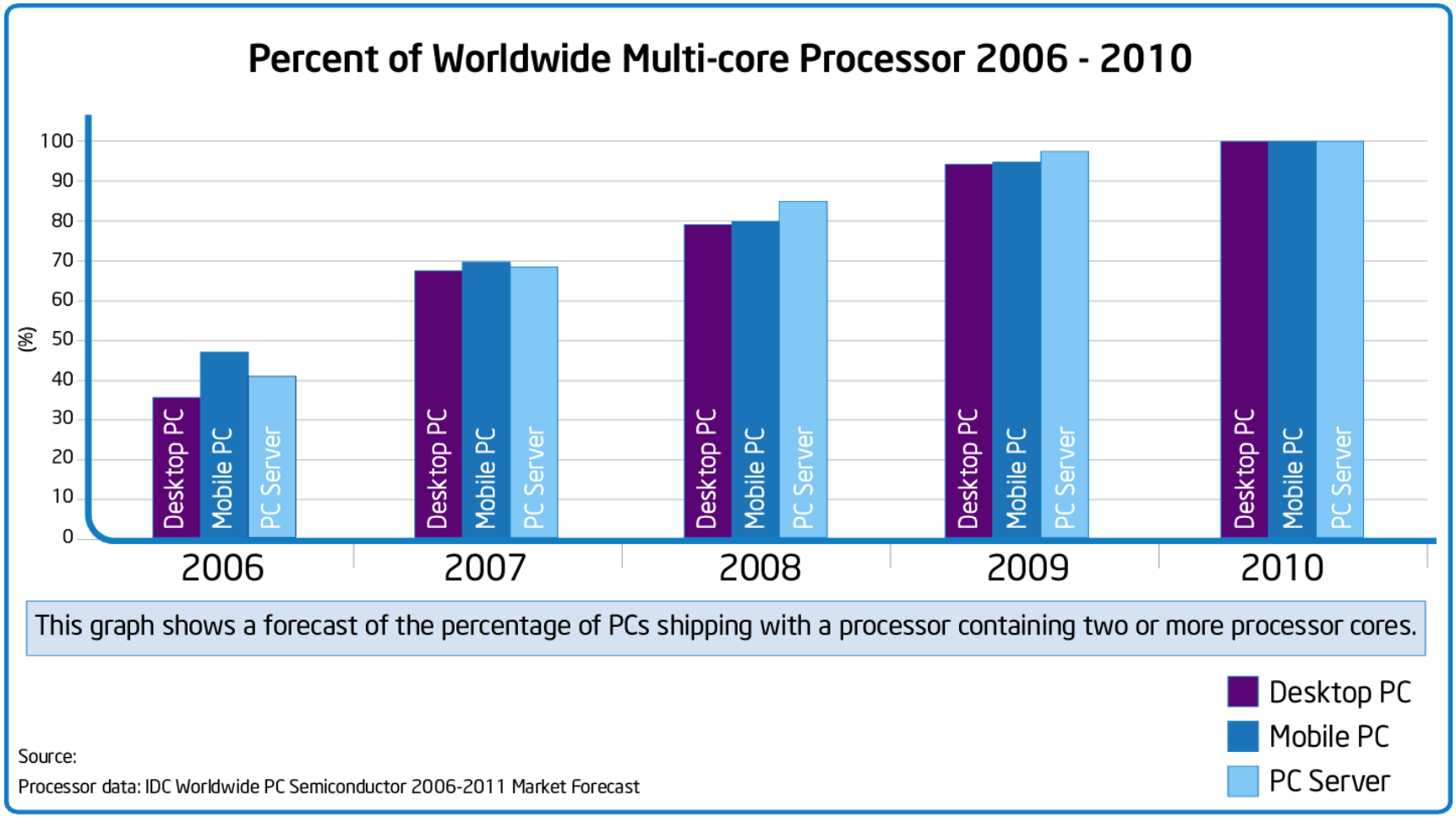
インテル® ソフトウェア開発製品、2007年
SDタイムスの 100 Influencer アワードを獲得

マルチコアのパワーをアプリケーションで活用

インテル® ソフトウェア製品概要
2007年 11月

コンピューティングの世界はマルチコアへ
準備はできていますか？



容易なソフトウェアのマルチスレッド化支援するソフトウェア開発製品

インテル® ソフトウェア開発製品は開発者が優れたコードを作成することを支援:

- 簡単にマルチスレッド・アプリケーションを開発
- アプリケーションの性能を最大限に
- 1つのツールセットで 32/64 ビットのアプリケーション開発に対応（Windows*, Linux* および Mac OS* X 版を提供中）
- インテルおよび AMD プロセッサーで動作する最適化されたアプリケーションの開発

同じツールセットを全ての環境で利用可能
32/64 ビット Windows*、Linux* および Mac OS* X

Intel® Software Development Products (Currently Available ●)		Itanium® 2		Core® 2 Duo / Centrino® / Pentium® 4 / Xeon® / VIIV				
		Operating systems		Operating Systems				
		Windows*	Linux*	Windows	Linux		Mac OS* X	
		Development Environments		Development Environments				
		Visual Studio*	GCC*	Visual Studio	GCC		XCode*	
Compilers	C++	●	●	●	●		●	
	Fortran	●	●	●	●		●	
Performance Analyzers	VTune™ Performance Analyzer	●	●	●	●			
Performance Libraries	Integrated Performance Primitives	●	●	●	●		●	
	Math Kernel Library	●	●	●	●		●	
	Threading Building Blocks		●	●	●		●	
Threading Analysis Tools	Thread Checker			●	●			
	Thread Profiler			●				
Cluster Tools	MPI Library		●		●			
	Trace Analyzer and Collector		●		●			
	Math Kernel Library Cluster Edition	●	●	●	●			
	Cluster Toolkit		●	●	●			

サーバーからモバイル / ワイレス・コンピューティングまで、インテル® ソフトウェア開発製品は
インテル® プラットフォームにまたがったアプリケーション開発を可能に

インテル® C++ コンパイラー プロフェッショナル版 10.1

マルチコアに対応した今日最良の C++ コンパイラー

- 簡単にマルチスレッド・アプリケーションを開発
  - OpenMP*、インテル® スレッディング・ビルディング・ブロック、自動並列化
  - ***新機能*** OpenMP 互換ライブラリーをサポート(MSC および GNU での OpenMP 実装で利用可能)
- アプリケーションの性能を最大限に
  - 高度な最適化、プロファイル・ガイド最適化そして最新のプロセッサーの機能(例:SSE4)の活用などを含む最適化機能を実装
- 1つのツールで 32/64ビットのアプリケーション開発に対応(Windows*、Linux* そして Mac OS* X 対応)
  - 既存の Windows*、Linux* および Mac OS* X 開発環境に完全互換
- インテルおよび AMD プロセッサーで動作する最適化されたアプリケーションの開発

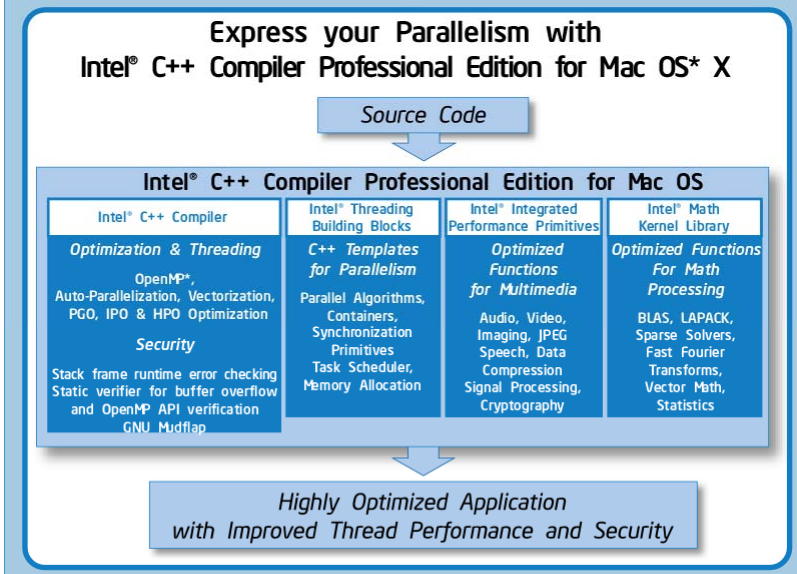
スレッドやロジックの問題のための最強のQAツール

プロフェッショナル版は次の製品を同梱します:

- インテル® C++ コンパイラー 10.1
- インテル® パフォーマンス・プリミティブ 5.3
- インテル® マス・カーネル・ライブラリー 10.0
- インテル® スレッディング・ビルディング・ブロック 2.0

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

> "We use the Intel C++ and Fortran compilers daily. We greatly appreciate that the Intel compilers are the one compiler that does it all. It supports all our 32 & 64 bit systems, plugs right into our development environments, supports both Windows and Linux and produces code with improved performance."
>
> *John Zollweg*
> *Senior Research Associate*
> *Cornell Center for Advanced Computing*



インテル® Fortran コンパイラー プロフェッショナル版 10.1

マルチコアに対応した業界標準の Fortran コンパイラー

- 簡単にマルチスレッド・アプリケーションを開発
  - OpenMP*、インテル® スレッディング・ビルディング・ブロック、自動並列化
  - ***新機能*** OpenMP 互換ライブラリーをサポート(MSC および GNU での OpenMP 実装で利用可能)
- アプリケーションの性能を最大限に
  - 高度な最適化、プロファイル・ガイド最適化そして最新のプロセッサーの機能(例:SSE4)の活用などを含む最適化機能を実装
- 1つのツールで 32/64ビットのアプリケーション開発に対応(Windows*、Linux* そして Mac OS* X 対応)
  - 既存の Windows*、Linux* および Mac OS* X 開発環境に完全互換
- インテルおよび AMD プロセッサーで動作する最適化されたアプリケーションの開発。スレッドやロジックの問題のための最強のQAツール

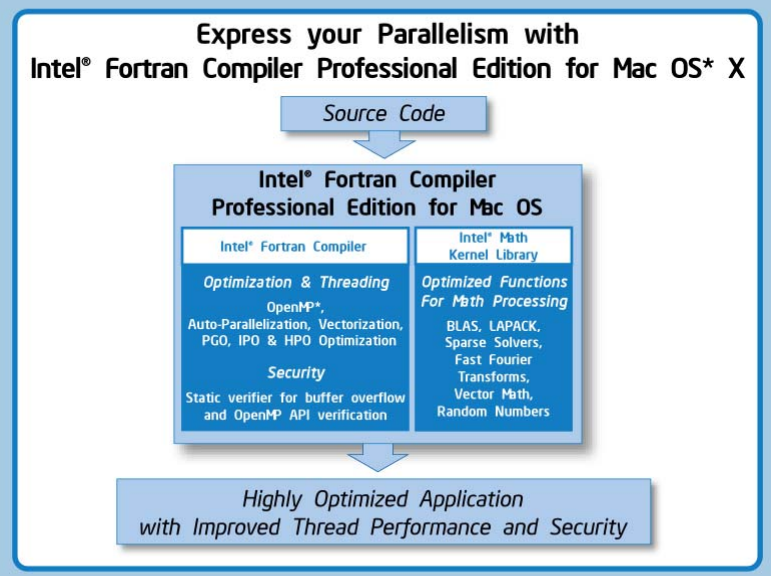
多くの Fortran 2003 仕様をサポート

プロフェッショナル版は次の製品を同梱します

- インテル® Fortran コンパイラー 10.1
  - Windows 版には、Microsoft Visual Studio* 2005 プレミア・パートナー・エディションが含まれます
- インテル® マス・カーネル・ライブラリー 10.0

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

> "I have been using the Intel Fortran compiler v10.0 along with the MKL library. The combination has given me complete satisfaction regarding speed and accuracy for fast generation of random numbers."
>
> *Dr Daniel Gandolfo*
> *Researcher*
> *Centre de Physique Théorique - CNRS*



HPO – ハイパフォーマンス・パラレル・オプティマイザー

IA32/インテル® 64/ IA64 システム向けの新しいパラレライザーおよびベクトライザー

- 以下のオプションを使用
  -O3 –Q[a]x[P,T,N..]
  -parallel –O3
- まったく新しいデザイン – HLO の後のベクトライザー（「ループ分配」のようなループの最適化）
  - IA32 およびインテル® 64 システムでループ変換を有効にし、ベクトル化。または、IPF でロードペア生成。
- 各最適化間のより良い相互作用
- 最適化されたシリアルコードよりも優れた並列化
- より効率的なマルチスレッド・コード
- 強化されたループ変換
- 1 つになったベクトル化と並列化のコストモデル

HPO はパフォーマンス、特にシリアルおよびパラレル実行のループを向上させる

コンパイラー 10.1 のスタティックの検証 (SV)

スタティックの検証機能は、コンパイル時またはリンク時に、アプリケーション全体に渡り、ユーザーコード内のさまざまな不具合や言語機能と矛盾している箇所を特定

- **C++ および Fortran ユーザーによる開発、そして、メモリーアクセス、バッファー・オーバーフロー、OpenMP* 使用問題についてのデバッグをサポート**
- **OpenMP API およびデータ依存性の問題を発見**

初/中級開発者にとってのメリット:
- 200 種類以上の潜在的なコーディング・エラーを検出
- C/C++ および Fortran 言語の機能を正しくユーザーに指導
  - （例: OpenMP 宣言子）

上級開発者にとってのメリット：
- 発見が困難なエラーを検出
  - （例: タイプミス、初期化されていない変数）
- 異なるプログラム単位中の矛盾するオブジェクト宣言の検出（例: 異なるデータ型の仮引数や実引数）

```
$ cat -n main.cpp
     1  #include <iostream>
     2  int foo(const char *);
     3  int main() {
     4     char * y=NULL;
     5     std::cout << foo(y);
     6     return(0);
     7  }
```

```
$ cat -n test.cpp
     1  #include <string>
     2  int foo(const char * widget) {
     3    return(std::strlen(widget));
     4  }
```

```
$ icc -diag-enable sv3 main.cpp test.cpp
main.cpp(5): error #12143: [SV] "y" is
uninitialized
test.cpp(3): warning #12086: [SV] header-file
containing the declaration of intrinsic "strlen"
should be included or forward declaration is
wrong
```

コンパイラー 10.1 Linux版 / Mac OS版の Mudflap サポート

危険なポインターの操作は、コンパイラーによりインストルメントされ、
バッファー・オーバーフローや不正なヒープの使用を防ぎます。

```
$ cat -n of-calc.c
 1  #include <math.h>
 2  void calcSqrt(double *a, int N){
 3    for (int i=1;i<N;i++)
 4      a[i]=sqrt( (double)i );
 5    return;
 6  }
$ cat -n of-main.c
 1  #include <stdio.h>
 2  void calcSqrt(double *a, int N);
 3  int main() {
 4    double a[10];
 5    for (int i=0;i<10;i++) {
 6      a[i] = (double) i;
 7    }
 8    calcSqrt(a,11);
 9    return 0;
10  }
```

```
$ icc –o t-mudflap of-calc.c of-main.c -fmudflap
–lmudflap

$ ./t-mudflap
*******
mudflap violation 1 (check/write):
time=1177545175.621017 ptr=0x7fff5bc461d0
size=8
pc=0x2aaaaaae74a1 location=`of-calc.c:4
(calcSqrt)'

/usr/lib64/libmudflap.so.0(__mf_check+0x41)
[0x2aaaaaae74a1]
      ./t-mudflap(calcSqrt+0x8a) [0x40187c]
      ./t-mudflap(main+0x3b5) [0x401c5d]
Nearby object 1: checked region begins 1B after
and ends 8B after
mudflap object 0x136ad320: name=`of-
main.c:4 (main) a`
bounds=[0x7fff5bc46180,0x7fff5bc461cf]
size=80 area=stack check=0r/10w liveness=10
alloc time=1177545175.621003
pc=0x2aaaaaae6fc1
```

10.1 コンパイラーの新機能 – OpenMPでのアフィニティーの設定

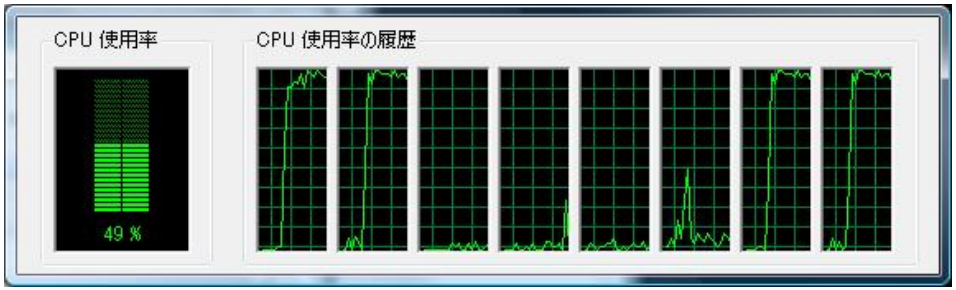
OpenMP で生成されるスレッドは OS のスケジューラーで管理され、特定のコアで実行することを制御できなかった


```
#pragma omp parallel for
    for (i =0; i < N; i++){
    .....
    .....
    .....
   }
```

8 コア・システムで 4 スレッドを実行した場合

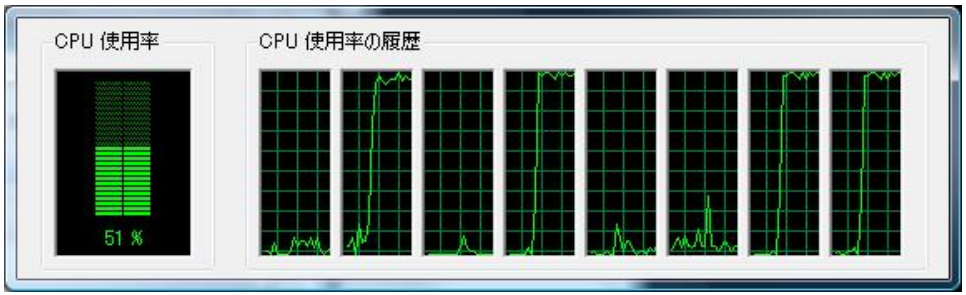
10.1 コンパイラーの新機能 – OpenMPでのアフィニティーの設定（続き）

環境変数 KMP_AFFINITY で OpenMP スレッドが使用するコアのアフィニティーを設定可能



KMP_AFFINITY=compact

```
#pragma omp parallel for
   for (i =0; i < N; i++){
   …..
   …..
   …..
  }
```



KMP_AFFINITY=granularity=fine,compact,0,1

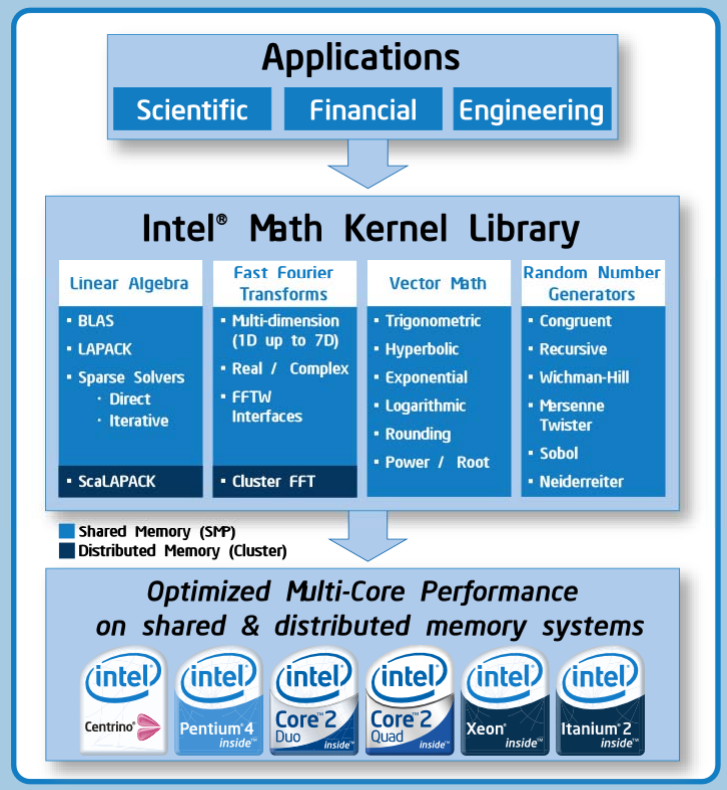
インテル® マス・カーネル・ライブラリー 10.0

業界でもトップレベルの数値演算ライブラリー

- 簡単にマルチスレッド・アプリケーションを開発
  - 優れたスケーラビリティーを実現するスレッド化数値演算関数
  - ***新機能*** ベクトル数値関数をスレッド化
  - ***新機能*** OpenMP 互換ライブラリーをサポート(MSC および GNU でのOpenMP 実装で利用可能)
- アプリケーションの性能を最大限に
  - アプリケーション実行時にプロセッサーを自動認識し、最適なライブラリー・コードを実行することでパフォーマンスを向上
  - ***新機能*** インテル® Core™2 Extreme QX9650 クワッドコア・プロセッサーを含む最新のインテル・プロセッサーに最適化
  - ***新機能*** クラスター機能を標準実装
- 1つのツールで 32/64ビットのアプリケーション開発に対応(Windows*、Linux* そして Mac OS* X 対応)
  - 既存の Windows*、Linux* および Mac OS* X 開発環境に完全互換

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

> "We achieved a 300-500% performance gain using the Intel Math Kernel Library PARDISO Solver with our NISA Finite Element Analysis tool during our trials on multi-core machines."
>
> *Ramdass Keshavamurthy*
> *VP Development, Cranes Software*

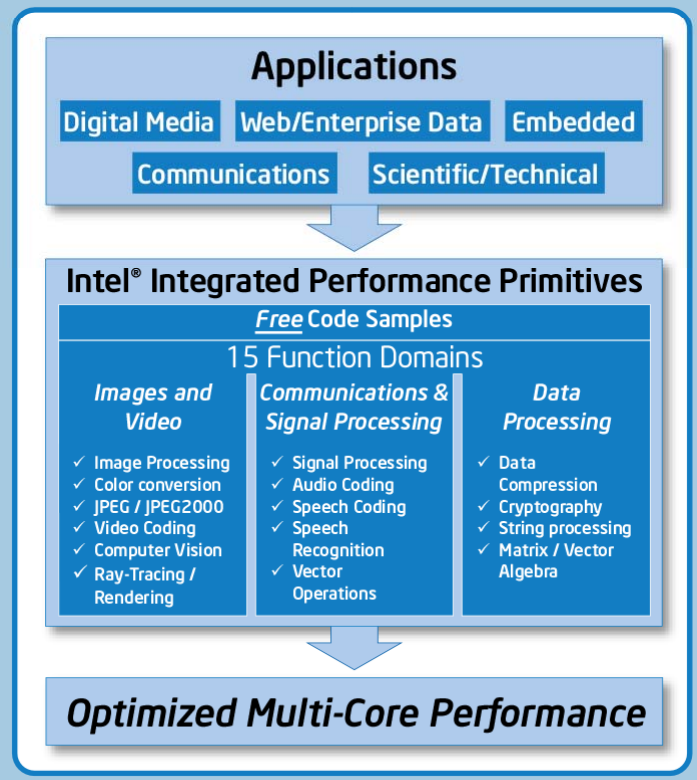


インテル® インテグレーテッド・パフォーマンス・プリミティブ 5.3

> "I recently compared Intel IPP to the open source library zlib. I observed a considerable performance improvement (at least 10-15%) using IPP compared to zlib on the latest Intel processors. I've concluded that we should migrate our open source part of the project to utilize IPP."
>
> *Vadim Kavaleroy*
> *Investment Analyst*
> *Susquehanna International Group*

イメージ、ビデオ、通信、信号およびデータ・プロセス処理をサポートする大規模配列を扱う関数群

- 簡単にマルチスレッド・アプリケーションを開発
  - 優れたスケーラビリティーを実現するスレッド化された関数群
- アプリケーションの性能を最大限に
  - アプリケーション実行時にプロセッサーを自動認識し、最適なライブラリー・コードを実行することでパフォーマンスを向上
  - VC-1 および H.264 HD ビデオ・コーデックをサポート
  - データ圧縮の性能を改善
  - ***新機能*** zlib および libbzip2 完全互換のデータ圧縮関数とサンプル・コードを提供
  - ***新機能*** インテル® Core™2 Extreme QX9650 クワッドコア・プロセッサーを含む最新のインテル・プロセッサーに最適化
- 1つのツールで 32/64ビットのアプリケーション開発に対応（Windows*、Linux* そして Mac OS* X 対応）
  - 既存の Windows*、Linux* および Mac OS* X 開発環境に完全互換



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

インテル® スレッド・チェッカー v3.1

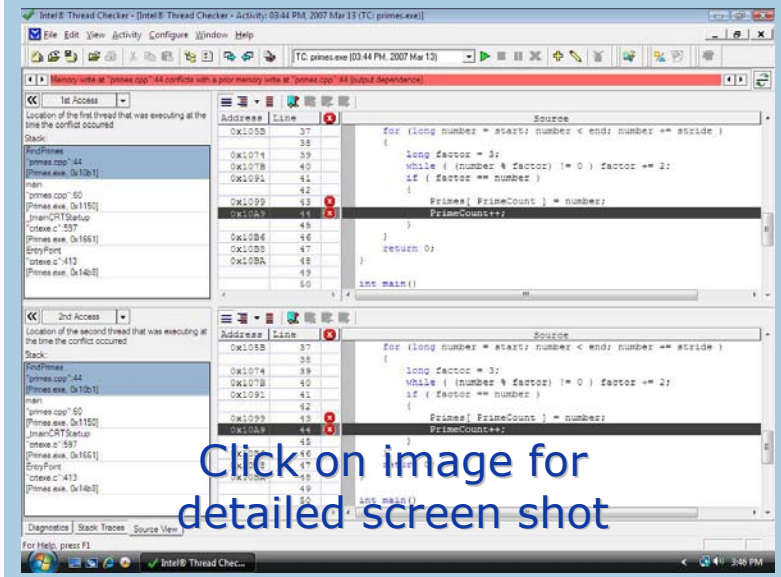
スレッド化の問題を確実に検出

- 簡単にマルチスレッド・アプリケーションを開発
  - データ競合やデッドロックなどの問題を発生する前に検出
  - ソースコードの行レベルでエラーを指摘
  - 再コンパイルなしで標準のデバッグ・ビルドで問題を検出
- 1つのツールで 32/64ビットのアプリケーション開発に対応（Windows*、Linux* 対応）
  - Windows および Linux 版ともにコマンドラインでのインターフェイスを提供
  - 機能テスト等向けにバッチスクリプトへの統合をサポート

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

"We couldn't have gotten the networking up and running as quickly and as efficiently without Thread Checker. Thread Checker is simply an awesome tool and we are not going to develop multi-threaded code without it."

Doug Service, Dir. of Tech. Dev.
Chris Stark, Software Engineer
Ritual Entertainment

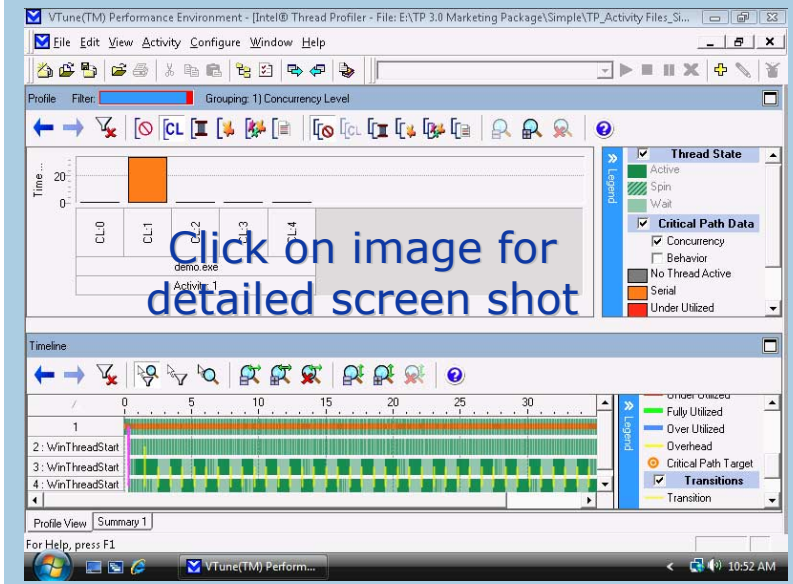


インテル® スレッド・プロファイラー v3.1

> "Intel Thread Profiler was very useful for analyzing bottlenecks in our threaded code. Thread Profiler quickly pinpointed problem areas and showed us the reasons for the slowdown, so we were able to restructure the code for better threaded performance."
>
> *Martin Watt*
> *Software Architect*
> *Alias*

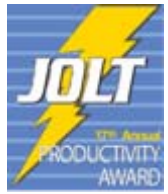
スレッドの性能を最大限に引き出すため スレッド化の問題をポンポイントで指摘

- 簡単にマルチスレッド・アプリケーションを開発
  - 簡単に利用 – カスタム設定の呼び出し
  - すばやく利用 – ユーザーによる選択が可能なスタック解析
  - スレッド・ビルディング・ブロックのAPIをサポート
- アプリケーションの性能を最大限に
  - プロセッサーコアの利用率を確認するため、アプリケーションの平行レベルを表示
  - パフォーマンスに影響するオーバーヘッドがスレッドのどこにあるかを特定
  - アクティブおよびインアクティブなスレッドの状態を検出
- 1つのスレッド・プロファイラーでWindows* 32ビットおよび64ビットに利用可能
- VTune™ アナライザー Windows* 版に同梱



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

インテル® スレッディング・ビルディング・ブロック 2.0（インテル® TBB）



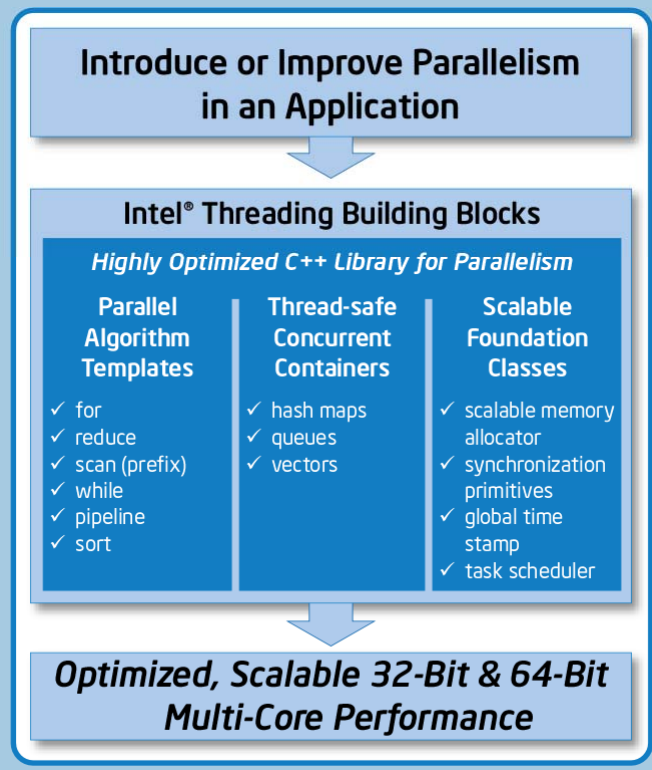
スケーラブルな高性能アプリケーションを容易に開発するため、30年にわたるスレッド実装の知識から生まれた製品

- 簡単にマルチスレッド・アプリケーションを開発
  - よく知られている C++ のテンプレート・ライブラリーを使用して、スレッドそのものではなく、タスクパターンを記述
  - 高度に抽象化された実装により、パフォーマンスのスケーリングを損なうことなく、シンプルなコードを記述可能
  - プロセッサーのコア数が増加してもメンテナンスは最小
- アプリケーションの性能を最大限に
  - 利用可能なプロセッサー・コアに比例してスケール
- 1つのツールで 32/64ビットのアプリケーション開発に対応（Windows*、Linux* そして Mac OS* X 対応）
  - Linux、Windows そしてMac OS で共通のAPIを提供
  - すべての C++ コンパイラーで利用可能（Microsoft、GNU そして インテル）
- 2 つの選択肢
  - サポート付きの商用版
  - オープンソース版
    - www.threadingbuildingblocks.org で入手可能
    - 米国オライリー社から書籍出版 （日本語訳出版予定）

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

"TBB helped KnowledgeMiner achieve speeds 8x faster on an 8 core system. In addition, a strict redesign of KnowledgeMiner for parallel computing is giving a total speedup over the previous version 400x. This astonishing change in speed allows KnowledgeMiner to operate in almost real time something we didn't previously think was possible."

Frank Lemke
Founder and President,
KnowledgeMiner Software



インテル・プレミア・サポート

インテルのエキスパートからのテクニカルサポート

ご購入されたインテル® ソフトウェア開発製品には、一年間の無制限プレミア・サポートが付属します

インテル® プレミア・サポートで提供されるサービス:

- すべてのインテルソフトウェア開発製品のサポート
- インテル プレミア・サポートのWebサイトへのオンラインアクセス
- 問い合わせと経過追跡
- 製品アップデートと関連情報のダウンロード
- ユーザーフォーラムなど

> "Registering for support was easy, and we value the security of knowing that Intel is there to help, even though we haven't needed it so far."
>
> *Rob Hoffmann*
> *Director of Marketing*
> *NewTek, Inc*

まとめと次のステップ

インテル® ソフトウェア開発製品
アプリケーションの並列化に必須な製品

- マルチスレッド化および MPI クラスターを含む並列プログラムの分析、導入、デバッグ、チューニングをサポート
- 既存のビルドプロセスをサポート
- ソースおよびバイナリー互換を提供
- クロスプラットフォーム (プロセッサー、OS) をサポート

次のステップ:

是非製品をご評価ください。
www.intel.co.jp/jp/software/products/


本日はありがとうございました
intel
Software
Products

わずかなコードの追加で並列化を実現
例: 3D 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;
```

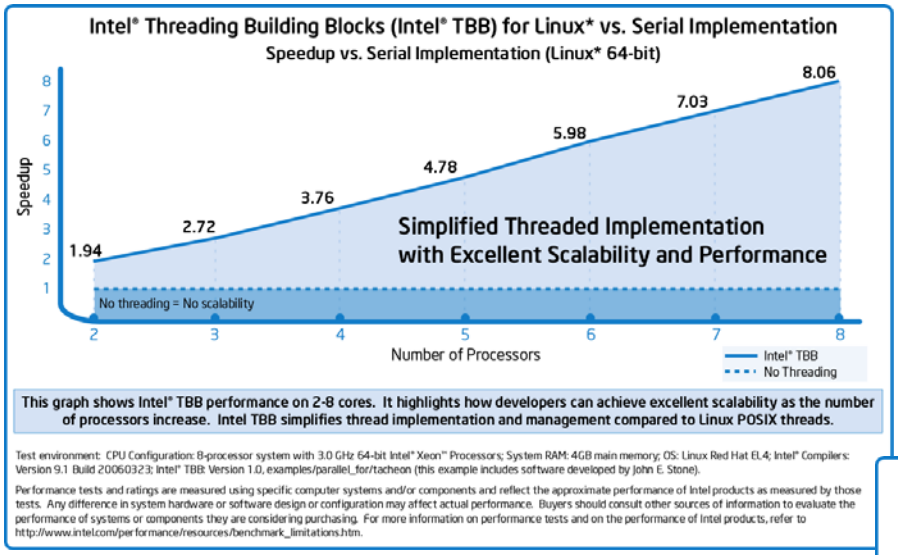
並列タスクのスケジューリングと実行

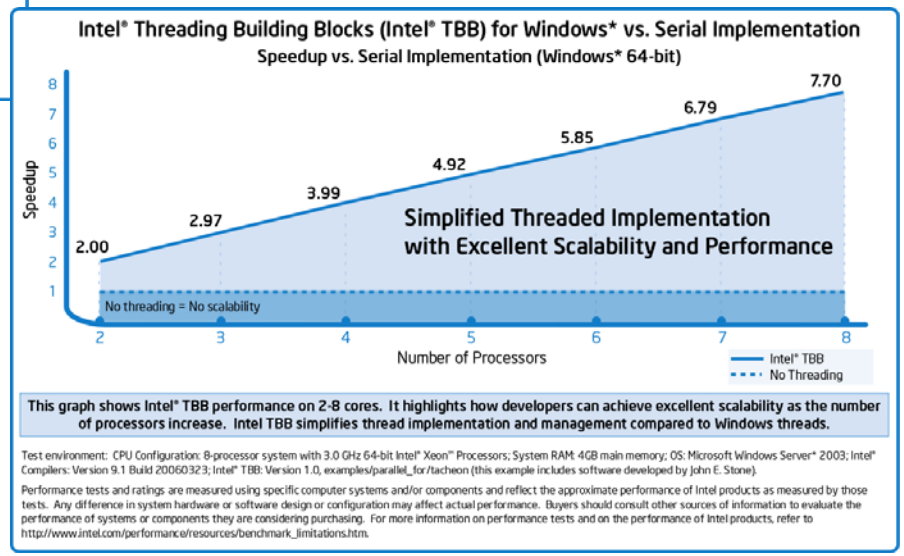
```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range2d.h"
class parallel_task {
public:
   void operator() (const tbb::blocked_range2d<int> &r) const {
      for (int y = r.rows().begin(); y != r.rows().end(); ++y) {
         for (int x = r.cols().begin(); x != r.cols().end(); x++) {
            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* X 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。

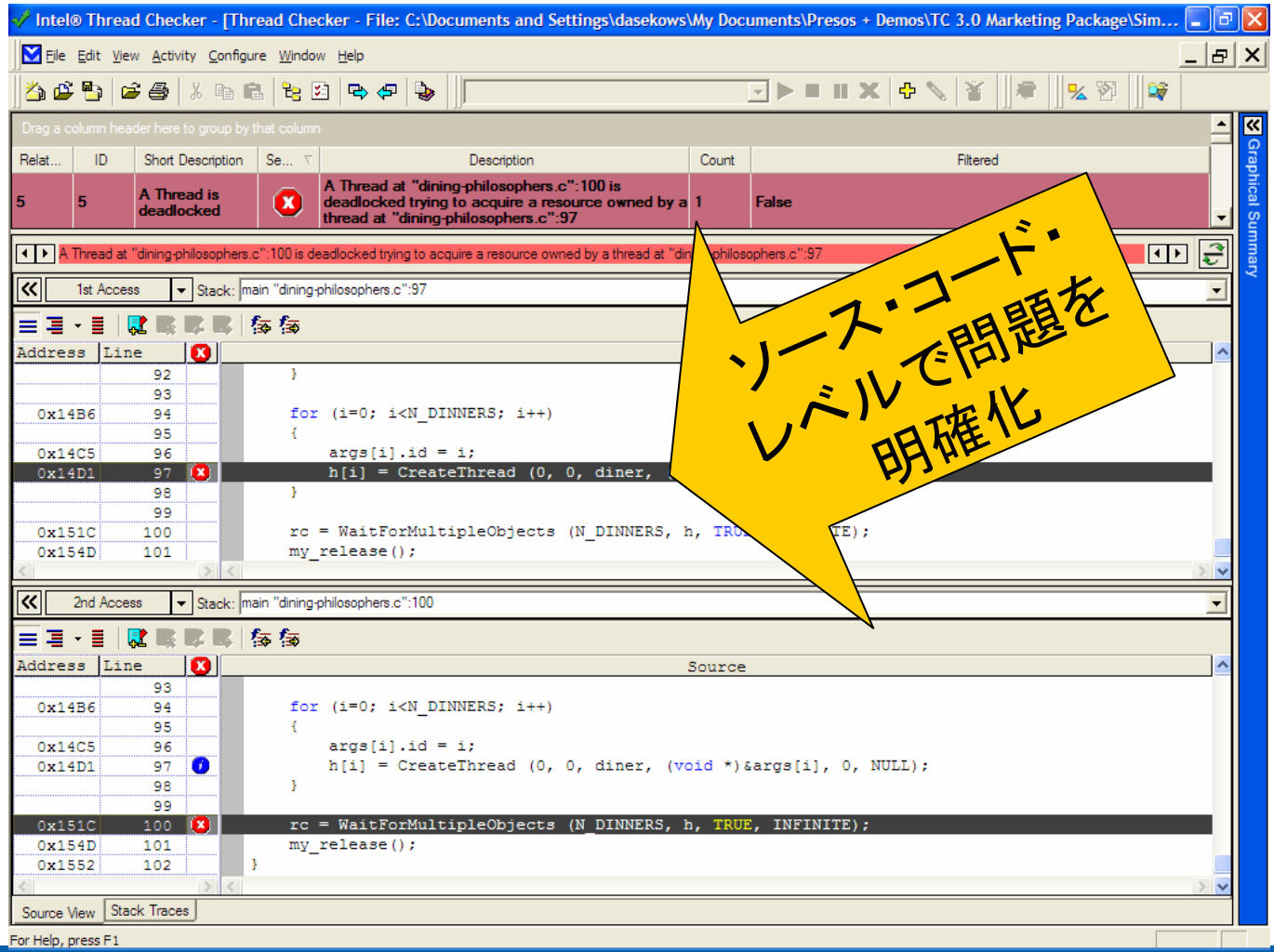
ネイティブスレッドを使用した場合とのスケーリング比
ベンチマーク: 3D レイ・トレーシング・アプリケーション





インテル® スレッドチェッカー Windows* 版

スレッドの問題をピンポイントで特定



インテル® スレッド・プロファイラー
非効率なスレッディング問題をピンポイント