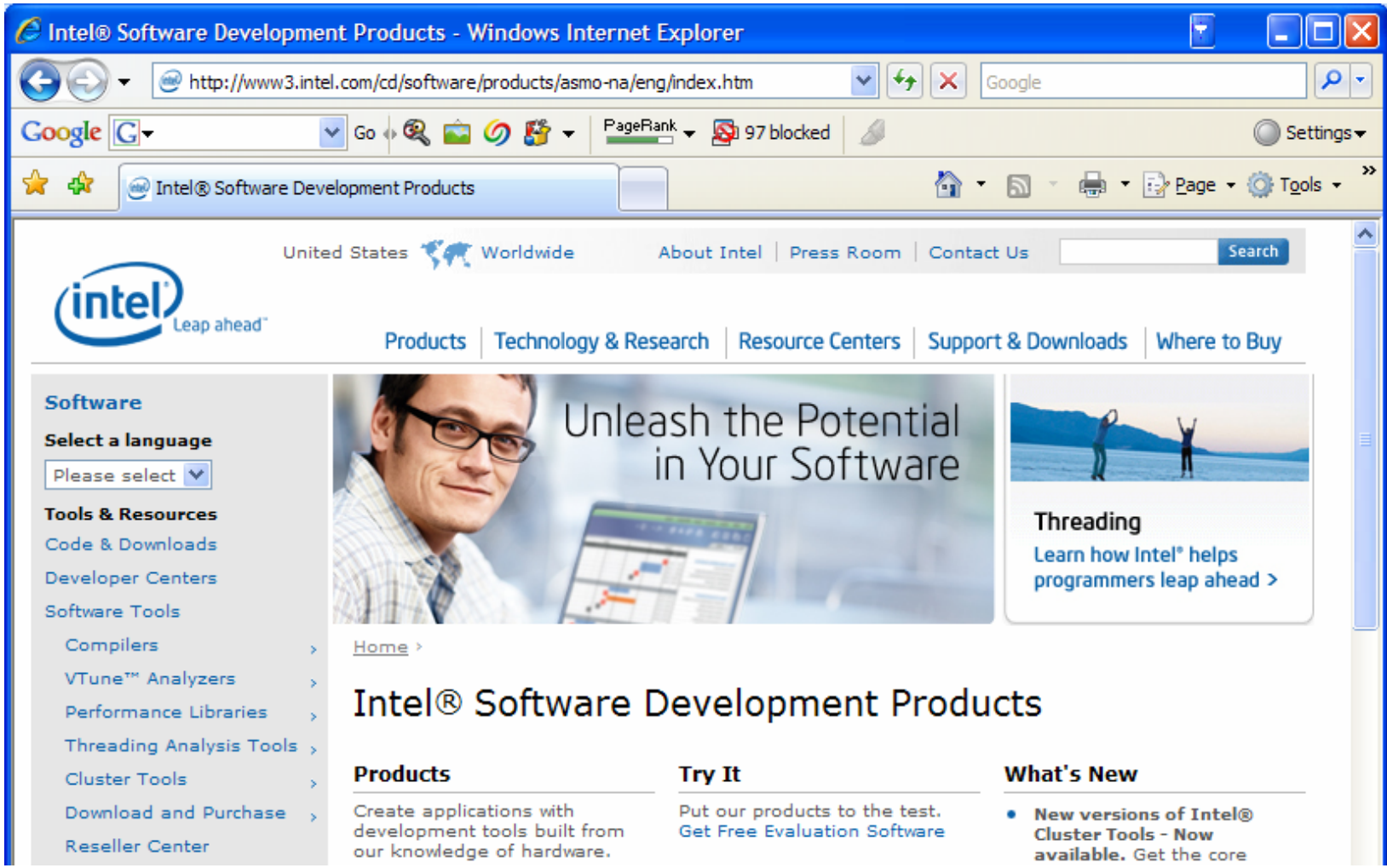
www.devx.com/go-parallel (英語)

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
*その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

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

必要な情報はここに

評価版
情報
フォーラム
ブログ
トレーニング
インテル® ソフトウェア ネットワーク (ISN)



© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

本日はセミナーにご参加いただき

誠にありがとうございました

事例

- Adobe® Premiere® Pro 2.0 – **プロフェッショナル・ビデオ編集**
- Pixar - RenderMan® Pro Server™ 13 – **プロダクション・レンダリング**

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

プロフェッショナル・ビデオ編集とマルチコア

- 新しいマルチコア・プラットフォーム
  - インテル® vPro™ テクノロジー
  - インテル® Viiv™ テクノロジー
  - インテル® Centrino® Duo モバイル・テクノロジー
- 利用モデルの影響
  - 高精細 (HD) ビデオ
  - エンドユーザーの生産性効率 – マルチストリーム
  - 高速映像処理
- 市場投入期間の短縮が重要
  - 旧バージョンに存在するスレッド化
  - 高度な HD ビデオ再生機能に専念

マルチコアでの高精細コンテンツの再生機能を強化

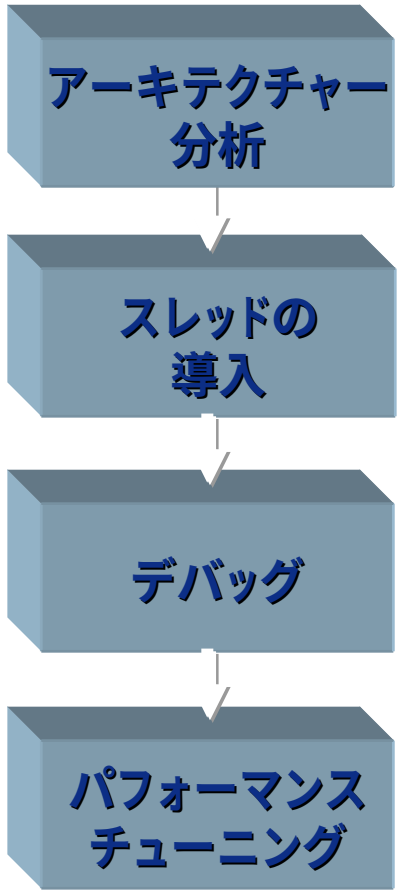
© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

需要に対応 – マルチコア向けのスケーラブルなスレッド化

スレッド化のコンサルティングと提携
- スレッド化対応のアドバイザーとしての役割
- スケーラビリティーの分析

インテル® ソフトウェア開発製品
- 開発サイクル全体で活用

インテル® VTune™ パフォーマンス・アナライザー →
→ インテル® C++ コンパイラー（OpenMP サポート付き）→
→ インテル® インテグレーテッド・パフォーマンス・プリミティブ（スレッド化対応）



マルチコア向け最適化を迅速に行うインテル® ソフトウェア製品

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

結果

- マルチコアの質の高いエンドユーザー体験
  - MPEG-2 ビデオで 4 スレッド以上
- Premiere Pro 2.0 の幅広い利用
  - アドインカード不要のハイエンドシステム

インテル® マルチコア・プラットフォームで ユーザー体験を強化

「スレッド化は、難しい作業で時間もかかります。インテルは、このことを十分に理解してすばらしい製品をソフトウェア開発者に提供しています。」

Bill Hensler 氏
エンジニアリング担当副社長
Video Solutions for Adobe

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

事例

- Adobe® Premiere® Pro 2.0 – プロフェッショナル・ビデオ編集
- Pixar - RenderMan® Pro Server™ 13 – プロダクション・レンダリング

RenderMan Pro Server

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

プロダクション・レンダリングとマルチコア

- 新しいマルチコア・プラットフォーム
  - デュアルコア インテル® Xeon® プロセッサー・ベース・システム
  - インテル® vPro™ テクノロジー
- 新しい利用モデル
  - レンダリングされた各フレームでマルチコアを活用
  - インタラクティブ・レンダリング
  - 画像の複雑度の向上
- 市場投入期間の短縮が重要
  - スレッド化のエラーにより発生する不正な結果、クラッシュ
  - スケーリングで期待するパフォーマンスが得られない
  - クロスプラットフォーム・サポート

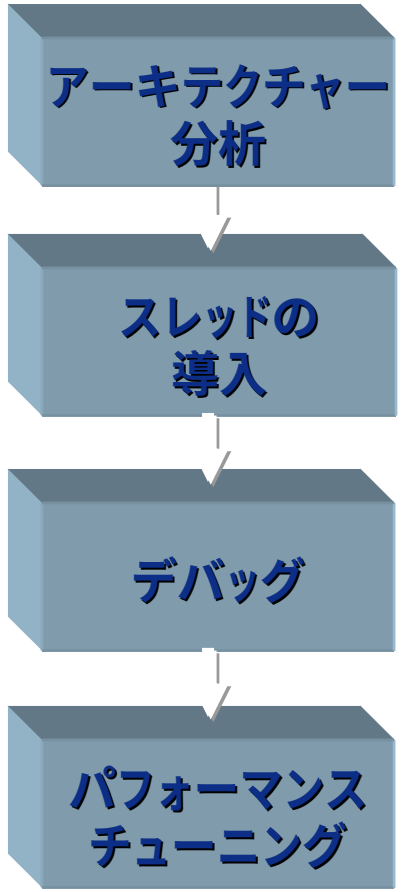
マルチコア上の高速レンダリング

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

需要に対応 – マルチコアのスケーラブルなスレッド化

スレッド化のコンサルティングと提携

- スレッド化対応のアドバイザーとしての役割
- スケーラビリティーの分析



インテル® ソフトウェア開発製品

- 生産的なクロスプラットフォーム活用
  - Windows → Linux
- 開発サイクル全体で活用

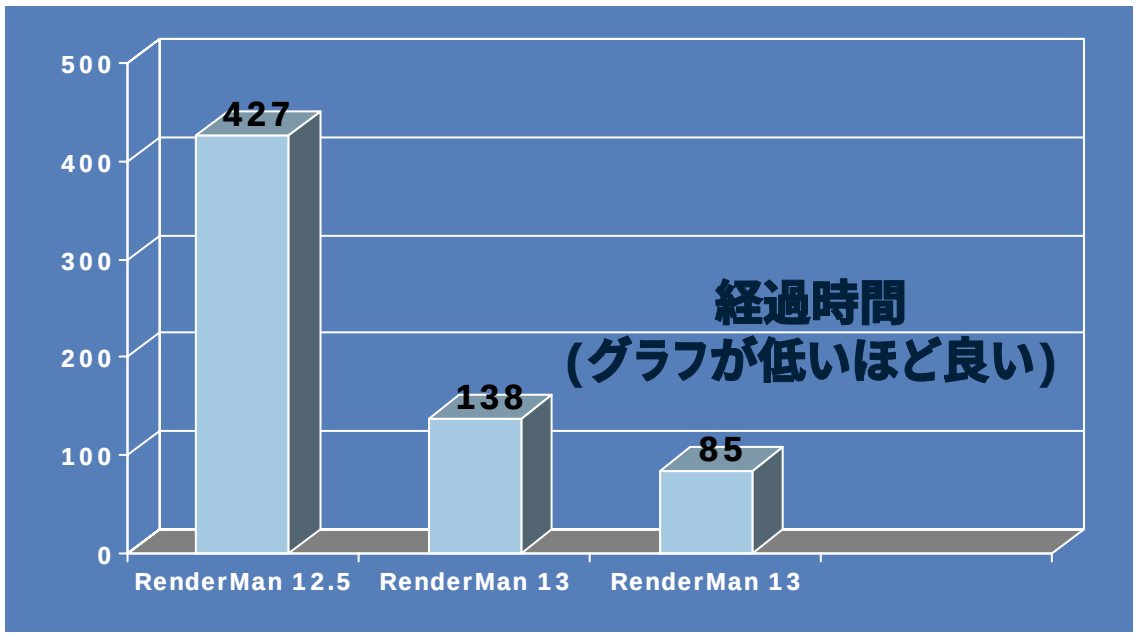
  インテル® VTune™ パフォーマンス・アナライザー →
  → インテル® C++ コンパイラー (OpenMP サポート付き) →
  → インテル® スレッドチェッカー →
  → インテル® スレッド・プロファイラー

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

結果

Pixar 社による新しい性能

- レイトレーシング >4 コアで 3 倍の速度
- メモリー消費の大幅な軽減 (60%)
- 映像製作に欠かせない「市場投入期間」の短縮



「インテルのスレッド化ツールで開発サイクルを大幅に短縮することができました。今ではインテル® スレッド化ツールでの作業は開発プロセスの一部に組み込まれています。」

Dana Batali 氏
Pixar 社
RenderMan
開発担当ディレクター

© 2007 Intel Corporation. 無断での引用、転載を禁じます。記載内容は予告なしに変更されることがあります。
＊その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。


intel®
Leap ahead™