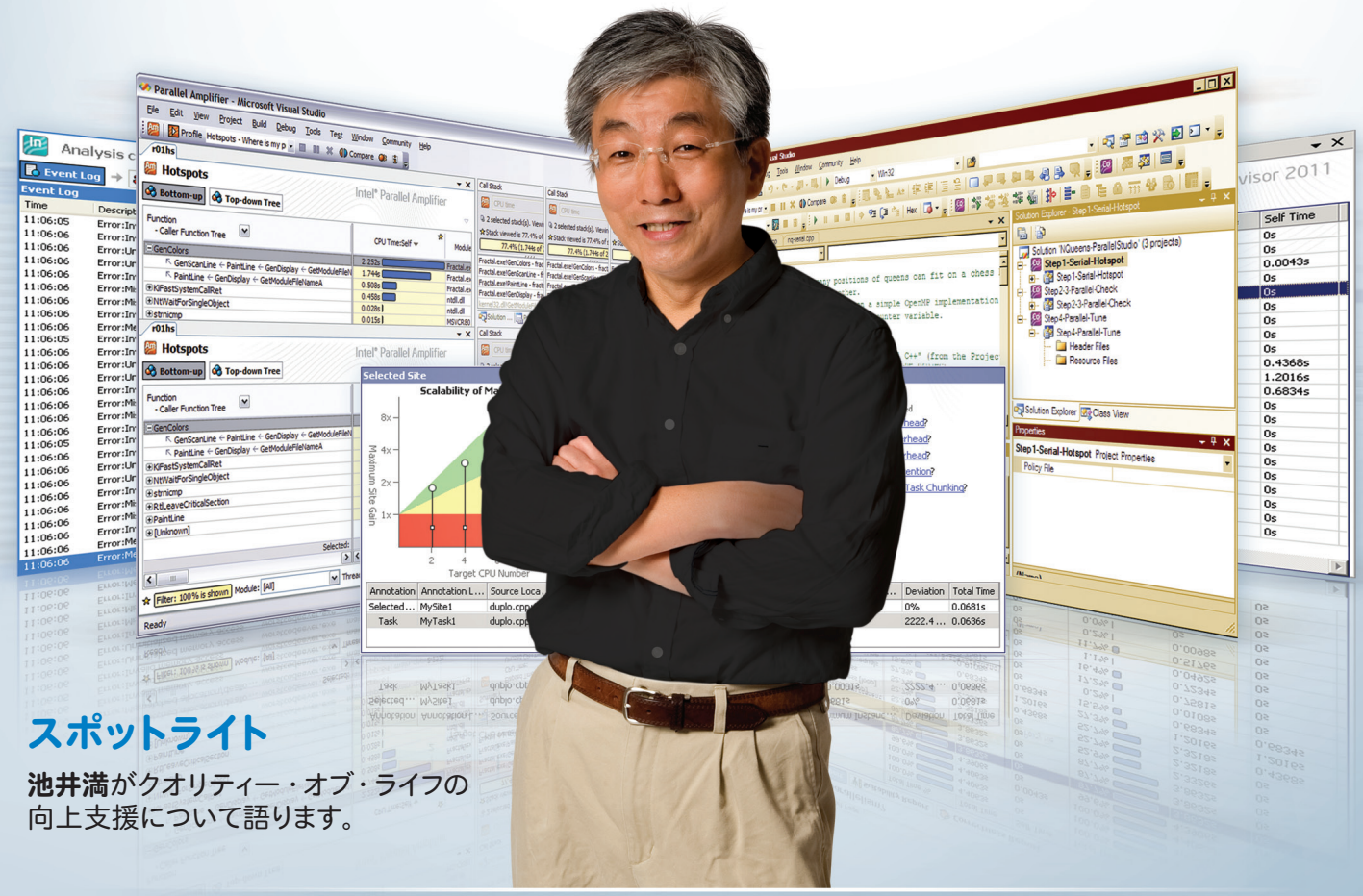
THE PARALLEL UNIVERSE

Issue 4
Fall 2010



スポットライト

池井満がクオリティー・オブ・ライフの向上支援について語ります。

MASTER MULTICORE

目次

革新的な新製品により、信頼関係を強化

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。

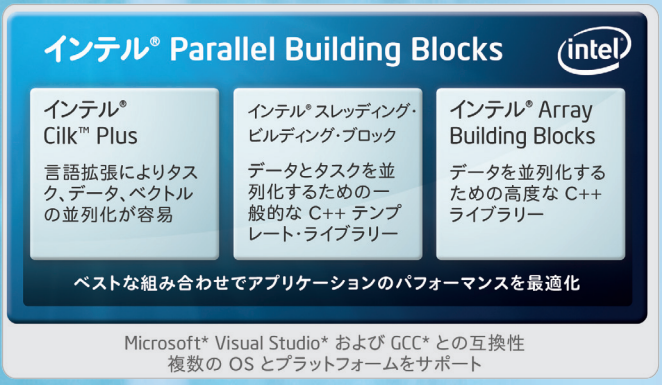


図 1. インテル® Parallel Building Blocks

マルチコア・プロセッサーにより、並列コンピューティングがメインストリームにも普及し、ソフトウェア開発者は並列化ツールやプログラミング・モデルを採用し始めています。

2010 年夏に世界中のユーザーの協力のもとに行われたベータ版のテストは成功に終わり、9 月には多数の新機能を備えたインテル® Parallel Studio 2011 製品版をリリースしています（インテル® Parallel Studio 2011 についてのトピックを参照）。以下は、新機能の一部です。

- 幅広い並列開発モデル – インテル® Parallel Building Blocks（インテル® PBB）
- 革新的なスレッド化アシスタント – インテル® Parallel Advisor 2011
- Microsoft* Visual Studio* 2005、2008、2010 のフルサポート
- インテル® プレミアサポートを通じたテクニカルサポート

インテル® Parallel Studio 2011 とインテル® PBB はともに既存の製品を基に開発されており、並列プログラミングおよび既存の実装との互換性のサポートが強化されています。

製品を選択するとき、それが信頼できるものかどうかを見極めることは重要です。個人的には、その製品に対する開発元の取り組みや熱意、そして実際にその製品を使用しているユーザーの声を重視します。

インテルが力を注ぎ、多数のユーザーからも支持される製品に関わっていることは私の誇りです。

最もポピュラーな並列化支援ツールであるインテル® スレッディング・ビルディング・ブロック（インテル® TBB）は、すでに幅広く採用されています。つい最近では、Adobe* Creative Suite* 5 にもインテル® TBB が採用されました。信頼できる移植性と優れたパフォーマンスを提供するインテル® TBB は、多様なオペレーティング・システムとプロセッサーにおいてソフトウェア開発者に高い価値をもたらすことが実証されています。

革新的なインテル® Parallel Studio は、Windows* ソフトウェア開発者がマルチコア・プロセッサーの能力を引き出せるように支援します。

インテルが既存ユーザーと新規ユーザーの両方に有益な製品と拡張機能を提供できることを非常に嬉しく思っています。

インテル® PBB はインテル® TBB をさらに強化したもので、コンパイラーによるインテル® Cilk™ Plus 機能のサポートとインテル® Array

設計フェーズ	ビルドとデバッグフェーズ	検証フェーズ	チューニング・フェーズ
 革新的なスレッド化アシスタント	 コンパイラーとスレッド化ライブラリー	 メモリーエラーとスレッド化のエラーチェッカー	 スレッドとパフォーマンス・プロファイラー
インテル® Parallel Advisor	**インテル® Parallel Composer**	**インテル® Parallel Inspector**	**インテル® Parallel Amplifier**
スレッド化設計ガイドツールにより並列アプリケーションの設計を簡素化、迅速化 › 並列化が有効なアプリケーションの領域を識別 › スレッド・アプリケーション向けに段階的なガイダンスを提供	最適化コンパイラーでパフォーマンスとスレッド・アプリケーションの設計を強化 › C++ コンパイラーとライブラリー › コードカバレッジ › デバッガー › インテル® Parallel Building Blocks — 多くの並列化開発手法をサポートする幅広い並列モデルのセット	コードの信頼性と品質をより高めるエラー検出分析ツール › コードの信頼性と品質をより高めるエラー検出分析ツール › メモリーリークとメモリー破壊の検出 › データ競合とデッドロックを検出	最適化されたパフォーマンスとスケーラビリティーのチューニング分析 › 最適化されたパフォーマンスとスケーラビリティーのチューニング分析 › パフォーマンスとスケーラビリティーの分析 › ロックと待機の分析

図 2: **インテル® Parallel Studio 2011**

Building Blocks（インテル® ArBB）の高度なベクトル並列化が新たに追加されています。これまでと同様に、インテル® TBB の主要機能であるスレッドに対応するメモリー・アロケーター、コンカレント・コンテナー、移植性の高いロックとアトミック操作、タスクスチール・スケジューラー、グローバルタイマーなども備えています。

インテル® TBB は進化し続けています。最新バージョンでは、FIFO スケジューリングのサポート、C++ 0x サポートの拡張、新しいコンテナー（concurrent_unordered_map）、「設計パターン」マニュアル、Microsoft* Visual Studio* 2010 のサポートが追加されています。また、tbb::graph と呼ばれるコミュニティー・プレビュー機能を利用して、アプリケーションの並列化をグラフで表現することが可能です。この機能は、単純な静的依存性グラフから複雑なメッセージ・パッシング・グラフまで幅広い実装に使用できます。この新機能により、調整作業が大幅に改善されるでしょう。

インテル® Cilk™ Plus の 4 つの構成要素：

- 並列化を表現するための強力で簡単な 3 つのキーワード：for ループを並列化する cilk_for と並列関数呼び出しと同期を行う cilk_spawn と cilk_sync。
- ハイパーオブジェクトと呼ばれる共有変数の競合を処理するためのソリューション。新しいキーワードによって作成されたタスク向けに共有変数のローカルコピーを作成し、並列処理終了後に共有値のリダクションを行います。
- 2 つのベクトルを合計する a[] = b[] + c[] や a[][1] = sqrt(b[][2]) のように、配列のスライスに対してデータ並列操作を行うためのアレイ・ノーテーション。
- 要素関数を使用して __sec_map(saxpy, 2.0, x[0:n], y[0:n]) のような計算を行うためのアレイ・ノーテーション。

インテル® Cilk™ Plus は、コンパイラーによる実装に重点を置いています。インテル® TBB とは異なり、インテル® Cilk™ Plus ではコンパイラーにより並列処理の最適化と管理を行っており、次のような利点があります。(1) キーワードとネイティブ構文を使用してプログラミング言語に統合されるため、コードの記述と理解が容易です。(2) コンパイラーによる最適化は、より多くのデータ競合状態を回避し、パフォーマンスをさらに向上させます。

コンパイラーはインテル® Cilk™ Plus の 4 つの構成要素を理解し、コンパイル時診断、最適化、ランタイム・エラー・チェックに役立てます。インテル® Cilk™ Plus はオープン仕様であるため、インテル® コンパイラー以外でもこの新しい C/C++ 言語機能の実装が可能です。

インテル® ArBB は、プログラミングが容易で移植性が高い高度なベクトル並列化機能を提供するために開発されました。SIMD 命令、マルチコア / メニーコア・プロセッサーの並列処理を活用するための機能を備えています。正則行列、逆行列、スパース行列のビルトインサポートにより、数学的なプログラムにおいて、開発者に過度の負担をかけることなく並列化によるパフォーマンスを引き出します。

インテル® Parallel Studio 2011 では、インテル® Parallel Studio（2009 年 5 月リリース）にインテル® Parallel Advisor とインテル® PBB が追加されました。インテル® Parallel Advisor により、ソフトウェア・アーキテクトは既存のアプリケーションの並列化にあたって、完全に並列化を実装しなくてもさまざまなアプローチを試すことができるようになります。そのため、実装の評価にかかる時間を短縮し、アプリケーションに最適な実装を選択することができます。

James Reinders
2010 年 9 月

アプリケーションのマルチコア対応がより簡単に

インテル® Parallel Studio 2011

インテル コーポレーション
製品マーケティング部門
Leila Chucri

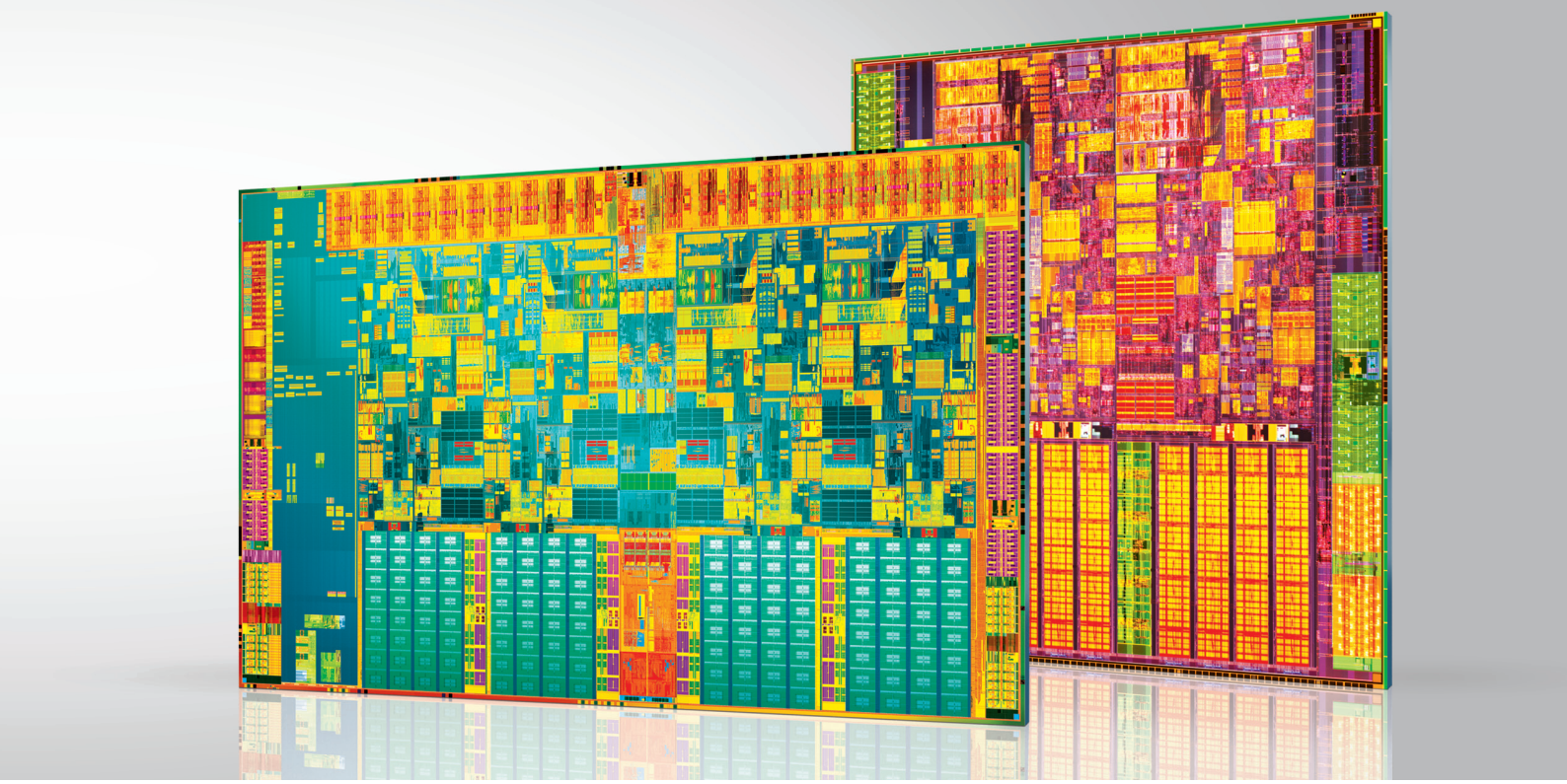
インテル® Parallel Studio 2011 について、マルチコア向けの開発環境を支援する新機能を主に紹介します。

マルチコア時代のアプリケーション開発

並列プログラミングは、これまでパフォーマンスが重視されるゲームなどの特定分野で注目されていましたが、メインストリーム開発者の間でも急速に普及しつつあります。アプリケーションの設計段階でマルチコア・テクノロジーのパフォーマンスとスケーラビリティーを活用することが「好ましい」と言われていた時代はもはや過去のことです。今日では、開発者は皆、シリアル・アプリケーションではプロセッサーのコア数が増加しても、最新のプロセッサー・チップに頼ることだけでは、もはやその限界を引き上げたりユーザー体験を向上させることができないことに気付いています。並列プログラミングはマルチコア・テクノロジーをフルに活用して、アプリケーションの競争力を高めます。既存のシリアルコードを最適化する場合も、マルチコア対応の新しいアプリケーションを作成する場合も、柔軟な設計モデルと信頼性の高い高速アプリケーションを開発するツールを使用することで、より高度な並列プログラミングを簡単に行うことができます。

並列アプリケーションの設計 – 簡単、迅速、そして効率的

インテルがメインストリーム開発者向けにマルチコア開発支援ツールをリリースしてから 1 年以上が経ちました。インテル® Parallel Studio は、コンパイラー、マルチスレッド・ライブラリー、メモリーエラー検出機能、分析ツールから成る、自動スケーリング機能を備えた最初の開発ツールでした。その後、Microsoft* の Visual Studio* PPL など、他社によるこの新分野への進出が続きましたが、**インテル® Parallel Studio 2011 のリリースにより市場におけるインテルの地位はより強固なものとなっています。**最新リリースでは、業界初となるマルチコアの利点が最も得られるコード領域を開発者にアドバイスする機能に加えて、新しいタスク / データ並列モデルが追加されています。ここでは、インテル® Parallel Studio 2011 の注目すべき新機能について紹介します。



インテル® Parallel Studio 2011 待望のリリース

インテル® Parallel Studio 2011 は、高度なツールと手法によりマルチコア開発を支援し、成功へと導きます。並列化への取り組みを容易にし、現在、そして将来のマルチコア・プロセッサーのパフォーマンスとスケーラビリティーを引き出すアプリケーションを開発できるように、Microsoft* Visual Studio* 2005、2008、2010 を使用する C/C++ 開発者を支援します。

新機能

- インテル® Parallel Building Blocks（インテル® PBB）によるスレッド化オプションの拡張：インテル® Parallel Building Blocks は、一般的なものから特別なものまで広範囲の並列モデルを提供します。特定の環境やニーズに合わせて、アプリケーション内で複数の並列モデルを組み合わせて使用できるため、マルチコアへの対応を容易に、スケーラブルな方法で行うことができます。
  - **インテル® スレッディング・ビルディング・ブロック 3.0:** 広く採用されている C++ テンプレート・ライブラリーで、一般的な並列化を行うための柔軟でクロスプラットフォームな並列ソリューションを提供します。
  - **インテル® Cilk™ Plus:** インテル® C/C++ コンパイラーのビルトイン機能で、使いやすく構造化されたモデルです。並列化のコーディング、検証、分析を容易にします。
  - **インテル® Array Building Blocks:** 洗練されたランタイム・ライブラリーに裏付けされた API で、汎用データ並列プログラミングのソリューションを提供します。開発者は、低レベルの並列化メカニズムやハードウェア・アーキテクチャーの依存性を考慮する必要がありません。（software.intel.com/en-us/data-parallel/ からベータ版を入手できます。）
- **インテル® Parallel Advisor:** 画期的なスレッド化のアドバイス機能により、並列アプリケーションの設計を簡単でわかりやすくします。
- **Microsoft* Visual Studio* 2010 のサポート**

「インテル® TBB の実装は驚くほど迅速かつ容易で、インテル® Core™ i7 プロセッサーで Simul Weather SDK のパフォーマンスが飛躍的に向上しました。Simul Weather* とインテルのツールにより、ゲームに天候や雲をリアルタイムに統合するという新たな可能性を実現できました。」

Simul Software 社
創立者兼社長
Roderick Kennedy 氏

インテル® Parallel Advisor: 画期的なスレッド化のアドバイス機能により、並列アプリケーションの設計を容易にします。

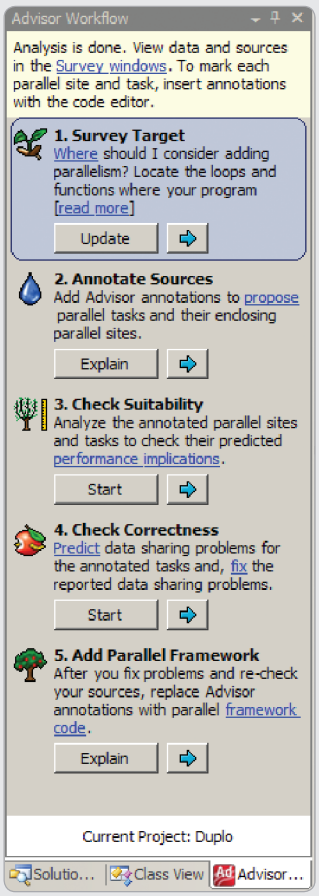
インテル® Parallel Advisor により、マルチコアのパフォーマンスとスケーラビリティーを活用する並列アプリケーションの設計を容易に行うことができます。

並列化に初めて取り組む場合も、すでに並列アプリケーションの開発を行っている場合も、インテル® Parallel Advisor は並列コードの設計と開発に不可欠なツールです。

インテル® Parallel Advisor で、C/C++ コード内の並列化候補を発見および検証し、並列化よって得られるパフォーマンス・ゲインを明らかにし、アプリケーションを再構成する方法を特定し、実際にアプリケーションの並列化に着手する前に投資対効果（ROI）を評価できます。

インテル® Parallel Advisor は、多くの採用実績を誇る並列化のための抽象化手法を使用して、開発者にアプリケーションを並列化するための明確なロードマップと段階的なガイダンスを提供します。タスクベースの並列化ステップで、粗粒度での並列化の可能性を探ってください。

設計フェーズ



インテル® Parallel Advisor はシリアルプログラムの実行を分析しながら作業が可能

- ターゲットの調査 — 並列化の候補として、処理に時間のかかる呼び出しツリーとループを重点的に調査します。
- ソースコードへの注釈の挿入 — 並列化の候補を示すため、注釈をソースコードに挿入します。
- 適合性チェック — スレッド化範囲ごとにパフォーマンス予測とプログラム全体への影響を表示し、各候補のパフォーマンスを評価します。
- 正当性チェック — 並列化の候補にあるデータ競合など、データアクセスに関する問題を特定します。

「コードを並列化する上でインテル® Parallel Advisor の設計アプローチが役立ちました。調査機能により、シリアルコードで多くの CPU 時間を費やしている部分と並列化の利点が得られる部分を特定することでコードを改善することができました。」

化学名誉教授
William Orttung 博士

「インテル® Parallel Advisor により、並列化すべきコード領域が明確になり、無駄な作業を排除して効率良く並列化することができました。」

Vickery Research Alliance
Matt Osterberg 氏

「長年 Microsoft* Visual Studio* で C++ 開発を行ってきましたが、並列プログラミングの経験は全くありませんでした。インテル® Parallel Advisor は、アプリケーションの設計および開発フェーズにおいて、並列化を容易に効率良く取り入れるのに非常に役立ちました。」

Brian Reynolds Research
Brian Reynolds 氏

インテル® Parallel Composer 2011: ライブラリー、並列モデル、デバッグ機能を備えた C/C++ コンパイラーでパフォーマンスを強化

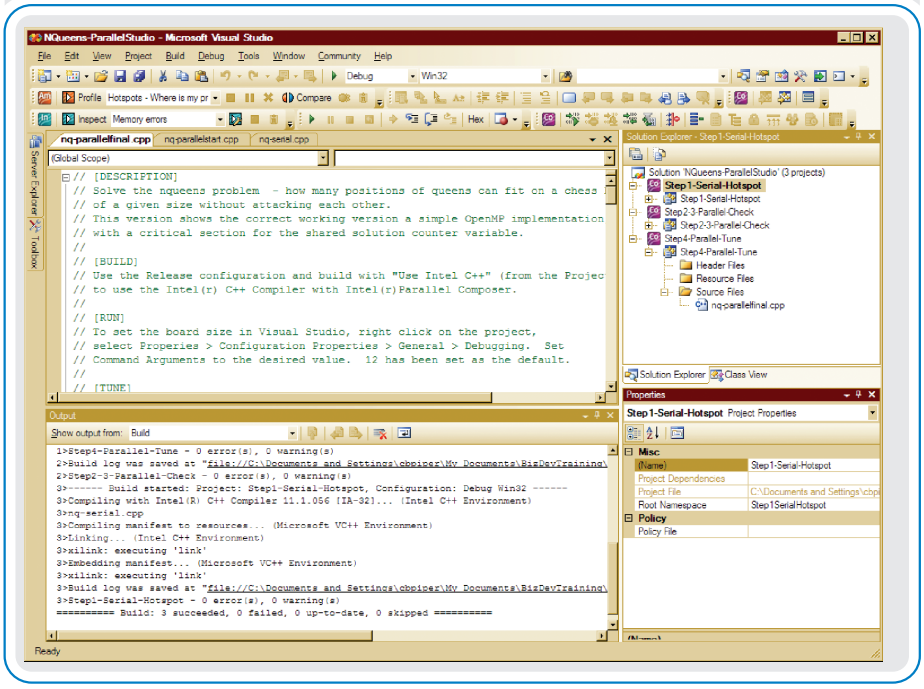
インテル® Parallel Composer 2011 には、C/C++ 最適化コンパイラー、インテル® インテグレーテッド・パフォーマンス・プリミティブ（インテル® IPP）の高度なライブラリー、インテル® Parallel Building Blocks（インテル® PBB）の幅広い並列開発モデルが含まれています。これらのツールを組み合わせることで、パフォーマンスの強化と並列アプリケーション開発の合理化を支援します。

インテル® Parallel Composer の最新バージョンのインテル® コンパイラー、並列化用ライブラリー、そしてすぐに使える多数のサンプルコードによって、アプリケーションの並列化がより簡単になります。アプリケーション・クラス固有のスレッドセーフなマルチスレッド・ライブラリーでは、単純なものから複雑なものまで、そしてデータからタスクまで、多様な並列表現をサポートしています。さまざまな自動化により、短時間でマルチコア・プロセッサーを活用することができます。

設計フェーズではコードの並列化を行うとともに、アプリケーションをさらに最適化、検証、チューニングし、マルチコアとメニーコアのパフォーマンスおよびスケーラビリティーを最大限に引き出すための、将来に渡って利用可能なツールが不可欠です。インテル® Parallel Studio 2011 は、アプリケーションの設計、コーディング、デバッグ、検証からパフォーマンスと信頼性のチューニングまで、開発サイクル全体を通して開発者を支援します。

ビルドとデバッグフェーズ

最適化コンパイラーとライブラリーによりアプリケーションのパフォーマンスを大幅に向上



「Trading Systems Lab では、インテル® Parallel Studio の C++ コンパイラーを使用することにより、TSL Algo Auto-Design Platform で使用されるマルチモードの取引シミュレーターのパフォーマンスを 10% から 20% 向上することができました。Microsoft* Visual C++* との互換性は素晴らしく、Parallel Studio でより多くの並列化機能を使用するのを楽しみにしています。」

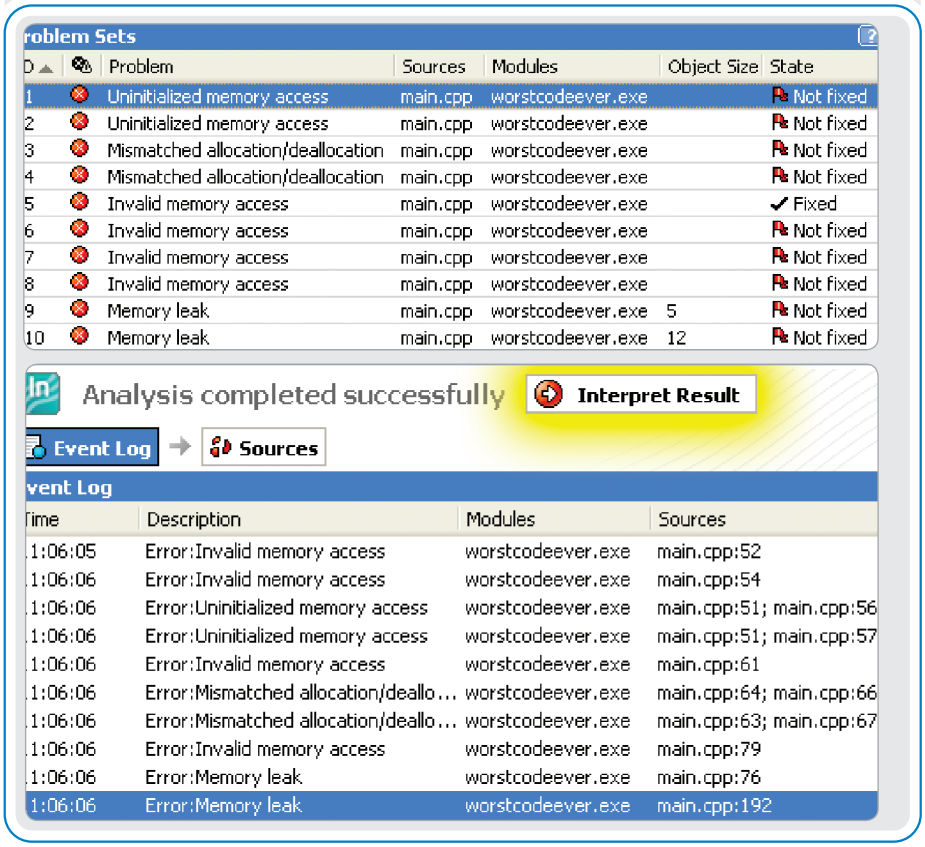
Trading Systems Lab
代表取締役
Mike Barna 氏

インテル® Parallel Inspector 2011: メモリー / スレッド化エラーチェッカーによりアプリケーションの信頼性を強化

開発サイクルの初期にソフトウェアの不具合を検出することで、時間とコストが節約され、投資収益率（ROI）を改善できます。インテル® Parallel Inspector は、シリアル・アプリケーションとマルチスレッド・アプリケーションの両方のエラーチェック機能を備えたソリューションです。メモリーリークやメモリー破壊はもちろんのこと、データ競合やデッドロックもピンポイントで検出します。

検証フェーズ

動的分析のメモリー / スレッドチェッカー



「マルチコアとメニーコアのパフォーマンスを活用することは SIMULIA のビジネスにとって非常に重要です。インテル® Parallel Inspector を利用することで、不安定なコードが含まれる場合に時間とコストがかかっていた従来の手法に比べて、安定した並列コードを効率良く開発できるようになりました。」

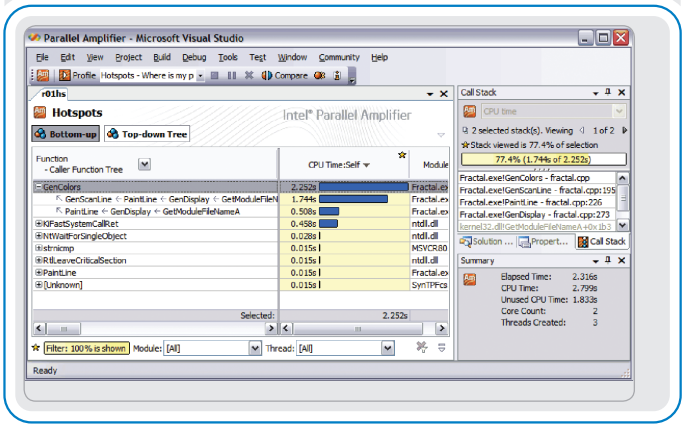
SIMULIA チーフ・アーキテクト
Matt Dunbar 氏

インテル® Parallel Amplifier 2011: スレッド化 / パフォーマンス・プロファイラーによりパフォーマンスを向上

インテル® Parallel Amplifier 2011 では、従来の推測作業を取り除き、Windows* アプリケーションのパフォーマンス動作を分析し、迅速かつ精度の高い意思決定を促すスケーリング情報を素早く入手することができます。また、最適なパフォーマンスのきめ細かなチューニングを支援し、コアが充分に有効活用され、新しい機能がサポートされるようにします。

チューニング・フェーズ

スレッドとパフォーマンス・プロファイラー



「インテルにはとても感謝しています。インテル® Parallel Amplifier を試してみたところ、ほとんどの時間を費やしているソース行が正確に示され、感激しました。すぐに変更を加え、開発アプリケーションは、今や 10 倍近い速さで動作しています。GUI も非常に使いやすいという印象です。」

ヒューストン大学生命医学研究所
研究アシスタント
Dat Chu 氏

DEVELOPER SPOTLIGHT

池井 満
ソフトウェア & サービス
エンジニアリング・マネージャー

これまでの経歴

日立化成研究所で、いくつかの化学系プログラムを最先端の商用並列コンピューター向けに移植、並列化して最適化を行いました。インテルでは、インテル® Paragon スーパーコンピューターやインテル® Itanium システムを使用する多数のプロジェクトに関わり、最近ではインテル® Xeon™ プロセッサー向けに ISV アプリケーション・プログラムの最適化と並列化に取り組みました。

現在の取り組み

インテル® Atom™ /SoC とハイパフォーマンス・コンピューティング（HPC）プロジェクトに従事しています。インテル® Atom™ グループでは、ISV アプリケーションをスマートフォンなどの SFF（スモール・フォーム・ファクター）デバイスに移植するための支援を行っています。インテルは、SFF 向けの真のオープンソース OS である MeeGO* を支持しており、その取り組みを支援しています。HPC グループでは、日本の主要な研究所、大学、各種業界と協力して、アプリケーションをより高速に効率良く実行できるように最適化を行っています。

この取り組みの重要性

現在、スマートフォン、タブレット、その他の SFF デバイスには電力消費とスペースの制約があるため、これらを PC と同じように使用することはできません。インテル® Atom™ プロセッサーは電力消費を大幅に削減することでこの課題をクリアします。インテル® Atom™ グループは、クオリティー・オブ・ライフを向上させる SFF デバイスの設計を支援しています。HPC グループは、日本最大のコンピューター・システムのパフォーマンスを向上させるべく取り組んでいます。研究者と密接に協力し、計算アプリケーションのパフォーマンスを向上させる過程で、そのアプリケーションの分野の発展に貢献しています。

ソフトウェア開発者としての目標

最先端の（そして最も広く普及している）プロセッサー向けにプログラムをただ書き続けることです。東京ドームで野球を観戦しながら、SFF モバイルデバイスでハイパフォーマンスなコンピューター・システムを開発するのが夢です。

世界初のヒント数 39 の究極数独問題

Intel Compiler Labs
Stephen Blair-Chappell

世界初のヒント数 39 の究極数独問題は、オスロの有名なエンジニアである Lars Peters Endresen 氏と Håvard Graff 氏の協力により完成しました。

究極問題への挑戦

数独の 9x9 の盤面には 6 x 10^{21} を超える有効な組み合わせが存在します。総当り手法を用いてすべての組み合わせを試すことは、それほど難しいプログラミング演習ではありませんが、時間がいくらあっても足りません。真の挑戦とは、我々に与えられた時間内に計算が完了するプログラムを作成することです。

ヒント数 38 の究極問題はすでに作成されていたため、今回はヒント数 39 の究極問題の作成に挑戦しました。

コードの開発
最初に最適化されていないコードを記述した後、3 つのステップを経てコードのパフォーマンスを向上させました。

- ステップ 1: アルゴリズムの変更 - 総当り手法をショートカット
- ステップ 2: シリアルコードの最適化 - SSE 命令を活用
- ステップ 3: 並列処理の追加

コードの開発には 2 年以上かかり、そのほとんどは勤務時間外に行われました。プログラミングに費やした時間は 2,000-3,000 時間にものぼります。

ステップ 1: アルゴリズムの変更

異なるソリューションをすべて作成する総当り手法を使用するのではなく、既存の問題から 1 つまたは 2 つのヒントを削除して新しい問題を作成する再帰的アルゴリズムを使用することに決めました。

新しいヒント数 17 の問題を見つけるため、ヒント数 18 の問題から 2 つのヒントを削除して、有効なすべてのソリューションを検索しました。例えば、図 1 では、最初に 1 列目からヒント 3 と 9 を削除しています。次に、有効な数字で空いているマスを埋めていきます。

さらに、余分なヒントがないように注意しながら、他の選択肢を再帰的に除去して有効な問題を見つけ出します。我々は、この新しい問題を作成する手法を、“-2 + 1” アルゴリズムと呼んでいます。

ヒント数 39 の問題作成にも同様のテクニックを使用しました。つまり、既存のヒント数 38 の究極問題から 1 つのヒントを削除して 2 つの新しいヒントを追加したのです。我々は、この手法を、“-1 + 2” アルゴリズムと呼んでいます。

ステップ 2: シリアルコードの最適化

最新の CPU は、同時に複数のデータを処理できる SIMD（Single Instruction Multiple Data）命令に対応しています。従来の命令を SIMD 命令に変更することで、コードの実行速度を大幅に向上できます。同様の命令には、MMX やストリーミング SIMD 拡張命令（SSE、SSE2、...）があります。

SSE 組み込み関数の追加

SSE 組み込み関数は、C/C++ コードから呼び出すことが可能な、アセンブラー形式のビルトイン関数であり、インライン・アセンブラーを使用しなくても、SIMD 機能の低レベルな実装を可能にします。インライン・アセンブラーを使用する場合と比べ、コードの可読性が向上します。また、命令スケジューリングのサポートにより、デバッグにかかる労力が軽減されます。組み込み関数を使用すると、C/C++ 言語の標準的な構造では生成できない命令を利用できます。

インテル® コンパイラーは、初期の MMX 命令から最新世代の SSE4.2 拡張命令まで、さまざまなアーキテクチャーの拡張命令をサポートしています。

図 2 のコードは、128 ビットの SSE2 レジスターが数独コードでどのように使用されているかを示します。

数独ジェネレーターの最初のバージョンでは、SSE 命令や組み込み関数を使用していませんでした。このコードを、SSE2 レジスターを使用するように変更するにはかなりの時間がかかりましたが、SSE 組み込み関数の追加により、コードは大幅に高速化されました。

SSE 組み込み関数の使用には短所もあります。それは、実装が特定世代のアーキテクチャーに制限されることです。また、SSE 関数の長い名前は C++ コードの可読性を低下させます。プログラマーが新しい関数を習得する必要もあります。しかし、数独ジェネレーターの場合、このような労力よりもパフォーマンスの向上がはるかに重要でした。

ステップ 3: 並列処理の追加

当初は、OpenMP* 3.0 で定義されている OpenMP* タスクを使用しました。OpenMP* 3.0 は、インテル® C/C++ コンパイラー 11.0 からサポートされています。並列処理の追加には約 2 週間かかりました。作業中は非常に長い時間がかかっているように感じましたが、プロジェクト全体の長さから見れば費やされた時間はごくわずかにすぎません。図 3 は、OpenMP* タスクの使用例です。

OpenMP* コードを追加する上で最も難しいポイントの 1 つは、変数の処理方法を理解することです。OpenMP* のデータは共有またはプライベートにできます。変数が正しいスコープレベルに収まるように、何度も繰り返しコードを微調整しました。並列処理の追加にかかった時間のほとんどは、異なる実行タスク間でのデータ共有を最小限に抑え、並列化された異なるループ間に依存性がないことを確実にするためのコード変更に費やされました。

OpenMP* の代わりにインテル® Cilk™ を使用する

インテル® Parallel Studio は、プログラムを並列化するさまざまな手法をサポートしています。図 4 のインテル® Parallel Building Blocks では、複数の並列プログラミングがサポートされています。

前述したように、ヒント数 39 の究極数独問題は OpenMP* を使用して並列化されており、ネイティブスレッドを使用するよりもはるかに簡単です。

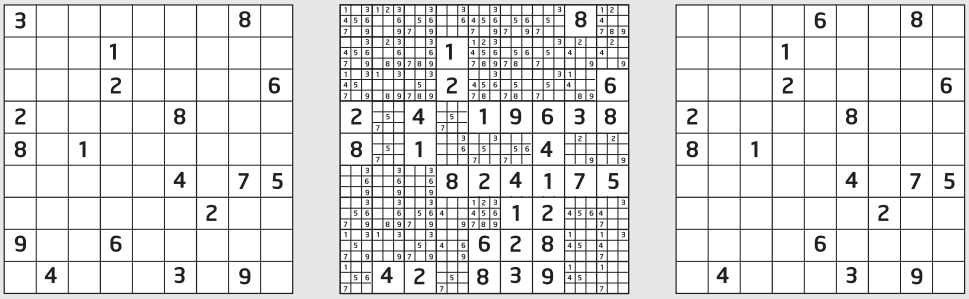


図 1: 新しいヒント数 17 の数独問題の作成

```
for(int num=0; num < 9; num++)
{
    __m128i xmm0 = _mm_and_si128(BinSmallNum, BinNum[num]);
    for(int i=0; i < 9; i++)
    {
     __m128i BoxSum    =   _mm_and_si128(BinBox[i],    xmm0);
     __m128i RowSum    =   _mm_and_si128(BinRow[i],    xmm0);
     __m128i ColumnSum =   _mm_and_si128(BinColumn[i], xmm0);
     if (ExactlyOneBit(BoxSum))
     {
        int cell=BitToNum(BoxSum);
        FoundNumber(cell, num);
        return true;
        }
     }
}
```

xmm0 は特定の数のすべての具体値に対するビットマスクになる

i は各列、行、ボックスにわたって使用される

1 つの数字が 1 回だけ使用されていることを確認

図 2: SSE コンパイラー組み込み関数を使用してパフォーマンスを向上

ただし、このプロジェクトを今から開始するのであれば、インテル® Cilk™ Plus は魅力的な選択肢と言えるでしょう。なぜなら、既存の C コードに Cilk を追加することは非常に簡単だからです。

Cilk は 3 つのキーワード（cilk_spawn、cilk_sync および cilk_for）を使用します。ファイルにヘッダーファイル cilk.h がインクルードされると、これらのキーワードを使用できるようになります。図 5 を参照してください。

Cilk では、プログラマーはプログラムの並列処理は制御しませんが、意図は示します。プログラムに Cilk キーワードを入れることで、プログラマーはコードを並列実行する許可を与えます。コードを並列で実行するかどうかの決定は、ランタイムに Cilk スケジューラーによって行われます。インテル® Cilk™ Plus ランタイムは、負荷分散を自動的に行います。

図 6 は、Cilk の cilk_for キーワードを使用してコードを並列に実行する方法を示しています。コードはオリジナルの OpenMP* タスクと同じ位置に挿入されます（図 3 を参照）。このソリューションは驚くほど単純です。

インテル® Cilk™ Plus は既存のプログラムを並列化する最も簡単な方法

コードを並列化する場合、プログラマーはデータ競合が発生しないように注意する必要があります。グローバル変数がある場合、代わりに、例えば、ローカル変数や自動変数を使用して、変数のスコープが制限されるようにコードを変更する必要があります。すべてのグローバル変数を削除することが不可能な場合、同時に 1 つのスレッドだけがその変数を変更できるように変数へのアクセスを保護します。

Cilk でグローバル変数と共有変数を処理する最も簡単な方法は、これらの変数をレデューサーとして宣言することです。ワーカーがレデューサーにアクセスすると、安全に操作できる独自のプライベート・ビューが提示されます。ビューは get_value() の呼び出しによりコードのシリアル部分にマージされます。図 7 のコードは、グローバル変数 gNumCilkPuzzlesSolved を reducer_opadd で宣言する方法を示しています。コードのシリアル部分で get_value() を呼び出すことにより、レデューサーのすべての値が結合され、正しい値が得られます。

```
#pragma omp parallel                    ← ここでスレッドのプールを作成
{
    #pragma omp single nowait           ← 1 つのスレッドでループを実行
    {
        for( int i=0; i< NUM_NODES -1; i++)
        {
            NODE Node1 = pPuzzle ->Nodes [i];
            if (Node1.number > 0)
        {
            //最上位レベルのノードのコピーを作成;
            memcpy (&gPuzzles[i];pPuzzle, sizeof (SUDOKU));
            #pragma omp taskprivate (i)
            GenDoWork (&gPuzzles[i],i;
        }                               ← 1 つの “for ループ” スレッドが GenDoWork () の各インスタンスのタスクを作成
    }
}
}
```

図 3. OpenMP* タスクを使用したコード

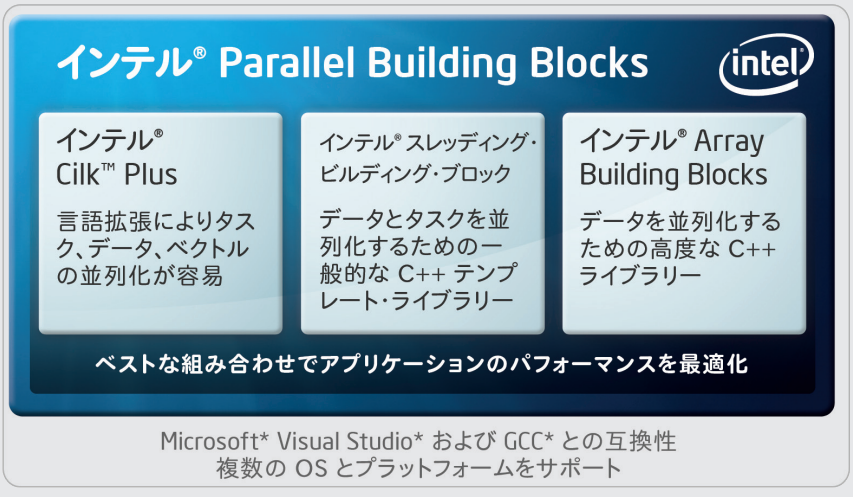


図 4. インテル® Parallel Building Blocks は並列プログラミングのさまざまなモデルを提供

```
#include <cilk/cilk.h>
void work(int num}
{
    // ここにコードを追加
}

void func1()
{
        cilk_spawn work(1);
        work(2);
        cilk_sync;
}

void func2()
{
    cilk_for(int i=0; i<9; i++)
    {
        work(3);
    }

}
```

cilk_spawn と cilk_sync
cilk_spawn と cilk_sync の間の行は継続処理として知られています。cilk_spawn は、継続処理と work(1) を並列で実行することをランタイムに許可します。別のワーカーが利用可能な場合、スケジューラーは最初のワーカーから継続処理をスチールして、別のワーカーに割り当てます。最初のワーカーは work(1) の実行を継続します。cilk_sync の後、コードはシリアル実行に戻ります。

cilk_for
C/C++ の for ループを置換します。ループは利用可能なワーカー間で共有されます。特定の実行順は保証されません。すべてのループが実行されると、プログラムは継続されます。ループの仕事量が等しくない場合、スケジューラーのワークスチール・アルゴリズムによって負荷が分散されます。

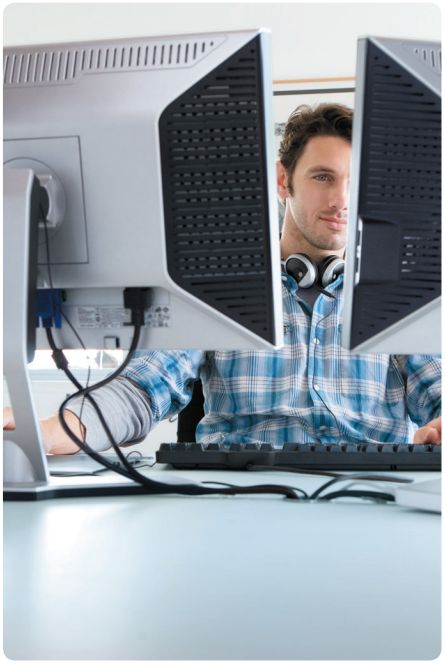


図 5. インテル® Cilk™ Plus の 3 つのキーワード

```
#include <cilk/cilk.h>
.
.
.
cilk_for(int i = 0 ; i < NUM_NODES -1; i++ )

   {
        NODE Node1 = pPuzzle->Nodes[i];
        if(Node1.number > 0)
        {
            // 最上位レベルのノードのコピーを作成;
            memcpy(&gPuzzles[i],pPuzzle,sizeof(SUDOKU));
            GenDoWork(&gPuzzles[i],i);
        }
    }
```

図 6. 数独コードに cilk_for を追加

「シリアルコードを最適化する場合でも、マルチコア対応の新しいアプリケーションを作成する場合でも、アプリケーションの信頼性と速度を高める柔軟性のある設計モデルとツールを使用することで、並列プログラミングは進化し、性能が向上します。」

```
int gNumCilkPuzzlesSolved;       // グローバル変数
.
.
gNumCilkPuzzlesSolved++;          // 並列コードのどこか
.
int Tmp = gNumCilkPuzzlesSolved; // シリアルコードのどこか
```

(a) グローバル変数 gNumCilkPuzzlesSolved は並列コードでは安全に使用できない。

```
#include <cilk/reducer_oppad.h>
cilk::reducer_opadd<int> gNumCilkPuzzlesSolved;
.
.
gNumCilkPuzzlesSolved++;           // 並列コード
.
.
int Tmp = gNumCilkPuzzlesSolved.get_value();// シリアルコード
```

(b) 安全に使用できるようにグローバル変数をレデューサーとして宣言する。

図 7. インテル® Cilk™ Plus のレデューサーを使用してデータ競合問題を解決

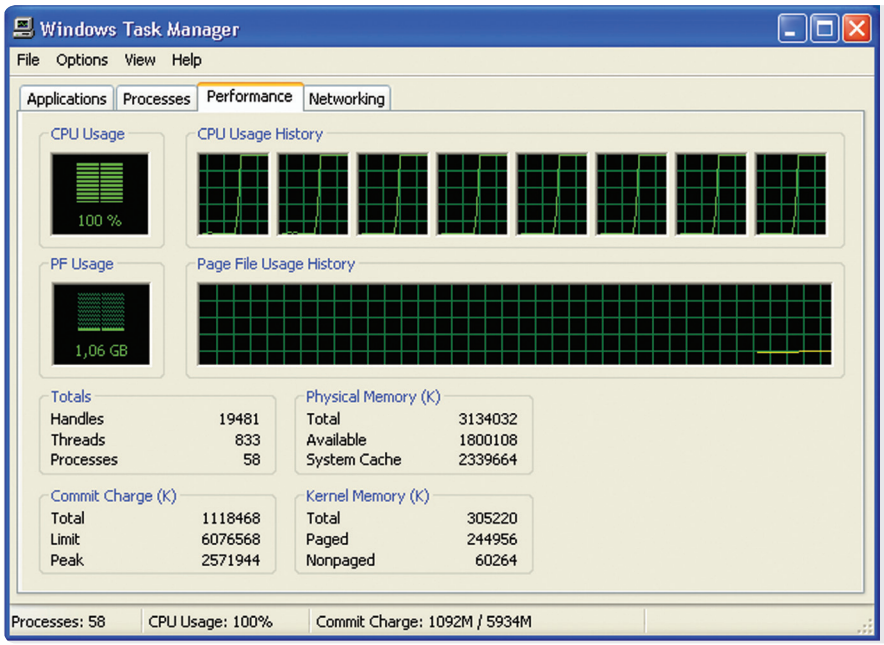
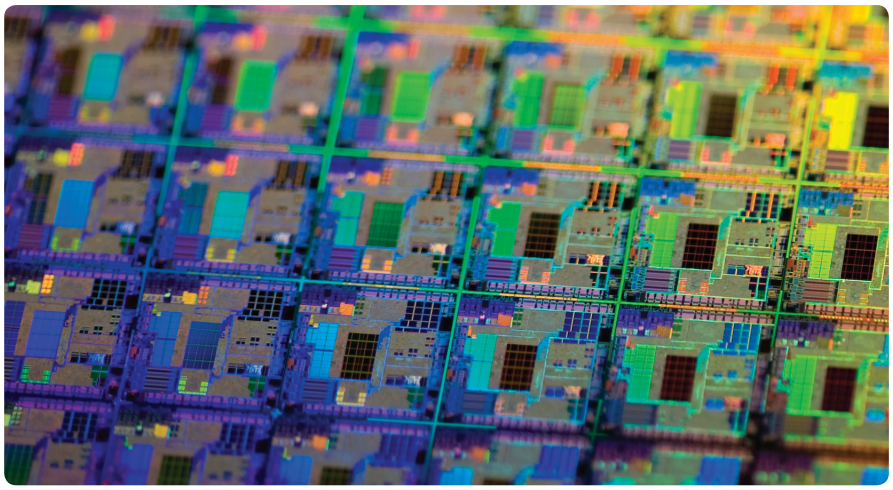


図 8. コードに実装された並列処理により、各ハードウェア・スレッドは CPU 使用率 100% で実行される。



結果

オリジナルの OpenMP* ソリューションでは、並列コードをプロジェクトに追加すると、SMT クアッドコア（8 つのハードウェア・スレッドをサポート）で 8 つのハードウェア・スレッドすべてが使用中になっていることがわかります。（図 8 を参照）

実装がはるかに容易であったにもかかわらず、Cilk を使用した場合の実験で、Cilk ソリューションは OpenMP* ソリューションと同じように実行されました。

図 9 は、今回数独ジェネレーターを使用して見つかった 3 つの新しいヒント数 39 の究極数独問題です。

				3				
		3	6		7			1
6		4		9	1	3		7
5					3		2	4
7	4			6	2	5		3
		2			5	7	1	
2		5	7	1	6	4		
4		6		2	9	1	7	5

				3				
		3	6		7			1
6		4		9	1	3		7
5					3		2	4
1	4			6	2	5		3
		2			5	1	7	
2		5	1	7	6	4		
4		6		2	9	7	1	5

		1		3	7	6	4	
							8	
6		4		9	1	3		7
		2			6			3
1	4			2	3	5		6
3				1	5	4	2	
2		5			9	7		4
4	1			7	2			
	7			5		2	6	

図 9. “-1 + 2” アルゴリズムを使用して見つかった 3 つのヒント数 39 の究極数独問題

インテル® Cilk™ Plus

インテル® Cilk™ Plus の主な機能

- 3 つのキーワード。タスクの並列化を指示します。
- レデューサー。各タスクの共有変数のビューを自動的に作成し、タスク完了後に共有値に戻すことで、タスク間の共有変数の競合をなくします。
- アレイ・ノーテーション。配列を操作して C/C++ 表記のセクションでデータを並列処理します。
- 要素関数。関数または演算全体でデータの並列処理を有効にします。配列 / スカラーの全体または一部に適用できます。
- simd プラグマ。インテル® コンパイラーを使用して標準準拠の C/C++ コードにおいて、ハードウェア SIMD 並列処理を利用するベクトル並列処理を表現できるようにします。

** OpenMP を使用したオリジナルプログラムは、Lars Peters Endresen 氏と Håvard Graff 氏によって記述されました。その後、Stephen Blair-Chappell によりインテル® Cilk™ Plus を使用して書き換えられました。

実践例を加えた完全なケーススタディーは、WROX シリーズの 書籍『Parallel Programming with Intel Parallel Studio』（著者 Stephen Blair-Chappell および Andrew Stokes、Wiley Publishing Inc. ISBN 9780470891650（March 2011））に記述されています。

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/

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を行わ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 (インテル® 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 #20101101