The Parallel Universe

intel Software

Issue 13
2013 年 2 月

プロダクト・エンジニアリングの革命:
高速かつ正確な有限要素解析 (FEA)

Noah Clemons、David Weinberg、
Jonas Dalidd、Dennis Sieminski

目次

編集者からのメッセージ
ソフトウェアが切り拓く未来

JAMES REINDERS 3

プロダクト・エンジニアリングの革命:
高速かつ正確な有限要素解析 (FEA)

NOAH CLEMONS、DAVID WEINBERG、JONAS DAALIDD、DENNIS SIEMINSKI 5

有限要素解析 (FEA) にインテル® マス・カーネル・ライブラリー (インテル® MKL) の PARDISO および DGEMM ルーチンを使用して解析時間を短縮する方法について説明します。エンジニアは非常に複雑な構造を検討できるようになります。

パラレルパワー:
インテル® Xeon Phi™ コプロセッサー用
ソフトウェアの最適化

AMANDA SHARP 13

インテル® Xeon® プロセッサー用に現在最適化されているアプリケーションをインテル® Xeon Phi™ コプロセッサー用に最適化する場合の手法について簡単に説明します。

インテル® Xeon Phi™ コプロセッサーのパフォーマンスを解き放つ

TODD ROSENQUIST および SHANE STORY 16

任意のコンパイラーとともに使用でき、開発者がインテル® Xeon Phi™ コプロセッサーのパフォーマンスを引き出せるように支援する、インテル® MKL の自動オフロード (AO) モデルについて簡単に説明します。

OpenMP* アプリケーションのチューニング

ALEXEI ALEXANDROV 17

インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサーで実行するアプリケーションの並列性を向上させパフォーマンスを引き出すチューニング手法について説明します。

さまざまなデバッグオプション

NICOLAS BLANC および GEORG ZITZLSBERGER 23

さまざまなインテル® Xeon Phi™ コプロセッサー向けのデバッグ・ソリューションを紹介し、オフロードモデルとネイティブモデルについて説明します。

HTML5 を利用した並列実行

MAX DOMEIKA 25

HTML5 アプリケーションにおける並列パフォーマンスの向上をサポートする API について簡単に説明します。

© 2013 Intel Corporation.無断での引用、転載を禁じます。
Intel、インテル、Intel ロゴ、Intel Core、Core Inside、Intel Xeon Phi、Xeon、Xeon Inside、Cilk、VTune は、アメリカ合衆国および / または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。

ソフトウェアが切り拓く未来

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) 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』 (IntelPress,2005) などの文献を発表しています。

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。

今年最初のこの号では、ソフトウェアによる革新を支える手法、ツール、コーディング・モデルを紹介します。ソフトウェアが世界を変えているという事実は、今回の特集記事でお分かりになるでしょう。

この特集記事、「高速かつ正確な有限要素解析 (FEA)」では有限要素解析 (FEA) の概要を紹介し、NASA や F1 レースカーから医療用インプラントおよび環境分析まで、さまざまなプロダクト・エンジニアリングに革命をもたらした複雑な 3D シミュレーションの役割について解説します。

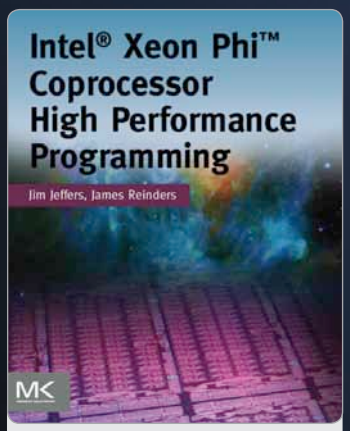
「パラレルパワー: インテル® Xeon Phi™ コプロセッサー向けソフトウェアの最適化」では、インテル® Xeon® プロセッサーのパフォーマンスを最大化して最新のインテル® Xeon Phi™ コプロセッサーを活用する単純な最適化手法を紹介します。関連記事である「インテル® Xeon Phi™ コプロセッサーのパフォーマンスを解き放つ」では、インテル® マス・カーネル・ライブラリー (インテル® MKL) による最適化のヒントとして、インテル® アーキテクチャーでアプリケーションのパフォーマンスを引き出す重要な要素について述べます。私はちょうど新しいインテル® Xeon Phi™ コプロセッサー向けのプログラミングに関する書籍 (詳細は **http://lotsofcores.com** を参照) を仕上げているときだったので、これらの記事から感銘を受けました。インテル® Xeon Phi™ コプロセッサーのプログラミングに興味のある方は、これらの記事と新しい書籍をお読みになることをお薦めします。

「OpenMP* アプリケーションのチューニング」では、並列 HPC アプリケーションに関する洞察を深めて、パフォーマンスを解析およびチューニングする手順を説明します。「さまざまなデバッグオプション」では、インテル® Xeon Phi™ コプロセッサーの使用事例におけるデバッグオプション、オフロードモデルおよびネイティブモデルについて紹介します。

「HTML5 を利用した並列実行」では、開発者に並列パフォーマンス向上の恩恵をもたらす、並列 HTML5 アプリケーション用のさまざまな API について解説します。

ソフトウェアの論理的な可能性が実現するのを見ることは、アプリケーションを次のレベルへ導く手法やツールについて探求する良い機会と言えるでしょう。この先、皆さんが何を実現するかを楽しみにしています。

James Reinders
2013 年 2 月

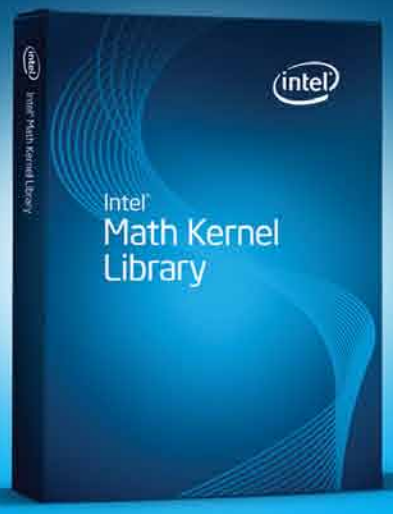


インテル® Xeon Phi™ コプロセッサーおよびインテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー向けのさまざまなプログラミングを解説

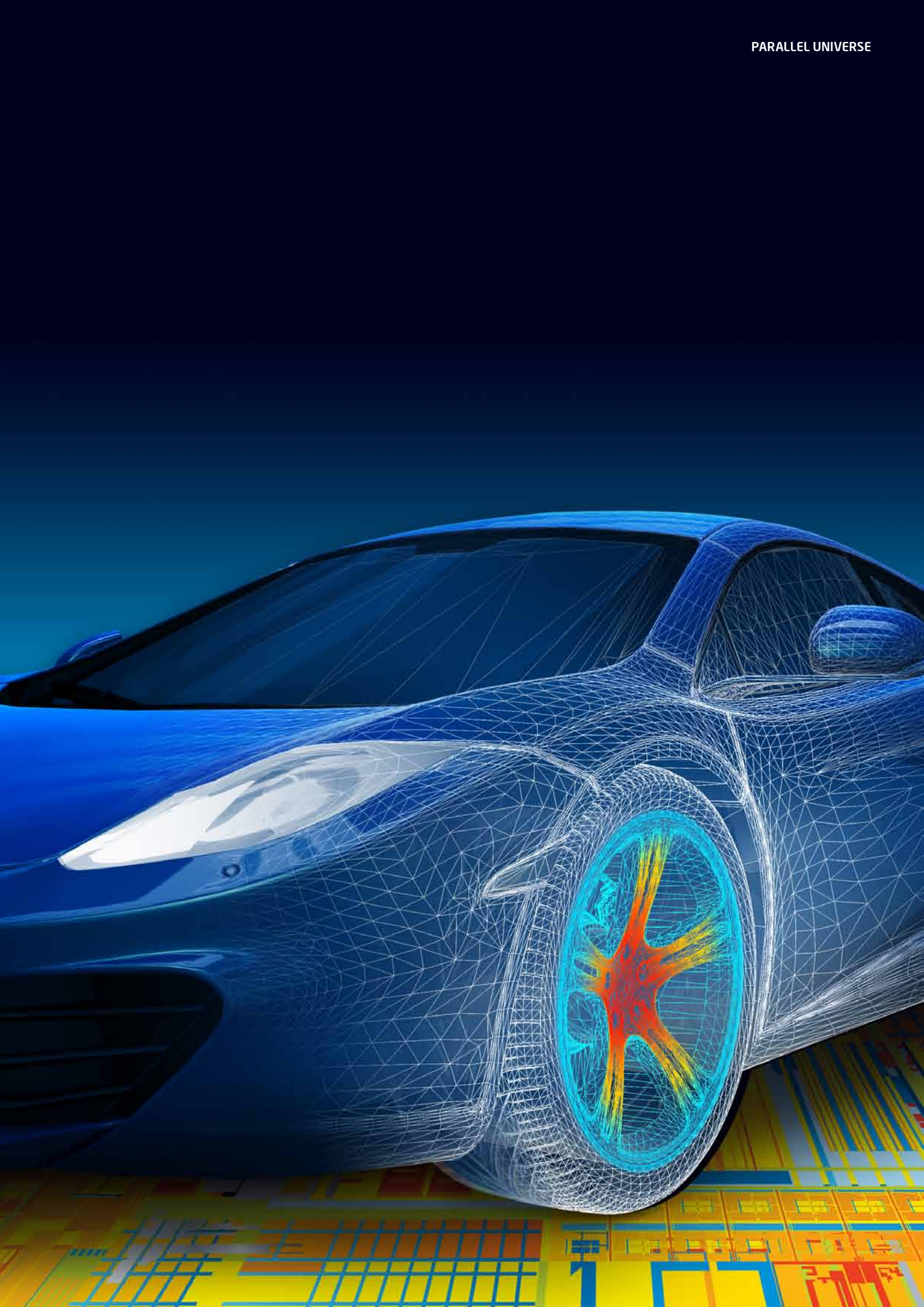
プロダクト・エンジニアリングの革命:

高速かつ正確な 有限要素解析 (FEA)

インテル コーポレーション
組込み計算、デバッガー、パフォーマンス・ライブラリー・グループ
テクニカル・コンサルティング・エンジニア
Noah Clemons、David Weinberg、Jonas Dalidd、Dennis Sieminski



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 を参照してください。



あらゆる種類のエンジニアリング製品は有限要素解析 (FEA) によって進化しています。FEA を利用することで、エンジニアは驚くほど綿密で柔軟性のある構造を検討することができます。この記事では、FEA の概要を紹介した後、解析者の生産性を損なうことなく解析の精度を向上させるために、インテル® マス・カーネル・ライブラリー (インテル® MKL) の高いパフォーマンスがどのように役立っているかを説明します。

最近のメディアでは、興味深いエンジニアリング・プロジェクトに関するニュースがよく取り上げられています。それは、民間人の宇宙旅行を可能にした宇宙船の進歩から、視力や聴力のような重要かつ敏感な身体機能を回復する医療用インプラントまで、非常に多岐にわたっています。スポーツの世界でも、ツール・ド・フランスの自転車、アメリカスカップのヨット、フォーミュラ 1 のレースカーなど、パフォーマンスの予想を変える新しい装備の設計が盛んに行われています。何がこのような広範囲にわたる洗練された技術能力をもたらしているのでしょうか。NEi Software 社のこれらの顧客ケーススタディー (www.NEiSoftware.com) は、有限要素解析 (FEA) として知られる計算手法が現代のエンジニアリング手法の向上に果たしてきた役割を示しています。FEA ソフトウェアは、プロダクトデザインの 3D モデルをバーチャルな環境でテストする手法をエンジニアに提供するツールです。エンジニアは、荷重、応力、衝撃、振動、熱および温度条件をアプリケーションでシミュレーションできます。数値データとグラフだけでなく、3D グラフィックスとアニメーションを効果的に使用して、仮想プロダクトがどのように反応するかを確認できます。応力、ひずみ、振動モードや温度勾配のような要素で生成される膨大なデータファイルをアニメーションや多彩なビジュアル・プレゼンテーションで表現することで、複雑な物理システムのパフォーマンスに関して包括的で技術的な理解を深めることができます。このようなデジタル・プロトタイプを使って得られる洞察には多くのメリットがあります。主なメリットの 1 つは、問題領域に集中できることです。変更がはるかに容易なデジタル設計段階で問題に取り組み、代案を試せるように設計して、満足できる結果が得られるまでさまざまな操作条件を徹底的に調査することができます。最初の部品やプロトタイプを製作する前にすべての変更を行えます。生産的で洞察力に富んだ設計プロセスにより、生産に入る前のプロトタイプ作成、物理的なテスト、設計反復の回数を減らすことができるため、製品開発サイクルの後のステップに大きな利点と節約がもたらされます。その結果、より低コストで質の高い製品を市場へ迅速に投入することが可能です。これらの利点から、FEA の導入がプロダクト・エンジニアリングの世界に深く広範囲な影響を与えたことは明らかです。

まず、FEA がどのように現在の段階まで達したかを簡単に見ておきましょう。FEA の過去を知ることは、テクノロジー開発の次の段階で直面する問題の背景を理解することにつながります。商用 FEA ソフトウェアの起源は、アメリカ航空宇宙局 (NASA) 向けに開発されたコードが一般向けにリリースされた 1970 年代初めまで遡ります。このプログラムは、NASA 構造解析 (NASA Structural Analysis) の頭文字から NASTRAN と呼ばれていました。コンピューター・リソースのコストとこのソフトウェアを使う専門の人員が必要であったことから、NASTRAN の利用は当初、高価で注目度の高いプロジェクトである、連邦政府により資金が提供された航空宇宙プログラムに限定されていました。その後 20 年以上にわたるハードウェアの向上、機能の拡張、コストの低下により、FEA ソフトウェアは航空宇宙、自動車、海運、核の分野でも一部利用されるようになりました。FEA ソフトウェアの利用が次に進んだのは、メインフレーム・コンピューターからの大幅な移行でした。NEi Software* 社が開発した PC 上で動作する NEi Nastran* (www.NEiNastran.com) はその一例です。FEA ソフトウェアが平均的な規模のエンジニアリング部門の手に届くようになったことで、従来の航空宇宙、自動車、海運業界でもさらに利用が広がりました。より重要な点は、土木、医療、コンシューマー、レクリエーションなど、非常に広範囲な分野でこのテクノロジーが支持されていることです。

精度を損なうことなく計算速度を向上させることは、今後も最も重要な課題です。テクノロジーが普及して広範囲に利用されるようになるには、コンピューターで FEA テストケースの結果が得られるまでに長い時間をかけることはできません。迅速に結果が得られる必要があります。以前と異なる点は、エンジニアが過去の成果を基にして、より複雑なシミュレーションに取り組んでいることです。モデルは、さらに規模が大きく詳細になっています。モデルで表現する物理的な現象は非線形で多岐にわたり、相互関係があります。また、化合物や記憶形状合金のような新素材の特性はより複雑になります。この予備知識があれば、FEA ソフトウェアの数学的構造とプログラミング構造のどこでどのように速度と精度の向上が達成されているかが分かるでしょう。

力学や物理学の問題を解くことは、指定された荷重の作用による機械的または物理的なシステムの動作を予測することを意味します。このような問題の解析は、幾何学的に単純な領域 (例えば、長方形や円) でのみ可能です。より複雑な領域に取り組むには、FEA のよう

図 1: セスナ 510 サイテーション・マスタングは NEi Nastran* により設計および解析されました。尾翼の上の色は、構造の応力レベルを示しています。スペクトルの赤は高い値を、青は低い値を表します。

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 を参照してください。

な数値離散化手法が必要です。その名前が意味するように、FEA は複雑な領域を有限数の小領域 (有限要素) に細分化して、各領域の物理的な問題を解きます。1 セットの有限要素で連続的な領域を表現する手法は、有限要素モデルと呼ばれます。モデルのすべての有限要素は有限要素メッシュを形成します。有限要素モデルの各要素には、荷重と境界条件を適用する要素の境界を定義した固定数の節点が配置されます。メッシュを細かくすると、構造の形状、荷重、および応力とひずみの勾配をより近似的に表現できます。ただし、メッシュが細かくなるほど問題が複雑になるため、より計算能力が必要になります。FEA の傾向として、より高い計算能力を必要とする、より大きく詳細で複雑なモデルには、高度なツール (インテル® MKL など) が FEA アプリケーションに組み込まれます。FEA は次の 3 つの段階で構成されます。

1. **前処理:** 解析者は構造の形状に基づいて有限要素メッシュを作成し、材料の特性、境界条件、荷重を適用します。

2. **解析:** FEA ソルバーが剛性行列と荷重ベクトルを組み立てて、変位、ひずみ、応力について解きます。変位の解析には多くの連立方程式を解く必要があるため、FEA アプリケーションで最も計算負荷が高くなります。

3. **後処理:** 解析者は通常、変形後の形状、コンタープロット、その他の視覚化されたグラフィックの形で結果を取得します。これは、解析の有効性の確認に役立ちます。

前処理段階と後処理段階は通常、Siemens Femap* や NEi Nastran* in-CAD などの FEA モデラーに依存します。Femap* は、CAD (コンピューター支援設計) アプリケーションとのインターフェイスを備える、スタンドアロンの FEA モデラーおよびポスト・プロセッサーです。(CAD は一般的に構造の形状を定義するツールです。) 一方、NEi Nastran* in-CAD は CAD プログラムに接続して動作する内部 CAD アプリケーションです。使い慣れた CAD プログラムから FEA モデルの構築、解析の実行、後処理を行うことができます。

解析段階では、NEi Nastran* のような FEA ソルバーを使用します。3 つの段階の中で最も計算負荷が高く、最もメモリーと CPU 処理時間が必要です。解析者は通常、最速かつ正確な解析を求めますが、高速であることは FEA のメッシュが粗くなる (精度が低くなる) ことを意味するため、これらの 2 つの要件は相反する傾向にあります。したがって、ここでの課題は両方を兼ね備えることです。FEA ソルバー内には、最適なパフォーマンスを実現するように最適化された専門のサブコンポーネントがあります。NEi Nastran* や他の多くの FEA ソルバーでは、重要なサブコンポーネントは線形方程式ソルバーと固有値/固有ベクトルソルバーです。最適なパフォーマンスを得るには、個々のサブコンポーネントが解析時間を左右すべきではありません。そのため、サブコンポーネントでメモリーとプロセッサー・アーキテクチャーを適切に利用することが重要です。NEi Nastran* は、インテル® MKL の PARDISO ソルバーと DGEMM 行列乗算ルーチンを用いて解析時間を短縮し、ソリューション・ボトルネックを回避しています。PARDISO ソルバーは、並列処理のスケーラビリティーに優れ、特定の

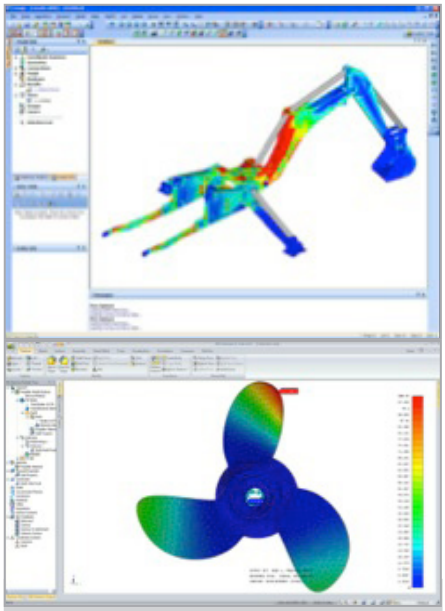


図 2: Femap* の前処理と後処理 (画面上) および NEi Nastran* in-CAD (画面下) のユーザー・インターフェイス。注: 例の応力レベルは色で表現されています。スペクトルの赤は高い値を、青は低い値を表します。

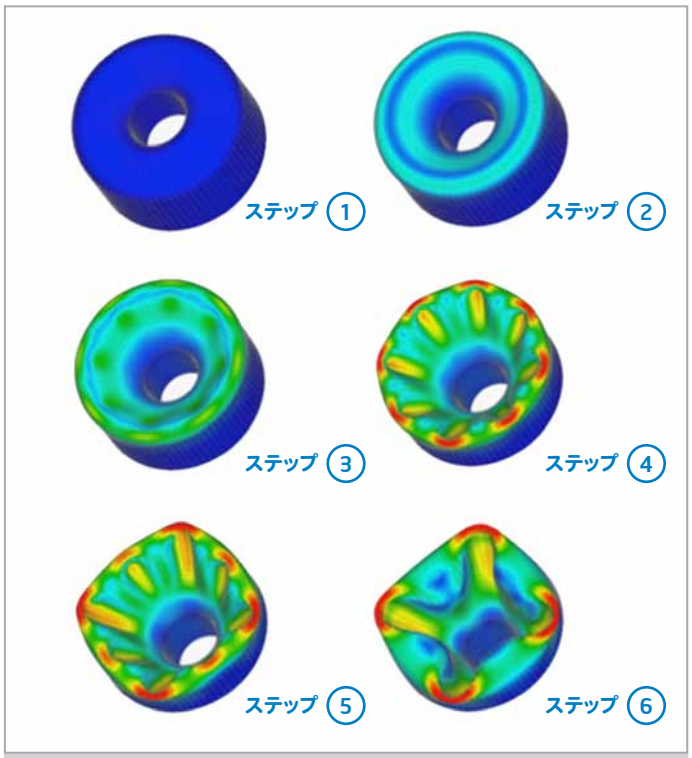


図 3: ゴム膜の非線形過渡解析。解析には、大規模な変位、回転、座屈後の動作が含まれます。番号は、荷重により変形される構造の部分を示しています。応力レベルは色で表現されています。この例のアニメーションおよびほかの例は、www.nenastran.com/fea/animations を参照してください。

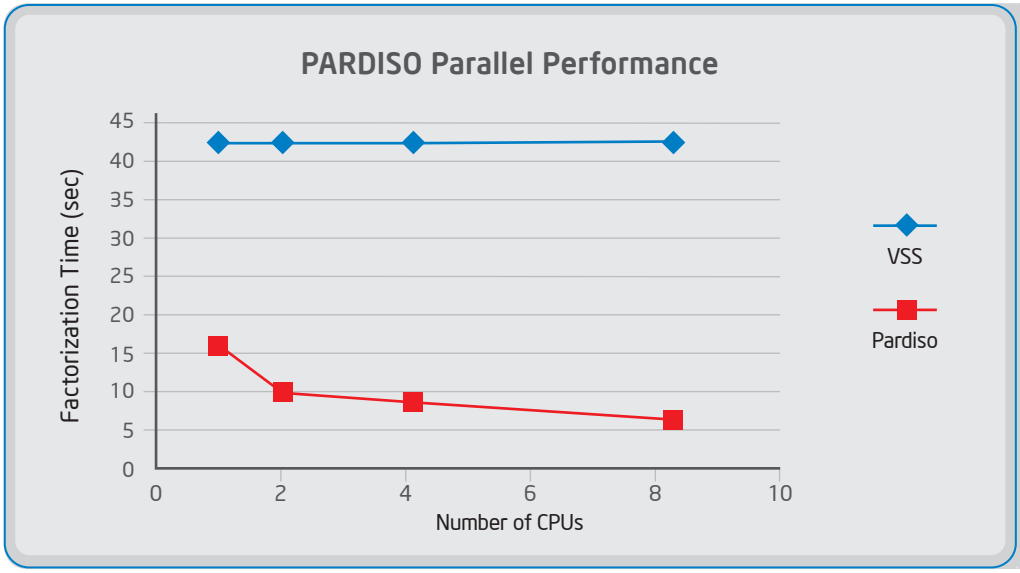


図 4: PARDISO ソルバーの並列スケーラビリティーを示すゴム膜解析の因数分解時間。典型的な非線形過渡解析では 1 回のシミュレーションごとに数百の因数分解を行います。(ハードウェア構成: 2 x インテル® Xeon® プロセッサー E5-2670 2.6GHz、60.5GB RAM、SAS; ソフトウェア: Windows Server* 2008 R2 64 ビット、インテル® MKL 11.0.1)

```
C
C ステータスを出力
C
          STR806 = 'REORDERING'
          CALL WRIT1STA( STR806)
C
          PHASE = 11
C
C 並び替えとシンボリック因数分解
C
          CALL PARDISO( PPOINTER, MAXFACTMATRIX, MATRIX, PMTYPE, PHASE,
     1                  NFDOF, A, LN, TLA, INTNUM, NRHS, INTPARAMETER,
     2                  MSGLVL, REALNUM, REALNUM, ERRORSTATUS)
          IF ( ERRORSTATUS .NE. 0 ) THEN
           IF ( (ERRORSTATUS .EQ. PSS_OOC_INSUFFICIENT_MEMORY) .OR.
     1          (ERRORSTATUS .EQ. PSS_INSUFFICIENT_MEMORY) ) THEN
            IF ( L .EQ. 1 ) THEN
             AMULTIBLOCK = .TRUE.
C
C スパースソルバーのメモリーを解放(割り当てられている場合)
C
             CALL FREEPSS
C
C コアモードを終了
C
             INTPARAMETER(60) = 2
C
C PSS 設定ファイルをリセット
C
             CALL SETPSSCF( 'RESET')
C
             CYCLE
            ELSE
             GOTO 307
            ENDIF
           ELSE
            GOTO 207
           ENDIF
          ENDIF
C
          SPARSEMEMALLOC = .TRUE.
C
C ステータスを出力
C
          WRITE (STR806, 1000) 0
          CALL WRIT1STA( STR806)
C
          PHASE = 22
C
C 数値因数分解
C
          CALL PARDISO( PPOINTER, MAXFACTMATRIX, MATRIX, PMTYPE, PHASE,
     1                  NFDOF, A, LN, TLA, INTNUM, NRHS, INTPARAMETER
     2                  MSGLVL, REALNUM, REALNUM, ERRORSTATUS)
C
```

図 5: PARDISO ソルバーを呼び出して因数分解を行います。インテル® MKL では、さまざまな方法で PARDISO ソルバーを呼び出せます。ここでは、最初に並び替えとシンボリック因数分解にソルバーを呼び出し、メモリー要件を決定しています (PHASE=11)。次に、最もメモリーと CPU を使うアクション因数分解にソルバーを呼び出しています (PHASE=22)。必要な場合はアウトオブコア・モードに戻るように処理が NEi Nastran* 内で行われるため、使用するメモリーは少なくなりますが、I/O が増えることでパフォーマンスは低下します。

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 を参照してください。

インテル® プロセッサー・アーキテクチャー向けにチューニングされているため、全体の解析時間を 100 分の 1 (場合によっては 1000 分の 1) の単位で大幅に短縮できます。PARDISO ソルバーの重要性は、完了までに数千の分解と後方解決を行う非線形の場合に大きくなります。PARDISO ソルバーは NEi Nastran* の非線形解析のデフォルトソルバーで、一般的には最も高速です。

PARDISO の実装は単純で、現代の FEA 問題を解くのに不可欠な疎行列形式 (非ゼロの要素のみ格納) を使用します。多くの FEA ソリューションでは大規模な疎行列を扱います。図 5 は典型的な PARDISO ソルバーの呼び出しです。ここでは、非常に大きな行列を物理メモリーに収める準備をします。

FEA 解析には、密行列とその乗算も含まれます。インテル® MKL は、

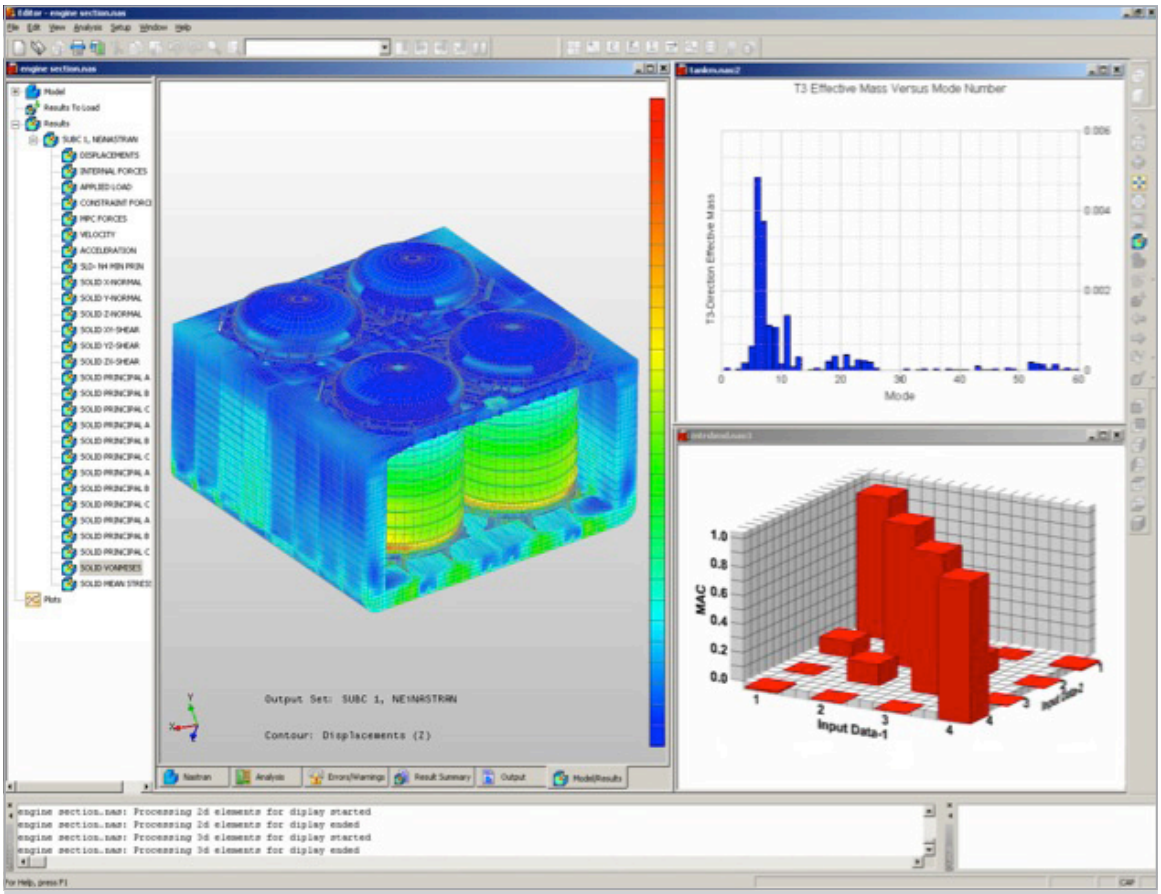


図 6: NEi Nastran* エディターで表示した船のセクションの応力コンター。
動的解析により、複数の方法で 2d プロットと 3d プロットを含む結果を得られます。

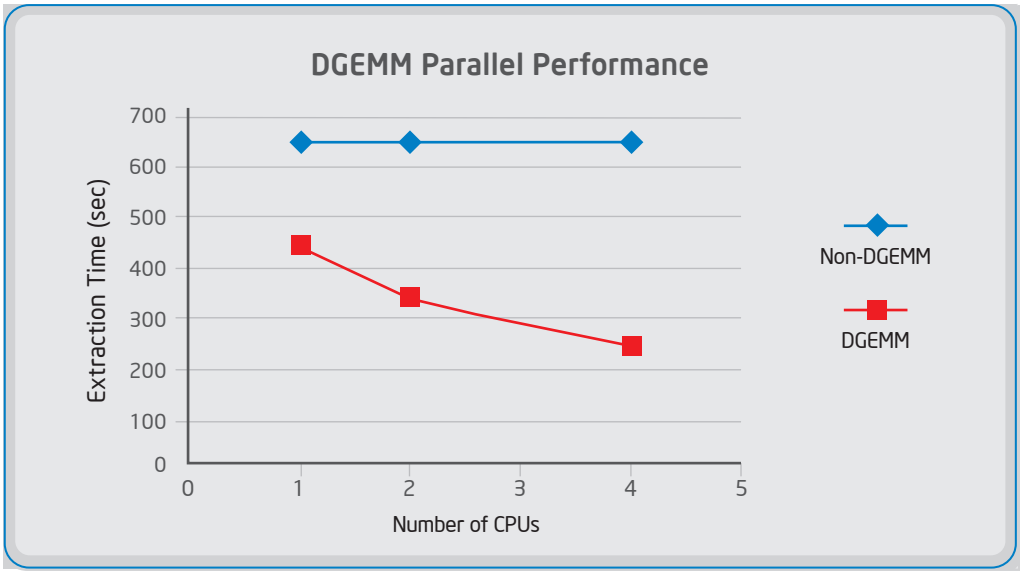


図 7: 船のセクションのモーダル抽出時間の比較。10 のモードを解析中に抽出。(ハードウェア構成: インテル® Core™ i7 860 2.8GHz、16GB RAM、Samsung* 830 シリーズ SSD; ソフトウェア: Windows* 7 SP1 64 ビット、インテル® MKL 11.0.1)

```
C
          STR801 = 'EXTRACTING EIGENVALUES FOR SUBCASE '
     1             //SUBCSTR(1:NCHARSUBC)//'  ITERATION '
     2             //ITERSTR(1:NCHARITER)
          STR802 = 'VECTOR:      1  PERCENT COMPLETE:    0'
          CALL WRIT2STA( STR801, STR802)
C
          IC = 0
C
C A と B の投影を計算
C
          CALL ASOLNEGS( UU, A, ADIAG, LA, NA, EIGVEC, NSOL, INCREMENT,
     1                   IC, INTERVAL, NC)
C
          CALL DGEMM( OPXT, OPXN, NC, NC, NN, ONE, EIGVEC, NDOF, UU, NDOF,
     1                ZERO, AR NC)
C
          DO J=1, NC
           DO K=1, NN
            EIGVEC(K, J) = UU(K, J)
           ENDDO
          ENDDO
C
          CALL AMULTEGS( UU, A, ADIAG, B, BDAIG, EIGVEC, LA, NA,
     1                   SHIFTFLAG, SHIFT, INCREMENT, IC, INTERVAL,
     2                   NC, 'B')
C
          CALL DGEMM( OPXT, OPXN, NC, NC, NN, ONE, EIGVEC, NDOF, UU, NDOF,
     1                ZERO, BR NC)
C
          IF ( .NOT.(CONVERGED) ) THEN
           DO J=1, NC
            DO K=1, NN
             EIGVEC(K, J) = UU(K, J)
            ENDDO
           ENDDO
          ENDIF
```

図 8: DGEMM 呼び出し。DGEMM は大規模な密行列の乗算向けに最適化されています。上記の部分空間固有値ソルバールーチンで、DGEMM はモーダルから物理的な空間に固有値ベクトルを拡張するのに使用されています。行列サイズの範囲は、10 ～ 1,000 x 10 ～ 1,000 の正方行列を 10 ～ 1,000 x 1,000 ～ 1,000,000 の矩形行列で乗算したものとなります。OPXT および OPXN は、行列や行列転置を乗算すべきかどうかを定義する変数です。NC および NN は行列の次元で、NN (モデルサイズ、自由度) は NC (決定する固有値の数) よりもはるかに大きくなります。

```
C
          PHASE = 33
C
C スパース前方/後方代入
C
          CALL PARDISO( PPOINTER, MAXFACTMATRIX, MATRIX, PMTYPE, PHASE,
     1                  NFDOF, A, LN, LA, INTNUM, NRHS, INTPARAMETER,
     2                  MSGLVL, S, X, ERRORSTATUS)
          DEALLOCATE (LN)
C
          IF ( ERRORSTATUS .NE. 0 ) THEN
           IF ( ERRORSTATUS .EQ. PSS_INSUFFICIENT_MEMORY ) THEN
            GOTO 307
           ELSE
            IOSTATUS = ERRORSTATUS
            GOTO 207
           ENDIF
          ENDIF
C
C X を分割解除
C
          CALL VSPR2GLB( X, V, LP)
```

図 9: 前方/後方代入と解析を行う PARDISO の呼び出し。図 5 で示されるように、行列が因数分解されると、さまざまな荷重ケースや右辺を繰り返し解くことができます。FEA (特に、多くの前方/後方置換呼び出しが必要な非線形解析) で行うと効果的です。

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 を参照してください。

大規模な密行列の乗算向けに高速かつスケーラブルな DGEMM ルーチンを提供します。PARDISO のように、このルーチンはインテル® プロセッサー・アーキテクチャー用に最適化されています。固有値解析は、動的な応答解析と同様に、構造の固有周波数とモード形状を決定するのによく行われます。

DGEMM の実装も単純です。図 8 は典型的な DGEMM 呼び出しです。ここでは、非常に大きな行列で乗算を行うことができます。この例では、サイズ NC x NDOF の 2 つの矩形行列を乗算してサイズ NC の 1 つの正方行列を形成しています。NC の範囲は通常 10 ～ 1,000 で、NDOF の範囲は 1,000 ～ 1,000,000 です。図 9 は、ASOLNEGS の呼び出しにも PARDISO ソルバーが使用されていることを示しています。その高速な後方解決のパフォーマンスにより、インテル® MKL は効率的に活用されています。

要約すると、FEA テクノロジーは驚くほど綿密かつ柔軟性のある構造を検討できる能力をエンジニアに提供します。エンジニアは、複雑な形状の構造を考慮し、非線形の特性を持つ新素材を利用した設計を調査して、風や地震のような環境的な要素を含む、さまざまな使用条件による影響を測定します。この作業には、高速かつ正確な解析が欠かせません。インテル® MKL の PARDISO および DGEMM ルーチンは、モデルサイズや複雑さがより増した場合でも非常に効果的に解析時間を短縮し、解析者の生産性を損なうことなく、解析の精度を向上させます。現在では、FEA テクノロジーが利用されていないエンジニアリング製品分野を探すのが困難なほどになっており、非常に高い品質と信頼性を備えた、新しいレベルの製品パフォーマンスの実現に寄与しています。

ブログを読む:
プロダクト・エンジニアリングの革命: NEi Nastran* を使用した高速かつ正確な有限要素解析 (英語)

「生産的で洞察力に富んだ設計プロセスにより、生産に入る前のプロトタイプ作成、物理的なテスト、設計反復の回数を減らすことができるため、製品開発サイクルの後のステップに大きな利点と節約がもたらされます。その結果、より低コストで質の高い製品を市場へ迅速に投入することが可能です。」

BLOG highlights

スケーリングしますか?

JACKSON M (インテル コーポレーション)

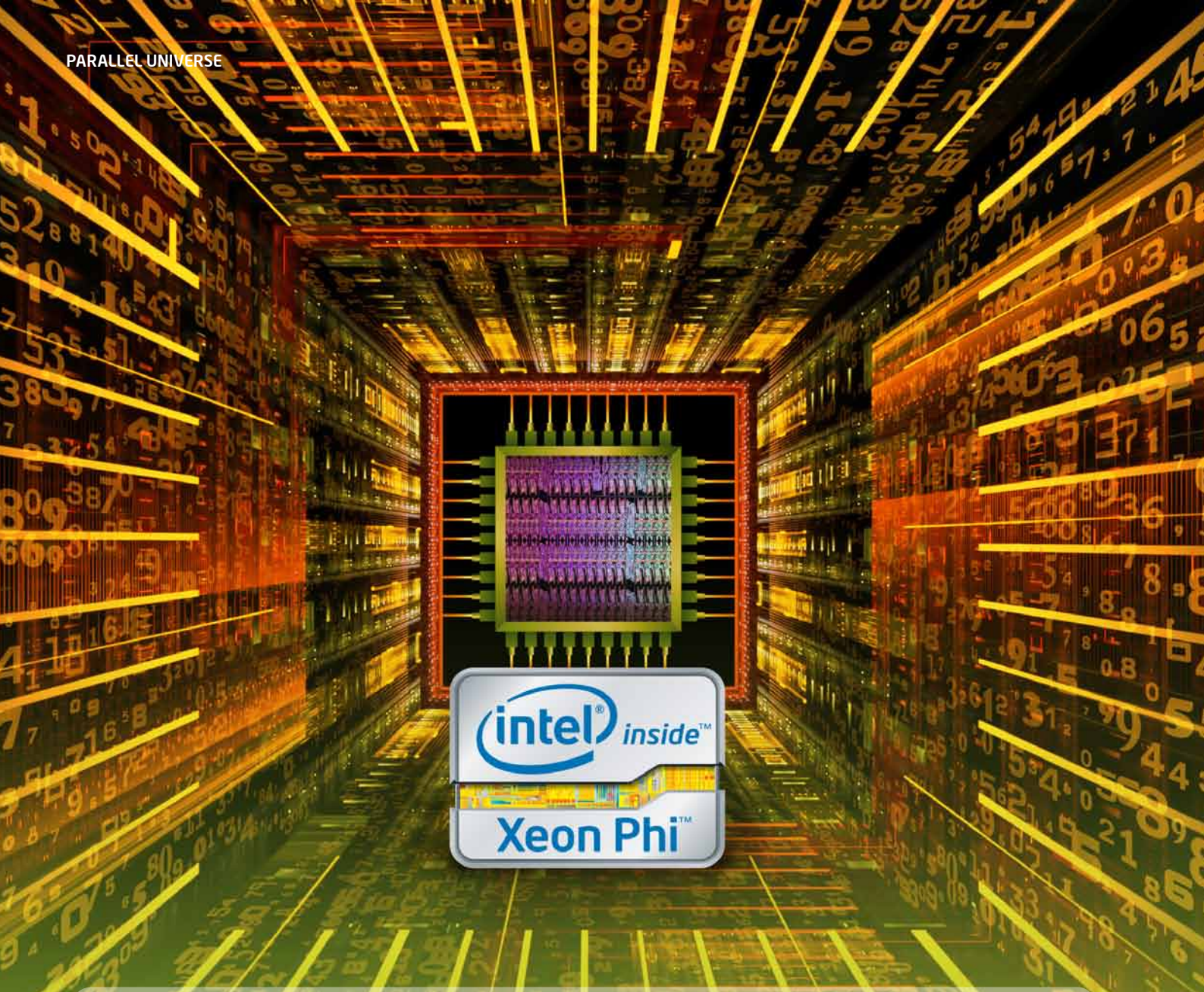
よくある話をご紹介しましょう。ある開発者は、シリアルコードにスレッドを実装するため懸命に作業しました。正当性テストにすべてパスして、アプリケーションは 2 コアのマシンでシリアルバージョンの約 2 倍で動作するようになりました。結果を見た知人が、インテル® ハイパースレッディング・テクノロジーを備えるクアッドコアのマシン (つまり、8 つの「論理」プロセッサーを利用可能) でそのプログラムを実行してみようと言い出しました。その開発者は、新しく並列化したアプリケーションは 2 コアのマシンの 4 倍近く高速に動くだろうと予想しました。しかし、結果は予想を裏切るものでした。アプリケーションは 2 コアのマシンとほぼ同じ速度で動作したのです。一体何が起こったのでしょうか。1 つ考えられるのは、スケーリング問題です。

ハードウェアでより多くのコアが利用可能なときに、並列化されたアプリケーションがそれらのコアを活用するように設計されていない場合、スケーリング問題が発生します。例えば、事前定義されたジョブをスレッドに割り当てるタスクレベルの並列化は、作成されるジョブの総数よりも多いコア数にスケーリングすることはできません。より多くのハードウェアを活用する作業に、これで十分という区切りはありません。

スケーリングする並列ソフトウェアの作成は、再設計に大きな労力を費やすことなく、ハードウェアの進化に伴って競争力の高さが維持されるアプリケーションの開発には欠かせません。インテル® Advisor XE を利用すると、コードのスレッド化に時間を費やす前に、新しい並列化ソリューションがより多くのコア数でもスケーリングするかどうかを検証できます。

この記事の続きはこちら (英語) でご覧になれます。

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)。



パラレルパワー:

インテル® Xeon Phi™ コプロセッサー向け

ソフトウェアの最適化

この記事では、インテル® Xeon® プロセッサー向けに現在最適化されているアプリケーションを、インテル® Xeon Phi™ コプロセッサー向けに最適化する手法について説明します。

インテル コーポレーション
HPC コンパイラー・サポート テクニカル・コンサルティング・エンジニア
Amanda Sharp

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 を参照してください。

概要

インテル® Xeon Phi™ コプロセッサーは、インテル® Xeon® プロセッサー・アーキテクチャー・ベースの高度に並列化されたプロセッサーで、コア数 (最大 61 コア) とベクトルエンジン (512 ビット幅) が拡張されています。両プロセッサーは、一般的なプログラム言語、コーディング手法、ソフトウェア・ツールを共有しています。インテル® Xeon® プロセッサーで動作するソフトウェアにすでに実装されている最適化は、インテル® Xeon Phi™ コプロセッサーで動作するアプリケーションにも適用できます。下記のステップに従うことで、アプリケーションのパフォーマンスをより向上させることができます。

コードの最適化ステップ

1. 適性の確認: 高度に並列化されたアーキテクチャーでアプリケーションが優れたパフォーマンスを実現する要件を満たしていることを確認します。
2. 実行モデルの選択: ホストシステムからコプロセッサーに計算を送るか、コプロセッサー上でアプリケーションを直接実行するかを選択します。
3. コードのチューニング: 並列化とベクトル化手法を用いて、インテル® Xeon Phi™ コプロセッサー向けにコードをチューニングします。

1. 適性の確認

高度に並列化されたアーキテクチャーをターゲットにする場合、アプリケーション全体またはアプリケーションの主要な部分がターゲット上で適切に動作するかどうかを検証する必要があります。インテル® Xeon Phi™ コプロセッサーのハードウェア・リソースを効率良く利用できるアプリケーションには、一般に次の特性があります。

- **最低 100 スレッドまでスケーラブルな高度な並列化アルゴリズム:** 各プロセッサー・コアは 4 つのハードウェア・スレッドをサポートします。実行時に、典型的なアプリケーションでは 200 を超えるアクティブなスレッドが動作します。適切な並列実装はプロセッサー・コアの最大数までほぼ線形にスケールします。
- **効率良くベクトル化された多くのコード:** 各プロセッサー・コアには、計算能力の主たるソースであるベクトル・プロセシング・ユニット (VPU) が備わっています。ベクトル化されたアプリケーションは、そうでないものより大幅にスピードアップします。

適切なアルゴリズムでは、最適化の後、実行時間の少なくとも 90 パーセントが並列化およびベクトル化されたコード部分で費やされます。アプリケーションがインテル® Xeon® プロセッサー向けに高度に並列化およびベクトル化されていて、これらの条件を満たしている場合、実行モデルを選択してコードをチューニングします。

2. 実行モデルの選択

インテル® Xeon Phi™ コプロセッサーは、コプロセッサーとして、あるいは独立したプロセッサーとしてプログラムできます。適切なモデルはアプリケーションとコンテキストに依存します。

ヘテロジニアス実行

ホストシステムがアプリケーションのスカラー部分を実行し、コプロセッサーに並列部分を任せた状態をヘテロジニアス (オフロード) 実行と呼びます。ホストとコプロセッサーはメモリーを共有しないため、データ交換はすべて PCI-E バス上で行われます。インテル® C/C++ コンパイラー/インテル® Fortran コンパイラーは、C++ プラグマおよび Fortran 宣言子としてオフロード用の言語拡張をサポートします。これには、データ転送と計算のオーバーラップを含む、データ割り当てとマーシャリングの広範囲なサポートも含まれます。インテル® C++ コンパイラーは、C++ アプリケーション用の仮想共有メモリーシステムを作成して管理する別のモデルもサポートします。

インテル® コンパイラーは、コード中のオフロード言語拡張を自動的に検出して、ホストとコプロセッサーの両方で実行するバイナリーを生成します。オフロード・プログラミングの構文に関する詳細は、「ヘテロジニアス・プログラミング・モデル」(英語) を参照してください。また、「オフロード用コンパイラー機能の効率的な利用」(英語) は、チューニングのヒントを含む、インテル® Xeon Phi™ コプロセッサーの効率良いプログラミングに関する包括的な記事です。

ネイティブ実行

アプリケーション全体をコプロセッサー上で実行している状態をネイティブ実行と呼びます。最適なパフォーマンスが得られるように、ネイティブ・アプリケーションのシリアル部分はできるだけ少なく、I/O の使用を制限し、コプロセッサーの物理メモリーよりも小さくなるようにします。インテル® Xeon Phi™ コプロセッサーは、Linux* ベースのオペレーティング・システムを実行します。インテル® コンパイラーの `-mmic` オプションを指定してコンパイルし、コプロセッサー用のバイナリーを生成します。セキュアシェル経由でコプロセッサーに接続し、コプロセッサーに必要なバイナリーをコピーして、アプリケーションを実行します。詳細は、「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」を参照してください。

3. コードのチューニング

インテル® Xeon Phi™ コプロセッサー向けコードのチューニングには多くの手法があります。パフォーマンスが向上する程度は、コードと使用する手法の数に依存します。

効率良い並列化

インテル® Xeon Phi™ コプロセッサーは、コアあたり 4 つのスレッド・コンテキストをサポートします。多くのコア数を指定して、並列リソースをすべて活用する効率良い並列化を実装することは、パフォーマンスを最大限にするための鍵です。通常はコアあたり少なくとも 2 つのスレッドを使用する必要があります。ここでは、OpenMP* ベースのコードの重要な推奨事項について説明します。

同期コストの削減

正当性を維持しつつ、コードの barrier 同期、ロック、クリティカル・セクションの使用を可能な限りなくすか減らします。可能な場合はリダクション操作を利用します。

ループ・スケジュール

デフォルトでは、ランタイム・ライブラリーは (一部のアプリケーションにとって理想的ではない) スタティック・ループ・スケジューリングを実装します。8 または 16 スレッドで適切に実行されるワークロードでは、多くのコア数でスタティック・ループ・スケジュールを指定すると、大幅なロード・インバランスが起こることがあります。この場合、異なるループ・スケジュール・アルゴリズムとチャンクサイズを指定してパフォーマンスを向上させます。場合によっては、OMP collapse 節が有効です。

スレッド・アフィニティーの制御

スレッド・アフィニティーはプログラムの実行速度に大きな影響を与えます。最適なアプリケーション・スレッドの数を決定してください。N-1 スレッドから 4 x (N-1) スレッドまで、異なるスレッド数を試します (N はコプロセッサーの物理コアの数)。OS とオフロードサービスを実行するコプロセッサー・コアでワーカースレッドのスケジュールを回避するには、最大 N-1 スレッド数を指定します。

OpenMP* パラメーターのデフォルト値は、ホストとコプロセッサー、およびオフロード実行とネイティブ実行では異なることに注意してください。詳細は、「OpenMP* を使用するための最適な手法」(英語) を参照してください。インテル® Composer XE は、インテル® Xeon Phi™ コプロセッサー向けのさまざまな並列化手法をサポートしています。詳細は、「OpenMP* スレッド・アフィニティーの制御」を参照してください。

ベクトル化

インテル® Xeon Phi™ コプロセッサーは、高度なデータ並列ワークロードを処理する幅広いベクトルユニットを備えています。下記の手法は、最適なパフォーマンスを得るために欠かせない、専用 VPU の利用を支援します。

データのアライメント

データ・アライメントはインテル® Xeon Phi™ コプロセッサーでは非常に重要です。適切なデータ・アライメントにすることで、データのロードおよびストアプロセスが合理化されます。次の 2 つのステップがあります。

1. データを 64 バイト境界でアライメントします。C/C++ の場合、静的配列には `__attribute__((aligned(64)))`、動的データの管理には `_mm_malloc()` および `cmm_free()` を使用します。Fortran の場合、`-align array64byte` コンパイラー・オプションを指定してコンパイルします。
2. コンパイラーがベクトル化されたコードを生成できるように、データがアライメントされていることを伝えます。1 つの方法は、ループの前にプラグマ/宣言子を挿入することです。C/C++ の場合は `#pragma vector aligned`、Fortran の場合は `!dir$ vector aligned` を追加します。

メモリー・アクセス・パターン

非ユニットストライド方式のメモリーアクセスは、ベクトル化が非効率的になり、パフォーマンスに大きな影響を及ぼします。内部ループの連続する反復で隣接していない位置からメモリーにアクセスすると、非ユニットストライド方式のメモリーアクセスが行われます。データ要素が異なるキャッシュラインからのものであると、キャッシュミスも発生します。この問題を回避するため、すべての hotspot でユニットストライド方式のメモリーアクセスを最大限にする、ベクトル化に適したデータ構造とアルゴリズムを採用してください。場合によっては、ベクトルの配列表記を利用すると効果的です。「構造体配列」から「配列構造体」へのデータレイアウトの変更が必要になることもあります。このトピックの詳細は、「メモリーレイアウト変換」を参照してください。

実行の指定

SIMD プラグマ/宣言子 (`#pragma simd` または `!dir$ simd` に適切な節を追加) は、ループをベクトル化するようコンパイラーに伝える便利な機能です。デフォルトでは、コンパイラーは入れ子のループ構造の最内ループをベクトル化しようと試みます。しかし、外部ループに多くのワークが含まれている場合、要素関数、ストリップマイニング、プラグマ/宣言子 SIMD を組み合わせて、外部ループをベクトル化できます。詳細は、「ベクトル化および最適化レポート」を参照してください。

高度な最適化

データ・プリフェッチやストリーミング・ストアを利用する高度な最適化は、アプリケーションのパフォーマンス向上に役立ちます。パフォーマンスと精度のトレードオフを考慮しつつ適切な浮動小数点オプションを指定することで、パフォーマンスを大幅に高められます。これらのトピックの詳細は、「インテル® MIC アーキテクチャー向けの高度な最適化」を参照してください。

まとめ

高度に並列化されたアーキテクチャーをターゲットとするアプリケーションは、広範なハードウェア・リソースを利用できます。アプリケーションがインテル® Xeon® プロセッサー向けに最適化されていれば、簡単なコード最適化手法に従うことで、インテル® Xeon Phi™ コプロセッサー上のパフォーマンスを向上させることができます。まず、高度に並列化されたアーキテクチャー上でアプリケーションを実行することで恩恵が得られることを確認します。次に、アプリケーションに最適な実行モデルを選択し、推奨する最適化手法でコードをチューニングします。高度な手法については、コードサンプルを含む記事、「インテル® MIC アーキテクチャー向けのコンパイラー手法」(英語)を参照してください。詳細は、インテル® デベロッパー・ゾーンのインテル® Xeon Phi™ コプロセッサー (英語) をご覧ください。

「高度に並列化されたアーキテクチャーをターゲットとするアプリケーションは、広範なハードウェア・リソースを利用できます。アプリケーションがインテル® Xeon® プロセッサー向けに最適化されていれば、簡単なコード最適化手法に従うことで、インテル® Xeon Phi™ コプロセッサー上のパフォーマンスを向上させることができます。」

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 を参照してください。

インテル® Xeon Phi™ コプロセッサーのパフォーマンスを解き放つ

インテル コーポレーション
インテル® マス・カーネル・ライブラリー テクニカル・コンサルティング・マネージャー
Todd Rosenquist
インテル コーポレーション
インテル® MKL テクニカル・ストラテジー マネージャー
Shane Story

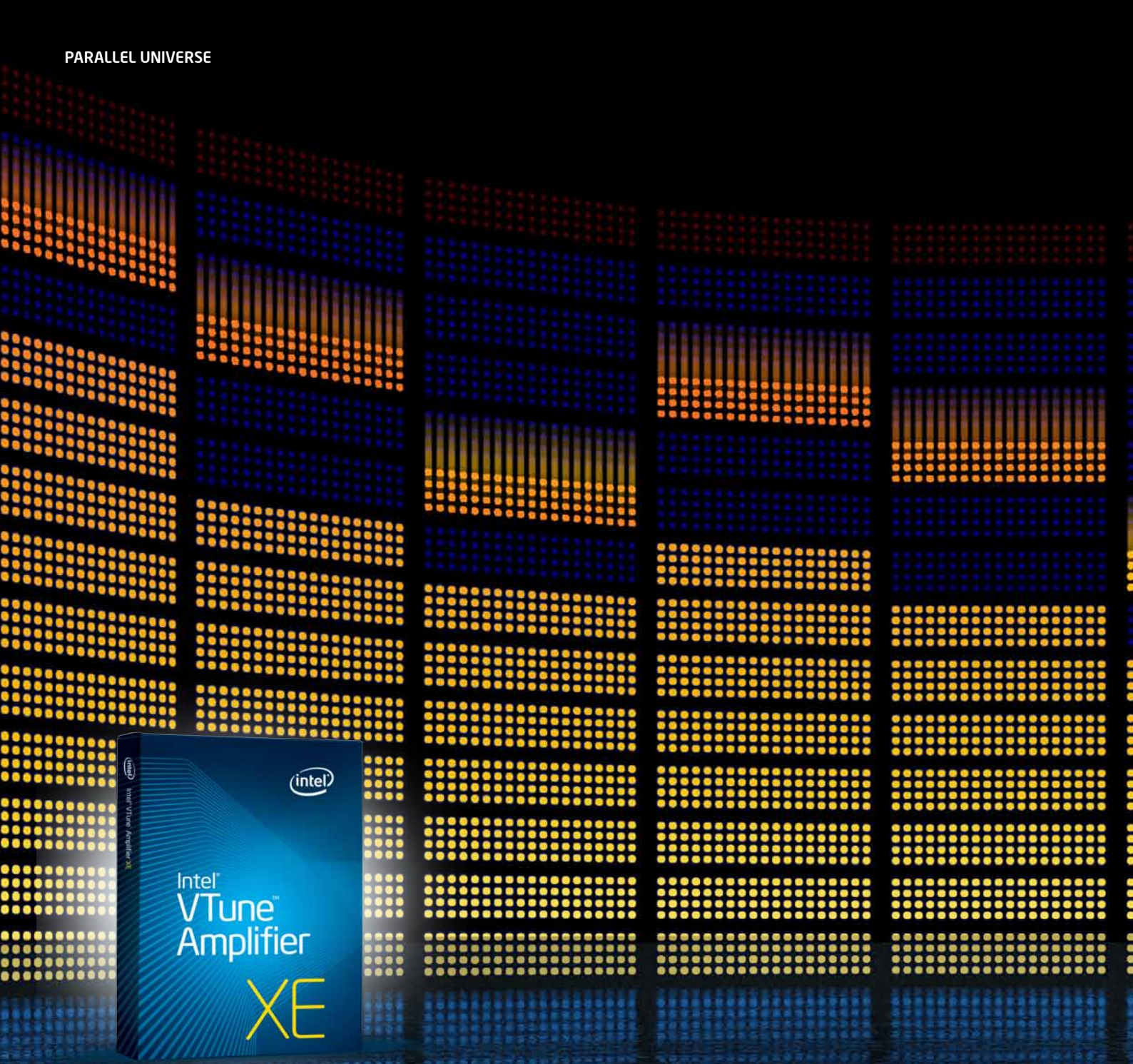
インテル® マス・カーネル・ライブラリー (インテル® MKL) を使用すると、ユーザーはインテル® Xeon Phi™ コプロセッサーを含む、新しいプロセッサーで提供されるパフォーマンスの利点を容易に活用することができます。インテル® MKL を使ったことがないユーザーは、自身のコードをチューニングするよりも、最適化されたこのカーネルを利用できる可能性を探ってください。インテル® MKL をすでに使っているユーザーは、インテル® Xeon Phi™ コプロセッサーにネイティブに最適化された関数のみならず、システムのすべてのプロセッサーとコプロセッサーを自動的に検出して最適に活用する自動オフロード (AO) 機能も利用できます。

インテル® MKL の AO モードは簡単に使用できます。現在と同じようにインテル® MKL をリンクして、プログラムで `mkl_mic_enable()` を呼び出すか、環境変数で `MKL_MIC_ENABLE=1` を設定します。大きな計算データ比率でインテル® MKL の線形代数関数を呼び出すと、利用可能なハードウェア・リソースが内部的に評価され、各リソースに計算ワークが自動的に配分されます。これは、必要なデータ転送がすべて透過的に行われることを意味します。マルチコア・プロセッサーでの実行に使われる関数は、インテル® Xeon Phi™ コプロセッサーの拡張された計算能力の恩恵を自動的に受けます。インテル® MKL の AO モードは、任意のコンパイラーで使用することができます。

インテル® MKL のネイティブにチューニングされた関数を利用する方法はいくつかあります。インテル® コンパイラーでサポートされているプラグマを指定して、これらの関数をコプロセッサーにオフロードするプログラムを記述することができます (コンパイラーによるオフロード支援 (CAO))。または、前の記事で説明されているように、コプロセッサーにログインして、プログラム全体をコンパイルして実行することもできます。

インテル® Xeon Phi™ が登場した後も、インテル® MKL が実用アプリケーションでインテル® アーキテクチャーのパフォーマンスを引き出す重要なツールであることに変わりはありません。インテル® MKL は、世界の高速なコンピューター (Top500) の性能指標として用いられるハイパフォーマンス LINPACK ベンチマークに優れたパフォーマンスを提供します。

インテル® Xeon Phi™ コプロセッサー向けプログラミングの詳細は、http://software.intel.com/en-us/articles/intel-mkl-on-the-intel-xeon-phi-coprocessors (英語) を参照してください。



OpenMP* アプリケーションのチューニング

インテル コーポレーション
シニア・ソフトウェア・デベロッパー
Alexei Alexandrov

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 を参照してください。

シニア・ソフトウェア・
デベロッパー
ALEXEI ALEXANDROV

ハイパフォーマンス・コンピューティング (HPC) は長い歴史を持ち、ビジネス、研究、科学に欠くことのできない存在となっています。数千のマシンで構成されるクラスターは、理論および実用の両面で現代科学の多くの進化を支援し、世界の人々の生活を豊かにするため休むことなく働いています。

HPC の並列化は、プロセス、スレッド、SIMD ベクトル化 (インテル® コンパイラーの自動ベクトル化を含む) の 3 つのレベルで行われます 。プロセスレベルの並列化では、メッセージ・パッシング・インターフェイス (MPI) が HPC 業界における事実上の標準となっています。スレッドレベルの並列化では、OpenMP* プログラミング・モデルが HPC コミュニティーで普及していますが、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) やインテル® Cilk™ Plus のようなほかのモデルも支持を広げています。インテル® VTune™ Amplifier XE は、これらのパラダイムをすべてサポートしていますが、この記事では OpenMP* による HPC プログラムのプロファイルに役立つ手法に注目します。

ここで紹介する例はインテル® Xeon Phi™ コプロセッサー上で実行する OpenMP* アプリケーションから集めたものですが、その手法は、すべての OpenMP* プログラムのチューニングに利用できます。例では、インテル® VTune™ Amplifier のコマンドラインを使ってデータを収集していますが (インテル® Xeon Phi™ コプロセッサー・システムでデータを収集する最も一般的な方法)、すべてのオプションを知らなくても心配ありません。解析のセットアップおよびコマンドラインの生成には、グラフィカル・ユーザー・インターフェイス (GUI) が一般的に使用されます。

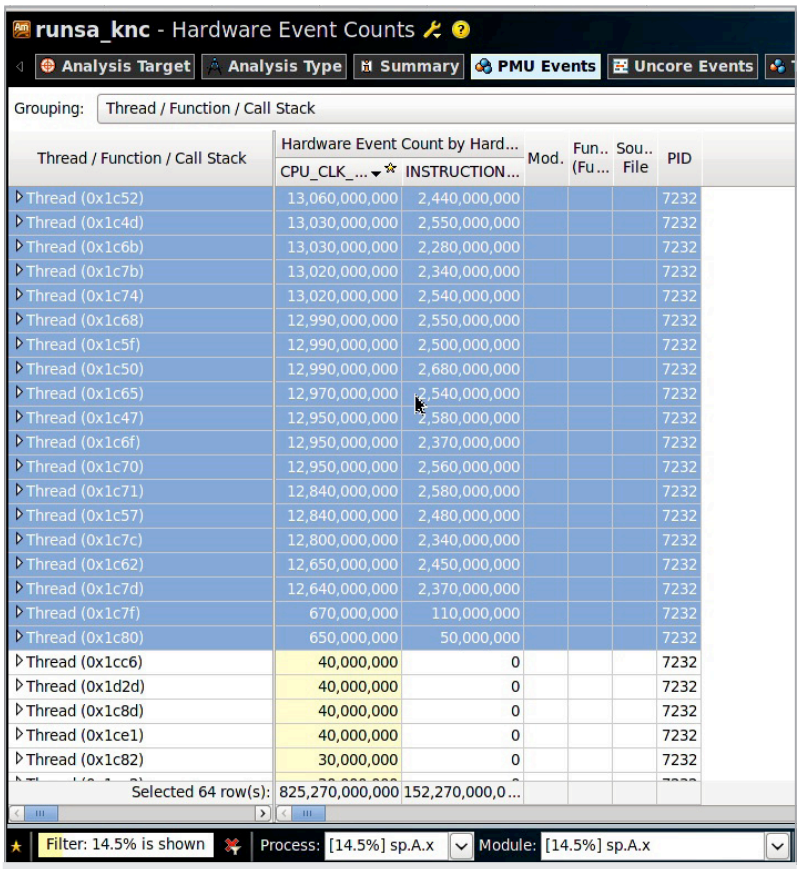


図 1: インテル® VTune™ Amplifier XE のグループ化とフィルターを利用して OpenMP* プログラムのバランスを解析します。

インテル® VTune™ Amplifier XE による OpenMP* プログラムのプロファイル

OpenMP* プログラムのパフォーマンス・データを収集して解析する簡単なシナリオから始めましょう。次の 3 つのステップがあります。

1. コマンドライン・インターフェイスからデータを収集します。
2. フィルターやループ解析のような機能を用いて、対話式の GUI でデータを解析します。
3. インテル® VTune™ Amplifier のコマンドライン・インターフェイスで hotspot のプロファイルを生成します。このステップは、パフォーマンス回帰テストの自動化や別のプログラムにデータを渡すときに役立ちます。

```
$ amplxe-cl -collect hotspots --
 ~/sp.A.x
```

ステップ 1 では、インテル® Xeon® プロセッサー・ベースのコンピューターで OpenMP* を使用するプログラムをプロファイルするため、サポートされている任意の解析タイプを用いてデータを収集します。

```
$ amplxe-cl -c knc-lightweight-
hotspots  -search-dir all:p=/lib/
firmware/mic -- ssh mic0 ~/sp.A.x
```

あるいは ハードウェア・イベントベースのサンプリング解析タイプを用いて、インテル® Xeon Phi™ コプロセッサーで実行されたプログラムをプロファイルします。

結果の後処理中にホスト上でインテル® Xeon Phi™ ランタイムバイナリーが見つかるように検索ディレクトリーを指定します。

ステップ 2 では、収集した結果をインテル® VTune™ Amplifier XE GUI で開き、解析を開始します。OpenMP* プログラムの場合、ボトムアップ・ビューでスレッド別にデータをグループ化してデータを確認すると良いでしょう。これには、グリッドの上の Grouping コンボボックスでスレッドベースのグループ化を選択します (図 1)。グリッドで複数の行を選択すると、選択しているスレッドの数とサマリー統計が表示されます。この統計は、スレッドの動作を理解する上で役立ちます。選択した項目を右クリックして、選択したデータをフィルターすることもできます。グループ化とフィルターを組み合わせると、必要なデータを細かく分けて解析することができます。フィルター機能は、OpenMP* ランタイムライブラリーで費やさ

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 を参照してください。

れた時間をフィルターして全体のパフォーマンスにおけるユーザーコードの純粋な貢献度を確認するときに役立ちます。例えば、図 1 は、CPU アフィニティーと実行している OpenMP* スレッドの数が最適でないため、ユーザーモジュールは CPU サイクルの 14.5 パーセントしか貢献せず、残りのサイクルはほかのモジュールで費やされていることを示しています。

上記のケースでは、スレッドをグループ化することで (61 コアのインテル® Xeon Phi™ コプロセッサーでは 244 のハードウェア ・スレッドが利用可能であるにもかかわらず)、64 のソフトウェア・スレッドのみでユーザーコードが効率良く実行されていることが判明しました。プログラムのソースコードを確認したところ、入力問題サイズの 1 次元で並列処理が制限されていることが明らかになりました。この問題を修正するには、利用可能な並列処理が増えるようにプログラムを変更します。あるいは、少なくとも、アフィニティーと使用する OpenMP* スレッドの数がワークロードの特性と一致するように設定します。KMP_AFFINITY を balanced、OMP_NUM_THREADS を 64 に設定すると、実行時間が向上し、スレッド間のバランスが良くなります。インテルの OpenMP* ライブラリーは待機の代わりにスピンを積極的に使用するため、ユーザーモジュールの外部 (および通常は OpenMP* モジュール) で費やされる CPU 時間はインバランスまたは過度のシリアル実行として示されます。コアによるグループ化を選択して、ハードウェア・コアの点からプログラムのバランスを確認することもできます。このグループ化は、同時マルチスレッディング (SMT) 並列性のパフォーマンスを確認するのに便利です (図 2)。

高レベルのバランス問題とオーバーヘッド問題を識別して修正した後、関数レベルの解析を行います (図 3)。インテル® コンパイラーとインテルの OpenMP* 実装を利用すると、OpenMP* 領域本体は compute_rhs_ompparallel@17 のような名前の擬似関数に集約されるため、領域本体の内部と外部で費やされた時間をより簡単に区別することができます。例えば、これは「関数 compute_rhs の行 17 の OpenMP* 並列領域」と読みます。

ホットな関数が分かれば、その C、C++、Fortran ソースとアセンブリーを調べて、最も時間がかかっているソース行とコンパイラーが生成したアセンブリー・コードを識別できます。後者を理解することで、内部ループをベクトル化するようにコンパイラーに指示すべきかどうかを判断できます。バージョン 3.0 から、インテル® VTune™ Amplifier XE はループの点からプログラムの構造を確認できるようになりました (ループを多用している HPC コードに役立ちます)。このモードを有効にするには、GUI フィルターバーの “Loop Mode” を “Loops and functions” に変更します (または、amplxe-cl コマンドライン・インターフェイスで “-loop-mode=loop-and-function” を指定します)。トップダウン・ビューではプログラムのループ構造が表示されます (図 4)。この例では、OpenMP* 領域の行 295 にループがあり、行 296 にその入れ子のループがあり、行 297 にさらに入れ子のループがある (後者のループはベクトル化プロセスの一部としてコンパイラーによりピールされている) ことが一目で分かります。行 297 に 2 つのループがあることは、2 つのループのインスタンスが同じソース行に属していることから識別できます。2 つ目のインスタンスは反復数の少ない残りのループであるため、1 つ目のインスタンスのほうが時間が長くなっていることに注意してください。

Grouping: Core / Thread / H/W Context / Function / Call Stack

Core / Thread / H/W Context / Function / Call Stack	Hardware Event Count by H	
	CPU_CLK_U...	INSTRUCTIO...
core_0	53,900,000,000	7,440,000,000
core_5	51,320,000,000	7,360,000,000
core_2	51,120,000,000	7,400,000,000
core_4	51,120,000,000	7,320,000,000
core_1	50,920,000,000	7,540,000,000
core_3	50,720,000,000	7,480,000,000
core_15	28,360,000,000	4,940,000,000
core_22	28,180,000,000	4,980,000,000
core_7	28,080,000,000	5,200,000,000
core_19	28,020,000,000	4,900,000,000
core_13	28,020,000,000	5,140,000,000
core_25	27,820,000,000	4,960,000,000
core_17	27,760,000,000	5,020,000,000
core_9	27,740,000,000	5,080,000,000
core_23	27,680,000,000	5,200,000,000
core_24	27,600,000,000	4,860,000,000
core_18	27,540,000,000	5,020,000,000
core_21	27,500,000,000	5,080,000,000
core_8	27,480,000,000	4,900,000,000
core_11	27,440,000,000	4,820,000,000
core_10	27,440,000,000	4,880,000,000
core_6	27,380,000,000	4,700,000,000
core_12	27,360,000,000	4,940,000,000
Selected 1 row(s):	53,900,000,000	7,440,000,000

図 2: コアによるグループ化でハードウェア・コアのバランスを確認します。

「高度に並列化されたアーキテクチャーをターゲットとするアプリケーションは、広範なハードウェア・リソースを利用できます。アプリケーションがインテル® Xeon® プロセッサー向けに最適化されていれば、簡単なコード最適化手法に従うことで、インテル® Xeon Phi™ コプロセッサー上のパフォーマンスを向上させることができます。」

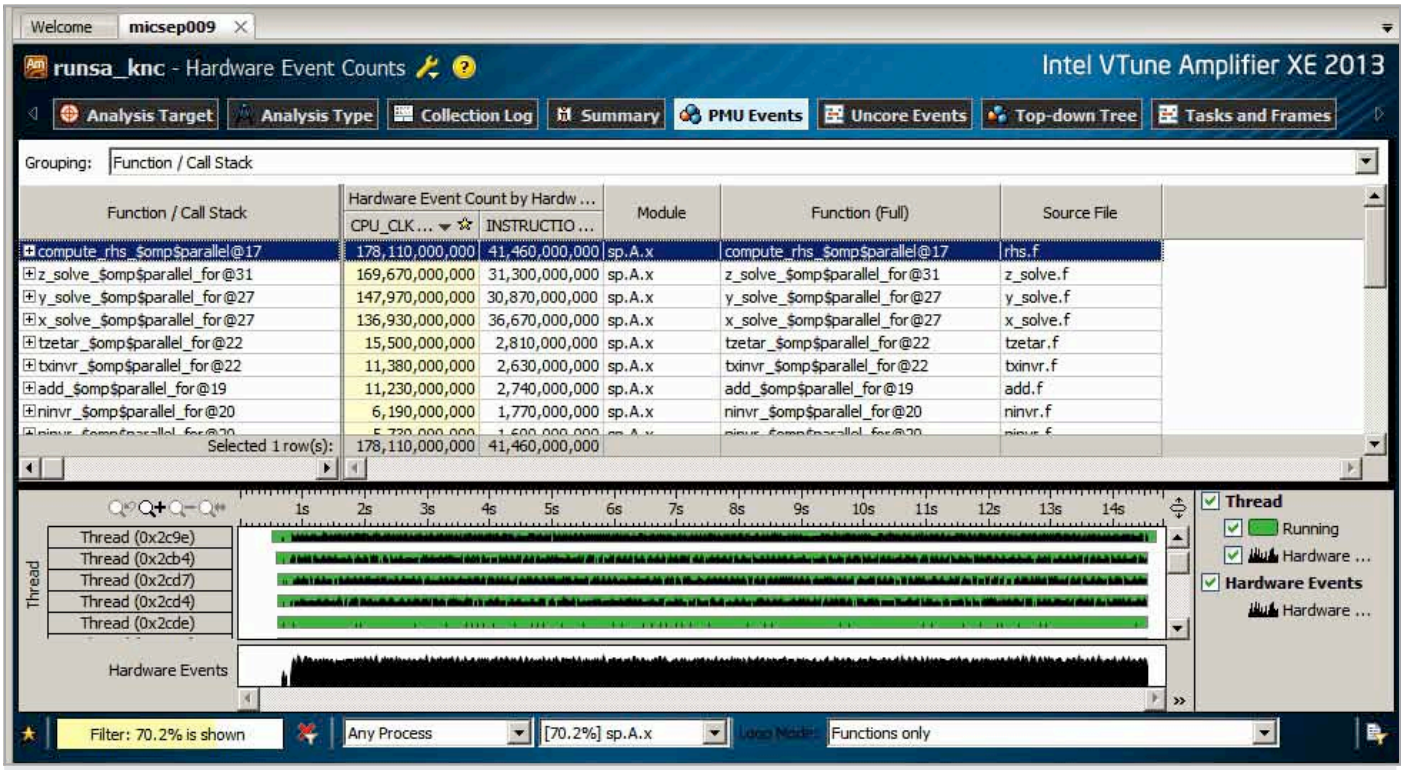


図 3: 関数レベルの hotspot ビュー

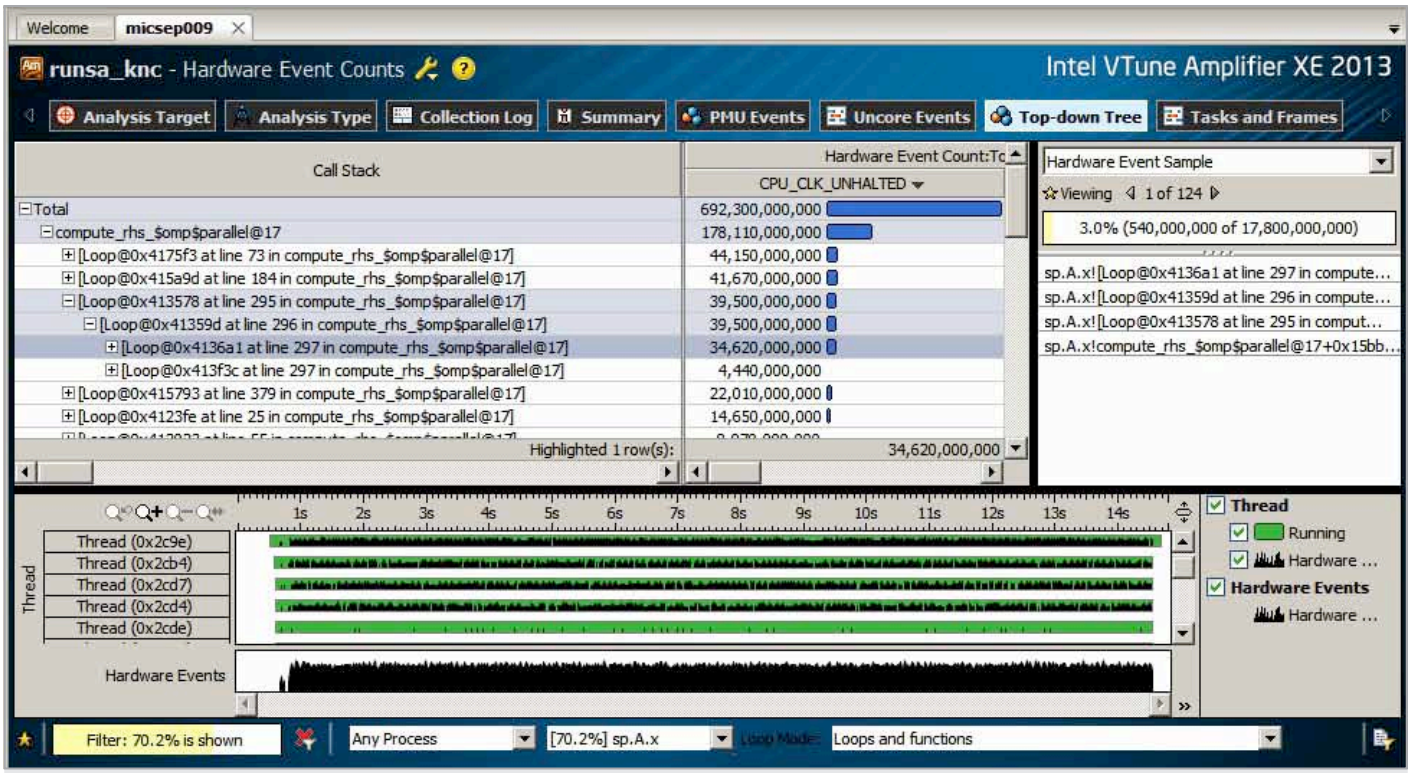


図 4: ループ解析でプログラムのループ構造を確認します。

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 を参照してください。

タイムライン・ビューは、特定の時間領域にデータの範囲を絞り込みます。例えば、プログラムのスタートアップの「コールド」フェーズをスキップしたり、特定の領域の実行にデータを拡大します。後者の場合、インテル® VTune™ Amplifier でサポートされている ITT API を使ってプログラムの実行をマークすると便利です。多くのスレッドを実行した結果を表示する場合は、“Tiny” モードのタイムライン・バンドを利用すると、より多くのデータが画面に表示されます。

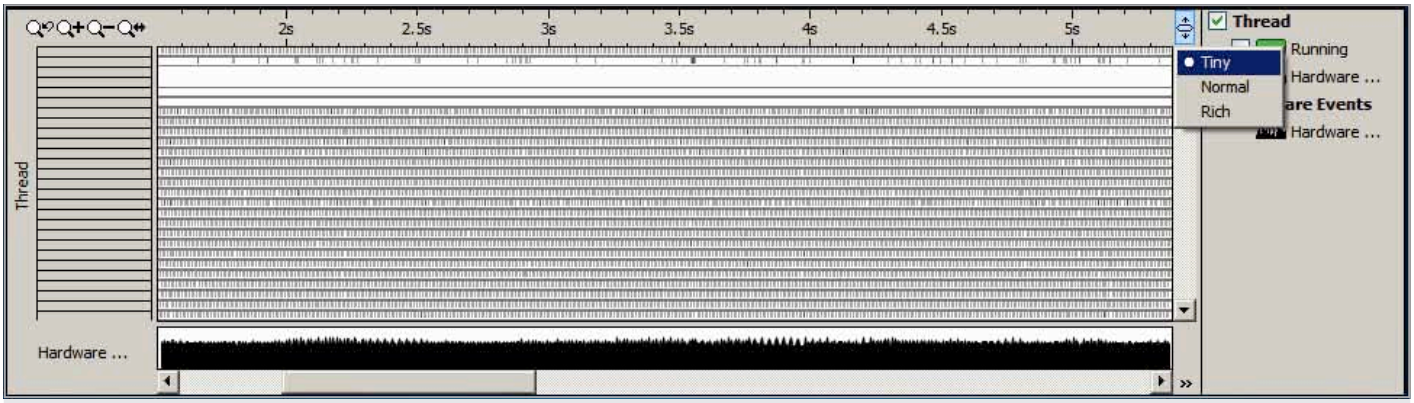


図 5: Tiny モードのタイムライン・バンドを利用すると、より多くのデータが画面に表示されます。

最後に、簡略化された最適化フローの**ステップ 3** では、コマンドライン・インターフェイスを使って hotspot の抽出を自動化すると便利です (パフォーマンス回帰テストの自動化など)。上記で説明した機能の多くは、コマンドラインから使用できます。例えば、次のコマンドラインは、スレッドと関数でグループ化し、特定のモジュールと時間領域でフィルターして、ループ解析を含む、上位 10 項目の hotspot を出力しています。

```
$ amplxe-cl.exe -R hotspots -loop-mode=loop-and-function -filter module=sp.A.x -limit=10 -group-by
thread,function -r <結果ディレクトリーのパス>

Thread            Function                                                        Module  CPU Time:Self
 ---------------  ---------------------------------------------------------------  ------  -------------
Thread (0x2cd3)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.394
Thread (0x2cd5)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.385
Thread (0x2cb5)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.376
Thread (0x2cd7)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.376
Thread (0x2cc1)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.358
Thread (0x2ccb)  [Loop@0x4200d8 at line 294 in x_solve_$omp$parallel_for@27]  sp.A.x          0.358
Thread (0x2ca2)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.349
Thread (0x2ccf)  [Loop@0x4200d8 at line 294 in x_solve_$omp$parallel_for@27]  sp.A.x          0.339
Thread (0x2cd2)  [Loop@0x4200d8 at line 294 in x_solve_$omp$parallel_for@27]  sp.A.x          0.339
Thread (0x2cab)  [Loop@0x4136a1 at line 297 in compute_rhs_$omp$parallel@17]  sp.A.x          0.330
```

図 6

まとめ

ここで説明したインテル® VTune™ Amplifier XE の手法は、HPC プログラムのパフォーマンス解析を始める上で役立つ情報として紹介したものですが、この他にも多くの情報があります。製品のドキュメントとオンラインリソース (インテルのナレッジベースを参照) では、MPI プログラムの解析、ループとインライン関数の解析、ITT API の使用方法、コマンドライン・レポートによるパフォーマンス解析の自動化など、製品の機能に関する詳細な情報を掲載しています。手法の説明にはインテル® Xeon Phi™ コプロセッサーを取り上げましたが、インテル® Xeon® プロセッサー・システムにも同じように適用できます。つまり、アプリケーションをインテル® Xeon® プロセッサーとインテル® Xeon Phi™ コプロセッサーの両方で実行する場合、アプリケーションの並列性が向上するチューニングを 1 つ見つければ、通常は両方のパフォーマンスを向上させることが可能です。

さまざまな デバッグオプション

インテル コーポレーション ソフトウェア・エンジニア
Nicolas Blanc
インテル コーポレーション テクニカル・コンサルティング・エンジニア
Georg Zitzlsberger

概要

インテル® Xeon Phi™ コプロセッサー向けアプリケーションの実行は通常、ホストと 1 つ以上のコプロセッサー間で分散されます。一般に、インテル® Xeon Phi™ コプロセッサー・ベースのアプリケーションには 2 つの基本的な種類があります。

- 明示的または暗黙的なオフロードモデルを利用してホストと選択したコプロセッサーでシームレスに実行されるヘテロジニアス・アプリケーション
- ネイティブモデルによりコプロセッサーで独立して実行されるアプリケーション

ネイティブモデルとオフロードモデルの両方のアプリケーションを開発およびデバッグするには、専用機能のサポートが必要です。インテル® Parallel Studio XE 2013 Linux* 版は、使いやすく、インテル® Xeon Phi™ アーキテクチャーに完全に対応した広範なソリューションを提供します。Allinea や Rogue Wave のようなベンダーは、さらなるデバッグオプションを提供することによりインテル® Xeon Phi™ コプロセッサー環境を効率良くサポートしています。このため、インテル® Xeon Phi™ コプロセッサー用アプリケーションの開発者は、アプリケーションの分野に応じて、多くのデバッグオプションを選択することができます。では、インテルのデバッグ・ソリューションを、2 つのグループに分けて見ていきましょう。

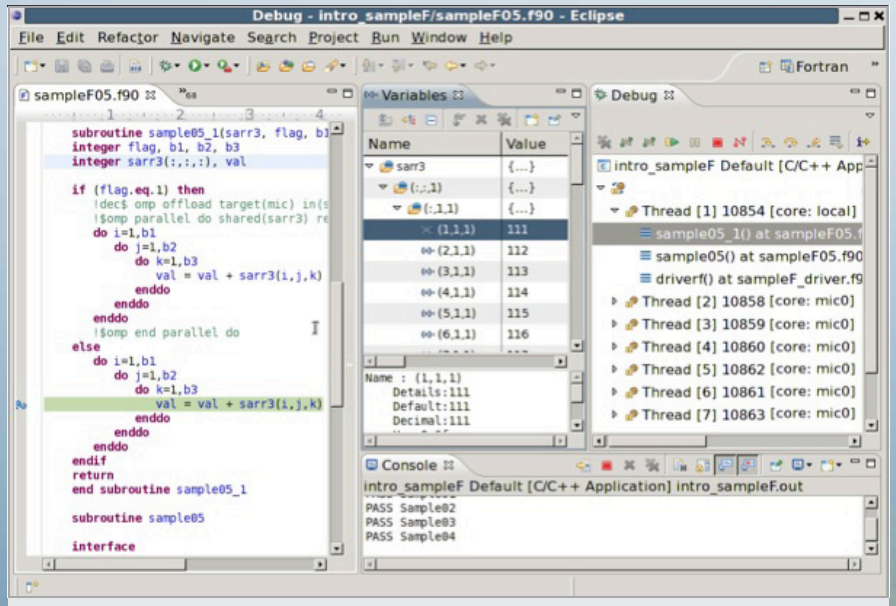


図 1

オフロードモデル

明示的と暗黙的の 2 つの異なるオフロードモデルがあります。どちらのモデルでも、インテル® Xeon Phi™ コプロセッサー上とホストシステム上で実行するアプリケーションを開発する単純かつ柔軟なプログラミング環境が提供されます。オフロード・アプリケーションは、ホストとターゲット両方のコードが含まれる単一の実行ファイルとして生成されます。データ転送と利用可能なコプロセッサーの検出は、実行時に自動的に制御され、ワークパッケージはコプロセッサーとホスト間で透過的にスケジュールされます。利用できるコプロセッサーがない場合、実行はホストシステム上で完全にネイティブのまま行われます。

プログラミングの柔軟性が増すと、標準ツールでのデバッグはより困難になります。このため、インテル® Parallel Studio XE 2013 Linux* 版では、両方のオフロードモデルに完全に対応した Eclipse* デバッガープラグインを提供しています。プラグインを利用することで、設定を追加することなく、すぐにデバッグを始められます。Eclipse* を統合開発環境 (IDE) として使用することにより、使いやすく慣れ親しんだグラフィカル・インターフェイスでデバッガーを実行できます。統合により、メニーコア・アーキテクチャーであるインテル® Xeon Phi™ コプロセッサーで必要不可欠な、マルチコア・コプロセッサーにおける数百スレッドまでのスケーラビリティーが C、C++、Fortran に提供されます。

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 を参照してください。

ネイティブモデル

ネイティブモデル (アプリケーションをコプロセッサーのみで実行) は、インテル® Parallel Studio XE 2013 Linux* 版でさまざまなデバッグ・ソリューションが提供されています。このソリューションは、開発システムで作成したアプリケーションをリモート・コプロセッサーで実行します。コプロセッサーは必ずしも同じ開発システムに搭載する必要はなく、ネットワーク経由でアクセス可能な別のホストシステムに搭載してかまいません。

このような使用状況の場合、デバッグをサポートするためには、リモート機能が必要です (前述の Eclipse* プラグインからも利用可能)。2 つ目のソリューションは、インテル® デバッガー (IDB) によるコマンドライン・デバッグです。通常の IDB デバッグセッションは、次のステップで行います。

```
$ idbc_mic -tco -rconnect=tcpip:<コプロセッサー>
(idb) idb file-remote <コプロセッサーの実行ファイルのパス>
(idb) file <開発システムの実行ファイルのパス>
...
```

idbc_mic コマンドにコプロセッサーの名前または IP アドレスを指定してデバッガーを開始します。デバッガーを開始したら、最初にコプロセッサーで実行する実行ファイルのパスを指定し、次にデバッガーを起動した開発システムの同じ実行ファイルのパスを指定します。この後、IDB を通常どおりに使用します。

あるいは、コプロセッサーですでに実行しているアプリケーションのプロセス ID <pid> を指定して、IDB をアプリケーションにアタッチすることもできます。

```
$ idbc_mic -tco -rconnect=tcpip:<コプロセッサー>
(idb) attach <pid> <開発システムの実行ファイルのパス>
...
```

IDB を使う利点は、その速度、C、C++、Fortran との互換性、習得しやすい GNU GDB 構文にあります。コマンドラインで動作するため、自動化されたスクリプトベースのテストにも簡単に利用できます。IDB の使用方法についての詳細は、インテル® デバッガー・ユーザー・リファレンス・ガイドを参照してください。

ネイティブモードの 3 つ目のソリューションは、インテル® Xeon Phi™ コプロセッサー対応の GNU GDB を利用することです。IDB と同様に、リモートデバッグ機能が提供されます。また、コプロセッサー上で直接ホストすることもできます。ホストで GDB を利用している開発者は、コプロセッサーでのデバッグにも GDB を使えます。インテルは、インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー向け GDB を追加でサポートしています。このバージョンはインテル® Parallel Studio XE 2013 Linux* 版に含まれていませんが、インテル® メニー・インテグレーテッド・コア・アーキテクチャーのフォーラム (英語) からダウンロードできます。

まとめ

幅広いデバッグツールが、インテル® Xeon Phi™ コプロセッサーのさまざまな利用状況をサポートしています。開発者は、使いやすい GUI ベースのデバッガーや高速でオーバーヘッドの低いコマンドライン・デバッガーから選択することができます。インテルをはじめ、多くのベンダーからさまざまなソリューションが提供されています。

関連資料 (英語)

インテル® Parallel Studio XE 2013

インテル® Xeon Phi™ コプロセッサーおよびインテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー

インテル® デバッガー・ユーザー・リファレンス・ガイド

インテル® メニー・インテグレーテッド・コア・アーキテクチャーのリソース (GNU GDB のソースを含む)

Allinea がインテル® Xeon Phi™ コプロセッサー開発者向けのツールをリリース

Rogue Wave が主力製品におけるインテル® Xeon® Phi™ コプロセッサーのサポートを発表

HTML5を利用した並列実行

インテル コーポレーション
HTML5 プロダクト・マネージャー
Max Domeika

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 を参照してください。

進化する HTML5

テキスト・マークアップ言語から発展した HTML5 には、ハイパフォーマンス・アプリケーション・プラットフォームに対する開発者の要望を満たす並列処理が採用されています。HTML5 言語仕様は、2014 年末までの正式勧告を目指して World Wide Web コンソーシアム (W3C) による策定が行われており、以前の HTML4 仕様よりも、ソースコードの可読性、マルチメディア、グラフィック、デバイスアクセス、オフラインストレージ、接続性、パフォーマンスが向上しています。

マルチコア対応の API

HTML5 開発者は、アプリケーションのパフォーマンスを評価して、マルチコア・プロセッサーのパフォーマンスを引き出す多くの API を開発しました。Web Workers*、Parallel JavaScript*、WebGL*、WebCL* などの API があり、ホモジニアス・マルチコア・プロセッサーとヘテロジニアス・マルチコア・プロセッサーの両方をプログラミング可能です。HTML5 アプリケーションの広範な採用はまだ始まったばかりです。しかし、サンプル・アプリケーションの提供に伴いその期待は日々増しており、多くのブラウザーベンダーが対応を進めています。これらのテクノロジーの例をいくつか見てみましょう。

図 1: Parallel JavaScript* で可能な処理の一例

Web Workers* API は、従来のマルチプロセシングに例えられる、メッセージ・パッシング・ベースのプログラミング・モデルを指定します。インスタンス化された Web Workers* は、互いに、またはドキュメント・オブジェクト・モデル (DOM) と呼ばれるプライマリー Web オブジェクトとアプリケーション・データ空間を共有しません。Web Workers* は多くの主要なブラウザーでサポートされ、API 仕様は承認前の最終段階に入っています。

図 2 は、Web Workers* を使った、2 つのファイル main.js と compute.js を含むサンプルプログラムです。main.js のコードは、compute.js のコードを実行する新しい Web Worker を作成します。通信は onMessage プロパティーと postMessage メソッドにより制御されます。

WebGL* は、Khronos Group から提供されている、OpenGL* ES 2.0 標準ベースの JavaScript* API です。HTML5 のキャンバス・オブジェクト内で作用する 3D グラフィックス・プリミティブの呼び出しが可能です。HTML5 のキャンバスは、2D グラフィックスと 3D グラフィックスの描画が可能な、ピクセルベースの描画サーフェイスです。WebGL* は、グラフィックス・アクセラレーターの並列処理機能を利用しています。詳細は、WebGL Web サイト (英語) を参照してください。

Khronos Group から提供されている別の標準である WebCL* は、OpenCL* に JavaScript* を結合した、ヘテロジニアス・マルチコア処理用のネイティブな標準です。WebCL* のプログラミング・モデルはアクセラレーター・モデルで、必要な処理が含まれる計算カーネルを作成して、一連のアクセラレーターにカーネルとデータをディスパッチします。

Parallel JavaScript* (開発コード名: River Trail) は、データ並列構造により JavaScript* を拡張します。このテクノロジーは OpenCL* を拡張したもので、WebCL* のアクセラレーター・モデルよりも高レベルの抽象性を提供します。Parallel JavaScript* は、ParallelArray データ構造と並列操作 (マップ、コンバイン、レデュース、スキャン、スキャッター、フィルターなど) を追加することにより JavaScript* を拡張します。図 3 は、値のセットの合計を計算する 2 つの JavaScript* コードです。1 つ目はシリアルに実行し、2 つ目は Parallel JavaScript* により並列に実行しています。詳細は、Intel Labs' River Trail Wiki (英語) を参照してください。

```
//main.js
var worker = new Worker('compute.js');
worker.onmessage = function(event) { alert (event.data); };
worker.postMessage('information');

//compute.js
self.onmessage = function(event) {  // Processing task
self.postMessage("received: " + event.data); };
```

図 2: Web Workers* の例

```
//シリアル
var val = new Array(1,2,3,4,5,6);
var sum = 0;
for (var i=0; i<6; i++){
  sum += val[i];
}

//並列
var val = new ParallelArray([1,2,3,4,5,6]);
var sum = val.reduce(function plus(x,y) { return x+y; });
```

図 3: Parallel JavaScript* の例

API の選択

次の表では、推奨する API とその利点を示しています。

アプリケーションの利点:	推奨する API:
グラフィックス・プロセシング・ユニットへの専用アクセスによる並列処理	WebGL*
ベクトルやグラフィカルでない、より汎用のタスク並列処理	Web Workers*
並列にベクトル計算を実行。WebGL よりプログラミング可能	
> 計算タスクがバンドルされプロセシング・ユニットにディスパッチされるアクセラレーター・プログラミング・モデル	WebCL*
> 並列計算で高いレベルの抽象性	Parallel JavaScript*

表 1

まとめ

これらの 4 つの API 標準で示されているように、HTML5 ではマルチコア処理が採用されています。これらの仕様が開発者とエンジニアにより実装され、今後も発展および改良されることを期待しています。

「HTML5 開発者は、アプリケーションのパフォーマンスを評価して、マルチコア・プロセッサーのパフォーマンスを引き出す多くの API を開発しました。Web Workers*、Parallel JavaScript*、WebGL*、WebCL* などの API があり、ホモジニアス・マルチコア・プロセッサーとヘテロジニアス・マルチコア・プロセッサーの両方をプログラミング可能です。」

BLOG highlights

Rogue Wave のツールがインテル® Xeon Phi™ コプロセッサーをサポート

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

Rogue Wave Software は先日、**主要製品におけるインテル® Xeon Phi™ コプロセッサーのサポートを発表しました。**このサポートには、SourcePro* C++、IMSL* 数値ライブラリー、TotalView* デバッガー、ThreadSpotter* が含まれています。詳細は、**プレスリリース**をご覧ください。Rogue Wave Software の Products & Alliances VP、Scott Lasica 氏からは、「インテル® Xeon Phi™ コプロセッサーで動作するように TotalView の移植を開始したところ、アクセラレーターでは 1 年以上かかったところまでを 1 週間で進めることができました。」というコメントをいただきました。一般的な x86 プログラミング・モデル、Linux* ベースの環境、インテル® 開発ツールを利用することで、インテル® Xeon Phi™ コプロセッサーを僅か数週間でサポートすることができたのです。

Rogue Wave がインテル® Xeon Phi™ コプロセッサー開発者向けソリューションの提供を開始したニュースを聞いてうれしく思います。インテル® Xeon Phi™ コプロセッサーをサポートした Rogue Wave ツールを待ち望んでいたユーザーからも歓迎する声が数多く寄せられています。

重要な点は、これらのツールは新しいツールや大幅に異なるアドオンではないことです。

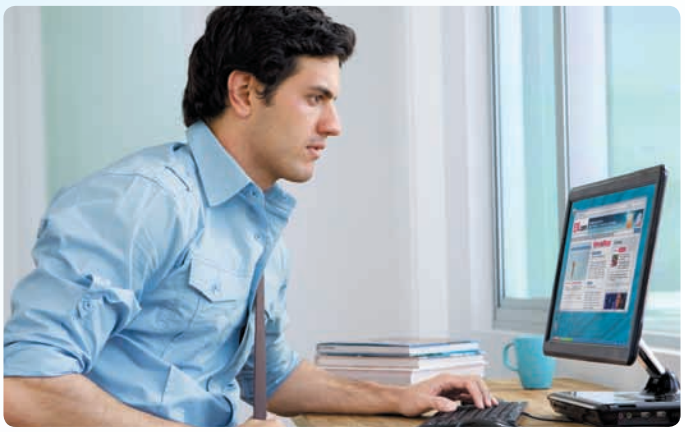
この記事の続きはこちら (英語) でご覧になれます。

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® 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 を参照してください。

リソースおよび Web サイト



Go Parallel

Go Parallel の目的は、「マルチコアのパワーをアプリケーションのパフォーマンスに変える」ことで開発者を支援することです。豊富な情報を掲載するこのサイトは、マルチコア関連のブログ、ニュース、ビデオ、機能説明、その他の役立つリソースの価値あるクリアリングハウスです。

"What If" Experimental Software

開発中のインテルの先端研究や最新テクノロジーを試せるとしたらどうでしょう? そして、ユーザーのフィードバックが将来の製品に役立つとしたら? それを可能にするのが What If Experimental Software です。最先端のツールをテストしたり、他のユーザーと協力して、"What If?" ブログやサポートフォーラムにより意見を交換できます。

インテル® ソフトウェア・ネットワーク

並列プログラミングのマネージャビリティーから仮想化やビジュアル・コンピューティングまで、さまざまな開発者向けに広範なソフトウェアのトピックが用意されています。豊富なコンテンツのコレクションには、インテル® ソフトウェア・ネットワーク TV、ブログ、ビデオ、ツール、ダウンロードが含まれます。

最新ソフトウェアの概要

最新のインテル® ソフトウェアの使い方を説明するビデオで、メモリーチェック、スレッドチェック、hotspot 解析、ロックと待機の解析、その他の重要な役割を担っているソフトウェアの機能をご覧になれます。

インテル® Inspector XE

インテル® VTune™ Amplifier XE

インテル® Software Evaluation Center

インテル® Software Evaluation Center では、インテル® ソフトウェア開発製品の 30 日間評価バージョンをダウンロードできます。評価用ライセンスをリクエストした後、インテル® プレミアサポートのアカウントを作成すると、評価期間中にインテル® プレミアサポートの無料サポートを受けることができます。また、インテル® ソフトウェア・ネットワーク・フォーラム (英語) の無料サポートにアクセスすることもできます。インテル® Parallel Studio を評価する場合は、インテル® ソフトウェア・ネットワーク・フォーラム (英語) の無料サポートにのみアクセスできます。

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