THE PARALLEL UNIVERSE

Issue 5
2010 年 11 月

XE

スポットライト

上野浩太郎がパフォーマンスの向上に対する取り組みについて語ります。

編集者からのメッセージ

James Reinders

インテル® Parallel Building Blocks: 並列化のパズル解明の鍵

David Sekowski

ハイパフォーマンスを容易に実現する

インテル® Parallel Studio XE

およびインテル® Cluster Studio ツールスイート

Sanjay Goil、John McHugh

目次

ハイパフォーマンスを達成するためのオプションがさらに充実

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ソフトウェア開発製品部門のチーフ・エバンジェリスト兼ディレクターです。『Intel Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』などの並列化に関する文献を発表しています。

今回の Parallel Universe では、今日のコンピューターのパフォーマンスを引き出すように設計された最新のインテル® ソフトウェア開発製品に注目してみたいと思います。

技術的な観点からすると、ハードウェアはこれまでになく複雑になってきています。例えば、並列性という 1 つの機能だけをとってみても、命令発行の頻度を高めるスーパースカラー・プロセッサー、ビット幅が拡張された SIMD 命令、マルチコア・プロセッサー、マルチプロセッサー・システムなど、さまざまなレベルのコンピューター・アーキテクチャーに備わっています。

システムが複雑になるにつれて、ソフトウェア開発者を支援するためのソフトウェア開発ツールも進化を遂げています。皆様からは、ツールの改善に役立つご意見をいただいて大変感謝しております。2010 年、インテルはハイパフォーマンスに必要なツールを単純化しつつ、現在および将来のハードウェアに対応するための開発者向けツールを提供するという大胆な戦略をとりました。インテルでは、開発者に幅広い選択肢を提供し、ソフトウェアへの投資を無駄にすることなく、現在 / 将来のハードウェア向けにフォワード・スケーリングを実現するべく取り組んでいます。

2010 年 11 月 9 日には、最も人気のあるソフトウェア開発製品の最新版ツールが統合され、インテル® Parallel Studio XE とインテル® Cluster Studio という 2 つの統合ソリューションとなり、リリースされました。インテル® Parallel Studio XE は、今日のマシンが抱える高度なパフォーマンスの課題に取り組む C、C++、Fortran 開発者を支援するツールです。そして、インテル® Cluster Studio は、特に MPI、C、C++、Fortran を使用したプログラムの分散コンピューティングに特化したツールです。

インテル® コンパイラー 12.0、最新バージョンのインテル® VTune™ パフォーマンス・アナライザー、並行処理に対応した新しいメモリーチェック機能、セキュリティーと堅牢性のためのコード解析、MPI サポートの拡張、C/C++ 向けのインテル® Parallel Building Blocks、Co-Array Fortran のサポート、コンパイルされたバイナリーだけでなく .NET コードにも対応した新しいスレッドエラー検出機能を含む多数の新機能が追加されています。

インテル® Parallel Studio XE には Linux* 版と Windows* 版があり、インテル® Composer XE（コンパイラーとライブラリー）、インテル® VTune™ Amplifier XE、および インテル® Inspector XE が含まれています。

インテル® Cluster Studio には Linux* 版と Windows* 版があり、インテル® Composer XE、インテル® MPI ライブラリー、インテル® MPI ベンチマーク、およびインテル® トレース・アナライザー / コレクターが含まれています。

インテルのツールは、アプリケーションごとに異なるさまざまな要件に対応するべく、幅広い並列プログラミング手法を提供します。OpenMP*、MPI、Co-Array Fortran、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® 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）、そしてインテル® 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® Cilk™ Plus、インテル® Array Building Blocks（インテル® ArBB）からなるインテル® Parallel Building Blocks（インテル® PBB）を利用して、他の追随を許さない堅固な並列実装を可能にします。

これらの革新的なツールの利点については、インテル® Parallel Building Blocks、Fortran の新機能、インテル® MKL に関するセクションで説明しています。不具合をなくすことは重要であり、インテルのツールにより、お使いのビルド環境で容易にこの課題に取り組めます。インテルのツールは、並列プログラムに必要なコードの品質、セキュリティー、アプリケーションの信頼性を高めるためのソリューションを提供します。次世代のメモリー / スレッド・エラー・チェッカーは、メモリー、スレッド、およびコード分析を組み合わせて、セキュリティーの向上を図ります。「Intel® Inspector XE: An essential tool during development along with Intel® Composer XE（インテル® Inspector XE: インテル® Composer XE とともに開発に不可欠なツール）」という記事では、インテル® Inspector XE が開発サイクルの一部として不可欠なツールであることが述べられています。まさに、そのとおりでしょう。

ハイパフォーマンスを達成するためには、hotspot を検出し、システムで何が起こっているのかを理解して、hotspot を軽減する上でパフォーマンス・プロファイリングが不可欠です。次世代のプロファイラーであるインテル® VTune™ Amplifier XE は、使いやすいだけでなく、早急に対応が必要なパフォーマンス問題の詳細な情報を提供します。
クラスター開発者向けには、最大規模のシステムでパフォーマンスを引き出せるよう設計された高度にスケーラブルな MPI 実装をインテル® MPI ライブラリーで提供しています。「On a path to petascale with commodity clusters and Intel MPI（クラスターとインテル® MPI によるペタスケールへの道）」という記事では、HPC 向けクラスターツールにおけるインテル® MPI ライブラリーの進化について紹介しています。

ハイパフォーマンスを達成するためのソフトウェアの継続的な開発は複雑な作業です。インテル® ソフトウェア開発製品は、現在および将来のインテル® アーキテクチャーで動作し、プロセッサー・テクノロジーとマルチコア / メニーコア・プロセッサー分野におけるインテルのリーダーシップをより強固なものにしています。そして、ソフトウェア開発を予測可能なものにし、お客様のソフトウェアへの投資を将来に渡って保護します。インテルのソフトウェア開発製品グループでは、ツールの購入、インストール、開発、サポートを容易にするべく取り組んでいます。

インテル® Parallel Studio XE およびインテル® Cluster Studio は、リリース前にベータユーザーから賞賛の声をいただいています。この新しいツールは、ハイパフォーマンスを達成するためのプログラミング、生産性、プログラム実現性を大幅に向上させるでしょう。

是非、実感してみてください。

JAMES REINDERS
オレゴン州ポートランド
2010 年 11 月

ハイパフォーマンスを
容易に実現

インテル® Parallel Studio XE

およびインテル® Cluster Studio ツールスイート

インテル® ソフトウェア開発製品部門
製品マーケティング・マネージャー **Sanjay Goil**
マーケティング・コミュニケーション・マネージャー **John McHugh**

インテル® Parallel Studio XE 2011

インテル® Composer XE 最適化コンパイラーと ライブラリー	**インテル® Inspector XE** メモリー、スレッド、 セキュリティー・アナライザー	**インテル® VTune™ Amplifier XE** パフォーマンス・ プロファイラー

図 1

2010 年 9 月にリリースされたインテル® Parallel Studio 2011 は、Microsoft* Visual Studio* を使用する Windows* ベースの C++ 開発者向けに、インテル® アーキテクチャーでのアプリケーション開発に必要不可欠なパフォーマンス・ツールを提供することを目的としています。このツールは、大幅な技術革新により、マルチコア向けの並列アプリケーションのビルド、デバッグ、チューニングにおいて、開発者の生産性をかつてないほど向上します。新しいインテル® Parallel Building Blocks（インテル® PBB）を使用することで、開発者は C/C++ アプリケーションの並列化と拡張を行い、ハイパフォーマンスなアプリケーションを効率良く作成できます。

そして、このたび、インテルでは次世代ツール群の範囲を拡大し、現在のマルチコアでの優れたパフォーマンスと、将来のメニーコア向けのフォワード・スケーリングを必要とする Windows*/Linux* の C/C++、Fortran 開発者向けにツールスイートを提供することになりました。このインテル® Parallel Studio XE 2011 には、次のツールが含まれます。

- インテル® C/C++ コンパイラーと Fortran コンパイラー
- インテル® マス・カーネル・ライブラリー（インテル® MKL）およびインテル® 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）のパフォーマンス・ライブラリー
- インテル® PBB ライブラリー（インテル® TBB、インテル® Cilk™ Plus、インテル® ArBB）
- インテル® Inspector XE メモリー / スレッド・エラー・チェッカー
- インテル® VTune™ Amplifier XE パフォーマンス・プロファイラー

これまで HPC プログラマーは、コンピューターで利用可能なあらゆる処理能力を活用してきました。インテル® アーキテクチャーでは、過去 10 年間、ムーアの法則によりパフォーマンスを向上させてきましたが、パフォーマンスのさらなる向上への要求は高まるばかりです。また、科学や工学の分野にはまだ未解決の大きな問題が多数ある一方、より細かい粒度での物理シミュレーションも求められています。経済的な理由から、コンピューターの処理能力が制限され最良の結果が得られていなかったり、あるいは大きな処理の一部を部分的にシミュレーションしているといった場合もあります。インテルは、今回の HPC 向け製品の提供に意欲的であり、ハードウェアとソフトウェアの両方の技術革新を活用して、並列化とパフォーマンスへの取り組みを大きく推進するべく力を入れています。

インテル® Cluster Studio は、MPI を使用した HPC クラスター開発向けのツールです。スケーラブルなインテル® MPI ライブラリー、インテル® トレース・アナライザー / コレクターのパフォーマンス・プロファイラー、業界をリードする C/C++ および Fortran コンパイラーを 1 つに統合したクラスター開発ツールです。インテル® クラスターレディー・プログラムによるデプロイメント支援を合わせて利用することで、クラスター・アプリケーションを効率良く導入できます。

新しいツールスイート

ハイパフォーマンスなアプリケーションの開発には、統合開発ツールが必要です。これらのツールには従来、コンパイラー、デバッガー、パフォーマンス・ライブラリー、並列ライブラリーなどがありますが、開発時にエラー検出の正確さとパフォーマンスのプロファイリングでよく問題が発生していました。同期ロックでデータ競合、デッドロック、パフォーマンス・ボトルネックが発生していたり、またランタイムでセキュリティーの脆弱性があると、コードが正しく実行されなかったり、特定の実行においてエラーを引き起こしやすくなる場合があります。インテルのメモリー / スレッド・エラー・チェッカーとパフォーマンス・プロファイラーを開発プロセスに加えることで、このような問題に対処し、安定性が高くセキュアなコード開発が行えるようになります（図 1 を参照）。

高度なパフォーマンスや分散パフォーマンスを実現するため、インテルでは、IA-32、インテル® 64 アーキテクチャー、互換プラットフォームにおいて、メッセージ・パッシング・インターフェイス（MPI）を使用した HPC クラスター プログラム開発を行うための HPC ツールを容易に入手、導入、使用できるようにしました（図 2 を参照）。

大幅なパフォーマンス向上。コードの信頼性。
フォワード・スケーリング。

フェーズ	生産性向上ツール	機能	利点
コードの改善	 インテル® Composer XE	C/C++ と Fortran コンパイラー、パフォーマンス・ライブラリー、並列モデル	› アプリケーションのパフォーマンスを向上させ、マルチコアのスケーラビリティーを引き出し、メニーコア向けにフォワード・スケーリング。セキュリティーと安定性を備えたコードを実現。
高度なメモリー / スレッド・エラー・チェック	 インテル® Inspector XE	コードの信頼性と品質を高めるメモリー / スレッドエラーのチェック	› 早期にメモリーとスレッドの不具合を発見し生産性を向上させ、コストを削減
高度なパフォーマンス	 インテル® VTune™ Amplifier XE	パフォーマンスとスケーラビリティーを最適化するパフォーマンス・プロファイラー	› 従来の推測作業をなくして時間を節約し、パフォーマンスとスケーラビリティーのボトルネックを簡単に発見。使いやすく、詳細な情報を取得可能。

図 2

インテル® Parallel Studio XE 2011 の主な機能

› **複数のオペレーティング・システムで利用可能 :** インテル® Parallel Studio XE は、Windows* プラットフォームにも Linux* プラットフォームにも同一のセットを提供します。Mac OS* X では、C/C++ および Fortran コンパイラー、パフォーマンス・ライブラリー、並列ライブラリーによる高度な最適化が利用可能です。

› **安定性 :** インテル® Inspector XE のメモリー / スレッド・アナライザーは、メモリーエラーおよびスレッドエラーを未然に検出します。

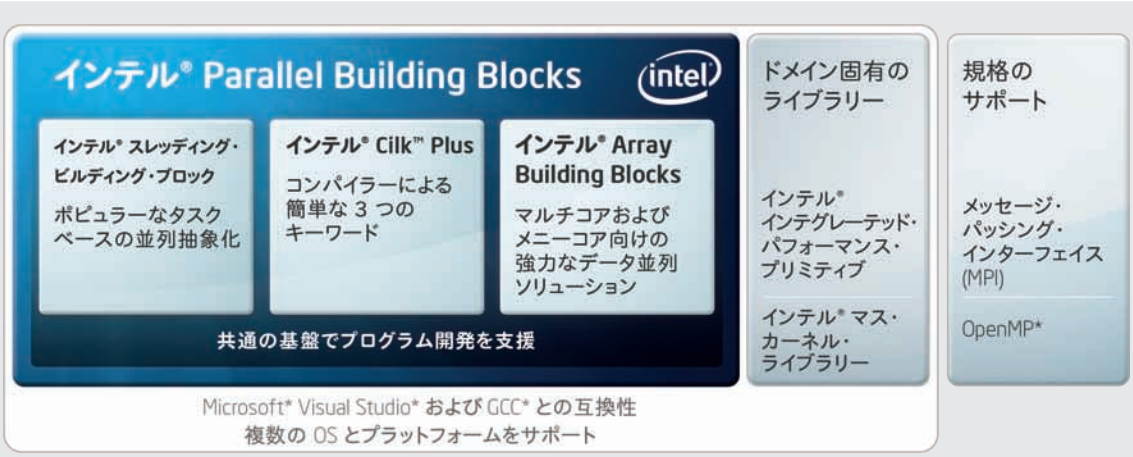
› **コードの品質 :** インテル® Parallel Studio XE のスタティック・セキュリティー解析により、開発者はソフトウェアのセキュリティーに関する問題を効率良く見つけられます。

› **高度な最適化 :** インテル® Composer XE のコンパイラーとライブラリーは、インテル® AVX のサポートを含む高度なベクトル化をサポートします。C/C++ 最適化コンパイラーに含まれるインテル® PBB ライブラリーは、優れたスケーラビリティーと信頼性を備え、並列化することで簡単に解決できる問題タイプを広げます。Fortran 開発者向けには、Fortran 2008 規格の Co-Array Fortran と一部の機能が追加されました。

› **パフォーマンス :** インテル® VTune™ Amplifier XE パフォーマンス・プロファイラーは、パフォーマンスを制限するシリアルおよび並列コードのボトルネックを発見します。より直感的なインターフェイスが備わり、統計コールグラフや時間軸ビューが向上しました。インテル® MKL、インテル® IPP のパフォーマンス・ライブラリー群は、よく使用される算術 / データ処理ルーチンに堅固なマルチコア・パフォーマンスをもたらします。アプリケーションにただリンクするだけで、マルチコア並列処理への最初のステップを簡単に踏み出せます。

› **互換性とサポート :** インテル® Parallel Studio XE は主要な開発環境やコンパイラーと互換性があります。インテルでは、フォーラムやインテル® プレミアサポートを通じて幅広いサポートを提供しています。インテル® プレミアサポートでは、テクニカルサポートと、すべての製品のアップデートが 1 年間ご利用いただけます。

ソフトウェア開発プロジェクトはいくつかのステップを経て、ターゲットシステムで最適なパフォーマンスを達成します。通常、開発者はアプリケーション実行の基本的なパフォーマンス・プロファイルと hotspot をまず確認します。最適化の機会が特定されると、コンパイラー、パフォーマンス・ライブラリー、並列ライブラリーによってコードの並列化が行われ、タスクレベル、データレベル、ベクトル化の候補が示されます。最後に、エラー検証ツールでスレッド / メモリーエラーとセキュリティーの脆弱性をチェックして、コードの安定性を高めます。このサイクルを繰り返すことで、より効率の良いアプリケーションに導きます。



旧名称	新名称
インテル® コンパイラー・スイート・プロフェッショナル・エディション	インテル® Composer XE
インテル® C++ コンパイラー・プロフェッショナル・エディション	インテル® C++ コンパイラー XE
インテル® Visual Fortran コンパイラー・プロフェッショナル・エディション	インテル® Visual Fortran Composer XE
インテル® Visual Fortran コンパイラー・プロフェッショナル・エディション IMSL* 同梱	インテル® Visual Fortran Composer XE IMSL* 同梱
インテル® VTune™ パフォーマンス・アナライザー（インテル® スレッド・プロファイラー付属）	インテル® VTune™ Amplifier XE
インテル® スレッド・チェッカー	インテル® Inspector XE
インテル® クラスター・ツールキット・コンパイラー・エディション	インテル® Cluster Studio

図 3

業界をリードする次世代ツール群、インテル® Parallel Studio XE 2011 は、Windows*/Linux* 上で最新の x86 プロセッサー用のクロスプラットフォーム機能を求める C/C++ および Fortran 開発者に最適です。今回のリリースでは、大幅な機能追加に伴って一部の製品名が変更されています。ここで明記していないものについては、引き続き従来の製品名を使用しています（図 3 を参照）。

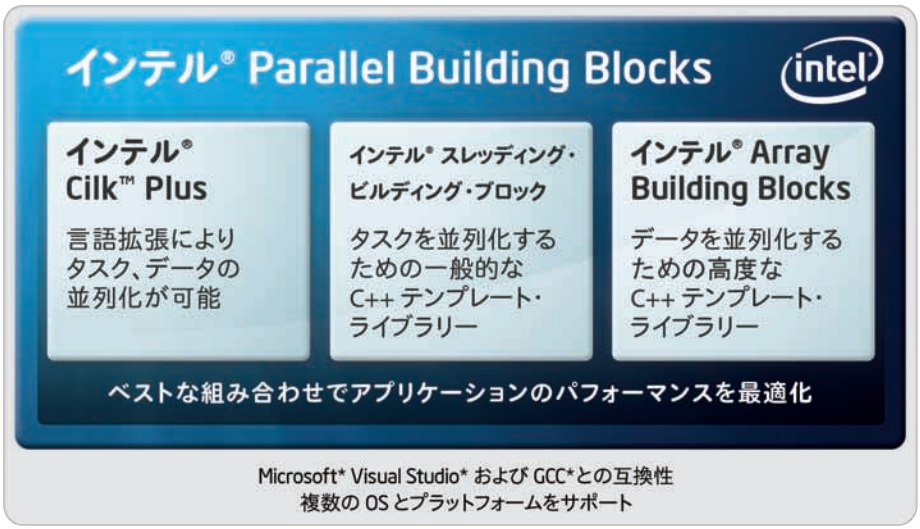


図 4

「インテル® Parallel Studio XE 2011 は、パフォーマンスの向上に取り組む Windows* ベースの C++ 開発者にとって素晴らしいソフトウェア開発ツールです。コードでインテル® Cilk™ Plus と拡張された配列表記を使用したことで、パフォーマンスが驚異的に向上しました。パフォーマンスを向上したいなら、インテル® Parallel Studio XE 2011 を試してみるべきです。」

BR&E Inc. 社
研究開発エンジニア
Jorge Martinis 氏

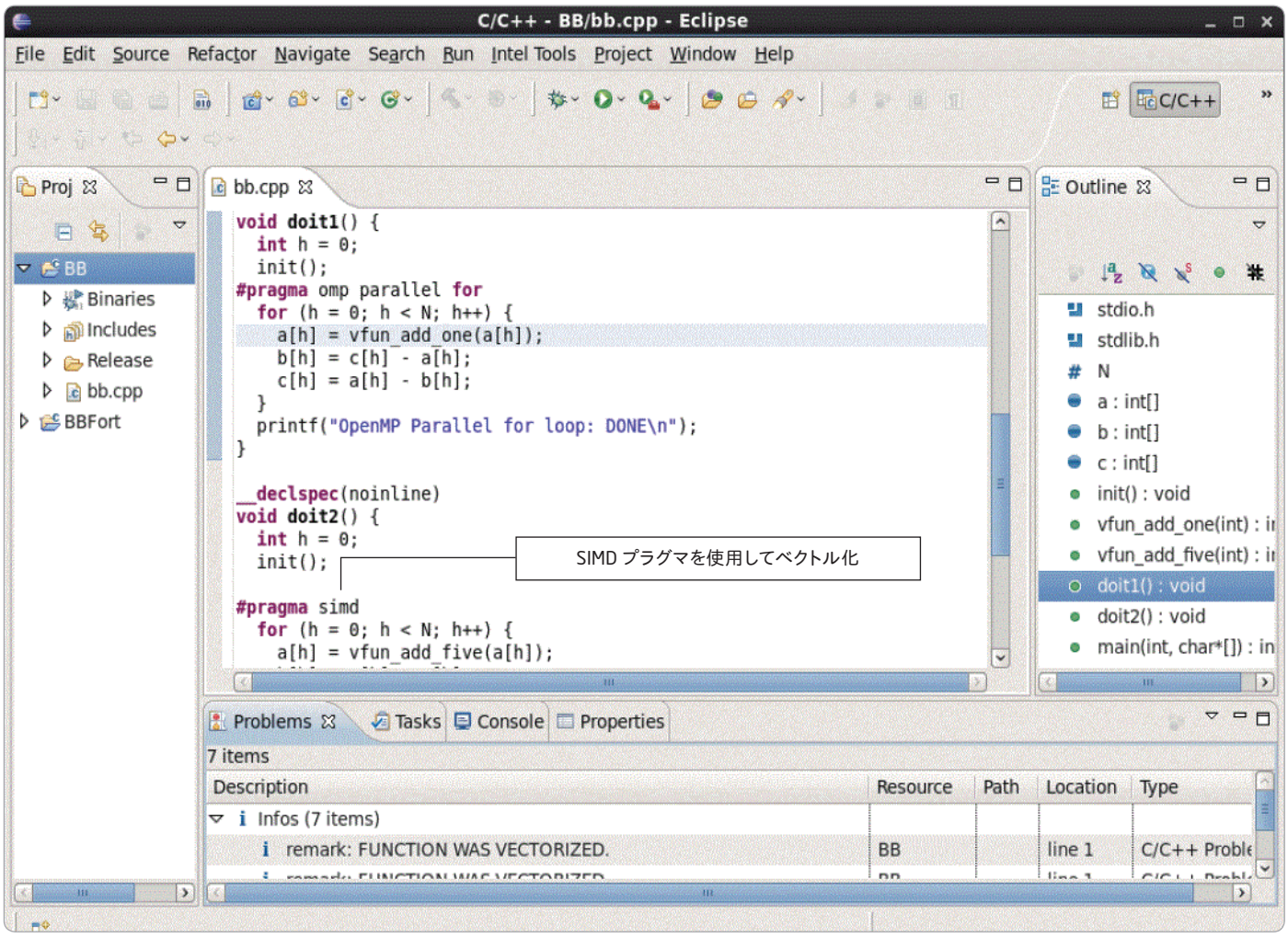


図 5

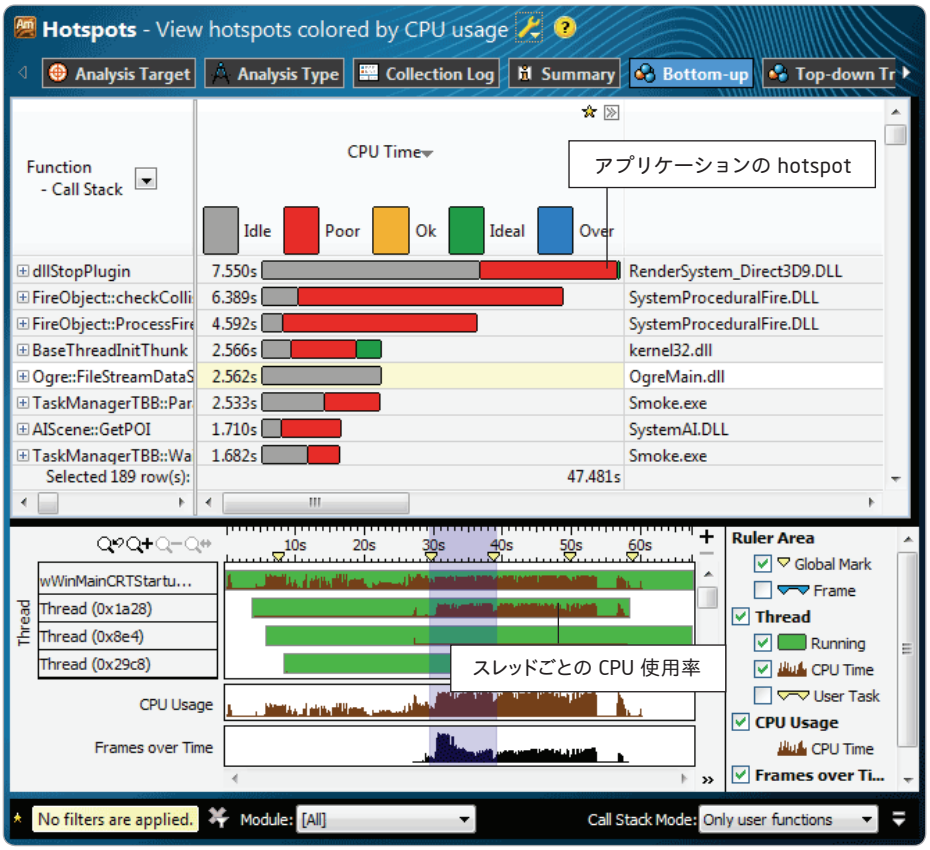


図 6

「BlueJeans Network では、次世代のビデオ処理クラウド・ソリューションに取り組んでおり、膨大な量のオーディオ、ビデオ、データコンテンツを処理しています。これらの処理は、プロセッサーに非常に大きな負担をかけます。インテル® IPP 7.0 の広範囲なオーディオ / ビデオ処理機能は、弊社の要件に合致した完璧なソリューションでした。もし、これらをゼロから作成していたら、いつまでたっても終わらなかったでしょう。インテル® IPP のおかげで時間を大幅に節約できました。もちろん、インテル® IPP 7.0 をお勧めします。」

BlueJeans Network 社
ソフトウェア・アーキテクト
Emmanuel Weber 氏

インテル® Composer XE の新機能

インテル® Composer XE には、次世代の C/C++ および Fortran コンパイラー (v12.0) とパフォーマンス・ライブラリーおよび並列ライブラリー (インテル® MKL 10.3、インテル ® IPP 7.0、インテル® TBB 3.0) が含まれています (図 2 を参照)。

最新のインテル® C/C++ コンパイラーであるインテル® C++ コンパイラー XE 12.0 は、インテル® AVX をサポートし、最新のインテル® アーキテクチャー・プロセッサー、Sandy Bridge (開発コード名) 向けにも最適化されています。インテル® PBB が含まれているため、インテル® Cilk™ Plus、インテル® TBB、インテル® ArBB (ベータ版。別途入手可能) を使用して、タスク、ベクトル、データ並列処理を組み合わせてマルチコアによる最適化を図ります (図 4 を参照)。また、インテル® AVX、SIMD プラグマを含むベクトル化サポート、最新の x86 マルチコア・プロセッサーにおける最高レベルのパフォーマンスの実現と並列化に向けたガイド付き自動並列化 (GAP) 機能も備えています。Windows* 版は、Visual Studio* 2010 にも対応しています。

インテル® Fortran コンパイラー XE 12.0 には、最新の x86 マルチコア・プロセッサーにおける優れたパフォーマンスの実現と並列化に向けたいくつかの強化機能が含まれています。例えば、Fortran 2003 規格の完全なサポート、Fortran 2008 規格の一部のサポート (Co-Array Fortran、AVX によるベクトル最適化)、自動並列化の支援などがあります (図 5 を参照)。

パフォーマンス・ライブラリーは、ハイパフォーマンスを実現する、高度に最適化され並列化された算術 / 科学関数とデータ処理ルーチンを引き続き提供します。算術ライブラリーであるインテル® MKL 10.3 では、向上したインテル® AVX サポート、サマリー統計ライブラリー、強化された LAPACK の C 言語パックを含むいくつかの点が強化されています。データ処理ライブラリーであるインテル® IPP 7.0 には、向上したデータ圧縮、コーデック、インテル® AVX および AES 命令のサポートが含まれ、引き続きデータ処理アプリケーションを支援します。

図 7

「インテル® Inspector XE 2011 は使いやすいツールです。分析レベルを設定して、収集したデータを視覚的に確認し、コード中の隠れたデータ競合に関する有益な情報を短時間で得ることができました。」

OTRADA Inc. 社
CEO 兼 CTO
Alex Migdalski 氏

メモリー / スレッド・エラー・チェッカーとパフォーマンス・プロファイラーにより向上した生産性

インテル® Parallel Studio XE 2011 は、インテル® Parallel Studio で導入された革新技術 ー ハイパフォーマンス、スケーラビリティー、コードの安定性を実現する高度な機能 ー を採用し、Linux* および Windows* プラットフォームに提供します。インテルでは以前から、Windows* と Linux* のプラットフォーム向けに開発ツールを用意しており、同一機能を両プラットフォームに提供できるよう努めてきました。これは、特に両プラットフォームで動作するアプリケーションを開発する上で重要な点です（図 6 を参照）。

インテル® Inspector XE のメモリー / スレッド・エラー・チェッカーの機能（図 7）は、スレッド / メモリー解析ツールを使用した静的 / 動的なコード解析により、安定性が高くセキュアで高度に最適化されたアプリケーションを開発するべく、C/C++ および Fortran 開発者を支援します。

インテル® Inspector XE メモリー / スレッド・エラー・チェッカーの新機能 :

- 容易な設定と解析の実行
- 次のようなコードの不具合を素早く発見
  - メモリーリークとメモリー破壊
  - スレッドのデータ競合とデッドロック
- ネイティブスレッドのサポート — スレッドを使用する並列モデルを理解
- 標準ビルドとバイナリーで動作する動的インストルメンテーション
- タイムライン・ビューでそれぞれのスレッドのコンテキストを確認
- Windows*/Linux* 向けの直感的なスタンドアロン GUI とコマンドライン・インターフェイス
- 高度なコマンドライン・レポート出力

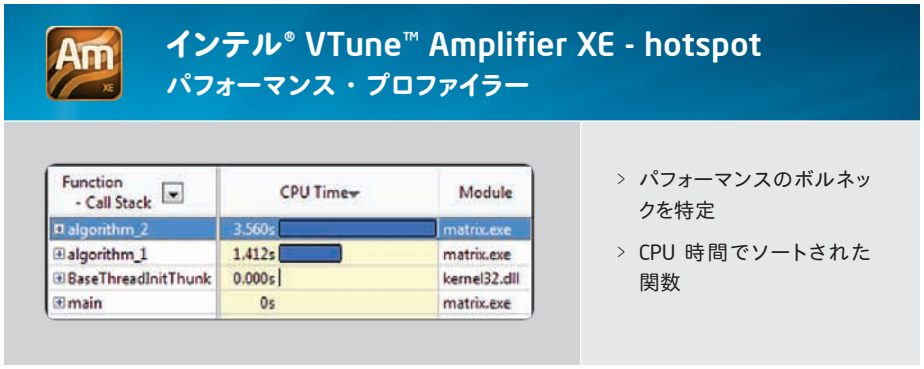


図 8

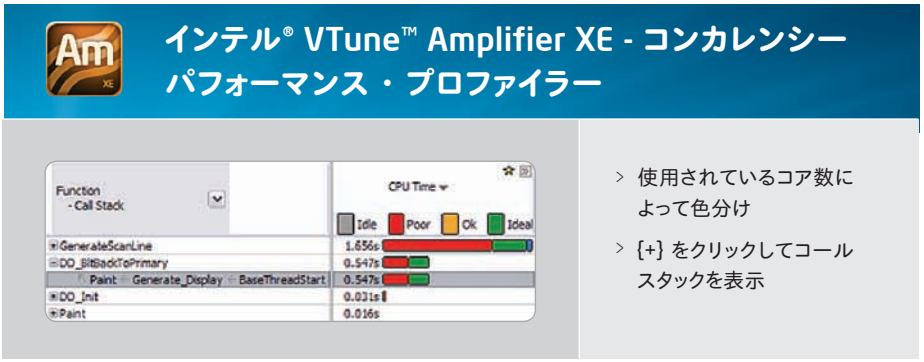


図 9

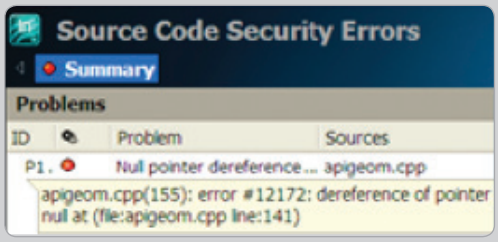
インテル® VTune™ Amplifier XE 2011 は新世代のインテル® VTune™ パフォーマンス・アナライザーで、マルチコア・パフォーマンスのボトルネックを素早く検知し、優れた洞察を提供します。従来の推測作業をなくして、Windows*/Linux* アプリケーションのパフォーマンス特性を分析することで、スケーラビリティーに影響するボトルネックへの手がかりが得られ、迅速かつ精度の高い意思決定が行えます（図 8、9 を参照）。

新世代のインテル® VTune™ パフォーマンス・プロファイラーの新機能：

- 事前定義された解析
- 迅速な hotspot 解析（時間がかかっている関数とコールスタック）
- 強力なフィルター
- スレッド・タイムライン
- フレーム解析
- 実行プロセスへのアタッチ（Windows*）
- イベントの多重化
- 容易なリモート収集
- 結果の比較機能の向上
- Visual Studio との緊密な統合
- root ユーザー以外でのインストール（Linux*）
- EBS ドライバーのインストールのみ root 権限が必要

「インテル® VTune™ Amplifier XE 2011 は、次世代の インテル® VTune™ パフォーマンス・アナライザー...」

インテル® Inspector XE - セキュリティー・チェッカー
メモリー、スレッド、セキュリティー・チェッカー



インテル® Parallel Studio XE とともに使用した場合

- スタティック解析によりコードのセキュリティーを向上
- バッファー・オーバーラン、安全ではないライブラリーの使用、初期化されていない変数、不正なポインターを発見

図 10

ソフトウェアのセキュリティー解析は開発の初期段階で開始すると良いでしょう。インテル® Parallel Studio XE 2011 を使用すると、出荷前にソフトウェアの問題を素早く特定し、解決できます。これにより、重大なソフトウェア・セキュリティーの脆弱性を早期に発見できるため、不具合の発見と修正にかかるコストを最小限に抑えられます（図 10、11 を参照）。

「スタティック・セキュリティー解析（SSA）機能により、多数の潜在的な不具合を簡単に見つけ、未然に防ぐことができました。最終的には、SSA が一般的なセキュリティー問題の検出に役立つことを期待しています。」

Envivio 社
シニア・アーキテクチャー・エンジニア
Mikael Le Guerroue 氏

インテル® Parallel Studio XE に含まれるスタティック・セキュリティー解析 (SSA) はコードの安定性を高める機能を提供

- 使いやすく、短時間の設定でスタティック解析結果を取得
- 簡単なステップでスタティック解析を設定、実行
- 開発サイクル全体にわたって不具合を発見、修正
- 250 種類以上のセキュリティー・エラーを検出
  - バッファー・オーバーラン、初期化されていない変数
  - 安全ではないライブラリーの使用、演算のオーバーフロー
  - チェックされていない入力、ヒープ破壊
- 問題の状態の追跡 — ソースが変更され、行番号が変わった場合にも対応
- 問題セットとソースの場所の表示
- フィルター、優先度の設定、問題セットの状態の管理
- Windows*/Linux* 向けの直感的なスタンドアロン GUI とコマンドライン・インターフェイス

機能	利点
Linux* および Windows* の両プラットフォームのサポート	開発を支援するさまざまな機能を持つ同一ツールセットを、Windows* および Linux* の両プラットフォームに提供 — 強化されたパフォーマンス、生産性、プログラム実現性を備えています。
C/C++ コンパイラーとインテル® Parallel Building Blocks	アプリケーション・パフォーマンスを最適化するためのさまざまな柔軟性を備えた並列化タイプの選択肢（タスク、データ、ベクトル）により、ブレイクスルーを図ります。C/C++ 規格がサポートされます。
Co-Array Fortran (CAF) を含む Fortran 2008 規格をサポートする Fortran コンパイラー	業界をリードする Fortran コンパイラーは、ノードおよびクラスターにおける新しくスケーラブルな並列化をサポートします。（クラスターは別製品のインテル® Cluster Studio 2011 でサポートされます）。Fortran 規格がサポートされます。
1 つのパッケージで提供されるメモリー、スレッド、セキュリティー解析ツール	発見が困難なコーティング・エラーの検出を容易かつ素早く行い、開発者の生産性と効率性を高めます。
アップデートされたパフォーマンス・ライブラリー	並列ライブラリーへ単純にリンクするだけで、よく使用される演算およびデータ処理タスクにおけるマルチコア・パフォーマンスを実現します。
アップデートされたパフォーマンス・プロファイラー	使いやすい強化機能、マイクロアーキテクチャーの深い洞察、向上した GUI、より高速なパフォーマンスを提供します。

図 11

インテル® Cluster Studio
分散パフォーマンス

次のコンポーネントが含まれます。

- インテル® Composer XE（コンパイラーおよびライブラリー）
- インテル® MPI ライブラリー
- インテル® トレース・アナライザー / コレクター

HPC クラスター・コンピューティングのパフォーマンスとスケーラビリティーを向上

インテル® Cluster Studio 2011 は、インテル® アーキテクチャー・ベースのクラスターにおける分散パフォーマンスの新しいスタンダードです。このツールスイートにより、IA-32 およびインテル® 64 アーキテクチャー・ベースの高度な共有メモリー型並列クラスターシステムで、MPI ベースのアプリケーションのパフォーマンスを発揮できる柔軟な開発が可能です。新たに再設計されたインテル® MPI ライブラリー 4.0 は、新しいレベルのクラスター・スケーラビリティー、多くのファブリックに対応したインターコネクト・サポート、ノード上のメッセージの高速化、ハイブリッド並列化のサポート、それぞれのクラスター・アーキテクチャーとアプリケーション構造向けのアプリケーション・チューニング機能によりその利点を発揮します。インテル® トレース・アナライザー / コレクター 8.0 では、開発者向けに MPI ベースのクラスター・アプリケーションの解析とチューニング・サイクルを向上させるための新機能が追加されました。最新のインテル® C/C++ および Fortran コンパイラー・テクノロジーは、インテル® MKL 10.3、インテル® IPP 7.0、およびインテル® PBB（これらはインテル® Composer XE にもバンドルされています）と組み合わせることで、クラスターの各計算ノードにおいて並列アプリケーションの実行をさらに最適化できます。インテル® Cluster Studio 2011 は、クラスター上で Co-Array Fortran (CAF) をサポートしています。

インテル® クラスターレディー (ICR) プログラムは、IA ベースのハイパフォーマンス・クラスターでより長時間の稼働に対応し生産性を向上させる一方、総所有コスト (TCO) を削減するようなクラスター・アーキテクチャーの定義を支援するプログラムです。このプログラムとともに利用することで、インテル® Cluster Studio 2011 は MPI ベースのクラスター・アプリケーションのコーディング、デバッグ、最適化を容易にし、最大でペタスケールへのスケーラビリティーを実現します。また、MPI と OpenMP* やインテル® PBB のマルチスレッド手法を組み合わせて、ハイブリッド並列コードの開発とチューニングも可能にします。

インテル® Cluster Studio 2011 は、すべてのインテル® アーキテクチャー用のインテル® C/C++ コンパイラーとインテル® Fortran コンパイラーに加え、インテル® クラスターツール群がすべて含まれたソフトウェア・パッケージです。Linux* や Windows* 上の並列アプリケーションの開発、解析、パフォーマンスの最適化に役立ちます。全コンパイラーとツールを 1 つのライセンスパッケージにまとめることで、クラスター・ソフトウェア・ツールで最上位クラスのシングル・インストール、相互運用性、サポートを提供します。

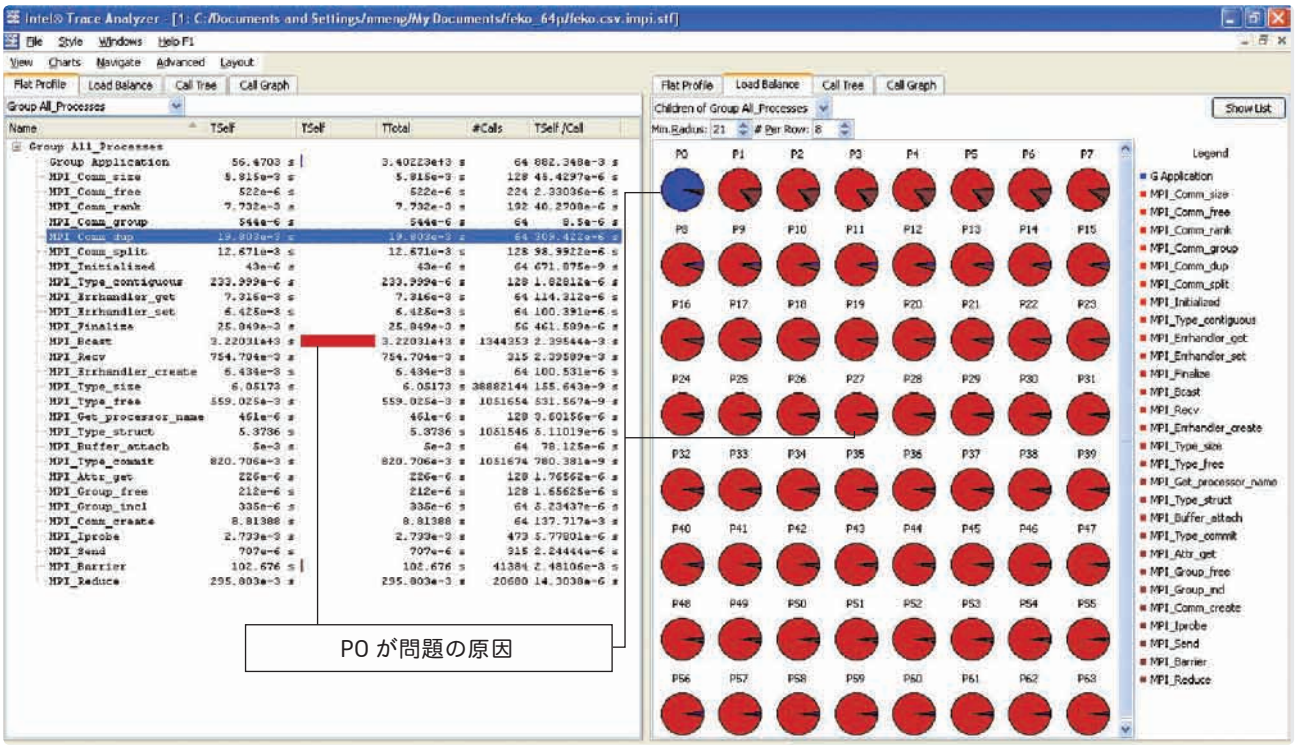


図 12

インテル® トレース・アナライザー / コレクター

- 並列アプリケーションの動作を分かりやすく視覚化
- プロファイリング統計と負荷のバランスを評価
- サブルーチンやコードブロックのパフォーマンス解析
- 通信 hotspot の特定

インテル® Cluster Studio 2011 の主な機能

- **スケーラビリティーとハイパフォーマンス：**マルチコア向けに最適化されたインターコネクト対応のインテル® MPI ライブラリーは、多数のインテル® アーキテクチャーおよび互換マルチコア・プロセッサーでアプリケーションのパフォーマンスを引き出します。
- **ビルトインの最適化：**インテル® Composer XE の最適化コンパイラーとライブラリーを使用することで、高度なプロセッサー・テクノロジーを最大限に活用することができます。C/C++ 最適化コンパイラーに含まれるインテル® PBB ライブラリーは、優れた信頼性を備え、並列化することで簡単に解決できる問題タイプを広げます。Fortran 開発者向けには、Fortran 2008 規格の Co-Array Fortran と一部の機能が追加されました。インテル® コンパイラーは、SIMD プラグマを使用した高度なベクトル化もサポートしています。
- **容易な MPI チューニング：**インテル® トレース・アナライザー / コレクターでは、MPI ベースのクラスター・アプリケーションの解析とチューニング・サイクルを向上させるための新機能が追加されています。
- **複数のオペレーティング・システム向けのアプリケーション：**インテル® コンパイラーとライブラリーで同一のソースコードを使用して、Windows* および Linux* 向けに高度な最適化が可能です。
- **インテル® クラスターレディー・プログラム：**このプログラムは、IA ベースの HPC クラスターでより長時間の稼働に対応し生産性を向上させる一方、総所有コスト (TCO) を削減するようなクラスター・アーキテクチャーを定義します。
- **互換性とサポート：**インテル® Cluster Studio は、主要な開発環境やコンパイラーと互換性があり、複数の世代のインテル® プロセッサーおよび互換プロセッサーをサポートしています。インテルでは、フォーラムやインテル® プレミアサポートを通じて幅広いサポートを提供しています。インテル® プレミアサポートでは、テクニカルサポートと、すべての製品のアップデートが 1 年間ご利用いただけます。

機能	利点
MPI 開発者向けの解析ツール — アプリケーションの負荷不均衡を示す図表や理想的なインターコネクト・シミュレーター	エラーの検出を容易かつ迅速に行い、MPI メッセージのパフォーマンス・プロファイリングを提供することで、開発者の生産性と効率性を高めます。
マルチレール InfiniBand* (IB) サポートとアプリケーション・チューナーを含むスケーラブルなインテル® MPI ライブラリー	業界で最もスケーラブルかつ堅固な MPI ライブラリーの 1 つで、数万個のコアまでスケーリングします。複数のクラスター・ファブリックにわたる動的な構成のサポートやマルチレール InfiniBand (IB) のサポートなど、利便性を備えています。
C/C++ コンパイラーとインテル® Parallel Building Blocks	SMP ノードのクラスターでアプリケーション・パフォーマンスを最適化するためのさまざまな柔軟性を備えた並列化タイプの選択肢（プロセス、タスク、データ、ベクトル）により、ブレイクスルーを図ります。C/C++ 規格がサポートされます。
クラスターでの Coarray Fortran (CAF) のサポートを含む Fortran 2008 規格の主要機能をサポートする Fortran コンパイラー（現在は Linux* のみ。Windows* は将来対応予定）	業界をリードするインテル® Fortran コンパイラーは、ノードおよびクラスターでスケーラブルな並列化を新しくサポートします。Fortran 2008 の主要機能および Fortran 2003 のより多くの機能をサポートしています。
アップデートされたパフォーマンス・ライブラリー — インテル® MKL とインテル® IPP	並列ライブラリーへ単純にリンクするだけで、よく使用される演算およびデータ処理タスクにおけるマルチコア・パフォーマンスを実現します。
Linux* および Windows* の両プラットフォームのサポート	開発を支援するさまざまな機能を持つ同一ツールセットを、Windows* および Linux* の両プラットフォームに提供 — 強化されたパフォーマンス、生産性、プログラム実現性を備えています。

図 13

まとめ

インテル® Parallel Studio XE およびインテル® Cluster Studio は、現在のマルチコアで優れたパフォーマンスを発揮し、将来のメニーコア向けにフォワード・スケーリングする必要のある Windows* および Linux* の C/C++、Fortran 開発者に次世代の開発ツールを提供します。

インテル® Parallel Studio XE 2011 には、最新バージョンの C/C++ および Fortran コンパイラー、インテル® MKL およびインテル® IPP のパフォーマンス・ライブラリー、インテル® PBB ライブラリー（インテル® TBB、インテル® Cilk™ Plus、インテル® ArBB（ベータ版））、インテル® Inspector XE メモリー / スレッド・エラー・チェッカー、インテル® VTune™ Amplifier XE パフォーマンス・プロファイラーが含まれます。

インテル® Cluster Studio 2011 には、最新バージョンのインテル® MPI ライブラリー、インテル® トレース・アナライザー / コレクター、インテル® C/C++ および Fortran コンパイラー、インテル® MKL およびインテル® IPP のパフォーマンス・ライブラリー、インテル® PBB ライブラリー（インテル® TBB、インテル® Cilk™ Plus、インテル® ArBB（ベータ版））が含まれます。

詳細は、次のサイトを参照してください : http://www.intel.co.jp/jp/software/products/

インテル® Parallel Building Blocks

並列化のパズル

解明の鍵

インテル コーポレーション
プログラム・マネージャー
David Sekowski

編集者からのメッセージ :

過去 5 年間で C++ 開発者から絶大なる支持を得たインテル® スレッディング・ビルディング・ブロック（インテル® TBB）は、多数のプラットフォームに移植され、さまざまなアプリケーションで使用されています。最近では有名な Adobe* 製品にも採用されました。ここでは、インテル® TBB を中心に拡張された並列モデルを紹介します。インテル® TBB の機能を拡大する補完的なモデルを使用して、インテル® Parallel Building Blocks（インテル® PBB）を理解し、使用することがいかに重要かを示します。

動機

マイクロプロセッサーのパフォーマンス・ゲインの主な要因がクロックスピードからマルチコアなどの機能に移行するのに伴い、アプリケーションを最適化してプラットフォーム機能を活用することの重要性が増しています。以前は、より高いクロックスピードを備えた新しいプロセッサーでは同一アプリケーションのパフォーマンスは自動的に向上していました。しかしながら、今日では最新のプロセッサーを購入しても、シリアルに記述されたアプリケーションや 1 つのプロセシング要素向けに記述されたアプリケーションでは、パフォーマンスの向上が得られない場合があります。別の方法でアプリケーションのパフォーマンスを引き出す必要にせまられたハードウェア設計者は、ハイパフォーマンス・コンピューティングから並列ハードウェア・プラットフォームへと移行しつつあります。これにより、並列ソフトウェアの開発経験がない大半のソフトウェア開発者は新たな課題に直面しています。

アムダールの法則が初めて発表された 1967 年以来、ハイパフォーマンス・コンピューティングの専門家はこの問題に取り組んできました。その結果、ソフトウェアを並列化してより多くのプロセッサーでジョブを処理することで、アプリケーションの合計実行時間が短縮できることが分かっています。1988 年に発表されたグスタフソンの法則により、アムダールの法則で暗黙的に定義されているスケーラビリティーの上限は排除されました。これらの法則を組み合わせることで、クロックスピードを上げることなく、メインストリーム・アプリケーションでソフトウェア・パフォーマンスを向上させる道が開けました。つまり、マルチコア・プロセッサー時代の始まりです。

現在および将来にわたって、最先端のアプリケーションは、最も一般的なメインストリーム・コンピューターに搭載されたデュアルコア、クアッドコア、そしてメニーコア・プロセッサーの強大な能力を利用するべく並列化へ向かうでしょう。Andy Grove は、その著書『Only the Paranoid Survive』で新しいテクノロジーが画期的なパフォーマンスの向上をもたらす場合、それがいかにビジネスの戦略的な転換点になるかについて語っています。この転換点こそが、業界に「進化」ではなく「革命」をもたらします。マルチコア、そしてメニーコア・プロセッサーは、メインストリームのソフトウェア・アプリケーション開発者にとって、新たなパフォーマンスと潜在的な機能を活用することができれば、まさにそのような機会といえます。

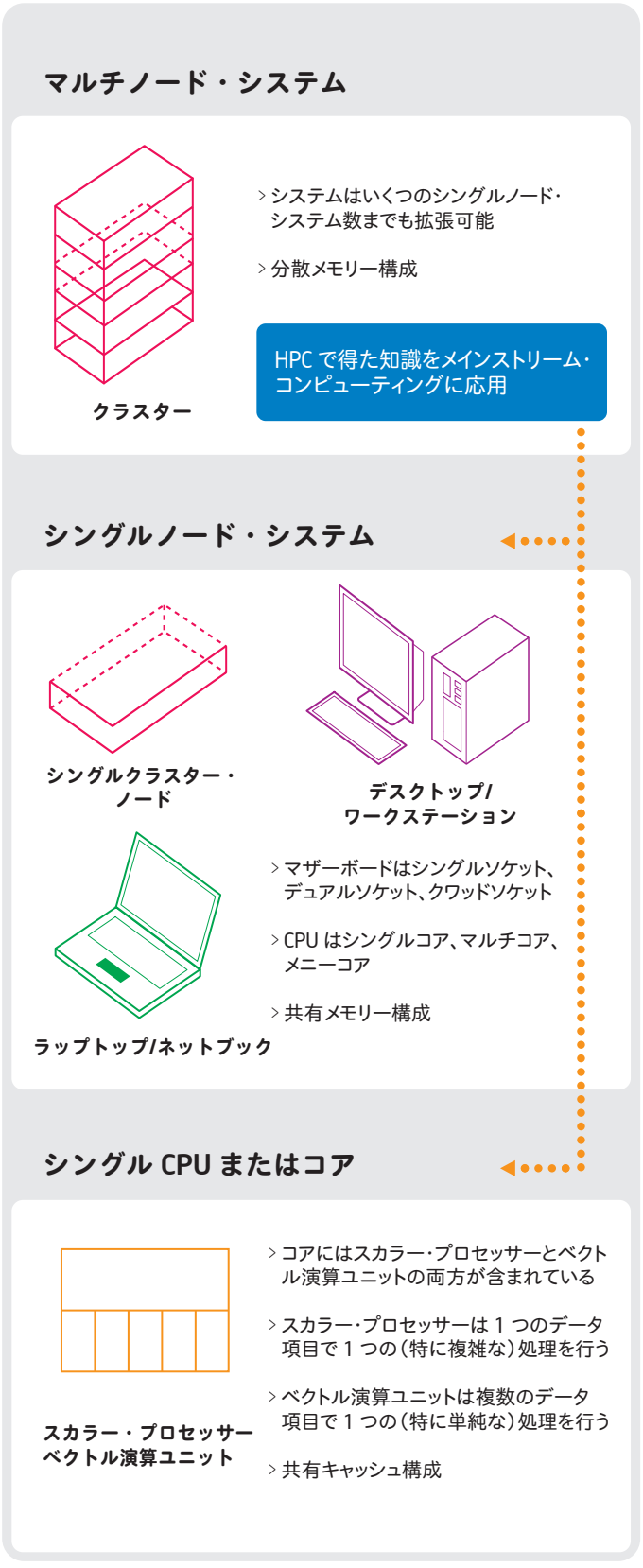


図 1

分散および共有メモリーシステム

ソフトウェアの並列化について理解する上で、最初に最高のパフォーマンスから最低のパフォーマンスまでのハードウェア・プラットフォームについて理解しておくと良いでしょう（図 1 を参照）。ハイエンドのプラットフォームを理解することで、その知識をメインストリームに応用できるからです。ハイパフォーマンス・コンピューティングでは、超並列ハードウェア / ソフトウェア・プラットフォームを使用して、気候変動からヒトゲノムの解読まで、世界最大級の問題の解明にあたっています。使用されるシステムは、インターネット経由で広域に分散したシステムのアイドルサイクルを使用するグリッド・コンピューターから、メッセージパッシングを使用して比較的近い場所にある複数のノードと通信するクラスター（例えば、インテル® メッセージ・パッシング・インターフェイス（インテル® MPI）ライブラリーを使用してクラスターノード間で情報を同期するもの）までさまざまです。どちらのシステムでも、共有メモリーではなく、分散システムの各ノードやワークステーション、ノートブックにあるような分散メモリーを使用します。分散メモリーシステムでは、ハイレベルでのノード間の調整とノード内の並列の調整が必要になります。通常、複数のノードの調整には、明示的なメッセージ・パッシング・モデルが使用されます。各プロセッサーにベクトル演算ユニットを備えたマルチコア、マルチスカラー・プロセッサーからなるマルチプロセッサーの出現により、ノードがさらに強力になったことで、ノード間のメッセージパッシングと各ノード内の並列ソフトウェアを組み合わせて使用しなければならなくなりました。

タスク並列化、データ並列化、ベクトル化

ノードレベルの並列化という用語は業界で広く使用されていますが、その意味はあいまいで、そのときどきによって意味が異なることがよくあります。ここでは、インテルのソフトウェア並列化ソリューションについて説明するために、主要な 3 つの並列化タイプを定義します（図 2 を参照）。

タスク並列化は、最高レベルのソフトウェア並列化です。一般に、タスク処理は不規則なデータセットを操作する、制御構造が不規則な問題で必要になります。タスク処理により、開発者はアプリケーションを論理的に分割することができます。例えば、ゲームエンジンではレンダリング・パイプライン、AI、物理計算、ネットワーク I/O モジュールなどのように分割できます。各論理要素は、タスクまたはタスクグループに割り当てられ、並行に処理されます。このほかにも、より一般的なタスク並列化として、メッセージパッシング、タスク処理、イベント処理、パイプラインがあります。アプリケーションの論理要素数は時間による変化がほとんどなく、グスタフソンの法則が適用されないため、一般にこれらの並列コードはプロセシング要素が追加されてもスケーリングしません。しかし、タスク並列化はアプリケーションのシリアルコード領域を減らすため、スケーラブルなソフトウェアを開発する上で重要な最初のステップであることに変わりはありません。

データ並列化は、タスク並列化を補完するものです。データ並列化で最もよく知られている実績あるソリューションのいくつかは、タスク並列プログラミング・モデルを使用して実装されています。データ並列化は、規則的な制御構造のアルゴリズムを使用して、コンカレント・コンテナーとその他の規則的なデータ構造を操作します。今日のアプリケーションの潜在的なスケーラビリティーのほとんどは、オーディオやビデオのエンコードなどの規則的な制御構造に存在します。通常、アプリケーションの論理要素を並べ替えてタスク並列化の効果を検証する前に、シリアルアプリケーションで for ループを並列 for ループに変更するなど、シリアル制御フロー構文の並列化を行うほうが簡単です。データ並列化とアルゴリズムの例として、並列ループ（「マップ」とも呼ばれる）、ソート、リダクション、スキャンがあります。

ベクトル化：タスク並列化とデータ並列化はどちらも、アプリケーションの処理をマルチコア・プロセッサーなどの複数のプロセシング要素に分散します。ベクトル化は、最新のマイクロプロセッサーのベクトル演算ユニットを活用するデータ並列化のサブセットです。ベクトル化（または SIMD (Single Instruction Multiple Data) とも呼ばれる）は、デー

並列化タイプ	メモリー構造	制御構造	アルゴリズム・タイプ	
タスク並列化	分散 共有	不定	メッセージ・ パッシング・タスク	イベント パイプライン
データ並列化	共有	一定	ループ ソート リダクション	ツリー グラフ リスト
ベクトル化	共有（キャッシュ）	一定	単純な配列 & ベクトル演算	要素関数 SIMD

図 2

タの複数の部分に対して単純な操作を並行して実行します。例えば、ベクトルの配列を複数追加するだけで、容易にベクトル演算ユニットを活用できます。科学、金融、その他の計算における演算カーネルのパフォーマンスに関するいくつかの報告では、算術ライブラリーでのベクトル演算ユニットの使用を強調しています。通常、実行時間のほとんどを単純なデータ並列操作に費やしているアプリケーションでは、ベクトル化により大幅なパフォーマンス向上が得られます。

コンパイラーとライブラリー

これまでにも、専門家や研究者によって、ソフトウェアの並列化を容易に実装するためのさまざまな方法が考案されてきました。現在では、全く新しい言語、既存の言語拡張、コンパイラーの自動機能、既存の言語とオペレーティング・システムの低レベル構造からなるランタイム・ライブラリーを利用することができます。インテルは業界のリーダーとして、これらをサポートするソフトウェア製品の研究、開発に取り組んでいます。現在、インテルでは主に次の 2 つの方法で、開発者の並列コード作成への取り組みを支援しています : (1) コンパイラー機能と言語拡張、(2) ライブラリー (図 3 を参照)。

特定のアプリケーションにおいて言語拡張を使用すべきかどうか、あるいはライブラリーを実装すべきかどうか判断する上で、それぞれによって得られる利点を考慮することは重要です。言語拡張はコンパイラーによって有効になり、スケジューリングの軽減や細粒度の並列化におけるパフォーマンス向上など、多くの利点があります。また、高レベルの抽象化が可能なため、並列処理の実装が容易になります。ただし、それと引き換えに、制御、汎用性、適応性が低下します。言語拡張は、キーワードや**プラグマ**を使用してコンパイラーにヒントを提供するだけなので、言語拡張だけでは並列コードの実装を明示的に制御することは難しいでしょう。

ライブラリーは既存のコンパイラーとインフラストラクチャーで使用できるものの、簡潔な標準の構文ではその機能を利用できないことがあります。ライブラリー・ベースのソリューションでは、一般にコンパイラー・ベースのソリューションよりも多くのオーバーヘッドが発生しますが、制御しやすく、より多くの機能を備えており、疎粒度の並列化では高いパフォーマンスを発揮します。コンパイラーとライブラリーはどちらも、並列化を実装するための有効な手段です。

コンパイラー拡張とライブラリーを使用した並列化の実装はともに、OS スレッドやロック・メカニズムなどのオペレーティング・システムの低レベル構造を使用します。そのため、知識と経験の豊富な開発者が時間をかけてコーディングした場合、特定のマシンのスレッドごとのパフォーマンスとシステムレベルのスケーラビリティーが、コンパイラー拡張やライブラリーによる抽象化と同じか、それ以上になることがあります。ただし、高レベルの抽象化を使用せずに、低レベルの構造を直接コーディングすることにより、アプリケーションの移植性と将来のスケーラビリティーが制限されます。インテルは、高レベルの抽象化を提供することで、開発者がこれらの制限を受けないようにします。

並列モデルの実装

比較項目	言語拡張	ライブラリー
コンパイラー	依存	非依存
標準規格への準拠	標準規格に準拠	通常は標準規格に準拠
移植性	低い	高い
使いやすさ	良い	悪い
ユーザーによる制御	少ない	多い
並列の粒度	細かい	粗い

図 3

プラグマ

プラグマとは、ソースコードにコンパイラーへの指示 (「ヒント」) を組み込むためのコンパイラー宣言子です。このヒントを解釈できる場合、コンパイラーは指示に従って特別なコンパイルを実行します。ヒントを解釈できない場合、コンパイラーはヒントを無視し、警告も出力しません。

インテル® Parallel Building Blocks		
選択時の条件		**付加価値**
抽象化 - OS に依存するスレッドや同期プリミティブを使用した並列化をできる限り排除	**ツールによるサポート** - インテル® Parallel Amplifier のパフォーマンス・チューニングやインテル® Parallel Inspector のデバッグなど、インテルのツール群で動作可能	**使いやすさ** - 実装と抽象化のテストが容易で、ハードウェアの並列化を活用できる
相互運用性 - アプリケーション内での共存と、確実なデータ交換が可能	**結合性** - 入れ子、または信頼性が高くパフォーマンスに優れた動作と組み合わせ可能	**信頼性** - 一般的なマルチスレッド・エラーを引き起こすことなく、さまざまな方法で組み合わせ可能
ハイパフォーマンス - スレッドごとのパフォーマンスと生産性に優れている	**スケーラブル** - プロセシング要素の追加に伴い、パフォーマンスがスケーリング	**将来性** - スレッドごとのパフォーマンスが十分な場合、プロセシング要素の追加に伴い、自動的にフォワード・スケーリング
柔軟性 - 開発者がどのプラットフォームでも使用できるように、必要に応じて複数のプラットフォームで動作可能	**オープン性** - 相互運用性を確保するために、標準化され公開されている	**移植性** - 柔軟なライセンシングで複数の OS、HW、IDE、コンパイラー・プラットフォームに移植可能

図 4

インテルの並列モデルファミリー

並列アプリケーションを作成するために、利用可能なさまざまな並列化タイプとその抽象化方法を理解することで、並列モデルを評価するためのフレームワークを築けます。インテルの並列モデルファミリーは、開発者にネイティブスレッド、自動ベクトル化、自動並列化、OpenMP*、OpenCL、その他多数の幅広いオプションを提供します。しかしながら、あらゆる状況に対応するソリューションというものはありません。個々のソリューションは状況によっては不向きなことがあり、開発者の求めるすべての並列タイプに対応できるわけではありません。

例えば、OpenMP* は、ハードウェア・システムをフル活用して処理を行い、解を返すような Fortran および C アプリケーションには最適です。しかし、ハードウェア・システム全体の制御を前提としているため、OpenMP* にはその構成、あるいはほかの並列手法との相互運用性において制約があります。OpenMP* は、ハイパフォーマンス・コンピューティングで解の算出に利用可能なリソースをフル活用する大規模な処理では優れたソリューションといえますが、ラップトップで実行する Web ブラウザー、メディアプレーヤー、メールクライアントなどの一般的なアプリケーションでは適応性が制限されます。このようにいくつかの制約はあるものの、インテルは、これからもコンパイラーで業界をリードする OpenMP* を継続してサポートします。

インテル® Parallel Building Blocks

インテルでは、一般的なアプリケーションを並列化する際の懸念事項に対応するため、インテル® TBB でタスクとデータの並列化を容易にしました。インテル® TBB は幅広い機能を備えた C++ テンプレート・ライブラリーで、タスクとデータの両方を並列化することができます。スケーラブルなメモリー割り当て、並列アルゴリズム、およびデータ構造、動的なタスク・スケジューリング、同期プリミティブ、移植性可能なスレッドを使用して、柔軟な構成と相互運用性を備えたノードレベルの並列化ソリューションを提供します。インテル® TBB を使用したタスクとデータの並列化は汎用性に優れていますが、コンパイラー・ベースのモデルや複雑な API よりも多くのオーバーヘッドがかかります。また、インテル® TBB では設計上、(1) 最新のプロセッサーのベクトル演算ユニットの活用、(2) インテルのメニーコア・コプロセッサーへの自動対応をサポートしていません。

これらの技術的な制約を克服するため、インテルの並列モデルファミリーに新たにインテル® Cilk™ Plus とインテル® Array Building Blocks (インテル® ArBB) が加わりました (図 5 を参照)。

インテル® Cilk™ Plus は、インテル® C/C++ コンパイラーで実装される C/C++ 言語拡張で、初めて並列プログラミングに取り組む場合に最も簡単な方法を提供します。この言語拡張は、タスクとデータの並列化を行うための簡単な 3 つのキーワードと、アプリケーションの一部を明示的にベクトル化するためのシンプルな構文からなり、シリアル・セマンティクスを保持し、決定性のある結果をもたらします。

インテル® TBB	オープンソース版（無料）	www.threadingbuildingblocks.org
	商用版（有償）	www.threadingbuildingblocks.com
インテル® Cilk™ Plus	一般情報	http://cilk.com
	Windows* の Microsoft* C++ コンパイラーのユーザーおよび Linux* の GCC コンパイラーのユーザー向けのインテル® Cilk™ ++ SDK	http://software.intel.com/en-us/articles/intel-cilk/
インテル® ArBB	一般情報	http://intel.com/go/ArBB
	ベータ版ダウンロード	http://software.intel.com/en-us/articles/intel-array-building-blocks

図 5

インテル® ArBB（旧インテル® Ct テクノロジー）は、インテル® TBB のようにハイパフォーマンスでスケーラブルなデータ並列化とベクトル化をもたらす C++ ライブラリーです。JIT（ジャストインタイム）コンパイラーで、メニーコア・プロセッサーとベクトル演算ユニットを含む、ターゲットの混合ハードウェア・プラットフォーム向けに動的に最適化します。CPU と（分散メモリーシステムの）アクセラレーターの両方で利用可能なマルチコア、マルチプロセッサー、ベクトル演算ユニットを活用する高度に最適化されたマシンコードを素早く生成します。動的にコードを生成することで、C++ のモジュール化に伴うオーバーヘッドも軽減します。例えば、インテル® ArBB は仮想関数をランタイムコストなしでサポートできます。インテル® ArBB の詳細については、次の記事を参照にしてください。ここで紹介した 3 つの並列モデル（インテル® TBB、インテル® Cilk™ Plus、およびインテル® ArBB）はすべて、共通のインフラストラクチャーを使用します。また、すべてのモデルは選択時の条件を満たしており、いくつかの付加価値も備えています（図 5 を参照）。これらのモデルは補完的なものであり、それぞれ特定のアプリケーション・コンテキストで優れた価値を提供します。3 つを組み合わせることで、言語拡張とライブラリーの両方を実装したインターフェイスを使用して、タスクの並列化、データの並列化、ベクトル化を行える統合ソリューションが得られます。この 3 つを組み合わせた並列化ソリューションは、インテル® Parallel Building Blocks（インテル® PBB）と呼ばれ、インテル® Parallel Studio 2011 やインテル® Parallel Studio XE 2011 に含まれています。インテル® PBB の各コンポーネントは、上記の表にあるリンクからご利用いただけます。

「インテル® ArBB（旧インテル® Ct テクノロジー）は、インテル® TBB のようにハイパフォーマンスでスケーラブルなデータ並列化とベクトル化をもたらす C++ ライブラリーです。」

インテル® Array Building Blocks

インテル コーポレーション
ソフトウェア・アーキテクト
Michael McCool

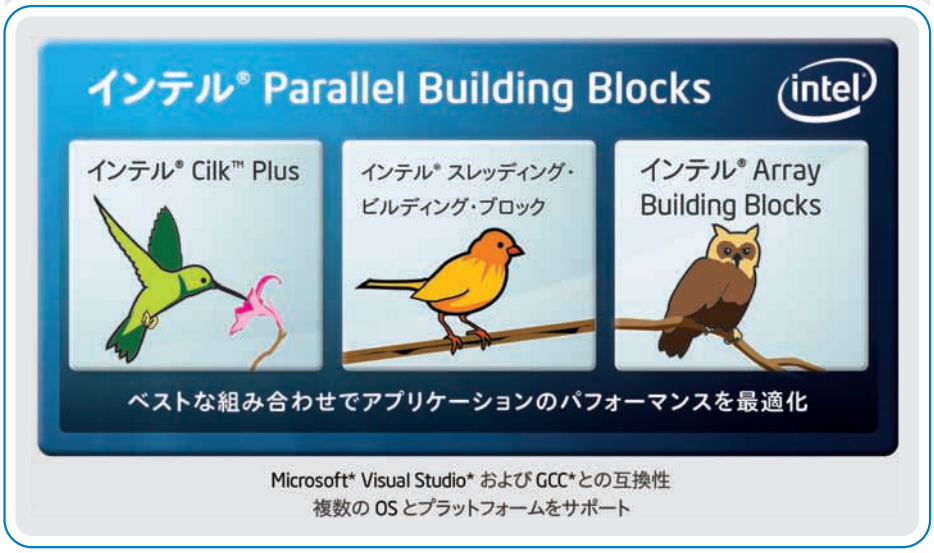
インテル® Array Building Blocks（インテル® ArBB）は、インテル® Parallel Building Blocks のコンポーネントで、移植性を備えたデータ並列ソフトウェア開発向けの強力かつ洗練されたプラットフォームです。並列プログラミングのためのバンドルされているその他のツールやライブラリーとともにご利用いただけます。

インテル® ArBB は、構造化された決定的なフレームワークで計算処理を多用するアプリケーションの並列化を支援します。強力なランタイムの汎用プログラミング・メカニズムを提供し、既存のコンパイラーで使用できます。特に、インテル、Microsoft*、および GNU の C++ コンパイラーでは動作検証されています。インテル® ArBB は現在ベータ版で、http://intel.com/go/ArBB から Windows* 版または Linux* 版をダウンロード可能です。是非、お試しになり、ご意見をお寄せください。

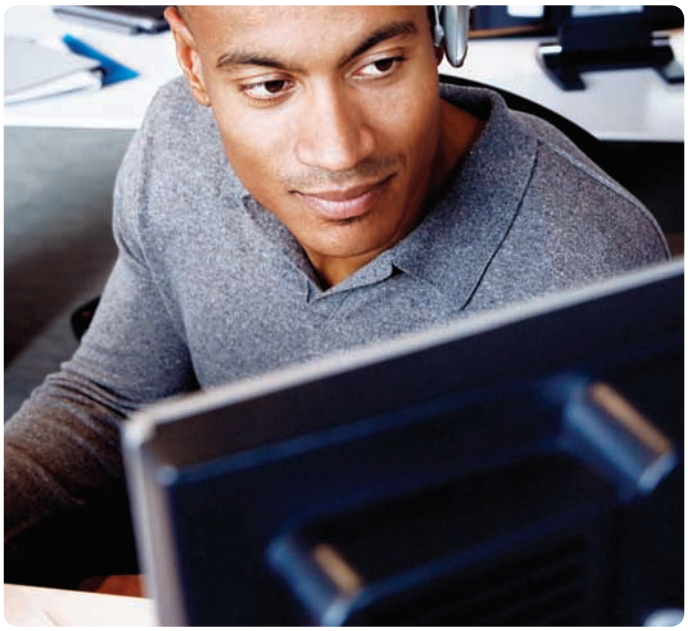
インテル® ArBB はプログラミング言語でしょうか？ それともライブラリーでしょうか？ その答えはどちらでもあります。インテル® ArBB は、ベクトル SIMD 命令を含む最近のプロセッサー・ハードウェアの並列メカニズムを、既存のプログラミング言語で利用するための移植性が高く一般的な方法で提供します。つまり、**組み込み言語**です。インテル® ArBB は API として実装された言語拡張です。ライブラリー・インターフェイスを備え、並列化およびベクトル化のマシン語を動的に生成し、最適化します。

最近のプロセッサーには、マルチコア、ハイパースレッディング、スーパースカラー命令の発行、パイプライン化、SIMD ベクトル命令など、並列化によりパフォーマンスを向上させるための多くのメカニズムが含まれています。最初の 2 つ、マルチコアとハイパースレッディングはスレッドを用いることで利用できますが、ハードウェア・スレッドを共有するオーバーヘッドの小さいタスクを使用したほうが効率的です。スーパースカラー命令の発行やパイプライン化などの命令レベルの並列化は、命令ストリームに不必要なデータ依存がない限り、プロセッサーにより自動的に行われます。最後の SIMD ベクトル命令による並列化は、一度に複数の操作を明示的に呼び出す特別な命令である SIMD 命令を生成することでのみ利用できます。SIMD 命令は、ベクトルの複数のコンポーネントに対して一度に同じ操作を行います。そのため、SIMD ベクトル命令と呼ばれることもあります。

インテル® Parallel Building Blocks（インテル® PBB）は並列処理に対するさまざまなアプローチをサポートし、並列プログラム開発モデルの総合セットを提供します。



SIMD ベクトル命令は非常に強力で、プロセッサーの進化とともにさらに処理能力が高まってきています。インテル® ストリーミング SIMD 拡張命令（インテル® SSE）対応のプロセッサーでは、1 つの SSE 命令で 4 つの単精度浮動小数点命令を実行することができます。また、インテル® AVX 対応プロセッサーでは、SIMD 命令の幅が倍増しているため、同様の命令を一度に 8 つ実行することができます。さらに、インテル® メニー・インテグレーテッド・コア (MIC) アーキテクチャーでは、SIMD 命令の幅がさらに倍増しているため、同様の命令を一度に 16 個も実行することができます。



「インテル® Parallel Building Blocks（インテル® PBB）には、移植性に優れたベクトル操作を行うための独立した 3 つのアプローチが含まれています。」

理論上では、プロセッサーの浮動小数点命令における最大パフォーマンスは、コア数、ベクトル演算ユニットのビット幅、クロックレートの積となります。将来、クロックレートが大幅に向上することは期待できませんが、コア数と各コアの SIMD ベクトルの幅は今後も増加し続けるでしょう。ベクトル化、つまり SIMD ベクトル命令を使用する計算は、最近のプロセッサーで最大限のパフォーマンスを引き出すために不可欠といえます。

ただし、ここで問題が 2 つあります。1 つ目は SIMD ベクトルユニットでは特定のマシン語のベクトル命令を使用する必要があること、2 つ目は SIMD ベクトル命令の拡張はプロセッサーにより異なることです。SSE、AVX、および MIC ベクトル命令はすべて異なります。AVX マシンは SSE 命令を実行できますが、SSE 命令だけでは AVX 対応プロセッサーのパフォーマンスを最大限に引き出すことはできません。2 つ目の問題はそれほど重大ではありません。現在のコンパイラー・テクノロジーでは、1 つのバイナリーで複数のコードパスを生成できるからです。例えば、インテル® C++ コンパイラーでは 1 つのソースプログラムから SSE と AVX の両方のマシン向けにコンパイルが可能で、生成されるプログラムは利用可能な場合に AVX コードを使用します。しかし、スタティック・コンパイラーではコンパイル時にターゲット・プロセッサーのセットを特定する必要があり、生成されるベクトル化コードがどの程度効率的なのかという疑問が残ります。

従来のアプローチでは、命令セットの拡張をサポートするために、新しい命令を出力するようコンパイラーを変更し、必要に応じてプログラムを再コンパイルしていました。このアプローチは、SIMD ベクトル命令にはあまり適していません。コンパイラーでプログラム中の SIMD ベクトル命令にマップ可能なシリアル構造を自動的に特定することは非常に困難です。場合によっては可能でしょうが、開発者が SIMD ベクトル命令を使用する場所とその方法を明示的に指定したほうが確実です。これには、信頼性の高いベクトル化を容易に行うための新しい構造がプログラミング言語で必要になります。残念ながら、C/C++ でベクトル化を指定するための、広く認められているマシンに依存しない標準規格はまだありません。

インテル® Parallel Building Blocks（インテル® PBB）には、移植性に優れたベクトル操作を行うための独立した 3 つのアプローチが含まれています。1 つ目のアプローチは、固定関数ライブラリーの使用です。これは、見過ごしてはならない重要なアプローチです。インテル® マス・カーネル・ライブラリー（インテル® MKL）とインテル® 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）には、すでにベクトル化された算術演算が多く含まれています。必要な操作がこれらの最適化されたライブラリーにある場合、通常はそれを利用するのが最良のソリューションです。そうでない場合、アルゴリズムをコーディングする必要があります。その場合、2 つのアプローチがあります。1 つ目は、配列のベクトル操作を明示的に指定するための表記を備えた、C/C++ の拡張であるインテル® Cilk™ Plus を使用することです。これは、インテル® C/C++ コンパイラーで利用できます。2 つ目は、汎用的なメカニズムであるインテル® ArBB を使用することです。

インテル® ArBB は C++ API として実装される組み込み言語で、理論的には ISO 準拠のどの C++ コンパイラーでも動作します。構文、データ・コレクションの型宣言、演算子の多重定義において標準の C++ メカニズムを採用し、コレクションに対する操作を表現できるようにしています。つまり、行列 - ベクトル算術ライブラリーのようなものです。ただし、トレードオフがあります。一般的なライブラリーでは C/C++ コンパイラーが静的にコードを生成しますが、インテル® ArBB ではライブラリー自体によって動的にマシンコードが生成されます。

インテル® ArBB の使用方法は、いくつかの例を後述しますが、非常に簡単です。例を理解するために、最初にいくつかの基本事項について説明します。インテル® ArBB の C++ API は型と操作の両方を定義します。型には、浮動小数点、整数、ブールのスカラー型、そしてこれらの型のコレクションを表す型、これらの型に基づくユーザー定義型があります。インテル® ArBB のスカラー型は一般的な C++ の浮動小数点型（float）や整数型（integer）の代わりに使用され、f32（単精度浮動小数点）、i32（32 ビットの符号付き整数）のような名前が付けられています。これらのスカラー型を使用すると、その操作に対応するマシン語は、C++ により静的に生成されるのではなく、インテル® ArBB により動的に生成されます。また、大規模なデータ・コレクションを管理するための型も用意されています。その中で最も単純なものは dense<T,D> で、要素型 T、次元 D の隣接して格納された（高密度な）多次元配列を表します。次元 D はオプションで、デフォルトは 1 です。要素型 T には、インテル® ArBB のあらゆるスカラー型、構造体、またはインテル® ArBB のスカラー型の要素を持つクラスを指定できます。

インテル® ArBB で並列コンピューティングを指定する方法は 2 つあります。コレクション全体に対する操作のシーケンスとして（ベクトルモード）、またはコレクションの各要素に対する複製された関数（要素モード）として指定できます。ベクトルモードは最も単純な方法です。コレクションに対する算術演算がそのコレクションの対応するメンバーに並列で適用されます。これは、要素がユーザー定義型で、演算子が多重定義されている場合でも使用できます。例えば、A、B、C、D という同じサイズの 4 つの dense<f32> 型のコレクションがある場合、次の式はこれらのコレクションのすべての要素に対して並列で処理されます。

```
A += (B/C) * D;
```

一般に、式の左辺と右辺の両方にコレクションがある場合、インテル® ArBB は結果が書き込まれる前にすべての入力が読み取られたかのように振る舞い、結果を出力します。実際には、この式を関数の中に配置し、call 操作で呼び出す必要があります。どのような並列ベクトル操作のシーケンスであっても関数の中に配置できます。

```
void
doit(dense<f32>& A, dense<f32> B,
dense<f32> C, dense<f32> D)
{
 A += (B/C) * D;
}
 ...
call(doit)(A,B,C,D);
```

実際には、call はこの式を 1 回だけ実行する関数を呼び出し、その関数により生成されるインテル® ArBB の型の生成、操作、破棄という一連の処理を監視します（実際に実行するわけではありません）。つまり、この一連の処理の記録、最適化されたマシン語へのコンパイル、（並列での）実行、キャッシュを行います。次回同じ関数が呼び出されると、call は再度 C++ 関数を呼び出す代わりに、キャッシュから内部的に生成されたマシンコードを取得します。これこそが、インテル® ArBB の単純な使用方法です。より高度な使用方法では、同じ C++ 関数から操作の別バージョンを生成します。例えば、一般的な C++ 変数と制御フローによりインテル® ArBB の操作シーケンスをパラメーター化し、それを使用して計算を行います。汎用プログラミングでこの強力なメカニズムを管理するには、クロージャーという別のインテル® ArBB の型を使用します。クロージャーは、キャプチャーされたインテル® ArBB 関数を表すオブジェクトです。概念的にはラムダ関数に似ていますが、動的に生成されます。call の戻り型は、実際には適切な型のクロージャーです。capture という別の関数も利用できます。この関数はクロージャーを作成するという点で call に似ていますが、キャッシュしないため、同じ C++ 関数を繰り返し呼び出して別バージョンを生成できます。前述のとおり、インテル® ArBB の単純な使用方法ではクロージャーを明示的に使用する必要はなく、call を通常の関数呼び出しと同様に見なすことができます。

インテル® Cilk™ Plus の仕様とランタイム ABI が無料でダウンロード可能に

インテル コーポレーションソフトウェア開発製品ディレクター JAMES REINDERS

インテル® Cilk™ Plus の仕様は、実装なくしては意味のない情報でしかありません。そのため、インテルではまず強力な実装を公開し、その直後に仕様の公開に踏み切りました。その結果、徹底的な評価、プロダクション環境での使用を経て、フィードバックを得ることができました。

インテルは、2010 年 11 月 2 日より、インテル® Cilk™ Plus の仕様とランタイム ABI の公開を cilk.com で開始しました。これは、すべてのコンパイラーでこれらの画期的な機能の採用を促進するための重要なステップです。インテルでは、インテル® Cilk™ Plus を普及させる最善の方法について関係各位と協議し、仕様の公開に踏み切ることが非常に重要であるとの結論に至りました。

しかし、実装を伴わない仕様だけでは、言語を普及させることは困難であり、徹底的に評価できる実装は不可欠であるため、まず最初に実装を公開し、その直後に仕様の公開に踏み切ったのです。

最新のインテル® コンパイラー Windows* 版および Linux* 版では、インテル® Cilk™ Plus を完全にサポートしています。これらのコンパイラーとインテル® Cilk™ Plus の仕様は、インテルが 2009 年に買収した Cilk Arts のエンジニア、経験、テクノロジーにより実現しました。

この JAMES の記事の続きは
Go-Parallel.com（英語）でご覧になれます。

Go Parallel では、このほか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）。

さらに、インテル® ArBB のスカラー型に対して、次のように要素関数を記述することもできます。

```
void
kernel(f32& a, f32 b, f32 c, f32 d)
{
 a += (b/c)*d;
}
```

要素関数は map 操作を使用することで call の中から呼び出せます。map 操作は入力コンテナーの各要素に対する関数を複製します。

```
void
doit(dense<f32>& A, dense<f32>
B, dense<f32> C, dense<f32> D)
{
  map(kernel)(A, B, C, D);
}
call(doit)(A,B,C,D);
```

また、要素関数内からその入力の近隣の要素にアクセスすることもできます。これにより、畳み込みなどのステンシル操作も非常に簡単になります。マップの各引数には、コンテナー全体**または**単一要素を渡せます。単一要素の場合は、引数として使用されているコンテナーのサイズと形状に一致するように複製されます。例えば、次のコードを考えてみましょう。

```
void
doit(dense<f32>& A, f32 b, f32 c,
dense<f32> D)
{
 map(kernel)(A, b, c, D);
}
call(doit)(A,b,c,D);
```

同じ kernel 関数を使用していますが、b と c の型 f32 が対応する関数の引数に一致しています。kernel の並列インスタンスは、コレクション A と D の要素数と同じだけありますが、それぞれのインスタンスで同じ値の b と c のコピーが作成されます。つまり、call の引数は完全に一致する必要がありますが、map は多相関数で、その引数には単一要素またはコレクションを指定できます。

並列操作を表現するこの 2 つの方法に加えて、インテル® ArBB ではコンテナー全体の処理を行ったり、コンテナー全体を入力値とするいくつかの集合操作を利用することも可能です。これらの操作は、コンテナーのコンテンツを移動したり、累積合計を取得したり（**prefix scan**）、一連の読み取り / 書き込み操作を実行したり（**スキャッター / ギャザー**）、要素を破棄し残りを連続シーケンスにパックしたり（**パック操作**。その逆は**アンパック操作** ）、または単純にすべての要素を 1 つの要素にまとめたりします。コンテナーのすべての要素を 1 つの要素にまとめることを**リダクション**といいます。例えば、次の式は 2 つのコンテナー A と B のドット積（ペア単位の積の和）を計算します。

```
f32 A_dot_B = add_reduce(A * B);
```

要素関数では制御フローも使用できます。前述のとおり、一般的な C++ 制御フローは関数が「キャプチャー」されたときに 1 回だけ実行されます。これは汎用プログラミングにおいて、別バージョンを作成したり、モジュール化と設定に伴うオーバーヘッドを減らすのに非常に役立ちます。ただし、制御フローをインテル® ArBB に見えるようにし、インテル® ArBB により生成されるベクトルマシン語にコンパイルするためには、特別なマクロを使用して「組み込み可能な」制御フローを表現する必要があります。

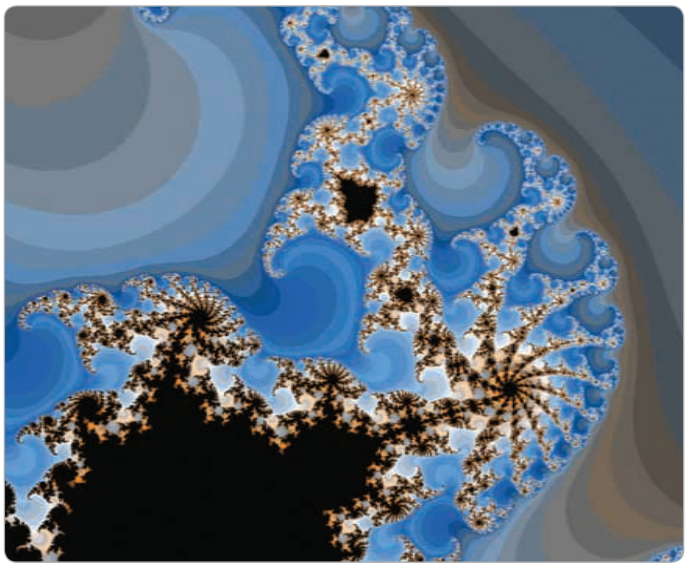


図 1

次にこの例を示します。このプログラムは、マンデルブロ集合を計算します。マンデルブロ集合とは、複素 2 次方程式が与えられた開始位置から分岐するために必要な反復回数をカウントすることで見つかるフラクタルです。この反復回数を複素平面に描画すると上記のイメージになります（図 1 を参照）。

このイメージの 1 つ 1 つのピクセルを計算する要素関数には、次のインテル® ArBB コードを使用します。

```
int max_count = MAX_COUNT;
void mandel(i32& d, std::complex<f32> c) {
 i32 i;
 std::complex<f32> z = 0.0f;
 _for (i = 0, i < max_count, i++) {
   _if (abs(z) >= 2.0f) {
     _break;
   } _end_if;
    z = z*z + c;
 } _end_for;
 d = i;
}
```

この例にはいくつかの興味深い点があります。まず、単純に std::complex 型とインテル® ArBB の要素型を使用するだけで複素数を表現しています。前述したように、これはユーザー定義型とユーザー定義型の演算子の多重定義でも行えます。演算子の多重定義は、ユーザー型のコレクションに適用されるベクトル操作でも利用できます。次に、この関数はローカルではない C++ 変数 max_count を参照しています。クロージャーを使用することで、この変数の値を変更して、別の値でこの関数をキャプチャーすることができます。call を使用すると、最初にこの関数を呼び出したときのこの変数の値をキャプチャーして「固定」できます。一方、capture を使用すると、この変数の値を変更して別バージョンをキャプチャーし、クロージャー・オブジェクトを使用して管理できます。最後に、インテル® ArBB の制御フローは、C++ の制御フローとやや異なることが分かります。インテル® ArBB の制御フローには先頭に下線があり、終了キーワード (_end_for など) が使用され、_for の引数はセミコロン (;) ではなくカンマ (,) で区切られています。

実際に処理を行うためには、インテル® ArBB コレクションへのデータの書き込み / 読み取りが必要になります。これはさまざまな方法で行えます。インテル® ArBB は、高度なアプリケーション向けに、イテレーター・ベースの STL フレンドリーで効率良いインターフェイスをサポートしています。ただし、最も簡単な方法は、bind を使用して、次のようにインテル® ArBB コレクションを C++ 配列に割り付ける方法です。この場合、ヘルパー call 関数「doit」を使用して map 内で要素関数を呼び出す必要があります。

```
void
doit(dense<i32,2>& D,
dense<std::complex<f32>,2> P)
{
 map(mandel)(D,P);
}
dense<std::complex<f32>,2> pos;
bind(pos, c_pos, cols, rows)
dense<i32,2> dest;
bind(dest, c_dest, cols, rows)
call(doit)(dest, pos);
```

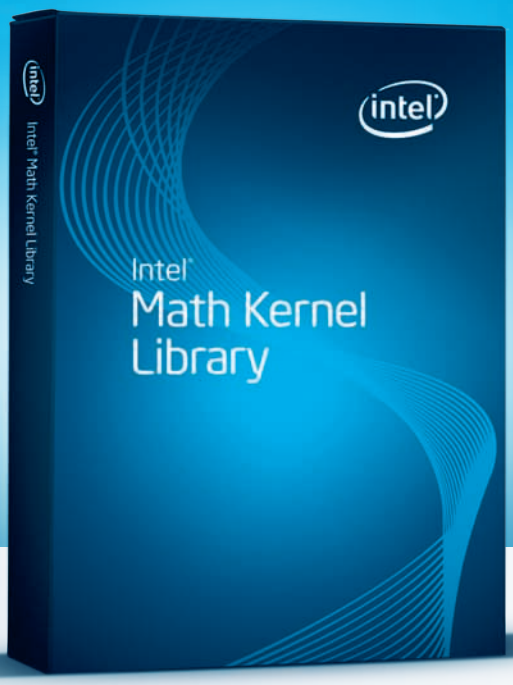
ここでは、インテル® Array Building Blocks について簡単に説明しました。インテル® ArBB は、効率良く高度なデータ並列計算処理を行うための移植性の高いメカニズムを提供します。プロセッサーとコンパイラーに依存しないベクトル命令を使用するために、インテル® ArBB は動的にコードを生成する機能を備えています。この機能により、インテル® ArBB のコード生成は「ホスト言語」である C++ とは別に行われ、多くの場合 C++ のオーバーヘッドを排除できます。インテル® ArBB は、効率的なデータ並列計算処理の実装を容易にし、汎用モジュラー・プログラミング向けにユニークで強力なメカニズムを提供する優れたツールです。

インテル® ArBB に関する詳細またはベータ版の評価については、http://intel.com/go/ArBB（英語）を参照してください。

自動並列化

インテル® マス・カーネル・ライブラリー（インテル® MKL）の使用

インテル コーポレーション
インテル® MKL アーキテクト **Greg Henry**
エンジニアリング・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）は、最適化および並列化された算術ライブラリー・ルーチンをソフトウェア開発者に提供します。スレッドセーフなルーチンは、工学、科学、金融、その他の分野のアプリケーションで幅広く利用できます。ここでは、最大限の並列化を行うために、インテル® MKL で採用されている手法と、スレッドにおける hotspot を解消するためのいくつかの方法を紹介します。

インテル® MKL には、デスクトップ、サーバー、クラスター向けのアプリケーション開発に役立つ多数のドメインが用意されており、業界スタンダードの BLAS (Basic Linear Algebra Subroutines)、最新バージョンの LAPACK (Linear Algebra PACKage)、高速フーリエ変換 (FFT)、ベクトル・マス・ライブラリー (VML)、ベクトル・スタティスティカル・ライブラリー (VSL)、直接法スパースソルバー (PARDISO)、スパース BLAS が含まれています。また、分散メモリー・アーキテクチャー（クラスター）におけるプログラミングを支援するため、インテル® MKL では ScaLAPACK、並列 BLAS (PBLAS)、クラスター FFT も提供しています。さらに、IA-32/ インテル® 64 アーキテクチャーのインテルおよび AMD* 製のハードウェア向けにチューニングされており、最新バージョンの Linux*、Windows*、Mac OS* X で利用できます。

インテル® MKL を使用する最大のメリットは、ソフトウェア開発者が最高レベルのパフォーマンスを容易に引き出せることです。ソフトウェア内部でディスパッチを自動化して、利用可能なハードウェア機能を活用できます。つまり、業界スタンダードの DGEMM などのサブルーチンを BLAS から呼び出す場合、アプリケーションの再リンクを行わなくても、さまざまなシステム上で優れたパフォーマンスを発揮できます。インテル® MKL はハードウェアを検出して、検出したプロセッサー向けに最適化されたコードをディスパッチするため、開発者の手を煩わせません。例えば、同じアプリケーションをインテル® MKL でリンクした場合、インテル® Core™ 2 Duo プロセッサーとインテル® Core™ i7 プロセッサー向けに最適化されたカーネルがすでに組み込まれており、ランタイムにディスパッチされるため、それぞれのプロセッサーで最適に実行されます。

マルチコアマシンはコンピューティング分野の最新のトレンドで、高度な並列化を可能にします。コア数が増加するにつれて、パフォーマンスのさらなる向上への期待が高まるのは当然といえますが、ソフトウェア開発者はその実現（並列化）に伴う課題への対応に直面しています。必要な並列化を効率良く簡単に得る方法として、一般的なルーチンの多くがスレッド化されているインテル® MKL などのライブラリーを使用する方法があります。

インテル® MKL は業界スタンダードの OpenMP* 仕様に基づき、直説法スパースソルバー、LAPACK、BLAS、スパース BLAS、VML、FFT、およびクラスター FFT ドメインのルーチンの多くがスレッド化されています。LAPACK や BLAS のような業界スタンダードのコンポーネントでは、パブリックドメインよりも高度なチューニングが行われています。例えば、LAPACK では線形方程式、直行因数分解、特異値分解、固有値問題のいくつかの計算ルーチンが最適化されています。すべてのケースでインテル® MKL はスレッドセーフなため、複数のスレッドがルーチンを同時に実行しても正しく動作します。

さらに、インテル® MKL では、ほかの OpenMP* ルーチンや環境変数に影響を与えることなく、インテル® MKL 固有の変数を設定することができます。例えば、MKL_NUM_THREADS を使用して OpenMP* スレッドの数を指定できます。インテル® MKL は、インテル® MKL 固有の変数を最初にチェックします。ただし、これは OpenMP* 環境により制約されます。インテル® MKL ルーチンと OpenMP* ルーチン / 変数のどちらも設定されていない場合は、OpenMP* のデフォルトが優先されます。特定の環境向けにアプリケーションをビルドし、OpenMP* 環境を制御する場合（環境変数を変更してテストできるようにしない場合）は、インテル® MKL のマルチスレッド・サービス関数を呼び出すことができます。このマルチスレッド・サービス関数は、環境変数よりも優先されます。インテル® MKL は、Microsoft* および GNU の OpenMP* ライブラリーに加え、インテル® コンパイラーの OpenMP* ライブラリーとも動作します。

開発者は、MKL_DOMAIN_NUM_THREADS 環境変数または対応するサービス関数 mkl_domain_set_num_threads() を使用して、インテル® MKL 全体だけでなく、ドメイン固有のレベルでスレッド数を制御できます。例えば、インテル® MKL では 2 つのスレッドを使用し、BLAS では 4 つのスレッドを使用する場合、それぞれの環境変数を “MKL_ALL=2, MKL_BLAS=4” のように設定します。すべての環境変数は実行時に一度だけ読み込まれます。実行中に動作を変更する場合は、サービス関数を呼び出します。

インテル® Cilk™ Plus ランタイム・ライブラリー、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）、pthreads（Linux*）など、OpenMP* 以外のスレッディング・モデルを使用してアプリケーションをビルドする場合は、最上位レベルのユーザーメソッドでスレッド化を行い、シリアルバージョンのインテル® MKL にリンクするか、マルチスレッド・バージョンのインテル® MKL では MKL_NUM_THREADS 環境変数を「1」に設定してください。

インテル® MKL のように、スレッド化はできるだけ高レベルで行ったほうが効果的です。例えば、LAPACK の当初の設計と現在公開されている実装は、BLAS ルーチン内の並列性に依存しています。ところが、弊社で行ったテストでは、LAPACK レベルでスレッド化を行ってシリアルバージョンの BLAS を呼び出したほうが、マルチスレッド・バージョンの BLAS を使用するよりもハイパフォーマンスが得られました。また、より高レベルでスレッド化を行ったほうが、スレッド数が増加するにつれてそのメリットが大きくなることも判明しました。LAPACK ルーチンの DGETRF（行の部分ピボットを用いたガウス消去法による倍精度一般行列の因数分解）において、メニーコアマシンで大きな行列サイズを指定し、BLAS レベルでのみスレッド化したものと、LAPACK ルーチンレベルでスレッド化したもののパフォーマンスを検証したところ、スレッド数が増えるにつれてその差は歴然となりました。

さらに、ノード間でメッセージ・パッシング・インターフェイス (MPI) を使用する分散メモリークラスターのように、複数の異なる並列化を活用したほうが良い場合があります。インテル® MKL の MP LINPACK ベンチマーク (DGETRF に似たクラスター問題の解の算出) では、より優れたパフォーマンスを達成するために MPI-OpenMP* によるハイブリッド並列化を採用しています。これは、前述の高レベルのスレッド化に通じるものがあります。クラスターにおける並列化の最も基本的なメカニズムでは、1 コアにつき 1 つの MPI プロセスを実行しますが、コードに OpenMP* 呼び出しを追加してより少ない MPI プロセスで実行することで、アプリケーションのより高いレベルでスレッド化が行われ、パフォーマンス・ゲインを達成できます。

インテル® MKL は、スレッド数を特定するのに OpenMP* ソフトウェアに依存しています。マルチスレッド向けに初期化されていない MPI を検出すると、インテル® MKL はスレッド数を 1 に設定します。同様に、OpenMP* の並列領域内から呼び出された場合もスレッド数は 1 に設定されます。OpenMP* で物理コア数を超えるスレッド数が使用されている場合（ハイパースレッディングが有効な場合など）、インテル® MKL はスレッド数を物理コア数と同じになるように減らします。場合によっては、インテル® MKL により設定されるスレッド数が最適でないことがあります。その場合は、MKL_DYNAMIC を FALSE（デフォルトは TRUE）に設定するか、mkl_set_dynamic() を呼び出して最適なスレッド数に変更します。FFT にはセットアップと実行の 2 つの段階がありますが、両方の段階でスレッド数は同じでなければなりません。

インテル® MKL の自動並列化を活用することで、最近のマルチコア・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さらに向上できるでしょう。

関連 Web サイト（すべて英語）:
[BLAS] http://www.netlib.org/blas/index.html
[LAPACK] http://www.netlib.org/lapack/index.html
[MKL] http://software.intel.com/en-us/intel-mkl/
[MPI] http://www.mcs.anl.gov/research/projects/mpi/
[MPI] http://www.intel.com/go/mpi
[OPENMP] www.openmp.org
[SCALAPACK] http://www.netlib.org/scalapack/index.html

print ステートメントとタイマーだけでは見つけられない問題の検出：

より有効的な並列化への投資

Don Gunning、
Nick Meng、Paul Besl

インテル® アーキテクチャーの進化とともに、クラスター・ソフトウェアでは次なる投資として、よりいっそうの並列化の必要性に迫られています。これに対応して、ソフトウェア開発者は、パフォーマンスを大幅に向上させるようソフトウェアの変更を求められています。通常、この変更には長い年月がかかり、複数の並列手法を組み合わせた実装が採用されます。まれに、比較的小さな変更で済む場合もありますが、どちらの場合でも、これまでの経験から変更点は明白であるとはいえません。

インテル® クラスター・ツールキット・コンパイラー・エディションの高度な機能、特にインテル® MPI 4.0 の新機能とインテル® トレース・アナライザー / コレクターを活用することで、この変更に伴う課題に取り組むことができます。

ここでは、実際の開発者がどのようにインテル® トレース・アナライザー / コレクターを使用して、print ステートメントとタイマーでは見つけられない問題の検出、修正を行い、インテル® MPI 4.0 でパフォーマンスを 30 ～ 50 % 向上しているかを紹介します（例えば、LSTC の dyna は 1,500 を超えるコアで、fluent は 3,000 を上回るコアでスケーリングしています）。これらのツールを使用することで、知識の豊富な開発者は、従来のツールでは何年もかかっていた結果を、何カ月かで得られるようになるでしょう。

最初に、既存のソフトウェアを新しいシステムで使用したり、大規模なデータセットで使用した際に直面する問題で、インテル® クラスター・ツールキット・コンパイラー・エディション 4.0 を使用する方法を見てみましょう。

次に、予測したスピードアップが得られなかった場合、インテル® トレース・アナライザー / コレクターを使用して問題の診断を行い、複数の並列手法を組み合わせた実装で結果の質を確保しながら大規模なデータセットを処理する方法を紹介します。

資金を投資したにもかかわらず、アプリケーションが遅くなりました。

ソフトウェア・ユーザーは、新しいインテル® マルチコア・アーキテクチャーのワークロードへの影響について、よく ISV へ問い合わせをします。通常は、図 1 の黄色い点線のように、非常に好ましい予測が得られます。しかし、実際に資金を投資して、新しいインテル® 64 システムにアプリケーションをインストールしてみた結果、図 1 の赤い線のようになることは珍しくありません。

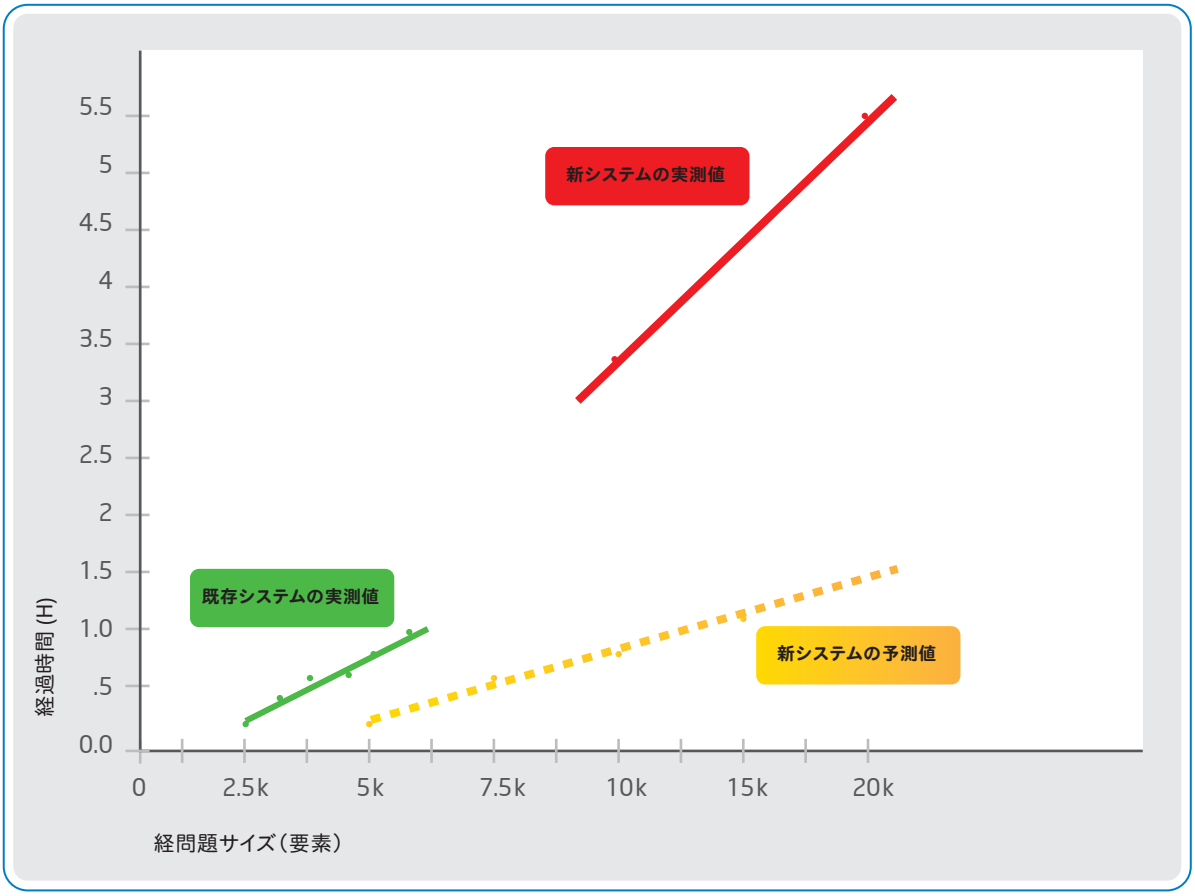


図 1. 実用的なテストとパフォーマンス予測

```
source /opt/intel/itac/8.0.1.001/bin/itacvars.sh
export LD_PRELOAD=/opt/intel/itac/8.0.1.001/slib/libVT.so
# 8p run
runexec small_model.pre -np 8 --mpi-options -trace --machines-file $PBS_NODEFILE
#256p run
runexec large_model.pre -np 256 --mpi-options -trace --machines-file $PBS_NODEFILE
```

言うまでもなく、なんらかの問題があることは明らかで、問題究明のために診断が必要になります。

インテル® トレース・アナライザー / コレクターを使用すると、この問題の診断を簡単に行うことができます。基本的には、MPI コマンドラインに “-trace” オプションを追加するか、実行スクリプトファイルに “-mpi-options -trace” オプションを追加するだけです。これで ITAC ツールを使用できます。

ここでは、インテル® トレース・アナライザー / コレクターを使用して、8 コアの小規模なテストケースと 256 コアの大規模なテストケースの 2 つを検証した結果について見てみましょう。まず、STF トレースファイルを取得した後に、アナライザー・ツールを使用して統計データを収集します。図 2 と 3 は、小規模なテストケースで収集した統計データをグラフィック・モードで表示したものです。図 2 では、P0 から P1 ～ P7 への通信量が多く、中でも P0 から P1 への通信量が最も多いことが分かります（線で囲まれている部

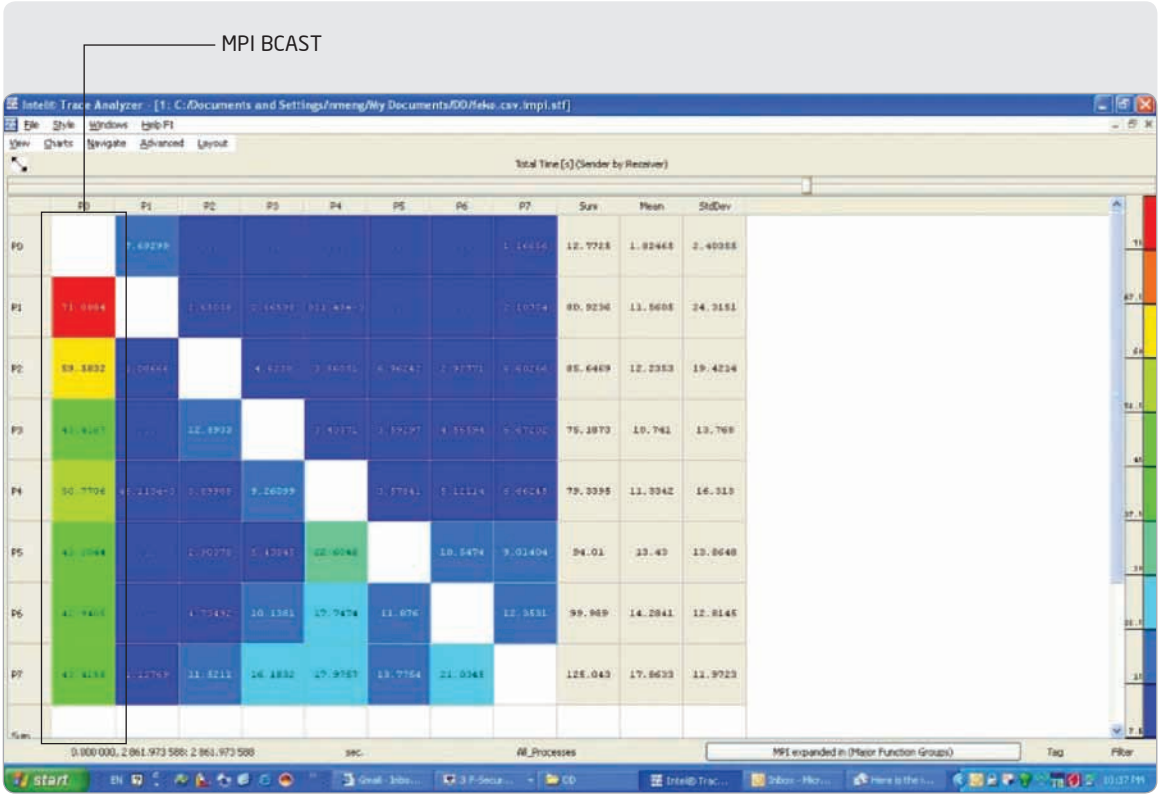


図 2. インテル® トレース・アナライザー / コレクターにより生成された各 MPI プロセスの MPI 集合関数の合計時間
（赤 : 時間のかかっている関数 [最もビジー]、青 : 時間のかかっていない関数 [あまりビジーではない]）

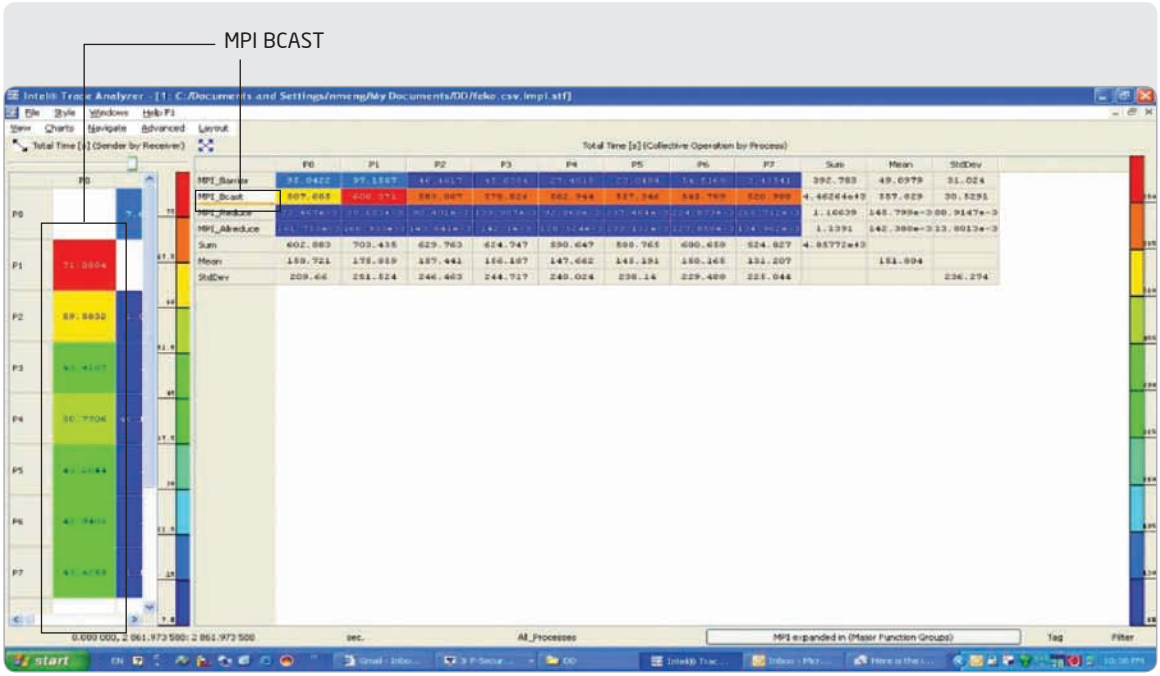


図 3. インテル® トレース・アナライザー / コレクターにより生成された各 MPI プロセスの MPI 集合関数の合計時間
（赤 : 時間のかかっている関数、青 : 時間のかかっていない関数）

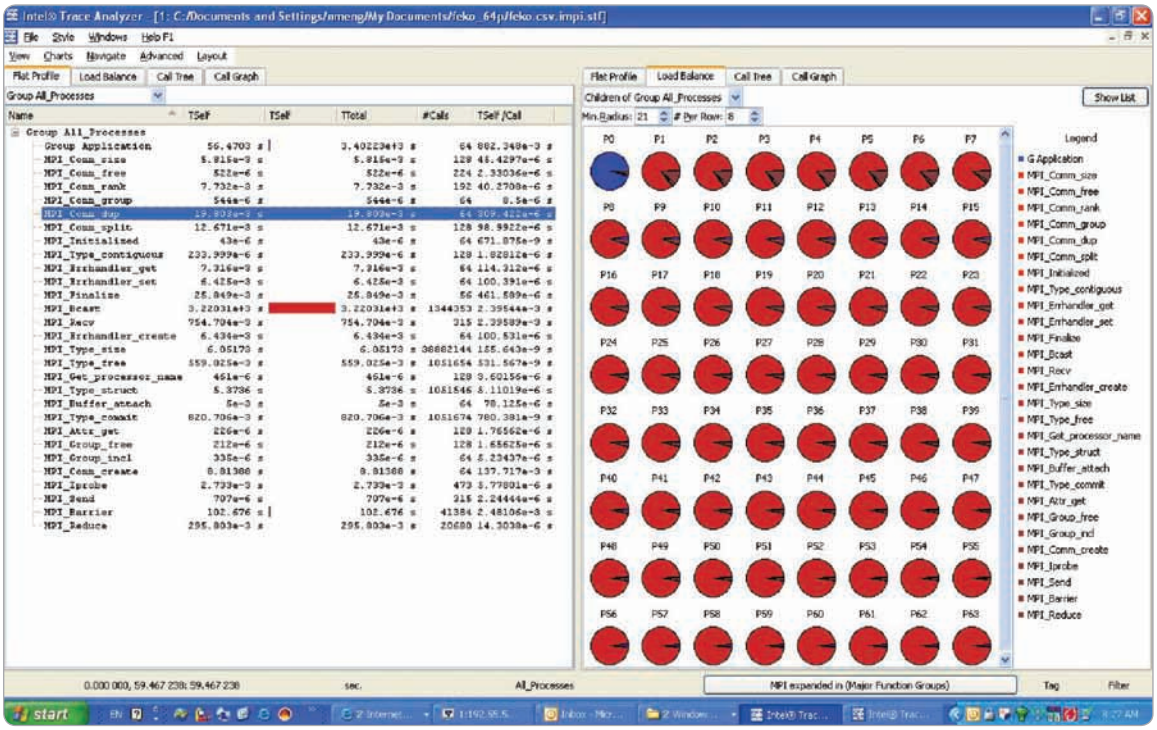


図 4. インテル® トレース・アナライザー / コレクターにより生成された 256 コアの大規模なテストケースにおけるロードバランスを円グラフで表示したもの（赤 : MPI コードの実行時間の割合、青 : アプリケーションの実行時間の割合）

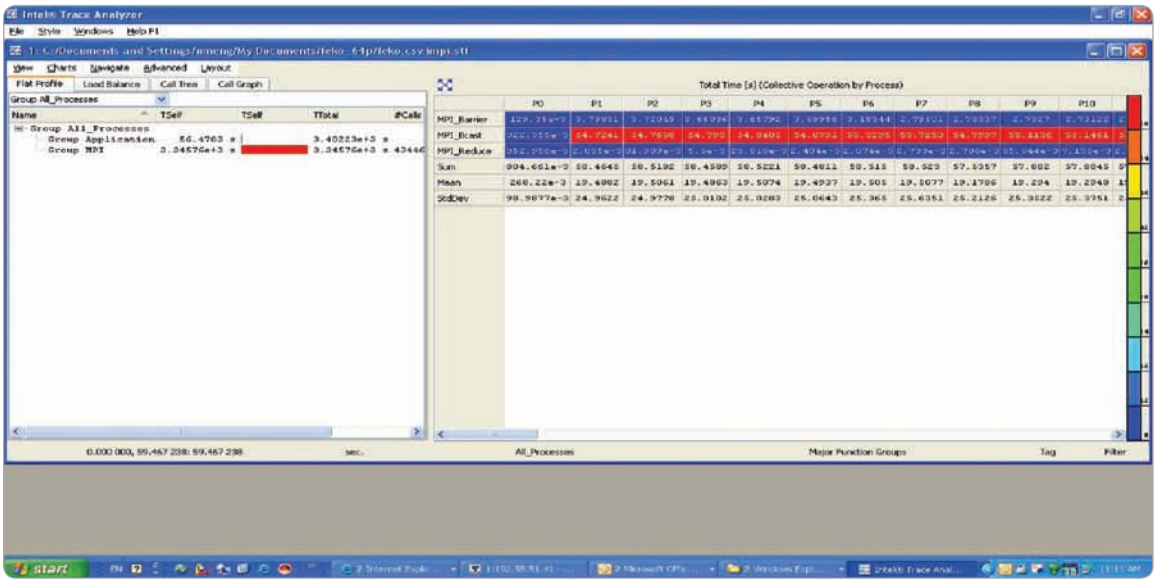


図 5. 大規模なテストケースのプロファイル・データとインテル® トレース・アナライザー / コレクターにより生成された各 MPI プロセスの MPI 集合関数の合計時間

分を参照）。

図 3 では、MPI_Bcast がこのテストケースで通信量が最も多い関数であることが分かります（線で囲まれている部分を参照）。256 コアの大規模なテストケースでより詳細なパフォーマンスの調査を行ったところ、図 4、5、6、7 のようなデータが得られました。

図 4 は、関数の実行時間の割合を円グラフで表示したものです。青い部分はアプリケーションの計算コスト、赤い部分は MPI 通信コストを表します。マスター MPI プロセスの円グラフはほとんどが青色で、スレーブ MPI プロセスの円グラフはほとんどが赤色です。つまり、非常に深刻な負荷不均衡が発生しています。図 6 と 7 から、この原因は MPI_BCAST 関数にあることが分かります。図 6 と 7 は、MPI 関数とアプリケーション関数のアクティビティーを表したものです。赤い部分は MPI 関数のアクティビティー、青い部分はアプリケーション関数のアクティビティーを表し

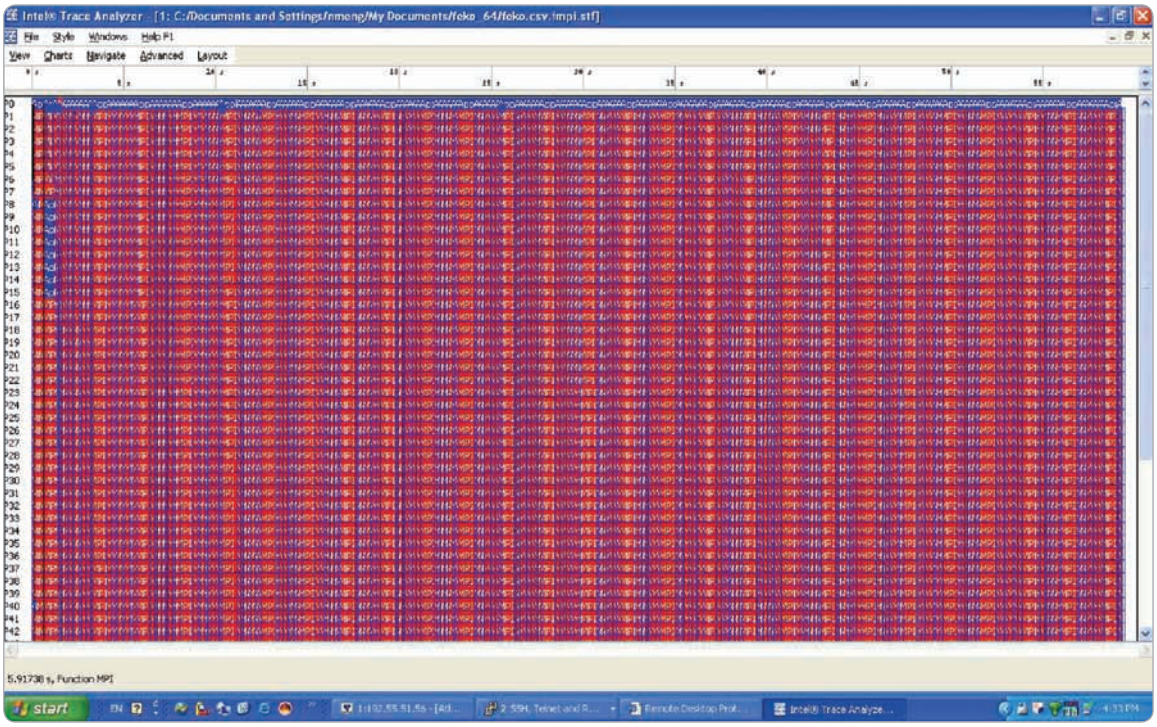


図 6. インテル® トレース・アナライザー / コレクターにより生成された 0 ～ 60 秒の間の関数のアクティビティー

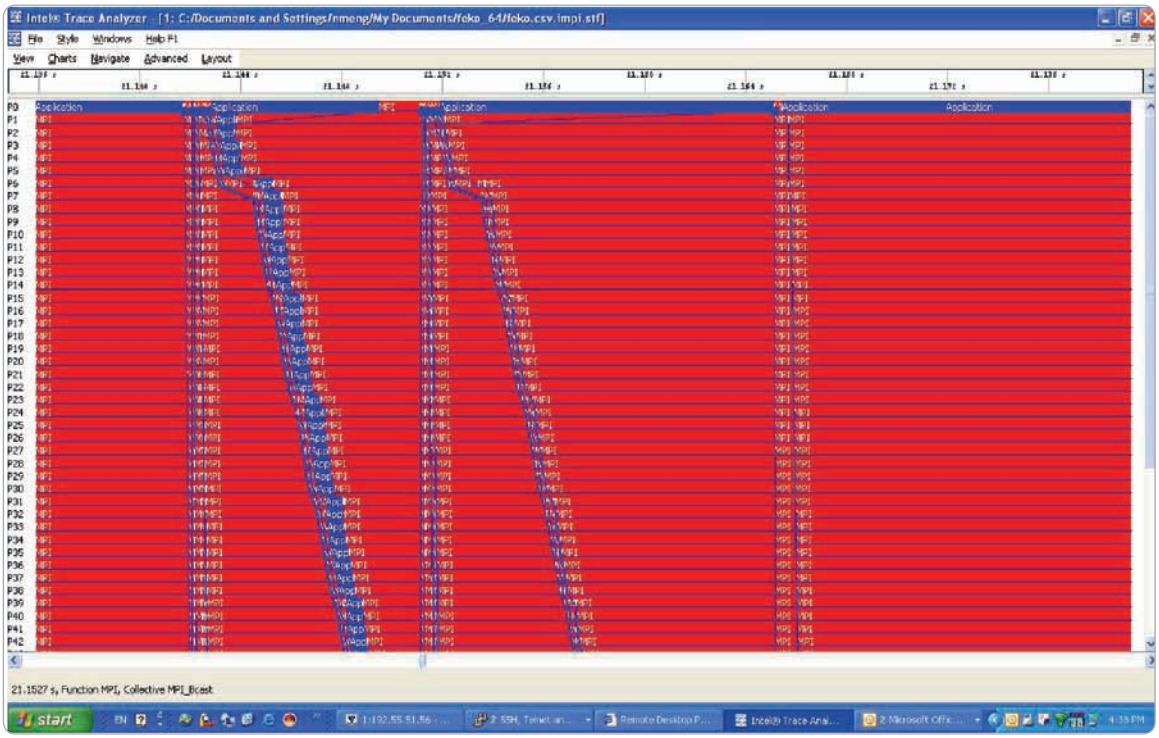


図 7. インテル® トレース・アナライザー / コレクターにより生成された 21.136 ～ 21.176 秒の間の関数のアクティビティー
（赤 : MPI コード、青 : アプリケーションコード。横軸は時間、縦軸は MPI プロセス / ランク。）

ます。大規模なテストケースでは、実行時間の多くが MPI 関数に費やされているのが一目瞭然です。これは極めて異常なことです。

前述のとおり、図 4 ではマスター MPI プロセス（左上の円グラフ）は青色で、すべてのスレーブ MPI プロセスはほとんど赤色で表示されています。また、図 5、6、7 から、負荷不均衡の原因は MPI _BCAST 関数であることが明白です。つまり、MPI_BCAST 関数のデフォルト設定（アルゴリズム）が、このテストケースには合っていないため、適切なアルゴリズムを選択する必要があります。小規模なテストケースでいくつかの設定をテストしてみたところ、このテストケースに最適な MPI_BCAST の設定は、I_MPI_ADJUST_BCAST=4 であることが分かりました。

このケースでは、インテル® トレース・アナライザー / コレクターのグラフィカル・データ・マイニング機能と ISV 開発者のソフトウェアに関する豊富な知識により、予期しない問題の原因を素早く見つけ、解決策を簡単に実装することができました。その結果、新しいインテル® 64 プラットフォームで合理的なパフォーマンスを得られました。

次に、より複雑な課題に目を向けてみましょう

複数の並列手法による複雑なタスクの解決

Livermore Software Technology Corporation (LSTC) では、大きくなり続けるデータセットをより高速に処理しなければならないという課題に常に直面しています。それだけでなく、多くの場合は結果の精度も求められます。LSTC では共有メモリー版と分散メモリー版のソフトウェアを提供していますが、共有メモリー版よりも分散メモリー版のほうがスケーラビリティーに優れています。共有メモリー版では OpenMP* を使用し、分散メモリー版では MPI を使用しています。

ここでは、インテル® トレース・アナライザー / コレクターを使用することで、インテル® マルチコア・アーキテクチャーにおいて、LSTC DYNA でこれまでよりもずっと大きなデータセットの処理を可能にした方法を紹介します。

課題：

LSTC では、複雑な実社会の問題のシミュレーションを行うことができる、動的陽解法有限要素法による汎用プログラム LS-DYNA を提供しています。これは、自動車、航空宇宙、一般消費財、バイオエンジニアリングの製造で使用される衝撃シミュレーション・ソフトウェアです。ソリューションの精度を向上するために、問題サイズはますます大きくなっており、コンピューターの処理時間も長くなる一方です（例えば 1,000 万要素の問題の処理には、クラスターで 43 時間かかります）。

LSTC は、限られたネットワーク帯域幅、固定されたノードメモリー、限られたメモリー帯域幅で、数値精度も維持しなければならない必要性に迫られました。通常、これらの条件下では、一定のノード数を超えると問題サイズの増加とともにスケーリングできなくなるため、実行時間が大幅に増加してしまいます。

DEVELOPER SPOTLIGHT

上野浩太郎
アプリケーションエンジニア
ソフトウェア & サービス

これまでの経歴

UNIX System V オペレーティング・システム開発チームでソフトウェア開発エンジニアとして 10 年間、インターナショナライゼーションとローカライゼーション、カーネル開発、ドライバーのデバッグ、チューニング、テクニカルサポート、ネットワーク管理に携わりました。

インテルには 8 年以上在籍しており、シニア・エンジニアリング・コンサルタントとして 6 年以上にわたってインテル® ソリューション・サービス (ISS) でエンタープライズ向け IA システム（32 ウェイ IA サーバー）のデータベースとソフトウェアのチューニング、エンタープライズ向け IA システム（32 ウェイ IA サーバー）の ERP とソフトウェアのチューニング、Sparc* ベースの Sun* Solaris* システムから IA ベースの Linux* システムへのソフトウェアの移植など、多数のプロジェクトに携わりました。

現在の取り組み

日本の大手ゲーム・ソフトウェア企業と共同で、有名なゲームコンテンツを使用して Larrabee（開発コード名）アーキテクチャーを有効活用するべく取り組んでいます。

この取り組みの重要性

これは、ビジュアル・コンピューティング分野において新しいグラフィックス技術を先導するという、インテルにとって技術的に大きな挑戦です。インテルでは、このプロジェクトを通して、既存のレンダリング・パイプラインと Larrabee アーキテクチャーのネイティブ・プログラム・コードによりプログラミング可能なアプローチを組み合わせた、新しいレンダリング技術を普及させるべく努めています。

ソフトウェア開発者としての目標

インテル® プロセッサーおよび IA-32、インテル® Itanium®、Larrabee アーキテクチャーなどのインテル® アーキテクチャーにおけるソフトウェア開発技術を牽引していくことです。最高のパフォーマンスを引き出すために、すべてのソフトウェアがそれぞれのアーキテクチャー向けに最適化されるべきだと考えています。

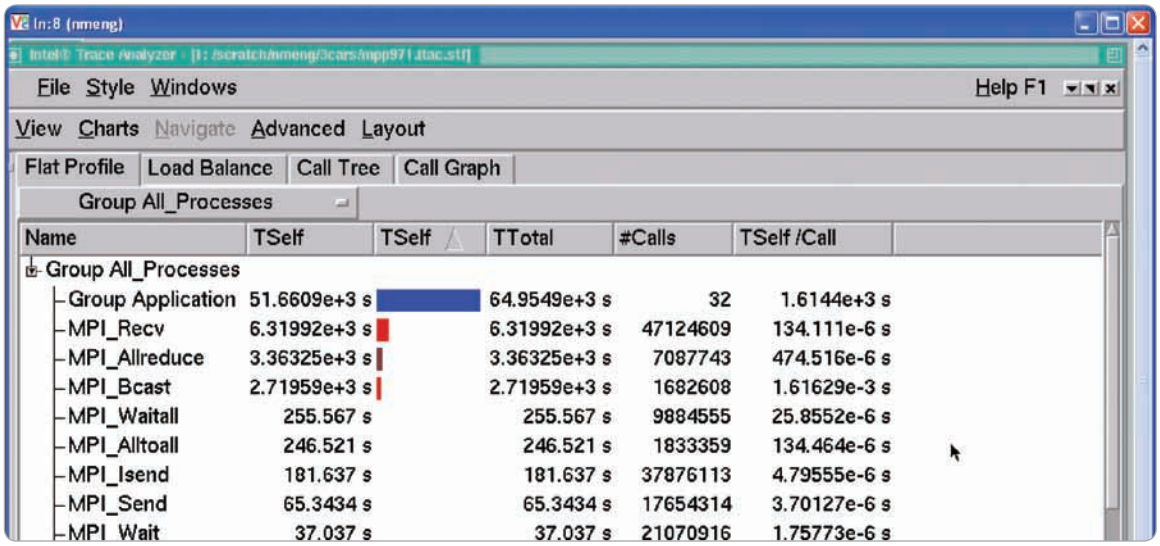


図 8. インテル® トレース・アナライザー / コレクターにより生成された 4 ノードのモデルのプロファイル・データ

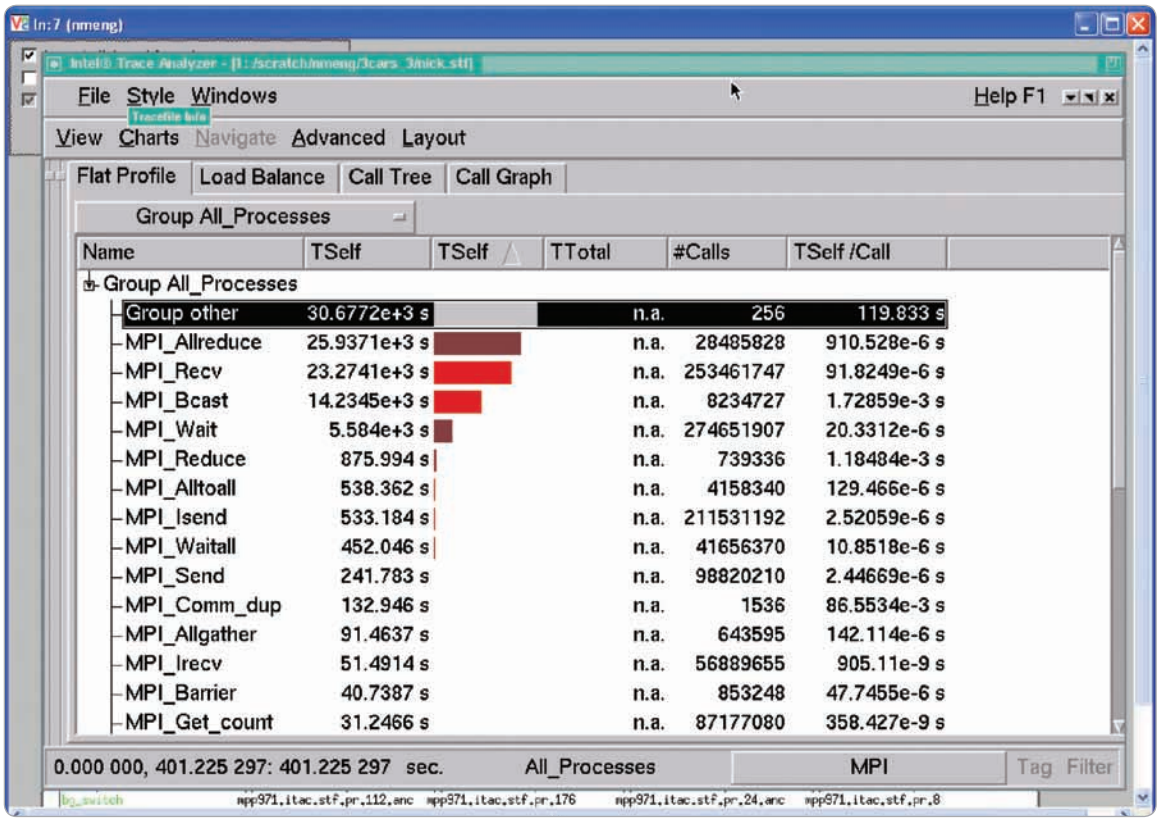


図 9. インテル® トレース・アナライザー / コレクターにより生成された 32 ノードのモデルのプロファイル・データ

クラスター構成	インテル® Xeon® プロセッサー 7560 1 ノード、32 コア	インテル® Xeon® プロセッサー 5560 ベースのクラスター 8 ノード、1 ノードにつき 8 コア（計 64 コア）
MPP (MPI) 版の経過時間	44013 秒	18521 秒
MPI と OpenMP* のハイブリッド・バージョンの経過時間	7047 秒	5541 秒
スピードアップ	6.25	3.34

図 10. LSTC 標準の暗黙的なベンチマーク・モデル CYL1E6

課題への取り組み：

LSTC とインテルは共同で、インテル® クラスターツールを使用して、大規模で複雑な問題のハイブリッド・スケーリングに取り組みました。以下に、インテル® トレース・アナライザー / コレクターを使用して、print ステートメントとタイマーでは見つけられない問題を検出する方法を紹介します。ここで重要なのは、数年ではなく、数カ月で 100 を超えるルーチンを検証し、ソリューションを見つけることができたことです。

MPP DYNA のパフォーマンス

MPI プロセスの数が増加するにつれて、サブドメインの数と通信コストも増加します。これは負荷不均衡を引き起こし、大きな通信オーバーヘッドにつながります。その結果、並列処理の効率が悪くなります。

LSTC では、インテル® トレース・アナライザー / コレクターを使用することで、この問題を効率良くピンポイントで検出することができました。**図 8** と **9** から、MPI 集合関数のパフォーマンスは 4 ノードでは発揮されているもの、32 ノードでは低下していることが分かります。

この 2 つの図から、各ノード内では OpenMP* を使用してすべてのコアを活用し、ノード間では MPI を使用する必要があることが見えてきます。そうすることで、MPI プロセスの数をできる限り抑え、MPI 関数のオーバーヘッドを減らすことが可能です。この変更は、100 を超えるルーチンに対して適用されました。

ソリューション：

LSTC では、LS DYNA の共有メモリー版と MPP DYNA を組み合わせることで HYBRID LS-DYNA が生まれました。この並列版では、次の機能を実現しています。

ユーザーの求める一貫した数値精度を備えたソリューション

› OpenMP* コードと MPI コードを 1 つに組み合わせたことで、一定の条件下で OpenMP* コードの一貫した機能を活用できるようになりました。その結果、品質の高い一貫した数値精度の結果を実現できました。

LS-DYNA のスケーラビリティーとインテルのマルチコア・ノード・アーキテクチャーにおける効率性の向上

› MPI 関数のオーバーヘッド・コストが減ったことで、インテルのマルチコア・ノード・アーキテクチャーでは、より多くのコア数でプロダクション・モデルを効率良く実行できるようになりました。特に暗黙的なソルバーでは、固定されたメモリー領域と限られた I/O パフォーマンスで、利用可能なすべてのコアを活用して最大限のパフォーマンスを引き出せます。

図 10 に、インテル® クラスターツールによって達成されたパフォーマンスの向上を示します。

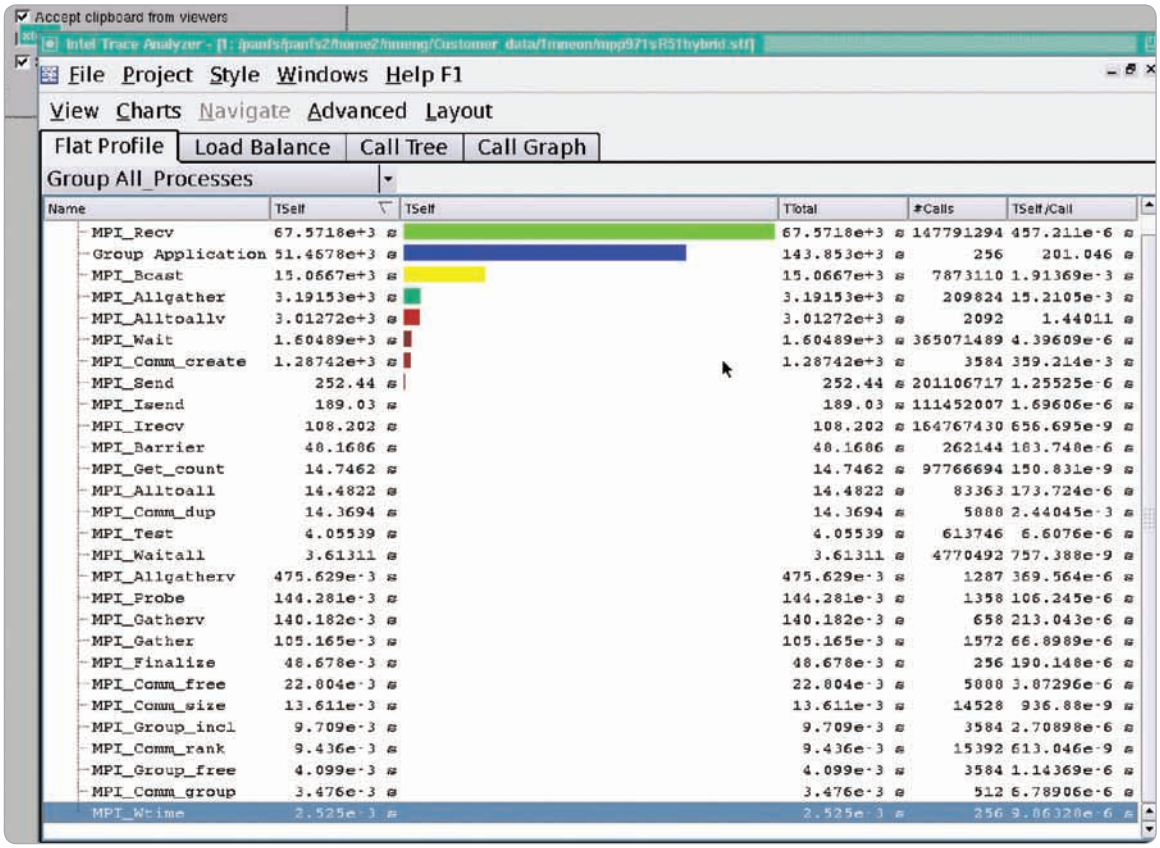


図 11. インテル® トレース・アナライザー / コレクターにより生成された 128 ノード、100 万要素のモデルのプロファイル・データ

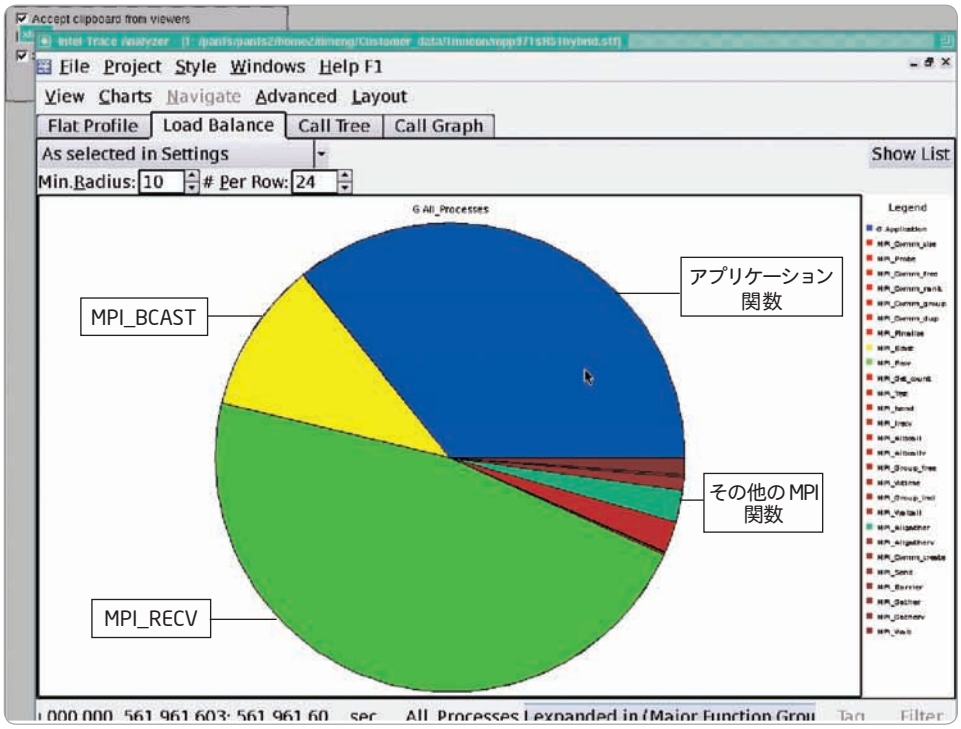


図 12. インテル® トレース・アナライザー/コレクターにより生成された 128 ノード、100 万要素のモデルのロードバランスを円グラフで表示したもの

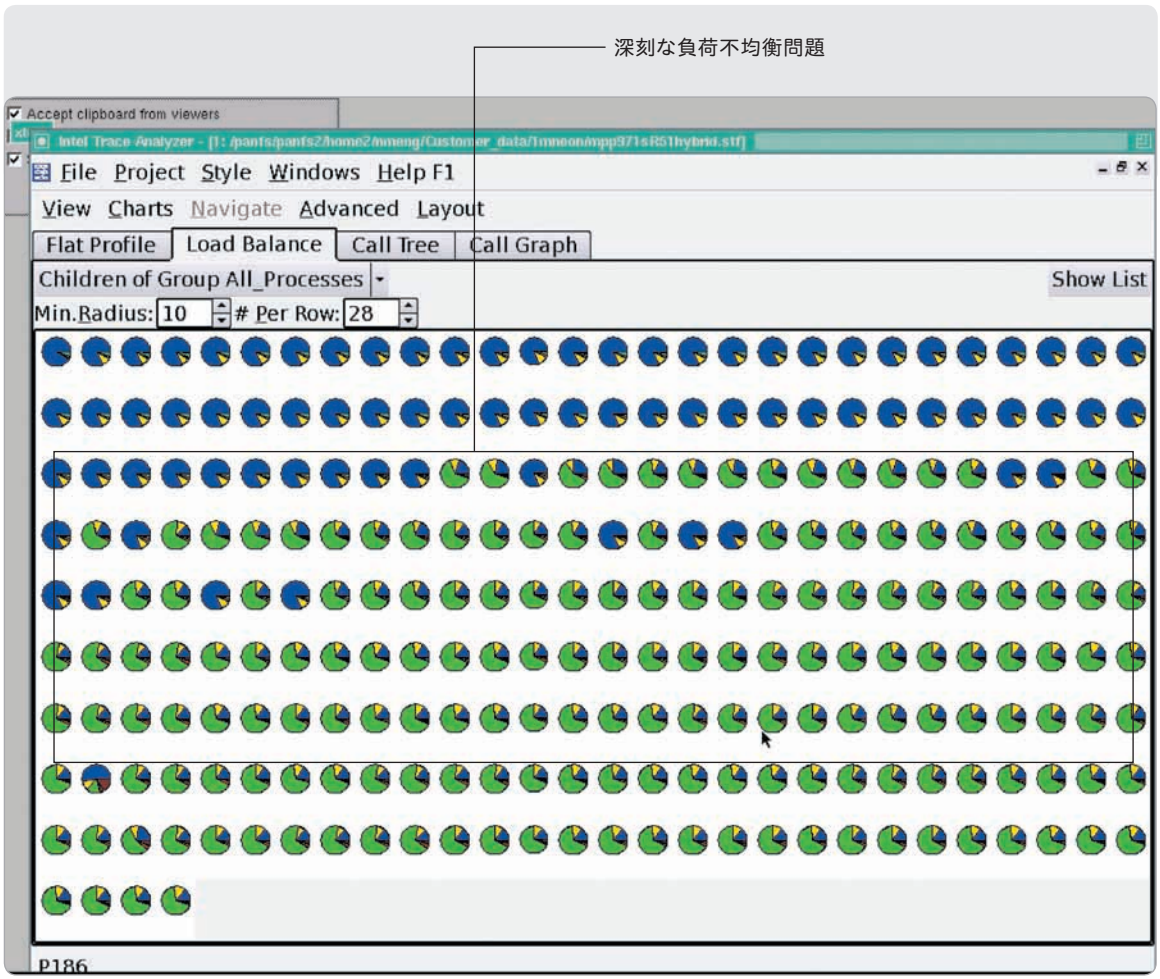


図 13. インテル® トレース・アナライザー/コレクターにより生成された 128 ノード、100 万要素のモデルのロードバランスを円グラフで表示したもの

ITAC により開発時に問題を解決

コード開発中、インテル® MPI 4.0 ライブラリーを使用して HYBRID LS-DYNA で、検証用テストケースとして 100 万要素のモデルのテストを行ったところ、重大なパフォーマンス問題が見つかりました。インテル® トレース・アナライザー / コレクター 8.0.1 のライブラリーを使用することで、このパフォーマンス問題を素早く再現することができました。図 11、12、13 により、その原因が明らかになりました。

円グラフの色分けは各関数の計算コストの割合を表し、青はアプリケーション関数、緑は MPI_RECV 関数、黄色は MPI_BCAST 関数です。インテル® トレース・アナライザー / コレクターのおかげで、パフォーマンス問題の原因を迅速に特定することができました。MPI_RECV 関数で深刻な負荷不均衡が発生していたのです。この問題はすぐに修正され、目標のパフォーマンスを達成することができました。

まとめ

この事例からも分かるように、インテルのクラスターツールは、パフォーマンス向上への取り組みにかかる時間を数年から数カ月へ短縮し、労力も大幅に減らします。それだけでなく、期待するパフォーマンスが得られていない場合にも役立ちます。

これらのツールは短時間で習得できるので、並列化の経験がある開発者や知識豊富なアプリケーション開発者は、クラスターツールで提供される詳細な情報をすぐに活用することができるでしょう。

最適化に関する注意事項

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールには、インテル製マイクロプロセッサーおよび互換マイクロプロセッサーで利用可能な命令セット（SIMD 命令セットなど）向けの最適化オプションが含まれているか、あるいはオプションを利用している可能性がありますが、両者では結果が異なります。また、インテル® コンパイラー用の特定のコンパイラー・オプション（インテル® マイクロアーキテクチャーに非固有のオプションを含む）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。これらのコンパイラー・オプションと関連する命令セットおよび特定のマイクロプロセッサーの詳細は、『インテル® コンパイラー・ユーザー・リファレンス・ガイド』の「コンパイラー・オプション」を参照してください。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のコンパイラーとライブラリーは、選択されたオプション、コード、およびその他の要因に基づいてインテル製マイクロプロセッサーおよび互換マイクロプロセッサー向けに最適化されますが、インテル製マイクロプロセッサーにおいてより優れたパフォーマンスが得られる傾向にあります。

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールは、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。

インテルでは、インテル® コンパイラーおよびライブラリーがインテル製マイクロプロセッサーおよび互換マイクロプロセッサーにおいて、優れたパフォーマンスを引き出すのに役立つ選択肢であると信じておりますが、お客様の要件に最適なコンパイラーを選択いただくよう、他のコンパイラーの評価を行うことを推奨しています。インテルでは、あらゆるコンパイラーやライブラリーで優れたパフォーマンスが引き出され、お客様のビジネスの成功のお役に立ちたいと願っております。お気づきの点がございましたら、お知らせください。

改訂 #20110307

製品情報および購入情報は、インテル® ソフトウェア開発製品 Web サイトを参照してください。
http://www.intel.co.jp/jp/software/products/