THE PARALLEL UNIVERSE

Issue 6
2011 年 3 月

スポットライト

菅原清文が最新の並列
プログラミングについて語ります。

並列化

インテル® Cilk™ Plus とインテル® スレッディング・ビルディング・ブロックを使用したインテル® インテグレーテッド・パフォーマンス・プリミティブ関数の並列化

Walter Shands

コーディング・ヒント

インテル® Array Building Blocks

Zhang Zhang

編集者

からのメッセージ

James Reinders

菅原清文

シニア・ソフトウェア・エンジニア

目次

編集者からのメッセージ

PBB、XE、カンブリア爆発とコンピューター・プログラミング

JAMES REINDERS 3

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ある James Reinders がプログラミングのニーズに合った製品とプログラミング・モデルを決定する際の選択肢について語ります。

インテル® Cilk™ Plus とインテル® スレッディング・ビルディング・ブロックを使用したインテル® インテグレーテッド・パフォーマンス・プリミティブ関数の並列化

WALTER SHANDS 5

インテル® Parallel Building Blocks の並列モデルは、既存のアプリケーションに容易に統合可能。既存コードへの投資を保護し、並列アプリケーションの開発を促進します。

インテル® Array Building Blocks コーディング・ヒント

ZHANG ZHANG 11

ランタイムシステムにおける高性能なコードの生成を支援するインテル® Array Building Blocks API を正しく理解することでエラーを回避します。

導入事例の紹介 19

Envivio、The Creative Assembly、Altair の各社がインテル® ソフトウェア開発ツールの使用によりパフォーマンス、生産性、信頼性をどのように高めることができたかを紹介します。

© 2011 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Cilk、Intel Core、VTune は、アメリカ合衆国およびその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

編集者からのメッセージ

PBB、XE、カンブリア爆発とコンピューター・プログラミング

選択肢があるのはとても大切なことです。コンピューター・プログラミングの世界でも、メカニックのように、作業に適したツールを選択することが重要です。今回の Parallel Universe では、作業に適したツールを選択できるように、さまざまなプログラミング手法とツールについて説明します。各トピックでは、プログラミング・モデルとツールを使用した結果とヒントを交えつつ、最新のインテル® Prallel Studio XE ツールを使った並列プログラミングを紹介します。

バイキング料理を食べるとき、選択肢の多さに困ったことはありませんか。大勢で中華料理のレストランへ行って 185 種類の料理が書かれたメニューを前にしたら、注文が決まるまでかなり時間がかかるでしょう。しかし、誰かがお薦めの料理を教えてくれたり、「本日のスペシャル」があれば、注文に悩まなくて済みますよね。

そこで、マルチコア・プログラミングでお悩みの方にはインテル® Parallel Studio XE とインテル® Parallel Building Blocks を、プロセッサーのクラスターを含む大規模な並列プログラミングでお悩みの方にはインテル® Cluster Studio とインテル® メッセージ・パッシング・インターフェイス (MPI) ライブラリーをそれぞれお薦めします。

これらの製品を推奨するにあたり、前回の記事について非常に良いフィードバックを頂きました。まだお読みになっていない方は、Parallel Universe Issue 5 をご覧ください。

今回は、これらの製品とプログラミング・モデルを使用して実際にできることを具体的に示すことで、お薦めする理由を詳しくご説明します。皆さんがプログラミングのニーズに合った製品とプログラミング・モデルを正しく選択できるようにお手伝いします。そのすべてを紹介することはできませんが、ここで紹介した内容はなんらかの参考になることでしょう。

最も使用されているプログラミング言語とプログラミング環境を拡張するソリューション

以前述べたように、コンピューター・ハードウェアはこれまでになく複雑になっています。プログラミング・モデルとツールの課題は、高水準の機能を利用してこれを上手に扱いつつ、非常に低水準の制御を行った場合と同じ結果が得られるようにすることです。

最善の努力がなされたにもかかわらず、非常に多くのソリューションが提唱され、私の同僚はこの現象を「並列プログラミングのカンブリア爆発」と呼んでいます。この中から勝者を選ぶことができるでしょうか？

私はできると思っています。なぜなら、この「カンブリア爆発」[†]は 1950 年代にプログラミング言語で始まったものであり、それなりの時間が経過しているからです。この新しい勝者は、最後の爆発の勝者を超える必要があります。それとも始めからやり直すべきでしょうか。私はそうは思いません。詳しくはまたの機会に述べることにしましょう。

最もポピュラーなソリューション、つまり「勝者」は、最も使用されているプログラミングを拡張するものになるでしょう。

インテル® Parallel Building Blocks とインテル® Parallel Studio XE が C、C++、Fortran 開発者向けに提供され、高い人気を誇っている理由はまさにここにあります。最も使用されているこれらのプログラミング言語とプログラミング環境を拡張するソリューションについて、さらに詳しく見ていくことにしましょう。

今回の記事では、さまざまなプログラミング・モデルとツールの利用方法とヒントを紹介します。

今後の記事をより良いものにするため、記事や書籍についてのご意見ご感想をお寄せください。お手数ですが、james.r.reinders@intel.com までお送りください。これまでさまざまなご意見ご感想をお送りいただきましてありがとうございました。引き続きご意見をお待ちしております。

JAMES REINDERS
オレゴン州ポートランド
2011 年 3 月

[†]カンブリア爆発とは？
カンブリア爆発 とは、一般に、古生代カンブリア紀（およそ 5 億 4200 万年前から 5 億 3000 万年前の間）に突如として今日見られる動物の「門（ボディプラン、生物の体制）」が出そろった現象であるとされる。

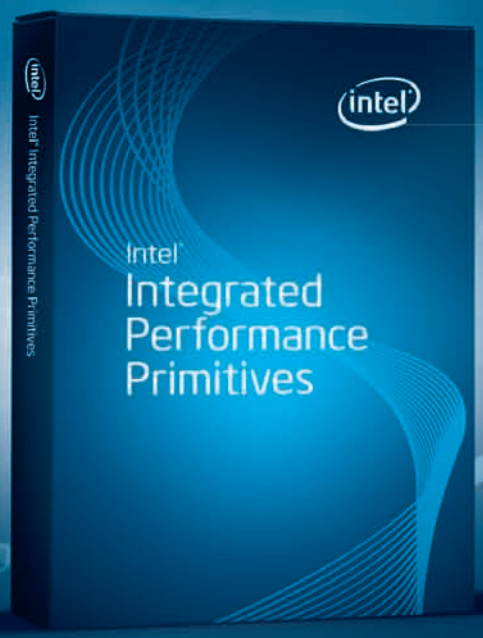
インテル® ソフトウェア製品のパフォーマンスおよび最適化に関する注意事項については、http://software.intel.com/en-us/articles/optimization-notice（英語）を参照してください。

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

インテル® Cilk™ Plus とインテル® スレッディング・ビルディング・ブロックを使用した

インテル® インテグレーテッド・パフォーマンス・プリミティブ関数の並列化

Walter Shands



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）は、マルチメディア、データ処理、通信アプリケーション向けに高度に最適化されたソフトウェア関数を備えた、マルチコア対応の広範囲なライブラリーです。インテル® IPP は、インテル® Parallel Studio XE のコンポーネントであるインテル® Composer XE と、インテル® C++ Studio XE のコンポーネントであるインテル® C++ Composer XE に含まれています。

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（英語）を参照してください。

public static exte
node;
tKind.Sequential,CharSet
struct IppiSize

インテル® IPP には、シングルスレッドとマルチスレッド・バージョンがあり、静的または動的にリンクできます。ソフトウェア開発者は、ライブラリーのシングルスレッド・バージョンをあえて利用することがあります。これにはさまざまな理由がありますが、その 1 つはインテル® IPP を使用するアプリケーションのスレッド数を制御するためです。例えば、スレッド化されたインテル® IPP ライブラリーを使用してインテル® IPP 関数がアプリケーションから呼び出された場合、インテル® IPP のスレッド・サブシステムにより、システムで利用可能なハードウェア・スレッドと同じ数のソフトウェア・スレッドが生成されます。アプリケーション・コードがより高いレベルでスレッド化されている場合、インテル® IPP 関数が呼び出されたときにアプリケーションで実行されるスレッドの総数が、利用可能なハードウェア・スレッド数を超えることがあります。これは、スレッド同期のオーバーヘッドが増加する原因となります。

それでも、インテル® IPP ライブラリーのシングルスレッド・バージョンを使用して、個々のインテル® IPP 関数をスレッド化することが重要になる場合があります。例えば、特定数のハードウェア・スレッド用にスレッド化されチューニングされたアプリケーションを、より多くのコアが搭載されたシステムで実行する場合などです。プロセッサーに負荷のかかるアプリケーションの部分が、より多くのコアで処理されると、少ないコア数では hotspot でなかったインテル® IPP 関数呼び出しが hotspot になることがあります。

インテル® Composer XE とインテル® C++ Composer XE には、シングルスレッドのインテル® IPP 関数をスレッド化するツール、インテル® Parallel Building Blocks (インテル® PBB) も含まれています。インテル® PBB とは、製品の名前ではなく、並列化の実装を支援するモデル (およびツール) のコレクションを表します。同じ基盤を共有しているため、並列実装の要件に合うモデルを組み合わせて使用できます。これらのモデルは既存のアプリケーションに容易に統合できるので、既存のコードへの投資を無駄にすることなく、並列アプリケーションを迅速に開発できます。インテル® PBB は、インテル® Cilk™ Plus、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® Array Building Blocks (インテル® ArBB) で構成されています。

これらの 2 つ、インテル® TBB とインテル® Cilk™ Plus はそれぞれ、既存のコードをスレッド化する並列アルゴリズムとキーワードを提供します。次に、インテル® TBB とインテル® Cilk™ Plus を使用してインテル® IPP 関数をスレッド化するラッパーを記述する方法について説明しましょう。

インテル® TBB は、一般的な並列化の実装に使用できる C++ テンプレート・ライブラリー・ソリューションであり、汎用ループとタスクの並列アプリケーションを記述する C++ ソフトウェア開発者向けです。スケーラブルなメモリー割り当て、負荷分散、ワークスチール・タスク・スケジューリング、スレッドセーフなパイプラインとコンカレント・コンテナー、高度な並列アルゴリズム、さまざまな同期プリミティブが含まれます。

インテル® TBB とラムダ関数は必要に応じてインテル® IPP 関数をスレッド化するのに使用できます。この基本的な概念は、parallel_for のようなインテル® TBB の並列アルゴリズムでインテル® IPP 関数をラップし、ラムダ関数を使用してコードを単純化することです。インテル® TBB のテンプレート関数 parallel_for は、反復空間をチャンク (タスク) に分割し、各チャンクを別のスレッドで実行することで for ループを並列化します。

インテル® TBB とラムダ関数を使用してスレッド化されたインテル® IPP 関数の例を次に示します (図 1)。

インテル® IPP 関数 ippiSet_8u_C1R は、通常、イメージの一部分を表す 8 ビットの符号なし整数配列の要素を特定の値に設定します。処理するイメージの対象部分は関心領域 (Region of Interest (ROI)) と呼ばれます。ROI をサポートしているインテル® IPP 関数には、名前に R 記述子が含まれています。

parallel_for では、インテル® TBB によって提供されるテンプレート・クラス blocked_range<T> で T 型の 1 次元の反復空間を指定します。このデータの範囲が分割され、個々のスレッドで実行されます。

ippiSet_8u_C1R では、値、ROI の開始地点のポインター、ROI の行間 (バイト単位)、ROI のサイズを示す構造を引数として指定します。blocked_range テンプレートに ROI の高さを行単位で渡すことで、ROI 行に基づいて反復空間を分割するように parallel_for ステートメントに指示しています。

blocked_range は反復空間をサブ空間に分割するため、ippiSet_8u_C1R の入力 ROI サイズが処理している ROI サブ空間のサイズになるように調整する必要があります。また、ROI 配列の適切な

```
inline
void inl_ippiSet_8u_C1R(const Ipp8u value, Ipp8u* pDstROI, int dstStep, IppiSize _roisize)
{
    parallel_for( blocked_range<int> (0,_roisize.height), [value, pDstROI, dstStep, _roisize ] (const blocked_range<int> &r) {
            IppiSize roisize = {_roisize.width, r.end() - r.begin()};
            ippiSet_8u_C1R( value, &pDstROI[ dstStep * r.begin()], dstStep, roisize);
    }, auto_partitioner());
}
```

図 1

 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 (英語) を参照してください。

インデックスを提供して、ROI 配列の入力ポインターが ROI サブ空間を指すように移動する必要もあります。

parallel_for の 2 つ目の引数としてラムダ関数を使用することにより、parallel_for ステートメントの外に STL スタイルの関数オブジェクトを記述することなく、parallel_for 内にすべてのコードを記述できます。さらに、ラッパー関数 inl_ippiSet_8u_C1R は余計な関数呼び出しによるコストを避けるため、インライン展開されます。

インテル® TBB ラッパーは、信頼性と移植性に富んだスケーラブルな並列バージョンの処理を提供します。インテル® IPP 関数は、ストリーミング SIMD 拡張命令 (SSE) やほかの最適化命令セットのような対象プロセッサーで利用可能な機能に基づいて低水準の最適化を提供します。

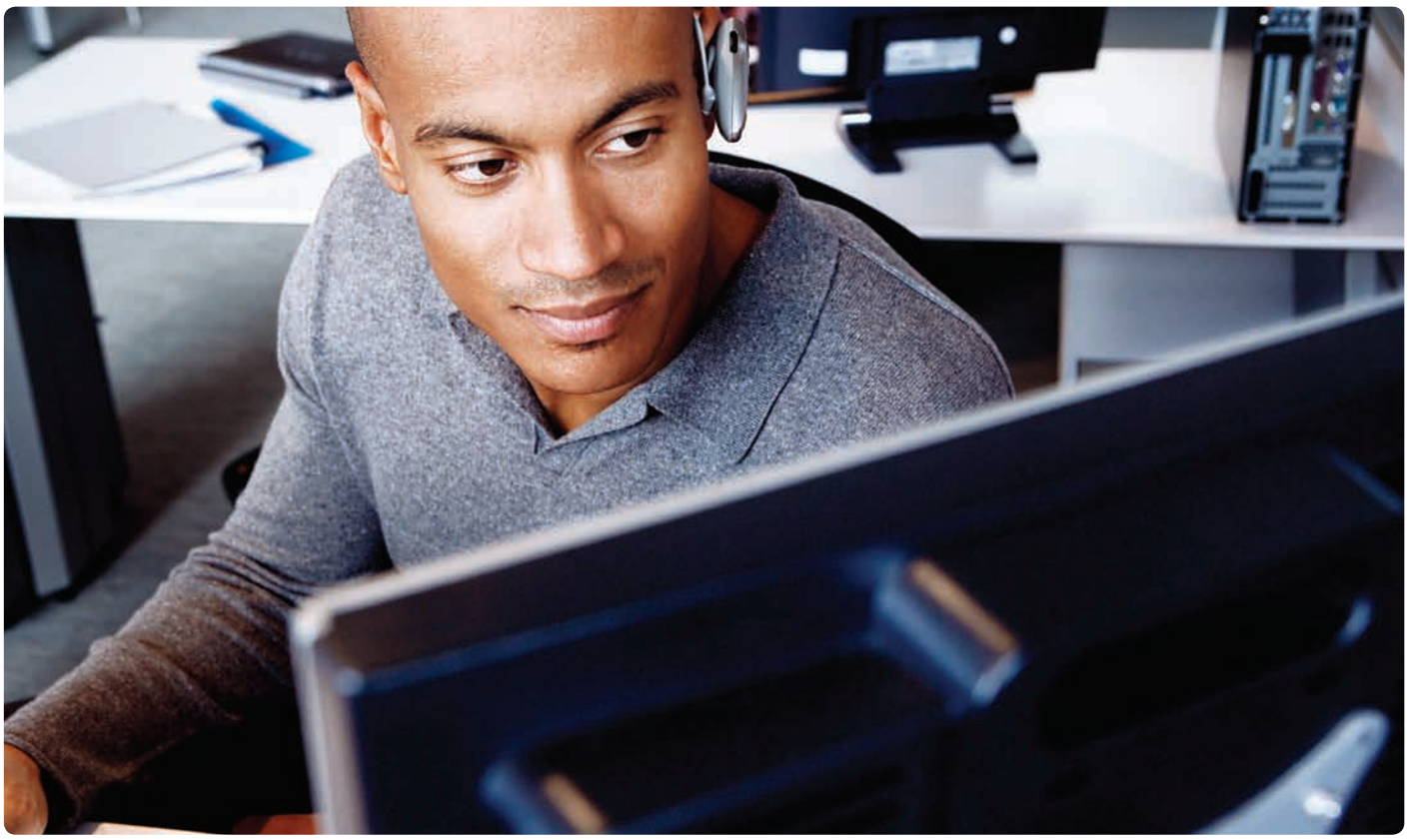
インテル® Cilk™ Plus は、インテル® C++ コンパイラーで実装される C/C++ 言語を拡張して、アプリケーションをスレッド化するための素早くかつ信頼性のある方法を提供します。インテル® Cilk™ Plus は、単純なループとタスクを使用して並列アプリケーションを作成する C++ 開発者向けです。ベクトル化機能と、高度なループベースのデータ並列処理およびタスク処理を組み合わせることで、優れた機能を提供します。

インテル® Cilk™ Plus を使用して構築された関数ラッパーをインテル® TBB ラッパーの代わりに使用してインテル® IPP 呼び出しをスレッド化できます。

インテル® Cilk™ Plus を使用して同じインテル® IPP 関数をスレッド化した例を次に示します (図 2)。

この例では、複数のコアでスケールするように、キーワード cilk_for を使用してオリジナルの for ループが並列化されています。cilk_for ループは、通常の C++ の for ループを置き換えて、ループの各反復を並列で実行します。

cilk_for ループでは ROI からのデータは一度に 1 行ずつ処理されるため、ROI の単位を 1 に設定しています。しかし、インテル® Cilk™ Plus はループで行われる処理を分割するため、複数の行を同じスレッドで処理できます。インテル® TBB のように、インテル® Cilk™ Plus はワークスチール・タスク・スケジューラーを使用して負荷分散を行い、利用可能なハードウェア・スレッドのオーバーサブスクリプションを回避します。



BLOG highlights

インテル® スレッディング・ビルディング・ブロックの Community Preview 機能を使用した「食事する哲学者の問題」の実装

MICHAEL VOSS

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）バージョン 3 Update 5 では、クラスグラフが Community Preview (CP) 機能として追加されました。一緒に使用できるクラスとノードの概要は、こちらの Web（英語）を参照してください。このリリースのオープンソース・バージョンは、www.threadingbuildingblocks.org からダウンロードできます。グラフに関するフィードバックはフォーラム（英語）までお寄せください。以前の記事では、単純なメッセージグラフを作成する例（英語）を紹介しました。今回の記事では API の特長を示す、より複雑な例を紹介します。

この例では、以下の点を説明します。

- グラフの実行関数を使用する方法
- 明示的な put と明示的な edge を混合する方法
- join_node の貪欲でない性質

この記事では、以下に示す「食事する哲学者の問題」を実装します。この問題では、数人の哲学者が食卓に一緒に座っています。それぞれの哲学者は食事をしたり考え事をしていますが、一度に 1 つのことしかできません。

この記事の続きは こちら（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）。

インテル® TBB やインテル® Cilk™ Plus のラッパーを使用して外部的にインテル® IPP 関数をスレッド化する手法は、関数に渡す引数を適切に調整することで、ほかの多くのインテル® IPP 関数にも使用できます。インテル® IPP 関数はスレッドセーフなので、入力データが保護されている限り、同じインテル® IPP 関数が異なるスレッドから呼び出された場合にデッドロックやデータ競合が発生する危険はありません。

これは、インテル® IPP 関数が内部グローバル変数を使用しないためです。あるスレッドで呼び出された同じインテル® IPP 関数が別の引数を使用して異なるスレッドで呼び出されてもステートは変更されません。

引数 1 でインテル® IPP の ippSetNumThreads API 呼び出しが行われ、ライブラリー内部の並列化が前もって無効にされている場合、スレッドラッパーをインテル® IPP スレッド・ライブラリーとともに使用できます。

さらに、インテル® TBB とインテル® Cilk™ Plus の構造は入れ子にできます。例えば、インテル® TBB の parallel_for の呼び出し内部でインテル® Cilk™ Plus の cilk_for 呼び出しを使用できます。逆も同様です。

そして、インテル® Composer XE とインテル® C++ Composer XE は Linux* と Microsoft* Windows* プラットフォームの両方で利用できます。Microsoft* Windows* プラットフォームで Microsoft* Visual Studio* を使用している場合、インテル® TBB とインテル® IPP をアプリケーションで利用できるように簡単に設定することができます。Microsoft* Visual Studio* 2010 の場合、プロジェクトのプロパティーを編集して、構成プロパティーでインテル® パフォーマンス・ライブラリー項目を選択し、インテル® インテグレーテッド・パフォーマンス・プリミティブとインテル® スレッディング・ビルディング・ブロックのドロップダウン・リスト・ボックスで [はい] を選択します。次に、インテル® IPP とインテル® TBB のヘッダーをアプリケーションのコードにインクルードします。

インテル® Composer XE とインテル® C++ Composer XE には、高度なベクトル化をサポートする最適化コンパイラーとハイパフォーマンス・ライブラリー、およびインテル® Parallel Building Blocks が含まれています。同じコードベースで Linux* と Microsoft* Windows* におけるスレッド化を促進し、単純化してパフォーマンスの向上を図ります。インテル® Composer XE とインテル® C++ Composer XE のインテル® PBB テクノロジーはインテル® IPP ライブラリーと互換性があるので、開発者はこれらのテクノロジーを活用して複数のコアでスケーリングされるソフトウェア・ソリューションを開発できます（図 3）。

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（英語）を参照してください。

```
inline
void inlc_ippiSet_8u_C1R(const Ipp8u value, Ipp8u* pDstROI, int dstStep, IppiSize _roisize)
{
        cilk_for( int i=0; i < _roisize.height; i++) {
                IppiSize roisize = {_roisize.width, 1};
                ippiSet_8u_C1R( value, &pDstROI[ dstStep * i], dstStep, roisize);
        };
}
```

図 2

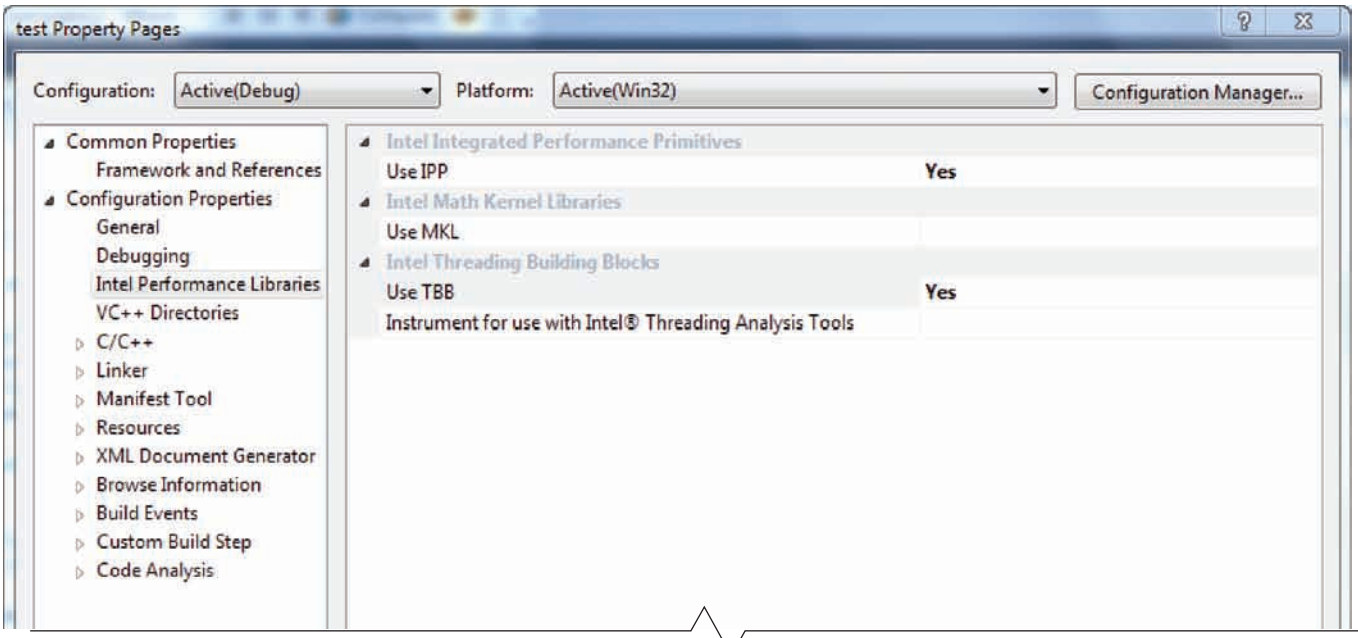


図 3

詳細は、以下のリンクを参照してください（英語）。

インテル® Parallel Studio XE: http://software.intel.com/en-us/articles/intel-parallel-studio-xe

インテル® Composer XE およびインテル® C++ Composer XE: http://software.intel.com/en-us/articles/intel-composer-xe

インテル® Parallel Building Blocks: http://software.intel.com/en-us/articles/intel-parallel-building-blocks

インテル® Cilk™ Plus: http://software.intel.com/en-us/articles/intel-cilk-plus

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 : www.intel.com/software/products/ipp

インテル® スレッディング・ビルディング・ブロック : www.intel.com/software/products/tbb

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他知的財産権の侵害への保証を含む）にも一切応じないものとします。

インテルによる書面での合意がない限り、インテル製品は、その欠陥や故障によって人身事故が発生するようなアプリケーションでの使用を想定した設計は行われていません。

インテル製品は、予告なく仕様や説明が変更される場合があります。機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を設計の前提にしないでください。これらの項目は、インテルが将来のために留保しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとしないでください。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

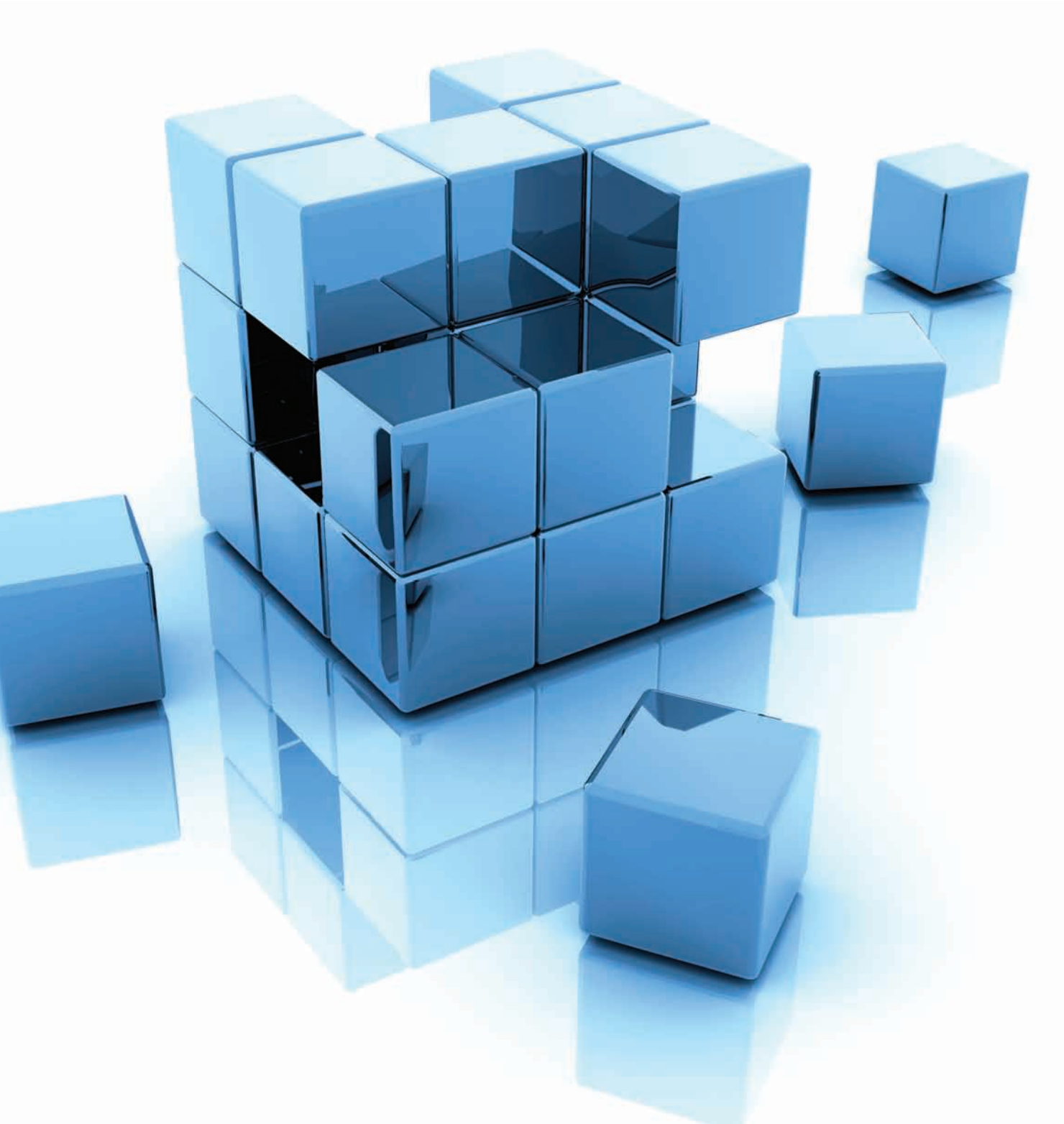
最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。

本資料で紹介されている注文番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725（アメリカ合衆国）までご連絡いただくか、インテルの Web サイトを参照してください。

http://www.intel.com/design/literature.htm

インテル® Array Building Blocks コーディング・ヒント

Zhang Zhang



 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（英語）を参照してください。

インテル® Array Building Blocks（インテル® ArBB）は動的なデータ並列化用に特別に設計された並列プログラミング・ソリューションで、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）とインテル® Cilk™ Plus に続く、インテル® Parallel Building Blocks（インテル® PBB）の最新のコンポーネントです。

インテル® ArBB の特長は、複数のターゲットに対し動的コンパイルを可能にするランタイム・システムであることです。このランタイムシステムには、JIT コンパイラーを使用して並列化、ベクトル化、最適化されたコードを動的に生成し、ターゲットのアーキテクチャーに適応する仮想マシン（VM）が含まれています。インテル® ArBB は、この VM の上位に組み込み言語として実装されます。その構文は、ISO C++ 規格と互換性のある C++ ライブラリー API として実装されます。この API を使用するプログラミング・モデルは、構造化された、決定性のある競合のないデータ並列化をサポートします。

インテル® ArBB でプログラミングを行う開発者は、そのアルゴリズムに集中できます。優れたパフォーマンスを得るために SIMD 命令、スレッド管理、組み込み関数のような低水準のアーキテクチャー上の機能を処理する必要はありません。これらの機能はランタイムシステムが処理します。ただし、ランタイムシステムがハイパフォーマンスなコードを生成するようにプログラマーが支援できることもあります。インテル® ArBB には、エラーを回避するためによく理解しておいたほうが良い点がいくつかあります。次に、これらのヒントを要約し、サンプルコードを使って説明します。

コーディング・ヒント 1:
並列化の表現

インテル® ArBB を使用してアルゴリズムをコードディングするときにプログラマーが最初にする質問は、「どのように並列性を表現すれば良いか?」でしょう。データ並列化用の最適な言語になるように設計されているインテル® ArBB では、2 つの方法で高水準な並列化を実装できます。

1. データ・コレクションにベクトル演算を直接使用する

ベクトル演算はコンテナーを引数とする関数です。コンテナーの要素における計算は独立しているか、あるいは既知の並列化の手法があるため、これらの演算を使用すると並列化されます。例として、m 次元空間の 2 点間のユークリッド距離の計算を考えてみましょう。これらのポイント間の距離は次の公式を使用して計算します。

$$d = \left(\sum_{i=1}^{m} | x_i - y_i |^2 \right)^{1/2}$$

n ポイントあると仮定すると、結果には n の距離が含まれます。

この計算をコード化する 1 つの方法は、複数の座標が必要な各ポイントのコレクションを表す 2-D dense コンテナーを使用することです。2-D dense コンテナーの幅は n（ポイント数）、高さは m（ポイントの次元）になります。図 4 にコードを示します。距離関数は 2 つのコンテナー（X と Y）全体で読み取りと処理を行い、別のコンテナー（d）全体に出力を書き込みます。

2. データ・コレクションに要素関数を適用する

要素関数（つまり、カーネル関数）は、データ・コレクションとは対照的に、個々のデータ要素を処理するように記述されています。要素関数は、コンテナーにわたって関数をマップする arbb::map 演算子を使用することで、コンテナーのすべての要素に対して並列に実行できます。図 5 に複素平面のマンデルブロ集合を計算する例を示します。カーネル関数は個々の複素数の二次多項式を評価します。次に、このカーネル関数へのポインターを引数として arbb::map オペレーターに渡し、

```
std::size_t length = N, dim = M;
dense<f64, 2> vec_X(length, dim), vec_Y(length, dim);

void distance(dense<f64, 2> X, dense<f64, 2> Y, dense<f64>& d) {
    dense<f64, 2> sqdiff = (X ^ Y) * (X ^ Y);
    dense<f64> s = add_reduce(sqdiff, 1);
    d = sqrt(s);
}
```

図 4. ベクトル演算を使用して並列化を実装する

使用されている 2-D dense コンテナー D と C のすべての要素にわたってマップします。**ここで使用されている _for は、並列 for ではなく、入力配列の 1 つの要素を処理するシリアル制御フロー構文であることに注意してください。**

コーディング・ヒント 2:
パフォーマンスのプログラミング

インテル® ArBB の仮想マシンはコード生成中にさまざまな最適化を行います。最適化はすべてユーザーに透過的に行われます。ユーザーは、より多くの最適化の機会を許可し、仮想マシンが効率的なコードを生成するように、ベスト・プラクティスのセットを採用することができます。

1. arbb::expect_size を使用して最適なメモリー割り当てとキャッシュフレンドリーなデータ提供を行う

インテル® ArBB は、データの安全性とパフォーマンスの理由により、通常の C++ オブジェクトから分離されたメモリー空間でインテル® ArBB コンテナー・オブジェクトを割り当てて管理する、分離メモリー管理モデルを採用しています。コンテナーが関数に渡される前にコンテナーのサイズが分かっていれば、ランタイムシステムはメモリーを事前に割り当てて、キャッシュの使用、ベクトル化、マルチスレッド化が効率的に行われる方法でデータをスライスして配置できます。そのために必要なのは、ユーザーが arbb::expect_size を使用して JIT コンパイラーにサイズの期待値を伝えることです。図 6 に 1-D、2-D、3-D dense コンテナーに形状の期待値を渡す例を示します。

arbb::expect_size ステートメントはインテル® ArBB コンテナーの引数を処理する関数の最初のステートメントでなければなりません。コンテナーの各次元についてサイズを指定する必要があります。サイズは整数定数、C++ 変数、または整数を評価する式として指定できます。サイズはランタイムに計算できますが、インテル® ArBB クロージャーの構築に使用されるサイズと、**クロージャーが実行されるときに使用される実サイズは同じでなければならないことに注意してください。**

```
arbb::expect_size(1d_dense, width);
arbb::expect_size(2d_dense, width, height);
arbb::expect_size(3d_dense, width, height, depth);
```

図 6. 形状の期待値を渡す

2. インテル® ArBB でメモリー管理を行って不要なデータのコピーを防ぐ

コンテナーを作成する方法は 2 つあります。インテル® ArBB ハンドルを既存の C++ 配列にバインドする方法と、インテル® ArBB でマネージメモリー空間にコンテナー全体を作成する方法です。最初のアプローチは既存の C++ データを使用して容易に処理できますが、ランタイムシステムは、処理の前にマネージメモリー空間にデータをコピーし、処理の後に元の空間にデータを戻さなければなりません。余分なデータの移動はパフォーマンスに影響を与え、追加の同期も必要になります。

一方、インテル® ArBB のメモリー管理を使用すると、コンテナーが存在する間、関連するデータはランタイムメモリー空間に存在できます。明示的に指定されない限り、マネージメモリー空間と通常のメモリー空間の間でデータのコピーは行われません。また、ランタイムはアライメント要件をより適切に（例えば、ハードウェア・プラットフォームに合わせて）管理できます。図 7 と 8 に両方のアプローチの例を示します。

```
static const int max_count = 1000;

// カーネル関数
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;
}
// map() を使用してカーネル関数を実行
void doit(dense<i32, 2>& D, dense<std::complex<f32>, 2> C) {
    map(mandel)(D,C);
}
// C++ コード内から
call(doit)(dest, pos);
```

図 5. map 演算子を使用してデータ・コレクションの各項目に要素関数を適用する

 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（英語）を参照してください。

```
dense<f32> a(1024), b;
// 書き込み範囲を使用して‘a’を初期化
range<f32> ra = a.write_only_range();
for (size_t i = 0; i < ra.size(); ++i) {
    ra[i] = ...;
}

// ‘a’のコピーイン
call(fun1)(a, b);
// ‘b’のコピーアウトなし

// ‘a’のコピーインなし
call(fun2)(a, b);
// ‘b’のコピーアウトなし

// ‘b’のデータは必要なときに取得
const_range<f32> rb = b.read_only_range();
std::vector<float> result(rb.size());
std::copy(rb.begin(), rb.end(), result.begin());
```

図 7. インテル® ArBB でメモリー管理を行う

```
dense<f32> a, b;
bind(a, c_arr, 1024);
bind(b, c_brr, 1024);
// ‘a’のコピーイン
call(fun1)(a, b);
// 同期；‘b’のコピーアウト
// ‘a’のコピーイン
call(fun2)(a, b);
// 同期；‘b’のコピーアウト
```

図 8. bind インターフェイスは便利だが本来必要のないコピーが行われる

3. 集合操作は控えめに使用する

多くのアルゴリズムの実装には、リダクション、スキャン、ギャザー、スキャッター、パーミュテーションなどの集合操作が必要です。しかし、集合操作は一般的に同期が必要な不規則性を計算フローにもたらします。同期はランタイムがコードの大きな塊（チャンク）にわたって最適化を行う機会を大幅に減らします。集合操作の使用の経験則は次のとおりです。

- 集合操作は控えめに必要な場合のみ使用する。
- 可能であれば、集合操作を使用する関数の始めまたは最後に集合操作を移動する。

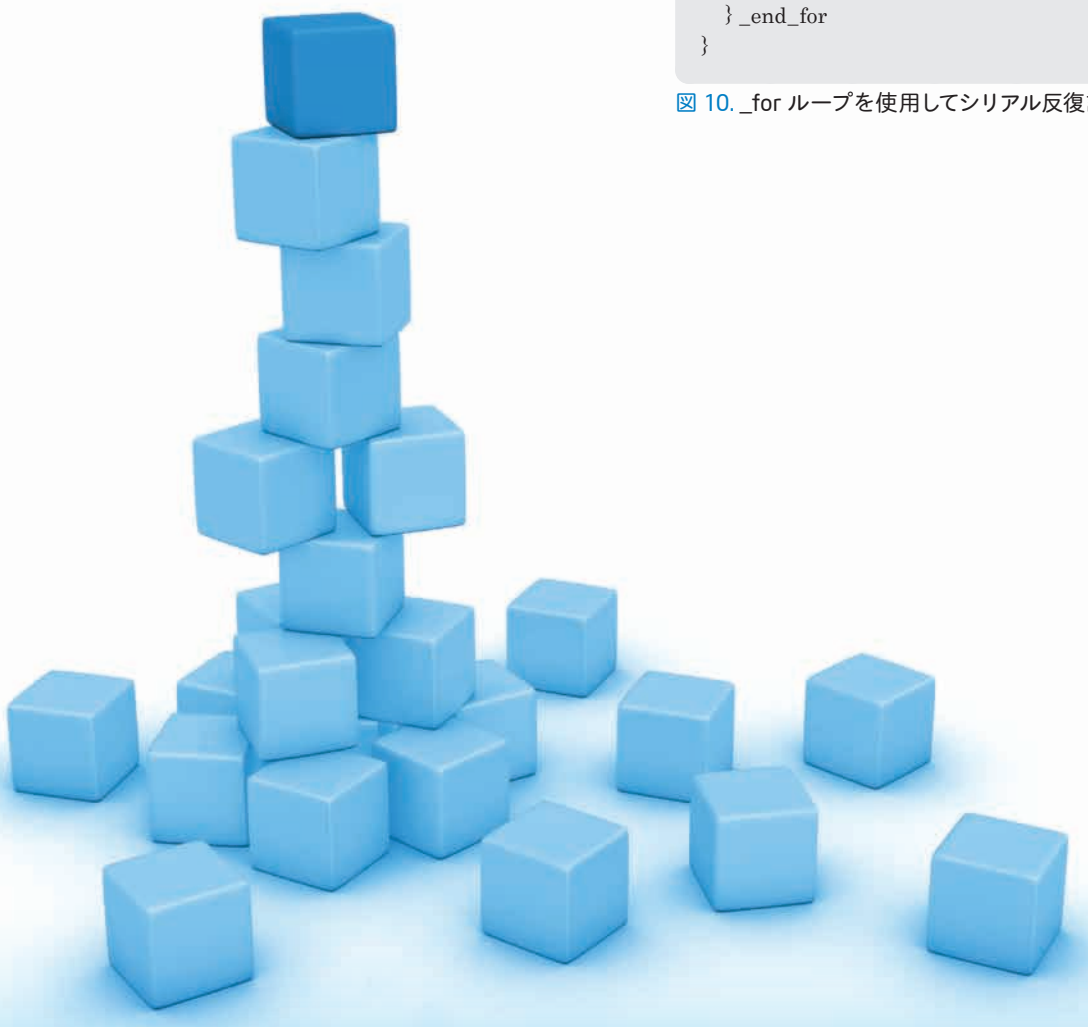
4. 専用の関数を使用して不要なデータ依存を回避する

ランタイムが効率的にコードを生成できるように、プログラマーは不要なデータ依存が起こらないようなコードを記述するべきです。これは 2 つのことを意味します。まず、専用の演算子を使用して 1 ステップで処理を完了できる場合は、より一般的な操作を使用して多くのステップで処理を行うべきではありません。次に、コンテナーの中間コピーはできるだけ避けてください。特に、不必要に一時変数を再利用しないでください。コレクションの一部を更新するだけであれば、操作を論理的に独立させるために新しい一時変数を使用してください。

図 9 にコンテナーの値をある範囲に制限する 3 つの方法を示します。すべて同じ結果になりますが、最初の 2 つの方法はそれぞれ 2 つのステートメントが必要です。1 番目の方法では一部のみ更新されたコンテナーのシャドーコピーが中間値として使用されており、2 番目の方法には不要なシリアル依存があります。対照的に、3 番目の方法は 1 つの専用ステートメントを使用して同じ結果を引き出し、最適なコードを生成できます。

```
dense<f32> x = ..;

// 'x' に依存性がある場合
x = select(x < 0.0f, 0.0f, x);
x = select(x > 255.f, 255.f, x);

// 'x' に依存性がある場合
x = max(0.0f, x)
x = min(255.f, x);

// 最適なコード
x = clamp(x, 0.0f, 225.f)
```

図 9. 不要なデータ依存を回避して利用可能な専用の操作を使用する



コーディング・ヒント 3:
シリアル制御フローに _for プロパティーを使用する

ユーザーは _for ループが並列化されているという誤った印象を持つことがありますが、インテル® ArBB で提供されている _for ループおよび他のすべてのループはシリアルループです。_for ループの反復はシーケンシャルに実行されます。インテル® ArBB は _for ループを自動並列化しません。つまり、_for ループはデータ並列の実装に使用できません。インテル® ArBB で並列化を適切に実装する方法は、コーディング・ヒント 1 を参照してください。

_for ループはシリアルに依存する反復計算を表現するように設計されています。計算は段階的に行われ、現在のステップは前のステップの結果に依存します。図 10 で示されている反復ステンシル計算は良い例の 1 つです。このコードでは、ステンシルベースの更新ステップはそれぞれ arbb::map 演算子を使用して並列化されていますが、ソリューションを計算するために複数回の更新が順に行われます。

```
void apply_stencil(dense<f64, 2>& grid, i32 iterations)
{
    _for(i32 i = 0, i < iterations, ++i)
    {
            map(stencil)(grid);
    } _end_for
}
```

図 10. _for ループを使用してシリアル反復計算を表現する

 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（英語）を参照してください。

コーディング・ヒント 4:
インテル® ArBB の制御フローにインテル® ArBB の式を使用する

インテル® ArBB の制御フロー・ステートメント（_for、_while、_do_until、_if/_else、その他）は、C++ ステートメントのように、条件式を使用して構築します。ただし、条件式にはインテル® ArBB の型を使用する必要があります。C++ の型の条件変数を使用するのはよくある間違いです。図 11 は _for ステートメントの誤った例と正しい例を示しています。この違いを理解するには、C++ の型を含むステートメントは、JIT 時（つまり、キャプチャー時）にすべて実行されるということを覚えておいてください。これらのステートメントの効果は生成されたコードにも「引き継がれ」、インテル® ArBB 実行の間は変更されません。

```
_for(int i = 0, i < iterations, ++i) { // 誤
    ...
} _end_for

_for(i32 i = 0, i < iterations, ++i) { // 正
    ...
} _end_for
```

図 11. 条件変数に C++ の型を使用しない

コーディング・ヒント 5:
インテル® ArBB の制御フローから正しく出る

インテル® ArBB の制御フローに関してもう 1 つのよくある間違いは、_if/_else 分岐の外にジャンプする（またはインテル® ArBB のループの外に出る）return ステートメントの使用です。インテル® ArBB の制御フロー・ステートメント内部では return ステートメントを使用できません。使用した場合、インテル® ArBB の制御フロー・ステートメントが正しく入れ子されないため、ランタイムエラーになります。このような状況は容易に解決できます。

ループ構造（_for、_while、_do/_until）については、インテル® ArBB が提供する _break ステートメントを使用してループを適切に終了できます。_if/_else 分岐については、return ステートメントを使用しないようにユーザーがコードを再構築する必要があります。図 12 の例は _break ステートメントと分岐から出る正しい方法を示しています。

DEVELOPER SPOTLIGHT

菅原清文
シニア・ソフトウェア・エンジニア

インテルでの勤続期間
14 + 6 年です。

これまでの経歴
ソフトウェア会社で日本での重要なテクノロジーの立ち上げを支援してきました。これまで Unix* オペレーティング・システムの開発、組み込みファームウェアの開発、コンパイラー等ツールの開発、Windows* 64 ビット・システムの開発者のサポート、Mac OS* X への統合支援などを行ってきました。

現在の取り組み
さまざまな機会を設けてソフトウェア開発者や学生の方々に並列プログラミングの重要性を伝えています。また、プロセッサーの能力をより引き出せるようにソフトウェア最適化のコンサルテーションも行っています。

この取り組みの重要性
ソフトウェアのチューニングに詳しいソフトウェア・エンジニアは比較的少なく、最新のプロセッサーやマルチコアの能力を活用できていないのが現状です。優れた並列プログラミングを知ることで、ソフトウェアを開発するときにマルチコアやプロセッサーを考慮する必要性が理解できるようになります。ソフトウェアのチューニングと並列化のポイントを明確にすることで、ソフトウェア開発における関連ツールの重要性を示すことができます。

ソフトウェア開発者としての目標
開発者に役立つ重要なテクノロジーを分かりやすく説明した書籍を執筆することです。

```
// _break を使用してループの外に出る
_for(int i = 0, i < iterations, ++i) {
    ...
    _break;
} _end_for

// これはランタイムエラーとなる
_if(x == 0) {
    return;
} _else {
    ...
} _end_if

// これは OK
_if(x != 0) {
    ...
} _end_if
```

図 12. インテル® ArBB の制御フローから正しく出る

コーディング・ヒント 6:
インテル® ArBB 変数の値はデバッグモードで調べる

インテル® ArBB の変数は、生成されたコードが実行されるときにのみ値を保持する不透明なハンドルで、「キャプチャー」(JIT) 時には値がありません。このため、printf や std::cout ステートメントを使用して「呼び出し」の内部で呼び出された関数内でインテル® ArBB 変数の値を調べることはできません。これらのステートメントは値がない JIT 時にのみ実行されます (図 13 を参照)。インテル® ArBB 変数の値を調べるには、デバッガーを使用するか、環境変数 ARBB_OPT_LEVEL を O0 に設定して printf または std::cout ステートメントを使用します。デバッグを行うことと ARBB_OPT_LEVEL=O0 に設定することは同じです。どちらも「エミュレーション・モード」で実行します。このモードでは、インテル® ArBB の操作は JIT コンパイルではなく通常の C++ 実行を使用してエミュレートされます。このモードはデバッグのために用意されているもので、パフォーマンスを引き出すことはできません。パフォーマンスを引き出すには、値を調べた後、ARBB_OPT_LEVEL を O2 または O3 に設定してください (図 13)。

コーディング・ヒント 7:
暗黙の型変換は未サポート

C/C++ プログラマーはインテル® ArBB がインテル® ArBB の型を自動的に変換しないことに最初は驚きます。例えば、図 14 は浮動小数点のコンテナーに整数 factor を掛けようとしている関数定義を示しています。関数が C++ の型で記述されている場合、コンパイラーは整数 factor を浮動小数点値に自動的に「変換」しますが、インテル® ArBB のコードでは整数 factor を明示的にキャストする必要があります。

暗黙の型変換はベクトル化のコンテキストにおける負荷が高く (レーン変更や中間値の生成などが必要)、エラーが発生しやすくなるため、インテル® ArBB では暗黙の型変換を意図的にサポートしていません。基本的に、暗黙のキャストを行うと、見えないところで潜在的に負荷の高い余分な処理が行われます。負荷の高い処理はすべて制御できるように明示的であるべきというのがインテル® ArBB のポリシーです。プログラマーは明示的なキャストを常に行うことができます。C と C++ のキャストはどちらも動作しますが、(例で示しているように) C++ のキャストを一般に推奨します。この規則は定数にもあてはまります。このため、定数も正しい型で記述する必要があることに注意してください。例えば、「1」の単精度の浮動小数点値は 1.0 ではなく 1.0f と記述する必要があります。1.0 だと倍精度実数型になります (図 14)。

```
void arbb_func(i32 var) {
    ...
    printf ("%d", var);            // O2 と O3 では動作しない
    printf ("%d", value(var));     // O2 と O3 では動作しない
    ...
}
```

図 13. この printf は O2 と O3 モードでは動作しない

```
void arbb_func(i32 factor, dense<f64>& vec) {
    vec = vec * factor;                          // 誤
    vec = vec * std::static_cast<f64>(factor);   // 正
}
```

図 14. インテル® ArBB は明示的なキャストをサポートしない

 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 (英語) を参照してください。

コーディング・ヒント 8:
インテル® ArBB の多次元コンテナーのインデックス規約に注意する

多次元コンテナーはメモリー内で行優先のレイアウトになっています。しかし、そのインデックス規約はインテル® ArBB と C/C++ で異なります。具体的には、インテル® ArBB の多次元コンテナー (2-D または 3-D dense コンテナー) は常に、幅、高さ、深さの順にインデックスされます。以下の状況ではこの点に注意する必要があります。

- 各次元のサイズが必要な多次元コンテナーを宣言またはバインドする場合、幅、高さ、深さの順に指定します。図 15 の例を参照してください。
- インデックスを使用して個々の要素にアクセスする場合、列のインデックス、行のインデックス、ページのインデックスの順になります。図 16 の例を参照してください。
- arbb::neighbor 演算子を使用して隣接する要素にアクセスする場合、列のオフセット、行のオフセット、ページのオフセットの順になります。図 17 の例を参照してください。

```
void stencil(f64& e) {
    f64 N = neighbor(e, 0, -1);
    f64 S = neighbor(e, 0, +1);
    f64 W = neighbor(e, -1, 0);
    f64 E = neighbor(e, +1, 0);
    ...
}
```

図 17. 2-D dense の隣接するセルへのアクセス

まとめ

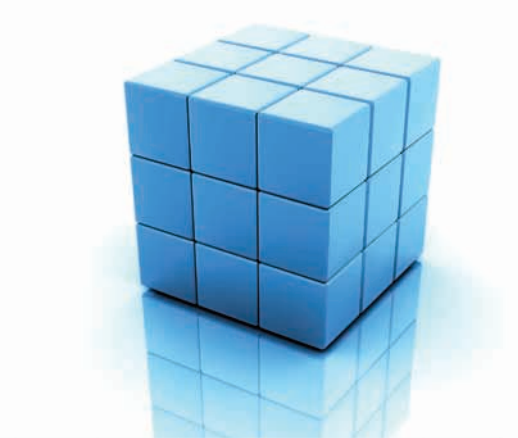
インテル® ArBB は、データ・コレクションを並列で処理するコードを容易に記述できるインターフェイスを提供します。構造化された、分かりやすく保守可能なコードを記述できるので、開発時間を短縮できます。ランタイムは SIMD ユニット、複数のコア、さまざまなアクセラレーター・デバイスにパフォーマンスの移植性とスケーラビリティーを提供します。構文と正しい使用法を理解することで、生産性とパフォーマンスを向上させることができます。

```
double c_mat[NROW][NCOL];

dense<f64, 2> matA(NCOL, NROW);  // c_mat と同じ形状

dense<f64, 2> matB;
bind(matB, &c_mat[0][0], NCOL, NROW);// NCOL と NROW の順に注意
```

図 15. 多次元コンテナーの宣言とバインド

```
void arbb_func(dense<f64, 2>& m) {
    ...
    f64 e = m(2, 3);
    ...
}
```

図 16. 2-D dense の個々の要素へのアクセス



インテル® ArBB のプログラミング手法の詳細は、以下のリンクを参照してください (英語)。

インテル® ArBB ナレッジベース : http://software.intel.com/en-us/articles/intel-array-building-blocks-kb/all/1/

インテル® オンラインフォーラム : http://software.intel.com/en-us/forums/intel-array-building-blocks/

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他知的財産権の侵害への保証を含む) にも一切応じないものとします。

インテルによる書面での合意がない限り、インテル製品は、その欠陥や故障によって人身事故が発生するようなアプリケーションでの使用を想定した設計は行われていません。

インテル製品は、予告なく仕様や説明が変更される場合があります。機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を設計の前提にしないでください。これらの項目は、インテルが将来のために留保しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとしないでください。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。

本資料で紹介されている注文番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725 (アメリカ合衆国) までご連絡いただくか、インテルの Web サイトを参照してください。http://www.intel.com/design/literature.htm

導入事例の紹介

最高のビデオ品質とパフォーマンスを保証

インテル® Parallel Studio XE で安心かつセキュアなコードを作成

会社名 : **Envivio Inc.**
業種 : **ビデオ圧縮**

「開発段階の初期にこのツールを使用したことで、エンジニアは本来、目を通す必要がないバグレポートから解放され、製品サイクルの最初から効率的にコードを作成できるようになりました。」

Envivio 社
高度画像処理チーム・マネージャー
Jean Kypreos 氏

会社概要

あらゆるビデオを、いつでも、どこでも。Envivio のゴールは、世界中のビデオコンテンツを、さまざまなビューアー、デバイス、ネットワークを介して、あらゆる所でいつでも楽しめるようにすることです。2000 年の会社創立以来、ビデオ・エンコーディング・テクノロジーの発明者として、Envivio は数多くの特許を取得してきました。VoIP（Video over IP）技術のパイオニアでもあり、新しいテクノロジーへの対応でも常に業界をリードしています。今日では、Envivio のソリューションは、世界中のあらゆる市場のコンテンツの所有者、テレコム・オペレーター、ケーブル・サテライト会社、モバイル・サービス・プロバイダーに代わって、さまざまなスタイルの携帯電話、セット・トップ・ボックス、PC プラットフォームに、膨大なコンテンツストリームを配信しています。Envivio の研究所で開発された標準 H.264 から Elite および Extreme までのコーデックを使用して、Envivio はあらゆる画面、あらゆるネットワークにおけるビデオ表示を最適化しています。事業展開とサポートは、世界中のさまざまな地域の技術、規制、市場のガイドラインに精通したグローバルなスタッフにより提供されています。Envivio の本社はサンフランシスコにあり、レンヌ（フランス）、北京、東京、シンガポールに支社があります。詳細は、**www.envivio.com** を参照してください。

目標

4Caster C4* 製品の品質を改良して、最高のビデオ品質で最高のパフォーマンスの製品を迅速に市場に投入すること。

製品

4Caster C4* は、IPTV 向けに 1 チャンネルの HD または多チャンネルの SD エンコーディング、VGA 解像度までのインターネット TV エンコーディング、3G モバイル TV エンコーディングをサポートするリアルタイム・トランスコード・アプリケーションです。

課題

コードのサイズが大きくなるにつれ、単純な IDE では大規模なコードのセクションを適切に理解することが困難になります。より大きなコードでは複雑さが増すことでバグの数も増加するため、信頼性が損なわれ、市場への提供も遅くなりがちでした。

効果

エンコーディング・チームに報告されるバグの数が少なくなり、開発者の生産性と市場への投入時間がどちらも改善されました。

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

Envivio の製品で得られた並列化のメリット

フォーマットによっては 1 つのエンコーディングの実行にも複数のコアが必要です。特に、MPEG2 から H.264 に HD コンテンツを変換するにあたって、シングルコアではリアルタイム・エンコーディング処理を満足にできません。アプリケーションの処理能力を満たすにはマルチコアとマルチプロセッサー（DP）テクノロジーの両方が必要になります。

SD コンテンツの場合でも、放送市場向けのハイレベルなビデオ品質を提供するには、リアルタイムという制約を受けたまま重い処理のアルゴリズムを使用する必要があります。同時に密度のモジュールレベル（プラットフォームあたり 2/4 SD チャンネル）を維持するには、並列化（チャンネルあたり 2/4 コア）手法を使用してプラットフォームの能力を引き出すしかありません。

問題となる密度は BOM コストの経済方程式（プラットフォームごとのチャンネル数が多くなると、COGS（売上原価）が少なくなる）から求めることができます。Envivio のアプリケーションでは、H.264 コーデックが最も「タイムクリティカルな」ソフトウェア・コンポーネントです。このため、このモジュールの並列化のバランスには特に注意が必要です。

結果

インテル® Parallel Studio XE のスタティック・セキュリティー解析（SSA）機能を使用してサンプルコードを解析した結果、意図的に挿入したバグがすべて検出されました。次に、実際のテストケースとして 4Caster C4* のエンコーディング・コアを SSA で解析しました。解析結果に基づいてコードの未使用セクションを削除したところ、パフォーマンスが向上しました。

潜在的なメモリーエラーが見つからなかった場合でも、エラー管理が追加され、潜在的な null ポインターエラーの追跡が行われたことで、コードの多くの部分で安全性が向上しました。SSA の最後のテストでは、エンコーダーのフロントエンドがセキュアになるように、入出力インターフェイスに特に注意を払い、このクリティカル・モジュールの誤使用が直ちに検出されるようにしました。

最初の解析で SSA は約 300 のエラーを検出しましたが、そのほとんどはポインターの使用法に関するものでした。重大なバグは 5 つ、マイナーバグは 3 つで、このほかに約 50 の脆弱性が検出されました。バグが検出されたセクションのコードは書き直し、ほとんどの未使用コードは削除して、多くのポインターがセキュアになりました。

結果、より信頼性の高いコードが統合チームに送られ、エンコーディング・チームにバグが返される可能性が非常に少なくなりました。

利用したインテル® ソフトウェア開発製品の機能

インテル® Parallel Studio XE の SSA 機能を製品のコアモジュールである H264 エンコーダーの解析に使用しました。製品の生産性の向上と納期の遵守を両立させるため、検証時間を短縮できるように、感度を最大限に設定しました。

SSA はコードに強固な環境をもたらします。開発者はコードを信頼してソフトウェアのモジュールを安全に結合できます。安心かつセキュアなコードからは直ちにエラーが返されるため、長い時間をかけてデバッグを行う必要がなくなります。また、SSA を開発の初期段階に使用することで、バグを発生前に修正して、時間とリソースを節約できます。さらに、開発の初期段階にインテル® Inspector XE と組み合わせることにより、ダイナミック・エラーに対する優れた障害耐性ももたらされます。

SSA ツールは容易にセットアップ、起動、使用することができます。すべての結果がインテル® Inspector XE 内に表示されるため、1 つのインターフェイスをマスターするだけで済みました。ソースコードを簡単に表示でき、問題を数分で発見して解決できました。

SSA はコンパイラーに完全に統合されており、さまざまな方法で結果を操作できます。ユーザーフレンドリーなインターフェイスとフィルターにより、開発者は問題のカテゴリーを絞り込むことができました。ドキュメントも充実しており、ややこしい高度なコーディング問題を明確に理解できました。

導入事例の紹介

思わず引き込まれるリアルな PC ゲームの世界を実現

The Creative Assembly の新しい Napoleon: Total War ゲームのスレッド化とグラフィックの課題をインテル® ソフトウェア開発製品が解消

会社名 : **The Creative Assembly**
業種 : **ゲーム開発**

「インテル® TBB に興味を持った理由は 2 つあります。既存の直列コードに統合できる容易さ、そして、習得に必要な時間が短いことでした。」

The Creative Assembly 社
シニア・エンジン・コーダー
Yuri O' Donnell 氏

会社概要

1987 年に創立された The Creative Assembly はイギリスのサセックスを本拠地とするコンピューター･ゲームの開発会社で、戦略･戦術シミュレーションの Total War*（トータルウォー）シリーズで有名です。Napoleon: Total War*（ナポレオン : トータルウォー）はフランチャイズにおける 5 つ目のメジャーリリースで、これまでの製品と同じように、PC 版のみのリリースとなります。Spartan: Total Warrior*（スパルタン ～ 古代ギリシャ英雄伝 ～ ）や Viking: Battle for Asgard* は家庭用ゲーム機市場で大ヒットしました。2003 年から 2005 年まで、Total War* のエンジンは国際的な称賛を得た BBC のゲームショー「Time Commanders」に使用されていました。2005 年に、The Creative Assembly は SEGA の子会社になりました。詳細は、**www.creative-assembly.co.uk/** を参照してください。

目標

リアルタイムの戦闘シーンとキャンペーン・マップで順を追って行われる戦術のやり取りをバランス良く融合させて、誰もが思わず引き込まれるリアルなゲームを作成すること。

製品

Napoleon: Total War* は数々の受賞暦を誇る The Creative Assembly の Total War RTS* フランチャイズによる新しいシリーズの最初の作品です。Napoleon: Total War* はフランチャイズの陸上と海上で繰り広げられる最先端の 3D 戦闘シーンをそのまま再現しています。ターンベースのキャンペーン・マップは 3 部構成からなり、ナポレオンの有名な戦争を通して彼の栄光と落日を描きます。

課題

没入型でリアルなゲーム環境の開発をサポートするため、マルチコア PC の能力を活用する必要がありました。

効果

The Creative Assembly のプログラマーは、提供されるコアの数に応じてスケーリングされるストラテジー・ゲーム用の将来性の高いエンジンを作成し、プロセッサーによる制限から解放されました。

The Creative Assembly の製品で得られた並列化のメリット

処理能力とグラフィック描画能力が高ければ高いほど、ゲームの見栄えは良くなり、ゲームに引き込まれる魅力もより強くなります。Napoleon: Total War* における詳細なカスタマイズはマルチスレッド化によって可能になりました。シングルコア・プロセッサー上でも動作は可能ですが、ゲームエンジンはデュアルコア以上のシステムを念頭にコード化されており、ロジックとレンダリングの 2 つのパーツに分割されています。これらのルーチンはさらに、海の形状の再描画、経路の検索のような個々の「ミニタスク」アルゴリズムに細分化されています。あるプロセスが別のプロセスの妨げにならないようにすることで、ゲームは利用可能なハードウェアの能力をフル活用できます。

結果

以前よりもシステム要件が低くなったにもかかわらず、より大きなアニメーションが追加されたことで、Napoleon: Total War* が提供するユーザー体験はよりリアルになり、圧倒的な臨場感を実現しました。開発チームは、インテル® スレッディング・ビルディング・ブロック（インテル® TBB）なしでは予定通りに製品を完成できなかっただろうと確信しています。

エンジンの効率が向上した結果、Napoleon の戦場や海上における戦闘で、それぞれが個々の AI ルーチンを持つユニットを広大かつ大量に配置できました。Napoleon では 1 万から 2 万におよぶキャラクターによる戦闘が行われ、それらのキャラクターは、画面に 10 から 20 のキャラクターしか登場しないゲームのキャラクターに匹敵するほどの LOD（level of detail）で作成されています。このゲームの鍵となる新しいツールの 1 つはカスタム・ユニット・ビルダーで、デザイナーがモデルを詳細にカスタマイズできます。Napoleon は以前の製品よりもシステム要件が低くなったにもかかわらず、Total War* シリーズで最も高度な表現が可能になったのは画期的なことです。

利用したインテル® ソフトウェア開発製品の機能

The Creative Assembly の開発チームは現在のゲームエンジンを設計するときにインテル・エンジニアの経験とインテル・ツールを最大限に活用しました。

すべてのメモリー操作の検証にはインテル® スレッド・チェッカーが役立ちました。共有メモリーに対する同時アクセスを通知すると、競合状態が検出され改善できました。さらに役立ったのは、開発者がコードの並列化に使用したインテル® TBB ライブラリーです。既存のシリアルコードに容易に統合できるだけでなく、プログラマーが使用法を素早く習得できるため、数日後にはコードのパフォーマンスが向上し始めました。

複数のプラットフォームではなく PC に限定することの利点の 1 つは、インテル® TBB をエンジンに完全に統合させることで、大幅に時間と費用を節約できる点でした。独自のジョブ・キュー・システムの開発を回避できたことで、開発、テスト、デバッグにかかる労力が節約できました。開発チームは、インテル® TBB なしでは予定通りに製品を完成できなかっただろうと確信しています。

システム要件を低くするためのチューニングにはインテル® グラフィックス・パフォーマンス・アナライザー (インテル® GPA) を使用しました。開発者は各 API 呼び出しのタイミングを調べて GPU の使用時間を確認し、その情報をもとにレンダリング状態を修正してパフォーマンスを調整しました。レンダリング・パイプラインのリアルタイム解析機能は、ボトルネックの特定と、各シーンにおけるオーバードローの多いインスタンスの発見に役立ちました。

導入事例の紹介

仮想衝突試験における新しい標準の作成

Altair の正面衝突シミュレーションを進化させたインテル® ソフトウェア開発製品

会社名 : **Altair Engineering, Inc.**
業種 : **製造業**

「CAE を駆使した設計と自動車の安全性との間にあったミッシングリンクが見つかりました。 我々のハイブリッド・ソルバーによる新しいアプローチを組み合わせることで、 仮想衝突試験につきものであるターンアラウンド・タイムによるボトルネックをなくすことができました。 衝突、 耐久性、NVH を複合的に最適化したことで、今後は設計工程に貴重なデータを提供することができるでしょう。」

Altair Engineering, Inc. 社
HYPERWORKS 部門 CTO
Uwe Schramm 博士

会社概要

Altair Engineering, Inc. は、ビジネスおよびエンジニアリング情報の解析、管理、可視化を最適化するテクノロジーを通して、革新力と意思決定力を提供します。1,300 名を超える従業員を有する株式非公開企業である Altair は、北米、南米、ヨーロッパ、アジア・パシフィックに拠点を置いています。プロダクトデザイン、先進のエンジニアリング・ソフトウェア、グリッドコンピューティング・テクノロジーにおける 25 年の実績により、Altair はさまざまな業界において一貫して高い競争力を提供しています。HyperWorks は、設計最適化、パフォーマンス・データ管理、プロセス・オートメーションを基盤として構築された、迅速な設計調査と意思決定のための企業シミュレーション・ソリューションです。詳細は、**www.altair.com** および **www.altairhyperworks.com** を参照してください。

目標

要素数が 100 万を超える乗用車の正面衝突シミュレーションを 5 分以内に処理する業界初のシミュレーション・コードを記述すること。

製品

RADIOSS は、線形および非線形シミュレーション向け有限要素解析（FEA）ソルバーで、構造、流体、流体 - 構造連成、シート・メタル・スタンピングおよび機構システムのシミュレーションが可能です。このロバストな複合領域ソリューション RADIOSS は、革新的な製品を市場により早く届けるために、設計の耐久性、優れた騒音振動性能、衝突吸収性能、安全性能および製造性を備えた製品開発を支援します。

課題

非常に多くのプロセッサーでよりスケーラビリティーの高い最適なパフォーマンスを実現するため、複数の異なる並列化手法を組み合わせた最先端のハイブリッド・プログラミングを使用すること。

効果

お客様のパフォーマンスとスケジュール要件を満たしたことにより、Altair に対する顧客満足度が向上しました。

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

Altair の製品で得られた並列化のメリット

仮想衝突試験は、自動車の開発工程のなかでも最も時間のかかるタスクの 1 つです。Altair は仮想衝突試験に必要なシミュレーションの時間を劇的に短縮できる新しい解析プロセスの可能性を証明したいと考えていました。製品のライフサイクルの点では、Altair のお客様にとって重要な競争優位となる、プロトタイプ作成のコストと市場投入にかかる時間の両方を減らす努力が行われてきました。

超並列プログラミングではドメイン分割手法と MPI 通信ライブラリーを使用することで非常に優れたスケーラビリティーが達成できますが、スケーラビリティーはプロセッサー数の増加と計算するデータ量の減少とともに低くなる傾向があります。

Altair は、非常に多くのプロセッサーでよりスケーラビリティーの高い最適なパフォーマンスを実現するため、複数の異なる並列化手法（MPI と OpenMP*）を組み合わせた最先端のハイブリッド・プログラミングを使用しました。これらの目標を達成するため、Altair では高い効率を誇るインテルのコンパイラー、ライブラリー、ツールによる最適化を行うことにしました。必要なレベルの性能を達成するためには、ほかに選択肢はなかったのです。

結果

5 分を切るという目標は達成され（1024 コア =128 MPI x 8 OpenMP* を使用して 294 秒）、全体的なパフォーマンスは約 10 倍になりました。ハイブリッド・プログラミングの結果、1024 コアまでコードのスケーラビリティーがあることが証明されました。さらに、AMS（Advance Mass Scaling）と呼ばれる新しい数値アルゴリズムにより、計算コストが下がりました。

設計の変更結果をこれまで以上にスピーディーに評価できるようになっただけでなく、設計の感度解析やロバスト性解析を日常的に実施できるようになったことで、お客様のパフォーマンスとスケジュール要件が満たされ、顧客満足度が向上したことをうれしく思います。市場にとっても、今回の結果は、以前の時間単位から分単位にシミュレーション時間を短縮し、メッシュの細密化、より優れた材料特性と破壊特性の組み合わせ、最適化、スキャッタリングといった仮想衝突試験における新しいプロセスと進歩を可能にした、飛躍的なアプローチであると考えることができます。仮想衝突試験にかかる時間が短縮されたことで、市場投入にかかる時間とコストの両方を大幅に削減できるようになったのです。

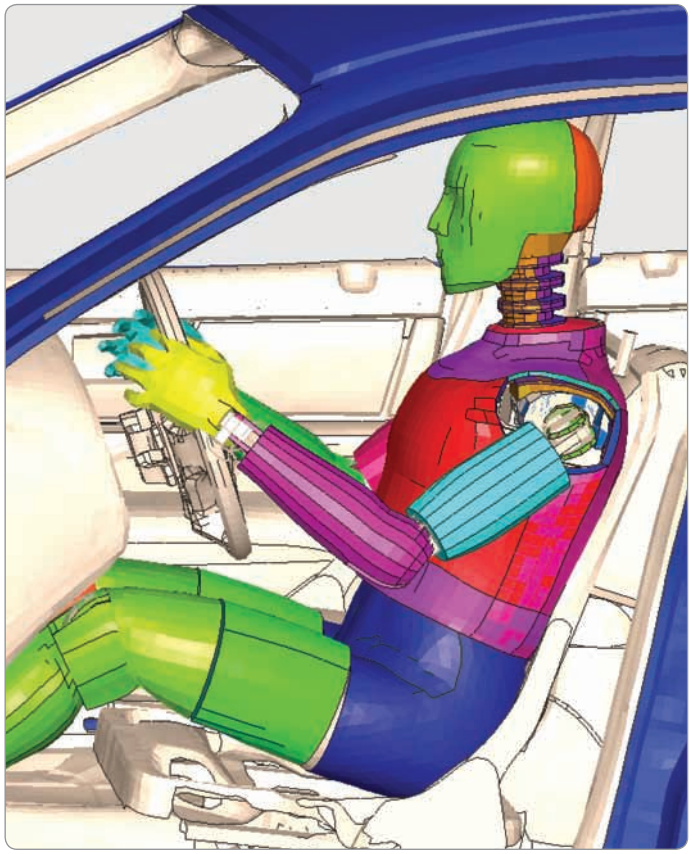
利用したインテル® ソフトウェア開発製品の機能

その速度、スケーラビリティー、パフォーマンス目標を満たすために、Altair では下記を含む一連のインテル® ソフトウェア開発製品を使用しました。

- インテル® コンパイラー
- インテル® VTune™ アナライザー（インテル® VTune Amplifier™ XE）
- インテル® クラスター・ツールキット
- インテル® トレース・アナライザー/ コレクター
- インテル® MPI ライブラリー

Altair とインテルは協力して、インテル® Xeon® プロセッサー 5500 番台ベースのクラスターの力を解き放つことに成功し、衝突シミュレーションの性能を大幅に向上させました。今回の開発は、我々の業界に長期的な利益をもたらすでしょう。

最後に、実行と最適化のための技術的な支援をはじめ、計算リソースへのアクセス、マーケティングのサポートまで、インテルの数々の協力に感謝します。



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/