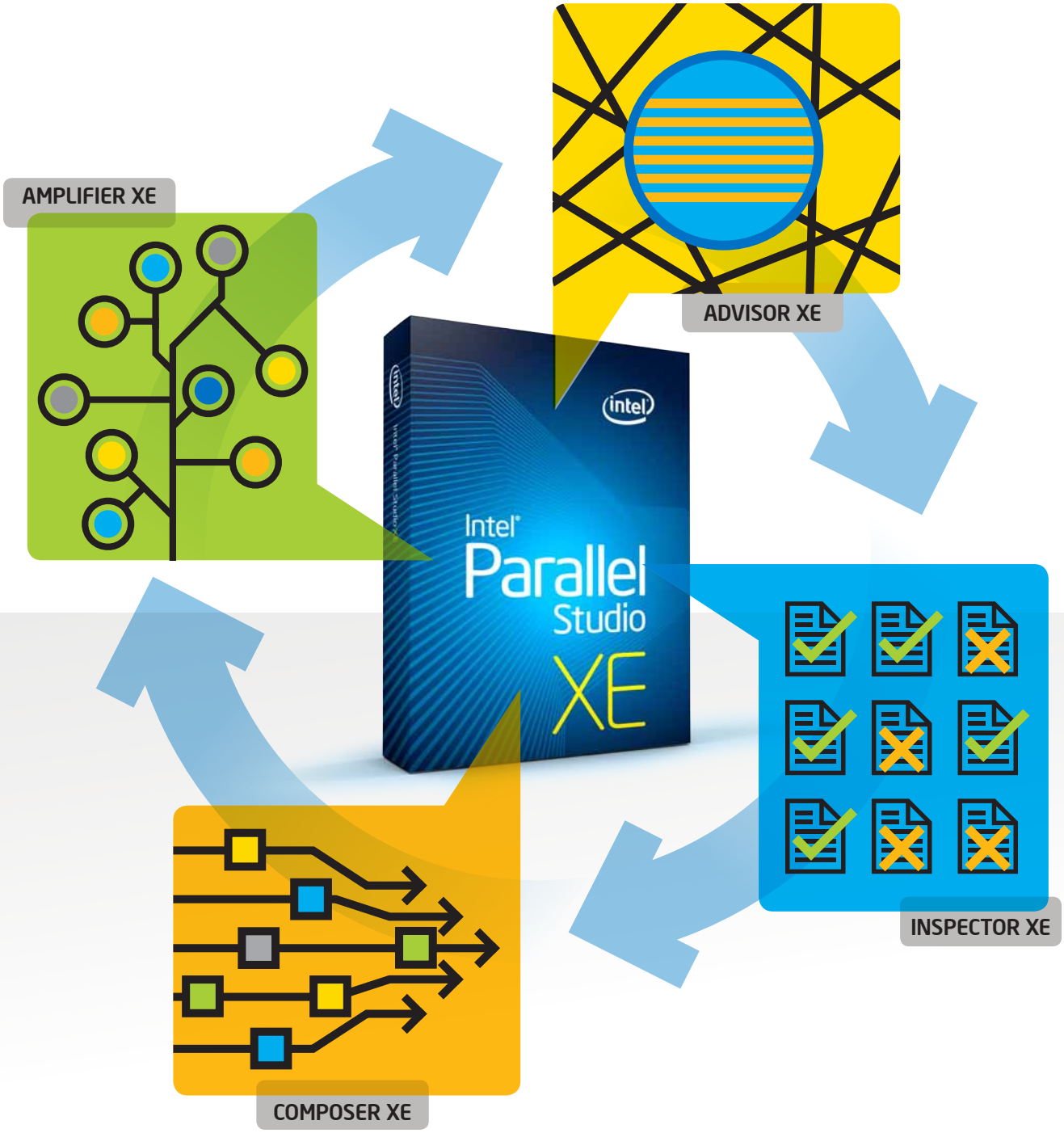
The Parallel Universe

intel Software



Issue 15 | 2013 年 8 月

目次

編集者からのメッセージ 3
結果の重要性

James Reinders

注目記事

効率良いソフトウェア開発： 4
インテル® Parallel Studio XE 2013 SP1 の新機能

Kirill Rogozhin

効率良いコプロセッサー向けプログラミング、強力な並列フレームワークの作成、最新のハードウェアにおける複雑なパフォーマンス問題の発見および解決を支援する新しい機能を紹介します。

インテル® Parallel Studio XE のコプロセッサー・デバッグのサポート 18

Keven Boell

インテル® Xeon Phi™ コプロセッサーで全体または一部を実行するネイティブ・アプリケーションとオフロードプログラム用のデバッグ・ソリューションを検証します。インテル® Parallel Studio XE のツールは、コマンドライン、Eclipse* および Visual Studio* で、プログラムの制御フローと変数に関する単一ソースレベル・ビューを提供します。

フルスケーリングに向けて： WRF (Weather Research and Forecast) 29
モデルとインテル® Cluster Studio XE 2013

Mark Lubin、Scott McMillan、Christopher G. Kruse、Davide Del Vento、Raffaele Montuoro

WRF モデルは、広範な気象アプリケーションで広く利用されている次世代のメソスケール気象予報システムです。この記事では、インテル® コンパイラーおよびインテル® MPI ライブラリーを含むインテル® Cluster Studio XE 2013 により達成した、“汎用” スーパーコンピューターにおける WRF モデルのスケーラビリティーについて説明します。

編集者からのメッセージ

James Reinders
インテル コーポレーションの並列プログラミング・エバンジェリスト兼ディレクター

Morgan Kaufmann から出版された 2 冊の新しい書籍、『Intel® Xeon Phi™ Coprocessor High Performance Programming』(2013、日本語版出版予定) および『Structured Parallel Programming』(2012、日本語 :『構造化並列プログラミング』) の共著者。このほかに、『Intel® Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』(O' Reilly Media、2007、日本語 :『インテル スレッディング・ビルディング・ブロック ―マルチコア時代の C++ 並列プログラミング』、中国語、韓国語の翻訳版があります) や『VTune™ Performance Analyzer Essentials』(In-tel Press, 2005) などの文献を発表しています。

結果の重要性

現状よりも優れたソフトウェアを提供するには、絶えずソフトウェアを改良し続ける必要があります。我々がイノベーターとして作成するソフトウェアは、企業のデータセンターからモバイルデバイスまで、あらゆる環境でより高速に実行し、新しい体験を創造し、ユーザーとサービスを繋ぎ、個人データや業務データを安全に守るために非常に重要なものです。この号では、より高い目標を目指すソフトウェア・イノベーターを支援する新しいツールと機能を紹介します。

「効率良いソフトウェア開発 : インテル® Parallel Studio XE 2013 SP1 の新機能」では、インテル® Parallel Studio XE の最新リリースの拡張機能について説明します。これらの新機能は、スケーラビリティーの予測からメモリーリークの検出、CPU と GPU 解析まで、幅広くカバーします。新機能に共通している点は、プログラミング効率の強化と製品品質の向上です。

開発を迅速に行えることは、デバッグをより迅速かつ正確に行えるということです。「インテル® Parallel Studio XE のコプロセッサー・デバッグのサポート」では、インテル® Xeon Phi™ コプロセッサー上で全体または一部を実行するネイティブ・アプリケーションとオフロードプログラムについて、Linux* と Windows* におけるデバッグのシナリオを解説します。

「フルスケーリングに向けて : WRF (Weather Research and Forecast) モデルとインテル® Cluster Studio XE 2013」では、スケーラビリティーは次世代のメソスケール気象予報研究における重要な要素であることを述べています。エンジニアと研究者の共同チームは、ハリケーン カトリーナのような災害対策の課題に対して、高度なスーパーコンピューティング・シミュレーションをテストしました。

常に増大するパフォーマンスの期待に応える、モノのインターネット (Internet of Things) をつなぐ、ビジョンを実現するなど、目標は違っていても、さまざまなツールを活用することで開発プロセスをサポートし、洞察を深め、アプリケーションで量的にも質的にも優れた結果を得ることができるでしょう。

James Reinders
2013 年 9 月

効率良いソフトウェア開発：インテル® Parallel Studio XE 2013 SP1 の新機能

Kirill Rogozhin
インテル コーポレーション
開発者製品部門 テクニカル・コンピューティング、アナライザー、ランタイム テクニカル・コンサルティング・エンジニア

インテル® Parallel Studio XE は、プロのソフトウェア開発者の間ではよく知られているツール・スイートです。マルチコアシステムで動作する高度に最適化され安定したアプリケーションを作成し、将来のアーキテクチャーにもスケーリングできるように支援します。このスイートの最新バージョンである、**インテル® Parallel Studio XE 2013 サービスパック 1** (SP1) には、多くの優れた新機能と最適化が含まれています。たとえば、インテル® Composer XE は、OpenMP* 4.0 (**http://openmp.org**) 規格の主な機能をサポートしました。スレッド・プロトタイピング・ツールであるインテル® Advisor XE には、さまざまなコア数におけるスケーラビリティーを予測する機能が追加され、ソフトウェア・アーキテクトがインテル® Xeon Phi™ システムのスケーラビリティーを予測できるようになりました。メモリー / スレッドデバッガーであるインテル® Inspector XE には、Rational Purify* および Valgrind* の除外ファイルをインポートする機能があり、ほかの検証ツールから移行することができます。また、解析が完了

してアプリケーションが終了する前にメモリーリークを検出できるようになりました。パフォーマンス・プロファイラーであるインテル® VTune™ Amplifier XE は、オーバーヘッド時間をより詳細に調査できるようになり、コマンドライン・インターフェイス (CLI) でソースへのドリルダウンがサポートされました。さらに、呼び出し元 / 呼び出し先ビューで、より簡単に関数のコールツリーを展開できるようになりました。CPU に加えて、GPU (インテル® グラフィックス・テクノロジー) も解析することができます。開発者は、インテル® VTune™ Amplifier XE の拡張された全般解析を利用して、第 4 世代インテル® Core™ プロセッサーなど最新のハードウェアの解析を行えます。

この記事では、インテル® Parallel Studio XE 2013 SP1 に追加された新機能を、スイートのコンポーネントごとに、インテル® Composer XE、インテル® Advisor XE、インテル® Inspector XE、インテル® VTune™ Amplifier XE の順に説明します。(インテル® Parallel Studio XE 2013 の基本的な機能についての詳細は、**http://software.intel.com/en-us/intel-parallel-studio-xe/** (英語) を参照してください)。

インテル® Composer XE

インテル® Composer XE には、C++ および Fortran コンパイラー、パフォーマンス・ライブラリー、並列プログラミング・モデル、Debugger Extension (Linux*) が含まれています。インテル® Composer XE 2013 SP1 では、ベクトル化の強化、コプロセッサーおよびアクセラレーター向けのコンパイル、**OpenMP* 4.0** で追加された主な機能のサポートなど、多くの改良が行われました。

OpenMP* SIMD 構文

SIMD (Single Instruction, Multiple Data) 命令 (ベクトル命令やパックド命令と呼ばれることもあります) を使用することは、データ並列を実装する効率良い方法の 1 つです。インテル® コンパイラーはコードを自動的にベクトル化するために最善を尽くしますが、プログラミング・スタイルによってはベクトル化が困難な場合があります。このため、

この記事では、インテル® Parallel Studio XE 2013 SP1 に追加された新機能を、スイートのコンポーネントごとに、インテル® Composer XE、インテル® Advisor XE、インテル® Inspector XE、インテル® VTune™ Amplifier XE の順に説明します。

1

```
double pi()
{
    double pi = 0.0;
    double t;
#pragma omp simd private(t) reduction(+:pi)
    for (i=0; i<count; i++) {
        t = (double)((i+0.5)/count);
        pi += 4.0/(1.0+t*t);
    }
    pi /= count
    return pi;
}
```

2

```
#pragma omp declare simd notinbranch
float min(float a, float b) {
    return a < b ? a : b;
}

#pragma omp declare simd notinbrach
float distsq(float x, float y) {
    return (x - y) * (x - y);
}
```

3

```
#pragma omp parallel for simd
    for (i=0; i<N; i++)
        d[i] = min(distsq(a[i], b[i]), c[i]);
```

インテル® コンパイラーでは、ユーザー定義または SIMD のベクトル化は “#pragma omp simd” を使用して実装されます。OpenMP* 4.0 では、C/C++ および Fortran に移植可能な標準化されたコードでこれらの SIMD 機能を利用できます。“omp simd” 構文が “for loop” の前で宣言されると、コンパイラーは複数の反復が SIMD 命令で同時に実行されるようにループのベクトルコードを生成します。“reduction” などの OpenMP* の節を追加することもできます。**(図 1)**

開発者は、SIMD (“要素”) 関数を定義して、各要素で実行する操作を指定することができます。この定義に基づいて、コンパイラーは関数のベクトルコードを生成し、関数を SIMD ループから利用できるようにします。SIMD 関数は “omp declare” 構文で定義します。**(図 2)**

SIMD レベルの並列性 (1 つのコア内) と高レベルのスレッド並列性 (複数のコア間) を組み合わせることも可能です。次のループは最初にベクトル化され、反復はスレッド間で分散されます。**(図 3)**

4

```
#pragma omp target map(to(b:count)) map(to(c,d)) map(from(a:count))
{
  #pragma omp parallel for
    for (i=0; i<count; i++)
      a[i] = b[i] * c + d;
}
```

これらのツールを利用することで、より効率的にコプロセッサー向けのプログラミング、強力な並列フレームワークでの作業、最新のハードウェアにおける複雑なパフォーマンス問題の発見および解決を行えます。

コプロセッサーの活用

インテル® Xeon Phi™ コプロセッサーのようなアクセラレーターやコプロセッサーはより一般的になりつつあります。OpenMP* 4.0 では、“omp target” 構文により計算 (理想的には高度に並列化された計算) をオフロードできます。構文の後のコードの基本ブロックはアクセラレーター / コプロセッサーで実行されます。ターゲットデバイスが存在しなかったり、実装でサポートされていない場合、ターゲット領域はホストデバイス (CPU) 上で実行されます。例のように、“map” 節を用いてデータ環境を制御できます (**図 4** を参照)。

インテル® Advisor XE

インテル® Advisor XE は、シリアルコードの並列実行をモデル化できるため、実際に並列命令をコードに追加する前に並列実装のプロトタイプを作成できます。ソフトウェア・アーキテクトは、並列実装に多くの労力をかける前にさまざまなスレッド設計を検証し、柔軟に設計を調整することができます。また、コードを並列化した後に発生するデータ競合を予測し、できるだけ早く競合を解決することが可能です。

スケーラビリティー解析の拡張

スケーラビリティーの予測は従来は 32 コアに制限されていました。この制限は CPU スレッドには十分ですが、インテル® Xeon Phi™ コプロセッサーに備わる多くのコアを予測するには不十分でした。このため、新しいインテル® Advisor XE では、任意のコア数のスケーラビリティーを予測できるようになりました。**図 5** は 512 コアの予測を示していますが、さらに多くのコアの予測も可能です。開発者は、インテル® Xeon Phi™ コプロセッサーで実行した場合に適切にスケーリングされるか、アルゴリズムの理論的なピークを迅速に予測できます。

結果のスナップショット

インテル® Inspector XE およびインテル® VTune™ Amplifier XE を使用すると、複数のテストを実行してすべての結果を保存し、パフォーマンスの違いを比較して問題の状態を追跡することができます。インテル® Advisor XE の以前のバージョンでは、インテル® Advisor XE のプロファイルの複雑な性質により、最新の結果のみが保存されていました。

インテル® Parallel Studio XE 2013 SP1 の最新リリースでは、インテル® Advisor XE の結果のスナップショットを作成できます。このコピーは読み取り専用で、開発者は以前の予測を参照することができます。プロジェクト、アルゴリズム、その他を変更した後、古いアプローチと現在のバージョンのパフォーマンスやスケーラビリティーを常に比較できます (**図 6** を参照)。

アノテーションの停止と再開

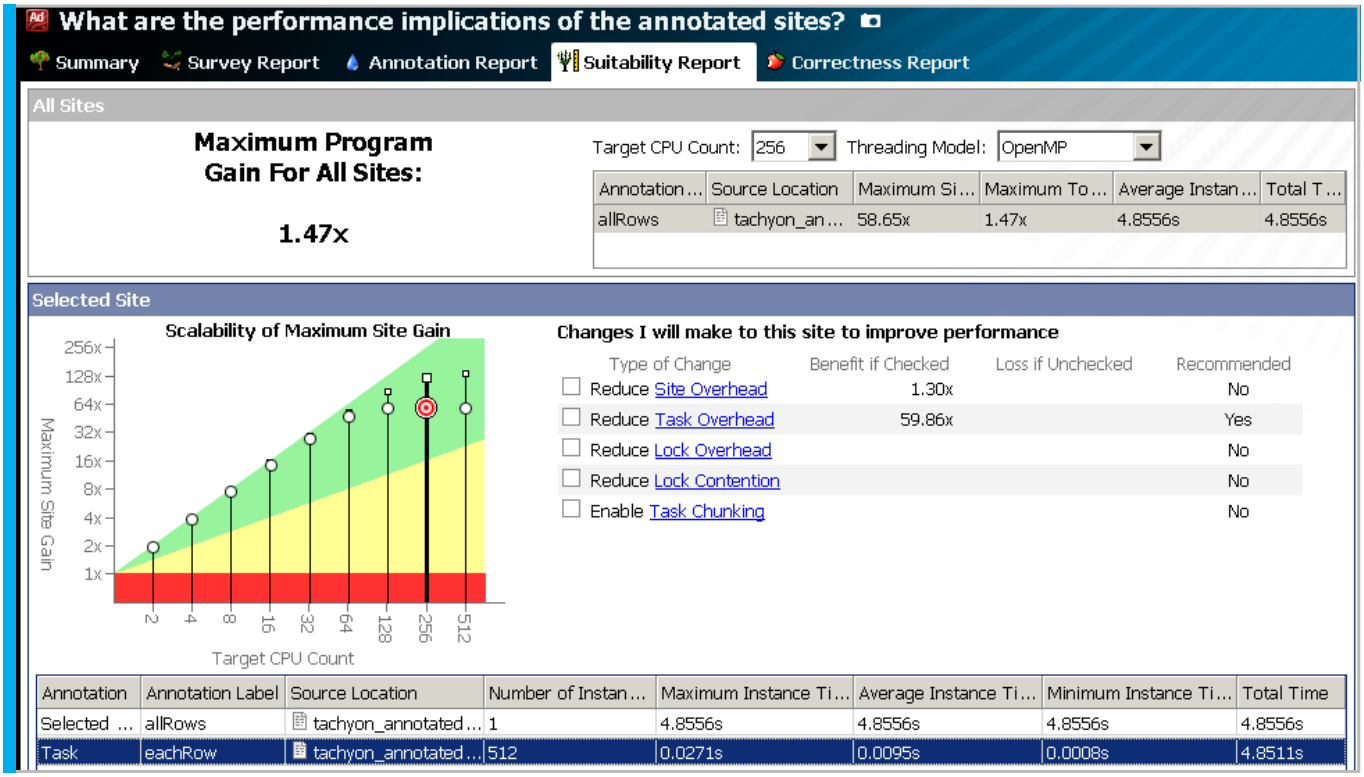
インテル® Advisor XE は、並列実行の複雑なモデル化を実行し、モデルのスレッドエラーを検出するため、著しいオーバーヘッドが発生します。大規模なアプリケーションのコレクション時間を短縮するために、ユーザーは特定のコード領域の解析を省略することができます。ツールにその領域の解析をスキップさせることで、時間を節約できます。また、特定のコード領域のみを解析した場合、大量の無関係のデータを処理せずに済むため、精度が向上します。特定のコード領域を解析するには、新しいアノテーション・タイプ (ANNOTATE_DISABLE_COLLECTION_PUSH および ANNOTATE_DISABLE_COLLECTION_POP) を指定します (グラフィカル・ユーザー・インターフェイス (GUI) ボタンも利用できます) (**図 7** を参照)。

インテル® Inspector XE

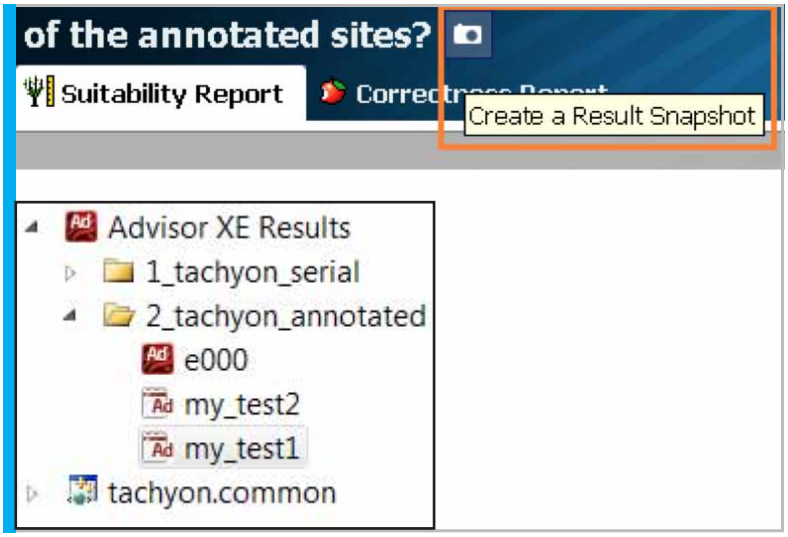
インテル® Inspector XE は、実行時にダイナミック解析を行うメモリーおよびスレッドデバッガーで、標準デバッガーに統合されます。メモリーリーク、初期化されていないメモリーアクセスや無効なメモリーアクセス、その他のメモリー問題を検出するメモリーデバッガーと、デッドロック、データ競合、その他のスレッド問題を検出するスレッドデバッガーが含まれています。これらの問題は、従来のリグレッション・テストやスタティック解析では検出できないことがあります。

Valgrind* および Rational Purify* の除外ファイルのインポート

インテル® Inspector XE は、あらかじめ選択した関連のない問題が解析中に検出されたときに表示しないように設定できます。ほかのベンダーの同様のツールにも同じ機能が用意されています。誤検出レポートや、修正できずレポートから除去すべきサードパーティーのモジュールの問題など、問題を表示しない理由はさまざまです。大規模なソフトウェア・プロジェクトのそのような除外ルールは、個別のファイルに格納され、除外す



5 512 コアまでのスケーラビリティーの予測



6 インテル® Advisor XE の結果のスナップショット

るエラーの定義 (コードの位置、モジュール、問題タイプ、その他の情報) が含まれます。

インテル® Inspector XE の最新バージョンでは、Valgrind* や Rational Purify* などのほかのツールから除外ファイルをインポートできるようになりました。これらのファイルはインテル® Inspector XE の除外ファイル形式に変換されます。この機能により、ほかの正当性チェックツールの除外ファイルを、インテル® Inspector XE へ簡単に移行することができます。除外ルールを継承することで、時間を節約し、過去に除外された問題の調査に対する投資を無駄にしません。

インテル® Inspector XE の最新バージョンでは、任意のテキストエディターで編集可能な、新しいテキスト形式で除外ファイルを保存します。

プログラム終了前にメモリーリークを検出

メモリーリークの検出は、インテル® Inspector XE のメモリー解析の主要な機能の 1 つです。以前のバージョンでメモリーリークを検出するには、プログラムで行われるヒープ割り当てと割り当て解除をすべてキャッチできるように、プログラムを開始から終了まで実行する必要がありました。ターゲット・アプリケーションの実行時間が長い場合や終了しない場合 (daemon など)、また終了前にクラッシュする場合は、インテル®

BLOG HIGHLIGHTS

AVX-512 命令

JAMES REINDERS »

最新の「Intel® Architecture Instruction Set Extensions Programming Reference (インテル® アーキテクチャー命令セット拡張プログラミング・リファレンス)」には、インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) 命令の定義が掲載されています。これらの命令は、SIMD サポートが 512 ビット幅に拡張されています。プログラムは、512 ビットのベクトルに、8 つの倍精度浮動小数点数、16 の単精度浮動小数点数、8 つの 64 ビット整数、または 16 の 32 ビット整数をパックすることができます。つまり、1 つの命令で AVX/AVX2 が処理できるデータ要素数の 2 倍 (SSE の 4 倍) の処理を行えます。

インテル® AVX-512 命令は、計算集約型のタスクでより高いパフォーマンスを発揮します。また、設計段階でかつてないレベルの豊富な機能を追加することにより、最も高度なコンパイラー・オプションを提供しています。インテル® AVX-512 には、512 ビット幅の 32 のベクトルレジスター、8 つの専用マスクレジスター、パックド浮動小数点データまたはパックド整数データの 512 ビット演算、組込み丸め制御 (オーバーライド・グローバル設定)、組込みブロードキャスト、組込み浮動小数点エラー抑制、組込みメモリーエラー抑制、新しい操作、ギャザー / スキャッターの追加サポート、高速算術命令、大きな変位値のコンパクト表現、その他のオプション機能が含まれています。32 の ZMM レジスターが 2000 のレジスター空間を表すことに注目するのも良いでしょう。

この記事の続きはこちら (英語) でご覧になれます。 >

Inspector XE でメモリーリークを解析することは不可能でした。インテル® Inspector XE の最新バージョンでは、これらの問題を解決する、“オンデマンド” のリークレポート機能が追加され、メモリーリークを調査する領域を定義できるようになりました。領域の定義は、GUI コントロールから、コマンドラインから、あるいはソースコードから API を呼び出して行います。この機能により、リークがないか特定のコードセグメントを確認できます。エラーは実行中に報告されるため、アプリケーションが終了するまで待つ必要はありません。

インテル® VTune™ Amplifier XE

インテル® VTune™ Amplifier XE は、パフォーマンス・プロファイラーです。スレッド、モジュール、関数、命令、その他の範囲を選択してアプリケーションで最も CPU 時間が費やされている部分を表示し、スレッド間のワークバランス、待機時間、待機を引き起こす同期プリミティブに関する詳細を提供します。開発者は、キャッシュミス、フォルス・シェアリングなど、マイクロアーキテクチャーのパフォーマンス・ボトルネックを丹念に調べられます。

オーバーヘッド時間の詳細なレポート

マルチスレッド・プログラムでは、一定の CPU 時間がスレッド同期、計算タスクの管理などに費やされます。“メインの” 計算に費やされない時間はオーバーヘッドと考えられます。優れたパフォーマンスを達成するには、オーバーヘッドを最小限に抑えなければなりません。インテル® VTune™ Amplifier XE の最新バージョンでは、オーバーヘッドとスピン時間 (ビジーウェイト) に関する詳細な情報が示され、スピンウェイトや特定の関数、モジュール、命令、その他のオーバーヘッドに CPU 時間のどの部分が費やされたかを確認できます。また、OpenMP*、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) あるいはインテル® Cilk™ Plus 並列フレームワークが原因のオーバーヘッドを検出することができます。開発者は、グリッドに表示される対象オブジェクトの時間を調べるか、タイムラインを参照して、オーバーヘッドとスピン時間がスレッド間でどのように分散されているかを確認します (**図 8** を参照)。

7

```
int main(int argc, char* argv[])
{
    ANNOTATE_DISABLE_COLLECTION_PUSH
    // 初期化を実行
    ANNOTATE_DISABLE_COLLECTION_POP

    // 処理を実行

    ANNOTATE_DISABLE_COLLECTION_PUSH
    // 終了処理を実行
    ANNOTATE_DISABLE_COLLECTION_POP

    return 0;
}
```

OpenMP* アプリケーション解析の拡張

インテル® VTune™ Amplifier XE の最新バージョンでは、OpenMP* 並列領域を解析する機能が拡張されました。ボトムアップ・ペインでフレームドメイン別にグループ化すると、OpenMP* 並列領域がフレームとして表示されます (**図 9** を参照)。フィルタリングを行い、特定の OpenMP* 並列領域にパフォーマンス・プロファイルを絞り込んで、費やされた時間、過去のパフォーマンス問題、ワークバランスを確認できます。“[No frame domain - Outside any frame]” ノードは、並列領域の外で費やされたシリアル時間を表します。この情報から、アプリケーションのシリアル / 並列のバランスが分かります (アムダールの法則を思い出してください)。

OpenMP* 実行に関連するオーバーヘッドとスピン時間は、インテルの OpenMP* だけでなく、GCC* や Microsoft* OpenMP* ランタイムでも認識できます。**図 8** の “[OpenMP worker]” ノードと “[OpenMP fork]” ノードは、Microsoft* OpenMP* ライブラリーの vcomp100.dll に対応しています。

コマンドライン・ツールでソースとアセンブリーを表示

たとえば、SSH 接続された Linux* サーバーでプロファイリングする場合、コマンドライン・インターフェイスを使用するほうが GUI よりも便利です。既存のレポート機能に加えて、コマンドライン・ツールからソースコードやアセンブリーに簡単にドリルダウンできるようになりました。リモートマシンに接続して結果を取得したり VNC を設定する必要はありません。コレクションに使用した同じシェル上でプロファイリング・データを表示することが可能です (**図 10** を参照)。

呼び出し元 / 呼び出し先ビューでコールツリーを展開

新しい呼び出し元 / 呼び出し先ビューは、ボトムアップ・ビューとトップダウン・ビューの長所を組み合わせたものです。左ペインには各関数の合計時間とセルフ時間、右上ペインには選択した関数の親 (呼び出し元)、右下ペインには選択した関数の子 (呼び出し先) が表示されます (**図 11** を参照)。呼び出しシーケンスをツリーで表示し、各関数の状況を CPU 時間で確認できます。特定の関数を合計時間ベースでフィルターして、その関数も含めたすべてのサブツリーを表示します。この機能を利用することにより、コードで最もタイム・クリティカルなコールパスを特定できます。

GPU プロファリング

インテル® VTune™ Amplifier XE で、インテル® プロセッサー・グラフィックスのコードをプロファイリングできるようになりました。全体的な GPU アクティビティーをモニターすることが可能です (GPU がビデオ・エンコーディングに使用されたか？ どの CPU スレッドが GPU 計算のトリガーになっているか？ すべての GPU リソースが使われているか？ など)。この機能は、OpenCL* カーネルを利用して GPU で実行している計算タスクで役立ちます。インテル® VTune™ Amplifier XE はグリッド形式でカーネルを表示するため、ワークサイ

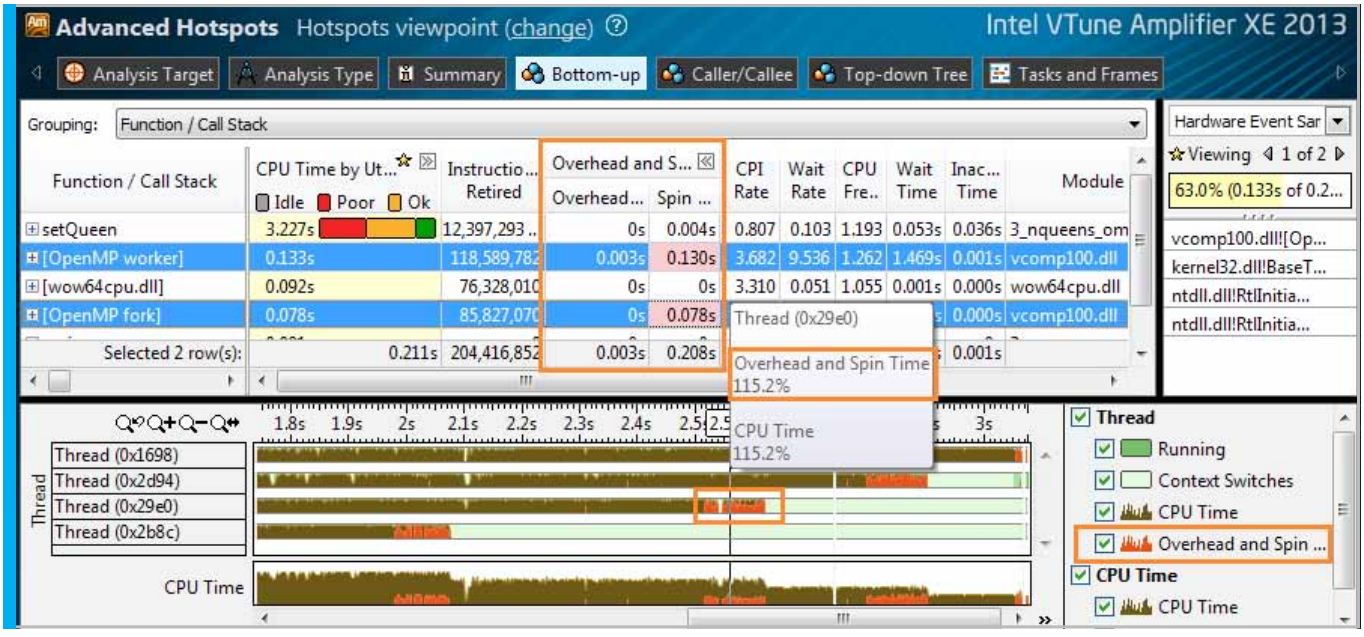
ズと GPU L3 キャッシュミスのようなハードウェア問題を一目で確認できます (**図 12** を参照)。タイムラインで、OpenCL* カーネルはカーネルを起動した CPU スレッド上にマークされます。GPU 実行ユニットの状態 (アクティブ、アイドル、ストール) の比率も示されます。GPU プロファイルにより、ワークロードが CPU と GPU のどちらで制限されているか、CPU で実行している最もホットな OpenCL* カーネルはどれか、より多くの GPU リソースを使用する余裕があるかどうかを把握することができます。

第 4 世代インテル® Core™ プロセッサーのトップダウン・パフォーマンス解析

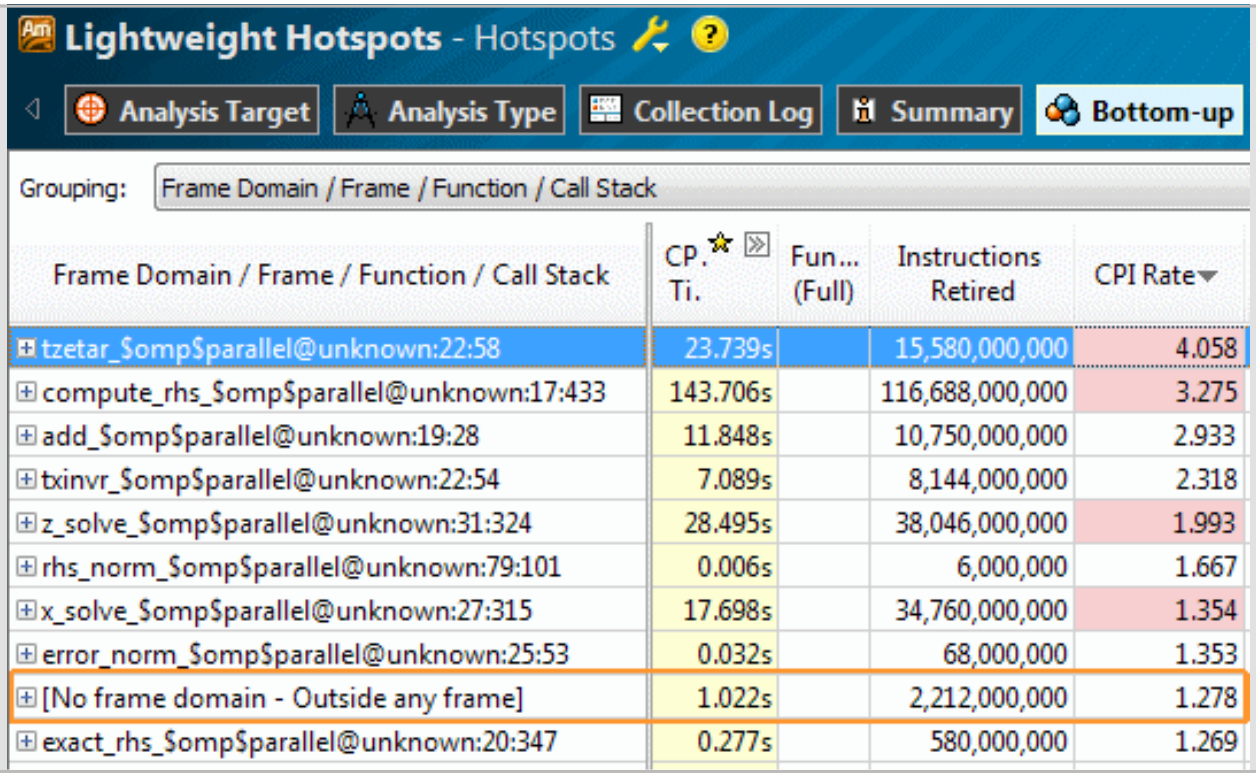
マイクロアーキテクチャーの問題の調査は複雑な作業です。調査には、CPU アーキテクチャーの深い理解と多大なハードウェア・イベントの知識が必要です。分かりやすく、より構造的で深いパフォーマンス解析を実行できるように、全般解析プロファイルにパフォーマンス・データのトップダウン構造が実装されました (**図 13** を参照)。パフォーマンス評価基準は事前に定義され、潜在的な問題がハイライトされます。階層的なデータ表示により、必要な詳細レベルを制御できます。高レベルの問題を詳細なグループと個々の問題に分類するブレークダウン構造であるため、ナビゲーションと理解がより簡単になりました。

まとめ

インテル® Parallel Studio XE 2013 SP1 は、ソフトウェア開発者にさまざまな新機能を提供します。これらのツールを利用することで、より効率的にコプロセッサー向けのプログラミング、強力な並列フレームワークでの作業、最新のハードウェアにおける複雑なパフォーマンス問題の発見および解決を行い、ハイエンドのソフトウェア製品を作成することができます。最新のスイートをダウンロードして、現在作業しているプロジェクトをどのように改善できるかぜひ確認してみてください。



8 インテル® VTune™ Amplifier XE のオーバーヘッドとスピン時間

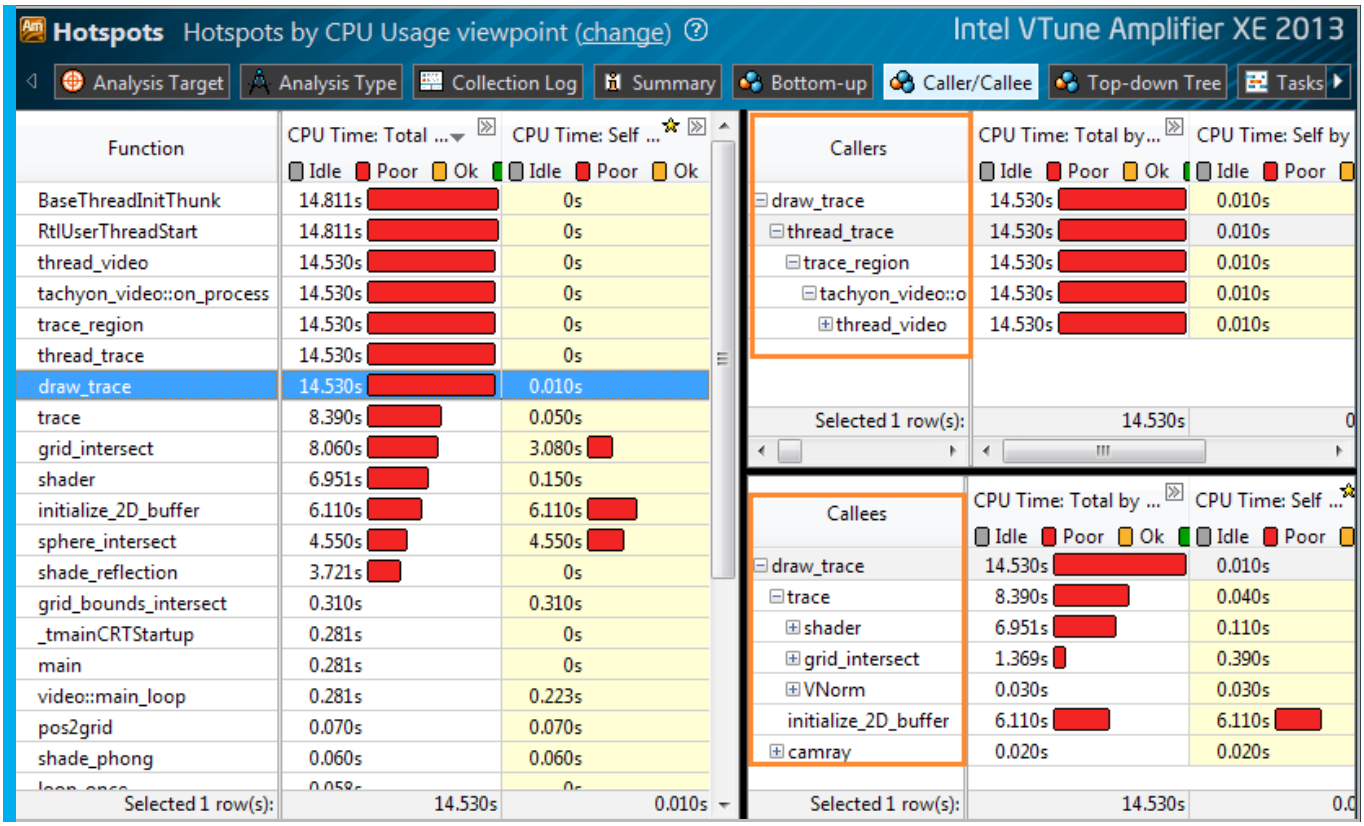


Frame Domain / Frame / Function / Call Stack	CP. Ti.	Fun... (Full)	Instructions Retired	CPI Rate
tzetar_ompparallel@unknown:22:58	23.739s		15,580,000,000	4.058
compute_rhs_ompparallel@unknown:17:433	143.706s		116,688,000,000	3.275
add_ompparallel@unknown:19:28	11.848s		10,750,000,000	2.933
txinvr_ompparallel@unknown:22:54	7.089s		8,144,000,000	2.318
z_solve_ompparallel@unknown:31:324	28.495s		38,046,000,000	1.993
rhs_norm_ompparallel@unknown:79:101	0.006s		6,000,000	1.667
x_solve_ompparallel@unknown:27:315	17.698s		34,760,000,000	1.354
error_norm_ompparallel@unknown:25:53	0.032s		68,000,000	1.353
[No frame domain - Outside any frame]	1.022s		2,212,000,000	1.278
exact_rhs_ompparallel@unknown:20:347	0.277s		580,000,000	1.269

9 OpenMP* 並列領域の解析

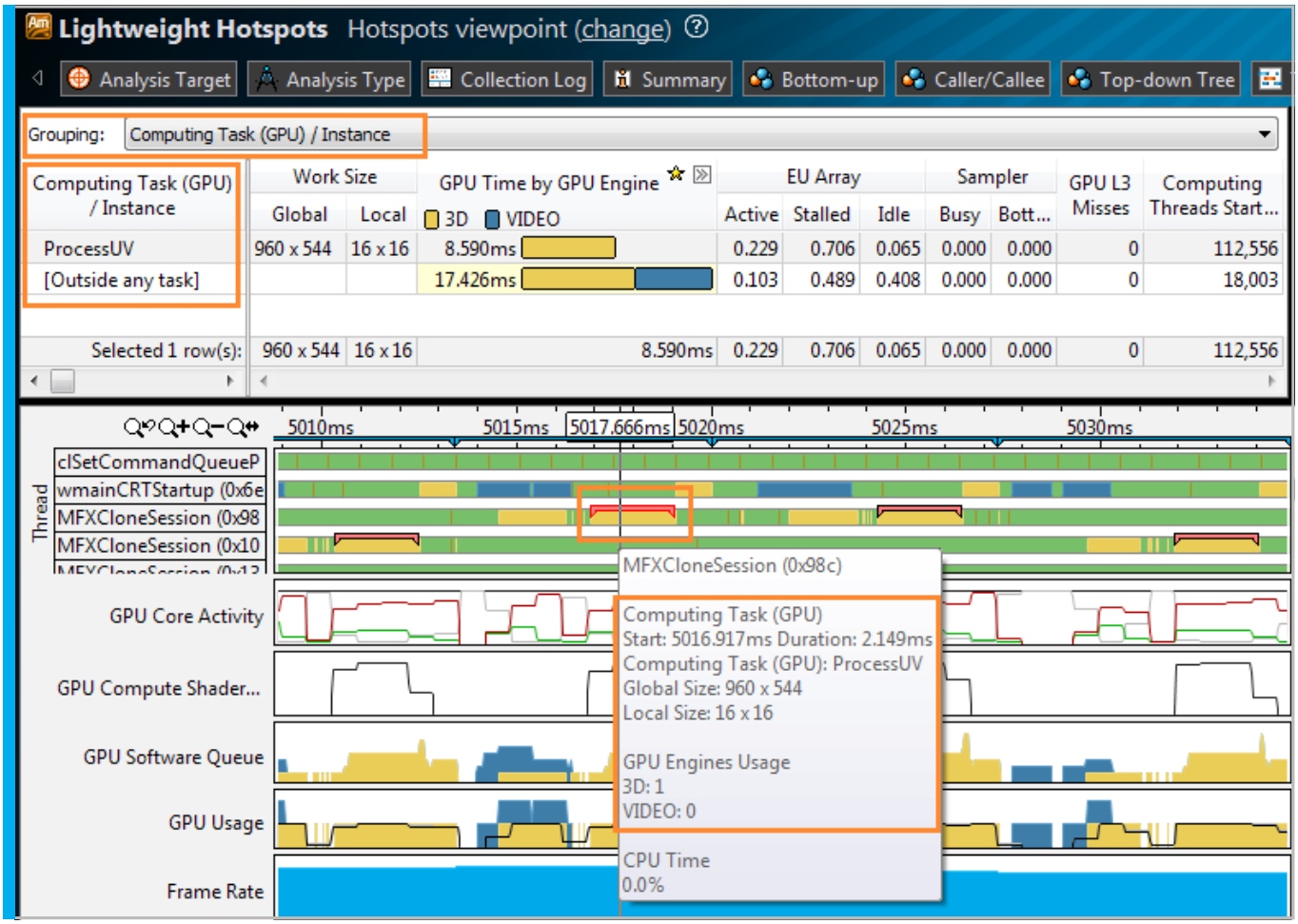
10

```
localhost:/tmp/tachyon # amplxe-cl -report hotspots -source-object function=grid_intersect -r r000hs/
Source Line  Source                                                                  CPU Time:Self
-----------  ------------------------------------------------------------------------  -------------
460            return 1;
461          }
462
463
464          /* the real thing */
465          static void grid_intersect(grid * g, ray * ry)                                  0.036
466          {
467
468
469            flt tnear, tfar, offset;
470            vector curpos, tmax, tdelta, pdeltaX, pdeltaY, pdeltaZ, nXp, nYp, nZp;
471            gridindex curvox, step, out;
472            int voxindex;
473            objectlist * cur;
474
475            if (ry->flags & RT_RAY_FINISHED)
476              return;
477
478            if (!grid_bounds_intersect(g, ry, &tnear, &tfar))
479              return;
480
481            if (ry->maxdist < tnear)                                                      0.020
```

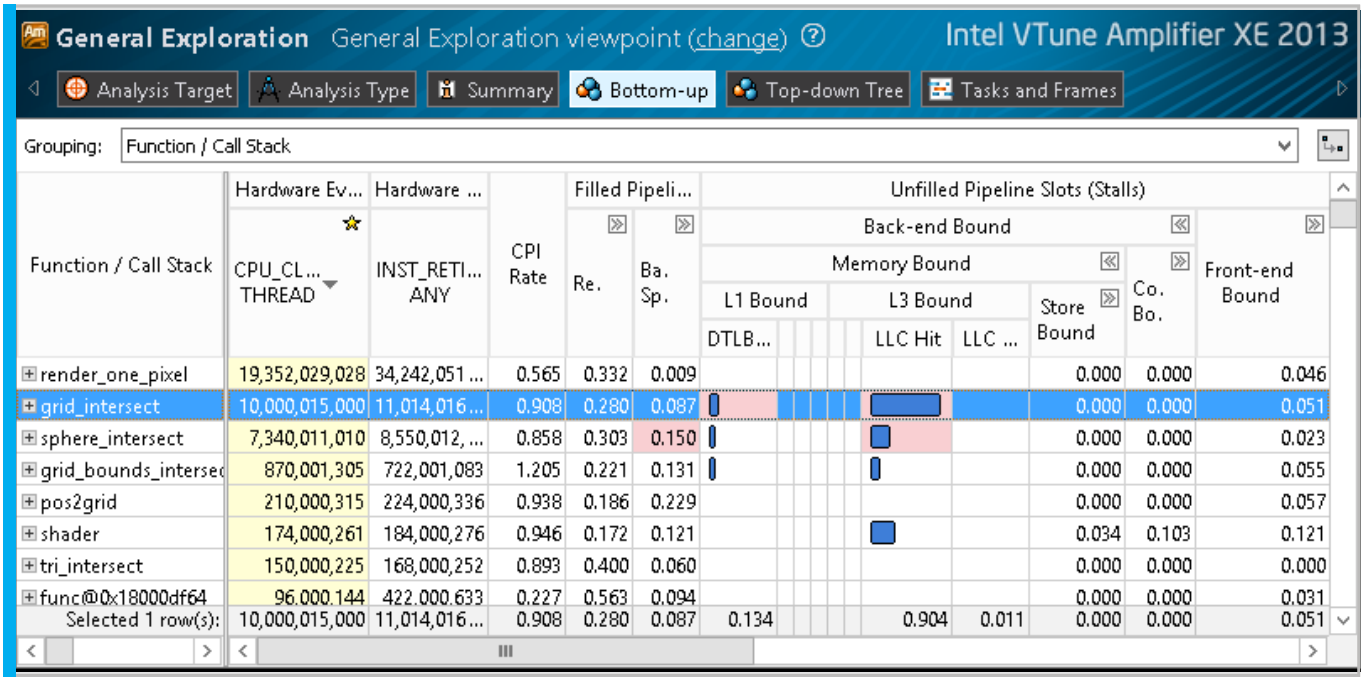


11 インテル® VTune™ Amplifier XE の呼び出し元/呼び出し先ビュー

評価版のダウンロード：
http://software.intel.com/en-us/intel-software-evaluation-center (英語)



12 GPU 計算のプロファイリング



13 第 4 世代インテル® Core™ プロセッサーの全般解析プロファイルのトップダウン構造

インテル® Parallel Studio XE のコプロセッサー・デバッグのサポート

Keven Boell
インテル コーポレーション
ソフトウェア・エンジニア

先日登場したインテル® Xeon Phi™ コプロセッサーは、PCI Express* 経由でホスト・プロセッサーに接続された、新しいタイプのハイブリッド・コンピューティング・システムを形成します。コプロセッサーには、コアごとに 4 つのハードウェア・スレッド、共有コヒーレント・キャッシュ、512 ビット幅の SIMD ベクトルユニットを備えた、最大 61 のコアが搭載されています。インテル® Xeon Phi™ コプロセッサー上で、このアーキテクチャー向けにクロスコンパイルされたアプリケーションをネイティブに実行することができます。「ネイティブ」とは、組込み Linux* システム内のコプロセッサーでプログラム全体が実行され、ホストでは何も実行されないことを意味します。

ホストまたはターゲットでプログラムをすべて実行するだけでなく、インテル® Xeon Phi™ ではアプリケーションのパフォーマンス・クリティカルな部分をコプロセッサーにオフロードして実行することもできます。選択した計算部分をオフロードするオフロードモデルは、C、C++、Fortran でコード修飾を挿入するだけです。修飾には、オフロード言語拡張 (LEO: Language Extensions for Offload) を用います。プラグマ (**図 1** を参照) または共有キーワード (**図 2** を参照) のいずれかを変数および関数の言語拡張として指定できます。インテルは、オフロード機能が OpenMP* に含まれるように、OpenMP* の標準化にあたってさまざまな取り組みを行ってきました。

1

```
void my_func() {
    int my_var = 0;
    #pragma offload target (mic)
    #pragma omp parallel for
    for (i=0; i < count; i++) {
          // 処理を実行
    }
}
```

プラグマコード修飾

2

```
_Cilk_shared int my_shared_var;

_Cilk_shared void my_func() {
    my_shared_var = 42;

    #pragma omp parallel for
    for (i=0; i < count; i++) {
          // 処理を実行
    }
}

// オフロード関数を呼び出す
_Cilk_offload my_func();
```

オフロード言語拡張 (LEO)

3

```
#include <stdio.h>
#include "offload.h"

#define SIZE 100

#ifdef OFFLOAD
#define MODE "OFFLOAD"
#else
#define MODE "NATIVE"
#endif

#ifdef OFFLOAD
__declspec(target(mic))
#endif
int A[SIZE], B[SIZE];

#ifdef OFFLOAD
__declspec(target(mic))
#endif
int dotproduct;

int compute_dotproduct () {
    int i;
    int ret;

    #ifdef OFFLOAD
```

コプロセッサーのサンプルコード (dot_product.c)

3 (続き)

```
    #pragma offload target (mic)
    #endif
    {
        #pragma omp parallel shared(A, B) private(i)
        {
            #pragma omp master
            {
                #ifdef __MIC__
                ret = 1;
                #else
                ret = 0;
                #endif
            }

            #pragma omp for reduction(+:dotproduct)
            for (i=0; i < SIZE; i++) {
                dotproduct = dotproduct + (A[i] * B[i]);
            }
        }
    }
    return ret;
}

void init () {
    int i;
    /* ダミー値で初期化 */
    for (i = 0; i < SIZE; i++) {
        A[i] = i * 2;
        B[i] = i + 2;
    }
}

int main (int argc, char *argv[]) {
    int mode;
    init ();
    mode = compute_dotproduct ();
    printf("[%s] computed dot product on %s = %i\n",
        MODE, (mode == 1 ?"MIC" :"HOST"), dotproduct);
    return 0;
}
```

コプロセッサーのサンプルコード (dot_product.c)

OpenMP* 4.0[1] 仕様のドラフト版は、LEO に含まれる機能の標準化バージョンを提供し、新しいコーディング作業のための新しい構文の選択肢となります。開発者にとっての主な利点は、コードにすでに存在する OpenMP* 宣言子を拡張するだけで並列領域をコプロセッサーにオフロードできることです。これらの機能はすべてインテル® コンパイラーおよびツールでサポートされます。

開発者は、絶対にバグがない完璧なソフトウェアが存在しないことを知っています。では、アプリケーションがコプロセッサーで実行されることを想定していない場合はどうすれば良いでしょうか。デバッグツールの重要性は、新しいコードを開発することよりもバグを発見することに時間をかける開発者にとっては明白です。デバッガーは、開発したプログラムのバグを追跡し、分離し、取り除く開発者の作業を支援します。この一連の処理は、オフロードモデルのように複数のターゲットに分散される実行モデルの場合は特に困難です。この記事では、インテル® Parallel Studio XE を利用してインテル® Xeon Phi™ コプロセッサーで全体または一部を実行するプログラムのデバッグ作業に取り組む方法を説明します。

コプロセッサー・デバッグのシナリオ

開発者にとって主なデバッグシナリオは 2 つあります。最初のシナリオは、コプロセッサー側で完全に実行されるネイティブ・アプリケーションのデバッグです。ネイティブ・アプリケーションをデバッグするには、コプロセッサーにターゲット・エージェント、ホストに (ターゲット・エージェントと接続している) デバッグエンジンが必要です。このシナリオの利点は、ターゲットの端末でコマンドライン・デバッグを行う代わりに、ホストのデバッグエンジンで GUI (たとえば、Eclipse*)[2] を使用できることです。

2 つ目のシナリオは、選択した部分を 1 つ以上のコプロセッサーにオフロードして残りをホストで実行するオフロード・アプリケーションのデバッグです。このシナリオでは、プログラムの特定の部分を異なるコプロセッサー・ターゲットで実行するため、複数のデバッグエンジンが必要になります。ホストとターゲットに対応するコードブロックは個別のプロセスで実行され、異なるデバッグエンジンでデバッグされます。開発者ができるだけスムーズにこのデバッグを行うには、デバッグ・アーキテクチャーを表示せずに、すべての異なるプロセスにわたって透過的なデバッグセッションを開発者に提供する GUI ソリューションが必要です。

インテル® Parallel Studio XE には、ネイティブ・アプリケーションのデバッグ用のコプロセッサー対応 GNU* Debugger (GDB*)[3] と、Linux* および Windows* 用 GDB に基づく完全なオフロードデバッグの両方のソリューションが備わっています。次に、ネイティブおよびオフロード・アプリケーションを例に、これらのソリューションと使用法を説明します。

1. http://openmp.org/wp/openmp-specifications/
2. http://www.eclipse.org/
3. http://www.gnu.org/software/gdb/

ネイティブ・コプロセッサー・アプリケーションのデバッグ

ネイティブ・コプロセッサー・アプリケーションは通常、ホストでクロスコンパイルした後、コプロセッサーにコピーし、コプロセッサーで実行します。論理的には、ターゲット上で直接デバッグすることも理にかなっています。インテル® Parallel Studio XE には、コプロセッサーでこれらのアプリケーションをデバッグする GDB ターゲット・エージェントとターゲット・エージェントに接続する GDB が含まれています。どちらも、アプリケーション・バイナリー・インターフェイス (ABI)、512 ビット幅のベクトルおよび関連マスクレジスターなど、アーキテクチャー固有の機能にすべて対応しています。

では、シンプルなネイティブ・アプリケーションを使ってターゲットで直接デバッグセッションを開始してみましょう (**図 3** を参照)。基本的なデバッグセッションの開始と実行方法は次のとおりです。

1) インテル® コンパイラーで、インテル® メニー・インテグレーテッド・コア (MIC) に基づく、インテル® Xeon Phi™ コプロセッサー向けにプログラムをクロスコンパイルします。

```
icc -g -O0 -mmic -openmp -o dot_product dot_product.c
```

2) アプリケーションとソースをターゲットにコピーします。

```
scp dot_product* mic0:~
```

3) さらに、コンパイラーのランタイムシステムとアプリケーションで必要な MIC 共有オブジェクトもコプロセッサー・ターゲットにコピーする必要があります。たとえば、インテル® Parallel Studio XE のイントール・ファイルに含まれる OpenMP* ライブラリー (libiomp5.so) などです。

4) ターゲット・エージェントをコプロセッサーにコピーします。

```
scp gdbserver mic0:~
```

5) コプロセッサーでターゲット・エージェントを開始します。

```
ssh mic0 "~/gdbserver --once :1234 ~/dot_product"
```

6) ホストで GDB を起動してターゲットに接続します。

```
gdb ./dot_product
```

```
(gdb) target remote mic0:1234
```

GDB がコプロセッサーのターゲット・エージェントに接続され、ターゲット上のアプリケーションを直接デバッグできるようになります。C/C++ アプリケーションのメイン・エントリー・ポイントに移動するには、GDB コンソールで “break main” および “continue” と入力します。main 関数にブレークポイントが設定され、デバッガーの制御下でデバッグが続行されます。並列領域の内部で停止したときに、現在のスレッドとスレッドの状態を確認できます。現在アクティブなすべてのスレッドの概要を表示するには、“info threads” と入力します。インテル® Xeon Phi™ コプロセッサー固有のレジスターの内容を確認することもできます。32 の ZMM (512 ビット SIMD) レジスターにアクセスするには、それぞれ “print $zmm0” から “print $zmm31” と入力します。このほかにも、デバッグセッション中に GDB を利用してさまざまな処理を行うことができますが、この記事では詳しく取り上げません。コマンドラインから実行できる処理についての詳細は、オンラインヘルプ (“help” と入力すると表示されます) および GDB のドキュメント [4] を参照してください。

4. http://sourceware.org/gdb/current/onlinedocs/gdb/

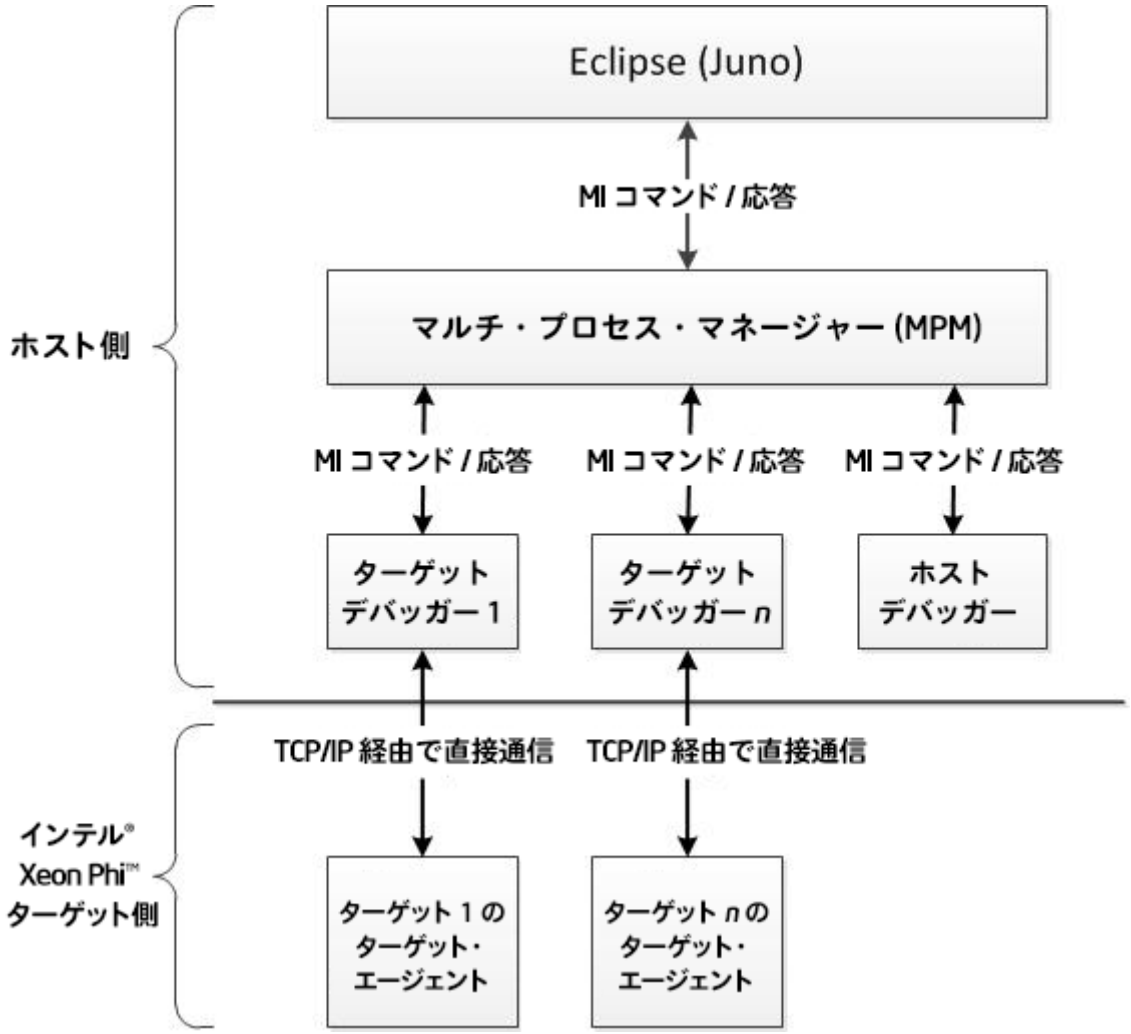
オフロード・アプリケーションのデバッグ

ネイティブ・コプロセッサー・アプリケーション用の従来のデバッグアプローチとは異なり、オフロードモデルでは、異なるプロセスで同時に実行するオフロード・アプリケーションのホスト側とターゲット側のコードをデバッグする必要があります。このデバッグ・ソリューションの究極の目標は、制御フローとデータレベルで、プログラムの透過的かつ単一ソースレベル・ビューを開発者に提供することです。プログラムの異なる部分を複数のホストが別々のプロセスで実行するため、課題は単一の制御とデータビューを再構築することです。

1 つのシステムで複数のコプロセッサー・カードを搭載できるオフロードモデルは、1 つの GDB デバッグエンジンではデバッグできない、大規模なヘテロジニアス・プログラミング環境を構成します。ホスト側のコードを 1 つのプロセスで実行し、各オフロード部分を別のプロセスで実行するため、開発者は n+1 のデバッグセッションを実行し、それぞれのオフロードプロセスにアタッチする必要があります。ホスト側のプロセスでは 1 つのデバッグセッション、ターゲット側のプロセスでは n のデバッグセッションを実行します (n はシステムに搭載されるインテル® Xeon Phi™ コプロセッサーの数)。デバッグセッションがプロセスの 1 つにアタッチされると、そのプロセスで実行する特定のコードのみデバッグできます (たとえば、1 つのオフロード領域)。主な短所は、どのデバッグセッションが現在アクティブで、どのセッションが何を実行しているかを開発者が把握しなければならないことです。これらの情報を直感的に認識できないことは明らかです。ここでの課題は、これらのセッションをできるだけ開発者に見せずに、単一のデバッグセッションのように見せることです。このレベルのデバッグの透過性を実現するため、インテル® Parallel Studio XE 2013 には Linux* デバッグ用のツール (マルチ・プロセス・マネージャー) と Windows* デバッグ用のツール (Visual Studio* Debug Extension) が含まれています。

Linux*: マルチ・プロセス・マネージャー (MPM)

MPM は、Linux* ホスト上の複数の GDB インスタンスと Eclipse* GUI 間の通信レイヤーの役目を果たします。MPM の主な処理は、GUI からのマシン・インターフェイス (MI) コマンドを正しいデバッグエンジンにリダイレクトする単純なものです。MI は、ラインベースでテキスト指向の、GDB および Eclipse* にアクセスするインターフェイスです。また、MPM は、ブレークポイントがヒットされると、デバッグエンジンからの MI 応答を集計して、応答を GUI に転送します。GUI は受け取った応答に応じてそれぞれのウィンドウを更新します。**図 4** は、一連の処理を図で示したものです。

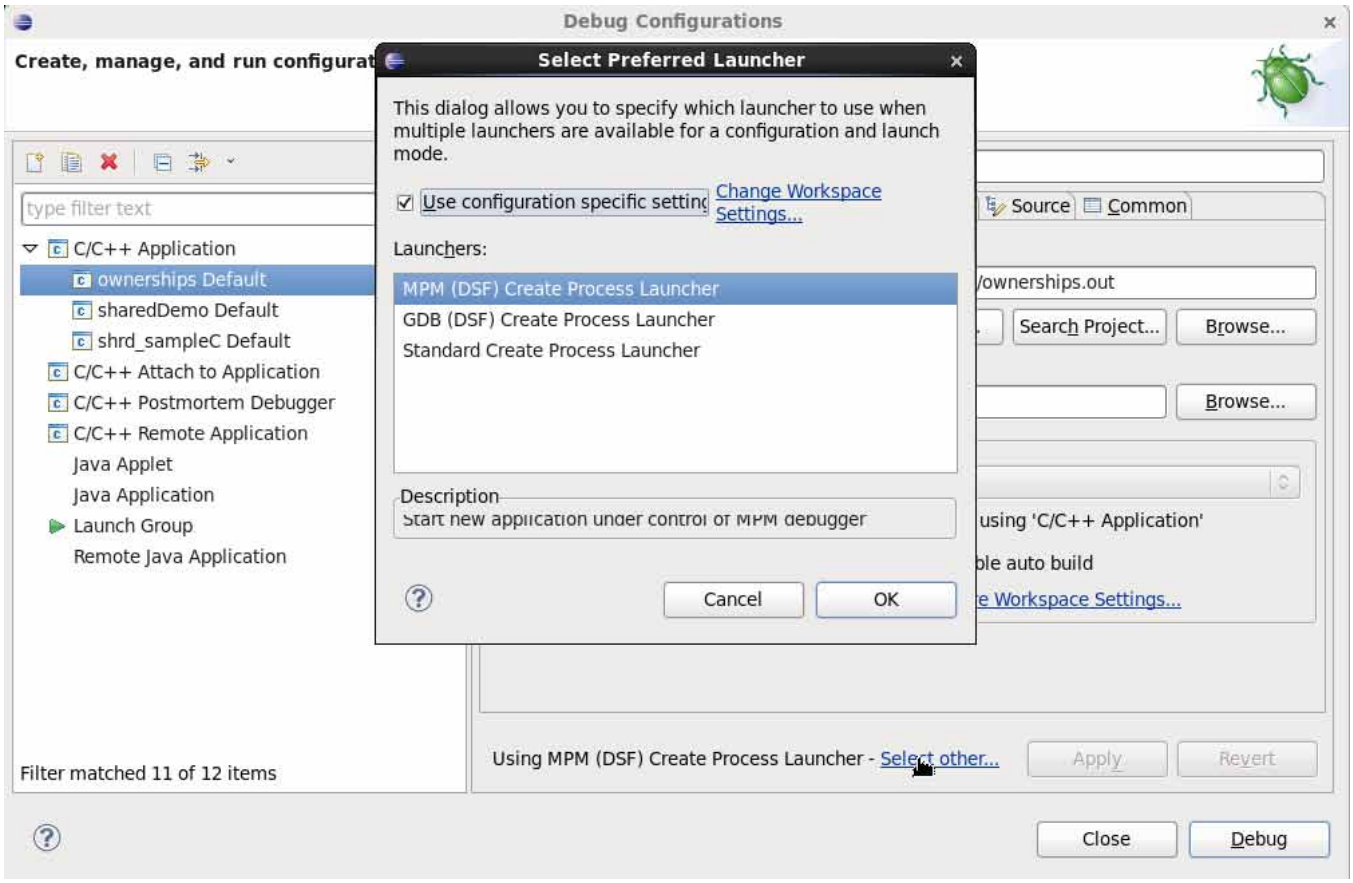


4 MPM の内部ワークフロー

Eclipse* 用 MPM プラグインは、インテル® Parallel Studio をインストールした場所に配置され、標準の Eclipse* プラグインのインストール手順でインストールできます。

プラグインがインストールされたら、開発者は既存のコードをインポートするかプロジェクトをゼロから作成します。**図 3** のサンプル・アプリケーションをオフロード・アプリケーションとして動かすには、-DOFFLOAD を指定してコンパイルします。Fortran オフロードプログラムをデバッグする場合は、Eclipse* 標準リポジトリーから “Photran”[5] コンポーネントをインストールして、Eclipse* で Fortran を処理できるようにします。

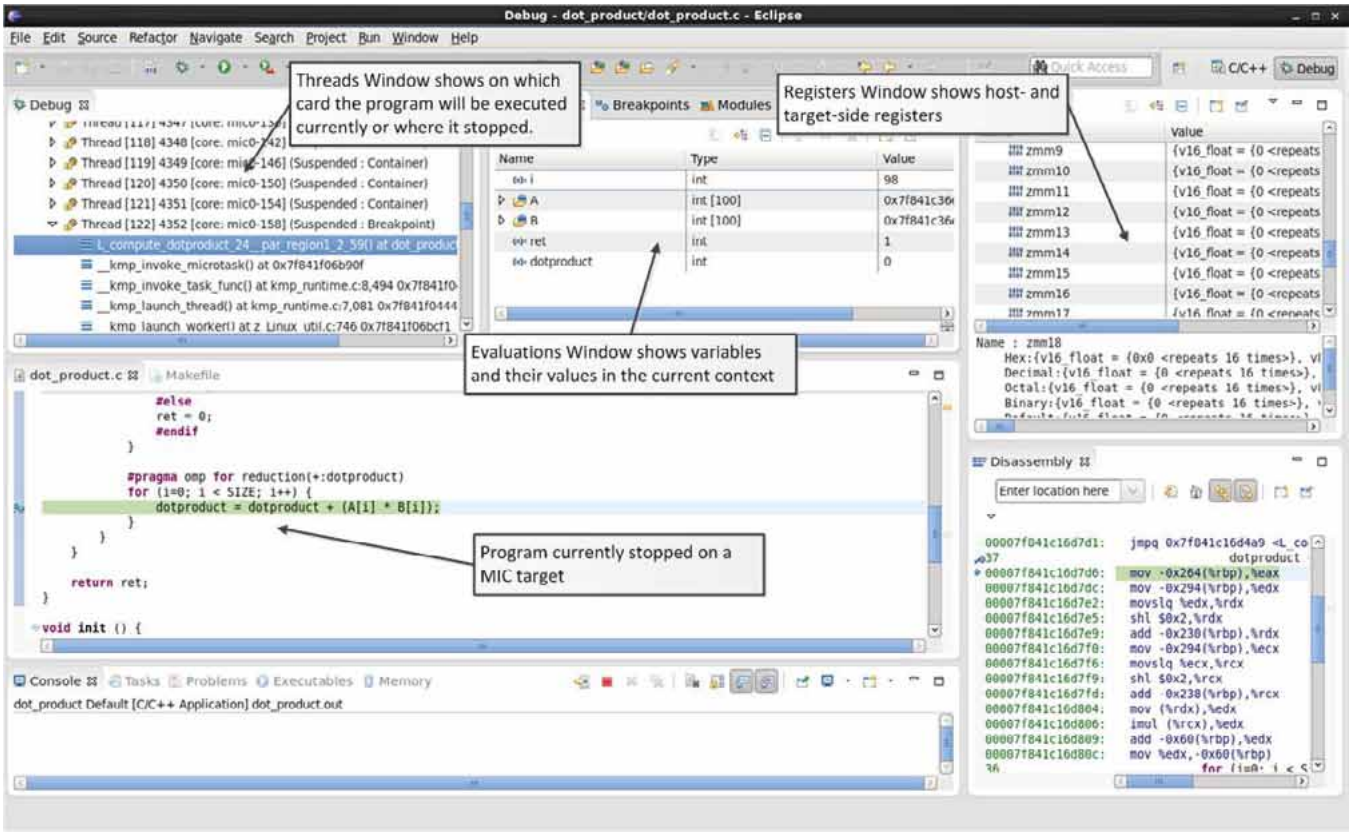
実際にデバッグセッションを開始するには、“デバッグ構成” が必要です。デバッグ構成には基本的なデバッグ設定を定義します。デバッグ構成を作成するには、Eclipse* のメインダイアログで [Run] > [Debug Configurations] を選択します。表示されたダイアログで、プロセスランチャーとして MPM を選択し (**図 5** を参照)、デバッグするアプリケーションを設定します。残りの項目は MPM により自動的に設定されます。



5 新規デバッグ構成作成時に Eclipse* MPM プラグインを選択

5. http://www.eclipse.org/photran/

デバッグセッションが開始すると、Eclipse* がデバッグビューになり、ホストとターゲット側のレジスター、現在スコープに含まれる変数の値、スタックトレース、その他の一般的なデバッグ関連情報をすべて確認できます。**図 6** は、Eclipse* のオフロードプログラムのデバッグセッションを示しています。



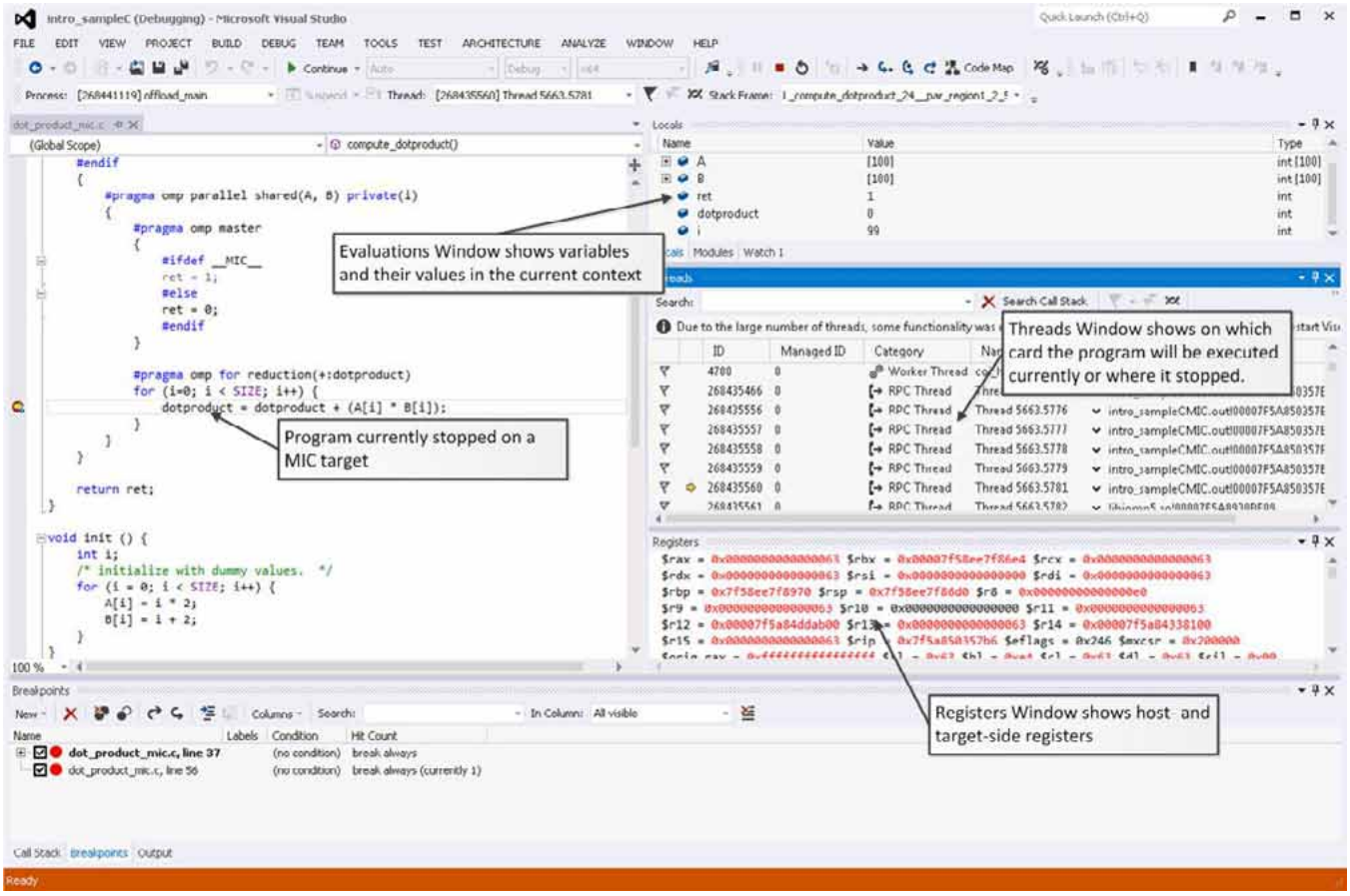
6 変数、レジスター、逆アセンブリー、スレッドを含む、Eclipse* のオフロード・デバッグ・セッションで最も重要なウィンドウ

Windows*: Visual Studio* (VS) Debug Extension

インテル® Parallel Studio XE は、Windows* 環境におけるオフロードモデルのデバッグもサポートしています。パッケージをインストールするとき、インテル® Xeon Phi™ コプロセッサー向け Visual Studio* Debugger Extension がインストールされ、インテル® コンパイラーに統合されます。このインテル® Xeon Phi™ コプロセッサー向け Visual Studio* Debug Extension の内部的な機能は、MPM とほぼ同じです。違いは、VS はデバッグエンジンとの通信に MI を使用していないため、GDB で理解できるように、Debug Extension が VS からの GUI リクエストを MI に翻訳していることです。逆に、GDB デバッグエンジンからの MI 応答は VS への GUI リクエストに翻訳されます。MPM と同様に、このソリューションは複数のコプロセッサー・ターゲットにわたって透過的なデバッグセッションを開発者に提供します。

オフロード・アプリケーションのデバッグを開始するには、Visual Studio* のソリューションが必要です。**図 3** のコードをサンプル・アプリケーションとしてここで使用することもできます。Linux* の Eclipse* デバッグ・ソリューションとの違いは、開発者が何も設定する必要がない点です。コプロセッサーのデバッグセッションは、Visual Studio* のシンボルバーの “Local Windows Debugger” ボタンをクリックして開始します。

Visual Studio* がデバッグビューになり、必要なデバッグ情報をすべて確認できます。**図 7** は、Visual Studio* 2012 のデバッグセッションの例です。



7 変数、スレッド、レジスターを含む、Visual Studio* のオフロード・デバッグ・セッションのウィンドウ

まとめ

ネイティブ・コプロセッサー・アプリケーションのデバッグとは異なり、オフロードモデルではプログラム実行が異なるプロセスとターゲットに分散されます。開発者がさまざまな問題に取り組めるように、インテル® Parallel Studio XE にはネイティブ・アプリケーションとオフロードプログラム用のデバッグ・ソリューションが用意されています。すべてのツールはインテル® Xeon Phi™ コプロセッサーに対応しており、コマンドライン、Eclipse*、Visual Studio* で、プログラムの制御フローと変数の単一ソースレベル・ビューを提供します。

関連ドキュメント

- インテル® Xeon Phi™ コプロセッサー向けネイティブ・アプリケーションのビルド
  http://www.isus.jp/article/mic-article/building-a-native-application-for-intel-xeon-phi
- インテル® メニー・インテグレーテッド・コア・アーキテクチャー向けプログラミングとコンパイル
  http://www.isus.jp/article/mic-article/xeon-phi/

フルスケーリングに向けて : WRF (Weather Research and Forecast) モデルとインテル® Cluster Studio XE 2013

Mark Lubin、Scott McMillan インテル コーポレーション
Christopher G. Kruse、Davide Del Vento 米国大気研究センター (NCAR)
Raffaele Montuoro テキサス A&M 大学

概要

WRF (Weather Research and Forecast) モデルは、広範な気象アプリケーションで広く利用されている次世代のメソスケール気象予報システムです。我々は、インテル® MPI ライブラリーと DAPL UD を利用して、65,536 コアのインテル® Xeon® E5-2670 プロセッサーを搭載したシステムで WRF モデルをスケーリングすることに成功しました。この研究には、ハリケーン カトリーナの高解像度 (1km) シミュレーションに基づく新しいワークロードが選択されました。この記事では、インテル® コンパイラーおよびインテル® MPI ライブラリーを含むインテル® Cluster Studio XE 2013 により達成した、“汎用” スーパーコンピューターにおける WRF モデルのスケーラビリティーについて説明します。米国大気研究センター (NCAR)、テキサス A&M 大学、インテルのチームは、NCAR の Yellowstone システム (2012 年 11 月の Top500 リストの 13 位) で WRF をスケーリングするべく協力して作業に取り組みました。通信プロコルの DAPL と DAPL UD との選択は、標準のインテル MPI を使用する上でそれほど難しくはなく、Yellowstone システムの限界までスケーリングすることができました。

WRF (Weather Research and Forecast) モデル

WRF (Weather Research and Forecast) モデルは、運用、研究、教育環境で利用される主要なオープンソース数値気象予報モデルの 1 つで、米国大気研究センター (NCAR)、米国海洋大気圏局 (NOAA)、米国空軍気象局 (AFWA)、米国海軍研究所 (NRL)、オクラホマ大学、米国連邦航空局 (FAA) などが連携して開発しています。WRF は現在、NCAR、NCEP、AFWA、その他のセンターで運用されており、環境科学分野で多くのユーザー数を誇っています (Skamarock et. al., 2008 を参照)。我々は、Advanced Research WRF (ARW) ダイナミック・コアで作成され、NCAR 地球システム研究所 (NESL) のメソスケール・マイクロスケール気象部門 (MMM) により開発および保守されている、WRF の最新バージョン 3.5 (2013 年 4 月 18 日) を使用しました。

インテル® MPI ライブラリーの最新リリースであるバージョン 4.1 には大規模なシステム向けに設計された機能が含まれているため、WRF はこれらのスケーリング機能をテストする最適な候補でした。

WRF は拡張性を備えており、最先端の物理学、科学、水文学モデルや、研究コミュニティーによって開発されたほかの機能を容易に追加することができます。WRF は多くのコア数で適切にスケーリングされることでも知られています (Michalakes et. al., 2008)。インテル® MPI ライブラリーの最新リリースであるバージョン 4.1 には大規模なシステム向けに設計された機能が含まれているため、WRF はこれらのスケーリング機能をテストする最適な候補でした。ハリケーン カトリーナの大規模な高解像度データセット (Patricola et. al., 2012) のシミュレーションには、NCAR のワイオミング・スーパーコンピューター・センター (NWSC) に設置されているインテル® Xeon® プロセッサー・ベースのペタスケール Yellowstone HPC が使われました。2005 年 8 月 25 日午前 0 時 UTC (協定世界時) に開始し、水平解像度 1Km でメキシコ湾をカバーするグリッドの初期条件と境界条件が WRF シミュレーションに設定され、ハリケーン カトリーナの上陸時間と場所の予測が行われました (実際には、2005 年 8 月 25 日にフロリダ半島に上陸してメキシコ湾に抜けた後、2005 年 8 月 29 日にルイジアナ州に再上陸)。

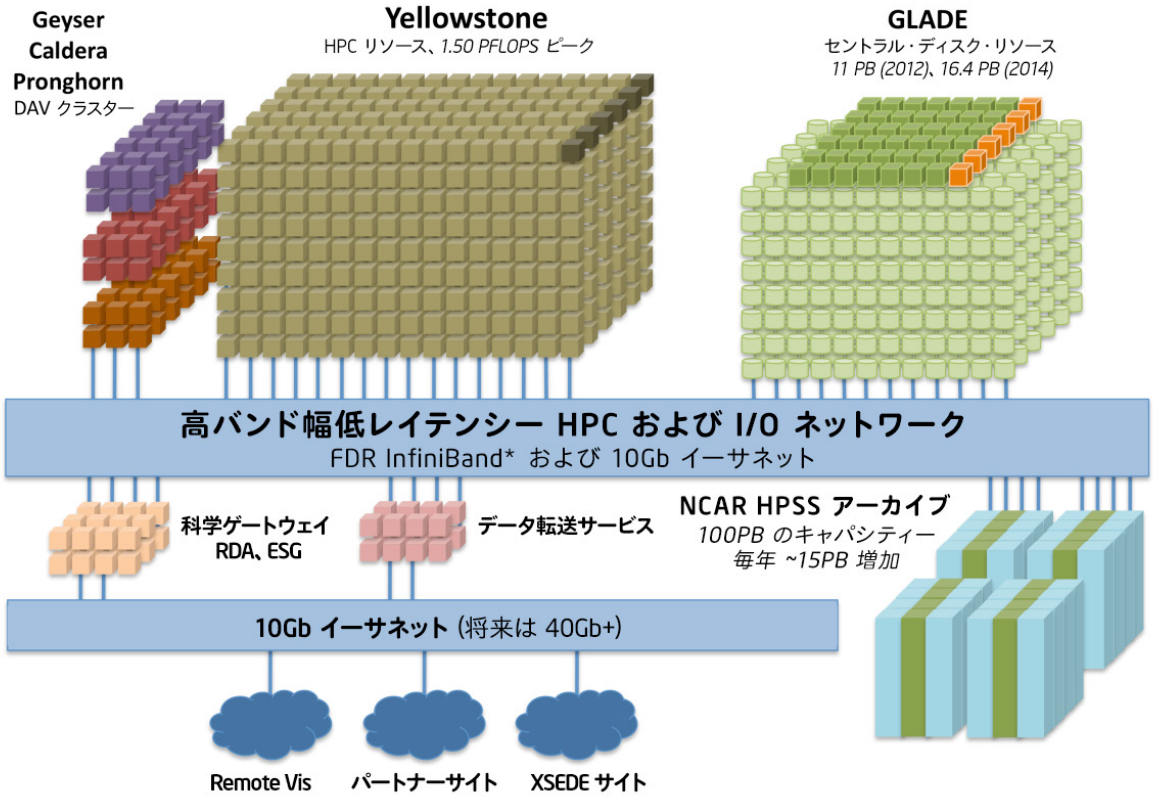
計算環境 : Yellowstone、インテル® Cluster Studio XE 2013、インテル® MPI ライブラリー、DAPL UD

この研究は、2012 年後半に行われた Yellowstone システムでフルスケーリングのリアルタイム WRF シミュレーションを実行するという最初の試みで得られた結果を収集したものです。

1 Yellowstone クラスター

Yellowstone は、2 つのインテル® Xeon® E5-2670 プロセッサー (8 コア) からなる 4,518 の計算ノードで構成されており、コアの総数は 72,288 個です。各ノードは Fat Tree トポロジー InfiniBand* ネットワークで接続されています。

WRF バージョン 3.5 は、インテル® Composer XE 2013、インテル® MPI ライブラリー 4.1、netCDF ライブラリー 4.2 を使用して作成されました。指定したコンパイラー・オプションは “-O3 -align all -fno-alias -fp-model precise” です。この研究のため、WRF ソースコードに多少の修正が加えられました。WRF 付属のランタイム・システム・ライブラリー (RSL) は MPI プロセスの上限が 10,000 に設定されていたため、より多くの MPI プロセスを利用できるように RSL_MAXPROC の値を増やしました。また、標準出力とエラーファイル名のすべてのフォーマット・インスタンスを変更しました。16K 以上のコア数ではディスク I/O が問題になりましたが、我々の目標はシミュレーション速度の評価であったため、出力を書き込まないように変えました。さらに、WRF モデルユーザーのサイトの “既知の問題と修正” パッチ (2013 年 6 月 19 時点) を適用しました (**http://www.mmm.ucar.edu/wrf/users/wrfv3.5/known-prob-3.5.html**)。



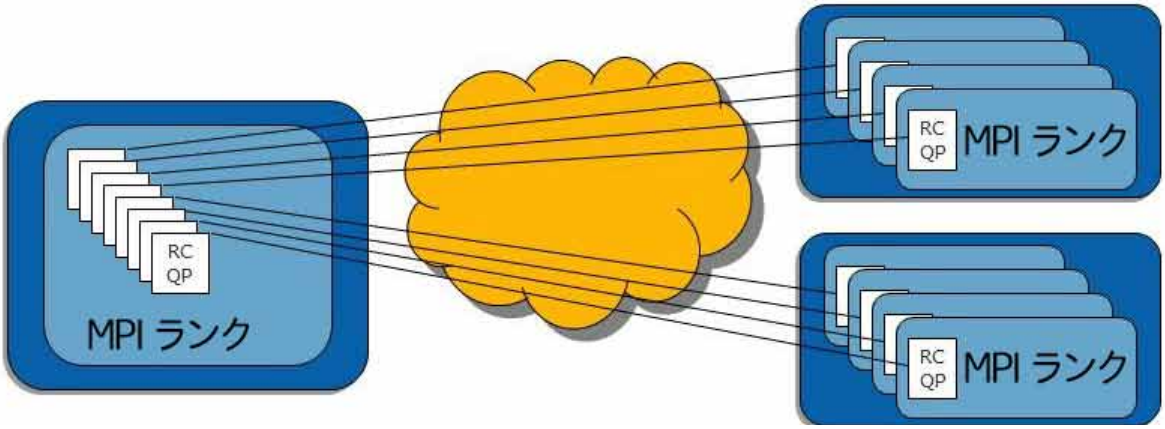
2 Yellowstone クラスター環境

インテル® MPI ライブラリーは、DAPL を含む複数のインターフェイスを InfiniBand* ネットワークに提供します。InfiniBand* は、キューペア (QP) オブジェクトによりデータ転送を開始し、特定のサービスタイプごとに、個別に InfiniBand* QP を設定します。次に、インテル® MPI ライブラリーの 2 つのタイプのサービス - 信頼性のある接続 (RC) と信頼性のないデータグラム (UD) - について説明します。

RC QP タイプは、すべてのメッセージおよび RDMA 操作について信頼性のある転送を行います。ただし、すべての MPI ランクについて十分なリソース割り当てが必要です。このメモリーはアプリケーションには利用できません。

この研究は、2012 年後半に行われた Yellowstone システムでフルスケーリングのリアルタイム WRF シミュレーションを実行するという最初の試みで得られた結果を収集したものです。

信頼性のある接続 (RC) モード

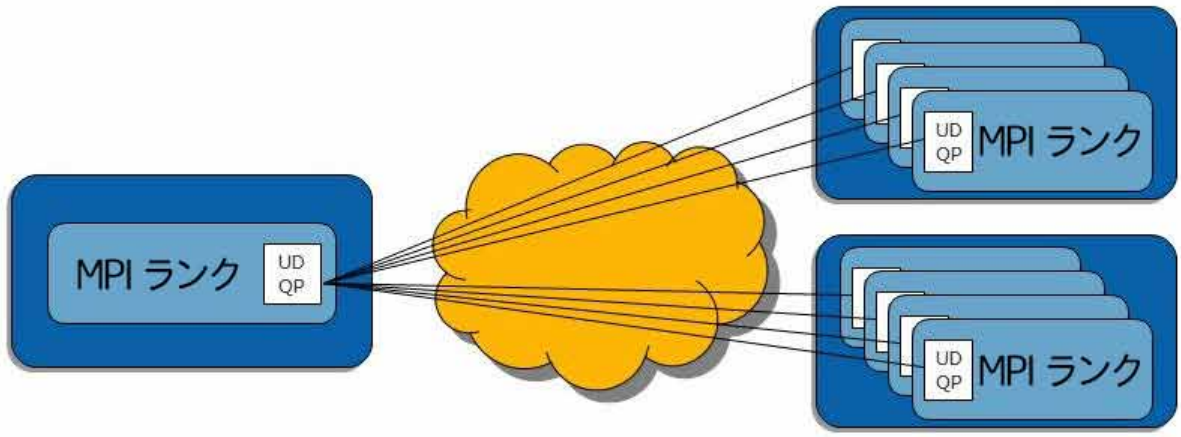


- RDMA サポート
- 大規模なメッセージ、セグメント化の必要なし
- 信頼性のあるインオーダー・デリバリー、フロー制御
- 1 対 1 エンドポイント = スケールに伴い QP フットプリントが増加
  - ノードあたりの QP = $Ncores^2$ * (Nnodes-1)

3 インテル® MPI ライブラリーの信頼性のある接続 (RC) モード

その名のとおり、UD QP タイプは送信するメッセージの配信を保証せず、RDMA 操作を実行できません。しばらくすると、配信されなかったメッセージの再送信リクエストが行われるため、一時的なネットワーク障害に対するフォールトトレランスは向上します。RC QP タイプと異なり、UD QP はほかの QP に接続されないので、その“アドレス”を知っているファブリックのほかの UD QP からメッセージを受け取ることができます。そのため、UD 操作をサポートするメモリーの最大量は、RC よりもはるかに少なくなります。欠点は、インテル® MPI ライブラリーが MTU サイズで各メッセージのセグメント化と再アセンブリーを行う必要があることです。ただし、この欠点は動的接続により最終的にすべて接続される大規模なクラスターでは問題になりません。メモリーの使用率が向上するだけでなく、アプリケーションによってはスケーラビリティーも向上します。

信頼性のないデータグラム (UD) モード



- RDMA サポートなし - 送信 / 受信操作のみ使用
- メッセージサイズが MTU で制限される - 現在の HCA では 2K
  - セグメント化 / 再アセンブリーが必要 > MTU
- メッセージデリバリーに信頼性がない
  - パケットがドロップする可能性あり
  - ソフトウェアで信頼性のあるフロー制御を提供する必要あり
- 多対 1 のエンドポイント = スケーリングしてもランクごとの QP が固定
  - ノードあたりの QP = Ncores*6 (bcopy/zcopy による MPI 実装)

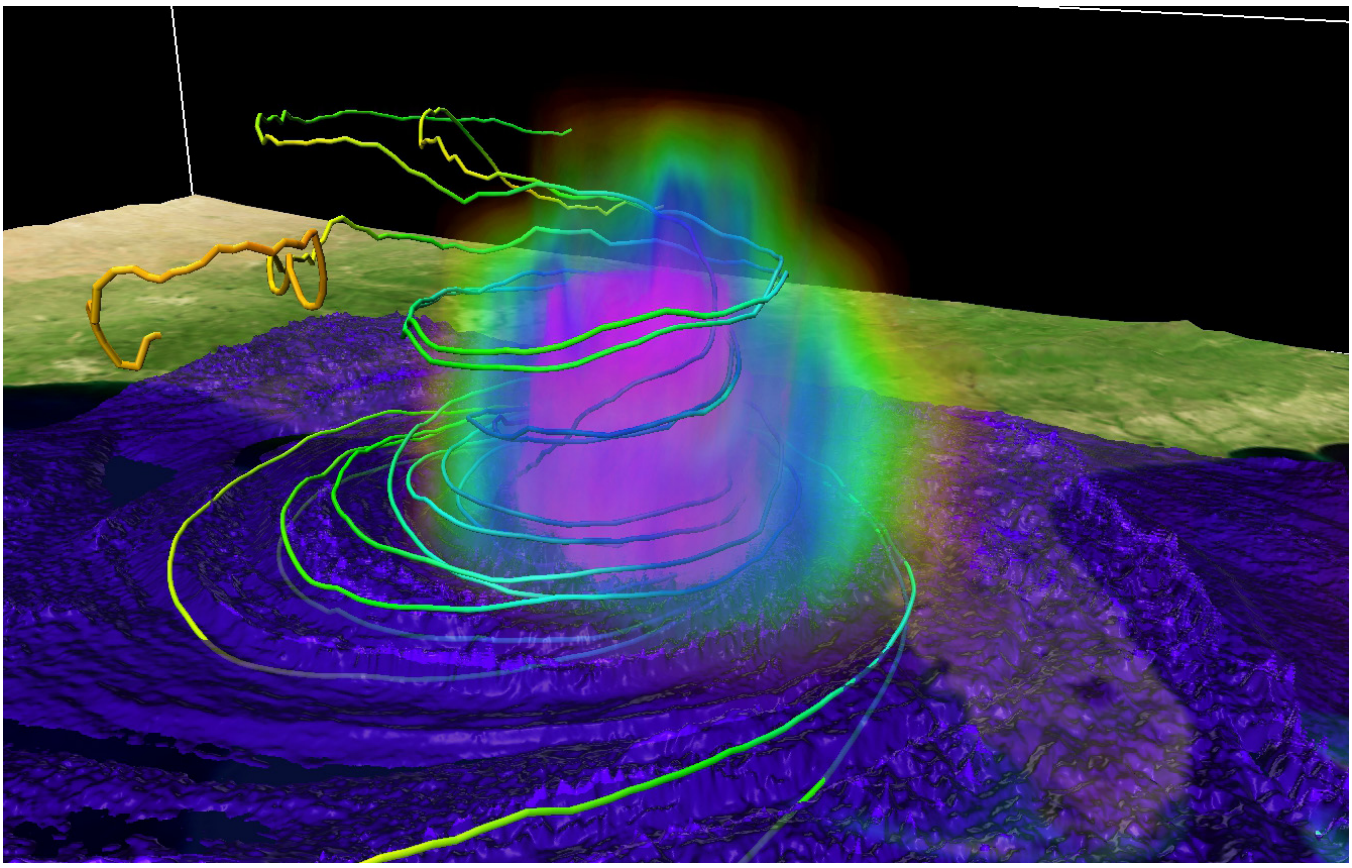
4 インテル® MPI ライブラリーの信頼性のないデータグラム (UD) モード

そのため、Yellowstone でより優れたスケーラビリティーが達成できるように、インテル® MPI ライブラリーの DAPL UD モードを使用しました。さらにメモリー使用量を減らし、タイムアウト設定を増やすためにデフォルトの DAPL UD 値を変更しました (**付録 A** を参照)。使用した DAPL のバージョンは 2.0.36 です。

ここでは、WRF の効率良い並列実装、インテル® MPI ライブラリー、インテル® コンパイラー、解析ツールのスケーラビリティー、インテル® Xeon® プロセッサー・ベースの Yellowstone システムのスケーラビリティーを実証しました。

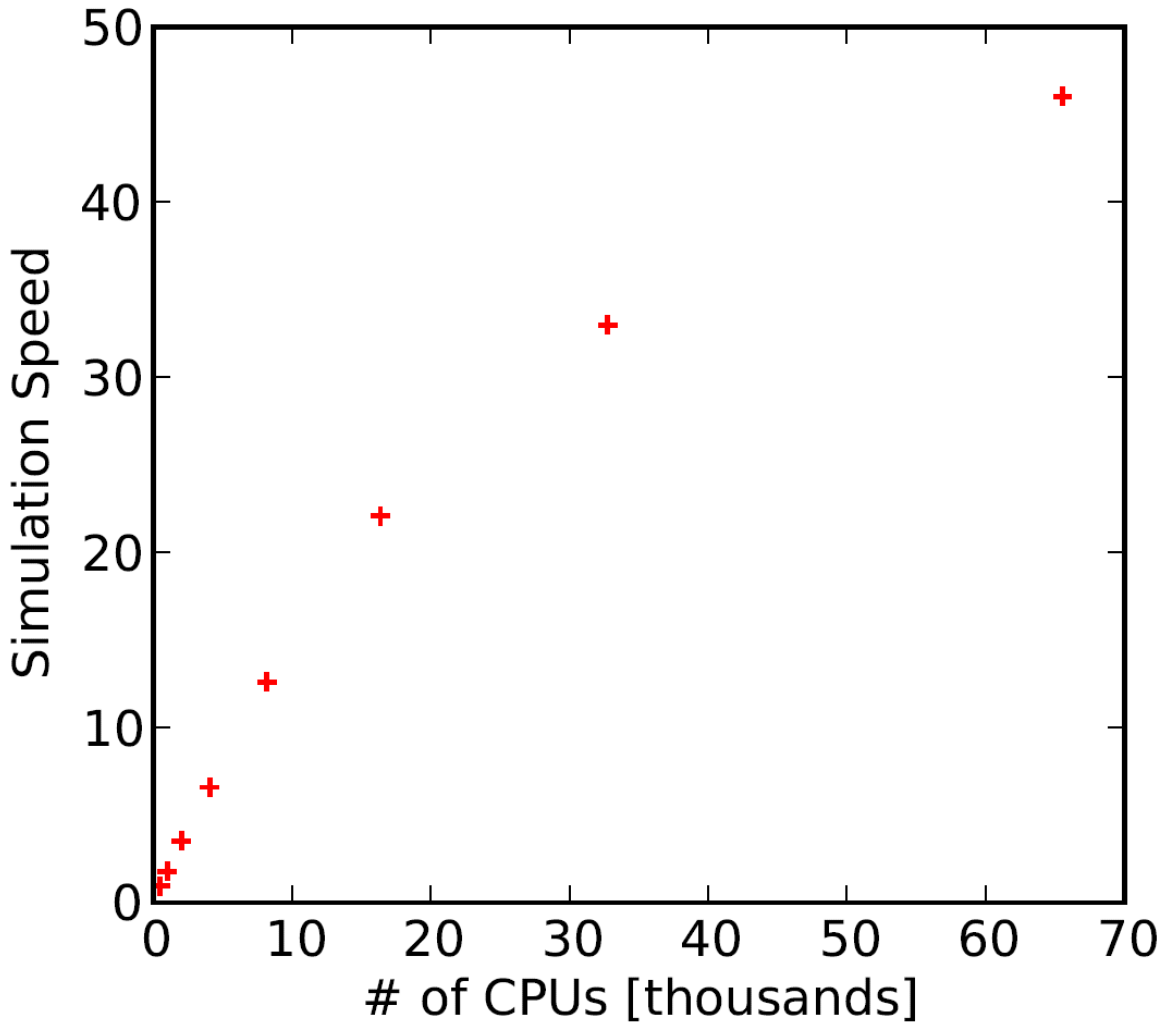
スケーリング結果

解像度 1Km のハリケーン カトリーナの大気シミュレーションは、35 の垂直レベルおよび 3 秒ステップで実行され、合計 371,227,850 のグリッドポイント (3,665 x 2,894 x 35) が含まれていました。**図 5** は、実行中に得られた結果の一部を示しています。



5 解像度 1Km でシミュレーションされたハリケーン カトリーナの 2005 年 8 月 28 日午前 0 時 UTC (協定世界時) 時点 (再上陸 11 時間前) の様子。風速と風向きはカラフルなボリューム・レンダリングと 3D 流線で表現されています。ハリケーンの中央のカラフルな部分は風速で、より不透明な部分はより速い風速を示します (89 マイル毎時まで表示)。水蒸気混合比 (11.2 g/kg) は青の等値面で表現されています。ビジュアル化には、VAPOR (Visualization and Analysis Platform for Ocean, Atmosphere, and Solar Researchers) を使用しました。

スケーラビリティーを評価するため、異なるコア数で実行してシミュレーション速度を測定しました。シミュレーション時間とウォールクロック時間の比率 (ディスク I/O に費やした時間を含まない、実行の時間ステップ数の平均) を実行のシミュレーション速度として定義しました (たとえば、速度 48 は 48 時間の気象予測のシミュレーションに 1 時間必要なことを表します)。コア数が 16K 以下の場合はベンチマークを 4 回実行し、32K の場合は 3 回、64K の場合は 1 回実行しました。**図 6** は、Yellowstone でカトリーナの 1Km ワークロードを実行したときの最良のシミュレーション速度を示しています。



6 Yellowstone クラスターでカトリーナの 1Km ワークロードを実行したときのスケーリング。WRF バージョン 3.5 は、インテル® Composer XE 2013 (“-O3 -align all -fno-alias -fp-model precise” コンパイラー・オプションを指定)、インテル® MPI ライブラリー 4.1、netCDF ライブラリー 4.2 で作成されました。また、付録 A の DAPL UD 設定を使用しています。

インテル® MPI ライブラリーを利用した実行ごとの変動は小さく、シミュレーション速度の相対標準偏差 (標準偏差 / 平均) は、16K および 32K 実行では約 3.5%、8K 以下では 0.2% 未満でした。初期の実行では、より大きな変動が計測されました。これらの問題は、InfiniBand* ケーブルの不具合、最適ではない IB ネットワーク・ルーティング、マシンで実行しているほかのジョブとの競合によるものでした。あるインスタンスでは、(システムチェックで認識されない) 1 つの “不良” ノードが含まれるとすべての実行でパフォーマンスが約 50% 低下することが分かりました。これらの問題はすべて、NWSC 開設直後の数カ月間は特に深刻でしたが、やがて大幅に改善されました。

カトリーナの 1Km ワークロードは、16K コアまではほぼ直線的にスケーリングしています。16K から 64K コアではスケーリングは直線ではなくなりますが、まだかなりの割合でシミュレーション速度が増加していることが分かります。計算するグリッドポイントが各コアに約 500 以上含まれる限り、スケーリングは直線のままです。計算の粒度が細かくなると、各 MPI タスクのハロ交換に含まれるグリッドポイントの一部を無視できなくなります。

まとめ

ハリケーン カトリーナがメキシコ湾に抜けてからルイジアナ州に再上陸するまで、わずか数日の出来事でした。このような急激な気象の変化に適切に対応するには、特に人口密度の高い地区に対して可能な限り早く警報を出せるように、高速かつ正確なシミュレーション能力が必要です。より大規模で詳細な気象予報を行う能力は、大規模な計算リソースの効率良い活用能力に依存します。この傾向は、比較的小規模のスーパーコンピューティング・センターでも一般的になりつつあります。ここでは、WRF の効率良い並列実装、インテル® MPI ライブラリー、インテル® コンパイラー、解析ツールのスケーラビリティー、インテル® Xeon® プロセッサー・ベースの Yellowstone システムのスケーラビリティーを実証しました。

謝辞

[著者より]
カトリーナのワークロードを提供していただいた C. M. Patricola、P. Chang、R. Saravanan (テキサス A&M 大学) の皆様に深く感謝します。

[M. Lubin、S. McMillan より]
インテルの同僚である Gergana Slavova、Arlin Davis の協力に感謝します。

[R. Montuoro より]
N. DE-SC0006824 の許可に伴う米国エネルギー省、科学部 (BER) の協力に感謝します。

[C. Kruse より]
Rich Loft、NCAR および NSF からの並列計算科学プログラムのサマー・インターンシップを通じた協力に感謝します。

付録 A

より多くのコア数で実行する際に使用されたインテル® MPI DAPL UD ライブラリー・ランタイムのパラメーターは次のとおりです。

I_MPI_FABRICS	shm:dapl
I_MPI_DAPL_UD_PROVIDER	ofa-v2-mlx4_0-1u
I_MPI_DAPL_UD	on
I_MPI_DAPL_UD_RNDV_EP_NUM	2
I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD	65536
I_MPI_DAPL_UD_SEND_BUFFER_NUM	8208
I_MPI_DAPL_UD_RECV_BUFFER_NUM	8208
I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE	8704
I_MPI_DAPL_UD_CONN_EVD_SIZE	2048
I_MPI_DAPL_UD_REQUEST_QUEUE_SIZE	80
DAPL_UCM_REP_TIME	16000
DAPL_UCM_RTU_TIME	8000
DAPL_UCM_RETRY	10
DAPL_UCM_QP_SIZE	8000
DAPL_UCM_CQ_SIZE	8000
DAPL_UCM_TX_BURST	100
DAPL_MAX_INLINE	64
DAPL_ACK_RETRY	10
DAPL_ACK_TIMER	20
DAPL_RNR_TIMER	12
DAPL_RNR_RETRY	10

参考資料

1. インテル® Composer XE 2013 (**http://software.intel.com/en-us/intel-composer-xe**)
2. インテル® MPI Library (**http://www.intel.com/go/mpi**)
3. Michalakes, J., et. al., 2008: WRF nature run. J. of Physics: Conference Series, 125, doi: 10.1088/1742-6596/125/1/012022.
4. NCAR, 2013: NCAR-Wyoming Super Computer (**http://www2.cisl.ucar.edu/resources/yellowstone**)
5. Patricola, C. M., P. Chang, R. Saravanan, and R. Montuoro, 2012:The Effect of Atmosphere-Ocean-Wave Interactions and Model Resolution on Hurricane Katrina Coupled Regional Climate Model.Geophys.Res.Abs., 14, **(http://meetingorganizer.copernicus.org/EGU2012/EGU2012-11855.pdf)**
6. Skamarock, W. C., et al., 2008:A description of the advanced research wrf version 3.Tech. rep., National Center for Atmospheric Research.NCAR Tech.Note NCAR/TN-4751STR, 113 pp (**http://www.mmm.ucar.edu/wrf/users/docs/arw_v3.pdf**)
7. WRF システム開発用のリサーチ･コミュニティーの優先事項。WRF リサーチ･アプリケーション委員会、2006 年 12 月で準備。

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。この注意事項で対象としている特定の命令セットに関する詳細は、該当製品のユーザーズガイドまたはリファレンス・ガイドを参照してください。

改訂 #20110804

BLOG HIGHLIGHTS

潜在的な問題 (インテル® Xeon Phi™ コプロセッサーのメモリー制限) の回避

FRANCES ROTH (インテル コーポレーション) »

メモリー空間の超過

あらゆる Linux* システムと同様に、コプロセッサー上のオペレーティング・システムでも、物理的に利用可能な量よりも多くのメモリー空間を状況によって割り当てることができます。多くのシステムでは、メモリーの一部のページをディスクにスワップすることで割り当てを行います。このようなシステムで、物理メモリーの量とスワップ空間の量が超過すると、オペレーティング・システムはジョブを kill し始めます。

インテル® Xeon Phi™ コプロセッサーにはディスクが直接付属しないため、状況は複雑になります。

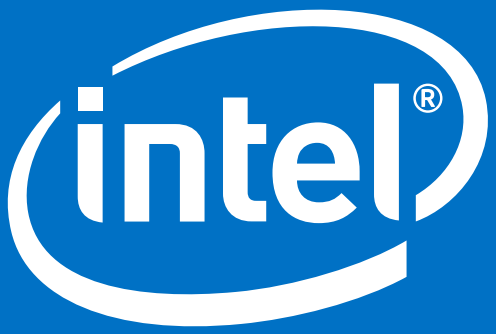
- 現在は、多くのシステムで 8GB の物理メモリーを利用可能です (私が最初に使ったパーソナル・コンピューターのメモリーは 4KB でしたから、その世代にとってはとてつもない大きさです)。
- デフォルトでは、メモリーの一部がコプロセッサーのファイルシステムに使用されます。
- デフォルトでは、コプロセッサーで利用可能なスワップ空間はありません。

RAM ディスクに使用するメモリーの制限

インテル® Xeon Phi™ コプロセッサーには直接アクセス可能なディスクドライブが存在しないため、コプロセッサーのデフォルトの root ファイルシステムは RAM ディスクに格納されます。これはファイルストレージに影響するだけでなく、実行するプログラムで利用可能なメモリーが少なくなります。

root ファイルシステムのサイズは、BusyBox* を利用して sh、cp、ls などの一般的な Linux* コマンドの多くを置換し、root ファイルシステムにコピーされる共有ライブラリーの数を制限することにより、小さく抑えられています。これらの対策を行っても、ファイルストレージに約 10MB のメモリーが消費されます。

この記事の続きはこちら (英語) でご覧になれます。 >



Software

The Parallel Universe