THE PARALLEL UNIVERSE

Issue 9
2012 年 2 月

DreamWorks Animation*
ファーシェーダーの並列化
Sheng Fu

編集者からのメッセージ
James Reinders

インテル® Cluster Studio XE の新しい解析ツール
David Mackay, Ph.D.
Krishna Ramkumar

HPC wire Editors' Choice Awards 2011

intel

究極のパフォーマンスに挑戦しよう

2011 Editor's Choice Award 受賞：最高の HPC ソフトウェア製品 / 技術

インテル® Parallel Studio XE は、業界最高レベルの C/C++ コンパイラーと Fortran コンパイラー、パフォーマンス・ライブラリーと並列ライブラリー、エラーチェッカー、パフォーマンス・アナライザーなどのテクノロジーを組合わせたツール・スイートです。

ハイパフォーマンス・コンピューティング・アプリケーションおよびエンタープライズ・アプリケーションのパフォーマンスを引き出し、コードの品質、セキュリティー、信頼性を向上させます。

インテル® Parallel Studio XE の詳細

© 2012 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国およびその他の国における Intel Corporation の商標です。* 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。

目次

編集者からのメッセージ
長編映画から高度なクラスターまで並列パフォーマンスを実現

JAMES REINDERS 4

データ並列構造を形状ジェネレーターに適用し、大規模な並列アプリケーションの正当性とパフォーマンスを解析してその影響を検証します。

DreamWorks Animation* のファーシェーダーの並列化
インテルのツールによる大規模アプリケーションの並列化の支援

SHENG FU 6

インテルの並列化ツールを利用して DreamWorks Animation* のファーシェーダーが、制作に必要な既存のライブラリーを維持しつつ、マルチコア・プロセッサーのパフォーマンスと機能を活用した方法を紹介します。

インテル® Cluster Studio XE の新しい解析ツール

DAVID MACKAY, PH.D.、KRISHNA RAMKUMAR 18

インテル® VTune ™ Amplifier XE を含む新しいクラスターツールは、ハイブリッド･アプリケーションにおいて優れた解析を実現します。ハイブリッド・アプリケーションで使用される共有メモリー・プログラミング・モデルに関係なく、分散型プログラムの各ノードの詳細なアクティビティーの解析を基に、MPI プログラムをより正確に、容易にチューニングできます。

© 2012 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Cilk、VTune、Xeon は、アメリカ合衆国およびその他の国における Intel Corporation の商標です。* 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。

並列パフォーマンス

長編映画から高度なクラスターまで可能性を拡大

編集者からのメッセージ

James Reinders は、インテル コーポレーションのチーフ・エバンジェリストです。『Intel Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』（日本語、中国語、韓国語の翻訳版があります）などの並列化に関する文献を発表しています。また、並列化に関する幅広いインタビューも受けています。

インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice/#opt-jp を参照してください。

子供達に自分がどのような仕事をしているかを説明するとき、子供達の好きな映画に連れて行って、その作品の制作にかかわっているといえたらどんなに簡単でしょう。インテル® ツールへの取り組みを通じて数々の映画を支援することで、私は長年にわたってそのような機会に幾度となく恵まれてきました。今回は、その素晴らしい経験を皆さんにご紹介できることを嬉しく思います。

ジャイアント･パンダ、ぐうたらドラゴン、ハイブリッド･アプリケーション、クラスター解析、これらすべてに共通するものが何か分かりますか？これらは、インテルの最新の並列プログラミング・ツールにより可能になった革新技術であり、今回の **Parallel Universe** で詳しく紹介します。

DreamWorks Animation* では、アニメーターが 200 を超えるファー（毛などのふさふさした質感を表現する）属性を作成し、開発者はそれらをループやスレッド間で簡単に移動して利用します。

DreamWorks Animation* のファーシェーダー（毛などのふさふさした質感を表す手法）は、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）、インテル® マス・カーネル・ライブラリー（インテル® MKL）、インテル® Inspector XE 2011、インテル® C++ Composer XE のコード・カバレッジ・ツール、インテル® VTune™ Amplifier XE などのツールを利用して、データ並列処理から形状ジェネレーターまで表現します。ツールが提供する解析情報をほかのドメインにも活かして、依存ライブラリーをスレッドセーフにするとともにマルチコア･プラットフォームでスケーラビリティーを向上できます。

新しい解析ツールであるインテル® Cluster Studio XE は、ファーのテクスチャーや草の葉といったミクロの世界から、クラスター・パフォーマンスの向上といったマクロレベルまで対応します。インテル® Inspector XE およびインテル® VTune™ Amplifier XE は、何百何千ものプロセッサーを搭載したシステムでも、システムの解析を支援します。最新バージョンでは、クラスター全体にわたってこれらのツールを利用し、大規模な並列アプリケーションの正当性とパフォーマンスを解析して、各ノードの動作を効率良く理解できます。プロセッサーが 1 つであっても、数千であっても、これらのツールについて理解し、利用する価値があるでしょう。

インテルのハイパフォーマンス・ツールにより、我々が制作にかかわった映画は素晴らしい出来になりました。今回はその制作過程を紹介します。さらに、プログラムの動作を良く理解し、スレッド化、メモリーおよびパフォーマンスの問題を特定する 2 つのツールも紹介します。ぜひ映画館へ足を運び、ポップコーンを頬張りながら、皆さんの知識で未来のコンピューター科学者達の想像力をかきたててください。

James Reinders
2012 年 2 月

 インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice/#opt-jp を参照してください。

DreamWorks Animation* のファーシェーダーの並列化

インテルのツールによる大規模アプリケーションの並列化の支援

ソフトウェア & サービス
Sheng Fu

コンピューター・アニメーション映画では、3D オブジェクトをよりリアルに見せるため、ファー表現は非常に重要です。DreamWorks Animation* で開発され、制作に使用されているファーシェーダーは、毛皮、髪の毛、草などのさまざまなファー表現をシミュレーションする強力な形状ジェネレーターです。ファーシェーダーは計算リソースを大量に消費する傾向にありますが、それぞれのファーの計算は個別のデータセットに対して行われ、互いに独立しているため、データ並列化の有力な候補でもあります。ここでは、DreamWorks Animation* がインテルの並列化ツールを使用して、決定性のある結果を生成し、依存ライブラリーをスレッドセーフにし、マルチコア・プラットフォームでスケーラビリティーを向上するべくファーシェーダーを並列化した方法について説明します。この DreamWorks Animation* が使用したファーシェーダーの並列化テクニック、ツール、および手法は、ほかの分野のアプリケーションの並列化にも適用できる一般的なものです。

はじめに

コンピューター・アニメーション映画では、3D オブジェクトをよりリアルに見せるため、ファー表現は非常に重要です。DreamWorks Animation* で開発され、制作に使用されているファーシェーダーは、毛皮、髪の毛、草などのさまざまなファー表現をシミュレーションする強力な形状ジェネレーターです。ファーシェーダーは、制作に必要な数百万以上の毛皮、髪の毛、草を表現するため、計算リソースを大量に消費する傾向にあります。ほんのわずかなファー表現のシーンであっても、レンダリングには数時間もかかります。図 1 から 3 は、DreamWorks Animation* によって制作された人気映画のシーンの例です。

DreamWorks Animation* のファーシェーダーは、それぞれのファーの計算を個別のデータセットに対して行い、互いに独立しているため、データ並列化の有力な候補です。しかし、ファーシェーダーは非常に複雑で、微調整可能な 200 を超えるファー属性、数千行ものコード、その他の DreamWorks Animation* ライブラリーとの依存性も考慮しなければなりません。ここでは、DreamWorks Animation* がインテル® TBB、インテル® MKL、インテル® Inspector XE 2011、インテル® C++ Composer XE のコード・カバレッジ・ツール、インテル® VTune™ Amplifier XE などのツールを使用してファーシェーダーを並列化し、依存ライブラリーをスレッドセーフにし、マルチコア・プラットフォームでスケーラビリティーを向上させた方法について説明します。

この DreamWorks Animation* のファーシェーダーの並列化テクニック、ツール、および手法は、ほかの分野のアプリケーションの並列化にも適用できる一般的なものです。

図 1:「カンフーパンダ」の草地のシーン

図 2:「ヒックとドラゴン」の髪の毛を含むシーン

図 3:「カンフーパンダ」の毛皮を含むシーン

ファーシェーダーの概要

ファーシェーダーは形状ジェネレーターです。表面に多数の細かいファーを生成し、ディテールを追加します。それぞれのファーに対して、図 4 に示すステップを実行します。

図 4: ファーシェーダーを介してファーを生成するステップ

形状サンプリング・ステップでは、基となる形状からランダムに場所を選択し、その位置、テクスチャー座標、関連要素を含む形状属性を評価します。

1 つのテクスチャーに多くのファー属性を結び付けられます。テクスチャー・サンプリング・ステップでは、テクスチャー・マップを調べてファー属性を評価します。

ファーの補間ステップでは、現在のファーに隣接するファーを見つけて、そのファーの形状の補間に基づいて現在のファーの形状を決定します。

スタイル調整ステップでは、縮れ、カール、重力、風といった属性を追加してファーの見た目を変更し、ファーがリアルに見えるようにその他の細かいディテールを追加します。

明らかに各ファーの計算コストは非常に高く、制作には膨大な量のファーが必要です。ただし、それぞれのファーの計算は互いに独立しているため、ファーシェーダーは並列処理の有力候補といえます。

インテル® IPP によるファーシェーダーの並列化

DreamWorks Animation* のファーシェーダーで処理に最も時間のかかるコード領域は、各ファーのファー生成ループです。インテル® ソフトウェア開発製品には、OpenMP*、インテル® TBB、OpenCL* など複数の並列プログラミング・モデルが用意されています。OpenCL* は、既存の C++ オブジェクト指向型アプリケーションに並列処理を実装するため、コードの大幅な書き直しが必要になります。OpenMP* はループの並列化に適しています。`for` ループの前に `pragma` を追加するだけです。しかし、ファーシェーダーのように、コードのほとんどが C++ およびオブジェクト指向型の場合、インテル® TBB のほうが最適でしょう。DreamWorks Animation* では、インテル® TBB をほかのプロジェクトでも幅広く使用していたため、ファーシェーダーの並列化にインテル® TBB が採用されたのも自然な流れでした。

変更は、通常の for ループをインテル® TBB の **`parallel_for`** に変えるという単純なものです。図 5 に、既存のファーコードの構造とインテル® TBB の **`parallel_for`** で必要な変更箇所を示します。インテル® TBB の **`parallel_for`** には 2 つの引数があります。第 1 引数は for ループの範囲を指定する **`blocked_range`** オブジェクトです。第 2 引数はループ本体のオブジェクトで、開発者は少なくともコピー・コンストラクター、デストラクター、operator() を持つループ本体のクラスを定義しなければなりません。図 5 のコードでは、C++ 0x 標準で定義されている C++ ラムダ式を使用しています。ラムダ式を使用すると、コンパイラー (例えば、インテル® C++ コンパイラー) は、**`parallel_for`** の要件を満たすように、必要なループ本体のクラスなどのコードを自動で生成します。ラムダ式の良い点は、式の中で **`parallel_for`** 文よりも前に定義されたローカル変数にアクセスできることです。

この例から分かるように、インテル® TBB を使用すると、インテル® TBB のランタイムがスレッドプールの生成、スレッドへのタスクのスケジューリングなどを処理するため、既存のアプリケーションを容易に並列化できます。

```
void Fur::generateFur()
{
  for(size_t i=0; i<numHairs; i++)
  {
      // それぞれのファーを計算する
  }
}
```

```
void Fur::generateFur()
{
  tbb::parallel_for(
      tbb::blocked_range<size_t>(0,numHairs),
      [=](const tbb::blocked_range<size_t> &r)
      {
          for(size_t i = r.begin() i<r.end(); i++)
          {
              // それぞれのファーを計算する
          }
      } );
}
```

図 5: ファーシェーダーのオリジナルコード (左) とインテル® TBB の `parallel_for` および C++ ラムダ式を使って変更したコード (右)

インテル® MKL の乱数生成を使用してファーシェーダーに決定性を持たせる

ファー生成ループの並列化は簡単ですが、実際に実装する前にその他の重要な問題を解決する必要があります。その 1 つ目は、ファーシェーダーの並列実装には決定性がなければなりません。つまり、使用されるスレッド数やプロセッサー・コア数に関係なく、結果が同じにならなければなりません。

DreamWorks Animation* のファーシェーダーは、ファーがより自然に見えるように、乱数生成関数を多用していました。例えば、ファーシェーダーは、ファーを生成する場所を見つけるため、密度マップをランダムにサンプリングします。さらに、ほとんどのファー属性にはユーザー定義のランダム要素があります。オリジナルの DreamWorks Animation* の実装では、ファーシェーダーは drand48 アルゴリズムの独自の実装を使用して、さまざまな乱数を生成していました。しかし、この乱数生成関数の使用には 2 つの問題がありました。1 つは、独自の実装がスレッドセーフではなかったことです。drand48 の実装をスレッドセーフにするためにミューテックスを使用しても、ファーシェーダーは乱数生成を頻繁に呼び出すため、パフォーマンス・ボトルネックになってしまいました。もう 1 つは、各ファーが並列に生成されるため、並列実行ごとにそれぞれのファーに対して生成される乱数が異なり、並列実行ではファーの見た目が異なってしまう点です。

しかし、ファーシェーダーのコードでは、各ファーに必要な乱数の最大値が固定されており、ファー生成ループの前に分かることが判明しました。シード値が指定されると一連の疑似乱数が固定されるため、それぞれのファーはファー ID により同じ乱数シーケンスを使用することができます (ファー ID は、シリアル実行 / 並列実行に関係ないループの反復番号)。例えば、ファーを 10 個生成し、それぞれのファーに対して乱数が 10 個必要な場合、ファーの乱数は次のようになります。

乱数: r1、r2、……r10、r11、r12,……r20、…….. r91,、r92、……r100

ファー ID: hair_1 hair_2 hair_10

k 番目のファーの乱数シーケンスの開始オフセット:

hair_k のオフセット = k * (ファーごとに必要な乱数の最大値)

決定性のある実装の 1 つとして、ファー生成ループの前にすべての乱数を生成して配列に格納しておき、後で hair_id のオフセットで使用することができます。ただし、数百万ものファーを生成するため、並列処理に依存せず順序が保証され、必要に応じて乱数を生成できる実装のほうが、よりメモリー効率が良いでしょう。インテル® MKL は、さまざまなニーズに対応した多数の乱数ジェネレーター (RNG) 関数をサポートしています。これらの RNG 関数はスレッドセーフでロックを使用しません。さらに、インテル® MKL にはストリーム / シーケンス中の乱数を特定の数だけスキップし、必要に応じて乱数を生成する関数があります。これこそ、ファー生成の並列処理に必要なものでした。図 6 にファーシェーダーでインテル® MKL の乱数関数を使用しているコード領域を示します。

```
// ベースの乱数ストリームを生成する
vslNewStream(&myBaseRandomBStream, VSL_BRNG_MCG59, seed);

...

// ファー生成ループの本体のコード

for each hair do
{
  vslCopyStream(&currentRandomStream, myBaseRandomStream);
  vslSkipAheadStream(currentRandomStream, hair_id * offset_for_each_hair);

  //必要に応じて 1 つの乱数を取得すると仮定する
  double random_number;
  vdRngUniform(VSL_METHOD_DUNIFORM_STD, currentRandomStream,
               1, &random_number, range_low, range_high);

  // ファー生成コードで乱数を使用する
  ...
}
```

図 6: インテル® MKL の RNG を使用して決定性のあるファーの結果を生成する

インテル® MKL 関数 `vslSkipAheadStream` は N 個先の乱数へジャンプし (N は指定した数)、シーケンス / ストリーム中のそれ以前の乱数は計算しません。この機能は、ファー生成の並列処理に決定性を与えました。

インテル® Inspector XE とコード・カバレッジ・ツールを併用してスレッドの安全性問題を検出

DreamWorks Animation* のファーシェーダーのロジックは、アーティストによる 256 を超える属性のチューニングを含む非常に複雑なものです。ファーシェーダーのコードだけでも 3 万行を超え、DreamWorks Animation* のほかのライブラリーを 10 個以上も使用しています。例えば、ファーシェーダーは形状ライブラリーを使用して、メッシュ、NURBS 面、細分面などの異なる種類の形状をサンプリングします。これらのライブラリーの一部は、長年にわたって開発され管理されており、並列に実行するように設計されていません。我々の次の課題は、ファーシェーダーとファーシェーダーで使用する既存のライブラリーのコードパスをスレッドセーフにすることでした。

並列化したファーシェーダーがスレッドセーフであるかどうか確認するには、インテル® Inspector XE が非常に役立ちました。インテル® Inspector XE は、シリアル / マルチスレッド・アプリケーションの開発において動的にメモリーエラーとスレッドエラーを検出するツールです。テストケースを定義し、インテル® Inspector XE で実行すると、アプリケーション中のデータ競合とデッドロックが検出され、それらが発生した際のコールスタックが記録されます。

図 7 のインテル® TBB の **`parallel_for`** サンプルコードは、インテル® Inspector XE の主要機能を示しています。

このサンプルコードでは、複数のスレッドが保護なしで変数 `g` の読み取りと書き込みを行えるため、グローバル変数 `g` でデータ競合が発生します。

```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range.h"
using namespace tbb;
float g    = 0.0f;
int   flag = 1;

class my_tbb_test {
private:
  float *a;
public:
  void operator() (const blocked_range<size_t> &r) const
  {
      for (size_t i=r.begin(); i<r.end(); i++)
      {
          g ++;
          if(flag) {a[i] += g;}
          else {a[i] -= g;}
      }
  }
  my_tbb_test(const my_tbb_test &other)
  {
      a = other.a;
  }
  my_tbb_test(float *a)
  {
      this->a = a;
  }
};

void func_foo()
{
  printf("Test function for code coverage\n");
}

int main()
{
  float data[10000];
  memset(data, 10000*sizeof(float), 0);
  parallel_for(blocked_range<size_t>(0, 10000),
               my_tbb_test(&data[0]));
  return 0;
}
```

図 7: インテル® Inspector XE の主要機能を示す parallel_for の例

「並列化したファーシェーダーがスレッドセーフであるかどうか確認するには、インテル® Inspector XE が非常に役立ちました。インテル® Inspector XE は、シリアル / マルチスレッド・アプリケーションの開発において動的にメモリーエラーとスレッドエラーを検出するツールです。」

インテル® Inspector XE でアプリケーションの解析を行うには、アプリケーションのコンパイル時に、インテル® コンパイラーのマクロ “TBB_USE_THREADING_TOOLS” を定義してください。インテル® コンパイラーのコマンドラインで図 8 のように指定します。

```
icpc -o tbb_for tbb_for.cpp -tbb -DTBB_USE_THREADING_TOOLS -O2 -g
```

図 8

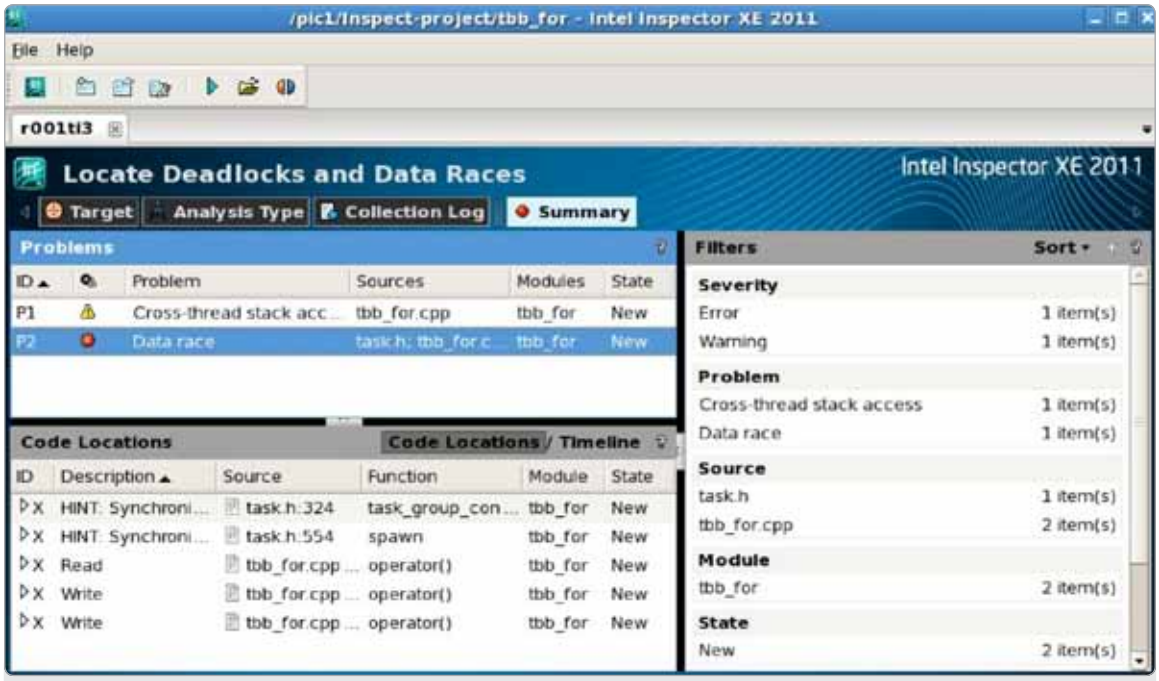


図 9: テストプログラム中のデータ競合を示すインテル® Inspector XE

コンパイル後、インテル® Inspector XE を起動し、アプリケーションを実行してデッドロックとデータ競合を検出します。図 9 は、解析完了後のインテル® Inspector XE のスクリーンショットです。

この例では、インテル® Inspector XE はデータ競合をレポートしています。データ競合の行を選択すると、そのデータ競合が発生したときのコールスタックが表示されます (図 10 を参照)。

データ競合のコールスタックから、通常、開発者はデータ競合が発生した原因を容易に見つけることができます。

インテル® Inspector XE はランタイム解析のみを実行するため、検出される問題は開発者が定義したテストケースに完全に依存します。スレッドセーフではないコードパスが実行されないと、インテル® Inspector XE では問題を検出できません。コードの開発者にとって適切なテストケースを作成することはそんなに難しいことではありません。しかし、ファーシェーダーのように長年にわたってさまざまな開発者達が携わってきた大規模なコードの場合、包括的なテストケースを定義することは非常に困難でしょう。

幸いにも、インテル® Composer XE に含まれる C++ コンパイラーで提供されるコード・カバレッジ・ツールを使用して、テストケースでカバーされるコードパスの範囲を解析することが可能です。単純なインテル® TBB の parallel for コードを例に考えてみましょう。コード・カバレッジ解析を有効にするには、図 11 のコンパイラー・オプションが必要になります。

```
icpc -o tbb_for tbb_for.cpp -tbb -g -prof-gen=srcpos -prof-dir ./code_coverage
```

図 11

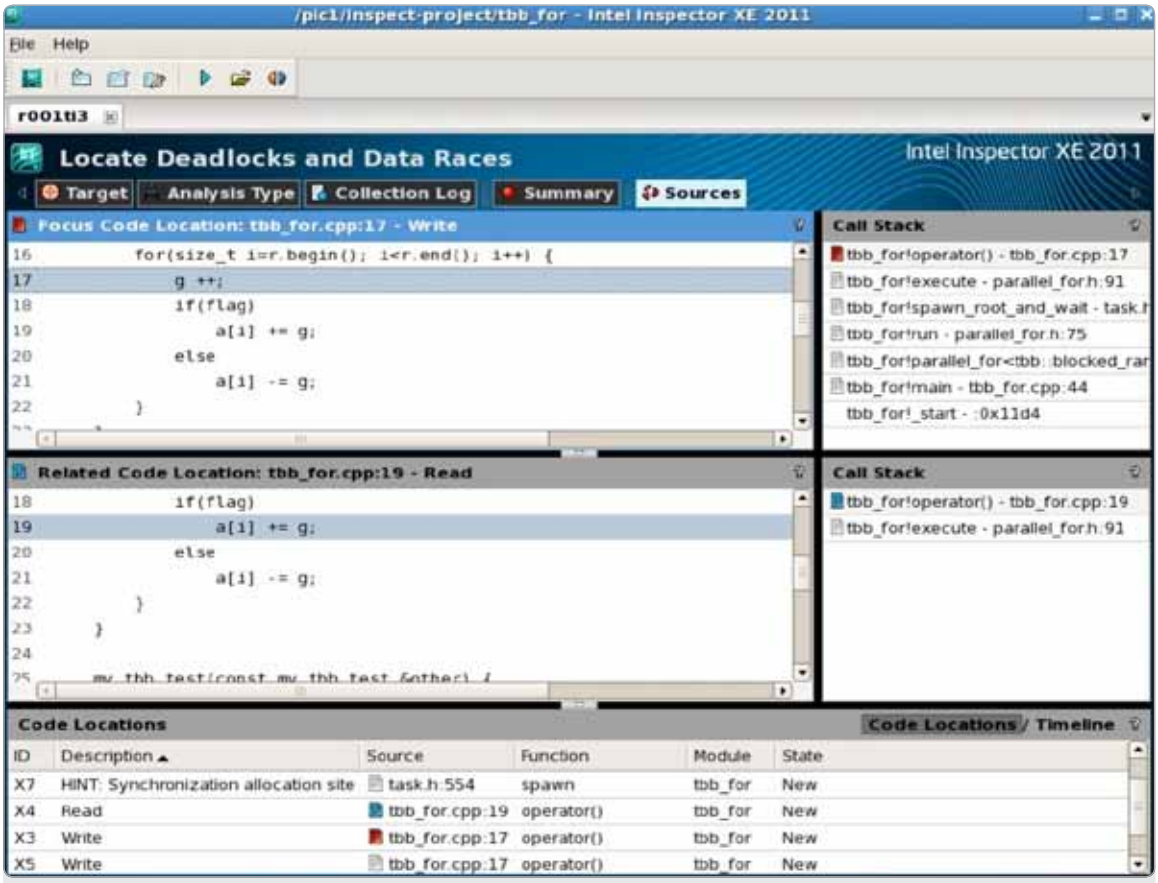


図 10: データ競合のコールスタック

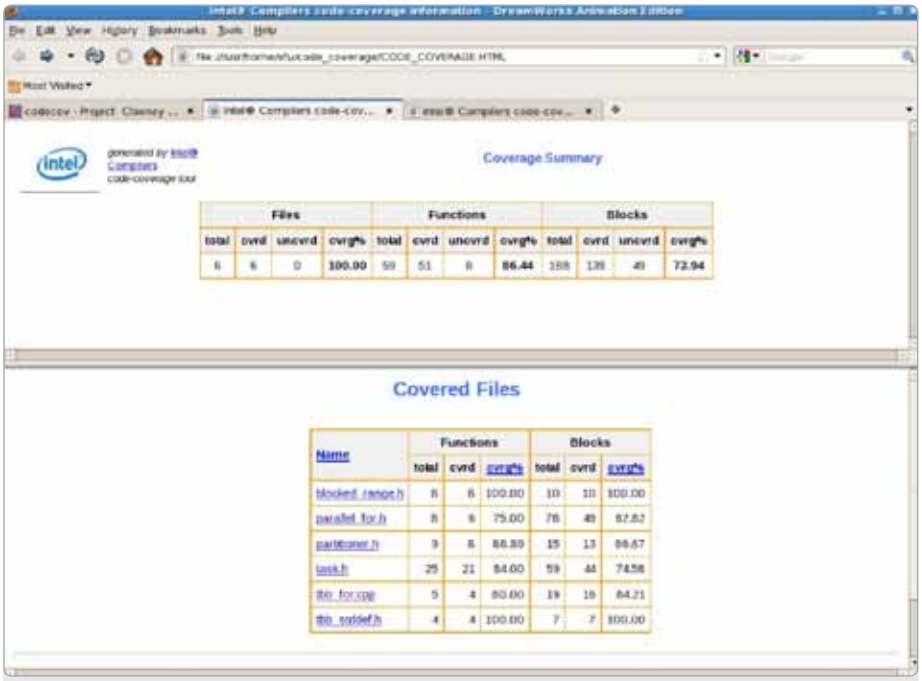


図 12: インテルのコード・カバレッジ・ツールの解析結果

「一般に、大規模なコードを迅速にスレッド化するには、関数がスレッドセーフかどうか明らかでない場合は、ミューテックスを使用してその関数全体を保護しなければなりません。」

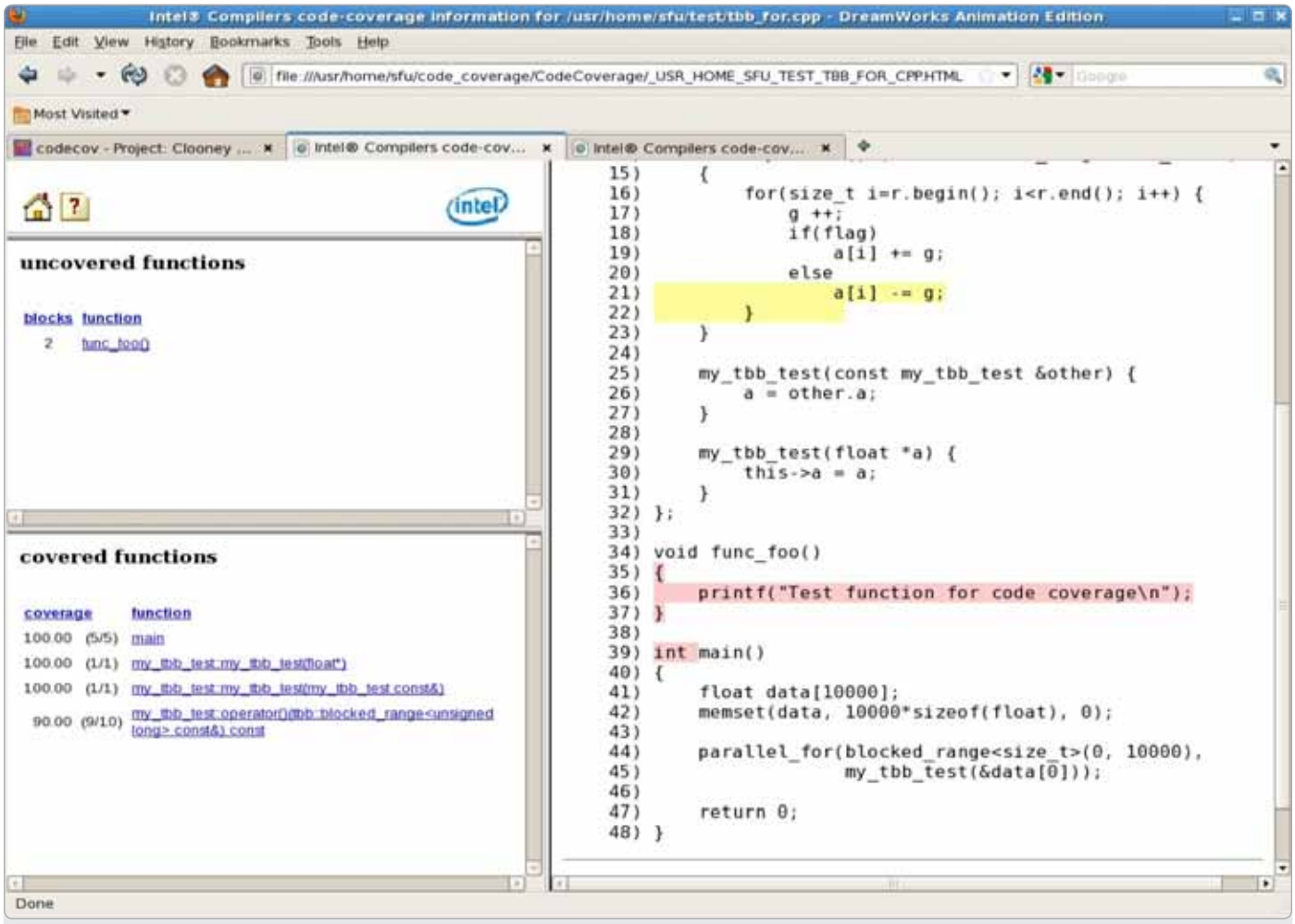


図 13: “tbb_for.cpp” ファイルに対するインテルのコード・カバレッジ解析の結果

“prof-gen” オプションはコード・カバレッジ解析を有効にし、“prof-dir” オプションはコード・カバレッジ解析の結果を保存するディレクトリーを設定します。

これらのコンパイラー・オプションを指定してビルドしたアプリケーションを、複数のテストケースで実行してみましょう。実行するたびに、さまざまなプロファイル情報を含む複数のファイルが ./code_coverage ディレクトリーに生成されます。次に ./code_coverage ディレクトリーに移動して “profmerge” を実行します。インテル® コンパイラーに含まれるこのコマンドは、複数の *.dyn ファイルを 1 つの “pgopti.dpi” ファイルにまとめます。最後に “codecov -dpi pgopti.dpi” を実行して html ページ “CODE_COVERAGE.HTML” を生成します。図 12 はコード・カバレッジ解析ページのスクリーンショットです。

ページ左の “tbb_for.cpp” リンクを選択すると、そのファイルの詳細なコード解析結果が表示されます (図 13)。

また左側には、テストケースでカバーされる関数とカバーされない関数が表示されます。右側のハイライトされていないコードは完全にカバーされ、ピンクと黄色でハイライトされているコードはカバーされません(ピンクはカバーされない関数を示し、黄色はカバーされない基本ブロックを示します)。この例から分かるように、インテル® コンパイラーのコード・カバレッジ・ツールを使用することで、さまざまなテストケースによりカバーされるコードパスを把握できます。

「DreamWorks Animation* のファーシェーダーでは、並列化の取り組み全体を通して、インテルのツールが重要な役割を果たしました。」

 インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice/#opt-jp を参照してください。

DreamWorks Animation* のファーシェーダーでは、コード・カバレッジ・ツールを使って約 100 のテストケースを定義し、コードパスが十分にカバーされるようにしました。そして、これらのテストケースを使ってインテル® Inspector XE を実行することで、コードを検査しただけでは見つけることが困難ないくつかのデータ競合を発見することができました。

インテル® Inspector XE で大規模なアプリケーションを実行する場合は注意が必要です。インテル® Inspector XE は大量のコードをインストルメントするため、多くの場合、アプリケーションはインストルメントされていない通常のバージョンと比べてかなり遅くなります。そのため、短時間で実行できるテストケースを設計することが重要です。DreamWorks Animation* のファーシェーダーでは、同じファー生成ループで生成されるファーは似たようなコードパスをたどるため、ファー生成ループで生成されるファーの数を減らしました。

スレッドの安全性に関する問題を修正する各種のアプローチ

インテル® Inspector XE で見つかったファーシェーダーのスレッドの安全性に関する問題はすべてデータ競合でした。クラス Fur のメンバー関数は単一のスレッドで実行されるように設計されており、これらの関数はクラス Fur のメンバー変数の読み取り / 書き込みを行います。しかし、ファー生成ループを並列化したことで、これらの関数が複数のスレッドから呼び出されるようになり、クラス Fur のメンバー変数へのアクセスがスレッドセーフでなくなっていました。メンバー変数の使用法に応じて、このようなデータ競合を修正するアプローチがいくつかあります。

1. メンバー変数がメンバー関数のループ中で使用され、ループの全反復を通してそのメンバー変数に依存関係がない場合、ループ文の範囲に限ってそのメンバー変数をローカル変数にすることができます。例えば、ファーシェーダーのメンバー変数 mOutputHair は、生成されたファーの最終結果を保存し、テッセレーションへ渡します。各スレッドでこの変数のコピーが必要になりますが、複数のスレッドで共有されることはないため、この変数はループ内でローカル変数に置き換えることができます。さらに、ファー生成ループ内のメンバー関数が mOutputHair にアクセスする場合、mOutputHair を関数引数として受け入れるようにそのメンバー変数を変更する必要があります。
2. メンバー関数が複数のスレッドにより共有されるものの、整数型、列挙型、またはポインター型で、“++” や “- - “ のような限られた算術演算だけを必要とする場合は、インテル® TBB のアトミック変数に置き換え可能なことがあります。アトミック変数は、ロックよりもオーバーヘッドが低いアトミック操作を行います。アトミック操作では、データ競合やデッドロックは発生しません。例えば、mTotoalHairs は生成されたファーの合計数をカウントします。この変数が行う演算は “++” だけであり、図 14 のようなアトミック変数に変換できます。

```
tbb::atomic<int> mTotalHairs;
```

図 14

3. その他すべての共有メンバー変数には、競合状態を防ぐためミューテックスが必要です。スレッドの競合が頻繁に発生する領域でミューテックスを使用すると、大きな代償を伴うことがあります。ファーシェーダーでは、論理的にそれぞれのファーは個別に処理されるため、そのようなミューテックスは必要ありませんでした。

前述のとおり、DreamWorks Animation* のファーシェーダーは、形状ライブラリーなどのさまざまなライブラリーに依存しています。これらのライブラリーの一部には、スタティック変数が含まれています。例えば、NURBS 面を評価する関数では、スタティック変数 fPrecision が評価の精度を制御します。競合状態を回避するために fPrecision を関数引数として渡すことも可能ですが、通常、このようなライブラリーは異なるアプリケーションで幅広く使用されており、それらすべてで関数のインターフェイスに新しい引数を追加するにはかなり時間がかかります。このような場合は、スレッドローカル変数を使用すると良いでしょう。スレッドローカル変数への変更は簡単で、変数の宣言にプリフィックス “__thread” を追加するだけです。図 15 に例を示します。

```
__thread float fPrecision;
```

図 15

スレッドローカル変数は、通常の変数と同じようにアクセスできます。コンパイラーはスレッドごとにこの変数の個別のコピーを保持します。ただし、スレッドローカル変数の使用には注意が必要です。まず、スレッドローカル変数の使用には、隠れたわずかなパフォーマンス・コストを伴うことがあります（詳細は、ブログ http://software.intel.com/en-us/blogs/2011/05/02/the-hidden-performance-cost-of-accessing-thread-local-variables/（英語）を参照）。次に、大きな配列をスレッドローカル変数にするのは避けるべきです。コンパイラーによりこの変数のコピーがスレッドのスタックに割り当てられると、利用可能なスタックサイズが減ってしまいます。

インテル® VTune™ Amplifier XE によるスケーラビリティーの向上

データ競合問題とスレッドの安全性に関する問題を修正し、スレッド化されたファーシェーダーを正しく実行できるようになったら、次にパフォーマンスの評価を行いました。特に、スレッド数 / コア数に応じたパフォーマンスのスケーリングを評価することが重要です。インテル® TBB のタスク・スケジューラーを使用することで、アプリケーションを実行するスレッド数を容易に制御できます。例えば、図 16 のようなコードを記述することができます。

```
tbb::task_scheduler_init init(nThread);
```

図 16

ファーシェーダーのスケーリング・テストでは、がっかりするような結果になりました。全くといっていいほど、プロセッサー・コア数に対応してスケーリングしないことが分かったのです。例えば、ファーシェーダーのパフォーマンスは 2 コアでは約 2 倍にスケーリングしたものの、8 コア / スレッドでも 2 倍を超えませんでした。

インテル® VTune™ Amplifier XE は、このようなスレッドのパフォーマンス・スケーリング問題を見つけるのに役立つ優れたツールです。特に “ロックと待機の解析” は、プロセッサー・コアの使用効率が悪い原因を特定するのに重要です。一般的なパフォーマンスの問題の 1 つは、スレッドがロックなどの同期オブジェクトを

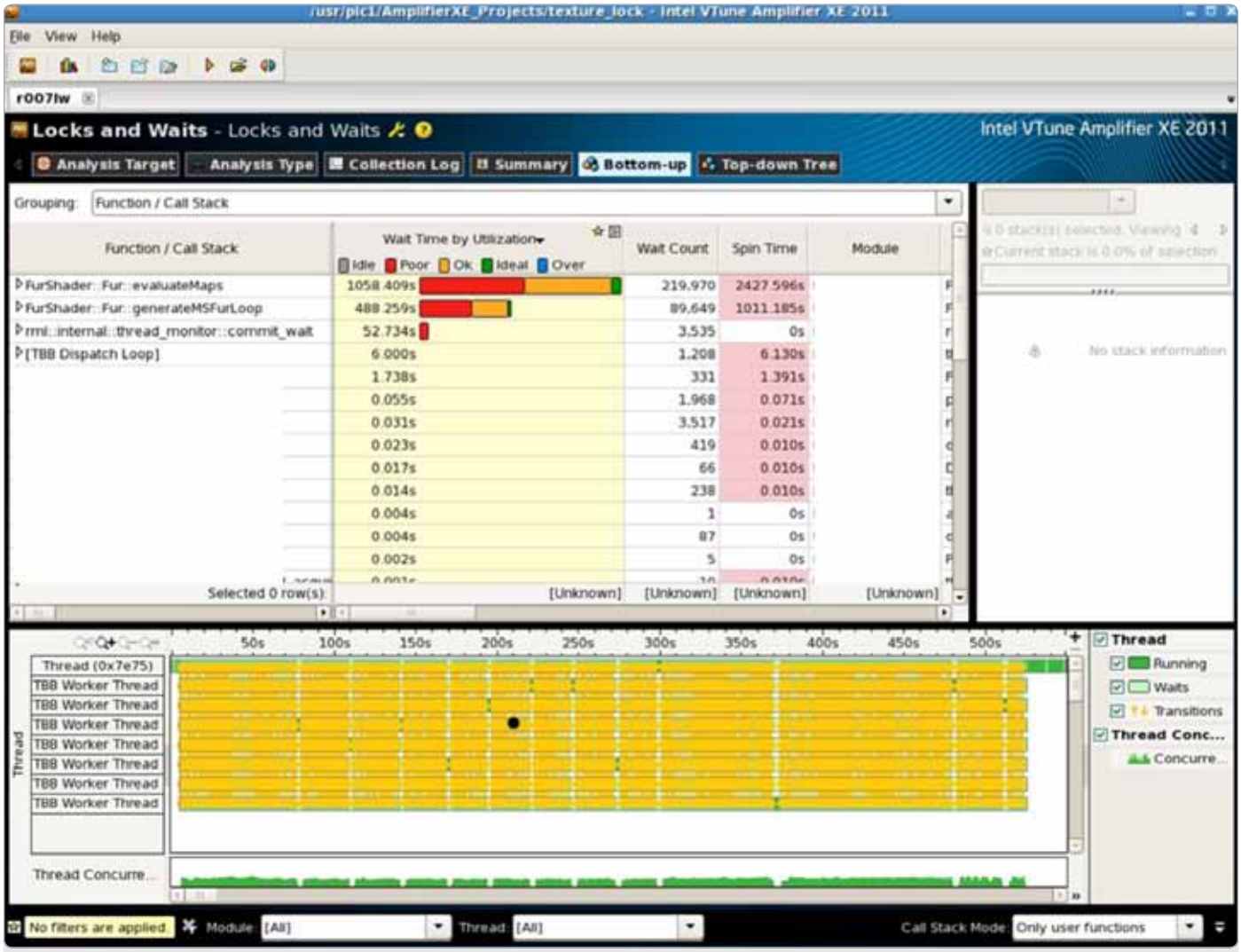


図 17: テクスチャー・サンプリング関数 evaluateMaps のスレッドの競合状態を示すインテル® VTune™ Amplifier XE（黄色のラインはスレッド間の遷移が多発していることを示す）

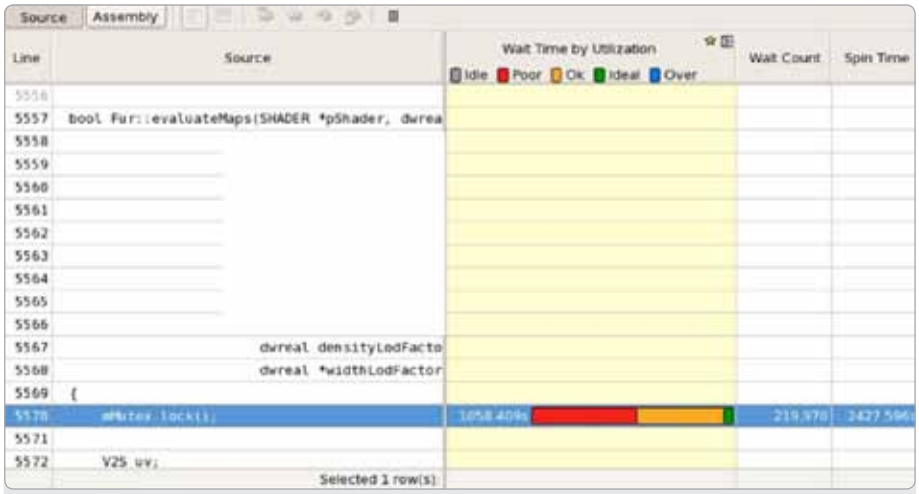


図 18: 図 17 の関数 evaluateMaps をドリルダウンすると表示される、競合が発生したソースコードとロックの場所

待機したり、保護 / ロックされたリソースへのアクセスを競り合うことです。ファーシェーダーでは、“ロックと待機の解析” により (図 17 と 18 を参照)、テクスチャー・サンプリング・ライブラリーのミューテックスで最も頻繁に競合が発生し、最も多くのコストがかかっていることが判明しました。

DreamWorks Animation* のテクスチャー・サンプリング・ライブラリーには、前回のテクスチャー・サンプリング呼び出しの結果を保存するキャッシュがあります。現在のテクスチャー・サンプリング呼び出しのテクスチャー座標が前回の呼び出しと同じ場合、サンプリング関数はただ単にキャッシュに保存されているテクスチャー値を返します。このキャッシュは、スレッドセーフにするためミューテックスで保護されています。しかし、テストしたワークロードでは、テクスチャー・サンプリング関数が頻繁に呼び出され、その結果テクスチャー・キャッシュへのアクセスでスレッドの競合が多発し、hotspot となっていました。このケースでは、ミューテックスの換わりに、テクスチャー・キャッシュをスレッドローカル変数にすることで競合を容易に排除することができました。

一般に、大規模なコードを迅速にスレッド化する際、関数がスレッドセーフかどうか明らかでない場合は、ミューテックスを使用してその関数全体を保護しなければなりません。これは、既存の関数がパフォーマンス・ボトルネックでないことが前提です。インテル® VTune™ Amplifier XE のプロファイリングでミューテックスが hotspot として表示された場合は、より良いソリューションを模索するため、詳細な調査が必要です。このアプローチに従うことで、大規模なアプリケーションを並列化するとき、常にパフォーマンスに重大な影響を与える領域に注目することができます。

まとめ

ここでは、DreamWorks Animation* のファー生成のケースについて、制作に必要な既存の各種ライブラリーを使用しつつ、マルチコア・プロセッサー向けに並列化したさまざまなアイデアを紹介しました。8 コアのインテル® プロセッサー・ベースのワークステーションで DreamWorks Animation* のプロダクション・ショットを使用してこの並列コードのパフォーマンスを評価したところ、ファー生成ループで平均 5 倍のスピードアップを達成しました。ファーシェーダー全体のスピードアップは、実行パス中のシリアル領域があるために、4 倍でした。

DreamWorks Animation* のファーシェーダーでは、並列化の取り組み全体を通して、インテルのツールが重要な役割を果たしました。ファー生成ループの並列化にはインテル® TBB の **`parallel_for`** 構文を使用し、並列化されたファー生成からスレッド数に依存せず独立性および決定性のある結果を得るにはインテル® MKL) の乱数生成関数が役立ち、スレッドの安全性に関連するすべての問題の検出にはインテル® Inspector XE とインテル® コンパイラーのコード・カバレッジ・ツールが役立ち、スレッドのスケーラビリティー問題の検出 / 修正にはインテル® VTune™ Amplifier XE が使用されました。これらのツールがなければ、DreamWorks Animation* のファーシェーダーの並列化は達成できなかったでしょう。

BLOG highlights

Fortran の明示的なインターフェイスについて

開発者向け製品部門
STEVE LIONEL

11 年ほど前に、Visual Fortran Newsletter で Fortran の明示的なインターフェイスという記事を執筆しました。ここ数週間、Fortran プログラマーの多くは明示的なインターフェイスの役割とルールを知らないことを理解してもらうため、たくさんの顧客にこの記事を紹介しなければなりませんでした。しかし、この記事を読み直してみたところ、Fortran 95 からさまざまな点が変わっており、見直しが必要であることに気付きました。

Fortran の " インターフェイス " とは、呼び出し可能なプロシージャーに関する情報を意味します。Fortran 77 には " 暗黙のインターフェイス " しかなく、プロシージャーに関して開発者が指定できるのは関数の戻り値のデータ型だけでした。呼び出し引数の数、順序、型、ランク (次元数) は必ず一致していなければならず、コンパイラーがチェックできるように引数を示す方法はありませんでした。さらに、これは引数の引き渡しには関係ないため、コンパイラーはこれらのことを知る必要がありませんでした。

Fortran 90 になり、突然さまざまなことが複雑になりました。例えば、仮引数に形状引継ぎ配列を指定できるようになり、呼び出しで配列の境界に関する情報を提供しなければならなくなりました ...

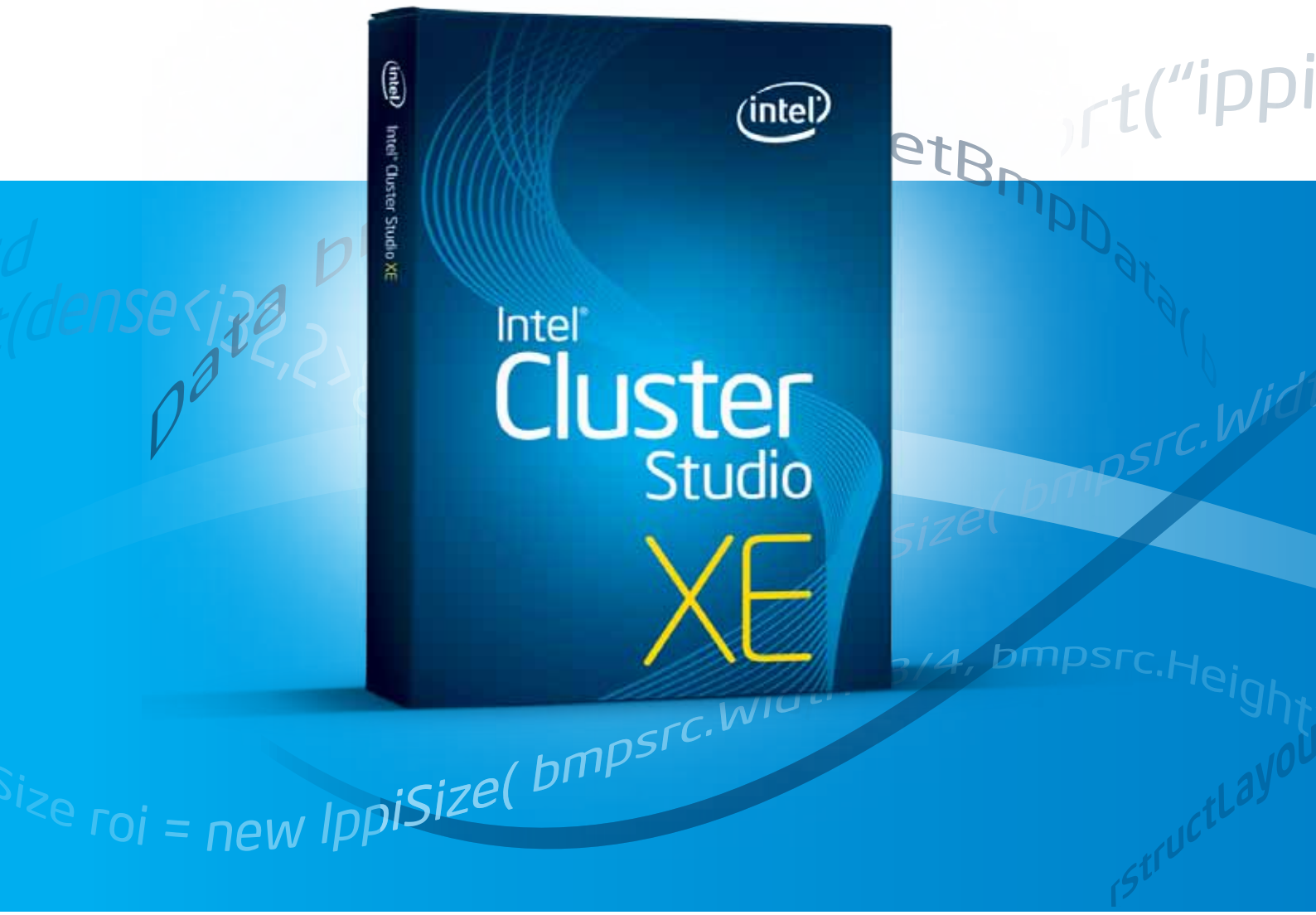
この記事の続きはこちら (英語)

Go-Parallel.com (英語) でご覧になれます。

Go Parallel では、この他にも次のような関連トピックのブログをご覧いただけます : Translating Multicore Power into Application Performance (マルチコアのパワーでアプリケーション・パフォーマンスを引き出そう)。

インテル® Cluster Studio XE の新しい解析ツール

ソフトウェア & サービス
David Mackay, Ph.D.
Krishna Ramkumar



インテル® Cluster Studio XE は、以前のバージョンのインテル® Cluster Studio に含まれていたツール群のアップデート版に加えて、新しい解析ツール（インテル® Inspector XE およびインテル® VTune™ Amplifier XE）を提供します。多くのアプリケーションが分散 / 共有メモリー･ハイブリッド型のプログラミング･モデルを採用する中、ノード内のソフトウェア解析機能は特に有益です。長年にわたってサポートされているノード間の相互作用に加え、ノード内の共有メモリー領域で何が起こっているのかを解析することができます。

インテル® VTune™ Amplifier XE とインテル® Inspector XE は、ソフトウェアのパフォーマンスとシステムの正当性に関する詳細情報を提供します。クラスター全体にわたってこれらのツールを利用し、ハイブリッド並列アプリケーションの正当性とパフォーマンスを解析して、各ノードの動作を効率良く把握できます。

ここでは、インテル® VTune™ Amplifier XE を使ってインテル® MPI ライブラリーを使用するプログラムの簡単なサンプル解析を紹介し、クラスター環境でツールを活用する典型的な例について説明します。

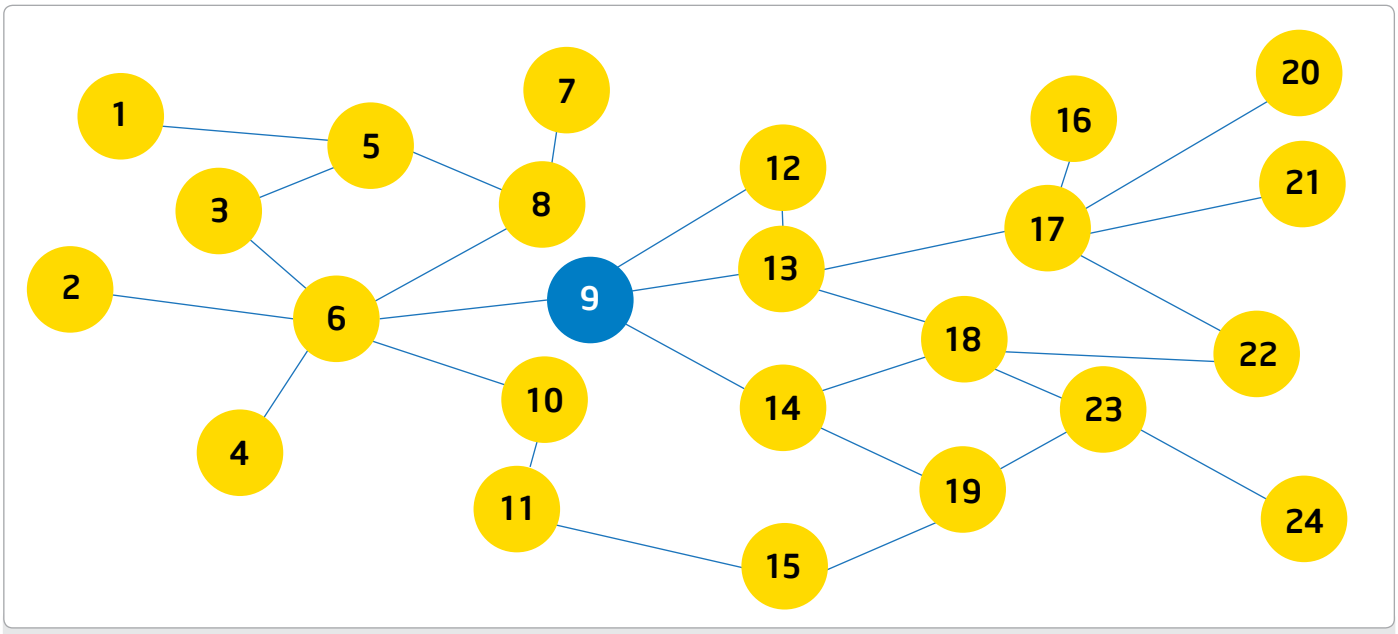


図 1: 抽象化したサンプルグラフ

インテル® Cluster Studio XE

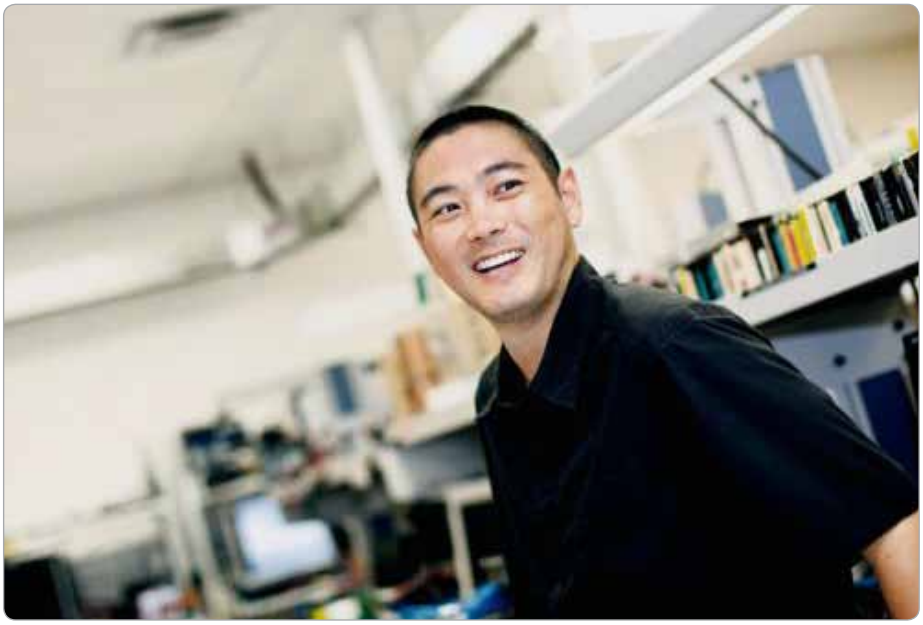
新しいインテル® Cluster Studio XE には、実績のあるインテル® Cluster Studio (インテル® MPI ライブラリー、インテル® コンパイラー、インテル® TBB、インテル® MKL、インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル ® IPP) およびインテル® トレース・アナライザー / コレクターを含む) に加えて、人気の解析ツールであるインテル® Inspector XE とインテル® VTune™ Amplifier XE が含まれました。ここでは、インテル® VTune™ Amplifier XE を使って、分散 / 共有メモリー・ハイブリッド型のクラスター・アプリケーションのパフォーマンスをチューニングする方法を紹介します。インテル® Inspector XE を利用してプログラミング上の問題を検出する方法は、将来の Parallel Universe で取り上げます。

インテル® MPI は長年にわたって、分散メモリー型のコンピューター・クラスターで大規模な問題を解くのに使用されてきました。当初、多くの開発者は 1 つの計算ノードで 1 つの MPI プロセスだけを実行していました。しかし、業界でマルチコアへの移行が進むにつれ、パフォーマンスが重要なアプリケーション、特にコア数に応じてパフォーマンスのスケーリングが求められるアプリケーションで、ハイブリッド並列アプリケーションがますます広く普及しつつあります。このモデルでは、利用可能な計算ノード間で計算処理を分散するため、分散 / 共有メモリー構造を使用して 2 レベルの並列処理を実装します。さらに計算ノードごとにスレッド化し、各計算ノードで利用可能なすべてのコアを活用することができます。これにはいくつかのメリットがあります。1 つ目は、多くのアプリケーションには、各プロセスで計算を行うためにアクセスしなければならないデータが大量にあります。計算クラスターのノードごとのコア数が増えれば、各 MPI プロセスがメモリーのコピーを個別に所有することになります。これは、システムリソースの効果的な使い方とはいえません。ハイブリッド並列アプリケーションはノード上でメモリーを共有します。そして、スレッドはメモリーヒープを共有し、簡単にメモリーデータセットを共有できます。2 つ目は、動的にタスクスチールやノードのタスク・バランシングを行えることです。いったん割り当てたタスクや作業を MPI プロセス間で移動するのは非常に困難なことが知られています。インテル® TBB やインテル® Cilk™ Plus のようなインテルの共有メモリー・プログラミング・モデルは、内部でタスクスチールを行い、自動的に計算ノード内のスレッド間でワークロードを分散します。

ここでは、分散 / 共有メモリー・ハイブリッド型のクラスター・アプリケーションを使って、インテル® VTune™ Amplifier XE の使用法を紹介します。サンプル・アプリケーションは、大規模な実際のデータセットに対して**媒介中心性 (Betweenness Centrality)** を計算するアルゴリズムです。点の媒介中心性は最短パスを基に計算します。これは、(グラフのすべての点のペアの) 最短パスの合計数に対する (グラフのすべての点のペアのうち) その点を通る最短パスの数の比率です。例えば、点の媒介中心性の値が大きい場合は、一般に通信には最短パスが求められるため、その点が通信ネットワークにおいて重要であることを示します。このアルゴリズムは、幅優先走査を使ってそれぞれの点からグラフ上のほかのすべての点への最短パスを計算します (ここでは、重み付けされていない無向グラフのみについて考えます)。疑似コードを含むこのアルゴリズムの詳細は、「Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks」(英語) の Algorithm 1 を参照してください。[1]

図 1 にそれぞれの点にラベルを付けて描画したネットワークのサンプルグラフを示します。この例では、媒介中心性の値が最も大きいのは点 9 です。上記の各点が交通ネットワークの異なる都市や通信ネットワークのルーターを表す場合、この情報が役立ちます。

「インテル® Cluster Studio XE によって、MPI プログラムをより正確かつ容易にチューニングできるようになりました。」



このアルゴリズムを実装するため、最初にインテル® MPI ライブラリーを使って分散並列処理に取りかかりました。次に、インテル® TBB を使って共有メモリー型の並列処理を実装しました。この例では、インテル® TBB の **parallel_for** フレームワークを使ってアルゴリズムのメインループを並列化し、パフォーマンスが向上しています。**parallel_for** 内で、インテル® TBB のスコープロックであるスピン・ミューテックスを使ってデータの一貫性を保証し、データ競合を回避しています（インテル® Inspector XE を利用してマルチスレッド・プログラムのデータ競合を検出する方法は、将来の Parallel Universe で取り上げる予定です）。

インテル® TBB の **spin_mutex** のコードを 図 2 に示します。

```
if(w != node)
{
    // ここでは、この 'if' ブロック内のコードを保護する scoped_lock を使用
    spin_mutex::scoped_lock lock(bwMutex);
    //betweennessCentrality はグローバルなデータ構造体
    betweennessCentrality[w] =
    betweennessCentrality[w] + dependency[w];
}
```

図 2

「インテル® TBB やインテル® Cilk ™ Plus のようなインテルの共有メモリー・プログラミング・モデルは、内部でタスクスチールを行い、自動的に計算ノードのスレッド間でワークロードを分散します。」

「パフォーマンスが重要なアプリケーション、特にコア数に応じてパフォーマンスのスケーリングが求められるアプリケーションで、ハイブリッド並列アプリケーションがますます広く普及しつつあります。」

Sync Object / Function / Call Stack	Wait Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Wait Count	Spin Time
tbb::spin_mutex 0x9187e748	350.712s	85,186	1097.028s
Condition Variable 0x8161ea88	34.858s	23	0s
TBB Scheduler	0.764s	6	0.760s
Socket 0x9a1d6107	0.041s	12	0s
Stream CA-GrQc.txt 0x484fce50	0.001s	2	0s
Signal	0.000s	1	0s

図 4: インテル® VTune ™ Amplifier XE によりレポートされた同期オブジェクトの待機時間を示すスクリーンショット

Sync Object / Function / Call Stack	Wait Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Wait Count	Spin Time
tbb::spin_mutex 0xe492abb1	328.116s	74,396	902.361s
computeBetweennessCentrality	328.080s	72,895	887.346s
ApplyAlgorithm::operator() ← [TBB parallel_for o	328.080s	72,895	887.346s
operator new	0.036s	1,501	15.015s
tbb::parallel_for<tbb::blocked_range<int>, Apply	0.036s	1,501	15.015s
Condition Variable 0x8161ea88	101.582s	23	0s
TBB Scheduler	4.653s	4	4.640s
Socket 0xdeb62f75	0.041s	12	0s
Stream CA-GrQc.txt 0xff615f84	0.001s	3	0s

図 5: トップの同期オブジェクトのソースビューのスクリーンショット

Line	Source	Wait Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Wait Count	Spin Time
236				
237	if(w != node)			
238	{			
239	spin_mutex::scoped_lock lock(betweennessMutex);	328.080s	72,895	887.346s
240	betweennessCentrality[w] = betweennessCentrality[w] + dependency[w];			
241	}			
242	}			
243	}			

図 6: トップの同期オブジェクトを展開したスクリーンショット

 インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice/#opt-jp を参照してください。

このアプリケーションをビルドする際には、-DTBB_USE_THREADING_TOOLS オプションを指定します。これにより、解析ツールはインテル® TBB のすべての構造を認識し、より有用な情報が解析結果に出力されます。このアプリケーションのパフォーマンス解析には、インテル® VTune™ Amplifier XE を使用します。単純な“hotspots 解析”により、多くの計算時間が費やされているルーチンが判明します。“ロックと待機の解析”では、共有リソースを待機するためにスレッドがストールしている場所が分かります。ここでは、ロックと待機の解析を行います。図 3 のコマンドを実行して、インテル® VTune™ Amplifier XE を起動します。

```
mpirun -f mpd.hosts -perhost 1 -n 1 amplxe-
cl -collect locksandwaits -r gmetrics_
lw_r1 -- ./graph_metrics CA-GrQc.txt
```

図 3

このモードでは、インテル® VTune™ Amplifier XE はクラスターの計算ノードごとに結果データを生成し、どのノードの結果か区別できるように結果データにノードのランク番号を追加します。スクリーンショットでは、ノード 0 のデータを示しています。ノードのランクは MPI ランタイムにより設定され、インテル® VTune™ Amplifier XE はこの値を環境変数 PMI_RANK または PMI_ID（ともにインテル® MPI により自動で設定される）から取得します。ここでは、ワークロードとして 5,242 の頂点と 28,980 の辺を使って解析を行いました。図 4 は、インテル® VTune™ Amplifier XE で解析結果を表示したスクリーンショットです。

ここでは、インテル® TBB の spin_mutex で長時間スピンしていることが分かります。**tbb::spin_mutex** の横の + 記号をクリックすると図 5 のように展開されます。

図 5 から、**computeBetweennessCentrality** ルーチンの周囲にある**インテル® TBB のスピン・ミューテックス**がボトルネックであることが分かります。これにより、どこに注目すべきかが明らかです。この行をダブルクリックするか、右クリックして [Source View（ソースビュー）] を選択すると、図 6 のようにソースが表示されます。

図 6 から、スレッドの競合が発生している場所がピンポイントで分かります。これは、スレッドの競合を最小限に抑える、より良いソリューションが必要であることを示しています。我々は問題の処理がグローバル変数の更新であり、この場合はインテル® TBB のアトミック変数のほうが適していることに気付きました。そこでインテル® TBB の spin_mutex を削除し、spin_mutex を使用しなくてもスレッドの安全性を確保できるインテル® TBB のアトミック変数を作成しました（図 7）。

```
map<int, atomic<double> >
betweennessCentrality;
```

図 7

BLOG highlights

インテル® Parallel Studio XE が名誉ある賞を受賞

ソフトウェア開発製品部門ディレクター
JAMES REINDERS

インテル® Parallel Studio XE が 2011 年の HPCwire Readers Choice Awards の「最高の HPC ソフトウェア製品 / 技術」に選ばれました。この賞は、HPCwire が毎年開催しているもので、ハイパフォーマンス・コンピューティング（HPC）分野で広く認められた名誉ある賞です。ワシントン州シアトルで開催された 2011 Supercomputing Conference で受賞者が発表され、授賞式が執り行われました。HPCwire では、「この賞は、HPC 分野での功績を認める、誰もが望む名誉ある賞である」としています。

インテル® Parallel Studio XE 2011 は、Windows* および Linux* 上で信頼性に優れたセキュアなコードを迅速に記述できるように、強化された最適化コンパイラー、ライブラリー、エラーチェッカー、パフォーマンス・アナライザーを 1 つにまとめた統合スイートです。

今回、世界中の非常に多くの開発者に使用されているインテル® Parallel Studio XE がこのような賞を受賞したことを大変嬉しく思います。インテルでは、最近クラスター・コンピューター（その特徴の 1 つは MPI を使用すること）の開発者向けにインテル® Parallel Studio XE の拡張バージョンであるインテル® Cluster Studio XE の提供を開始しました ....

この記事の続きはこちら（英語）

Go-Parallel.com（英語）でご覧になれます。

Go Parallel では、この他にも次のような関連トピックのブログをご覧いただけます :Translating Multicore Power into Application Performance（マルチコアのパワーでアプリケーション・パフォーマンスを引き出そう）。

この変更の適用後、16 ノードの Linux* クラスターで 2 つのアルゴリズム (スコープロックのスピン・ミューテックスを使用したものとアトミック変数を使用したもの)を比較してみました。各ノードは、2 ソケットでソケットごとに 6 物理コアを持つハイパースレッディング対応のインテル® Xeon® プロセッサー x5680 3.33 GHz を搭載した 64 ビット・マシンです (合計 384 のインテル® TBB スレッド)。解析には、36,692 の頂点と 367,662 の辺から構成されるデータセットを使用しました。プログラムの実行時間は、spin_mutex を使用した場合は 284 秒で、アトミック変数を使用した場合は 222 秒に減りました。**パフォーマンスが 22% も向上したのです。**

パフォーマンス解析ツールであるインテル® VTune™ Amplifier XE は、ハイブリッド並列アプリケーションのパフォーマンス解析に役立ちます。一般に、ソフトウェアのチューニングはシングルノード上で行うことを推奨します。チューニングの手順は次のとおりです。

1. インテル® SSE4 やインテル® AVX のような SIMD 命令を使用してソフトウェアがベクトル化されていることを確認します。
2. MPI ライブラリーを使ってメッセージパッシングを追加し、インテル® トレース・アナライザー / コレクターを使用してパフォーマンスを向上します。
3. 各ノードのパフォーマンスが向上するように共有メモリーの並列処理を実装します。
4. インテル® Inspector XE を使用して決定性のない結果を排除し、インテル® VTune™ Amplifier XE を使用してパフォーマンス・ボトルネックを見つけます。

まとめ

インテル® VTune™ Amplifier XE を使用して分散プログラムの各ノードのアクティビティーの詳細を解析することで、ハイブリッド・プログラミングに必要な重要な情報が得られます。インテル® VTune™ Amplifier XE は、使用されている共有メモリー・プログラミング・モデル — インテル® TBB、インテル® Cilk™ Plus、OpenMP*、Co-Array Fortran、pthread*、Windows* スレッド — に関係なく、ハイブリッド・アプリケーションを支援します。インテル® Cluster Studio XE の新しいツールにより、MPI プログラムをより正確かつ容易にチューニングできます。

参考資料

1. D.A.Bader, K. Madduri, “Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks,” The 35th International Conference on Parallel Processing (ICPP 2006), Columbus, OH, August 14–18, 2006.
2. Stanford Large Network Dataset Collection - http://snap.stanford.edu/data/index.html
3. http://software.intel.com/en-us/articles/intel-cluster-studio/
4. http://software.intel.com/en-us/articles/intel-parallel-studio-xe/

「インテル® VTune ™ Amplifier XE は、システム上のソフトウェアのパフォーマンスに関する詳細な情報を提供します。」

最適化に関する注意事項

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項の適用対象である特定の命令セットの詳細は、該当する製品のユーザー・リファレンス・ガイドを参照してください。

改訂 #20110804