Intel Software

THE PARALLEL UNIVERSE

OpenVINO™ ツールキットと FPGA

インテル® ソフトウェア・ツールにおける浮動小数点結果の再現性

C++ メモリー割り当てライブラリーの比較

Issue 34 2018

目次

編集者からのメッセージ
OpenVINO™ ツールキットを利用したエッジツークラウドのヘテロジニアス並列処理 3
Henry A. Gabb インテル コーポレーション シニア主席エンジニア

注目記事
OpenVINO™ ツールキットと FPGA 5
FPGA 向け汎用ビジュアル・コンピューティング・ツールの概要

インテル® ソフトウェア・ツールにおける浮動小数点結果の再現性 15
不確実性の払拭

C++ メモリー割り当てライブラリーの比較 25
優れた動的メモリー割り当てによりパフォーマンスを大幅に向上

LIBXSMM: インテルのハードウェアとソフトウェア開発にインスピレーションを与えるオープンソース・プロジェクト 37
特殊な密行列 / 疎行列演算とディープラーニング・プリミティブ向けのインテル® アーキテクチャーをターゲットとするライブラリー

ECHO-3DHPC による天体物理シミュレーションのパフォーマンスを向上 49
最新のインテル® ソフトウェア開発ツールを使用してハードウェアを効率良く利用

システム・パフォーマンスを理解するためのガイド 57
インテル® VTune™ Amplifier のプラットフォーム・プロファイラー

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

編集者からのメッセージ

Henry A. Gabb インテル コーポレーション シニア主席エンジニア
HPC と並列コンピューティング分野で長年の経験があり、並列プログラミングに関する記事を多数出版しています。『Developing Multithreaded Applications: A Platform Consistent Approach』の編集者 / 共著者で、インテルと Microsoft* による Universal Parallel Computing Research Centers のプログラム・マネージャーを務めました。

OpenVINO™ ツールキットを利用したエッジツークラウドのヘテロジニアス並列処理

前号では、以前は来るべきヘテロジニアス並列コンピューティング時代に不安を感じていたことを述べました。通常の並列処理でさえ十分に複雑です。同時操作を異なるプロセッサー・アーキテクチャーに分散することは、私のプログラミング能力を超えるレベルの複雑さになるでしょう。幸いにも、2006 年にカリフォルニア大学バークレー校の並列コンピューティング研究所が予測したように、複雑さが増すことでドメイン・エキスパートとチューニング・エキスパートの間の課題の分業はさらに大きくなるでしょう。(詳細は、「**並列コンピューティング研究の展望 : バークレーの見識**」(英語) を参照してください。) 例えば、私は科学ドメインで高速フーリエ変換 (FFT) を利用する方法は分かりますが、自分で FFT を記述しようとは思いません。なぜならば、すでに専門家によって記述されたライブラリーを利用することで、彼らの専門知識を活用できるからです。

このことを念頭に置いて、我々の名誉編集長である James Reinders は、今号でも FPGA プログラミング・シリーズの 1 つとして、インテルの新しい **OpenVINO™ ツールキット** (英語) を使用してヘテロジニアス並列処理を行う方法を紹介しています (OpenVINO™ は、Open Visual Inference and Neural network Optimization—オープン・ビジュアル・インファレンスとニューラル・ネットワークの最適化—の略です)。「**OpenVINO™ ツールキットと FPGA**」では、エッジデバイスからクラウドやデータセンターまで、FPGA を含む幅広いプロセッサー・アーキテクチャーにおいて、このツールキットを使用してコンピューター・ビジョンをアプリケーションに組み込む方法を紹介します。OpenVINO™ ツールキットは、コンピューター・ビジョンとハードウェア・エキスパートの専門知識をカプセル化して、アプリケーション開発者が利用できるようにします。(私は最近、**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)** について James にインタビューする機会がありました。インテル® TBB の並列抽象化によりアプリケーション開発者と並列ランタイム開発者の間の課題が明確になり、インテル® TBB のフローグラフ API によりヘテロジニアス並列処理を実現する方法について議論しました。このインタビューは、**Tech. Decoded** (英語) でご覧いただけます。)

今号のほかの記事では、ハードウェア / ソフトウェア・スタックを下層から上層へ徐々に移動していきます。「**インテル® ソフトウェア・ツールにおける浮動小数点結果の再現性**」では、バイナリー浮動小数点表現の不正確さと、インテル® コンパイラーとパフォーマンス・ライブラリーを使用してそれに対応する方法を述べます。「**C++ メモリー割り当てライブラリーの比較**」は、タイトルが示すとおりです。2 つのベンチマークでさまざまな

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

C++ メモリー割り当てライブラリーを比較して、**インテル® VTune™ Amplifier** でプロファイルの詳細を調査し、主なパフォーマンスの相違点を説明します。スタックをさらに上層へと進み、「**LIBXSMM: インテルのハードウェアとソフトウェア開発にインスピレーションを与えるオープンソース・プロジェクト**」では、ハイパフォーマンスな小行列乗算向けの研究ツールでもあり、JIT コード・ジェネレーターでもあるライブラリーを紹介します。小行列乗算は、畳み込みニューラル・ネットワークやその他多くのアルゴリズムで重要な計算カーネルです。

ハードウェア / ソフトウェア·スタックのアプリケーション·レベルの記事「**ECHO-3DHPC による天体物理シミュレーションのパフォーマンスを向上**」では、フランス原子力庁 (CEA) サクレー研究所とドイツのライプニッツ研究センター (LRZ) の協力者が、**インテル® Parallel Studio XE** を使用して重要なアプリケーションの 1 つを最適化した方法を紹介します。最後に、スタックの最上位の記事「**システム・パフォーマンスを理解するためのガイド**」では、インテル® VTune™ Amplifier の技術プレビュー機能であるプラットフォーム·プロファイラーの概要を示します。名前のとおり、プラットフォーム・プロファイラーは、プラットフォーム全体をモニタリングして、パフォーマンスに影響するシステム構成問題の診断を支援します。

今後の The Parallel Universe では、Python* による並列コンピューティング、大規模な分散データ解析に対する新しいアプローチ、インテル® ソフトウェア・ツールの新機能に関する記事をお届けします。コードの現代化、ビジュアル・コンピューティング、データセンターとクラウド・コンピューティング、データサイエンス、システムと IoT 開発向けのインテルのソリューションの詳細は、**Tech.Decoded** (英語) を参照してください。

Henry A. Gabb
2018 年 10 月

OpenVINO™ ツールキットと FPGA

FPGA 向け汎用ビジュアル・コンピューティング・ツールの概要

James Reinders The Parallel Universe 名誉編集長

この記事では、**インテル® Arria® 10 FPGA** で **OpenVINO™ ツールキット** (英語) を使用する方法を紹介します (OpenVINO™ は、Open Visual Inference and Neural network Optimization—オープン・ビジュアル・インファレンスとニューラル・ネットワークの最適化—の略です)。OpenVINO™ ツールキットは、さまざまな機能を提供します。最初に、一般的な API を使用して人間の視覚をエミュレートするアプリケーションやソリューションの開発に役立つことを示す、ハイレベルの概要を提供します。インテルは、一般的な API で CPU、GPU、**ニューラル・コンピュート・スティック** (英語) を含む**インテル® Movidius™ 製品** (英語)、および FPGA ターゲットをサポートしています。ここでは特に、OpenCL* や VHDL* の知識がなくても優れたパフォーマンスが得られる FPGA の使用法に注目します。ただし、ほかのパフォーマンスを最大化する作業と同様に、内部で何が起こっているのか理解していると役立ちます。そのため、皆さんの好奇心を満たし、設定のデバッグに必要な専門用語の理解を深めるため、必要に応じて補足します。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

まず、OpenVINO™ ツールキットと、一般的な API を使用してさまざまなプラットフォームでビジョン指向のアプリケーションをサポートする機能について簡単に説明します。そして、FPGA で OpenVINO™ ツールキットを使用するのに必要なソフトウェア・スタックを見てみましょう。これは、ドキュメントで使用されている主要な用語を定義し、必要に応じてマシンの設定をデバッグするのに役立ちます。次に、実際に CPU と CPU + FPGA で OpenVINO™ ツールキットを使用してみます。ここでは、「ヘテロジニアス」が重要な概念である理由を述べます (すべてを FPGA で実行できるわけではありません)。具体的には、ハイパフォーマンスな**インテル® プログラマブル・アクセラレーション・カード (インテル® PAC) インテル® Arria® 10 GX FPGA 搭載版**を使用します。最後に、OpenVINO™ ツールキットの内部をのぞいてみましょう。私は、車の仕組みを理解せずに車を運転したいとは思いません。OpenVINO™ ツールキットについても同様で、FPGA をターゲットとする場合の魔法のような仕組みのいくつかについて簡単に説明します。

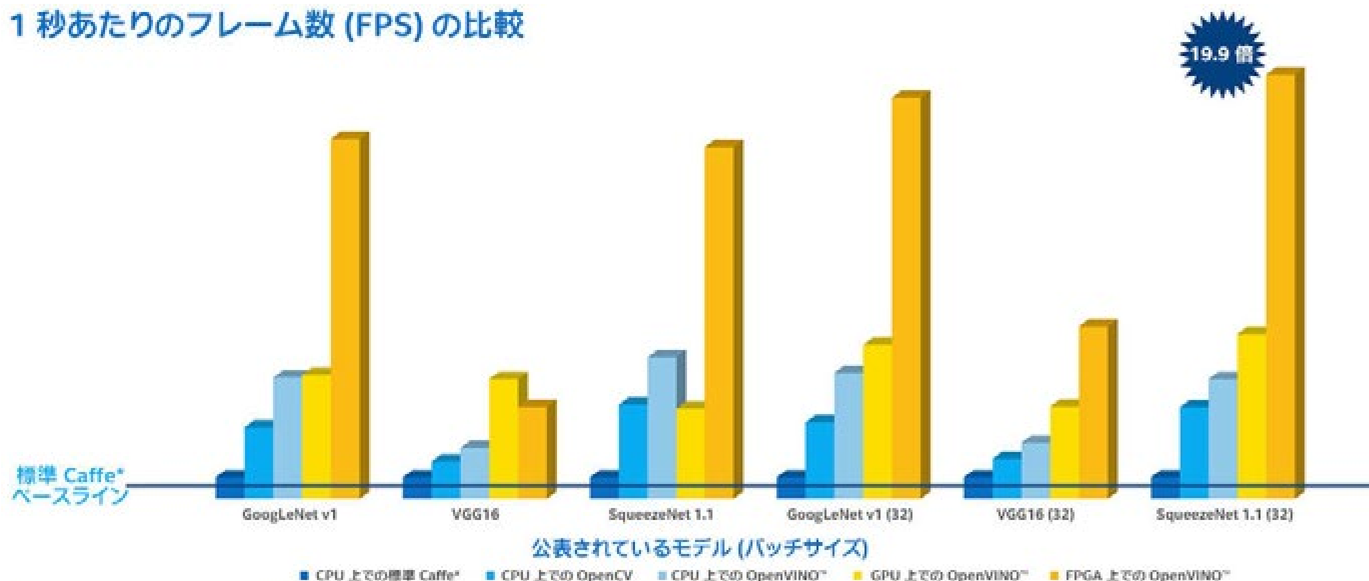
インテル® Arria® 10 GX FPGA は、150 ドル程度の FPGA 開発キットで使用されるような FPGA ではなく (私もそのような FPGA をいくつか持っています)、数千ドルする PCIe* カードです。この記事の執筆にあたり、インテルの厚意により、インテル® プログラマブル・アクセラレーション・カード (インテル® Arria® 10 GX FPGA 搭載版) を搭載した Dell EMC* PowerEdge* R740 を数週間利用することができました。その間に CPU だけでなく FPGA で OpenVINO™ ツールキットのインストールと使用法を確認することができました。

OpenVINO™ ツールキット

まず、OpenVINO™ ツールキットと、一般的な API を使用してさまざまなプラットフォームでビジョン指向のアプリケーションをサポートする機能について簡単に説明します。最近インテルは、インテル® コンピューター・ビジョン SDK の名称を OpenVINO™ ツールキットに変更しました。追加された新機能を理解すると、インテルが新名称を採用したのも当然であることが分かるでしょう。このツールキットには、3 つの新しい API が含まれています: ディープラーニング・デプロイメント・ツールキット、一般的なディープラーニング推論ツールキット、および ONNX*、TensorFlow*、MXNet*、Caffe* フレームワークをサポートする OpenCV と OpenVX* 向けに最適化された関数。

OpenVINO™ ツールキットは、ソフトウェア開発者に人間のような視覚機能を求めるアプリケーション向けの単一のツールキットを提供します。ヘテロジニアス対応のディープラーニング、コンピューター・ビジョン、ハードウェア・アクセラレーションを単一のツールキットでサポートします。OpenVINO™ ツールキットは、コンピューター・ビジョン、ニューラル・ネットワーク推論、ディープラーニングのデプロイメントに取り組んでおり、複数のハードウェア・プラットフォームでソリューションを高速化したいと考えているデータ・サイエンティストとソフトウェア開発者を対象としています。エッジからクラウドまで、アプリケーションにビジョン・インテリジェンスを取り入れることを支援します。**図 1** は、このツールキットを使用することで得られるパフォーマンス向上の可能性を示しています。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。



FP16 では精度が変わる可能性があります。上記のベンチマーク結果は、追加のテストによって変更が必要になる可能性があります。結果は、テストに使用される特定のプラットフォーム構成や作業負荷に依存します。個々のユーザーのコンポーネント、コンピューター・システム、作業負荷では同様の結果が得られない可能性があります。結果は、必ずしもほかのベンチマークを代表するものではなく、ほかのベンチマークでは、結果が異なることがあります。性能やベンチマーク結果について、さらに詳しい情報をお知りになりたい場合は、http://www.intel.com/benchmarks/ (英語) を参照してください。システム構成 : インテル® Core™ i7-6700 プロセッサー @ 2.90GHz 固定。GPU GT2 @ 1.00GHz 固定。2018 年 6 月 13 日に実施したインテル社内でのテスト。test v3 15.21、Ubuntu* 16.04、OpenVINO™ ツールキット 2018 RC4、インテル® Arria® 10 1150GX FPGA。テストは、使用モデル (公開されているもの)、バッチサイズ、その他の要因などのさまざまなパラメーターの基づきます。異なるモデルを異なるインテル® ハードウェアで高速化できますが、同じインテル® ソフトウェア・ツール・ソリューションが使用されます。ベンチマークの出典 : インテル コーポレーション

1 OpenVINO™ ツールキットの使用によるパフォーマンスの向上

インテル® ハードウェア向けに最適化されたサポートが含まれていることはもちろんですが、OpenVX* API を完全にサポートすることにより他社製ハードウェアを利用する強力な機能も提供します。ツールキットは、OpenCV と OpenVX* の両方をサポートしています。Wikipedia* では、「**OpenVX*** (英語) は、オープンソース・ビジョン・ライブラリー **OpenCV** を補完し、一部のアプリケーションでは OpenCV よりも優れた最適化グラフ管理を提供します。」と要約されています。ツールキットには、関数ライブラリー、最適化済みカーネル、OpenCV と OpenVX* 向けに最適化された呼び出しが含まれています。

OpenVINO™ ツールキットは、エッジ上での CNN ベースのディープラーニング推論専用の機能を提供します。また、CPU に加えて、GPU、インテル® Movidius™ 製品、FPGA を含むコンピューター・ビジョン・アクセラレーター全体にわたってヘテロジニアス実行をサポートする一般的な API も用意されています。

ビジョンシステムは、世界を変えるような問題の解決に役立つことが期待されます。OpenVINO™ ツールキットは、ハイパフォーマンスなコンピューター・ビジョンとディープラーニング推論の開発に役立ち、**無料でダウンロード** (英語) できます。

FPGA ソフトウェア・スタック — FPGA から OpenVINO™ ツールキットまで

FPGA で OpenVINO™ ツールキットを使用する前に、必要なソフトウェアと設定を見てみましょう。ここでは、基本的な用語を定義し、インストールが必要なものを説明します。内部処理についてはあまり細かく掘り下げずスタック内部の動作については、この記事の最後のセクションで詳しく説明します。

幸いにも、OpenVINO™ ツールキットで FPGA を利用するのに必要なほとんどのものは、インテル® Acceleration Stack (**インテル® FPGA Acceleration Hub** (英語) からダウンロード可能) のランタイムバージョン (619MB) をインストールするだけで準備できます。ランタイムバージョンが含まれる大きなサイズの開発バージョン (16.9GB) を利用することもできます。どちらのバージョンをインストールしてもかまいません。これは、Java* のランタイムをインストールするか、完全な Java* Development Kit をインストールするかを選択するのに似ています。Acceleration Stack for Runtime には、以下が含まれます。

- **FPGA プログラマー** (インテル® Quartus® Prime Pro Edition ソフトウェア — プログラマーのみ)
- **OpenCL* ランタイム** (OpenCL* 向けインテル® FPGA ランタイム環境)
- **インテル® FPGA Acceleration Stack** (**Open Programmable Acceleration Engine (OPAE)** (英語) を含む)。OPAE は、プログラマブル・アクセラレーターの管理とアクセスを行う、オープンソースのソフトウェア・フレームワークです。

経験上、FPGA 環境を設定する際に忘れがちなものとして、FPGA 用のファームウェアと OpenCL* Board Support Package (BSP) があります。FPGA の環境設定は、私にとって全く新しい体験であり、FPGA ユーザーフォーラムから多くの開発者が同様の疑問を持っていることが分かりました。ここで紹介する「最新のアクセラレーション・スタック、最新のファームウェア、最新の OpenCL* と BSP」のサマリーは、皆さんのシステムで確認すべきことのチェックリストとして役立てることができるでしょう。

FPGA ボードのファームウェア : 最新バージョンを使用する

ファームウェアに関する一般的なアドバイスは、最新バージョンを入手してインストールすることです。BIOS アップデートと PCIe* カードのファームウェアについても同じことが言えます。(ここで使用するインテル® プログラマブル・アクセラレーション・カード (インテル® PAC) インテル® Arria® 10 GX FPGA 搭載版のように) ファームウェアはボードメーカーから提供されます。インテルの場合、ファームウェア・バージョンとアクセラレーション・スタック・バージョンの対応表が用意されています。最新のアクセラレーション・スタックにアップデートするには、最新のファームウェアが必要です。最新のファームウェア・バージョンは、`sudo fpgainfo fme` コマンドで確認できます。

OpenCL* BSP: 最新バージョンを使用する

OpenCL* を滅多に使用しない場合は、BSP のバージョンについて心配しなくても良いでしょう。OpenCL* が誕生する前、BSP は元々組込み分野でボードとリアルタイム・オペレーティング・システムの接続に使用されていました。現在の FPGA では、BSP はシステムに搭載された FPGA と OpenCL* の接続に使用されます。OpenCL* サポートはプラットフォームとともに進化するため、特定の FPGA カード向けの最新バージョンの BSP が不可欠です。インテルでは、アクセラレーション・スタックとともに BSP を配布しているため、最新のソフトウェアをインストールしていれば、BSP と OpenCL* の互換性が維持されます。ここでは、手順に従って使用したボード用の BSP を選択し、**`aocl install`** コマンドで OpenCL* と BSP をインストールしました (**`aocl`** は Altera* OpenCL* の略です)。

FPGA が使用可能か確認するには？

`aocl list-devices` を実行して、適切な応答が得られれば FPGA を使用できます。そうでない場合は、FPGA が認識され、動作するようにしなければなりません。次の 3 項目をチェックします。

1. 最新のアクセラレーション・スタック・ソフトウェアを**インストール**します。
2. ファームウェアが最新であることを**確認**します。
3. OpenCL* と適切な BSP がインストールされていることを**確認**します。

私は項目 2 と 3 で問題の解決に多くの時間を費やしたため、次の出力が表示されたときは思わず笑みがこぼれました。

```
--------------------------------------------------------------------
Device Name:
acl0

Package Pat:
/home/james/tools/intelrtestack/a10_gx_pac_ias_1_1_pv/opencl/opencl_bsp

Vendor: Intel Corp

Physical Dev Name    Status           Information

pac_a10_eb00000      Passed           PAC Arria 10 Platform
(pac_a10_eb00000)
                                      PCIe 134:00.0
                                      FPGA temperature = 57 degrees C.

DIAGNOSTIC_PASSED
--------------------------------------------------------------------
```

図 2 は、この記事の執筆に使用した PAC です。

2 **インテル® プログラマブル・アクセラレーション・カード (インテル® Arria® 10 GX FPGA 搭載版)**

CPU + FPGA 向け OpenVINO™ ツールキット

FPGA アクセラレーション・スタックをインストールし、ボードのファームウェアをアップデートして、OpenCL* と適切な BSP が有効な状態であることを確認したら、OpenVINO™ ツールキットをインストールします。ビルド済みのツールキットを入手するため、**OpenVINO™ ツールキットの Web サイト** (英語) にアクセスして、「Linux* for FPGA v2018R3」をダウンロードします (登録が必要)。オフライン・ダウンロード・パッケージのサイズは 2.3GB でした。インストールは簡単で、コマンドライン・インストーラーと GUI インストーラー (`setup_GUI.sh`) の両方を試しましたが、GUI インストーラーはポップアップ・ウィンドウの表示に X11 を使用しており、コマンドラインよりも使いやすいと思われます。

最初に、CPU 向けに OpenVINO™ ツールキットを使用してから、インテル® プログラマブル・アクセラレーション・カード (インテル® Arria® 10 GX FPGA 搭載版) でパフォーマンス向上を検証します。

SqueezeNet

OpenVINO™ ツールキットには、使用法を示すため SqueezeNet を含むいくつかのデモが含まれています。SqueezeNet は、1/50 のパラメーター数で ImageNet* について AlexNet レベルの精度を達成する小さな CNN アーキテクチャーです。作者は**論文** (英語) で次のように述べています:「ディープラーニングの大半がパラメーターのチューニングであることは周知の事実です。パラメーター数を大幅に減らすには、畳み込みニューラル・ネットワークの設計に関する研究を増やす必要があります。」インテルのデモは、Caffe* SqueezeNet モデルを使用して、OpenVINO™ ツールキットと一般的なプラットフォームの接続方法を示します。

CPU で SqueezeNet を実行するには、次のコマンドを使用します。

```
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/demo
./demo_squeezenet_download_convert_run.sh
```

FPGA で SqueezeNet を実行するには、次のコマンドを使用します。

```
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/demo
./demo_squeezenet_download_convert_run.sh -d HETERO:FPGA,CPU
```

コマンドでは、「FPGA」ではなく **HETERO:FPGA,CPU** を使用します。これは、厳密には FPGA はプログラム全体ではなく、ニューラル・ネットワークのコア (推論) を実行するためです。次のように「FPGA」と指定すると、

```
./demo_squeezenet_download_convert_run.sh -d FPGA
```

推論エンジンは、次のようなエラーメッセージを出力します。

```
Graph is not supported on FPGA plugin due to existence of layer (Name:prob,
Type: SoftMax) in topology. Most likely you need to use heterogeneous plugin
instead of FPGA plugin directly.
```

この単純なデモは、簡潔すぎて実行時間のほとんどが FPGA の設定オーバーヘッドに費やされるため、FPGA で実行すると低速です。この問題を解決するため、次のコマンドを実行しました。

```
export myDIR=/opt/intel/computer_vision_sdk_fpga_2018.3.343

cd $myDIR/deployment_tools/demo/

aocl program acl0 $myDIR/a10_dcp_bitstreams/2-0-1_RC_FP11_SqueezeNet.aocx

alias
csa='~/inference_engine_samples/intel64/Release/classification_sample_async'

export myPIC=$IE_INSTALL/demo/car.png

csa -m squeezenet1.1.xml -i $myPIC -d HETERO:FPGA,CPU -ni 100 -nireq 3

csa -m squeezenet1.1.xml -i $myPIC -ni 100 -nireq 3
```

これらのコマンドを使用することで、スクリプト内の冗長なコマンドを回避できます。手動で反復回数 (**`-ni`** パラメーター) を増やして 1 回の実行の FPGA 設定コストを相殺する現実的なワークロードで、データセンターの FPGA 搭載システムに適した長時間または連続的な推論をシミュレーションできます。

私のシステムでは、CPU は 368fps を達成し、CPU + FPGA ではそれを上回る 850fps を達成しました。素晴らしい結果と言えるでしょう。より大きな推論ワークロードでは、FPGA は CPU をさらに大きく上回る可能性があります。テストに使用した CPU は、デュアルソケットの**インテル® Xeon® Silver プロセッサー** (ソケットごとに 8 コア、ハイパースレッディング有効) ですが、このような CPU よりも優れた結果が得られたことは興味深いです。

FPGA で実行されるのはビットストリーム

一般に「プログラム」と呼ばれるものは、通常 FPGA では「ビットストリーム」と呼ばれます。そのため、FPGA 開発者の間では「どのビットストリームを実行していますか ?」といった会話が行われます。**`demo_squeezenet_download_convert_run.sh`** スクリプトからは、ビットストリームの作成とロードを知ることができません。ビットストリームのコンパイルには時間がかかりますが、ロードは高速です。コンパイルとロードは毎回行う必要はなく、ビットストリームは、一度 FPGA にロードすれば以降の実行で利用できます。ここで使用した **`aocl program acl0…`** コマンドは、サポートされているニューラル・ネットワーク向けにインテルによって提供されたビットストリームをロードします。実際には、ビットストリームを再ロードする必要はありませんが、次の実行の前に FPGA で別のプログラムを実行した後でもコマンドが確実に動作するように追加しました。

これですべてでしょうか ?

私が FPGA で OpenVINO™ ツールキットを使用するのを好む理由は、「次にすることはありませんか ?」と簡単に言えることです。ここでカバーした内容を振り返ってみましょう。

- **コンピューター・ビジョン・アプリケーション**を (Caffe* のような) 一般的なプラットフォームで訓練できる場合、OpenVINO™ ツールキットを利用して訓練済みネットワークをさまざまなシステムにデプロイできます。
- **FPGA で実行する**とは、適切なアクセラレーション・スタックのインストール、ボードのファームウェアのアップデート、OpenCL* と適切な BSP のインストールを行った後、OpenVINO™ ツールキットの推論エンジンのステップに従って、ニューラル・ネットワーク向けに適切な FPGA ビットストリームを生成することを意味します。
- **後は実行するだけです。**

ここでは、OpenCL* や VHDL プログラミングについては説明しません。(OpenCL* プログラミングについては、**The Parallel Universe 31 号**の記事をご覧ください。)

コンピューター・ビジョンに関しては、OpenVINO™ ツールキットとその推論エンジンを利用することで、コーディングは FPGA エキスパートに任せて、私たちはモデルの作成に集中することができます。

OpenVINO™ ツールキットの FPGA サポートの仕組み

FPGA 向け OpenVINO™ ツールキットが優れている理由は、2 つの非常に異なる内部処理にあります。

- さまざまなデバイスと FPGA をサポートする**抽象化**
- **非常に優れた** FPGA サポート

上記の抽象化は、インテルのモデル・オプティマイザーとインテルの推論エンジンによるその使用法に関してです。モデル・オプティマイザーは、次の機能を提供するクロスプラットフォームのコマンドライン・ツールです。

- 訓練環境からデプロイメント環境への移行を**高速化**します。
- スタティック・モデル解析を**実行**します。
- エンドポイントのターゲットデバイスで最適に実行できるようにディープラーニング・モデルを**調整**します。

図 3 は、モデル・オプティマイザーの動作フローです。サポートされるフレームワークで訓練されたネットワーク・モデルから開始して、その後は訓練済みディープラーニング・モデルをデプロイする通常のワークフローと同じです。

3 **モデル・オプティマイザーの動作フロー**

SqueezeNet サンプルの推論エンジンは、単純にコマンドに応じてワークを CPU または FPGA へ送ります。推論エンジンは、モデル・オプティマイザーによって生成される中間表現 (IR) を使って、CPU、GPU、インテル® Movidius™ 製品、FPGA を含むさまざまなデバイスで処理できます。インテルではまた、IR を使用してネットワークを処理するように構成する、FPGA 向けに最適化されたビットストリームを生成するためのコーディング作業も行っています。これは 2 番目の内部処理に関連しています。

非常に優れた FPGA サポート機能は、FPGA エキスパートによって記述され、注意深くチューニングされたコードのコレクションによって実現されています。このコレクションは、FPGA 向けディープラーニング・アクセラレーター (DLA) と呼ばれ、OpenVINO™ ツールキットの FPGA の高速化に重要な役割を果たしています。FPGA エキスパートの知識と経験を集結した DLA を使用することで、カスタム・ハードウェア設計のようなソフトウェアのプログラマビリティーが得られます。(DLA について詳しく知りたい方は、DLA チームの論文「**DLA: ニューラル・ネットワークの推論を高速化するためのコンパイラーと FPGA のオーバーレイ**」(英語) をご覧になることを推奨します。彼らは、自分たちの取り組みを「ドメイン固有のカスタマイズ可能なオーバーレイ・アーキテクチャーを実装することで、ソフトウェアの使いやすさとハードウェアの効率を実現する方法」と説明しています。)

まとめと関連情報

インテル® Arria® 10 GX FPGA 搭載システムを利用する機会を提供してくれたインテルの皆さんに感謝します。実際に評価することで、ヘテロジニアス並列処理と FPGA ベースの高速化を容易に利用できることが分かりました。私はスピードを重視するプログラマーですが、FPGA プログラミングの知識がなくても FPGA を利用することができ、納得のいくスピードが得られました。

この記事が皆さんにとって興味深く、役に立つものであることを願っています。今後も、FPGA の機能がソフトウェアでサポートされたら皆さんと共有していきたいと思います。

以下のリンクは、FPGA の可能性について学んだり、調査するのに役立ちます。

- **OpenVINO™ ツールキットの製品 Web サイト** (英語) (ソース /GitHub* サイトは**こちら** (英語))
- **OpenVINO™ ツールキットの推論エンジン・デベロッパー・ガイド** (英語)
- インテル® Acceleration Stack は**インテル® FPGA Acceleration Hub** (英語) からダウンロード可能
- ディープラーニング・モデルを表現するオープン・フォーマットの **ONNX*** (英語)
- **ベータ版 インテル® ディープラーニング・デプロイメント・ツールキット** (英語)
- 「OpenCL* による FPGA プログラミング」James Reinders、Tom Hill 著、『**The Parallel Universe 31 号**』
- **OpenCL* 標準規格の公式情報** (英語)
- インテル® FPGA 製品情報 : **インテル® Cyclone® 10 LP FPGA**、**インテル® Arria® 10 FPGA**、および**インテル® Stratix® 10 FPGA**

OpenVINO™ ツールキット
マルチプラットフォーム向けコンピューター・ビジョン・ソリューションの開発
無料
ダウンロード

インテル® ソフトウェア・ツールにおける浮動小数点結果の再現性

不確実性の払拭

Martyn Corden、Xiaoping Duan、および Barbara Perz
インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア

ほとんどの実数のバイナリー浮動小数点 (FP) 表現は不正確で、浮動小数点数を含むその演算結果には特有の不確実性があります。そのため、異なる条件下で計算を行うと、結果は予想される不確実さの範囲内で一貫していますが、異なる可能性もあります。通常は問題になりませんが、一部のコンテキストではより高い再現性が求められます (例 : 品質保証、法的問題、機能安全要件など)。しかし、完全な再現性または高い再現性を達成するには、通常、パフォーマンスが犠牲になります。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

再現性とは？

再現性の定義は人それぞれです。最も基本的なことは、同じプロセッサー上で、同じデータを使用して、同じ実行ファイルを繰り返し実行した場合、常に同一の結果が生成されることです。これは、繰り返し性または実行再現性と呼ばれます。結果が必ずしも決定的ではなく、再現性が自動的に提供されないことに驚いたり、ショックを受けるユーザーもいます。

再現性はまた、異なるプロセッサー・タイプをターゲットとし実行した場合、異なる最適化レベルでビルドした場合、あるいは異なる並列処理タイプやレベルで実行した場合に、同一の結果が生成されることを意味します。これは、条件付き数値再現性と呼ばれます。完全に再現性のある結果を得るために必要な条件はコンテキストに依存し、パフォーマンスが低下する可能性があります。

多くのソフトウェア・ツールは、デフォルトでは完全に再現性のある結果をもたらしません。

差異の原因

浮動小数点結果が異なる主な原因は、最適化です。最適化には、次のものが含まれます。

- ビルド時または実行時に特定のプロセッサーと命令セットを**ターゲット**にする
- **さまざまな**形式の並列処理

現代のプロセッサーでは、パフォーマンスの利点が非常に大きいため、大規模なアプリケーションでも最適化を行わないことはほとんどありません。精度が異なる原因として、次のようなものがあります。

- 数学関数や除算などの操作の**近似値が異なる**
- 中間結果の計算と格納に使用される**精度**
- **正規化されていない**非常に小さな値がゼロとして扱われる
- FMA (Fused Multiply Add) 命令などの**特殊命令の使用**

FMA のような特殊命令は、通常、乗算と加算を個別に行うよりも高い精度が得られますが、最終結果は変わる可能性があります。

FMA の生成は、インテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) 以上の命令セットをターゲットとする場合に O1 以上で有効になる最適化です。言語標準ではカバーされていないため、コンパイラーは異なるコンテキストでは異なる最適化を適用する可能性があります (例 : FMA をサポートする異なるプロセッサーに対して異なる最適化を適用することがあります)。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

おそらく、差異の最も重要な原因は、特に並列アプリケーションの場合、操作の順序でしょう。異なる順序は数学的には等価かもしれませんが、有限精度演算では、丸め誤差により差異が生じたり、合計結果が異なることがあります。異なる結果が必ずしも精度が低いわけではありませんが、ユーザーは最適化されていない結果が正しいと見なすことがあります。

例えば、**図 1** に示すような、コンパイラーがパフォーマンス向上のために行う変換があります。

```
(x[i] + y) + z  →  x[i] + (y + z);

 a*b + a*c →  a*(b+c)
```

1 コンパイラーによる変換の例

これまでに説明した最適化は、シーケンシャル・アプリケーションと並列アプリケーションに同様の影響を与えます。コンパイルされるコードでは、コンパイラー・オプションにより最適化を制御または抑止できます。

リダクション

リダクションは、結果が浮動小数点演算の順序に依存することを示す特に重要な例です。ここでは合計を例に説明しますが、ここで示す内容は、積、最大値、最小値などのほかのリダクション操作にも適用されます。合計操作の並列実装は、スレッドごと (OpenMP* など)、プロセスごと (MPI など)、SIMD レーンごと (ベクトル化) の部分和に分割します。これらの部分和はすべて安全に並列にインクリメントできます。**図 2** に例を示します。

```
//  original sequential
version

float Sum(const float A[],
int n)
{
    float sum=0;
    for (int i=0; i<n; i++)
        sum = sum + A[i];
    return sum;
}
```

```
float Sum( const float A[], int n, int nt ) {
  float sum=0.;                       // total sum
  float part_sum[nt]                  // 1 partial sum per thread
  for (int it=0; it<nt; it++) part_sum[it]=0.;
  for (int it=0; it<nt; it++)  {  // loop over threads or ranks
      for (int j=it*n/nt; j<(it+1)*n/nt; j++)
          part_sum[it] += A[j];   //  partial sum within a thread
  }
  for (int it=0; it<nt; it++) sum += part_sum[it];
  return sum;
}                                    // parallel version
```

2 リダクションを使用した並列合計処理

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

2 つのケースでは、A の要素が加算される順序が大きく異なり、中間結果がマシン精度に丸められます。正の項と負の項の間に多くの相殺がある場合、最終結果への影響は驚くほど大きくなります。ユーザーは最初のシリアルバージョンが「正しい」結果と見なしがちですが、複数の部分和を使用する並列バージョンは、特に要素数が多い場合、丸め誤差の累積を減らし、有限精度の結果に近い結果を生成する傾向にあります。並列バージョンは演算も高速になります。

リダクションで再現性のある結果が得られるのか？

リダクションで再現性のある結果を得るためには、部分和の構成が常に同じでなければなりません。ベクトル化では、ベクトル長が常に同じでなければなりません。OpenMP* では、スレッド数が一定でなければなりません。**インテル® MPI ライブラリー**では、ランク数が常に同じでなければなりません。また、部分和を同じ一定の順序で合計する必要があります。ベクトル化ではこの処理が自動的に行われます。

OpenMP* によるスレッド化では、任意の順序で部分和を結合できます。インテルの実装では、スレッド数が小さい場合 (**インテル® Xeon® プロセッサー**では 4 未満、**インテル® Xeon Phi™ プロセッサー**では 8 未満)、デフォルトは先着順です。部分和が一定の順序で合計されることを確実にするには、環境変数 **`KMP_DETERMINISTIC_REDUCTION=true`** を設定して、静的スケジュール (デフォルトのスケジュール・プロトコル) を使用します。

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、動的スケジュールを使用するため、**`parallel_reduce()`** メソッドは実行再現性のある結果を生成しません。代わりに、**`parallel_deterministic_reduce()`** メソッドがサポートされています。このメソッドは、部分和を計算する固定タスクと、それを結合するための固定の順序付けされたツリーを作成します。動的スケジューラーは、依存関係を維持したまま、動的にタスクをスケジュールできます。これにより、同じ環境で実行再現性のある結果を生成できるだけでなく、ワーカースレッドの数が変わっても確実に再現性のある結果が得られます。(OpenMP* 標準では、固定ツリーに基づく類似のリダクション操作を提供していませんが、OpenMP* タスクと依存性を利用して記述できます。)

インテル® MPI ライブラリーでは、プロセッサー・ノード間の MPI ランクの分散状況に応じて、部分和の結合順序を最適化できます。再現性のある結果を得る唯一の方法は、トポロジーを意識しないリダクション・アルゴリズムの制限されたセットから選択することです (以下を参照)。

我々が検証したすべての例では、並列またはベクトル化されたリダクションの結果は、通常、シーケンシャル・リダクションの結果とは異なりました。これが許容できない場合は、単一のスレッド、プロセス、または SIMD レーンでリダクションを実行する必要があります。単一の SIMD レーンで実行するには、リダクション・ループが自動でベクトル化されないように **`/fp:precise`** (Windows*) または **`-fp-model precise`** (Linux* および macOS*) を指定してコンパイルします。

インテル® コンパイラー

異なる実行、異なる最適化レベル、および同じアーキテクチャーの異なるプロセッサー・タイプで最高の再現性を実現するには、ハイレベルのオプション **/fp:consistent** (Windows*) または **-fp-model-consistent** (Linux* および macOS*) を推奨します。これは、FMA 生成を無効にする **/Qfma-** (**-no-fma**)、すべてのプロセッサー・タイプで同じ結果を生成する実装に数学関数を制限する **/Qimf-arch-consistency:true** (**-fimf-arch-consistency=true**)、および結果を変える可能性のあるその他のコンパイラーによる最適化を無効にする **/fp:precise** (**-fp-model-precise**) を指定した場合と同じです。

この再現性は、パフォーマンスを低下させます。どの程度低下するかはアプリケーションに依存しますが、一般に約 10% 低下します。通常、浮動小数点リダクションや超越数学関数呼び出しを含む、多くのベクトル化可能なループを持つ計算集約型アプリケーションが最も大きな影響を受けます。ベクトル呼び出しだけでなく、数学関数のスカラー呼び出しにも SVML (Short Vector Mathematical Library) ライブラリー関数を使用し、数学関数を含むループの自動ベクトル化の一貫性と再有効化を保証する **/Qimf-use-svml** (**-fimf-use-svml**) オプションを追加することでこの影響を軽減できることがあります。

デフォルトの **/fp:fast** (**-fp-model fast**) は、コンパイラーに再現性を考慮しない最適化を許可します。同じプロセッサー上で、同じデータを使用して、同じ実行ファイルを繰り返し実行すると同じ結果が生成される、繰り返し性のみが必要な場合は、**/Qopt-dynamic-align-** (**-qno-opt-dynamic-align**) を指定して再コンパイルするだけで済む可能性があります。このオプションは、実行時にデータ・アライメントをテストするピールループの生成のみを無効にするため、前述の **/fp** (**-fp-model**) オプションと比較するとパフォーマンスへの影響はわずかです。

異なるコンパイラーとオペレーティング・システムでの再現性

ほとんどの数学関数の結果には一般的に許容されている要件がないため、異なるコンパイラーとオペレーティング・システムによって再現性が制限されます。数学関数の最終的な基準 (例えば、正確な丸めを必要とするなど) を遵守すれば一貫性は向上しますが、パフォーマンスが大幅に低下します。

現在、Windows* や Linux* などの異なるオペレーティング・システムをターゲットとするコードの結果の再現性を体系的にテストする手段はありません。**/Qimf-use-svml** と **-fimf-use-svml** オプションは、数学関数を含むループのベクトル化に関連する既知の差異の原因に対応し、Windows* と Linux* の両方で浮動小数点結果の一貫性を向上するために推奨されます。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

インテル® コンパイラーの異なるメジャーバージョンでビルドされたアプリケーションの間で一貫性を保証する手段はありません。改善された数学ライブラリー関数の実装では、結果の精度は向上するかもしれませんが、以前の実装とは異なる結果になる可能性があります。**/Qimf-precision:high** (**-fimf-precision=high**) オプションは、このような差分を軽減します。同様に、インテル® コンパイラーでビルドされたアプリケーションと他社製コンパイラーでビルドされたアプリケーションの間で再現性を保証することはできません。**/fp:consistent** (**-fp-model consistent**) などのオプションや他社製コンパイラーの同等のオプションは、コンパイル済みコードの結果の差異を軽減するのに役立ちます。可能な場合は、両方のコンパイラーで同じ数学ランタイム・ライブラリーを使用すると良いでしょう。

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® マス・カーネル・ライブラリー (インテル® MKL) は、OpenMP* またはインテル® スレッディング・ビルディング・ブロック (インテル® TBB) により内部でベクトル化またはスレッド化されている、線形代数、高速フーリエ変換、スパースソルバー、統計解析、およびその他のドメイン向けの高度に最適化された関数を提供します。デフォルトでは、同じプロセッサーで繰り返し実行した場合、最適化された関数内の操作の順序が変わるため同一の結果を得ることができません。インテル® MKL 関数は、実行時にプロセッサーを検出して、そのプロセッサー向けに最適化されたコードパスを実行するため、異なるプロセッサーで実行すると結果が異なる可能性があります。この問題を回避するため、インテル® MKL は条件付き数値再現性を実装しています。次の条件を満たす必要があります。

- インテル® TBB ではなく OpenMP* ベースのインテル® MKL バージョンを使用します。
- スレッド数を一定にします。
- 静的スケジューリングを使用します (デフォルトの **OMP_SCHEDULE=static**)。
- アクティブなスレッド数の動的調整を無効にします (デフォルトの **OMP_DYNAMIC=false** と **MKL_DYNAMIC=false**)。
- 同じオペレーティング・システムとアーキテクチャーを使用します (例 : インテル® 64 アーキテクチャー・ベースの Linux*)。
- 同じマイクロアーキテクチャーを使用するか、最小マイクロアーキテクチャーを指定します。

最小マイクロアーキテクチャーは、関数またはサブルーチンを呼び出すか (例 : **mkl_cbwr_set** (**MKL_CBWR_AVX**))、ランタイム環境変数を設定して (例 : **MKL_CBWR_BRANCH=MKL_CBWR_AVX**) 指定します。これにより、インテル® AVX2 やインテル® AVX-512 などのインテル® AVX 以降の命令セットをサポートするすべてのインテル® プロセッサーで一貫した結果が得られます。ただし、より高度な命令セットをサポートするプロセッサーではパフォーマンスが低下します。引数 **MKL_CBWR_COMPATIBLE** は、同じアーキテクチャーのインテル® プロセッサーおよび互換プロセッサーで一貫した結果をもたらします。引数 **MKL_CBWR_AUTO** は、実行時に検出されたプロセッサーに対応するコードパスが実行されるようにします。そして、そのプロセッサーで繰り返し実行した場合に同じ結果が生成されることを保証します。ただし、ほかのプロセッサー・タイプでは結果が異なります。ランタイム・プロセッサーが指定された最小マイクロアーキテクチャーをサポートしていない場合、実行ファイルは実行できますが、**MKL_CBWR_AUTO** を指定した場合と同様に、実際のランタイム・マイクロアーキテクチャーに対応するコードパスが実行されます。結果は、警告なしに、ほかのプロセッサーの結果と異なる可能性があります。

計算集約型のインテル® MKL 関数では、命令セットの制限によるパフォーマンスへの影響は顕著に現れます。**表 1** は、**インテル® Xeon® スケーラブル・プロセッサー**上で、異なる最小マイクロアーキテクチャーを選択した場合の DGEMM 行列 - 行列乗算の相対パフォーマンスを示しています。

表 1. インテル® Xeon® スケーラブル・プロセッサーで DGEMM を実行した場合の命令セット・アーキテクチャー (ISA) への影響

ターゲット ISA	予測される相対パフォーマンス
MKL_CBWR_AUTO	1.0
MKL_CBWR_AVX512	1.0
MKL_CBWR_AVX2	0.50
MKL_CBWR_AVX	0.27
MKL_CBWR_COMPATIBLE	0.12

パフォーマンス結果は 2018 年 9 月 6 日現在のインテル社内での測定値であり、すべての公開済みセキュリティー・アップデートが適用されていない可能性があります。詳細は、システム構成を参照してください。絶対的なセキュリティーを提供できる製品はありません。性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。システム構成 : インテル® Xeon® スケーラブル·プロセッサーで DGEMM 実行を実行した場合の命令セット·アーキテクチャー (ISA)。詳細については、**www.intel.com/benchmarks** (英語) を参照してください。

インテル® MPI ライブラリー

次の条件を満たす場合、インテル® MPI ライブラリーを使用する結果は再現性があります。

- **コンパイル済みコードとライブラリー呼び出し**がコンパイラーとライブラリーの再現性条件を満たす場合
- **MPI とクラスター環境に変更がない場合** (ランク数とプロセッサー・タイプを含む)

一般に、総和やリダクションなどの集合操作は、環境の小さな変化に対して最も敏感です。集合操作の多くの実装は、クラスターノード間の MPI ランクの分散状況に応じて最適化されており、操作の順序が変更されたり、結果に差異が生じることがあります。インテル® MPI ライブラリーは、条件付き数値再現性をサポートしており、ノード間のランクの分散状況が変更されても、アプリケーションは同じバイナリーで再現性のある結果を得られます。それには、**I_MPI_ADJUST_** 環境変数を使用して、トポロジーを意識しない (つまり、ノード間のランクの分散状況に応じて最適化しない) アルゴリズムを選択する必要があります。

- **I_MPI_ADJUST_ALLREDUCE**
- **I_MPI_ADJUST_REDUCE**
- **I_MPI_ADJUST_REDUCE_SCATTER**
- **I_MPI_ADJUST_SCAN**
- **I_MPI_ADJUST_EXSCAN**
- ほか

例えば、インテル® MPI ライブラリー・デベロッパー・リファレンスには、11 種類の **MPI_REDUCE()** の実装があります。そのうちの最初の 7 つを**表 2** に示します。

表 2. 異なるランクの分散状況での `MPI_REDUCE()` の結果の比較

	ランク分散 アルゴリズム	0246 node1 1357 node2	すべての node1	0145 node1 2367 node2	0123 node1 4567 node2
1	Shumilin				
2	二項				
3	トポロジーを意識した Shumilin				
4	トポロジーを意識した二項				
5	Rabenseifner				
6	トポロジーを意識した Rabenseifner				
7	Knomial				

インテル® Core™ i5-4670T プロセッサー @ 2.30GHz、4 コア、8GB メモリーベースの 2 つのノード。1 つは Red Hat* EL 6.5、もう一方は Ubuntu* 16.04 を実行。The Parallel Universe 21 号の「インテル® MPI ライブラリーの条件付き再現性」のサンプルコードを使用 (記事の最後にある「 参考文献」を参照)。

表 2 は、選択した **`MPI_REDUCE()`** 実装のサンプルプログラムを、8 つの MPI ランクを 2 つのクラスターノードに 4 つの異なる方法で分散して実行した結果の比較です。測定された 5 つの異なる結果に応じて色分けしています。結果の差は非常に小さく、精度の限界に近いものですが、大規模な計算では小さな差分が取り消しによって増幅されることがあります。トポロジーに依存しない実装は、ノード間のランクの分散状況に関係なく同じ結果を生成しますが、トポロジーを意識した実装では結果が異なります。**`MPI_REDUCE`** のデフォルトの実装 (ここには記載されていません) は、ワークロードとトポロジーに依存するアルゴリズムで構成されており、ノード間のランクの分散状況に応じて結果が異なります。

結論

インテル® ソフトウェア開発ツールは、明確に定義された条件下で再現性のある浮動小数点結果を取得する方法を提供します。

参考文献

1. 「**インテル® コンパイラーの浮動小数点演算における結果の一貫性**」
2. 『**インテル® MKL 2019 for Linux* デベロッパー・ガイド**』(英語) の「Obtaining Numerically Reproducible Results (数値再現性のある結果を得る)」セクション
3. The Parallel Universe 21 号の「**インテル® MPI ライブラリーの条件付き再現性**」
4. 『**インテル® MPI ライブラリーのチューニング : 基本的な手法**』(英語) の「Tuning for Numerical Stability (数値安定性のためのチューニング)」セクション
5. 『**インテル® C++ コンパイラー・デベロッパー・ガイドおよびリファレンス**』(英語) と『**インテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス**』(英語) の「Floating-Point Operations (浮動小数点演算)」セクション

BLOG HIGHLIGHTS

ドメイン・エキスパートとチューニング・エキスパートの間のギャップを解消

Henry A. Gabb インテル コーポレーション シニア主席エンジニア

2006 年にカリフォルニア大学バークレー校の並列コンピューティング研究所は、並列処理が広く普及することでドメイン・エキスパートとチューニング・エキスパートの間の課題の分業はさらに大きくなると予測しました (図 1)。一方には問題を解こうとするあらゆる分野のユーザーがおり、彼らのコンピューター・バックグラウンドはさまざまです。彼らは問題を解くために必要なコーディングを最小限に抑え、ビジネス上の意思決定、研究論文、工学的設計などのより大きなタスクに集中したいと考えています。コードのチューニングは、パフォーマンスのボトルネックが目標達成を妨げる場合にのみ考慮されます。もう一方には、その対極と言えるチューニング・エキスパート (内部では通称忍者プログラマーと呼ばれている) がおり、彼らは大規模なアプリケーションにおいて重要ではないコード領域から最大限のパフォーマンスを引き出そうとします。

この記事の続きはこちら (英語) でご覧になれます。 >

システムと IoT アプリケーションを高速化

インテル® System Studio 2019 の新機能

新しい電力およびパフォーマンス・ツールにより、開発サイクルを短縮して迅速に市場へ投入

高度なクラウドコネクターと 400 を超えるセンサーを利用可能。

パフォーマンス・ボトルネックを素早く特定、電力使用を軽減、ほか

無料のダウンロード >

https://www.isus.jp/intel-system-studio/

[1] 性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、www.intel.com/benchmarks (英語) を参照してください。

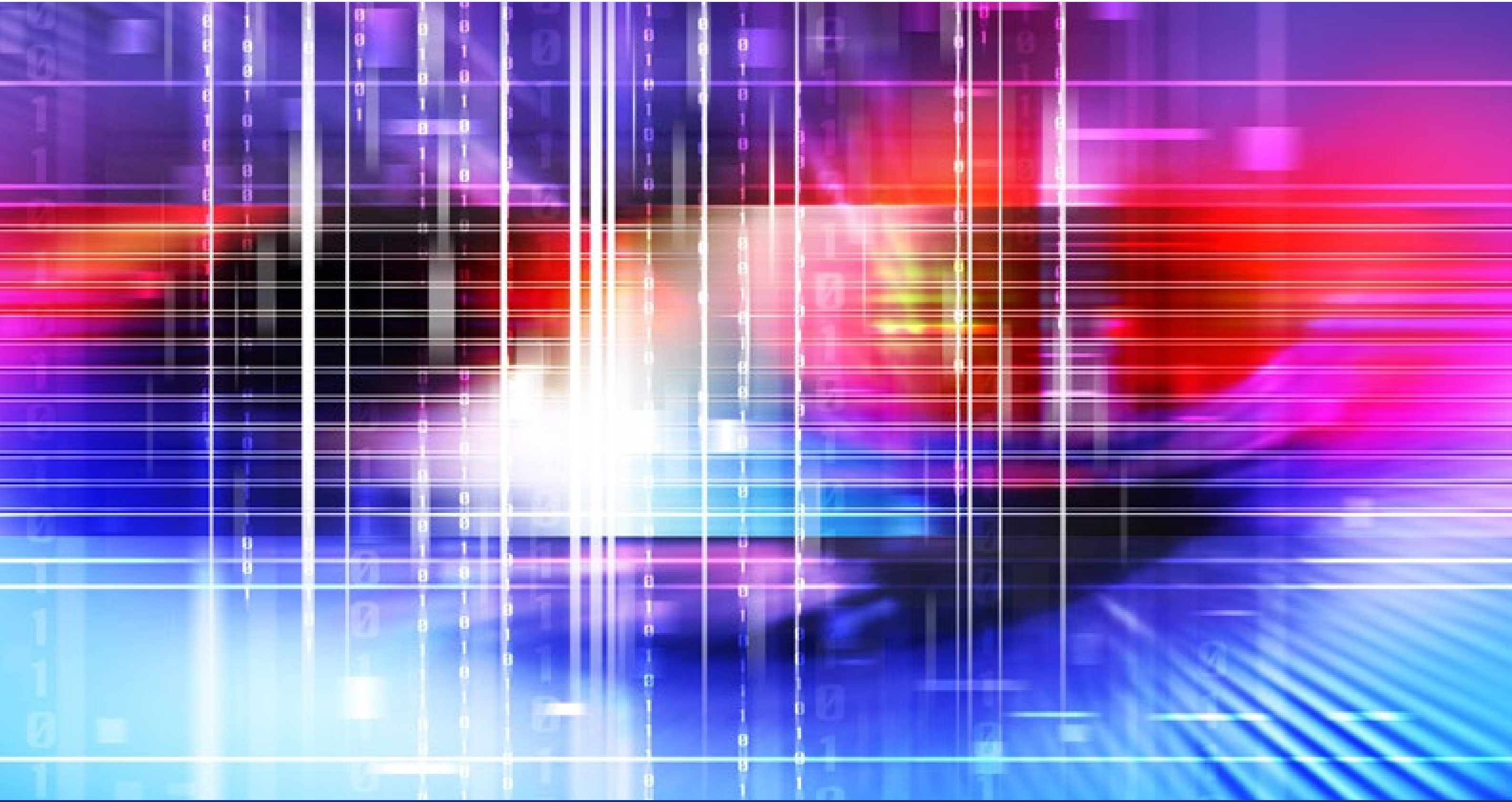
詳細は、software.intel.com/en-us/intel-parallel-studio-xe/details#configurations (英語) を参照してください。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (software.intel.com/en-us/articles/optimization-notice#opt-jp) を参照してください。

Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

* 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。

© Intel Corporation.



C++ メモリー割り当てライブラリーの比較

優れた動的メモリー割り当てによりパフォーマンスを大幅に向上

Rama Kishan Malladi インテル コーポレーション テクニカル・マーケティング・エンジニア
Nikhil Prasad 同 GPU パフォーマンス・モデリング・エンジニア

C++ 開発は、コミュニティー主導であるため、開発者はフレームワークを利用することで問題に細かく対応できます。しかし、実装がオープンソースであるため、サードパーティーが C++ ライブラリーの代替実装を開発し、それを独自の実装とすることができます (例 : **インテル® C++ コンパイラー**)。そのため、同じコンポーネントの複数の実装が存在し、ニーズに最適な実装を見つけるのは容易ではありません。

それぞれの実装の違いを明確にするため、**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)** の アロケーターを含む、いくつかのメモリー割り当てライブラリーを比較しました。メモリー割り当ては、プログラミングに不可欠な部分であり、アロケーターの違いはアプリケーション・パフォーマンスに影響します。

この調査では、**OMNeT++** (英語) と **Xalan*-C++** (英語) ベンチマークを使用しました。テストでは、異なるメモリー・アロケーターを使用した場合、これら 2 つのベンチマークの実行時間に大きな違いが見られました。

OMNeT++

OMNeT++ は、オブジェクト指向のモジュラー離散事象ネットワーク・シミュレーション・フレームワークです。汎用アーキテクチャーであるため、次のようなさまざまな問題で利用できます。

- キュー・ネットワークの**モデリング**
- ハードウェア・アーキテクチャーの**検証**
- 離散事象アプローチが適切なシステムの**一般的なモデリングとシミュレーション**

評価したベンチマークは、10 ギガビット・イーサネット・ネットワークの離散事象シミュレーションを実行します。

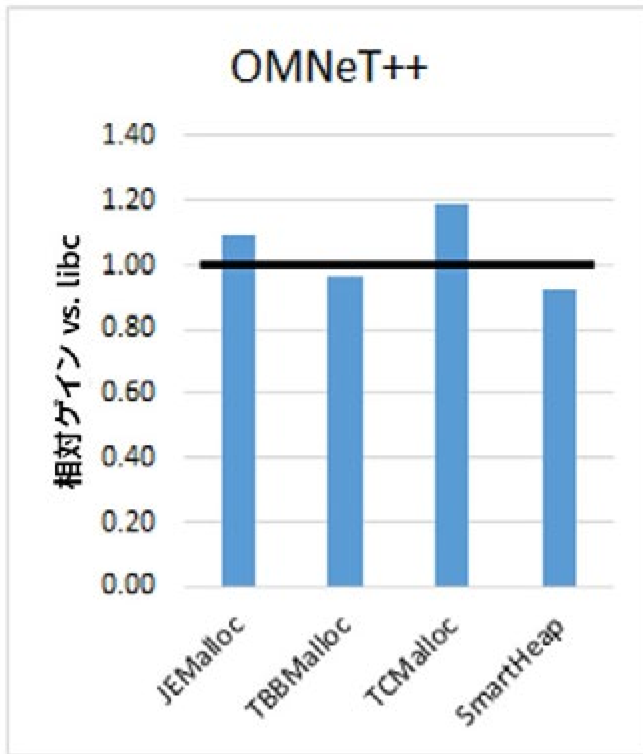
Xalan*-C++

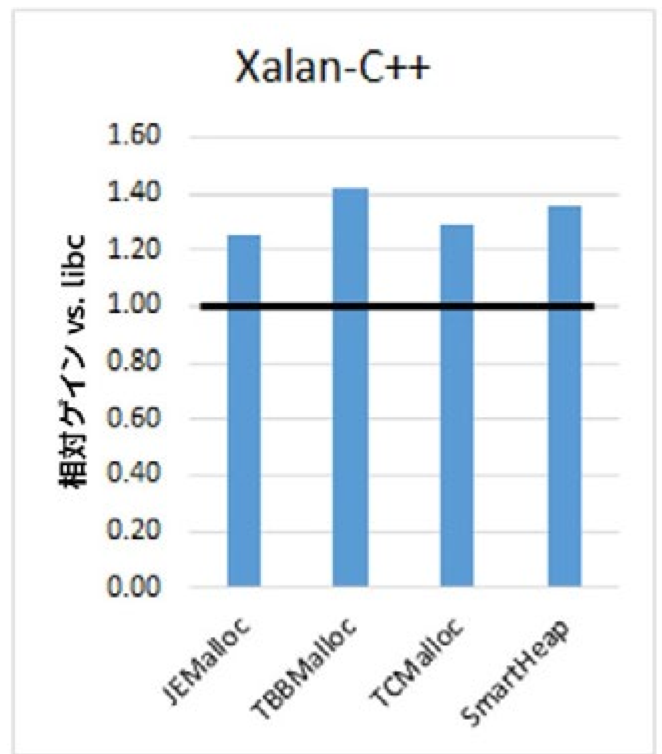
Xalan* は、XML ドキュメントを HTML、テキスト、その他の XML ドキュメント・タイプに変換する XSLT プロセッサーです。Xalan*-C++ バージョン 1.10 は、XSL 変換 (XSLT) と XML パス言語 (XPath) に関する W3C* 勧告の堅牢な実装です。Xerces*-C++ XML パーサーの互換リリースである、Xerces*-C++ バージョン 3.0.1 で動作します。評価したベンチマーク・プログラムは、C++ の移植可能なサブセットで記述された XSLT プロセッサーの Xalan*-C++ を変更したものです。

パフォーマンス・データ

図 1 は、2 つのベンチマークのパフォーマンスは、異なるメモリー割り当てライブラリーを使用した場合、大きく異なることを示しています。よく理解するため、実装の違いを調査しました (**表 1**)。

インテル® TBB
効率良い並列プログラミングへのショートカット
無料
ダウンロード
(英語)





パフォーマンス結果は 2018 年 7 月 9 日現在の測定値であり、すべての公開済みセキュリティー・アップデートが適用されていない可能性があります。詳細は、システム構成を参照してください。絶対的なセキュリティーを提供できる製品はありません。性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細は、パフォーマンス・ベンチマーク・テストの開示 (英語) を参照してください。2018 年 7 月 9 日現在のインテル社内の測定値です。システム構成 : デュアルソケット インテル® Xeon® Gold 6148 プロセッサー、192GB DDR4 メモリー、Red Hat* Enterprise Linux* 7.3。. インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。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® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。注意事項の改訂 #20110804。

1 **デュアルソケット インテル® Xeon® Gold 6148 プロセッサー、192GB DDR4 メモリー、Red Hat* Enterprise Linux* 7.3 での実行結果**

表 1. メモリー割り当てライブラリーの簡単な比較

	Malloc	JEMalloc	TBBMalloc	TCMalloc	SmartHeap
メモリー・トラッキング	フリーストアからメモリーを割り当て	アリーナを使用	リンクリストを使用	リンクリストを使用	リンクリストを使用
スピードアップ機能	割り当てサイズに基づくビニング	割り当てサイズに基づくビニング スレッド固有のキャッシング	割り当てサイズに基づくビニング スレッドローカル・キャッシュと中央キャッシュ	割り当てサイズに基づくビニング スレッドローカル・キャッシュと中央キャッシュ	メタデータの追加ヘッダー

メモリー割り当てライブラリー

ほかのメソッドの影響は無視できることが分かったため、**new** 演算子と **new []** 演算子の実装についてのみ述べます (**図 2**)。

```
void* operator new  ( std::size_t count );
void* operator new[]( std::size_t count );
```

2 演算子の構文

new 演算子は、単一のオブジェクトに必要なストレージの割り当てに使用されます。標準ライブラリーの実装は、フリーストアから **count** バイトを割り当てます。失敗した場合、標準ライブラリーの実装は **std::get_new_handler** によって返された関数ポインターを呼び出し、**new_handler** 関数がリターンしなくなるか NULL ポインターとなり **std::bad_alloc** をスローするまで、繰り返し割り当てを試みます。この関数は、基本的なアライメントのオブジェクトを格納するため、適切にアライメントされたポインターをリターンするのに必要です。

new[] 演算子は、**new** の配列形式で、オブジェクト配列に必要なストレージを割り当てます。標準ライブラリーの **new[]** 実装は **new** を呼び出します。

デフォルトの malloc

標準の C ライブラリーで実装されているデフォルトの **malloc** の呼び出しフローを**図 3** に示します。

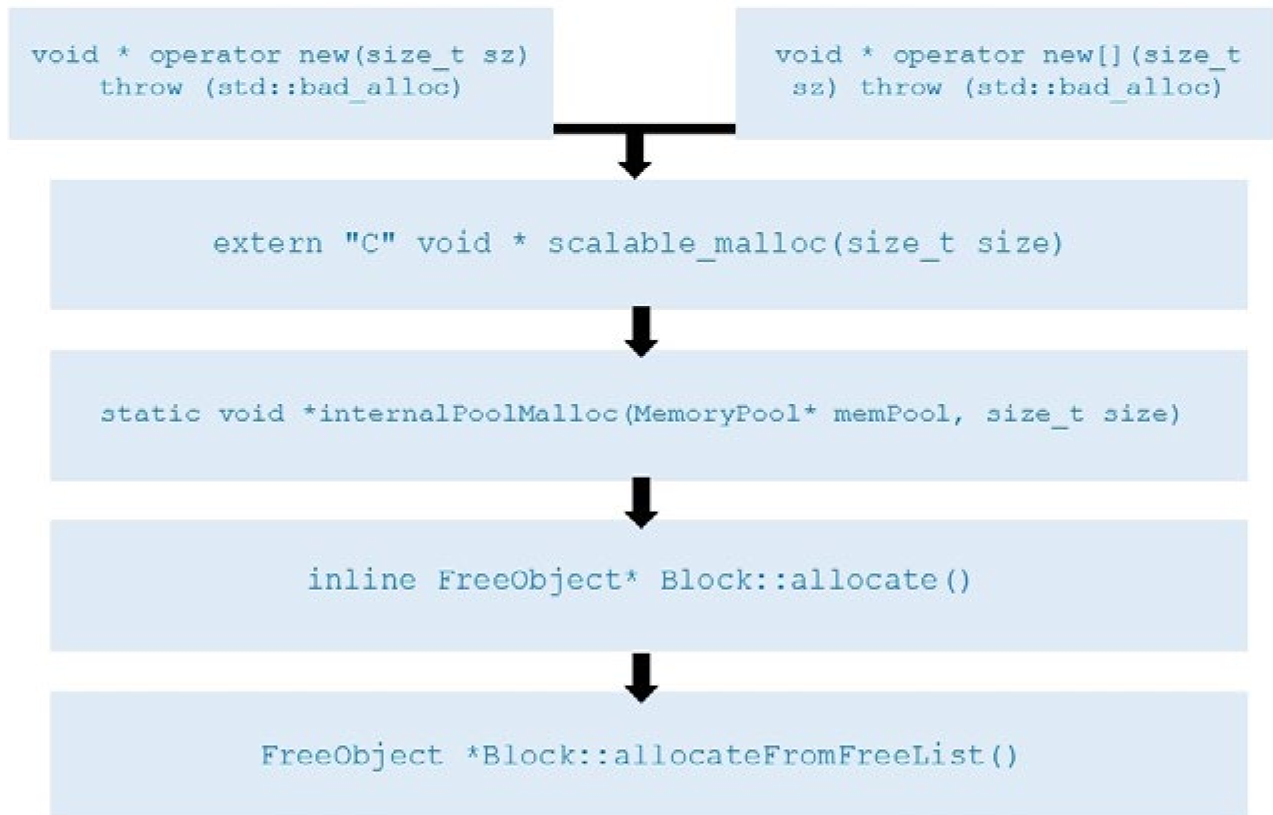
```
operator new (std::size_t sz) _GLIBCXX_THROW (std::bad_alloc)
```

```
void* __libc_malloc(size_t bytes)
```

3 デフォルトの malloc 呼び出しフロー

TBBMalloc

インテル® TBB では、デフォルトの **malloc** 実装はオーバーライドされます。この実装は、並列処理のスケーラビリティーを向上することを目標としています。メモリー内のフリーチャンクのリンクリストを、異なるサイズクラスごとに個別のフリーリストとして維持します (従来の **malloc** のビンに似ています)。空間の局所性を向上するため、できるだけスレッドローカル・キャッシュを使用し、スレッドローカル・フリー・リストで割り当てを実行します。**図 4** は、デフォルトの **TBBMalloc** 呼び出しフローです。



4 **デフォルトの TBBMalloc 呼び出しフロー**

JEMalloc

伝統的に、アロケーターは **sbrk(2)** を使用してメモリーを取得してきました。これは、競合状態、断片化の増加、最大使用可能メモリーの人為的制限を含むいくつかの理由から最適ではありません。オペレーティング・システムで **sbrk(2)** がサポートされる場合、**JEMalloc** アロケーターは **mmap(2)** と **sbrk(2)** の順に両方を使用します。そうでない場合、**mmap(2)** のみを使用します。マルチプロセッサー・システム上のスレッド化されたプログラムでは、アロケーターはロックの競合を軽減するため複数のアリーナを使用します。これは、スレッド化のスケーラビリティーを向上しますが、コストがかかります。アリーナごとにわずかな固定コストが発生します。

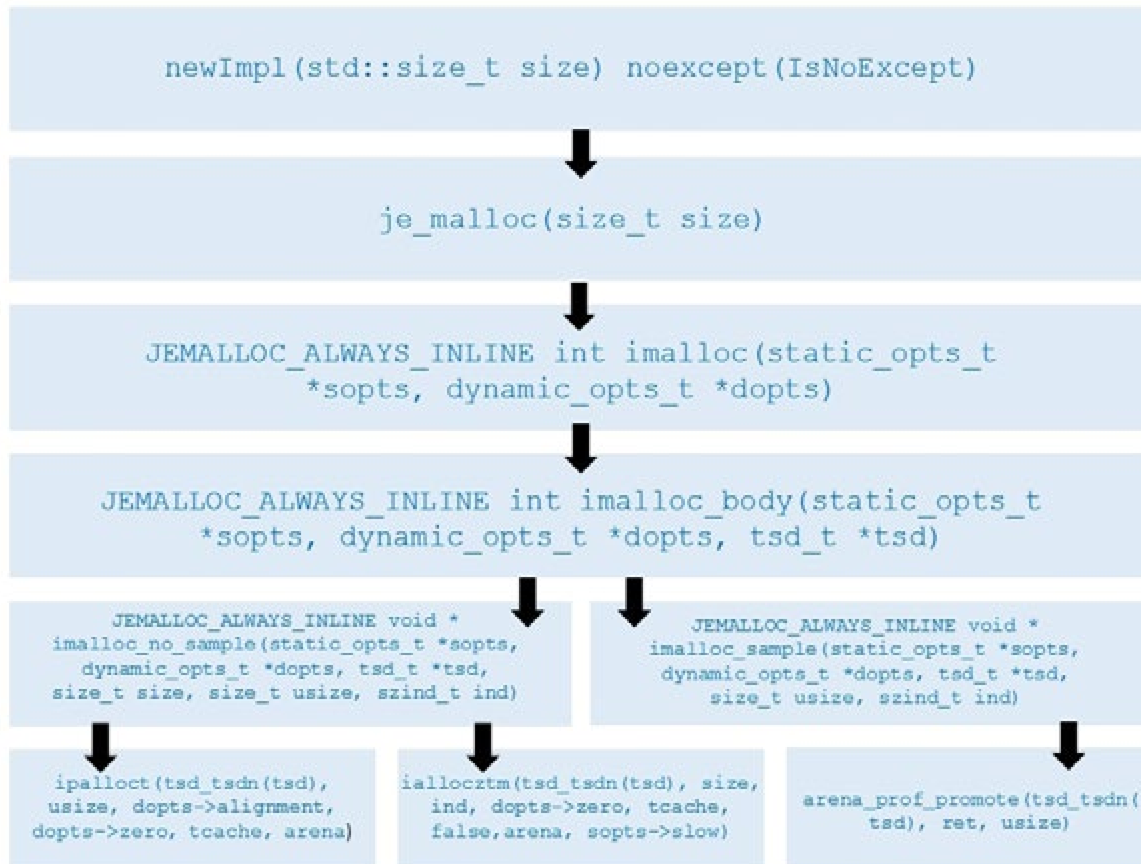
さらに、アリーナはメモリーを完全に独立して管理するため、メモリー全体の断片化がわずかに固定的に増加します。これらのオーバーヘッドは、通常使用されるアリーナの数では、一般に問題になりません。複数のアリーナに加えて、ほとんどの割り当て要求で同期を完全に回避できるように、このアロケーターはスレッド固有のキャッシングをサポートします。このキャッシングは、ほとんどのケースで非常に高速な割り当てを可能にしますが、各スレッドキャッシュに割り当てたままにできるオブジェクトの数には制限があるため、メモリー使用量と断片化が増加します。

メモリーは概念的に範囲に分割されます。範囲は常にページサイズの倍数にアライメントされます。このアライメントにより、ユーザー・オブジェクトのメタデータを素早く見つけることができます。ユーザー・オブジェクトは、サイズに応じて「小」と「大」に分類されます。連続する小さなオブジェクトはスラブを構成し、スラブは単一の範囲内に存在します。大きなオブジェクトにはそれぞれ個別の範囲があります。

従来のメモリー割り当て関数の定義はオーバーライドされます。**図 5** は、**new** 演算子の代替の関数呼び出しです。

Smartheap

Smartheap は、小さなオブジェクトに固定サイズのアロケーターを使用します。その他のオブジェクトでは、フリーリスト内のページの最大フリーブロックのみを格納します。フリーリストのメモリートラバースのランダム性を軽減するため、同じページの連続するオブジェクトを割り当てます。**free()** 呼び出しの後にフリーリストをトラバースするのではなく、ブロックヘッダーのビットを使用して隣接するフリーブロックのチェックとマージを行います。



5 **JEMalloc 関数呼び出し**

TCMalloc

TCMalloc は、各スレッドにスレッドローカル・キャッシュを割り当てます。小さな割り当てはスレッドローカル・キャッシュで行われます。必要に応じて、オブジェクトが中央のデータ構造からスレッドローカル・キャッシュに移動されます。スレッドローカル・キャッシュから中央のデータ構造へのメモリーの移行には、定期的なガベージ・コレクションが使用されます。

インテル® VTune™ Amplifer による詳細なパフォーマンス解析

図 6 から **21** は、単一のユーザーフレンドリーなインターフェイスで高度なプロファイル機能を提供する**インテル® VTune™ Amplifier** のパフォーマンス解析の詳細な結果です。

OMNeT++ のパフォーマンス結果

図 6 から **9** は、**new** 演算子と **new []** 演算子の結果を示しています。

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle / Poor / Ok / Ideal / Over)
__dynamic_cast	19.349s
operator new	8.850s
operator new[]	6.732s
func@0x87db0	0.120s
operator delete	0.120s

6 **libc malloc パフォーマンス解析**

Function / Call Stack	CPU Time
rml::internal::Block::allocateFromFreeList	12.569s
rml::internal::Block::allocate ← rml::internal::internalPoolMalloc ←	12.569s
operator new[] ← opp_strdup ← cNamedObject::setName	6.433s
operator new	6.136s

7 **TBBMalloc パフォーマンス解析**

▸ newImpl<(bool)0>	12.156s	
▸ free	9.553s	
▸ func@0x5a60	0.156s	
▸ operator new[]	0.148s	

8 **JEMalloc パフォーマンス解析**

▾ shi_allocSmall2	22.232s	
▾ ↖ MemAllocPtr ← malloc	22.232s	
▸ operator new	12.752s	
▸ operator new[]	9.468s	
▸ ↖ std::__cxx11::basic_string<char, std::char_trai	0.012s	

9 **Smartheap パフォーマンス解析**

TCMalloc ライブラリーのアプリケーション・プロファイルを取得する試みは失敗しました。アセンブリー・コードにある **mov** と **cmp** 命令のグループがライブラリー間のパフォーマンスの違いを示しています。(**図 10** から **13**)。

0x48413a	204	movq 0x28(%rdi), %r9	0%	0.424s
0x48413e	204	movq (%r9,%rsi,8), %r11	0%	0.368s
0x484142	204	movq 0x68(%r11), %r10		
0x484146	204	cmp %rax, %rsi	0%	24.496s
0x484149	204	jnl 0x484184 <Block 10>		

10 **libc malloc パフォーマンス**

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

0x48423a	204	movq 0x28(%rdi), %r9	0.492s
0x48423e	204	movq (%r9,%rsi,8), %r11	0.368s
0x484242	204	movq 0x68(%r11), %r10	
0x484246	204	cmp %rax, %rsi	15.756s
0x484249	204	jnl 0x484284 <Block 10>	

11 TBBMalloc パフォーマンス

0x48420a	204	movq 0x28(%rdi), %r9	0.300s
0x48420e	204	movq (%r9,%rsi,8), %r11	0.256s
0x484212	204	movq 0x68(%r11), %r10	
0x484216	204	cmp %rax, %rsi	12.304s
0x484219	204	jnl 0x484254 <Block 10>	

12 JEMalloc パフォーマンス

0x4841ba	204	movq 0x28(%rdi), %r9	0.572s
0x4841be	204	movq (%r9,%rsi,8), %r11	0.360s
0x4841c2	204	movq 0x68(%r11), %r10	
0x4841c6	204	cmp %rax, %rsi	18.076s
0x4841c9	204	jnl 0x484204 <Block 10>	

13 Smartheap パフォーマンス

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

Xalan*-C++ のパフォーマンス結果

図 14 から **17** は、**new** 演算子と **new []** 演算子の結果を示しています。

operator new	5.268s
std::ostream::write	0.152s
operator delete	0.048s
func@0x87db0	0.032s

14 **libc malloc の Xalan*-C++ パフォーマンスへの影響**

rml::internal::MemoryPool::getFromLLOCache	1.996s
rml::internal::removeBackRef	0.824s
rml::internal::Bin::processLessUsedBlock	0.600s
rml::internal::ExtMemoryPool::init	0.404s
_ZN3rml10pool_resetEPNS_10MemoryPoolE	0.336s
rml::pool_create	0.316s
rml::internal::Bin::moveBlockToFront	0.264s

15 **TBBmalloc の Xalan*-C++ パフォーマンスへの影響**

newImpl<(bool)0>	3.092s
free	2.104s
operator new	0.160s
je_tcache_event_hard	0.064s

16 **JEMalloc の Xalan*-C++ パフォーマンスへの影響**

_shi_allocBlock	3.116s
_shi_freeVar	1.808s
_shi_allocVar	1.328s
shi_allocSmall2	0.932s
MemFreePtr	0.704s
MemAllocPtr	0.668s
malloc	0.256s

17 **Smartheap の Xalan*-C++ パフォーマンスへの影響**

ここでは、CPU 時間を費やしている上位のいくつかの関数のみを表示しています。コールスタックには多くのほかのメソッドがありますが、実行時間に対するそれらの割合は非常に小さいため無視しています。

図 18 から **21** の一連のアセンブリー命令は、異なるメモリー・アロケーターのパフォーマンス・ゲインを示しています。

0x72c7aa	133	movzxw 0xa(%r14), %esi	4.309s
0x72c7af	133	movzxw 0x8(%r14), %edx	27.037s
0x72c7b4	133	cmp %esi, %edx	8.224s
0x72c7b6	133	jnl 0x72c7db	

18 **libc malloc のアセンブリー命令とパフォーマンス**

0x72c87a	133	movzxw 0xa(%r14), %esi	8.032s
0x72c87f	133	movzxw 0x8(%r14), %edx	13.205s
0x72c884	133	cmp %esi, %edx	3.480s
0x72c886	133	jnl 0x72c8ab	

19 **JEMalloc のアセンブリー命令とパフォーマンス**

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

0x72c8ba	133	movzxw 0xa(%r14), %esi	0.0%	4.620s
0x72c8bf	133	movzxw 0x8(%r14), %edx	0.0%	10.675s
0x72c8c4	133	cmp %esi, %edx	0.0%	2.072s
0x72c8c6	133	jnl 0x72c8eb		

20 **TBBmalloc のアセンブリー命令とパフォーマンス**

0x72c82a	133	movzxw 0xa(%r14), %esi	4.546s
0x72c82f	133	movzxw 0x8(%r14), %edx	10.972s
0x72c834	133	cmp %esi, %edx	1.772s
0x72c836	133	jnl 0x72c85b	

21 **Smartheap のアセンブリー命令とパフォーマンス**

適切なツールで実行時間をスピードアップ

テストしたメモリー割り当てライブラリーのパフォーマンスは、デフォルトのメモリー・アロケーターと比較して大幅な向上を示しています。アセンブリー・コード内の hotspot から、各アロケーターのメモリーの処理方法によりメモリーアクセスが高速化され、パフォーマンスの向上につながったと予測できます。一部のアロケーターは、その主張どおり、断片化の軽減に成功しています。メモリー・アクセス・パターンの詳細な解析により、これらの仮定を裏付けることができます。アプリケーションと注目するメモリー・アロケーター・ライブラリーをリンクするだけで、メモリー・アロケーターを簡単に評価できます。

参考文献

- **OMNeT++ 離散事象シミュレーター (英語)**
- **Xalan*–C++ バージョン 1.10 (英語)**
- **cppreference.com (new 演算子と new[] 演算子) (英語)**
- **JEMalloc (英語)**
- **MicroQuill (英語)**
- **TCMalloc (英語)**
- **インテル® C++ コンパイラー**
- **インテル® TBB**
- **インテル® VTune™ Amplifier**

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

LIBXSMM: インテルのハードウェアとソフトウェア開発にインスピレーションを与えるオープンソース・プロジェクト

特殊な密行列 / 疎行列演算とディープラーニング・プリミティブ向けのインテル® アーキテクチャーをターゲットとするライブラリー

Hans Pabst インテル コーポレーション アプリケーション・エンジニア
Greg Henry 同パスファインディング・エンジニア
Alexander Heinecke 同リサーチ・サイエンティスト

LIBXSMM とは？

LIBXSMM (英語) は、次の 2 つを主な目的としたオープンソース・ライブラリーです。

1. 将来のハードウェアとソフトウェアの方向性 (例えば、協調設計) を**リサーチ**する
2. 科学を**促進**し、オープンソース・ソフトウェア開発にインスピレーションを与える

LIBXSMM は、**インテル® Xeon® プロセッサー**を中心としたインテル® アーキテクチャーをターゲットとしています。一般に、サーバークラスのプロセッサー向けに最適化されていますが、マイクロアーキテクチャーの共通性により、

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

デスクトップ CPU や互換プロセッサーにも適用できます。LIBXSMM は、オープンソース開発の利点を生かし、関連ツールチェーン (GNU* GCC、Clang など) と互換性があります。その重要な技術革新は、JIT (Just-in-Time) コード生成です。コードの特殊化 (実行時に限定されない) は、他に類のないパフォーマンスを引き出します。LIBXSMM は以下を高速化します。

- 小行列乗算とパックド行列乗算
- 行列の転置とコピー
- スパース機能
- 小さな畳み込み

LIBXSMM は、(2015 年以降) **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® MKL-DNN の開発に情報を提供してきたリサーチコードです。ソースコードと Linux* (RPM* および Debian* ベースのディストリビューション) 向けビルド済みパッケージとして提供されています。

行列乗算

この記事では、最初にカバーされた関数ドメインである小行列乗算 (SMM) に注目します。LIBXSMM は、汎用行列 - 行列乗算 (GEMM) の業界標準インターフェイスとバイナリー互換であり、既存の GEMM 呼び出しをインターセプトできます。呼び出しのインターセプトは、BLAS ライブラリー (インテル® MKL など) と静的または動的にリンクされるアプリケーションで動作します。動的なケース (`LD_PRELOAD`) では、再リンクは不要です。どちらの場合も、ソースコードを変更する必要はありません。LIBXSMM の独自の API ではさらに優れたパフォーマンスを達成できます。

$C_{m\ x\ n} = alpha \cdot A_{m\ x\ k} \cdot B_{k\ x\ n} + beta \cdot Cm\ x\ n$ で、alpha ≠ 1、beta ≠ {1, 0}、TransA ≠ {'N', 'n'}、または TransB ≠ {'N', 'n'} の場合、LIBXSMM は LAPACK/BLAS にフォールバックします。しかし、SMM (マイクロカーネル) を転置とコピーカーネルとともに使用することで、alpha-beta の制約のみが残るようにすることができます。例えば、`libxsmm_gemm` は、このケースや $(M\ N\ K)^{1/3}$ が調整可能なしきい値を超えた場合、フォールバックします (**図 1**)。

インテル® MKL
インテル® アーキテクチャー・ベースのシステムで演算処理を高速化
無料
ダウンロード

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

```
char transa = 'n', transb = 'n';
int m = 25, n = 7, k = 75;
int lda = m, ldb = k, ldc = m;

double a[lda*k], b[ldb*n], c[ldc*n];
double alpha = 1, beta = 1;

libxsmm_dgemm(&transa, &transb, &m, &n, &k,
  &alpha, a, &lda, b, &ldb,
   &beta, c, &ldc);
```

1 **libxsmm_dgemm では、SMM は設定可能なしきい値に応じて処理される (簡略化のため行列の初期化は省略)**

BLAS の依存関係は、LIBXSMM (インテル® MKL、OpenBLAS など) のリンク時ではなく、アプリケーションのリンク時に解決されます。フォールバックが不要な場合､BLAS はダミー関数に置換できます (**libxsmmnoblas**)。

コードのディスパッチ

libxsmm_dgemm (または **dgemm_**) を呼び出すと、自動的に JIT 生成コードがディスパッチされます。LIBXSMM の強力な API は、しきい値以外をカスタマイズしたり、あるいはコードを生成またはクエリーしてアプリケーションの存続期間中有効な通常の関数ポインター (C/C++) または PROCEDURE POINTER (Fortran) (スタティックまたは SAVE) を返すことができます (**図 2**)。

```
int flags = LIBXSMM_GEMM_FLAG_NONE;
libxsmm_dmmfunction fun = libxsmm_dmmdispatch(

  m, n, k, &lda, &ldb, &ldc, &alpha, &beta,
  &flags, NULL/*prefetch*/);

fun(a, b, c);
```

2 **図 1 と同じ乗算を実行する手動でディスパッチされた SMM**

libxsmm_dmmdispatch のポインター引数では、NULL はデフォルト値を示します。LDx は、蜜なリーディング･ディメンジョンを意味します。alpha、beta、またはフラグのデフォルトは、コンパイル時の設定により決まります。

C/C++ か、Fortran か ?

これまで C コードのみを見てきましたが、より自然なユーザー構文を利用可能な C++ 固有の要素があります (型プリフィクスを回避するためオーバーロードされた関数名、および型に適したオーバーロード関数を推論する関数テンプレート)。また、C インターフェイスが (オプション引数を指定するため) NULL 引数を受け付ける場合、C++ および Fortran インターフェイスでは、それらの引数を省略できます (**図 3**)。

```
libxsmm_mmfunction<double> fun(m, n, k,
  &lda, &ldb, &ldc, &alpha, &beta);

fun(a, b, c);
```

3 **型テンプレート libxsmm_mmfunction は SMM カーネルを表すファンクターを指定**

LIBXSMM には明示的な Fortran インターフェイス (**IMPLICIT NONE**) があり、コンパイラー固有のモジュールに (事前) コンパイルしたり、単純にアプリケーションにインクルードできます (**libxsmm.f**)。このインターフェイスを利用するには、Fortran 2003 標準をサポートするコンパイラーが必要です。Fortran 77 では、**ISO_C_BINDING** を必要とせず暗黙的にこの機能のサブセットにアクセスできます (**図 4**)。

```
TYPE(LIBXSMM_DMMFUNCTION) :: fun
CALL libxsmm_dmmdispatch(fun, m, n, k, &
  lda, ldb, ldc, alpha, beta)

CALL libxsmm_dmmcall(fun, a, b, c)
```

4 **前述の C/C++ 例のようなコードのディスパッチ。追加の汎用プロシージャーの多重定義により、型プリフィクスを省略可能 (libxsmm _ mmdispatch と libxsmm _ mmcall)。**

LIBXSMM は、C++ と C ではヘッダーのみの使用がサポートされます (C ではこれはまれです)。ファイル **libxsmm_source.h** を利用することで、ライブラリーをビルドしなくても済みますが、明確に定義されたアプリケーション・バイナリー・インターフェイス (ABI) は利用できません。ヘッダーファイルには、実装 (**src** ディレクトリー) が含まれるため、意図的に **libxsmm_source.h** という名前が付けられています。汎用プログラミング (テンプレート) の登場以来、C++ コードでは一般に ABI がサポートされなくなっています。ABI は、配布後に動的にリンクされるアプリケーションのホットフィックスを可能にします。ヘッダーのみをサポートする場合、一般にアプリケーションの開発期間を短縮できます (コンパイル時間は長くなります)。

汎用型 API

これまでに紹介した言語インターフェイスはすべてタイプセーフであり、コンパイル時に型推定が行われます (Fortran 77 はインターフェイスごとにサポートされません)。C/C++ および Fortran 向けに追加の低レベルの記述子またはハンドルベースのインターフェイスが用意されています。これは、ほかのライブラリーとの統合を容易にしたり、汎用プログラミングを可能にします (**図 5**)。

```
libxsmm_descriptor_blob blob;
libxsmm_xmmfunction fun;
libxsmm_gemm_descriptor* desc;
desc = libxsmm_gemm_descriptor_init(&blob,
  LIBXSMM_GEMM_PRECISION_F64, m, n, k, lda, ldb, ldc,
  &alpha, &beta, flags, NULL/*prefetch*/);
fun = libxsmm_xmmdispatch(desc);

fun.xmm(a, b, c);
```

5 **ハンドルベースのアプローチを使用した低レベルのコード・ディスパッチ。記述子は前方宣言されているだけですが、動的メモリー割り当てなしで効率良く作成できます (desc を有効にするには blob が有効でなければなりません)。**

通常、カーネルを呼び出すたびにディスパッチすることは避けられます。これは、同じ SMM が連続して呼び出される場合には簡単です (ディスパッチ API の理想的なアプリケーションと言えます)。ディスパッチのコストは、約数十ナノ秒です。JIT コード生成は、最初にコードバージョンが要求されたときにのみ行われ、数十マイクロ秒かかります。LIBXSMM のすべての関数と同様に、コード生成とディスパッチはスレッドセーフです。すべての GEMM 引数がキーとして機能するため、コードのクエリーも簡単に行うことができます。さらに、JIT 生成コード (関数ポインター) は、(スタティック・コードを参照する通常の関数ポインターと同様に) プログラムが終了するまで有効です。

プリフェッチ

バッチ SMM は、同じカーネルを複数の乗算に使用できます。次の場所を渡すことで、後続の処理のオペランドを事前にプリフェッチできます。LIBXSMM の場合、次に乗算する行列のアドレスが次の場所です。次のオペランドは、メモリー内で連続している必要はありません。

プリフェッチ手法 (**`LIBXSMM_PREFETCH_NONE`** 以外) は、カーネルのシグネチャーを変更して 3 つの引数 (**a**、**b**、**c**) の代わりに 6 つの引数 (**a**、**b**、**c** と **pa**、**pb**、**pc**) を受け取ります。また、**`LIBXSMM_PREFETCH_AUTO`** は、オペランドがストリーミングされると仮定し、**`CPUID`** に基づいて手法を選択します (**図 6**)。

```
int prefetch = LIBXSMM_PREFETCH_AUTO;
libxsmm_dmmfunction fun = libxsmm_dmmdispatch(
  m, n, k, &lda, &ldb, &ldc, &alpha, &beta,
  &flags, &prefetch);

fun(a, b, c, pa, pb, pc);
```

6 **プリフェッチの場所を受け取るカーネル (pa、pb、pc)**

プリフェッチ手法に対して **`NULL`** ポインターを使用すると、プリフェッチは参照されず、**`LIBXSMM_PREFETCH_NONE`** を指定した場合と同じです。設計上、呼び出し位置を変更せずに手法を調整できます (6 つの有効な引数を渡す必要があります) (**図 7**)。

インテル® MPI ライブラリー
柔軟で、効率良い、スケーラブルなクラスター・メッセージング
無料ダウンロード

```
if (0 < n) {
  const int an = lda * k, bn = ldb * n, cn = ldc * n;

# pragma parallel omp private(i)
  for (i = 0; i < (n - 1); ++i) {
    const double* ai = a + i * an;
    const double* bi = b + i * bn;
          double* ci = c + i * cn;
    dmm(ai, bi, ci, ai + an, bi + bn, ci + cn);
  }
  dmm(a + (n - 1) * an, b + (n - 1) * bn,
      c + (n - 1) * cn,
      a + (n - 1) * an, b + (n - 1) * bn,
      c + (n - 1) * cn);
```

7 **次のオペランドをプリフェッチする一連の乗算に同じカーネル (dmm) を使用。範囲外のプリフェッチを回避するため最後の乗算をループからピール。**

無効なアドレスからのプリフェッチは、例外はトラップしませんが、ページフォルトが発生します (これはループの一連の乗算の最後をピールすることで回避できます)。

ドメイン固有ライブラリー

ドメイン固有の言語とライブラリーには、行列乗算だけでなく、小さな問題サイズや SMM に関しても、豊富な歴史があります。インテル® MKL などの完全な LAPACK/BLAS 実装であっても、小さな問題サイズ向けの最適化 (**MKL_DIRECT_CALL**) が含まれており、継続的な取り組みが行われています (インテル® MKL 2019 は SMM カーネルを JIT 生成可能)。主要なオープンソース C++ ライブラリー (出典 : Google Trends*) を比較してみましょう。

- Armadillo*
- Blaze*
- Eigen*
- uBLAS (Boost)

公正を期すため、継続的に開発されており、(例えば、出版物などで) パフォーマンスを公表しているか、パフォーマンスの方向性が明らかなものを比較の候補とします (つまり、uBlas は対象外です)。既存のメモリーバッファーを利用 / 再利用し、リーディング・ディメンジョンをサポートすること (非干渉型デザイン) は、従来のオブジェクトから

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

テンプレートへ移行しても、データ構造 (行列データ型) を持つ C++ ライブラリーでは重要です。LAPACK/BLAS は、アルゴリズムであるため (動作であり、状態を持たないため)、驚くほど STL に似ています。基本的に、BLAS のようなパラメーター化が前提条件になります。最後に、LAPACK/BLAS へのマッピングが完璧であっても新たなパフォーマンス結果が得られないため、候補となるライブラリーは、実装 (フォールバック・コード以外) を提供している必要があります (つまり、Armadillo* は対象外です)。

言語としての C++ と関連する最先端の手法 (式テンプレート) に基づく人気、ドメイン固有言語の設計、継続的な開発、コード品質、パフォーマンスについて検討した結果、最終的に次の理由から Blaze* と Eigen* を評価することになりました。

- **Blaze*** は、ドメイン固有言語の洗練された使いやすさと HPC グレード のパフォーマンスを備えています。
- **Eigen*** は、優れたコンパイラー・サポートを備え、高速で、汎用性と信頼性が高く、洗練されたライブラリーです。[**編集者注**: このライブラリーは、**The Parallel Universe 31 号**の「自動運転ワークロードに適した Eigen* 数学ライブラリーの構築」で紹介しました。]

この記事では、LIBXSMM のサンプル·コレクションで開発されたコードを使用します (**図 8**)。**A**、**B**、**C** がストリーミングされるように (つまり、行列 **A** と **B** が行列 **C** に集計され、**[C += A * B、alpha=1、beta=1]** となるように) 一連の行列を乗算します。これ以外には、行列オペランドをメモリーからロードせずに (最初の反復を除く)、カーネルを実行するだけです。

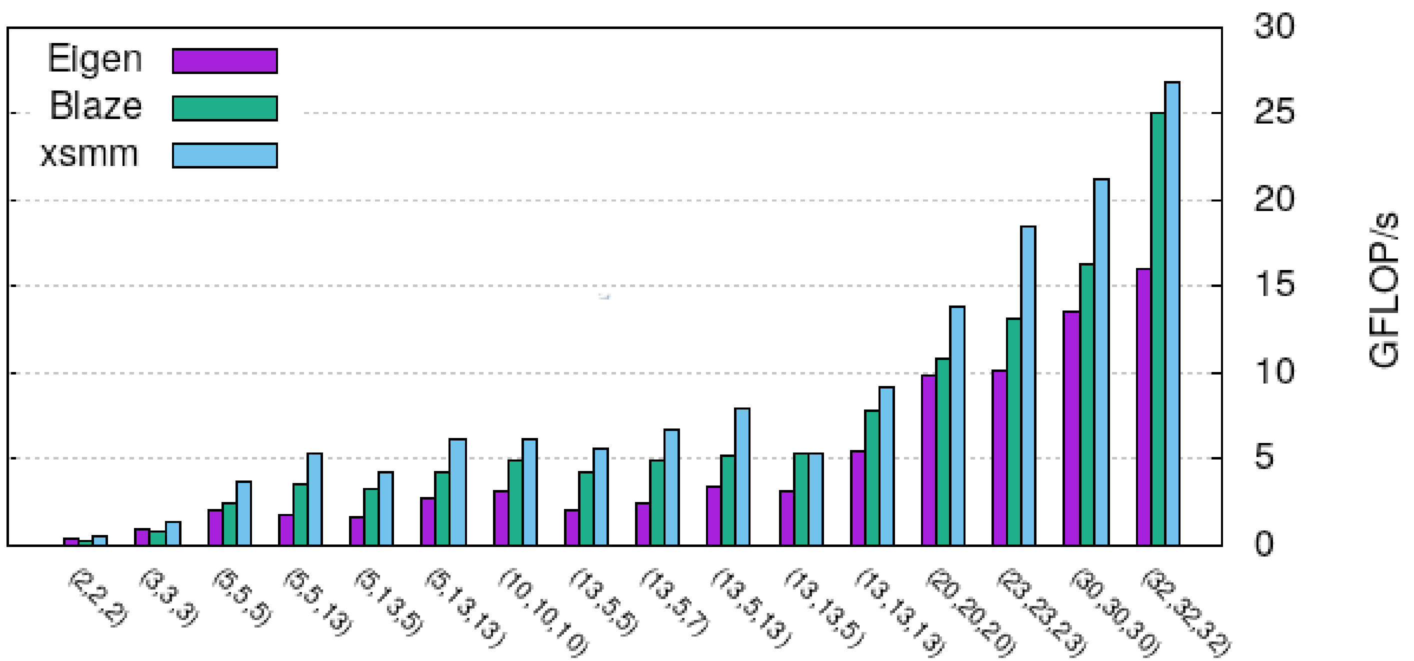
```
for (i = 0; i < n; ++i) {
  const MatrixType ai = Map(a + i * an);
  const MatrixType bi = Map(b + i * bn);
        MatrixType ci = Map(c + i * cn);

  ci += ai * bi;
}
```

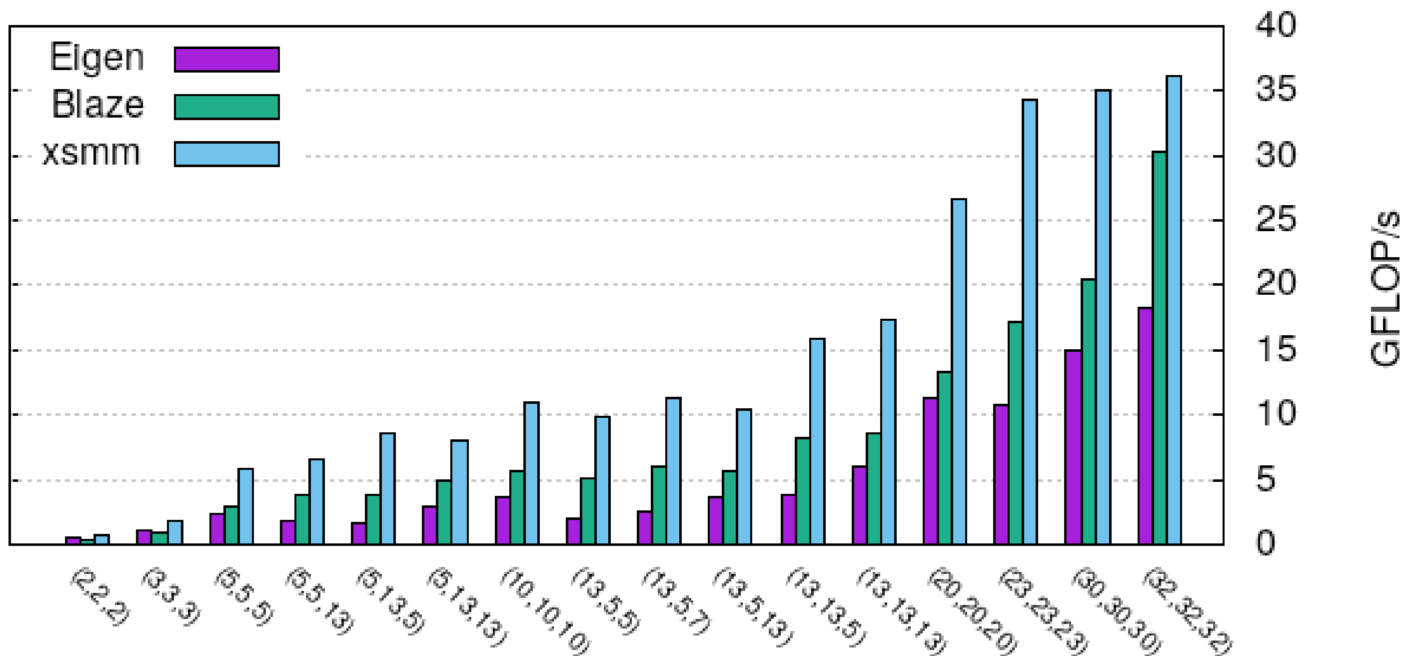
8 **Blaze* と Eigen* で実装したベンチマークの疑似コード。既存データはライブラリーの行列データ型の変数に (理想的にはコピーなしで) マップ。オペランド ai、bi、ci は各反復でロード / ストア。**

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

遅延式評価 (式テンプレート) は評価の対象ではないため、標準 (GEMM) 式 **`ci = alpha * ai * bi + beta * ci`** は除外しました。Blaze* の **`CustomMatrix`** では、ユーザーデータ (リーディング・ディメンジョンを含む) を使用し、標準 GEMM 式の中間コピーを回避しました。Eigen* では、**`Matrix::Map`** の結果を **`Matrix`** (代わりに **`auto`** を使用) に変換するためコピーを使用し、動的ストライドと関連するコピー操作によりリーディング・ディメンジョンのパフォーマンスが低下しました。どちらの問題も回避策をとったことで、パフォーマンス結果には影響していません (**図 9** と **10**)。

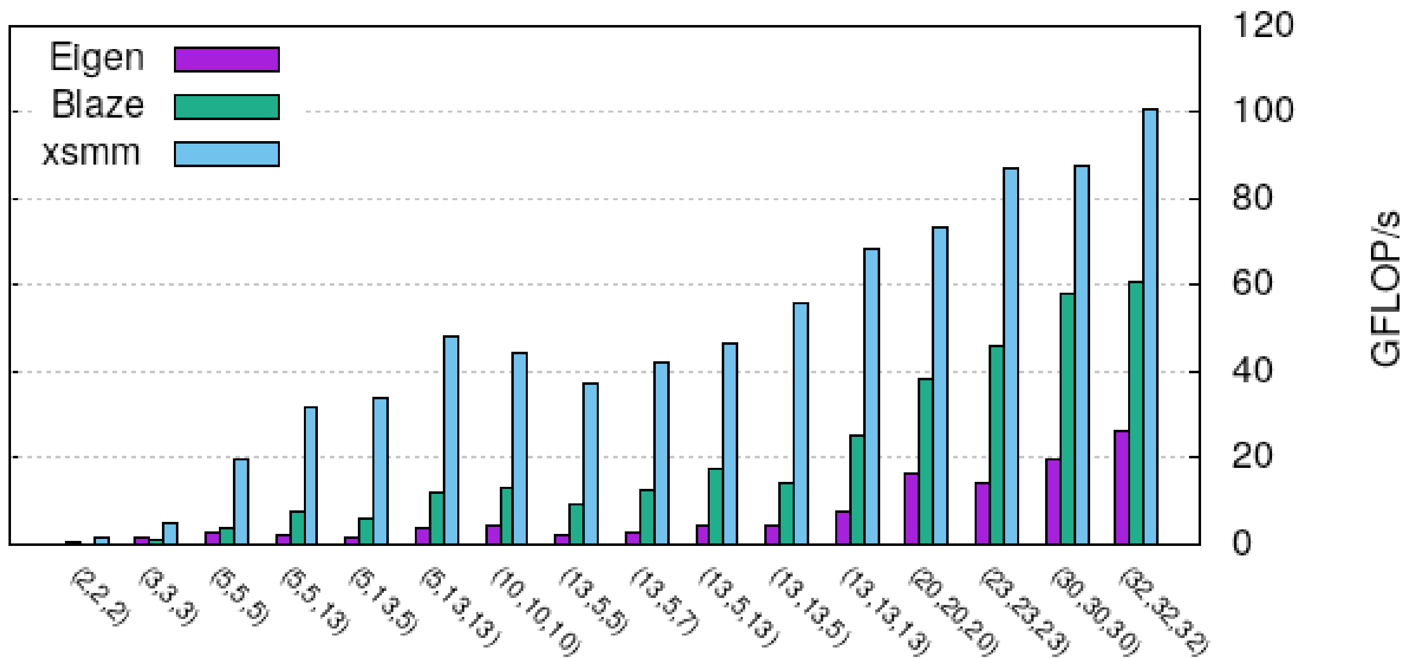


9 **完全にストリーミングされた $C^i \mathrel{+}= A^i * B^i$ の倍精度 SMM (M,N,K) (GNU* GCC 8.2、LIBXSMM 1.10、Eigen* 3.3.5、Blaze* 3.4)。シングルスレッド、インテル® Xeon® プロセッサー 8168 (2666MHz DIMM、HT/Turbo 有効) を使用。**



10 **ストリーミングされた入力の倍精度 SMM (M,N,K) (GNU* GCC 8.2、LIBXSMM 1.10、Eigen* 3.3.5、Blaze* 3.4)。シングルスレッド、インテル® Xeon® プロセッサー 8168 (2666MHz DIMM、HT/Turbo 有効) を使用。例えば、オープンソースの分子動力学シミュレーション CP2K は C_i += A_i * B_i を使用し、SMM をスレッドごとまたはランクごと (MPI) に C 行列に集計。**

SIMD ベクトル処理に必要な範囲を超えてカーネルをアンロールすることは、キャッシュ負荷の高い場合にのみ役立ちます。倍精度のオペランド **A**、**B**、**C** をストリーミングし、かつ **C** は所有権の読み取理の場合、**FLOPS = 2 * m * n * k**、**BYTES = 8 * (m * k + k * n + m * n * 2)** であり、**C** の所有権を読み取る場合、SMM は計算依存ではなくメモリー依存です。正方形の場合、キャッシュラインの粒度を無視すると、$\mathbf{AI(n) = 2 * n^3 / (32 * n^2) = 0.0625 * n}$ になります (演算強度 (AI) の単位はバイトあたりの浮動小数点演算数 (FLOPS))。例えば、上記の手法でオペランドをストリーミングする 16x16 行列の乗算では、1 FLOPS/ バイトにしかなりません (**図 11**)。しかし、デュアルソケットのインテル® Xeon® プロセッサー・ベースのサーバーなどの科学計算に使用される一般的なシステムでは、倍精度で 15 FLOPS/ バイトになります。そのため、ストリーミングされる SMM では、メモリー操作をうまく隠蔽してメモリー帯域幅を活用する命令の組み合わせにより利点が得られます。



11 **キャッシュ負荷の高い `C += A * B` の倍精度 SMM (M,N,K) (GNU* GCC 8.2、LIBXSMM 1.10、Eigen* 3.3.5、Blaze* 3.4)。シングルスレッド、インテル® Xeon® プロセッサー 8168 (2666MHz DIMM、HT/Turbo 有効) を使用。**

科学アプリケーションを大幅にスピードアップ

LIBXSMM は 2014 年の初めに公開され、2015 年末には SMM 向けのインメモリー JIT コード生成が実装されました。SMM ドメインは安定しており (ほとんどは API への追加)、主要アプリケーションで採用されています。SMM ドメインは、高速で特殊なコードに加えて、高速なコードの生成 (数十マイクロ秒) とクエリー (数十ナノ秒) を提供しています。LIBXSMM ライブラリーは研究に役立ちます。インテルの命令セットマニュアルで新しい命令が公開されるとすぐにそれらをサポートします。LIBXSMM は、従来行列乗算が FLOPS 依存としてのみ認識されていた分野において、科学アプリケーションに大幅なスピードアップをもたらします。

関連資料

ドキュメント

- オンライン : **メイン・ドキュメント** (英語) と**サンプル・ドキュメント** (英語) 全文検索可 (ReadtheDocs)
- PDF: **メイン・ドキュメント** (英語) と**サンプル・ドキュメント** (英語)
- LIBXSMM アプリケーションの一覧は、**GitHub* LIBXSMM ホームページ** (英語) を参照してください。

記事 (英語)

- "**LIBXSMM Brings Deep Learning Lessons Learned to Many HPC Applications**" by Rob Farber, 2018.
- "**Largest Supercomputer Simulation of Sumatra-Andaman Earthquake**" by Linda Barney, 2018.
- SC'18: "**Anatomy of High-Performance Deep Learning Convolutions on SIMD Architectures**"
- SC'16: "**LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation**"

BLOG HIGHLIGHTS

オープンソースとビジュアルクラウドの将来

IMAD SOUSOU

2018 年 7 月 1 日に Netflix* の加入者数が 1 億 3,000 万人に達しました。これは、ロサンゼルス、ニューヨーク、トロント、メキシコシティー、ロンドン、上海、東京の人口の合計を上回ります。

インテル コーポレーションの副社長である Lynn Comp は、「大部分のネットワーク・トラフィックは、間もなくビデオになるでしょう。」と述べています。Netflix* だけでも Amazon Web Services* (AWS*) に数十ペタバイトのストレージがあります。ビデオコンテンツは爆発的な速度で消費および生成されています。AR/VR の訓練からグラフィックス・レンダリングまで、さまざまなビジネス・ワークロードによってクラウドとエッジ・ネットワークの変換の必要性が高まっています。さらに、クラウドゲーミングの広大かつ世界的な将来があります。この要求に応えるため､インテルはビジュアルクラウドの 4 つの基本ビルディング・ブロック (レンダリング、エンコード、デコード、インターフェイス) に対応する新しいオープンソース・ソフトウェア・プロジェクトを発表しました。

この記事の続きはこちら (英語) でご覧になれます。 >

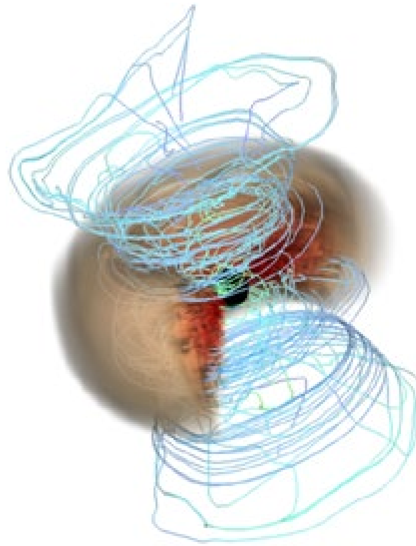
ECHO-3DHPC による天体物理シミュレーションのパフォーマンスを向上

最新のインテル® ソフトウェア開発ツールを利用してハードウェアを効率良く使用する

Matteo Bugli 博士 CEA Saclay 天体物理学者
Luigi Iapichino 博士 LRZ 科学計算エキスパート
Fabio Baruffa 博士 インテル コーポレーション テクニカル・コンサルティング・エンジニア

正確で高速な数値モデリングは、中性子星やブラックホールなどの天体物理の研究には不可欠であり、宇宙の銀河を構成し、高エネルギー放射線の天体物理学的な起源を理解する上で重要な役割を果たします。特に高温プラズマがブラックホールとどのように降着するのかについての研究は重要です。この物理現象は、膨大な量のエネルギーを放出し、宇宙で最もエネルギーの高い物体 (例えば、ガンマ線バースト、活動銀河核、X 線連星など) に作用します。降着シナリオの研究は、特に基本的な物理メカニズムの理解に複数のパラメーターを必要とする場合、計算負荷が高くなります。そのため、相対論的プラズマのモデリングは、計算効率を最大化することで大きな利点が得られます。

ECHO-3DHPC は、3D 有限差分と高次再構成アルゴリズムを使用して一般相対性理論で磁気流体力学方程式を解く Fortran アプリケーションです (**図 1**)。元々は ECHO[1] ベースでしたが、最新バージョンでは多くのコア数にスケーリングする多次元の MPI ドメイン分割スキームを採用しています。



1 ECHO-3DHPC でシミュレーションした厚い降着円盤の質量密度の体積レンダリング (密集領域は赤色、希薄なエンベロープ領域は茶色、流速は流線で表しています)[2]

最近のコードの改善では、OpenMP* によるスレッドレベルの並列処理が追加され、アプリケーションは MPI 通信オーバーヘッドによる現在の制限を超えてスケーリングすることが可能です。**インテル® Fortran コンパイラー**のプロファイルに基づく最適化 (PGO)、**インテル® MPI ライブラリー**、**インテル® VTune™ Amplifier**、**インテル® Inspector** などのインテル® ソフトウェア開発ツールを利用することで、パフォーマンスの問題を調査して、スケーラビリティーの改善と実行時間の短縮を達成できます。

インテル® Fortran コンパイラーを使用してパフォーマンスを最適化

パフォーマンス最適化の鍵は、PGO などのコンパイラーの機能を利用して、開発者がコードレイアウトの並べ替え作業を容易に行えるようにし、命令キャッシュの問題の軽減、コードサイズの縮小、分岐予測ミスの軽減を達成できるようにすることです。PGO には次の 3 つのステップがあります。

1. **コンパイル :** バイナリーをインストルメントする `-prof-gen` オプションを指定してアプリケーションをコンパイルします。
2. **実行 :** ステップ 1 で生成した実行ファイルを実行して動的情報ファイルを生成します。
3. **再コンパイル :** 収集した情報をマージして最適化された実行ファイルを生成する `-prof-use` オプションを指定してコードを再コンパイルします。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

パフォーマンスの評価では、最も大きなグリッドサイズを 16,384 コアで実行した場合、実行時間が最大 15% 向上しました (**表 1**)。

表 1. ハードウェア構成 : デュアルソケット インテル® Xeon プロセッサー E5-2680 v1 @ 2.70GHz、ノードごとに 16 コア。ソフトウェア構成 : SLES11、インテル® MPI ライブラリー 2018

グリッドサイズ	コア数	インテル® Fortran コンパイラー 18 (秒/反復)	インテル® Fortran コンパイラー 18 PGO 使用 (秒/反復)
512^3	8,192	1.27	1.13
512^3	16,384	0.66	0.57
$1{,}024^3$	16,384	4.91	4.22

インテル® VTune™ Amplifier とインテル® Inspector を使用した OpenMP* による最適化

最近、ECHO-3DHPC に OpenMP* によるスレッドレベルの並列処理が追加されました。これにより、CPU ノード内で OpenMP* スレッドを使用して MPI タスクの数を減らし、一部の MPI 通信オーバーヘッドを排除することが可能です。

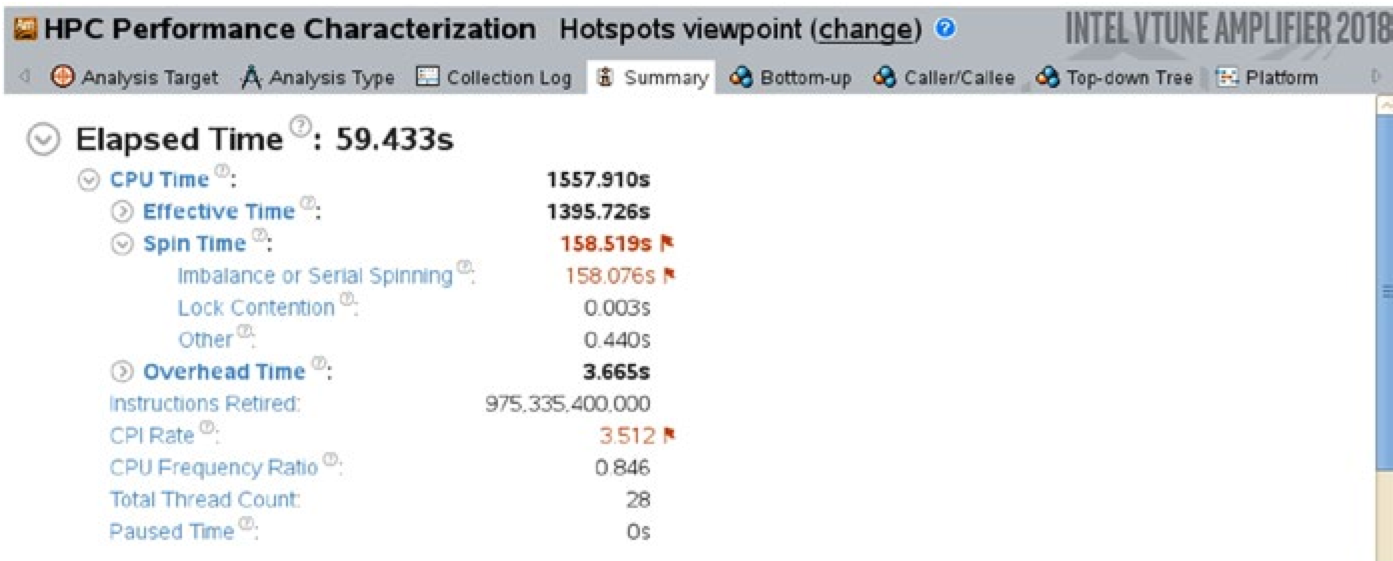
パフォーマンスの最適化は、デュアルソケット インテル® Xeon® プロセッサー E5-2697 v3 (@ 2.60GHz、合計 28 コア) のシングルノード上でのテストに基づいています。インテル® VTune™ Amplifier で HPC Performance Characterization Analysis (HPC パフォーマンス特性解析) を行うには、次のコマンドを実行します。

```
amplxe-cl -collect hpc-performance -- <実行ファイルと引数>
```

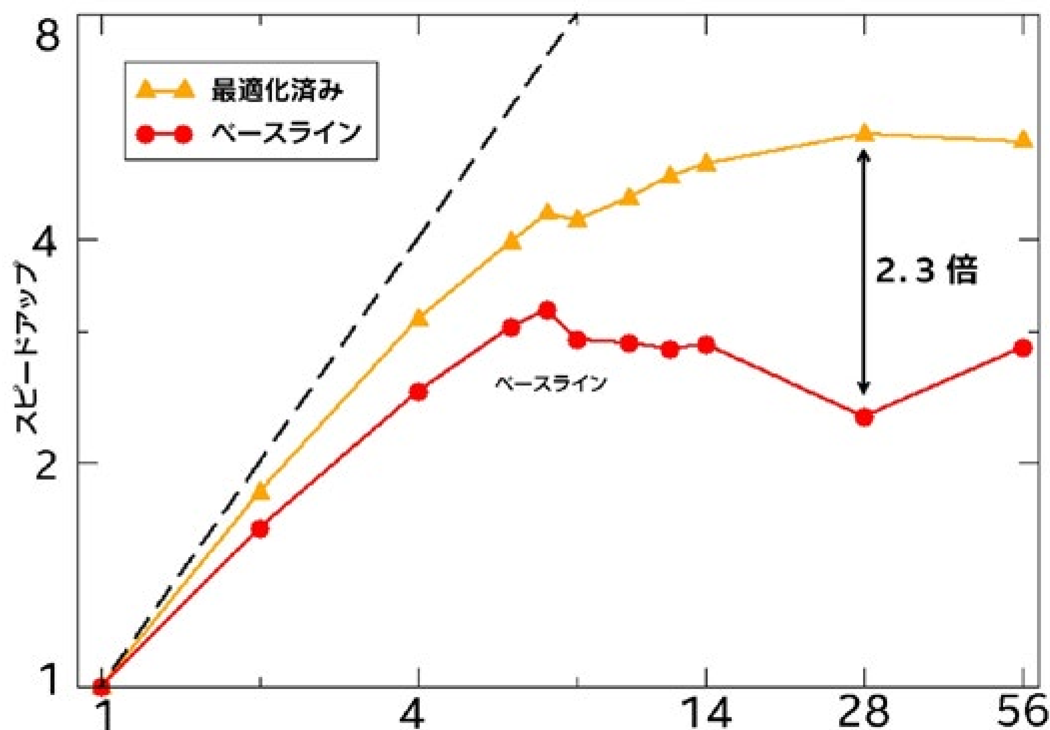
ベースライン・コードのパフォーマンスを解析すると、次のボトルネックが検出されます。

- ほとんどの CPU 時間が Imbalance or Serial Spinning (インバランスまたはシリアルスピン) に分類されています (**図 2**)。これは、シーケンシャル実行が大部分を占めており、ワーカースレッドの並列性が十分ではないためです。
- このインバランスにより、スレッド数が増加するとノードレベルのスケーラビリティーが伸びません (**図 3** の赤色の線)。
- インテル® VTune™ Amplifier のメモリー解析は、コードがメモリー依存であることを示しています。実行パイプライン・スロットの約 80% がメモリーのロードとストア要求によりストールしています (**図 4**)。

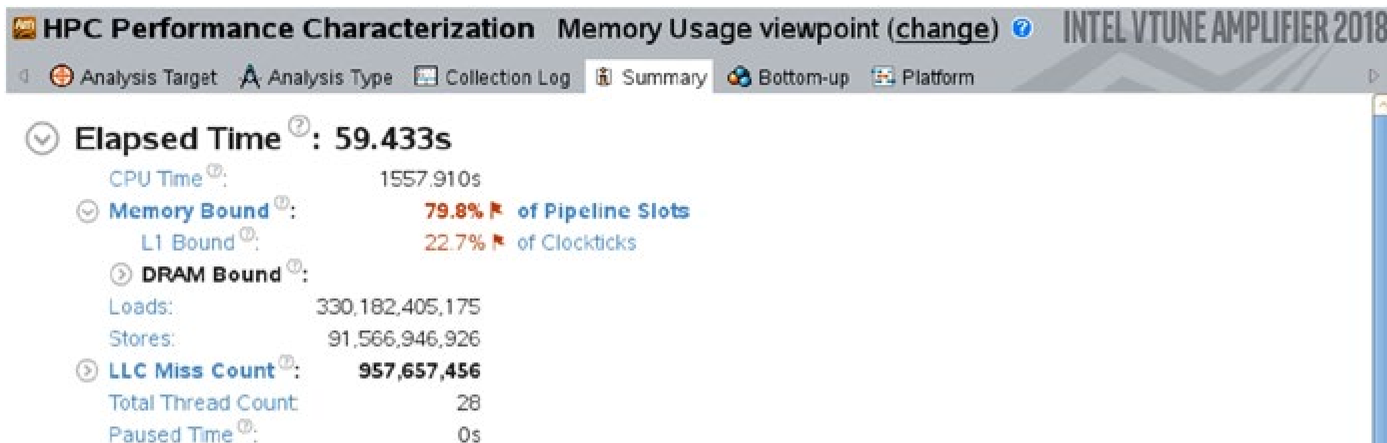
コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。



2 インテル® VTune™ Amplifier で ECHO-3DHPC ベースライン・バージョンの HPC performance Characterization (HPC パフォーマンス特性) 解析を実行して表示された Hotspots viewpoint (ホットスポット・ビューポイント) のスクリーンショット



3 スレッド数の関数としてのノード内の並列スピードアップ (最後のポイントはハイパースレッディング)。赤色の線はベースライン・バージョン、オレンジ色の線は最適化されたバージョン、黒色の点線は理想的なスケーリングを示しています。



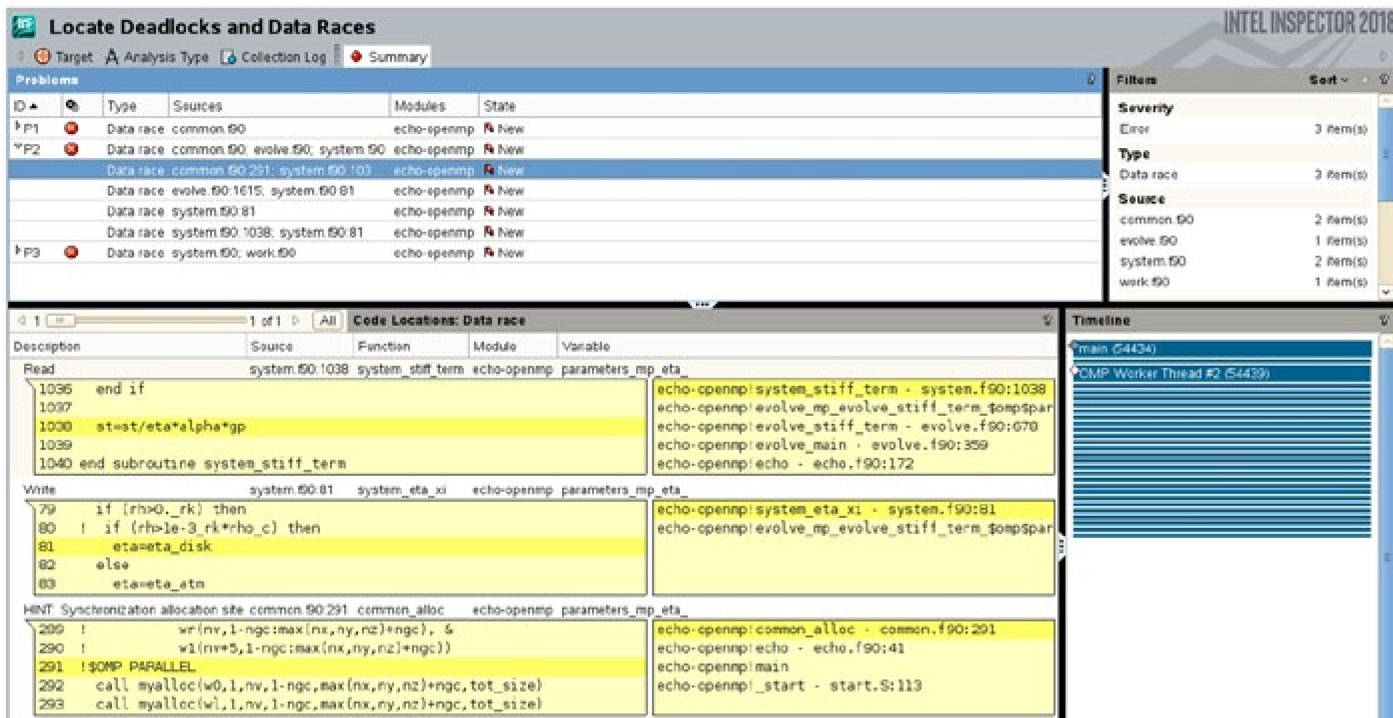
4 **インテル® VTune™ Amplifier で ECHO-3DHPC ベースライン・バージョンの HPC performance Characterization (HPC パフォーマンス特性) 解析を実行して表示された Memory Usage viewpoint (メモリー使用ビューポイント) のスクリーンショット**

上記のパフォーマンスに関する考慮事項に加えて、使いやすいメモリーとスレッドのデバッガーであるインテル® Inspector で (最近コードに追加された) OpenMP* 実装の正当性を調査すると良いでしょう。特別なコンパイラーやビルド構成は必要ありません。通常のデバッグまたはプロダクション・ビルドを使用できます。ただし、このような解析は、インストルメンテーション・オーバーヘッドを伴うため、アプリケーションの実行時間に影響します。スレッドエラーを調査して修正に取り組みには、次のコマンドを実行します。

```
inspxe-cl -collect=ti3 -- <実行ファイルと引数>
```

解析結果はいくつかのデータ競合を示しています (**図 5**)。これは、複数のスレッドが同時に同じメモリー位置にアクセスしているためで、非決定的な動作になる可能性があります。

インテル® Advisor
現代のハードウェア向けにコードを最適化
詳細



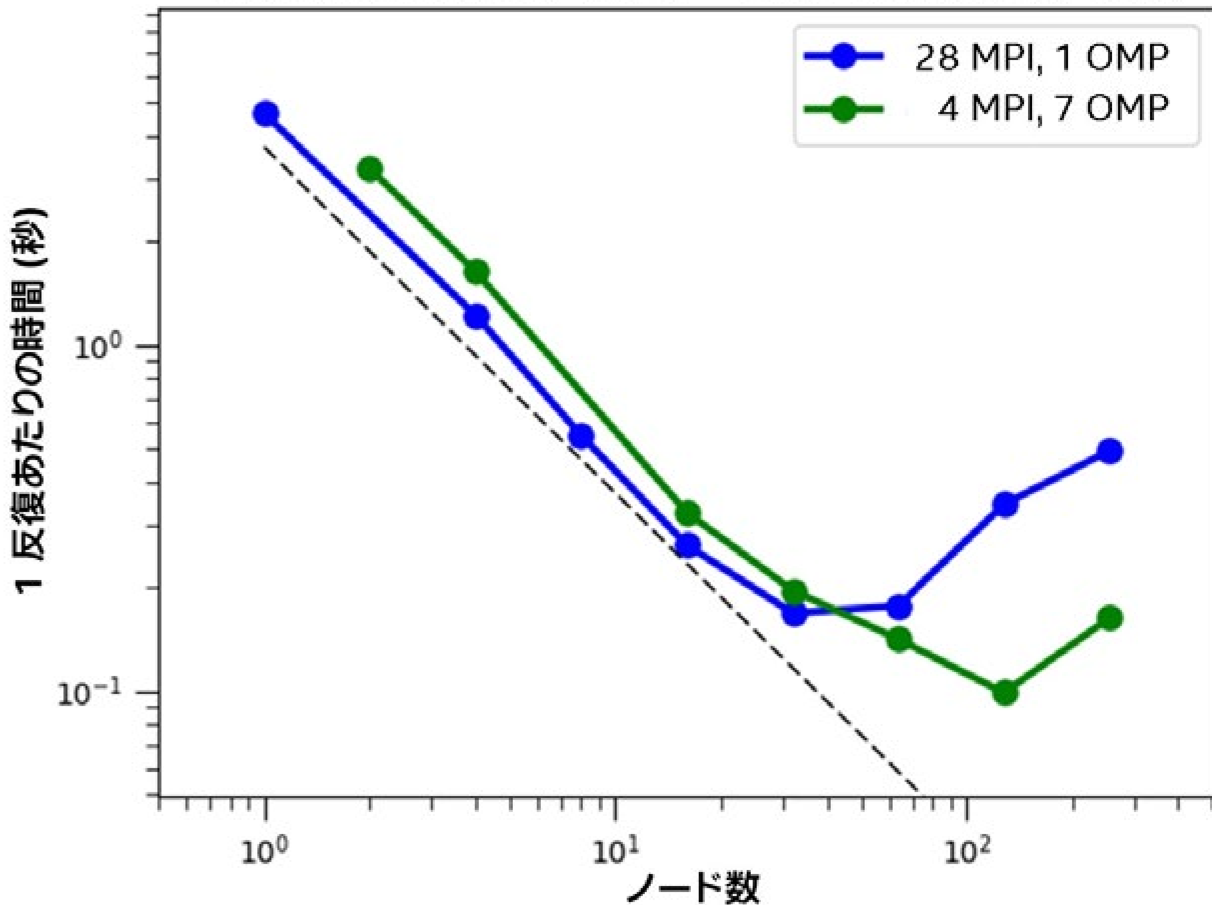
5 **インテル® Inspector で ECHO-3DHPC ベースライン・バージョンの Locate Deadlocks and Data Races (デッドロックとデータ競合の特定) 解析を実行した結果のスクリーンショット**

ここまでの解析結果を基に、OpenMP* 並列領域の数を増やし、スケジューリングを SCHEDULE(GUIDED) に変更し、OpenMP* 並列ループで一部の変数に PRIVATE 節を適用して、インテル® Inspector により検出されたデータ競合を排除しました。これらの最適化は、次の効果をもたらしました。

- **28 スレッドでは、最適化したコードはベースライン・バージョンよりもパフォーマンスが 2.3 倍向上し (図 3)、5.5 倍の並列スピードアップを達成しました。**これは、インテル® Advisor で測定された 15% のシーケンシャル実行を含む並列アプリケーションに対するアムダールの法則の予測とも一致します。
- **スピン時間は 1/1.7 に**減少しました。
- コードはまだメモリー依存ですが、ストールしたパイプライン・スロットの割合がやや減り 68% になりました。

MPI 通信ボトルネックへの対応

OpenMP* による最適化の効果は、大規模な MPI/OpenMP* ハイブリッド実行を解析することでも確認できます。**図 6** は、MPI のみのコードと MPI/OpenMP* ハイブリッド・コードのスケーラビリティーを反復あたりの時間で比較しています (値が小さいほうが良い)。この図は、計算ノードごとに 4 つの MPI タスク、タスクごとに 7 つの OpenMP* スレッドの最適なハイブリッド構成の結果を示しています。計算ノードが少ない場合、MPI のみのコードのほうが優れたパフォーマンスを発揮します。一方、ノード数が多い場合、MPI 通信オーバーヘッドによりスケーラビリティーが低下します。MPI タスクごとの問題サイズは、通信時間を相殺できるほど大きくありません (通信が計算に影響します)。最高のパフォーマンス (スケーリング曲線の最も低い位置にあるポイント) が**図 6** では右に



6 **ECHO-3DHPC の MPI のみのバージョン (青色の線) と MPI/OpenMP* ハイブリッド・バージョン (緑色の線) のスケーラビリティー。点線は理想的な強力なスケーリングを示しています。各ノードには 28 コアあります。**

移動しており、より大規模な HPC システムを効率良く利用できます。最高のパフォーマンスを達成した実行を比較すると、MPI/OpenMP* ハイブリッド構成では MPI 通信時間が 1/2 に減少しています。追加の最適化により、ノードごとの OpenMP* スレッド数をより多くすることができ、通信オーバーヘッドをさらに軽減できます。

実行時間を短縮するためハードウェアを効率良く使用

相対論的プラズマのモデリングに使用される天体物理アプリケーションの ECHO-3DHPC に最近実装された並列処理スキームにより、ハードウェアを効率良く使用し、実行時間を短縮できます。新しいバージョンは、最適化された MPI/OpenMP* ハイブリッド並列アルゴリズムを採用しており、65,000 を超えるコアに対して優れたスケーリングを示しています。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

参考文献

1. Del Zanna et al. (2007). "ECHO: A Eulerian Conservative High-Order Scheme for General Relativistic Magnetohydrodynamics and Magnetodynamics," Astronomy & Physics, 473, 11-30.
2. Bugli et al. (2018). "Papaloizou-Pringle Instability Suppression by the Magnetorotational Instability in Relativistic Accretion Discs," Monthly Notices of the Royal Astronomical Society, 475, 108-120.

技術ウェビナー

スキルアップ、エキスパートからの回答、
新しい開発分野への参入

オンデマンド	**PYTHON* アプリケーションのスピードアップとコア計算の高速化** データ・サイエンティストの皆さん、Python* は言語 "X" よりも遅いと聞いたことはありませんか？もう違います。 インテル® Distribution for Python* は、そのまま利用するだけでネイティブコードに近いパフォーマンスを実現できます。
オンデマンド	**プラットフォームとアプリケーションの高速化に特化したコードアナリスト** インテル® VTune™ Amplifier のプラットフォーム・プロファイラーを利用して、システム全体のワークロードを素早く解析して、最適化すべき場所をピンポイントで特定できます。 無料のテクニカルプレビューをダウンロード。

今すぐ登録 (英語) >

コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (software.intel.com/en-us/articles/optimization-notice#opt-jp) を参照してください。
Intel、インテル、Intel ロゴ、VTune は、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。
* 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。
© Intel Corporation.

システム・パフォーマンスを理解するためのガイド

インテル® VTune™ Amplifier のプラットフォーム・プロファイラー

Bhanu Shankar インテル コーポレーション パフォーマンス・ツール・アーキテクト
Munara Tolubaeva 同ソフトウェア・テクニカル・コンサルティング・エンジニア

アプリケーションの実行全体を通して、システムがどれぐらい効率良く利用されているか疑問に思ったことはありませんか？ 不適切なシステム構成によりパフォーマンスが低下していないか、あるいは、さらに重要なこととして、コードのパフォーマンスを最大限に引き出すためにはどのように再構成すべきか考えたことはありませんか？ 長時間の実行のパフォーマンス・データを収集できる最先端のパフォーマンス解析ツールは、必ずしも詳細なパフォーマンス・メトリックを提供しません。一方、短時間の実行に適したパフォーマンス解析ツールは、膨大な量のデータを提供することがあります。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

この記事では、**インテル® VTune™ Amplifier** のプラットフォーム・プロファイラーを紹介します。プラットフォーム・プロファイラーは、パフォーマンスを低下させるシステム構成の問題や、パフォーマンスのボトルネックとなる特定のシステム·コンポーネントへの負荷に関するデータを提供します。システムまたはハードウェアの観点からパフォーマンスを解析して、過小または過度に使用されているリソースを特定します。プラットフォーム・プロファイラーは段階的開示方式を採用しているため、大量の情報に惑わされることはありません。つまり、開発環境や運用環境で長時間または常時実行しているワークロードをモニタリングしたり、解析することができます。

プラットフォーム・プロファイラーを使用して、次の操作を行うことができます。

- 共通のシステム構成の問題を**特定**
- プラットフォームのパフォーマンスを**解析**して、パフォーマンスのボトルネックを特定

第 1 に、プラットフォーム・プロファイラーが提供するプラットフォーム構成図は、システム構成を確認し、潜在的な問題を特定するのに役立ちます。第 2 に、次のようなシステム・パフォーマンス・メトリックを取得できます。

- CPU とメモリーの使用状況
- メモリーとソケット・インターコネクトの帯域幅
- 命令ごとのサイクル数
- キャッシュミス比率
- 実行された命令タイプ
- ストレージ・デバイス・アクセスのメトリック

これらのメトリックは、過小または過度に使用されているシステムまたは CPU、メモリー、ストレージ、ネットワークなどのプラットフォーム・コンポーネントを特定し、全体のパフォーマンスを向上するためそれらのコンポーネントをアップグレードまたは再構成する必要があるか判断できるように、システム全体のデータを提供します。

プラットフォーム・プロファイラーの使用

実際に、オープンソースの **HPC Challenge (HPCC) ベンチマーク・スイート** (英語) を実行して収集した解析結果を使用して、テストシステムの使用状況を確認してみましょう。HPCC は、次のパフォーマンスを測定する 7 つのテストで構成されています。

- 浮動小数点 (FP) 演算の実行
- メモリーアクセス
- ネットワーク通信操作

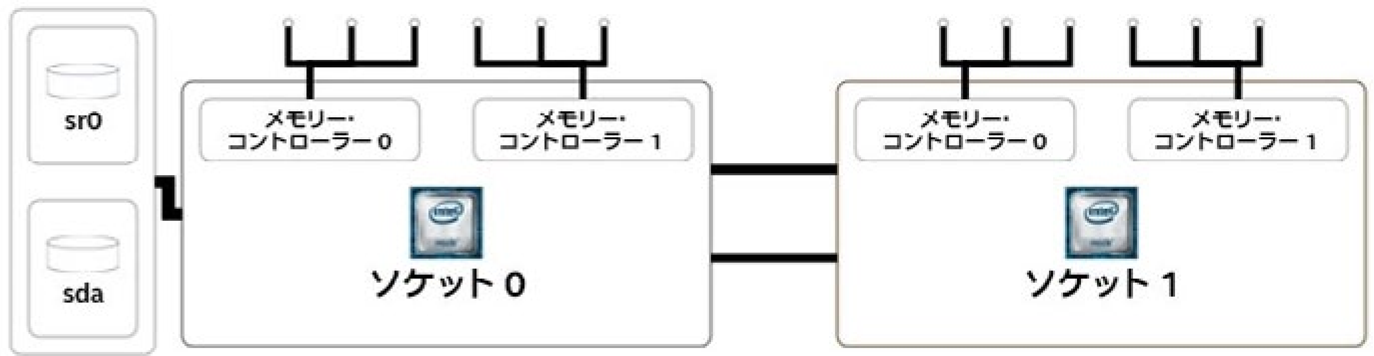
図 1 は、テストを実行したマシンのシステム構成図です。2 ソケットのマシンで、それぞれのソケットには **インテル® Xeon® Platinum 8168 プロセッサー**、2 つのメモリー・コントローラー、6 つのメモリーチャネルがあり、ソケット 0 に 2 つのストレージデバイスが接続されています。

図 2 は、CPU 使用率メトリックと CPU の作業量を測定する命令ごとのサイクル数 (CPI) メトリックです。**図 3** は、メモリー、ソケット・インターコネクト、I/O 帯域幅のメトリックです。**図 4** は、各コアで使用されたロード、ストア、分岐、FP 命令の比率です。**図 5** と **6** は、各メモリーチャネルのメモリー帯域幅とレイテンシーのグラフです。**図 7** は、すべての命令に対する分岐と FP 命令の比率です。**図 8** は、命令ごとの L1 と L2 キャッシュミス比率、**図 9** は、メモリー使用量のグラフです。平均して、実行全体で 51% のメモリーしか使用されていません。より大きなテストケースでは、メモリー使用量を向上できる可能性があります。

図 5 と **6** から、6 チャネルのうち 2 チャネルしか使用されていないことが分かります。テストシステムのメモリー DIMM 構成に問題があり、メモリーチャネルを最大限に利用できず、HPCC パフォーマンスが低下していることは明らかです。

図中の CPI (**図 2**)、DDR メモリー帯域幅の使用率、命令ミックスは、HPCC 実行中の特定の時点で実行されたテスト (計算や FP 演算、あるいはメモリー操作) を示しています。例えば、実行の 80-130 秒と 200-260 秒の間は、メモリー帯域幅の利用率と CPI レートがともに上昇しており、この間に HPCC 内のメモリーベースのテストが実行されたことを確認できます。さらに、**図 7** の命令ミックスは、スレッドが 280-410 秒の間に FP 命令を実行したことを示し、**図 3** は 275-360 秒の間にいくつかのメモリーアクセスを行ったことを示しています。これらのことから、この間に計算とメモリー操作の組み合わせを使用するテストが実行されたことが分かります。また、コードのベクトル化により FP 操作の実行を最適化することで、テストの計算部分のパフォーマンスを向上できる可能性があります。

インテル® VTune™ Amplifier
現代のプロセッサーのパフォーマンス解析
無料評価版のダウンロード

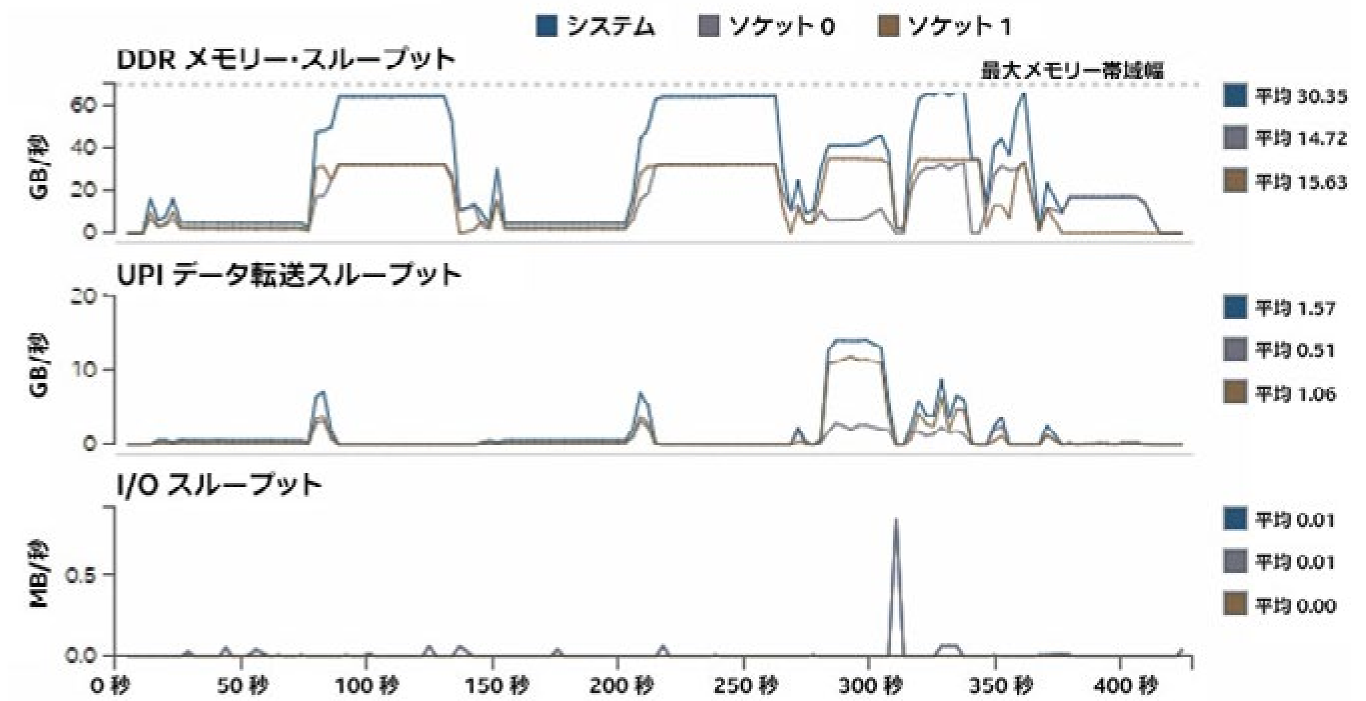


1 システム構成図

CPU メトリック

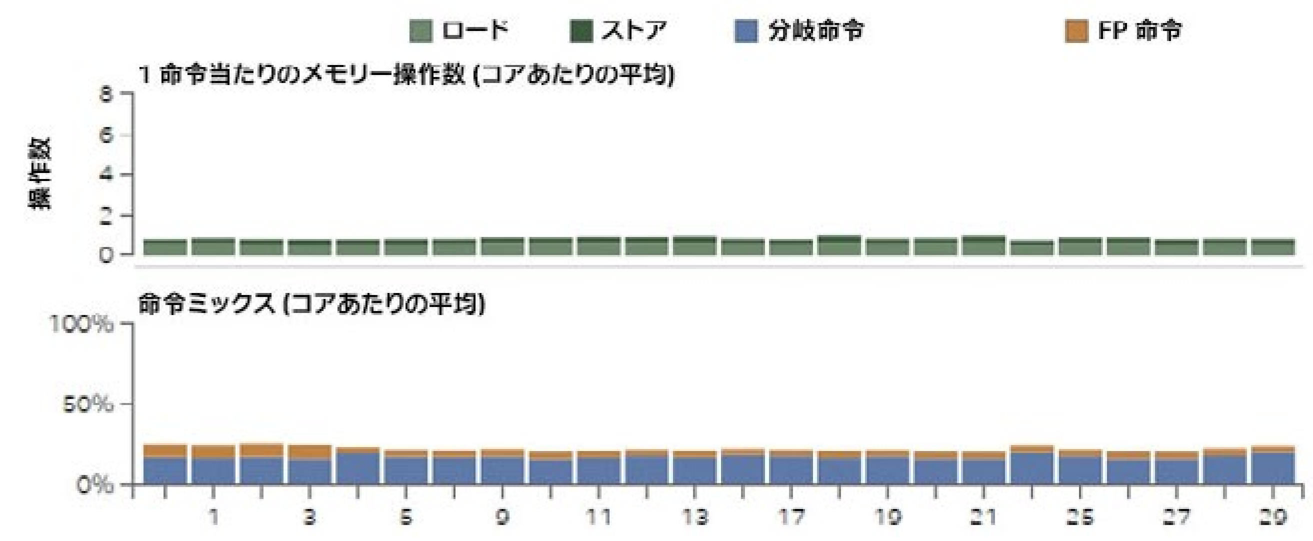
2 CPU 使用率メトリック

スループット・メトリック



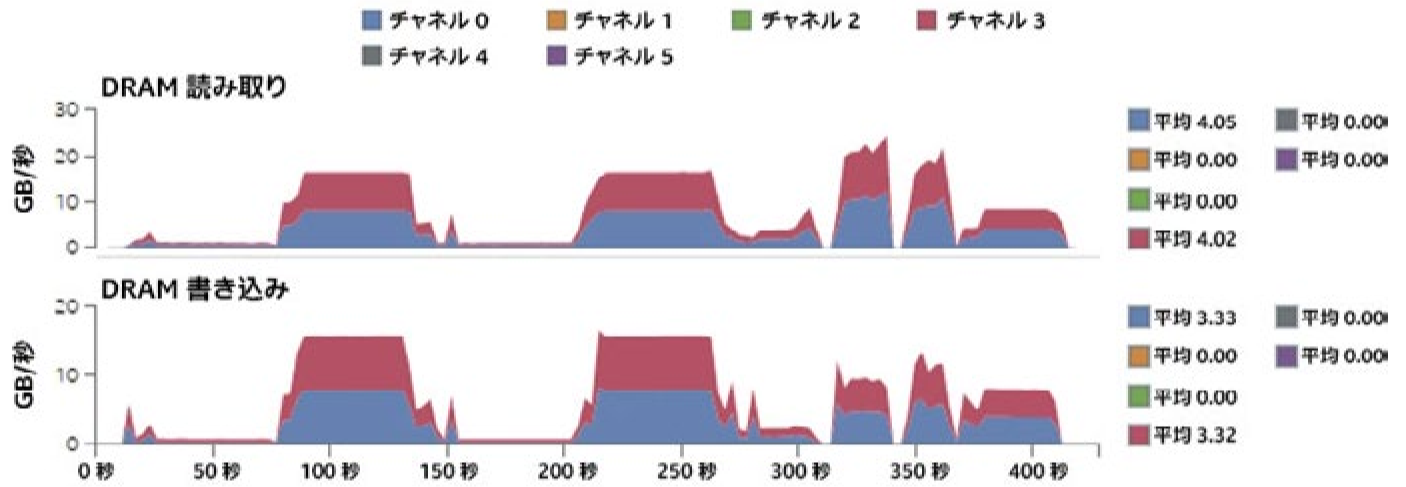
3 メモリー、UPI、I/O のスループット・メトリック

操作メトリック



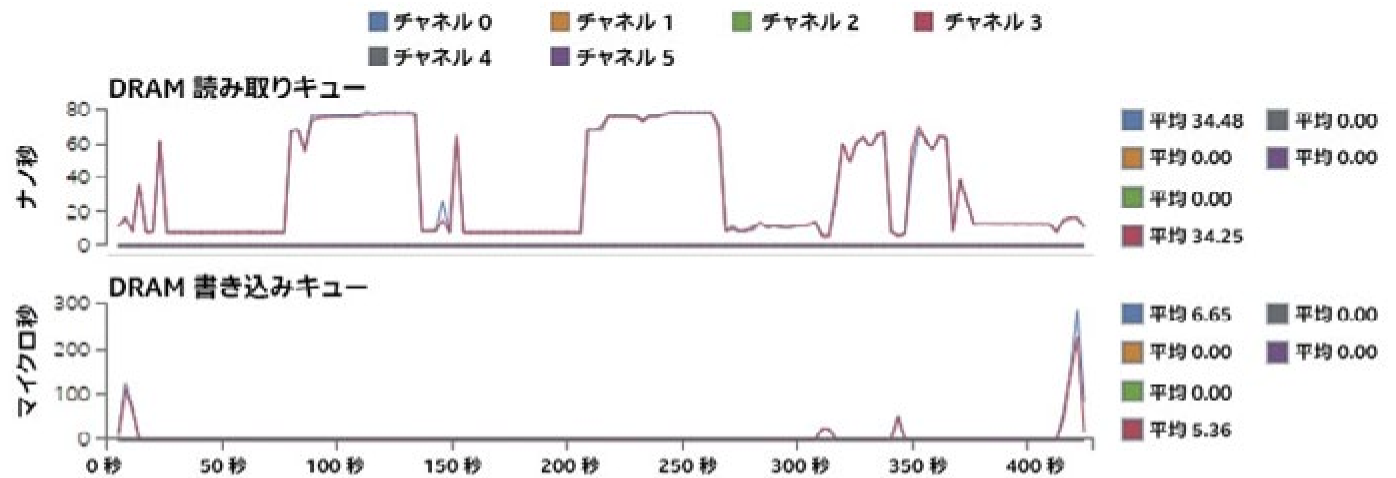
4 スループット・プログラムの実行に使用された命令タイプ

メモリー・スループット



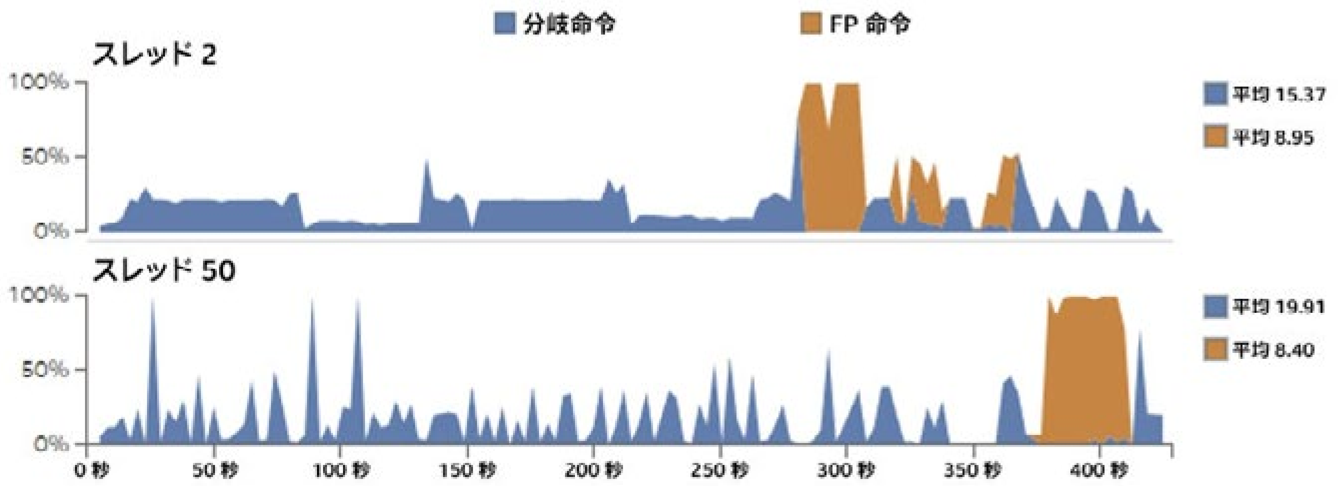
5 メモリー・チャネル・レベルのメモリー帯域幅グラフ

メモリー・レイテンシー



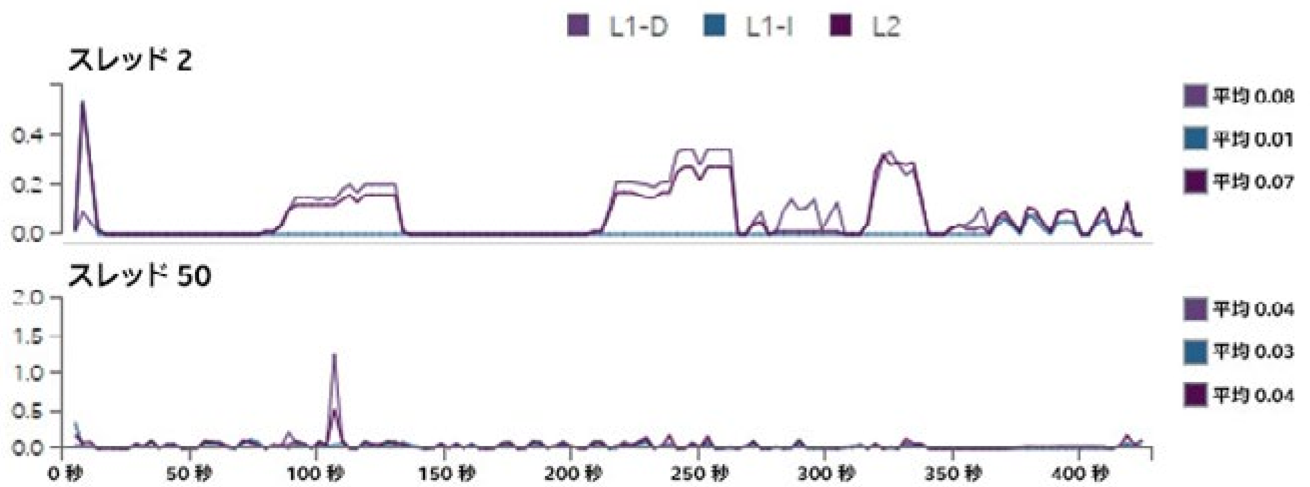
6 メモリー・チャネル・レベルのメモリー・レイテンシー・グラフ

命令ミックス



7 **すべての命令に対する分岐と浮動小数点命令の比率**

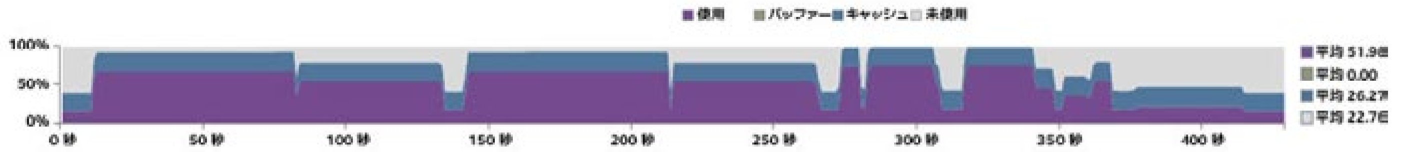
1 命令あたりの L1 および L2 ミス



8 **1 命令あたりの L1 および L2 ミス比率**

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

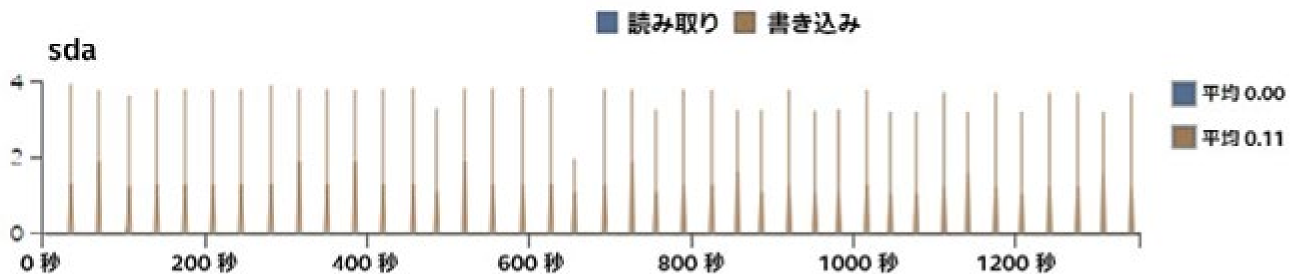
メモリー使用量



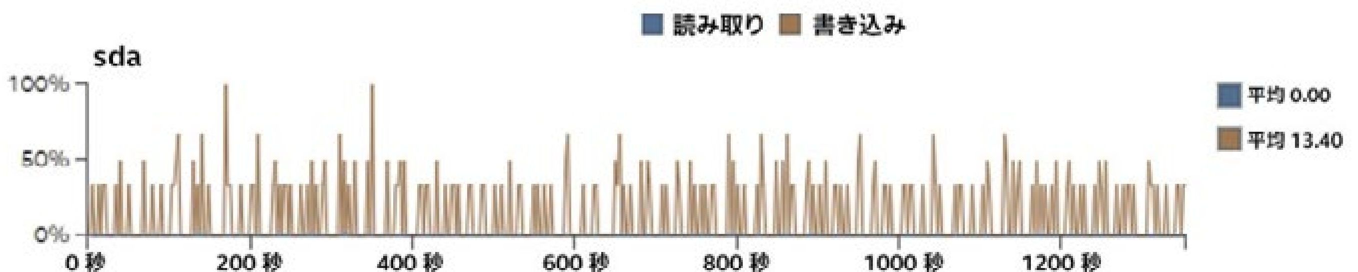
9 メモリー消費

HPCC は I/O を含むテストを実行しないため、2 つ目のテストとして LSTC により開発された独自のマルチフィジックス・シミュレーション・ソフトウェアである **LS-Dyna*** (英語) のディスクアクセスについて、プラットフォーム・プロファイラーで解析した結果を示します。**図 10** は、LS-Dyna* のディスク I/O スループットを示し、**図 11** は、1 秒あたりの I/O とレイテンシー・メトリックを示します。LS-Dyna* の暗黙モデルは、定期的にデータをディスクにフラッシュするため、I/O スループット・グラフには定期的な山形が見られます (**図 10** の読み取り / 書き込みスループットを参照)。書き込むデータは大きくないため、実行全体を通して I/O レイテンシーは一定です (**図 11** の読み取り / 書き込みレイテンシーを参照)。

読み取り / 書き込みスループット

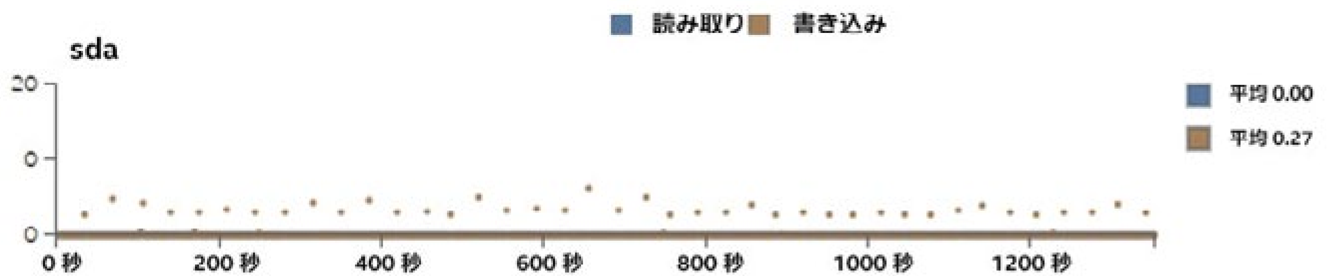


読み取り / 書き込み操作のミックス

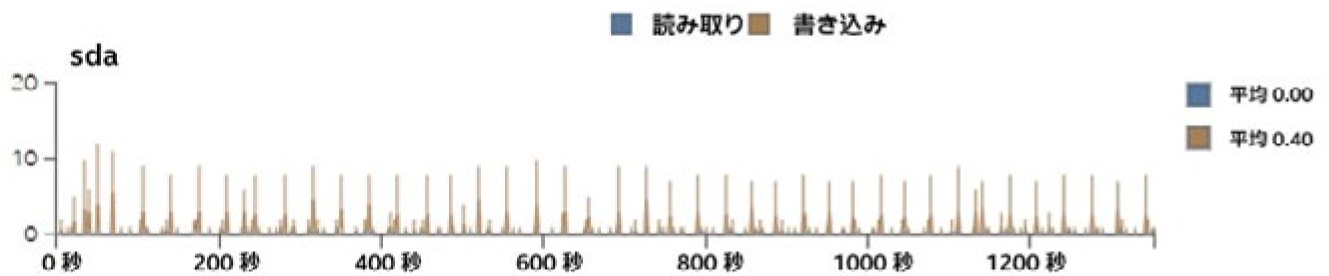


10 LS-Dyna* のディスク I/O スループット

読み取り / 書き込みレイテンシー



IOPS



11 **LS-Dyna* の IOPS とレイテンシー・メトリック**

システム・パフォーマンスの考察

この記事では、システムやハードウェアの観点からパフォーマンスを解析するツール、プラットフォーム・プロファイラーを紹介しました。プラットフォーム・プロファイラーは、システムのボトルネックに関する情報を提供し、過小または過度に使用されているサブシステムやプラットフォームレベルのインバランスを特定します。また、ここではその使用法と、HPCC ベンチマーク・スイートと LS-Dyna* アプリケーションで収集した結果も示しました。ツールを使用することで、メモリー帯域幅を制限しているメモリー DIMM の問題を見つけることができました。また、テストには FP 実行が集中している部分があり、コードのベクトル化によりパフォーマンスを最適化できることが分かりました。全体的に、この HPCC と LS-Dyna* のテストケースでは、テストシステムに過度の負荷がかかることはなく、システムリソースにはまだ余裕がありました。これは、次回はさらに大きなテストを実行できることを意味します。

Software

THE PARALLEL UNIVERSE

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。§ さらに詳しい情報をお知りになりたい場合は、https://www.intel.com/benchmarks (英語) を参照してください。

パフォーマンス結果は 2018 年 10 月 1 日現在の測定値であり、すべての公開済みセキュリティー・アップデートが適用されていない可能性があります。詳細は、システム構成を参照してください。絶対的なセキュリティーを提供できる製品はありません。

インテル® ソフトウェア開発製品のパフォーマンスおよび最適化に関する詳細は、最適化に関する注意事項 (https://software.intel.com/articles/optimization-notice#opt-jp) を参照してください。

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。実際の性能はシステム構成によって異なります。絶対的なセキュリティーを提供できるコンピューター・システムはありません。詳細については、各システムメーカーまたは販売店にお問い合わせください。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。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

本資料には、開発中の製品、サービスおよびプロセスについての情報が含まれています。本資料に含まれる情報は予告なく変更されることがあります。最新の予測、スケジュール、仕様、ロードマップについては、インテルの担当者までお問い合わせください。

本資料で説明されている製品およびサービスには、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。

本資料で紹介されている資料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、www.intel.com/design/literature.htm (英語) を参照してください。

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Arria、Intel Core、Cyclone、Quartus、Stratix、Movidius、Xeon、Intel Xeon Phi、OpenVINO、VTune は、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

OpenCL および OpenCL ロゴは、Apple Inc. の商標であり、Khronos の使用許諾を受けて使用しています。

* 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。

JPN/1811/PDF/XL/CVCG/SS