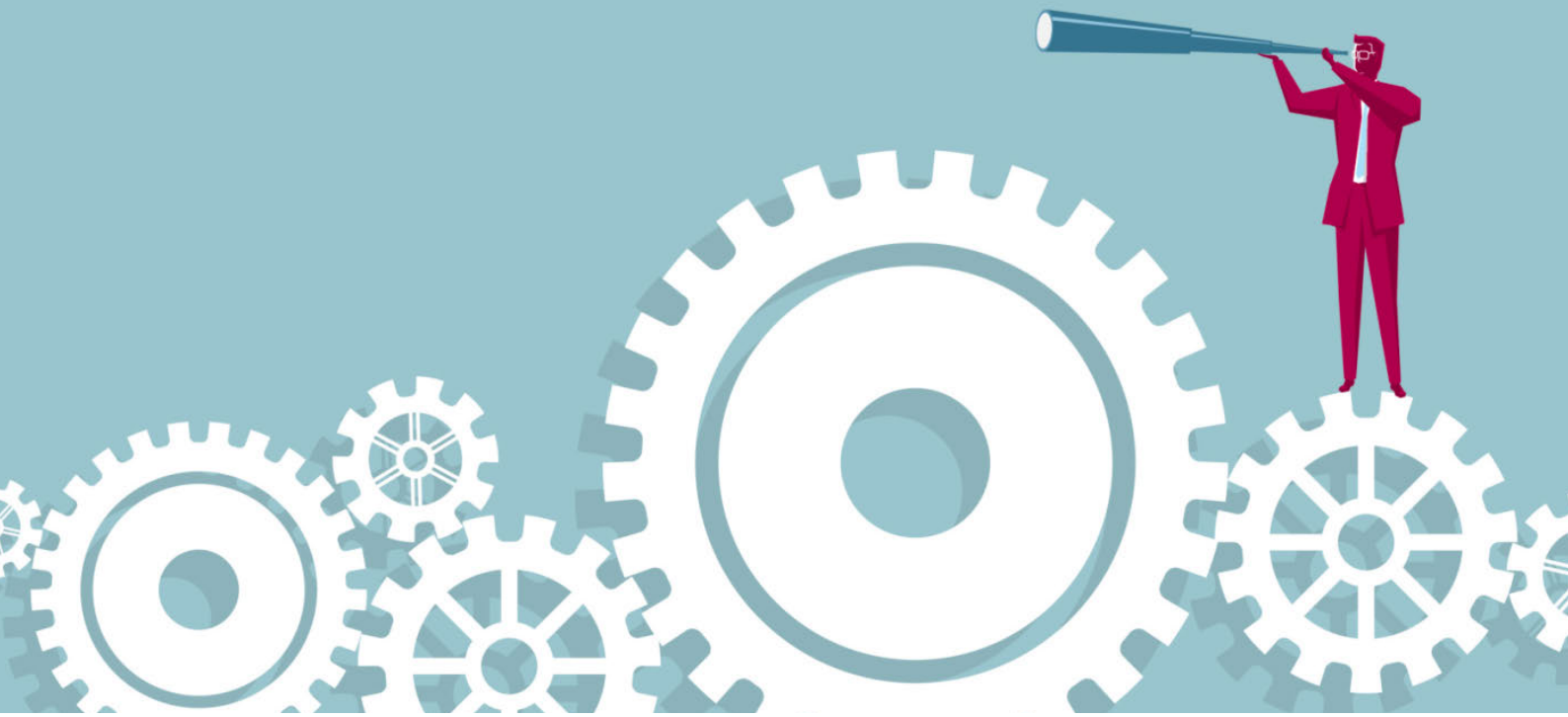
THE PARALLEL UNIVERSE

intel Software



パフォーマンスを向上する最適なツールの選択

コードの現代化

機械学習計算プラットフォームの最適化

Fortran サブモジュールによる生産性の向上

Issue 23 2015

目次

編集者からのメッセージ

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

新しい書籍『High Performance Parallel Programming Pearls Volume Two』の共著者で、このほかに、『High Performance Parallel Programming Pearls (Volume One)』(2014)、『Multithreading for Visual Effects』(2014)、『Intel® Xeon Phi™ Coprocessor High Performance Programming』(2013、日本語:『インテル® Xeon Phi™ コプロセッサー ハイパフォーマンス・プログラミング』)、『Structured Parallel Programming』(2012、日本語:『構造化並列プログラミング』)、『Intel® Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』(2007、日本語:『インテル スレッディング・ビルディング・ブロック — マルチコア時代の C++ 並列プログラミング』、中国語、韓国語の翻訳版があります)、『VTune™ Performance Analyzer Essentials』(2005) などの文献を発表しています。

人間の「思考」に近づくコンピューター

並列プログラミングの普及とともに、人間にとって普通のことはコンピューターにとっても普通のことになりつつあります。人間なら、たとえ小さな子供でも、写真の猫を指さすように言われたときに戸惑うことはないでしょう。現在では、そのような動作を行うコンピューター・プログラムを記述することも一般的になり、コンピューターはより人間に近づいています。

中国のインターネット・セキュリティー企業 Qihoo* は、機械学習関連の計算モデルをサポートする分散型オフライン計算プラットフォームを最適化する際に、そのような能力を効率的に実装する方法を発見しました。64 ページから始まるケーススタディーでは、インテル® C++ コンパイラーとインテル® マス・カーネル・ライブラリー (インテル® MKL) を利用することで計算速度が 5 倍に向上しました。

コンピューターは並列処理を行ってもまだ人間の思考能力にかないませんが、コンピューターの飛躍的な進歩により、その差は徐々に縮まっています。「ベクトル化アドバイザーのヒントの活用」 (41 ページ) で説明するように、インテル® Advisor XE のようなツールを利用することにより、Hartree Centre のような企業は、最近のインテル® プラットフォームのベクトル並列化機能を活用して、アプリケーションの速度を向上することができます。

もちろん、現時点では、コンピューターはまだ人間を必要としています。世界最高のコンピューターでも、人間の助けがなければ役に立ちません。この号の注目記事「最適なツールの選択 - アプリケーションのパフォーマンスを向上するためのロードマップ」で説明するように、インテル® ソフトウェア・ツールとテクノロジーを使用することにより、望みの結果を得ることができます。その先の道は、人間の独創性によって切り開かれるでしょう。

James Reinders
2015 年 11 月

最適なツールの選択

アプリケーションのパフォーマンスを向上するためのロードマップ

Kevin O'Leary 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

現在および将来のハードウェアを活用するには、ソフトウェアの現代化が必要です。アプリケーションのスレッド化とベクトル化を始めとするプロセスを支援する**ソフトウェア開発ツール**およびライブラリーを使用することで、この現代化を最も適切に行うことができます。

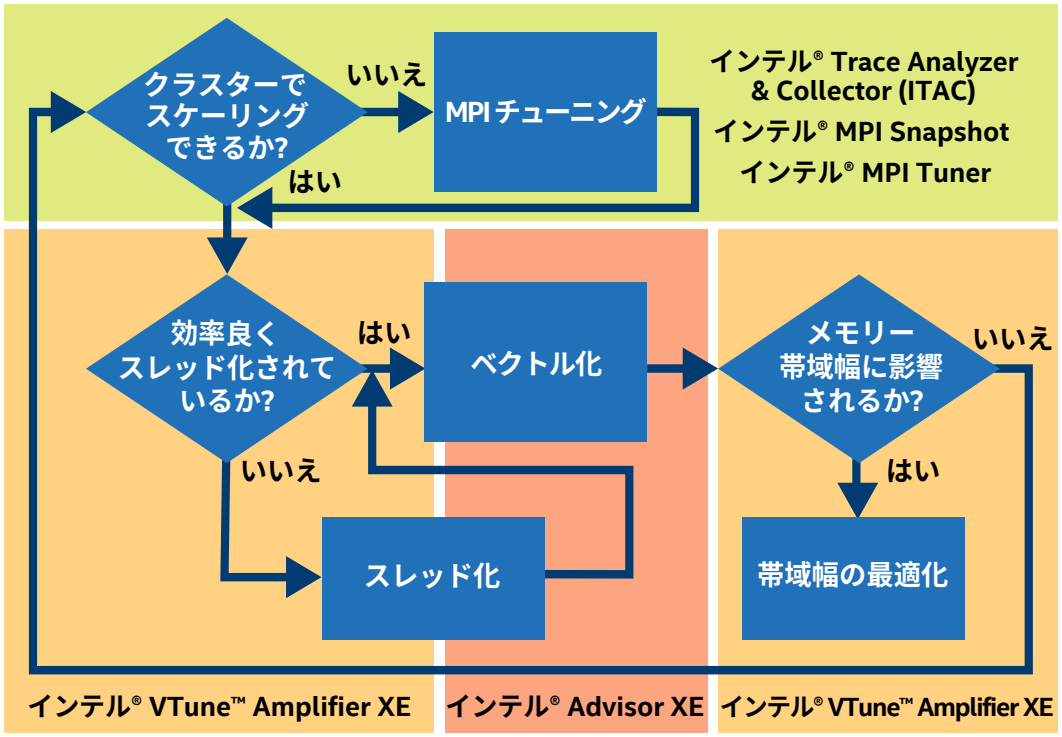
では、どこから始めればよいのでしょうか。

この記事では、ハードウェアから最高のパフォーマンスを引き出すための手法を推奨します。アプリケーションのスレッド化とベクトル化に加えて、メモリー階層を最大限に活用することも重要です。そうしないと、スレッド化やベクトル化によるメモリー・トラフィックの増加がボトルネックになってしまいます。また、MPI を使用してノード間の並列処理をチューニングする方法についても説明します。

パフォーマンス・チューニングの各段階で正しいツールを使用することで、より低いコストでパフォーマンスを大幅に向上することができます。

はじめに

最初の質問は、「問題をクラスターでスケーリングできるか？」です。



1 パフォーマンス解析の診断項目と利用するツール

答えが「いいえ」の場合は、**MPI** チューニングを行います (**図 1**)。この記事の最後に、MPI チューニングで役立つ次の 3 つのツールを説明します。

- インテル® Trace Analyzer & Collector
- MPI Performance Snapshot
- MPI Tuner

MPI チューニングの後は、次の順に 3 段階のパフォーマンス最適化を行います。

- スレッド化
- **ベクトル化**
- メモリー帯域幅

スレッド化

最新のプロセッサーには多くのコアが搭載されています。このハードウェアをすべて活用するには、コードをスレッド化する必要があります。コードをスレッド化する最適な方法は、OpenMP*、インテル® スレッディング・ビルディング・ブロック (インテル® TBB)、インテル® Cilk™ Plus のようなタスク抽象化を利用することです。そうすることで、プロセッサーのコア数に合わせて自動的にスケーリングするコードを作成できます。

スレッド化のアプローチを決定したら、スレッド化したアプリケーションの有効性を解析することが重要です。ここでは次のツールを使用します。

- インテル® VTune™ Amplifier XE - パフォーマンス・プロファイリング
- インテル® Advisor XE - スレッドのプロトタイプ作成
- インテル® Inspector XE - スレッド化とメモリーの正当性検証

ベクトル化

最近のプロセッサーはコア数が増加しているだけでなく、ベクトルレジスターの幅も増加しています。そのため、コードをさらに並列化できる可能性があります。ベクトル化解析には、インテル® Advisor XE を使用します。

メモリー帯域幅

最後に、メモリー使用量がアプリケーション・パフォーマンスの最適化にとって重要な理由を説明します。メモリー帯域幅解析には、インテル® VTune™ Amplifier XE を使用します。

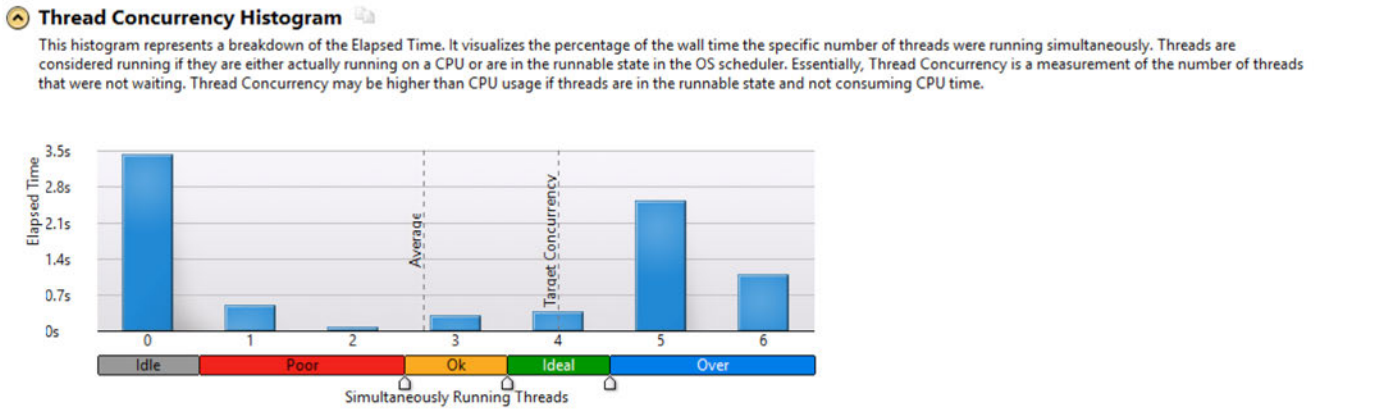
それでは、最初に、ノードレベルの最適化 (**図 1** の下半分) から行っていきましょう。

ステップ 1: パフォーマンス・プロファイラーとスレッドのプロトタイプ作成ツールを使用して効率良くスレッド化する

「アプリケーションは効率良くスレッド化されていますか?」この質問に答えるには、インテル® VTune™ Amplifier XE で解析後、インテル® Advisor XE を使用してスレッドのプロトタイプ作成を行います。スレッド化とメモリー割り当ての正当性検証には、インテル® Inspector XE が役立ちます。

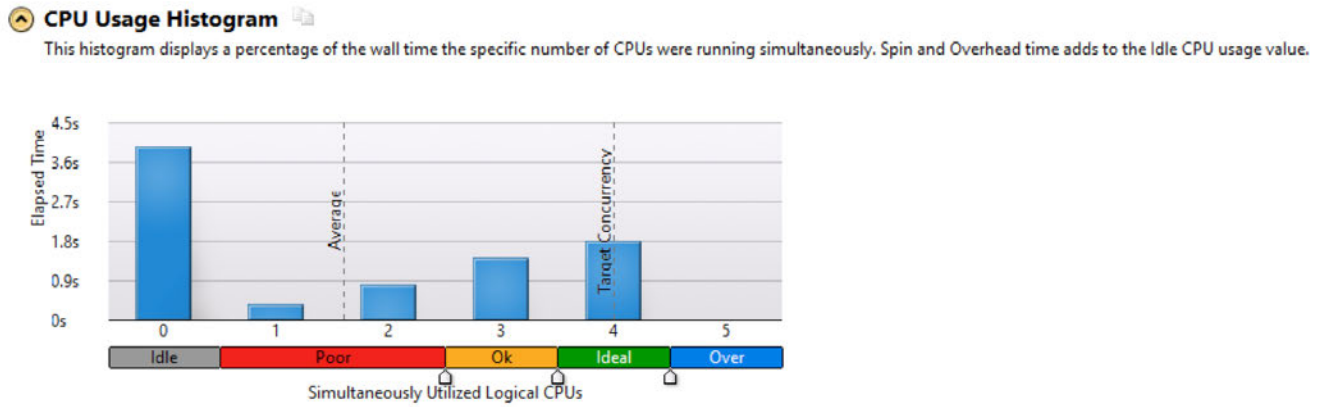
CPU 使用率とスレッド・コンカレンシーの解析

アプリケーションは、マイクロアーキテクチャーではなく、アプリケーションのスレッド化効率や CPU の使用状況によって制限されることがあります。アプリケーションで同時に実行しているスレッドの数は、インテル® VTune™ Amplifier XE のロックと待機 (Lock & Wait) 解析タイプを使用して確認できます (**図 2**)。



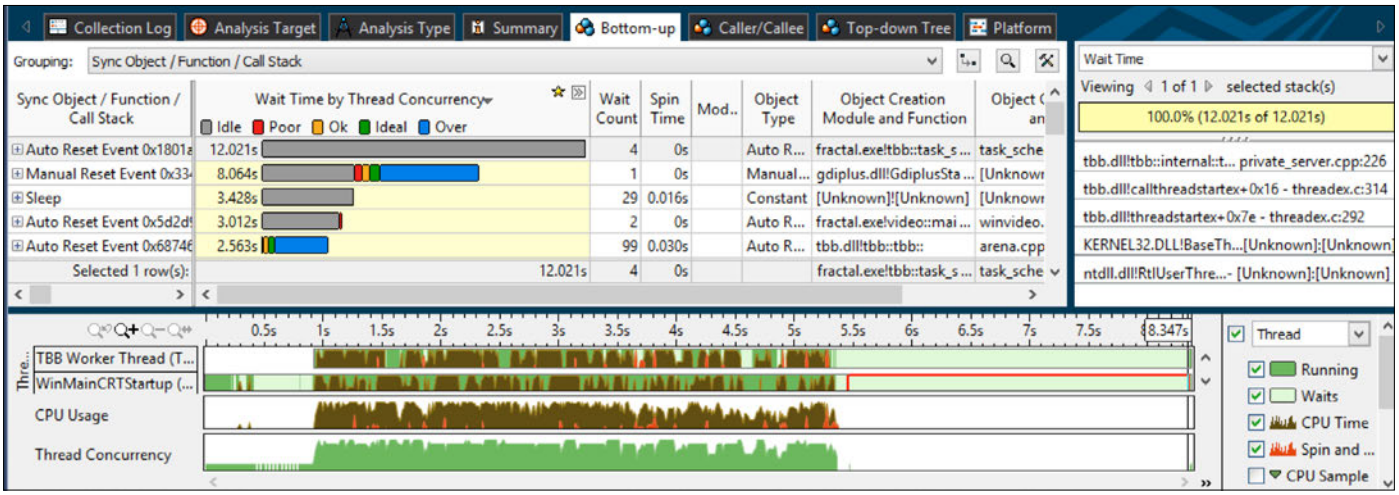
2 インテル® VTune™ Amplifier XE のスレッド・コンカレンシー・ヒストグラム

インテル® VTune™ Amplifier XE で CPU の使用率を確認することもできます (**図 3**)。



3 インテル® VTune™ Amplifier XE の CPU 使用率ヒストグラム

アプリケーションの同期回数と待機中のアイドル時間を確認して、使用率を解析します。**図 4** は、上位の同期ポイント、そのポイントが呼び出された回数、そのポイントで費やされた時間を示しています。



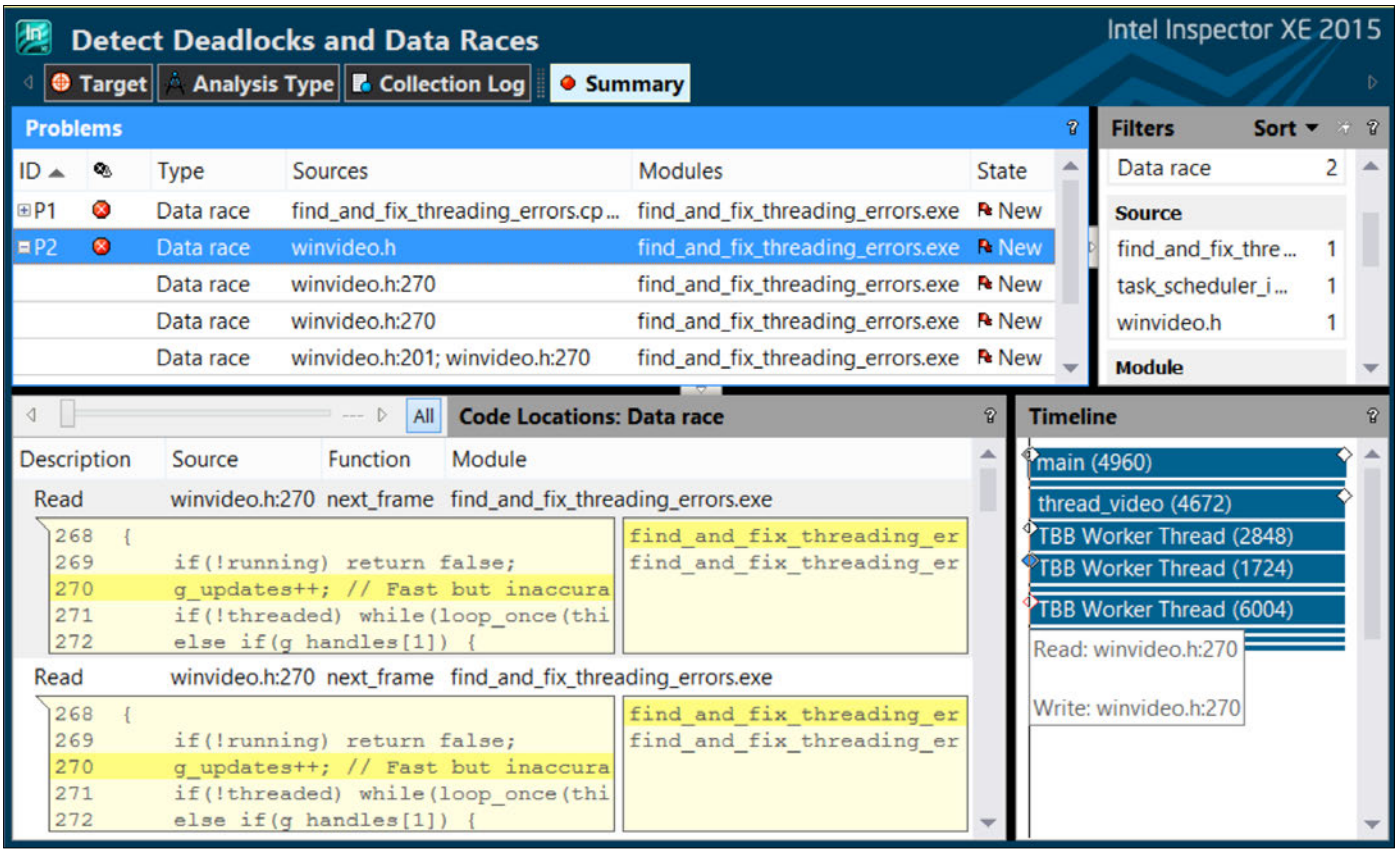
4 インテル® VTune™ Amplifier XE の [Bottom-up (ボトムアップ)] ウィンドウ

目標は、同時に実行するスレッド数を利用可能なコア数に近づけて CPU を効率良く利用することです。待機中に費やされるアイドル時間は、理想的な使用率の達成を妨げます。インテル® VTune™ Amplifier XE のようなツールを使用することにより、同期の場所と回数、待機中の経過時間を調査することができます。

スレッド化とメモリー割り当ての正当性解析

コードをスレッド化する際に、データ競合のようなスレッド化の問題が潜在的に発生することがあります。これらの問題は追跡およびデバッグが非常に困難です。アプリケーションを解析して潜在的なデータ競合を検出できるツールへの投資は、非常に高い効果が得られます。

インテル® Inspector XE は、コードに潜在的なデータ競合が含まれているかどうか動的に確認します (**図 5**)。同期プリミティブ関数の呼び出しは必要か？コードのより小さなセクションで同期できるか？インテル® Inspector XE は、アプリケーションのすべてのスレッドとメモリー参照を解析するため、必要なときにのみ同期が行われているか検証できます。これにより、待機時間を減らして CPU 使用率を向上することで、アプリケーションの高速化が可能です。

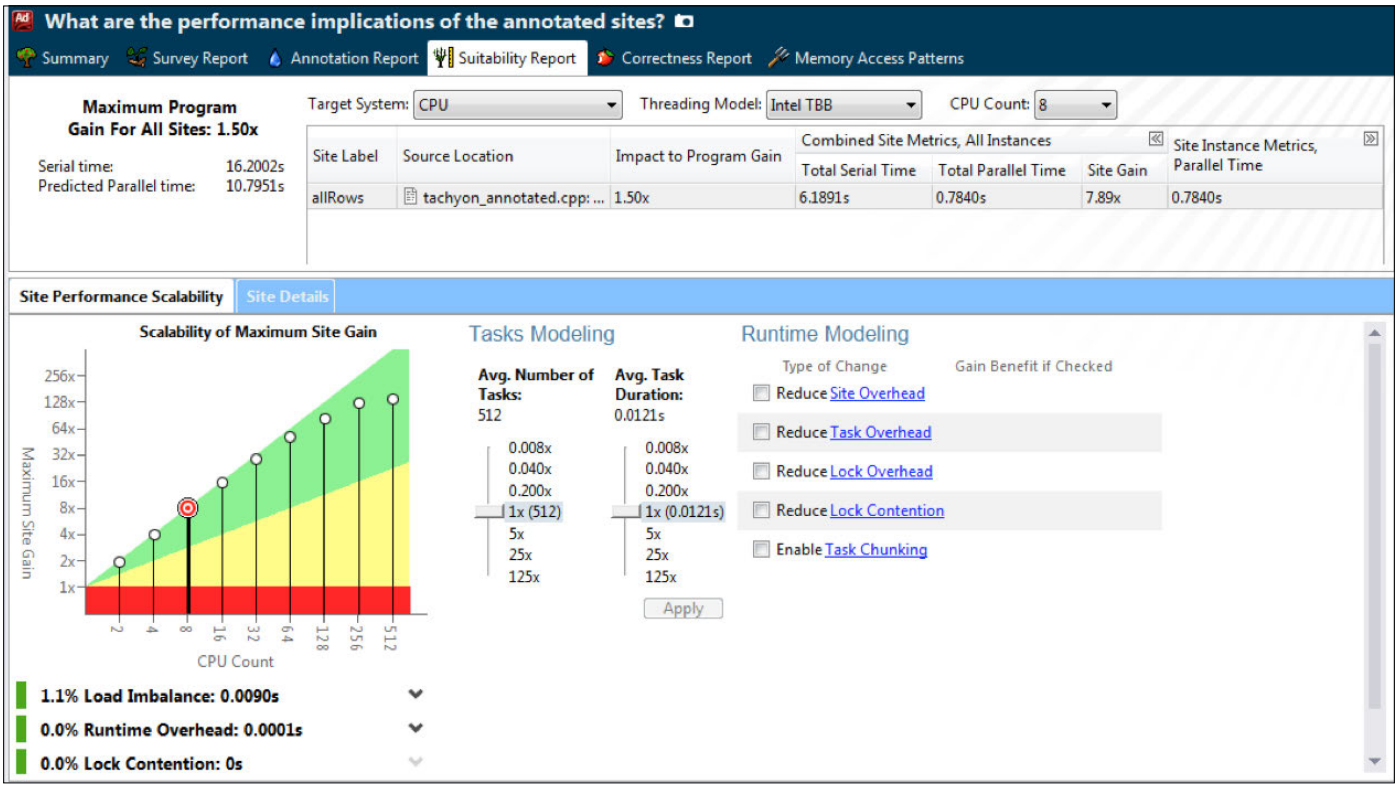


5 インテル® Inspector XE の [Summary (サマリー)] ウィンドウ

このようにツールを使用することで、製品の開発中に問題を発見することができます。そのため、製品リリース後に修正するよりも、問題の修正にかかるコストが大幅に少なくなります。

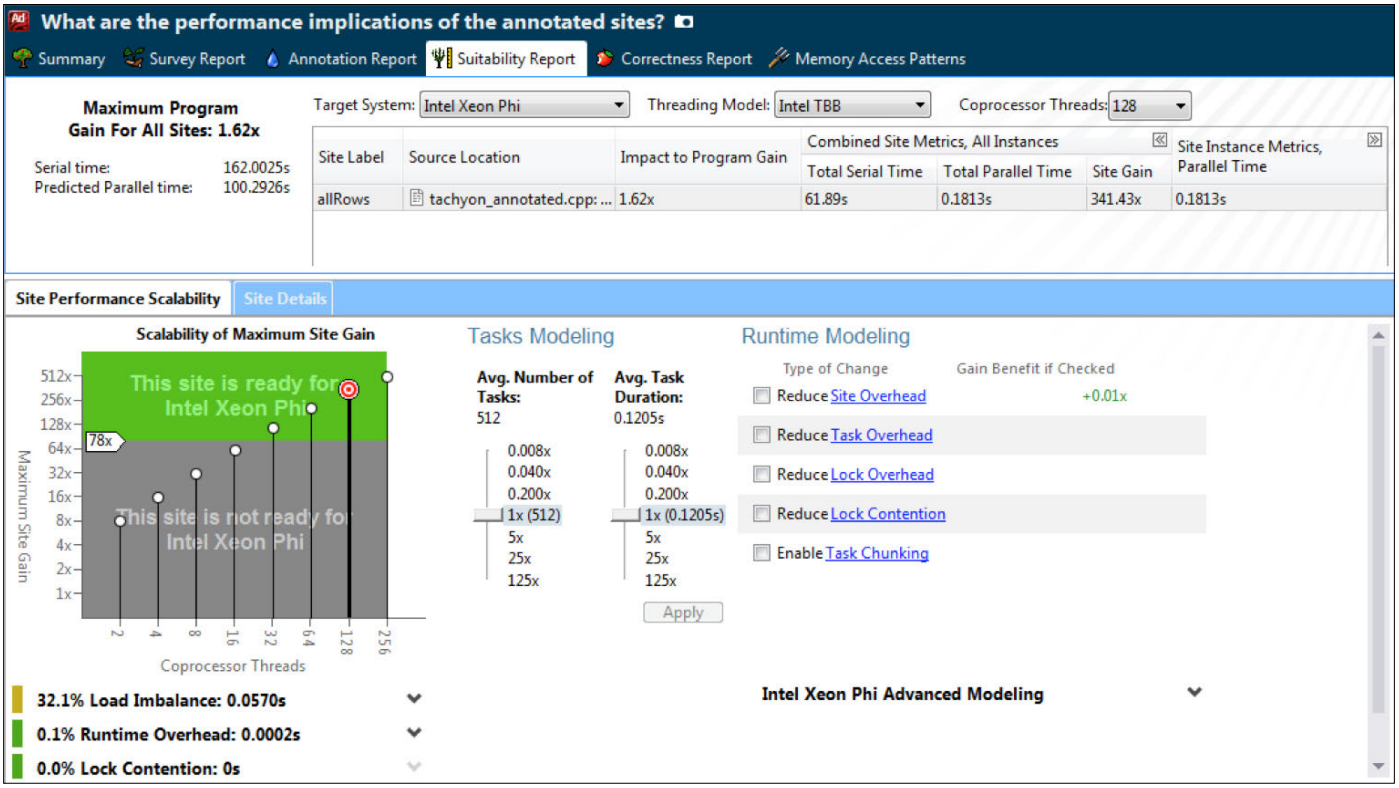
設計ツールを使用して効率良くスレッド化する

インテル® Advisor XE は、アプリケーションを解析して非効率なスレッド化が行われたコード領域を検証します。次に、それらの領域に並列処理を追加してアプリケーションが適切にスケーリングするかテストします。試験的な解析を動的に実行して、入力条件を変更した場合にアプリケーションが適切にスケーリングするか検証することもできます (**図 6**)。



6 インテル® Advisor XE のスケーラビリティーに関する [Suitability Report (適応性レポート)]

プロセッサーを効率良く使用するため、高度な並列性を備えたアプリケーションをターゲット・プロセッサーに対応させる作業を行っている場合、アプリケーションが十分な並列性を備えているかどうかテストすることができます (**図 7**)。



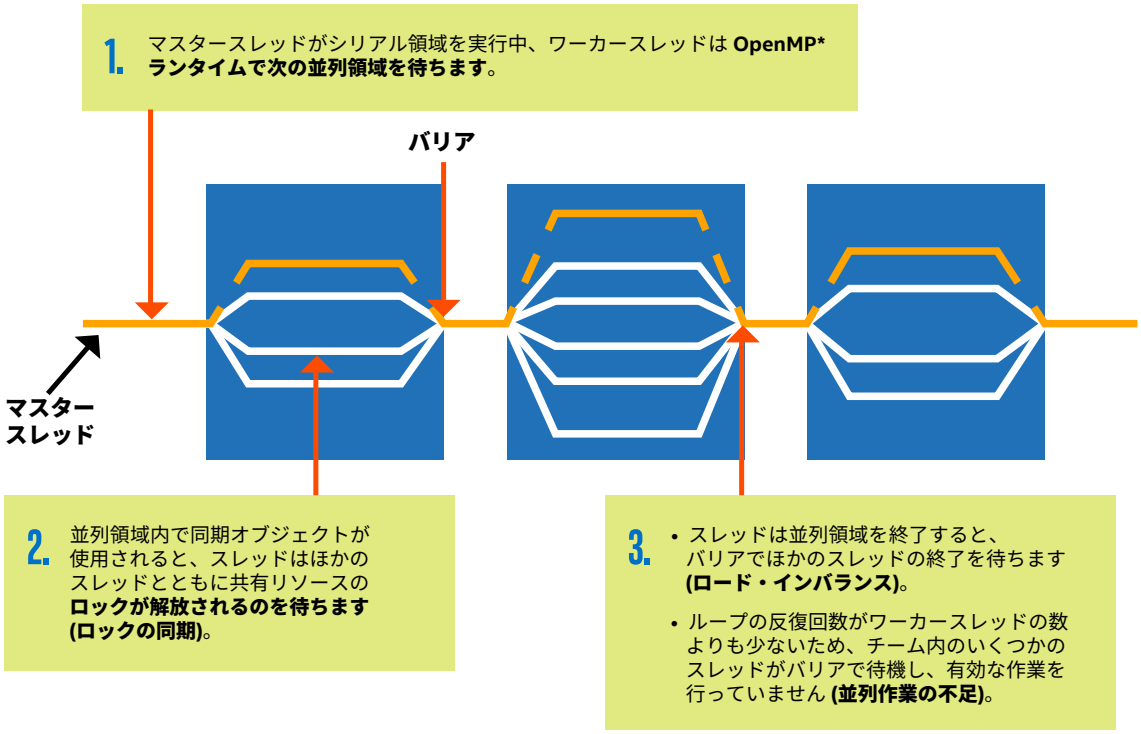
7 インテル® Advisor XE の即応性に関する [Suitability Report (適応性レポート)]

スレッド設計 / プロトタイプ作成ツールは、コードを効率良くスレッド化するのに役立ちます。実装する前に、新しいスレッドが適切にスケーリングするかテストすることができます。また、アプリケーションをチューニングするため、さまざまな選択肢を試すこともできます。

OpenMP* アプリケーションがプロセッサーを適切に利用しているか解析する

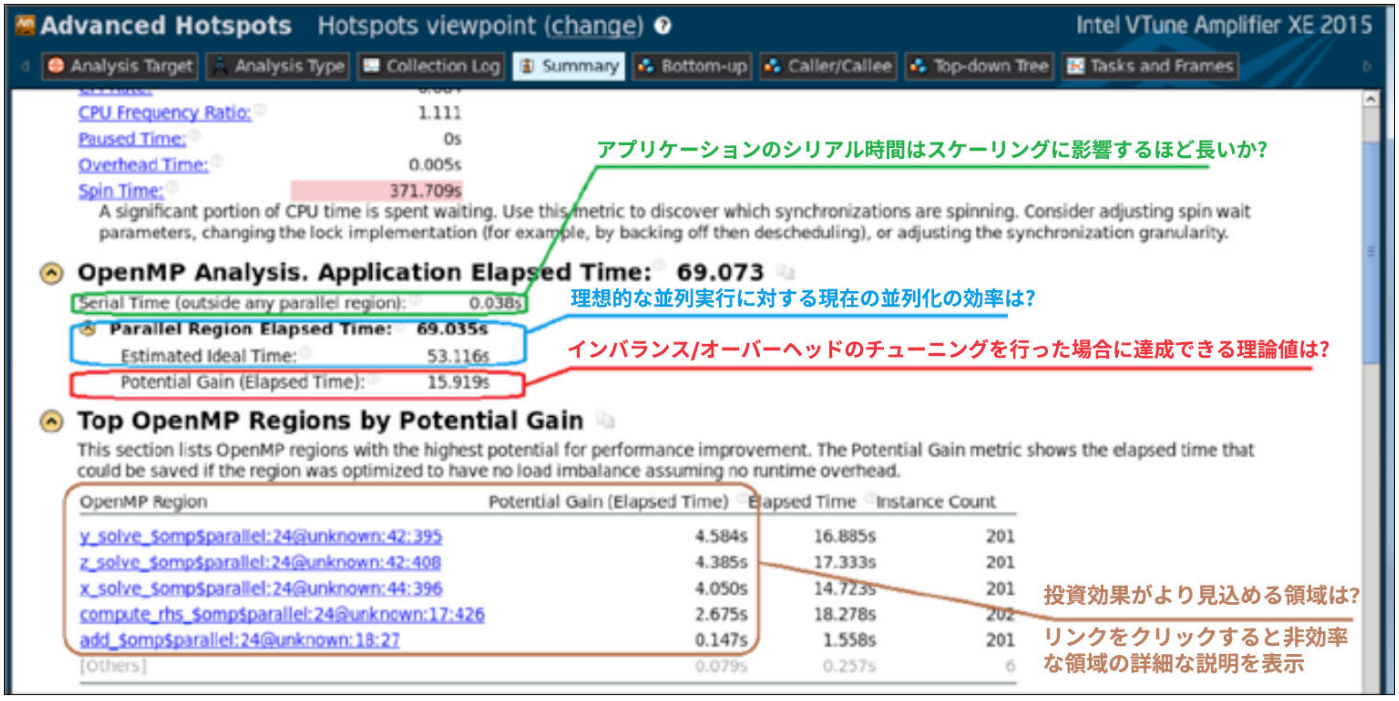
OpenMP* はアプリケーションに並列処理を簡単に追加できる強力な方法です。並列処理を追加した後、OpenMP* を効率的に使用しているかどうか解析することは重要です。**図 8** に示すように、OpenMP* ランタイムはロックと待機を使用してこのセマンティクスを実装します。待機で多くの時間を費やしている場合、パフォーマンスは低くなります。次の点を確認することが重要です。

- アプリケーションのシリアル時間はスケーリングに影響するほど長いか？
- OpenMP* 並列化の効率は？
- ロード・インバランス / オーバーヘッドを減らすとどの程度パフォーマンスを向上できるか？
- どの領域を最適化すべきか？



8 インテル® VTune™ Amplifier XE の OpenMP* 解析

インテル® VTune™ Amplifier XE の OpenMP* 解析で、これらすべてを確認することができます (**図 9**)。



9 インテル® VTune™ Amplifier XE の [Summary (サマリー)] ウィンドウ

アムダールの法則により、並列プログラムのスピードアップはプログラムのシリアル領域で費やしている時間によって制限されます。コードのシリアル領域で多くの時間を費やしている場合、そのアプリケーションはスケーリングに適していません。インテル® VTune™ Amplifier XE は、経過時間を予測 (理想) 時間と比較することにより、OpenMP* 並列化の効率を示します。すべての OpenMP* 領域でこのアプローチを行うことにより、最もパフォーマンス向上率の高い領域を最適化することができます。

OpenMP* は非常に強力なタスク抽象化です。OpenMP* アプリケーションのプロファイルと最適化に時間をかけることにより、CPU のコアを効率的に利用することができます。

ステップ 2: アプリケーションを効率良くベクトル化する

ベクトル化解析

プログラムがどの程度適切にベクトル化されているか、現在の状況を知らせるツールを使用します。現状が把握できれば、より適切にベクトル化が行われるようにプログラムを再設計することができます。コンパイラーは、ループにベクトル依存性が存在していると仮定してループをベクトル化しないことがあります。これらの依存性が実際に発生するかどうか判断できる動的なツールを使用することにより、依存性のないループにはプラグマを追加してベクトル化することができます。また、コンパイラーがベクトル化を行うこともありますが、データ構造がメモリーに適切に配置されない場合、そのベクトル化は非効率的です。メモリー参照を解析することにより、データ構造がベクトル化に適しているかどうか判断することができます。

このプロセスには、インテル® Advisor XE の次の機能を使用します。

- **調査解析。** ベクトル化により最大の効果が得られる場所が分かるように、コンパイラー・レポートとパフォーマンス・データをまとめて表示します。
- **トリップカウント解析。** 各ループの反復回数と呼び出し回数を測定します。
- **推奨事項。** 問題の解決方法について具体的なアドバイスを示します。
- **依存性解析。** ダイナミックな依存性解析を行い、ループ反復間に依存関係がないか確認します。
- **メモリー・アクセス・パターン解析。** アプリケーションがベクトル化に適した方法でメモリーにアクセスしているか確認します。

調査

ベクトル化解析の最初のステップは、アプリケーションの調査です。このステップで、アプリケーションが時間を費やしているループが分かります。「ホットな」ループは、最適化による恩恵が最も得られる場所です。**図 10** は、アプリケーションの調査レポートを示しています。

インテル® Advisor XE では、ループの種類 (ベクトル化されているかどうか) でフィルターすることができます。ベクトル化されなかったループには、ベクトル化を妨げている原因が表示されます。ベクトル化されたループには、ベクトル化の効率に影響を与える可能性に関する詳細情報と次の情報が表示されます。

- **効率と期待値。** ループをベクトル化することにより達成されるスピードアップは？インテル® Advisor XE は、データサイズとベクトル長を計算して理論的に可能なスピードアップを表示します。例えば、ベクトル幅 256 ビットのインテル® AVX2 でアプリケーションがベクトル要素として 32 ビット整数を使用している場合、インテル® Advisor XE は最大のスピードアップを 256/32 = 8 倍と計算します。ループのスピードアップが 8 倍に近ければ、ほぼ最適なスピードアップで、効率はほぼ 100% と言えるでしょう。
- **ベクトル命令セット。** 古い命令セット (インテル® SSE など) を使用していることで、パフォーマンスが損なわれていないか？
- **データ型と特性。** 遅くなる可能性のある命令 (抽出や挿入など) が使用されているか？
- **ベクトル長とベクトル幅。** ベクトルをすべて利用しているか、一部のみ利用しているか？

情報にすばやくアクセスできるように、調査レポートには次のタブが表示されます。

- **Top down (トップダウン)。** 選択したループとアプリケーションの残りの領域の関係を表示します。また、選択したループが内部ループなのか外部ループなのか確認できます。(注 : 一般に、内部ループのみ自動的にベクトル化されます。)
- **Source (ソース)。** アプリケーションのソースをスキャンして、コード内にコンパイラー・レポートを表示します。
- **Loop Assembly (ループ・アセンブリー)。** 生成された命令と実行時間を表示します。
- **Recommendations (推奨事項)。** インテル® Advisor XE が推奨する問題の解決方法を表示します。
- **Compiler Diagnostic Details (コンパイラー診断詳細)。** 問題の解決に役立つ追加情報とサンプルコードを表示します。

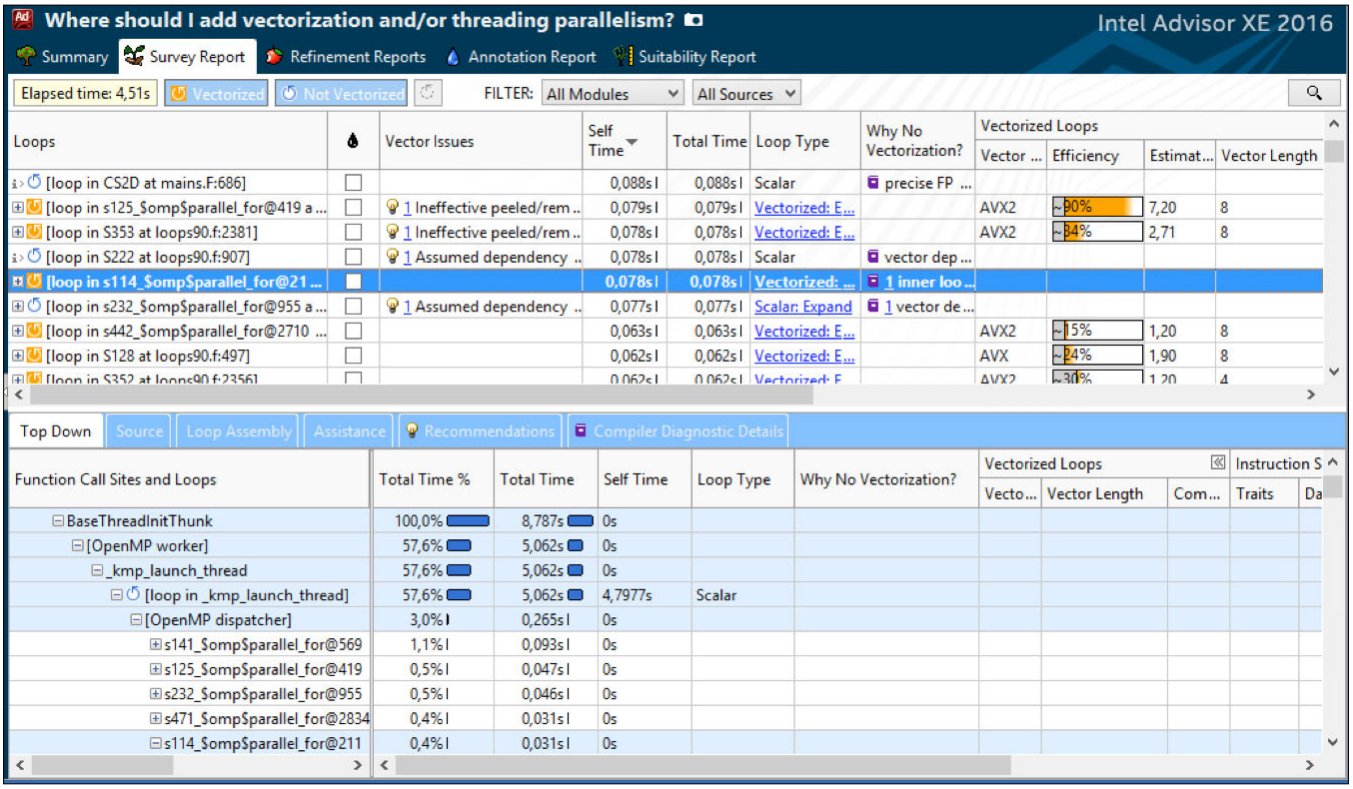
データ構造がメモリー上にどのように配置され、ループでどのようにアクセスされるか知っていれば、ベクトル化の効率を大幅に引き上げることができます。

ループの依存性

正しいコードを生成するため、コンパイラーは、言語のセマンティクスに対して保守的な見地に立たなければいけません。言語の規則に基づいて依存性が想定される場合、コンパイラーは依存性が存在すると仮定します。インテル® Advisor XE のような動的なツールを使用することにより、仮定した依存性が事実かどうか確認することができます。結果は次の 2 つのいずれかです。

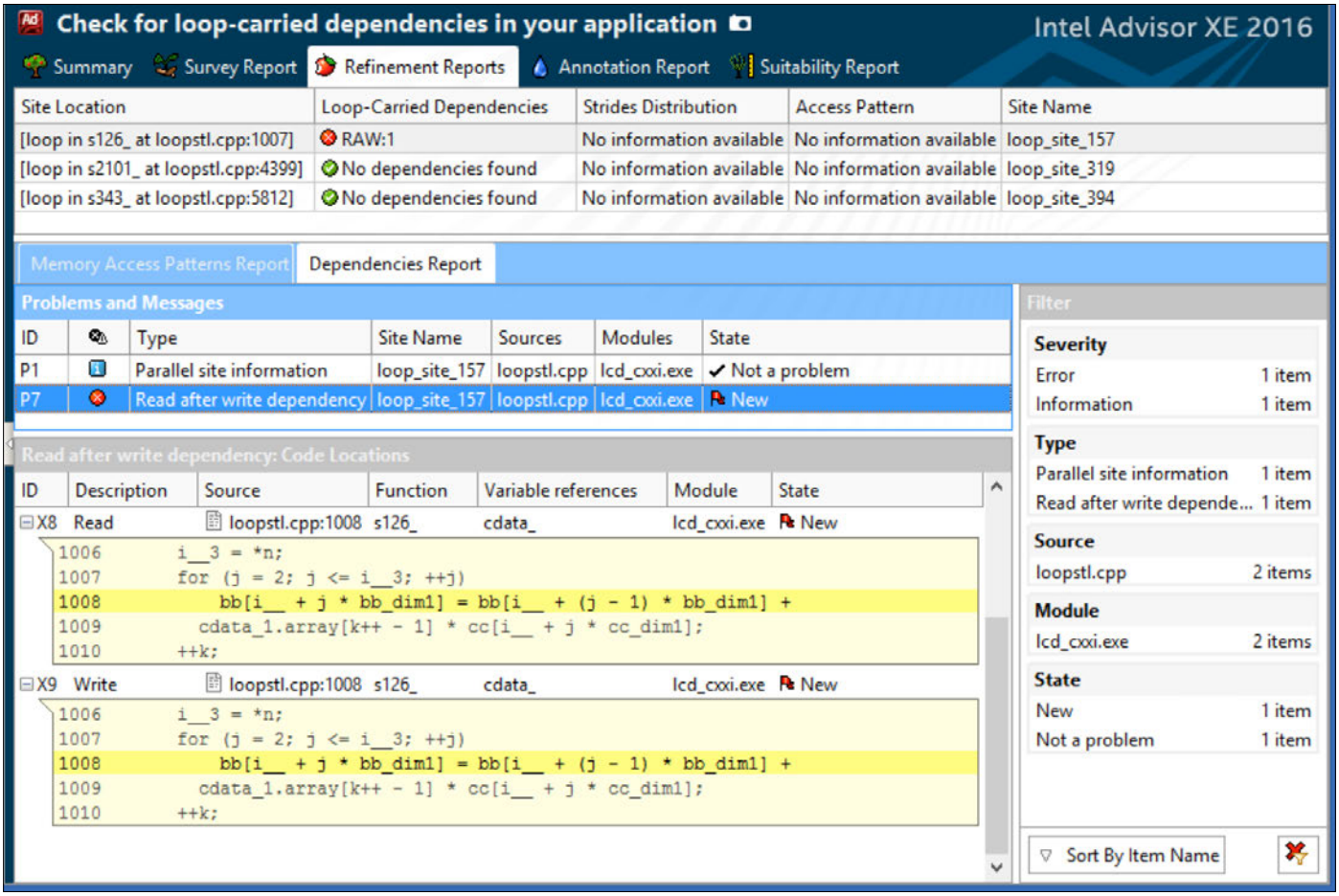
1. **実際の依存性はありません。**この場合、`#pragma omp simd` や `#pragma ivdep` などのディレクティブを使用して、ベクトル化しても安全であることをコンパイラーに伝えます。
2. **依存性が見つかりました。**ループをベクトル化する前に依存性を排除する必要があります。ローカル変数を使用するか、ループやデータ構造などを変更します。

確認するループを選択して、依存性解析を実行します (**図 10**)。



10 インテル® Advisor XE の [Survey Report (調査レポート)] ウィンドウ

図 11 は、4399 行と 5812 行のループに依存性がないことを示しています。そのため、これらのベクトル化を強制するコンパイラー・ディレクティブを使用しても安全です。1007 行のループには依存性があります。インテル® Advisor XE は、正確なソース行と依存性タイプ (書き込みの後の読み取り) を表示します。



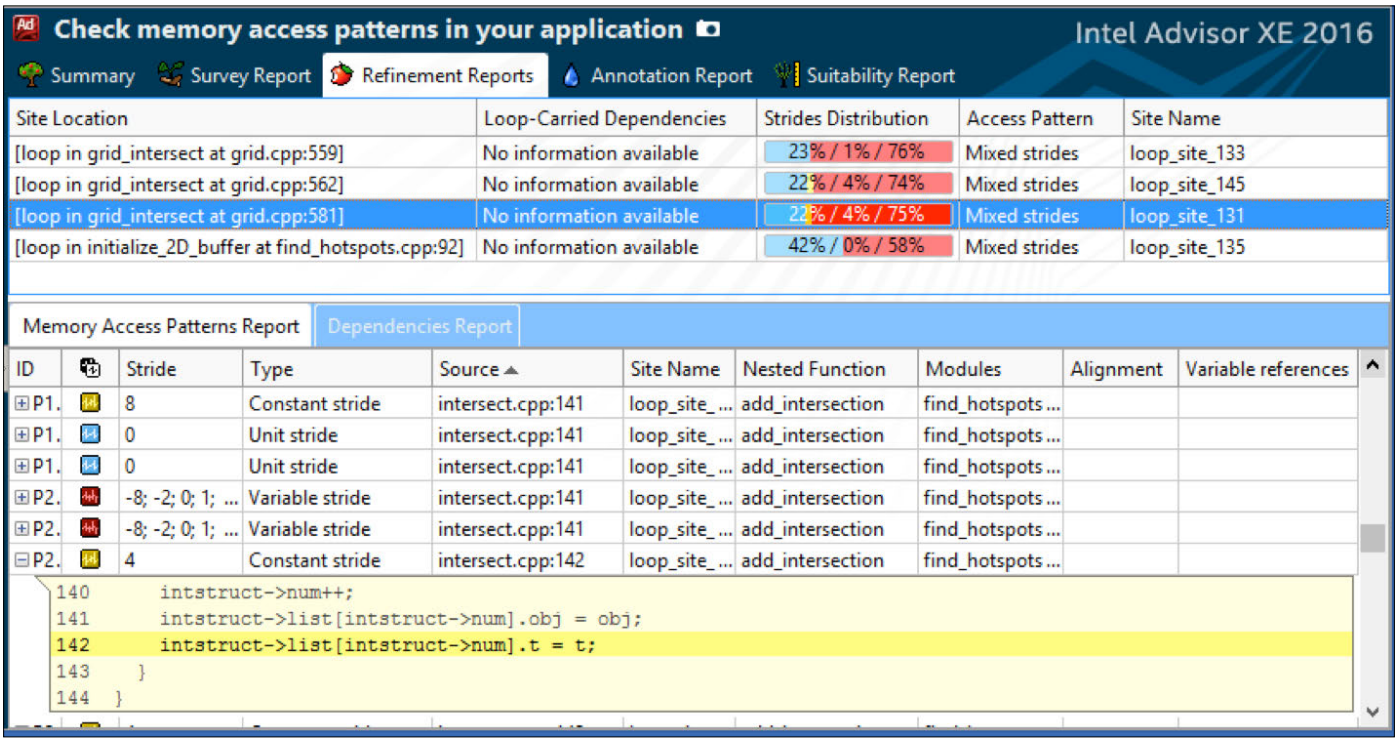
11 インテル® Advisor XE の [Refinement Reports (詳細レポート)] - [Dependencies Report (依存性レポート)] ウィンドウ

データ・アクセス・パターン

データ構造がメモリー上にどのように配置され、ループでどのようにアクセスされるか知っていれば、アプリケーションのベクトル化の効率を大幅に引き上げることができます。メモリー参照がユニットストライド方式でアライメントされていることは非常に重要です。構造体配列 (AoS) から配列構造体 (SoA) へのデータ構造の変換のように、ベクトル化を支援するメモリーアクセスに関連するいくつかの手法があります。メモリー・アクセス・パターン (MAP) 解析を使用すると、非効率的なベクトル化のパターンを見つけることができます。

インテル® Advisor XE の MAP 解析は、メモリーアクセス命令のソースコードとアセンブリー行、データ型とサイズ、アライメントなどを示します。次の 3 種類のメモリーアクセスを識別します (**図 12**)。

1. **ユニット・ストライド・アクセス。**命令は、各反復で 1 つの要素間隔でメモリー位置に連続してアクセスします。
2. **定数ストライド。**命令は、各反復で N 個の要素間隔でメモリー位置に連続してアクセスします (構造体配列など)。
3. **不規則なストライド。**命令は、ギャザー / スキャッター・パターン、配列インデックスなど、各反復で予測できない数の要素間隔でメモリー位置にアクセスします。パフォーマンスは最低です。



12 インテル® Advisor XE の [Refinement Reports (詳細レポート)] - [Memory Access Patterns Report (メモリー・アクセス・パターン・レポート)] ウィンドウ

インテル® Advisor XE のようなツールを使用することにより、ベクトル化の状況を迅速に判断でき、パフォーマンスが最適でない場合はさらなる最適化の目標となります。

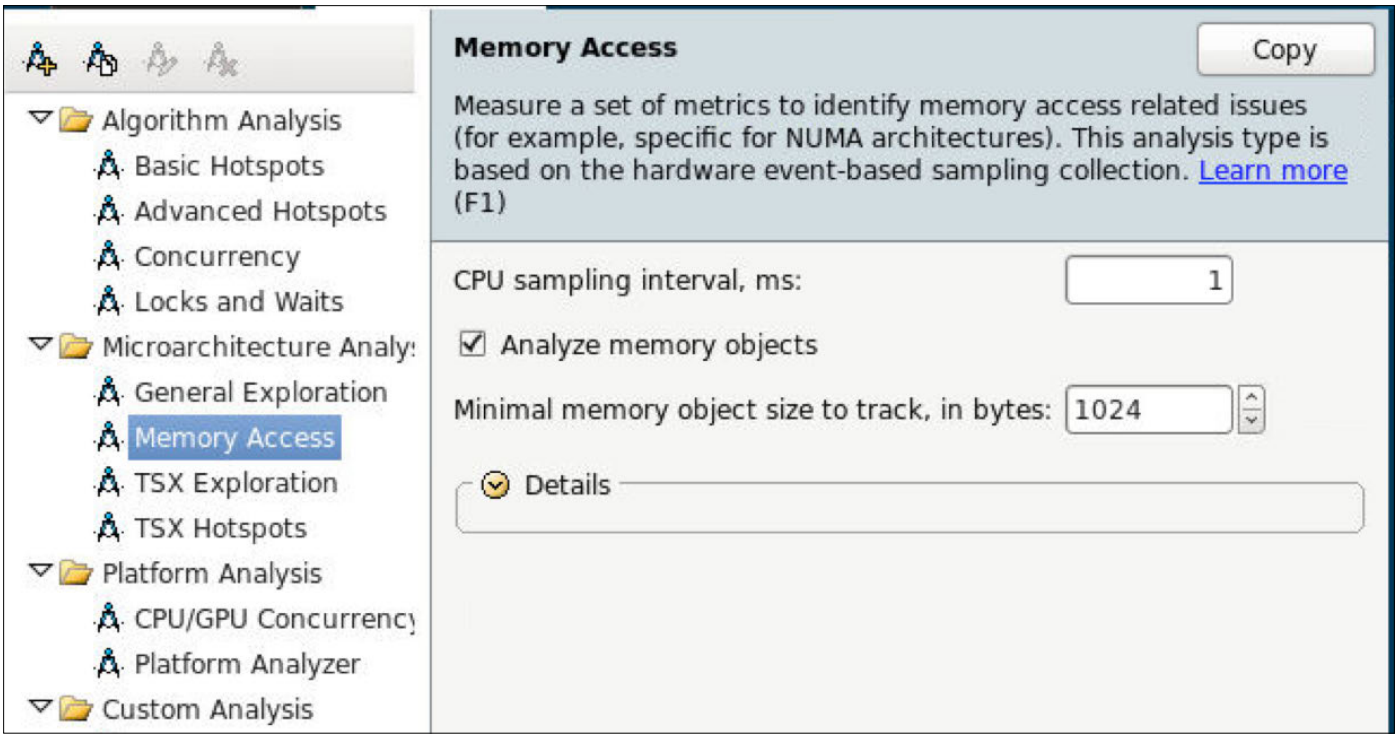
ステップ 3: メモリーアクセスを最適化する

メモリーアクセス解析

アプリケーションのメモリー・アクセス・パターンはパフォーマンスに大きく影響します。最近のプロセッサーは、多様なメモリーアクセスを行います。例えば、L1 キャッシュヒットのレイテンシーは、すべてのキャッシュをミスして DRAM にアクセスしなければならない場合のレイテンシーとは大きく異なります。不均等メモリーアクセス (NUMA) アーキテクチャーによる複雑さもあります。

インテル® VTune™ Amplifier XE には、メモリーアクセスの解析を支援する機能が用意されています。

- **メモリー階層でパフォーマンスの問題を検出** (L1-、L2-、LLC-、DRAM- バウンドなど)。メモリーアクセスのレイテンシーを適切なコードとデータ構造に関連付けます。
- **帯域幅が制限されたアクセス** (DRAM およびインテル® QuickPath インターコネクト [インテル® QPI] の帯域幅をカバー)。プログラムの帯域幅をタイムラインで示す DRAM およびインテル® QPI のグラフとヒストグラムを素早く確認できます。インテル® VTune™ Amplifier XE のメモリーアクセス機能を使用するには、新しいメモリーアクセス解析タイプをクリックして、[Start (開始)] をクリックします。



13 メモリーアクセス解析

アプリケーションを実行し、インテル® VTune™ Amplifier XE がメモリーアクセスをすべて解析します。

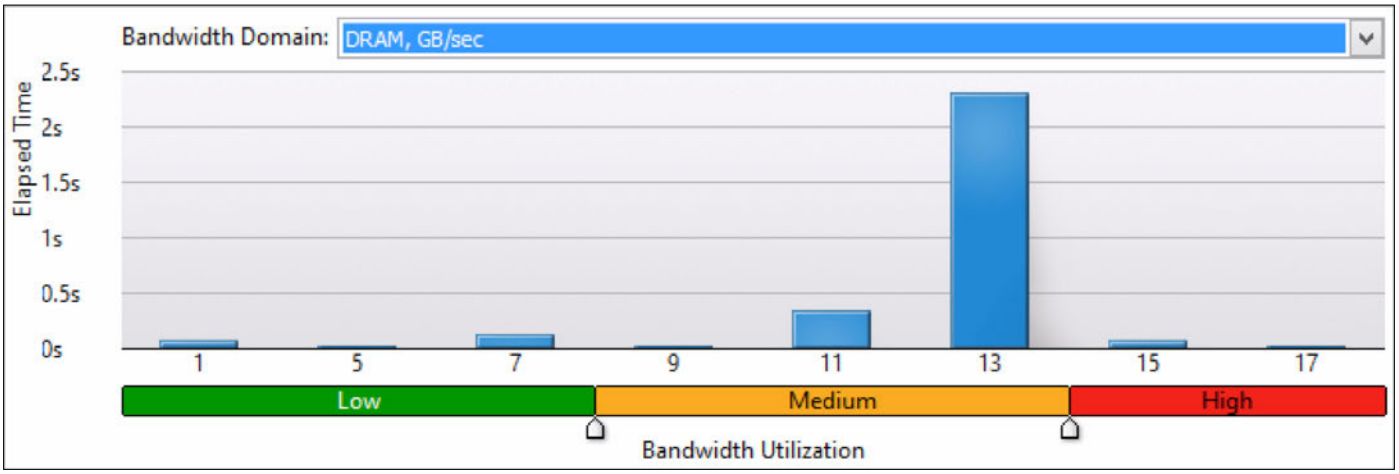
メモリーの問題の原因となっているコード領域を確認することもできます。**図 14** では、[Memory Bound (メモリーバウンド)] メトリックがピンクで表示されています。これは潜在的な問題を示しています。平均レイテンシーはメモリーアクセスで注目すべき重要なメトリックです。

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)	KNL Bandwidth Estimate (GB/s)
main	2.092s	0.651	1,074,003,222	674,410,116	33,602,016	248	18.898
lin_stream.cpp:100 (152 MB)			456,001,368	207,603,114	13,800,828	226	
lin_stream.cpp:99 (152 MB)			318,000,954	214,803,222	12,000,720	356	
lin_stream.cpp:98 (152 MB)			300,000,900	252,003,780	7,800,468	125	
[Unknown]			0	0	0	0	
__intel_ssse3_rep_memcpy	0.566s	0.000	252,000,756	128,401,926	0	41	19.131
LEVEL_BASE::LinuxProcMapsReader::ParseLine	0.002s	0.000	48,000,144	0	0	0	0.000
func@0x5042e7	0.008s	0.000	36,000,108	0	0	0	2.987
func@0x2c15f5	0s	0.000	24,000,072	0	0	0	0.000

14 潜在的な問題の特定

DRAM およびインテル® QPI の帯域幅を確認することもできます (**図 15**)。



15 帯域幅ヒストグラム

高い帯域幅使用率は問題となります。この問題を修正するには、帯域幅に影響しているコード領域を見つけます (**図 16**)。

帯域幅に影響しているソースコードとメモリー・オブジェクトを特定します。帯域幅ドメインでグループ化すると、メモリー帯域幅に最も影響しているメモリー・オブジェクトを特定できます (**図 16**)。DRAM やインテル® QPI に関連する問題を含むコード領域を確認できます。

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack

Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack	CPU Time	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)
DRAM, GB/sec	2.873s	0.818	1,764,005,292	846,012,690	37,202,232	195
High	2.291s	0.924	1,176,003,528	601,209,018	33,602,016	196
lin_stream.cpp:99 (152 MI			300,000,900	140,402,106	12,600,756	316
lin_stream.cpp:100 (152 M			426,001,278	261,603,924	12,000,720	199
lin_stream.cpp:98 (152 MI			444,001,332	193,202,898	9,000,540	73

16 メモリー帯域幅

グラフでアプリケーションのメモリー帯域幅をタイムライン表示

メモリー帯域幅は、一般にプログラムの実行とともに変動します。読み取り / 書き込みの帯域幅を GB/ 秒で表示したグラフ (**図 17**) から、アプリケーションのメモリー使用量が増加した場所を確認し、その領域に注目します。タイムラインでメモリー使用量が増加した場所を選択してフィルターし、その時間に動作していたコードを確認します。

BLOG HIGHLIGHTS

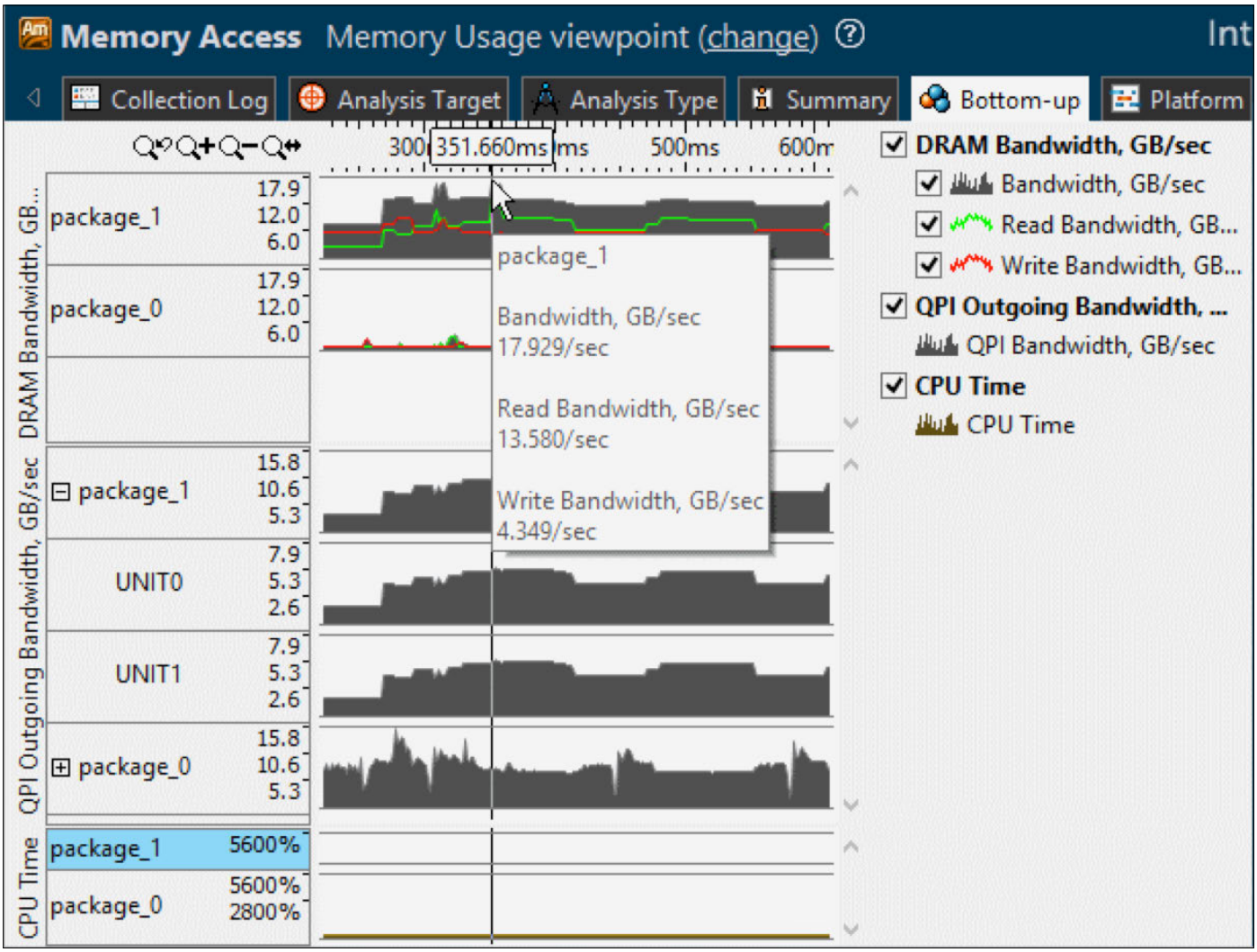
HPC クラウドの課題: インテル、Altair、AWS の開発者の取り組み

MIKE PEARCE >

ハイパフォーマンス・コンピューティング (HPC) は、もはや科学技術の世界に限られたものではありません。米国では、2015 年 7 月末に署名された、HPC の研究開発を推進する米国大統領令 (英語) により、National Strategic Computing Initiative が発足されました。HPC 分野における米国の地位向上を支援するよう開発者に求めるこのイニシアチブは、米国だけでなく、世界中の HPC の目標を引き上げることになりました。

これは、過去 20 年にわたり、大規模システムだけでなく、あらゆる規模の HPC のパフォーマンスとユーザビリティーの向上に取り組んできたインテルと Altair にとって、思いがけないニュースでした。我々はともに、ターンキー計算集約エンジニアリング向けクラウド製品 (物理および仮想) のようなソリューションにより、HPC がさらに利用しやすく、手頃で、便利になるように努力しています。

この記事の続きはこちら (英語) でご覧になれます。 >



17 メモリー使用状況

メモリー帯域幅に影響しているアプリケーションのコード領域を追跡する機能は非常に役立ちます。平均レイテンシーはメモリーアクセスをチューニングする際の重要なメトリックです。タイムライン・グラフで帯域幅を表示すると、アプリケーション実行時のメモリー使用量を簡単に区別することができます。

NUMA 解析

NUMA をサポートするプロセッサーでは、CPU のキャッシュミスを把握するだけでは十分ではありません。NUMA アーキテクチャーでは、別の CPU のキャッシュと DRAM を参照することもできます。このようなアクセスのレイテンシーは、ローカルよりも大きくなります。これらのリモート・メモリー・アクセスを識別して最適化する機能が必要です。

インテル® VTune™ Amplifier XE は、アクセスしているメモリー位置を識別できます。メモリーアクセス解析を実行後、[Memory Bound (メモリーバウンド)] をクリックして展開し、次の階層を確認することができます。

[Memory Bound (メモリーバウンド)] > [DRAM Bound (DRAM バウンド)] > [Local DRAM (ローカル DRAM)]
[Memory Bound (メモリーバウンド)] > [DRAM Bound (DRAM バウンド)] > [Remote DRAM (リモート DRAM)]
[Memory Bound (メモリーバウンド)] > [DRAM Bound (DRAM バウンド)] > [Remote Cache (リモートキャッシュ)]

[LLC Miss Count (LLC ミスカウント)] を展開することもできます。

[LLC Miss Count (LLC ミスカウント)] > [Local DRAM Access Count (ローカル DRAM アクセスカウント)]
[LLC Miss Count (LLC ミスカウント)] > [Remote DRAM Access Count (リモート DRAM アクセスカウント)]
[LLC Miss Count (LLC ミスカウント)] > [Remote Cache Access Count (リモート・キャッシュ・アクセス・カウント)]

平均レイテンシー (サイクル)

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time	Memory Bound							Loads	Stores	LLC Miss Count			Average Latency (cycles)
		L1 Bou..	L3 Late...	DRAM Bound				Other			Local DRA.	Remote DRAM A...	Rem.. Cac...	
				Rem..	Loc...	Rem..	Rem..							
main	2.092s	0.000	0.774			1.000		0.000	1,074,003,222	674,410,116	0	33,000,990	0	248
lin_stream.cpp:100 (152 MB)									456,001,368	207,603,114	0	16,200,486	0	226
lin_stream.cpp:99 (152 MB)									318,000,954	214,803,222	0	12,600,378	0	356
lin_stream.cpp:98 (152 MB)									300,000,900	252,003,780	0	4,200,126	0	125
[Unknown]									0	0	0	0	0	0
__intel_ssse3_rep_memcpy	0.566s	0.066	0.070		0.000	0.087	0.000	0.000	252,000,756	128,401,926	0	600,018	0	41
LEVEL_BASE::LinuxProcMapsReader::ParseLine	0.002s	0.000	0.000	0.000	0.000	0.000	0.000	0.000	48,000,144	0	0	0	0	0
func@0x5042e7	0.008s	0.000	0.000	0.000	0.000	0.000	0.000	0.000	36,000,108	0	0	0	0	0
func@0x2c15f5	0s	0.000	0.000	0.000	0.000	0.000	0.000	0.000	24,000,072	0	0	0	0	0
ATOMIC_CompareAndSwap64	0s	0.000	0.000	0.000	0.000	0.000	0.000	0.000	18,000,054	0	0	0	0	0

18 [Memory Access Latency (メモリー・アクセス・レイテンシー)] ウィンドウ

NUMA アーキテクチャーは複雑なため、メモリーアクセスに細心の注意を払う必要があります。アプリケーションのレイテンシーが最大のメモリーアクセスを最適化すると、最もパフォーマンスを向上することができます。

クラスター・スケーラビリティー用ツール

これまでのセクションでは、単一ノード内の最適化について説明しました。MPI を使用してアプリケーションをクラスターへスケーリングする場合、クラスター固有の問題を考慮する必要があります。クラスター (ノード間) のチューニングを最初に行った後、ノード内の最適化を行うことが一般に推奨されます。クラスターのチューニングでは、ノード間の通信を考慮してボトルネックを識別する必要があります。

アプリケーションはクラスターでスケーリングしますか？ アプリケーションは MPI を効率良く使用していますか？ 以下で説明するツールは、MPI アプリケーションの動作を理解したり、アプリケーションの実行に向けて正しくチューニングされた MPI 設定ファイルを作成することを支援します。

BLOG HIGHLIGHTS

インテル® MPI ライブラリーの初期化時間の短縮

MICHAEL STEYER >

InfiniBand* クラスターで大規模なインテル® MPI ライブラリー・アプリケーションを実行すると、MPI_Init() ルーチン内で費やされる時間が増加します。この原因は、すべての MPI ランクの環境が同じであることを確認するために必要な MPI ランタイム・インフラストラクチャーの管理操作です。数千ランクの大規模な MPI を実行すると、MPI 初期化フェーズの時間が非常に長くなります。

スタートアップ時間が長くなる要因はいくつかあります。ファブリックが利用可能になる前に PMI (プロセス管理インターフェイス) で行われる通信も要因の 1 つです。また、最初にグローバル集合操作があると、スタートアップ時のファブリック・ロードが多くなります。そのため、MPI ランクの数が増えると、ファブリックで渡されるメッセージの量が指数的に増加し、スタートアップ時間が非常に長くなります (特にスケーリング時)。

この問題に対応するには、個々のランクの環境が一貫していることを確認する特定のスタートアップ・チェックをオフにして、スタートアップ時間を短縮します。

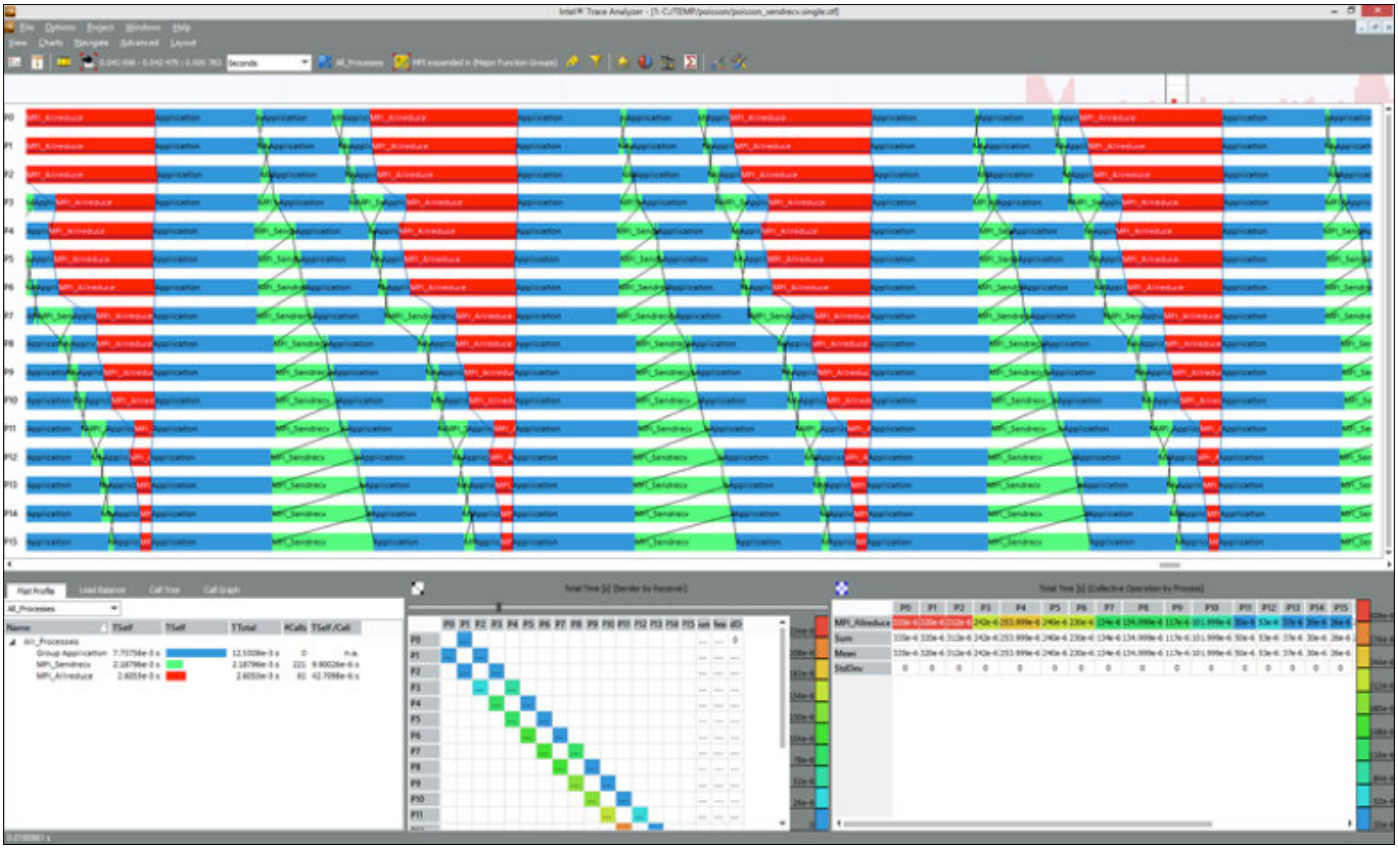
この記事の続きはこちら (英語) でご覧になれます。 >

インテル® Trace Analyzer & Collector (ITAC)

インテル® Trace Analyzer & Collector は、MPI アプリケーションの動作を理解し、ボトルネックを素早く特定し、正当性を向上し、インテル® アーキテクチャー・ベースの並列クラスター・アプリケーションで優れたパフォーマンスを実現するためのグラフィカルなツールです。次の機能を提供します。

- 並列アプリケーションの動作を**視覚化して確認**
- プロファイル統計とロードバランスを**評価**
- サブルーチンやコードブロックのパフォーマンスを**解析**
- 通信パターン、パラメーター、パフォーマンス・データを**表示**
- 通信 hotspot を**特定**
- **問題を短期間で解決**し、アプリケーションの効率を向上

図 20 は、解析トレースの例です。



20 解析トレースの例

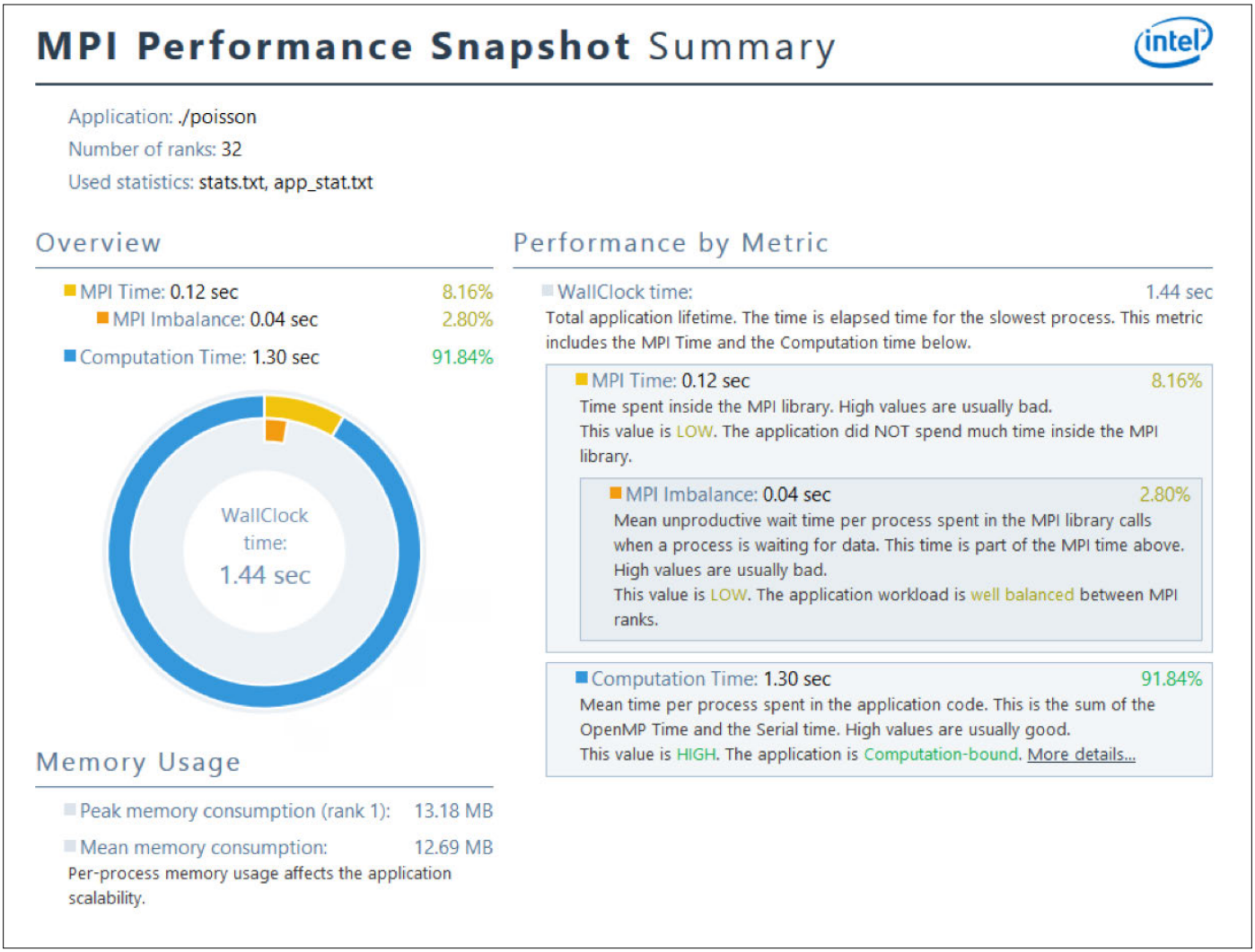
MPI Performance Snapshot (MPS)

MPI Performance Snapshot (MPS) は、MPI アプリケーション用のスケーラブルで軽量のパフォーマンス・ツールです。さまざまな MPI アプリケーションの統計を収集して、分かりやすい形式で表示します。このツールは単独では提供されません。インテル® Trace Analyzer & Collector の一部として提供されます。

MPS は、MPI アプリケーションの大規模スケーリングに関連する次の問題を解決できるように支援します。

- **スケーラビリティー。** クラスターのサイズは増え続けているため、アプリケーションのスケーラビリティーをさらに高める必要があります。大規模なプロファイリングを行うと大量のデータが収集され収拾がつかなくなりがちです。
- **メトリック。** 多くのデータを収集する場合、追跡すべき重要なメトリックを識別するのは困難です。
- **分類。** MPS は、インテル® MPI ライブラリーの軽量の統計と OS/ ハードウェア・レベルのカウンターを組み合わせて、通信、アクティビティー、ロードバランス、メモリー使用状況、MPI と OpenMP* のロード・インバランス情報、MPI/ 計算 / シリアル時間を含む、アプリケーションの高レベルの分類を提供します。

図 21 に MPI Performance Snapshot のサマリーを示します。



21 MPI Performance Snapshot のサマリー

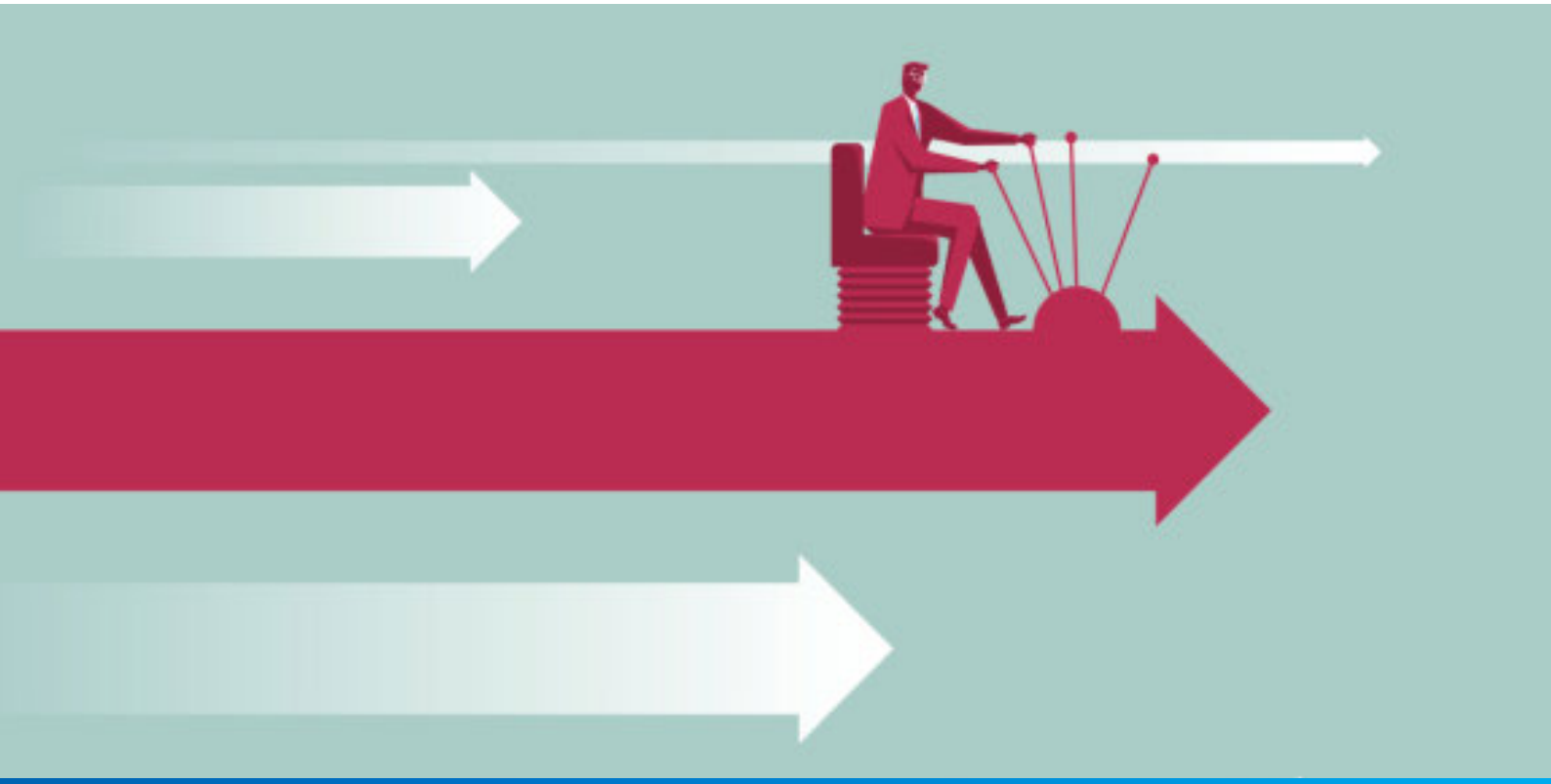
MPI Tuner

MPI Tuner (mpitune) は、特定のクラスターやアプリケーション向けの最適な設定を含むインテル® MPI ライブラリーの設定ファイルを作成するユーティリティーです。MPI Tuner ユーティリティーは 2 つのモードで動作します。

- **クラスター固有 :** インテル® MPI Benchmarks やユーザーのベンチマーク・プログラムを使用してクラスター環境を評価し、最適なインテル® MPI ライブラリーの設定を見つけます。
- **アプリケーション固有 :** MPI アプリケーションのパフォーマンスを評価して、最適なインテル® MPI ライブラリーの設定を見つけます。

コードの現代化

ハードウェアを最大限に活用するには、ベクトル化とスレッド化を導入してコードを現代化する必要があります。この記事で説明した系統的なアプローチを採用し、インテル® Parallel Studio XE の強力なツールを活用することで、簡単にこの作業を行うことができます。



将来の HPC 問題に向けたコードの現代化

並列アプリケーションを必要とする新しい市場への対応

Don Gunning インテル コーポレーション プロダクト・マーケティング・エンジニア

大量の非体系的データやインターネットを利用したモデリング、シミュレーション、仮想プロトタイプ作成の組み合わせへの依存を高め、変化し続ける経済の要求を満たすには、コードの現代化—および将来のソフトウェアの開発—が必要なことは明らかです。これは、ハイパフォーマンス・コンピューティング (HPC) ソフトウェア開発者が、(博士号を持つ) HPC スペシャリストから新しい市場セグメントのドメイン・スペシャリストと大量の情報を解析するデータ・サイエンティストに変わったことを意味します。

この記事では、HPC ソフトウェア開発者、ドメイン・スペシャリスト、データ・サイエンティストとの共同作業により役立つことが分かった、コードの現代化についてのヒントを紹介します。私は、これらのヒントが重要になる HPC 市場の 3 つの変化に気付きました。

- 「Code Modernization (コードの現代化)」と呼ばれる、並列プログラミングへの移行
- ハイパフォーマンス・データ解析や機械学習のような多くの新しい分野でドメイン・スペシャリストやデータ・サイエンティストが直面している大量の非体系的データセットの処理に関する課題
- 仮想環境やクラウド環境でのアプリケーションの実行

ドメイン・スペシャリストは、新しい斬新な方法で HPC アプリケーションを作成しています。その中には、単純な計算流体力学ソフトウェアを記述している (将来の科学者である) 高校生も含まれます。ドメイン・スペシャリストは、リアルタイムで解析されるセンサーデータに、MapReduce/Hadoop*、グラフ解析、意味解析、知識発見アルゴリズムのような、新しい高度な解析手法を適用しています。

この動きは、ソフトウェア開発環境の変化につながっています。環境をどのように使用するかという点でも変化があります。そのため、生成したアプリケーションを効率良く実行する方法を考慮することも重要です。それでは、これらの分野と必要な新しい機能の実現方法について見てみましょう。

「インテル® Advisor XE は、既存のコードの並列化に必要な作業を理解する上で非常に貴重なツールです。並列化の可能性の発見、テストの設計、シナリオのモデル化、不具合の特定に役立ちます。」

Vickery Research Alliance
シニア・ソフトウェア・エンジニア
Matt Osterberg 氏

ソフトウェア開発環境

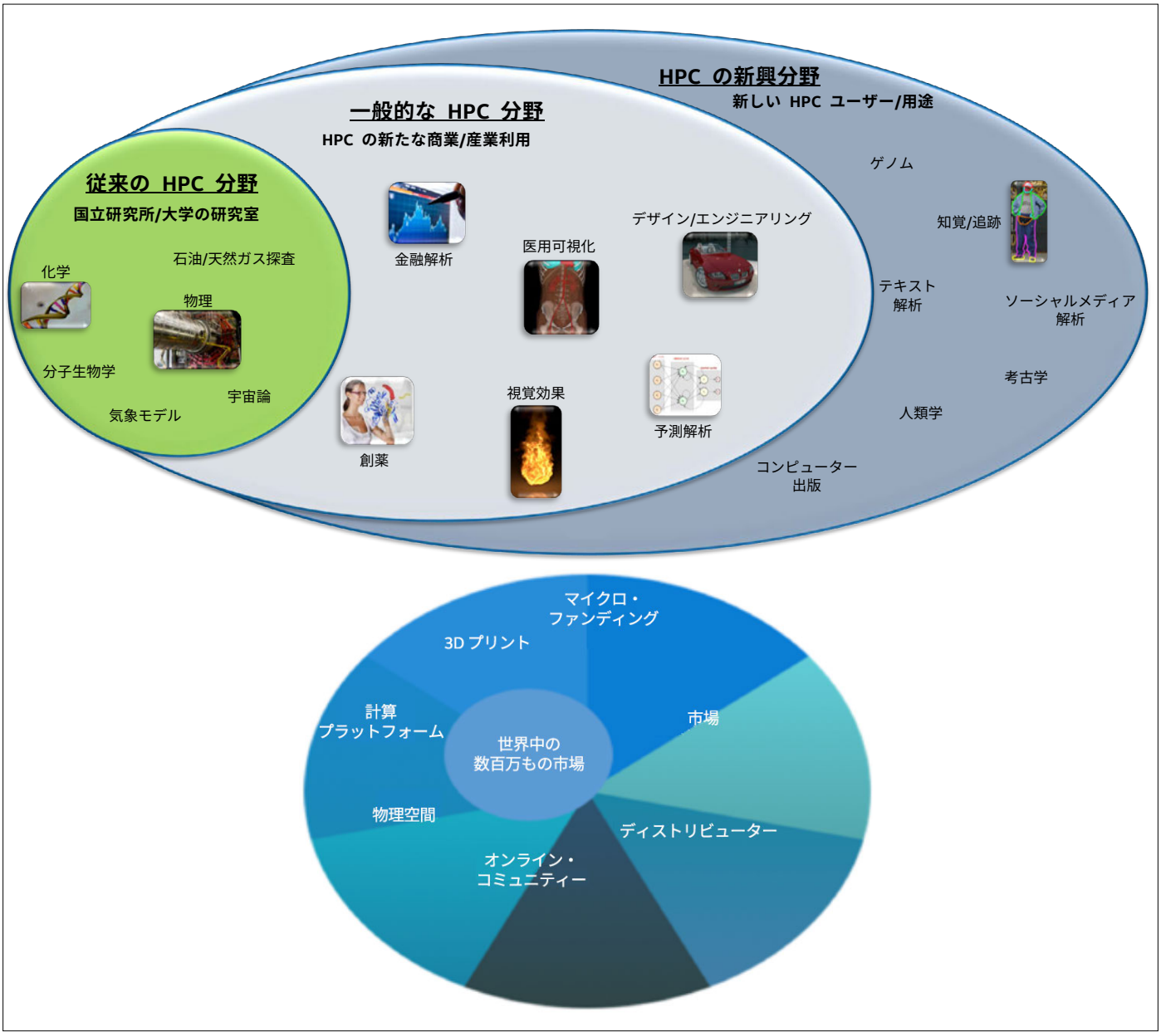
開発者は多くの壁に直面しています。

- さまざまな並列計算の理解
- 難しいインターフェイスへの対応
- 既存のコンポーネントの統合
- 実行のリソース要件の定義

インテルは、生成したアプリケーションが最適なスケールで実行されるように、ユーザーが最も効率的な方法で C++、スクリプト言語 (Python* など)、オブジェクト、およびライブラリーを利用できる環境を提供しています。

従来の HPC 市場では、専門のプログラマーが C/C++ や Fortran でプログラムを記述し、基本的または高度な数学機能とリンクした後、分散メモリー構造で並列処理を追加していました。プログラマーは、研究所、大学、企業からのイノベーションに依存していました。エンドユーザーは、これらのプログラムを利用して商業や産業の問題に対応していました。

HPC の新興分野では、ドメイン・スペシャリストとデータ・サイエンティストが、R、Python*、Java*、Julia のような言語を使用して、事前ビルド済みのライブラリーとソルバーを組み合わせた開発フレームワークで問題に対応しています。つまり、イノベーションがほかの分野 (例えばメーカー) で起こり、新しい実行モデルが登場しているのです。**図 1** は、これらの分野が拡大している様子を示しています。



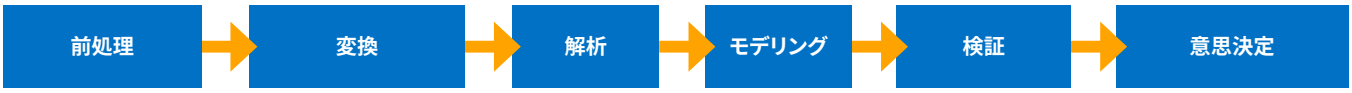
1 HPC 分野の拡大

ソフトウェア開発環境の活用

マルチコア / メニーコア・アーキテクチャーと高速インターコネクトの登場により、プログラミングの性質は変化し、次の要件が生じました。

- より複雑なソフトウェアを迅速に作成する
- 大規模で複雑なデータセットを処理する
- さまざまなレベルの並列処理を適用する

これらの要件を満たすため、C++ のようなオブジェクト指向言語の利用が増えるとともに、プログラムの多くの部分でスクリプト言語が使用されるようになりました。ワークフローを**図 2** に示します。

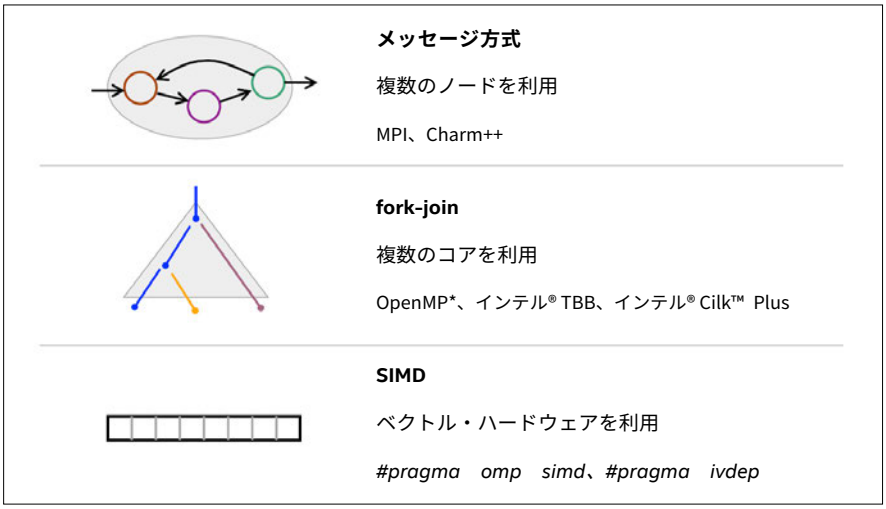


2 典型的な開発ワークフロー

従来の HPC 開発者は次の点に注目しました。

- 最大ノード数にコードをスケーリングする
- 各ノードのコアを効率的に利用する
- 各コア内のベクトル機能を活用する

そして、**図 3** のような並列化モデルが誕生しました。



3 並列化モデル

メモリーとキャッシュを管理するソフトウェア・ツールを使用してデータを配置することで、チャネル使用率を最大限にしてフラグメントを減らすようにアライメントが行われました。これらのツールは、ドメイン・スペシャリストとデータ・サイエンティストが直面している正当性とパフォーマンスの課題を効果的に解決できるように拡張されました。

インテルのツールを使用した複雑なコードの管理

現在の HPC 環境では、開発者は高水準言語 (C/C++ や Fortran) やスクリプト言語 (Python*) を組み合わせてさまざまなレベルの並列処理を実装しています。複雑なコードの管理には、優れたツールを効率良く使用する新しい方法が必要です。

まず、インテルのコードの現代化**開発フレームワーク**に従って、アルゴリズムが現在および将来のハードウェアで効率的にスケーリングできるように、最近のハードウェアで利用可能なベクトル化、マルチスレッド化、マルチノード最適化などの機能を活用して複数レベルの並列化を行います。

このフレームワークは、ソフトウェアを現代化するための方法を提供します。

- 最適な並列化レベルを利用する
- 複数のスレッドのメモリーおよびキャッシュ効率を向上する
- 高チャネル使用率と低フラグメントになるようにデータレイアウトとアライメントを最適化する

このフレームワークにおいて、**インテル® Parallel Studio XE Cluster Edition** は前述の 3 つの課題を解決できるように支援します。

- 新しいアプリケーションの開発
- 既存のコードの並列化
- 既存の並列コードの利用

インテル® Parallel Studio XE には、C/C++ コンパイラー、Fortran コンパイラー、スレッドレベルの並列化 (インテル® TBB、OpenMP*) 用のスレッド化ライブラリー、分散メモリー用のメッセージ・パッシング・インターフェイス (MPI) ライブラリーが含まれています。近い将来、Python* Basic ライブラリーのサポートも提供される予定です。

「我々は 1 週間かけてクラッシュの解決に取り組みました。問題は分かりましたが、その場所を特定することは非常に困難でした。インテル® Inspector XE を実行したところ、実際にクラッシュが発生するかなり前に境界外の配列を使用していることが直ちに判明しました。1 週間の作業があっという間に完了したのです。」

Envivio
シニア・コーデック・アーキテクチャー・エンジニア
Mikael Le Guerroué 氏

インテル® マス・カーネル・ライブラリー (インテル® MKL) と**インテル® インテグレーテッド・パフォーマンス・プリミティブ** (インテル® IPP) は、基本的な数学関数のシリアル / 並列プログラミングをサポートします。HPC 開発者は、これらのライブラリーのことを知っているかすでに使用しているでしょう。新しいコードを記述しているドメイン・スペシャリストやデータ・サイエンティストは、多くの計算がこれらのライブラリーに含まれる関数で可能な場合、関数を置換することでコードの現代化を効率的に行うことができます。

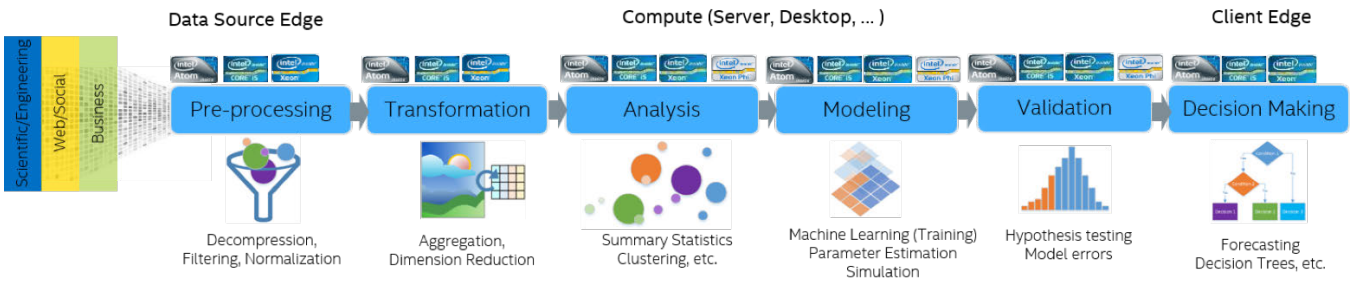
これらのライブラリーは、複数レベルの並列化を完全にサポートするように最適化されています。**インテル® Data Analytics Acceleration Library** (インテル® DAAL) は、HPC 開発者とドメイン・スペシャリスト向けの新しいライブラリーです。このライブラリーは、インテル® MKL とインテル® IPP の基本的な機能に、前述のワークフローで示した追加機能を加えたものです。

「インテル® VTune™ Amplifier XE は、複雑なコードを解析し、迅速にボトルネックを特定するのに役立ちました。ほかのインテル® ソフトウェア開発ツールと併用することで、以前のバージョンと比較して PIPESIM* のパフォーマンスを 10 倍も向上することができました。」

Schlumberger
シニア・サイエンティスト
Rodney Lessard 氏

もしインテル® MKL、インテル® IPP、インテル® DAAL で解決できない場合はどうすればいいのでしょうか？

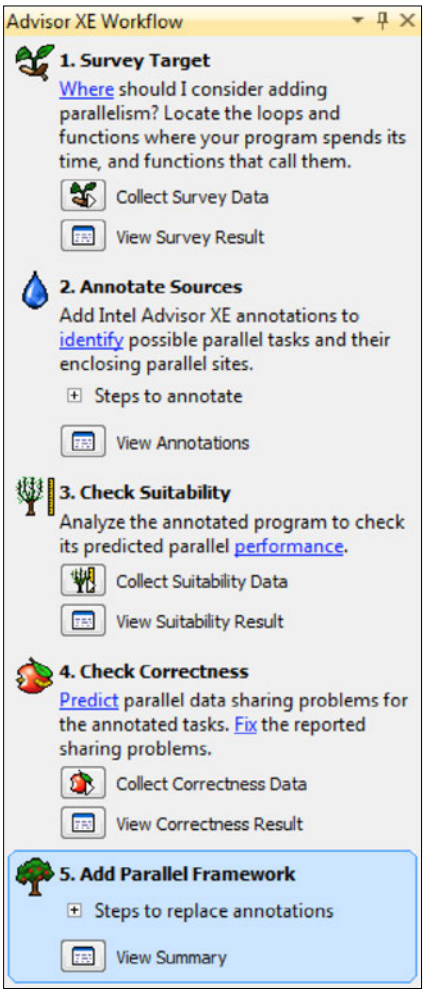
事前ビルド済みライブラリー (**図 4**) を使用することで、ドメイン・スペシャリストは、並列処理の実装方法を心配することなく並列コードを利用できます。



4 事前ビルド済みライブラリー

ソフトウェアを変更または開発する前に、インテル® Parallel Studio XE を使用することで (並列アプリケーションのエキスパートの知識に基づいて)、スレッドレベルおよびベクトルレベルの並列化のプロトタイプ作成を行うことができます。

インテル® Advisor XE のワークフロー (**図 5**) は、並列化の実装を支援します。新しいソフトウェアの開発と既存のシリアルコードの並列化のどちらにも対応しています。インテル® Advisor XE は、問題の仕様を実装する別の方法を示します。すべてのソフトウェア開発と同様に、仕様が優れていることが重要です。次に、シリアルプログラムでどんな計算をどのように行うかを記述します。インテル® Advisor XE は、並列プログラムに変更可能なシリアルプログラムを特定するのに役立ちます。インテル® Advisor XE は、シリアルプログラムを調査して、並列化により最も大きな利点がもたらされるコード領域を見つけます。



「インテル® Trace Analyzer & Collector for Linux* は、理研の分子動力学クラスター・ソフトウェアのパフォーマンス向上に大いに役立ちました。非ブロッキング通信パターンによるボトルネックを特定して排除することにより、MPI 通信時間を半分に短縮することができました。インテル® Trace Analyzer & Collector はインストルメンテーション・コードをプログラムに組込むことができるため、各関数の実行時間とロードバランスを表示して、どこを最適化すれば良いか非常に簡単に理解することができました。インテル® MPI ライブラリーとインテルのクラスターツールは最高のクラスター開発環境です。」

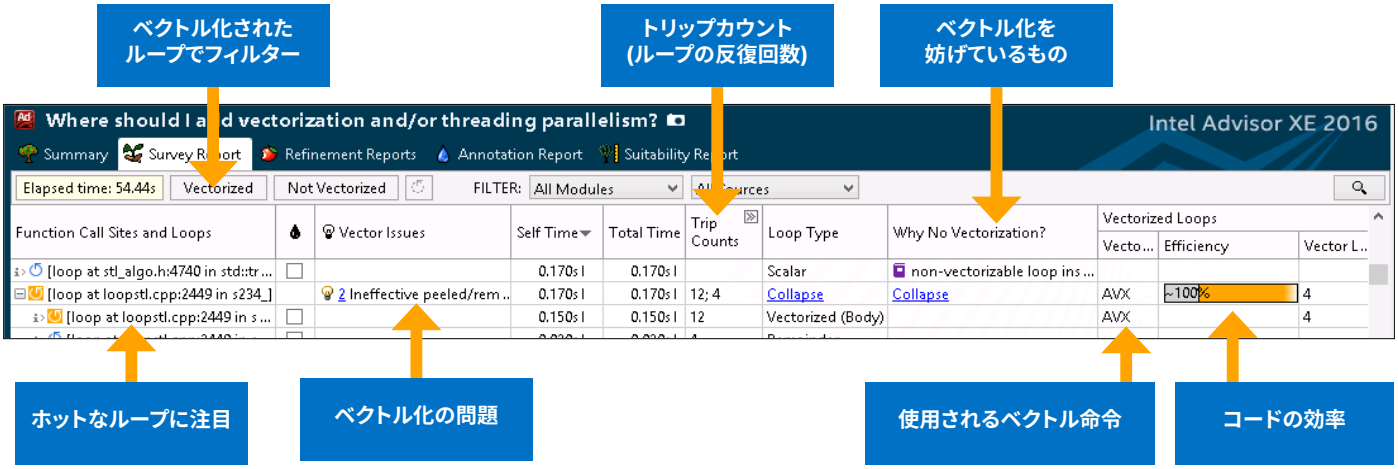
理研
古石 貴裕 博士

5 インテル® Advisor XE ワークフロー

見つかったコードに開発者が注釈を追加すると、インテル® Advisor XE は、モデリング言語を起動してパフォーマンスと正当性に関する質問を行います。開発者は質問に答えるだけで、並列呼び出しを追加して正当性を検証しなくても、並列化の有効性をテストすることができます。

インテル® Advisor XE は、開発者が直面している重要な問題—「コードをベクトル化する場所とタイミング」—を分かりやすく説明します。これまで開発者は、コンパイラーの自動ベクトル化機能だけしか利用できませんでした。インテル® Advisor XE を使用することで、開発者は次の質問の答えから必要な情報を得ることができます。

- どのベクトル化により最も大きな利点が得られるか？
- ベクトル化を妨げている原因は？
- ループがベクトル化に適しているか？
- データを再構成することでパフォーマンスが向上するか？
- `#pragma simd` だけで大丈夫か？

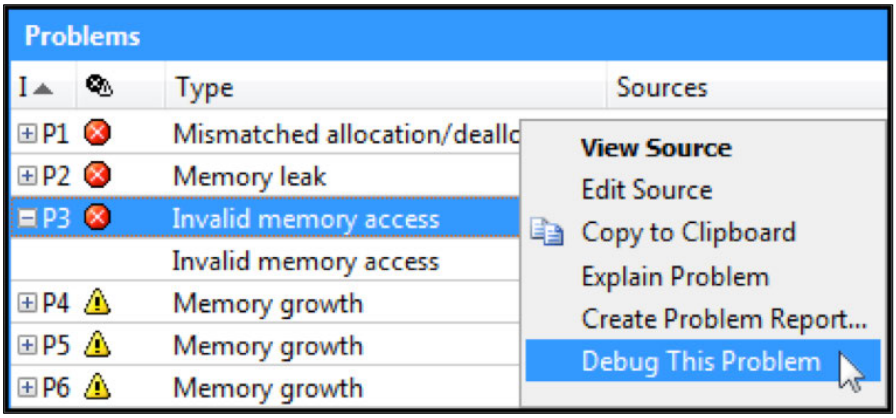


6 インテル® Advisor XE の機能

インテル® Advisor XE は、パフォーマンス・データを診断して、「ホットな」ベクトル化されなかったループやベクトル化が十分でないループを特定します。さらに、(依存性およびメモリー・アクセス・パターン解析により) 正当性を考慮しながら推奨する問題解決方法を提示します。

インテル® Advisor XE のすべてのステップを完了後、プログラマーは、インテル® Advisor XE が推奨した場所で実際のプログラミング・モデル (インテル® TBB、OpenMP* など) を使用してプログラムを変更することができます。

新規 / 既存のスレッドレベルの並列コードは正しいか？

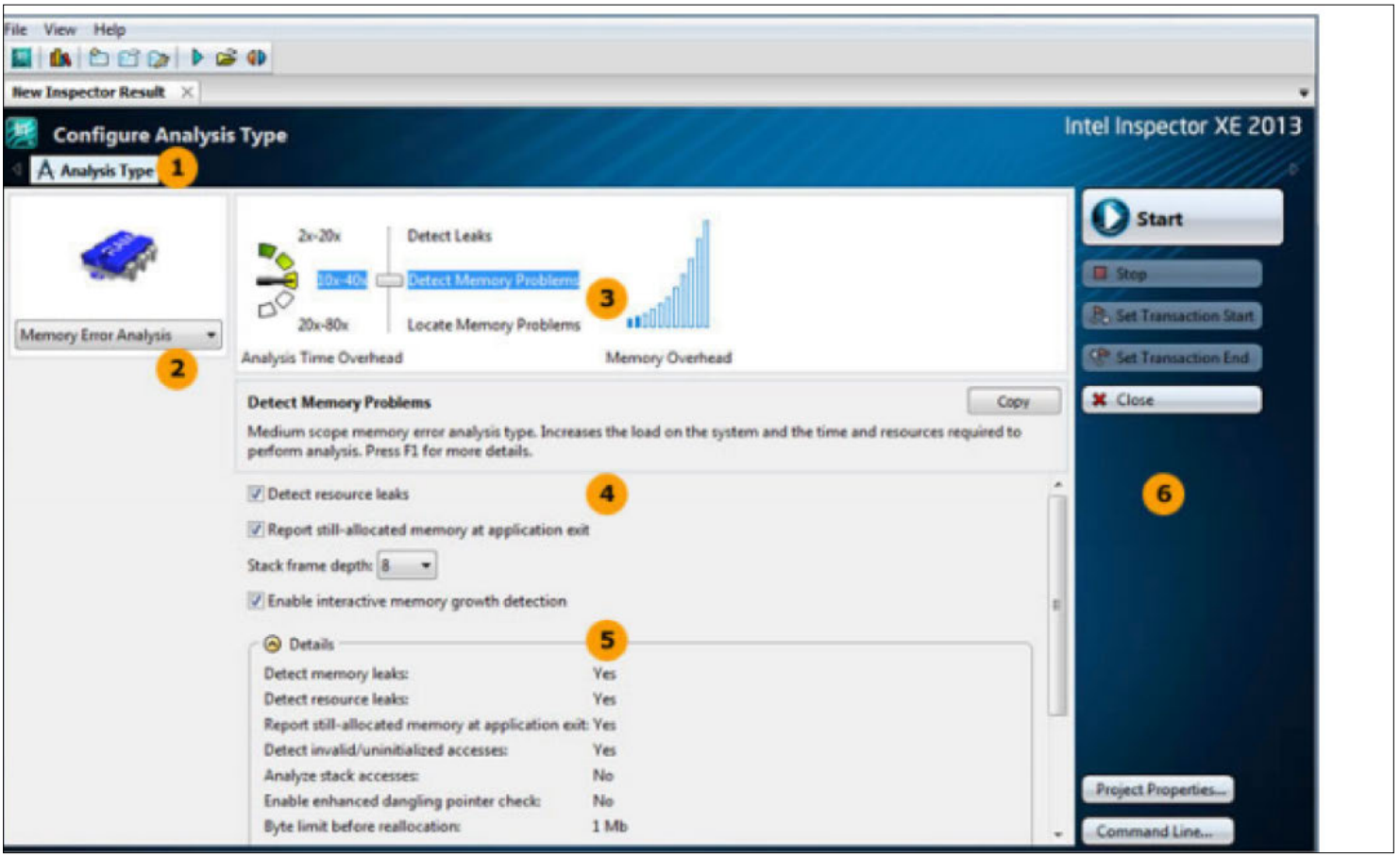


7 インテル® Inspector XE の正当性検証

前述の説明に従って構築されたコードや既存のアプリケーションの拡張 (特に小規模の拡張) を検証する場合、最初の作業は並列化が正しいことを保証することです。

図 7 は、**インテル® Inspector XE** の正当性検証機能を示しています。このツールは、通常は発見が困難な競合、デッドロック、メモリーエラーをグラフィカルに表示します。

インテル® Inspector XE の詳細解析は実行時間が長くなるため、使用するデータセットは、プログラム関数の処理回数が最小限になるものにします。このツールを利用することで、これまで数カ月かかっていた診断と修正を数時間で完了できます。



8 インテル® Inspector XE で無効なメモリーアクセスを検証

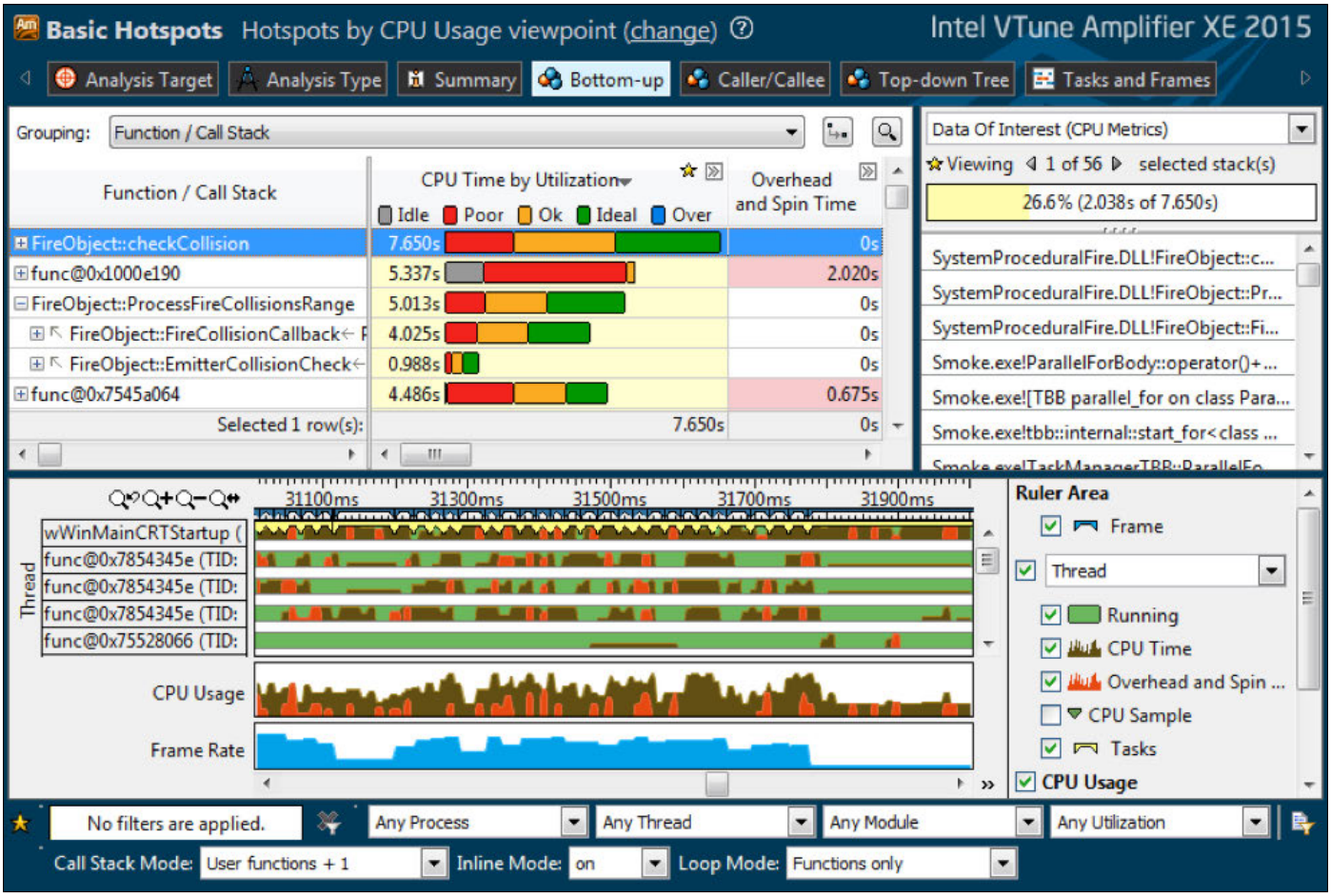
動作するか？

コードが本番に近いレベルになり、ハードウェアやソフトウェアの変更が行われたら、メモリーとキャッシュの効率を向上して、高チャネル使用率と低フラグメントになるようにデータレイアウトを最適化します。

一般に、この段階では、スレッド化のさまざまな問題をツールを使用して解決します。**インテル® VTune™ Amplifier XE** (**図 9**) の 2 つの新しい機能は次のとおりです。

- 1 つ目は、ドメイン・スペシャリスト向けに拡張されたことです。C++ ドメイン・スペシャリストからは、このツールは使いやすく効果的であるとコメントが寄せられています。近い将来、Python* スクリプト言語のサポートも提供される予定です。
- 2 つ目は、ソフトウェアを MPI 対応にする可能性を効率良く見つけられることです。ツールを起動するだけで、レポートに必要なデータが自動的に収集されます。

インテル® VTune™ Amplifier XE は、hotspot と呼び出しカウントの統計を提供し、コンカレンシー、キャッシュミス、帯域幅解析を行うことができます。インテル® Xeon Phi™ コプロセッサーでは、オフロードの可能性を指摘します。

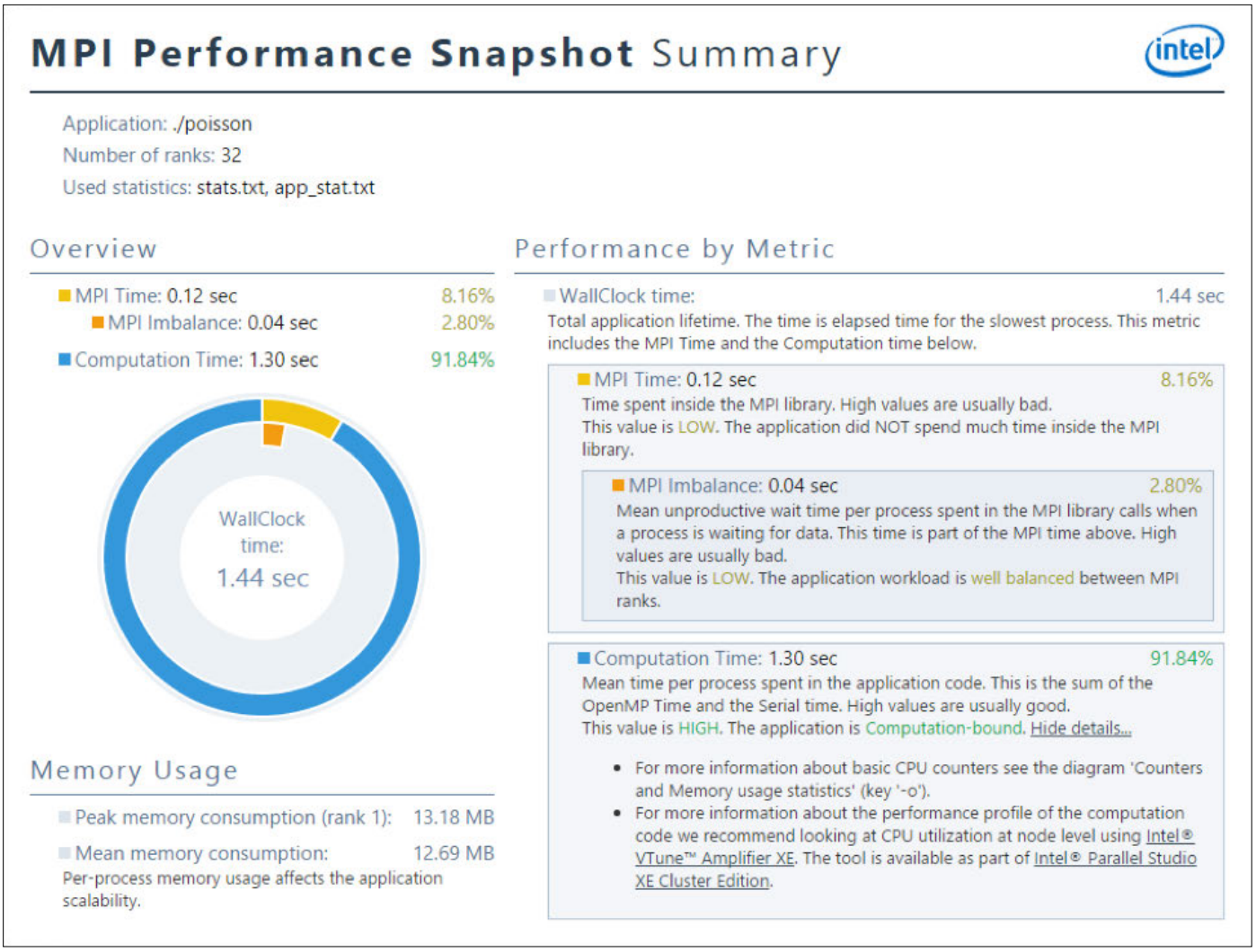


9 インテル® VTune™ Amplifier XE

クラスター・スケーリングにより利点がもたらされる場合 (つまり、より速く結果が得られる場合や同じ時間内により大きなデータセットを処理できる場合)、マルチコアまたはマルチコア / メニーコア・クラスター対応にします (これは、HPC では通常 MPI 対応または MPI + OpenMP* 対応を意味します)。

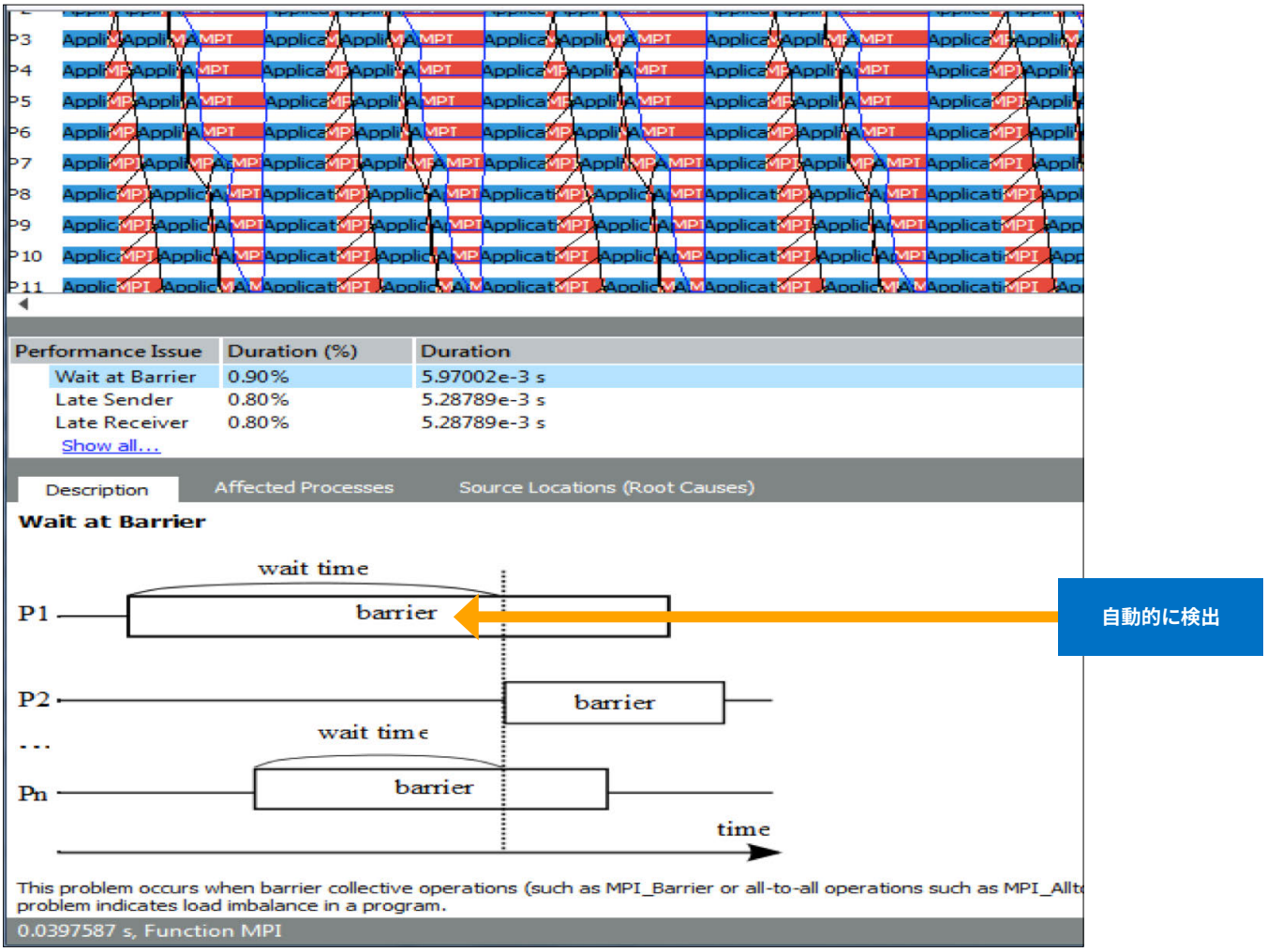
前述のツールで行った作業に基づいて、開発者はソフトウェアの適切な場所に MPI 呼び出しを追加します。そして、正当性を検証した後、パフォーマンスの向上に取り組みます。インテル® ソフトウェア・ツールは、さまざまな方法でこれを支援します。

正当性検証が完了したら、実際のデータセットで MPI Performance Snapshot を実行することで、プログラム全体の MPI パフォーマンスのサマリーを確認し、パフォーマンス向上の可能性がある場所を特定できます (**図 10**)。



10 MPI Performance Snapshot のサマリー

続いて、インテル® Trace Analyzer & Collector を実行します。インテル® Trace Analyzer & Collector は、インテル® VTune™ Amplifier XE の概念を拡張したツールで、プログラム全体の統計を収集します。開発者は、理想的なネットワーク条件でアプリケーションのパフォーマンスをシミュレーションすることができます。また、イベントデータ収集と詳細解析を実行するプログラム領域を指定して、パフォーマンスの問題とランタイムの影響を自動的に検出することができます (**図 11**)。



11 パフォーマンスの問題を自動的に検出

ターゲット環境で動作するか？

すべてのアプリケーションには、ターゲット実行環境があります。開発者は、通常クラスターでソフトウェアをテストして、データセットのパフォーマンスを測定します。**インテル® Parallel Studio XE Cluster Edition** に含まれるインテル® Cluster Checker は、100 を超えるチェックを実行して、クラスターのライフサイクル全体にわたって動作するクラスター・コンポーネントを検証できるように支援します。開発者およびドメイン・スペシャリストは、このツールを使用してパフォーマンス・レベルをテストし、アプリケーション・ユーザーに実行環境を推奨することができます。インテル® Cluster Checker は問題を検出すると、エキスパート・システム機能で重要度、根本的原因、修正方法を知らせます。また、インテル® Cluster Checker の制限付き関数ランタイムバージョンを開発したアプリケーションに組込むこともできます。アプリケーション・ユーザーは、このソフトウェアを使用して実行クラスターをチェックし、問題の予備診断を行うことができます。

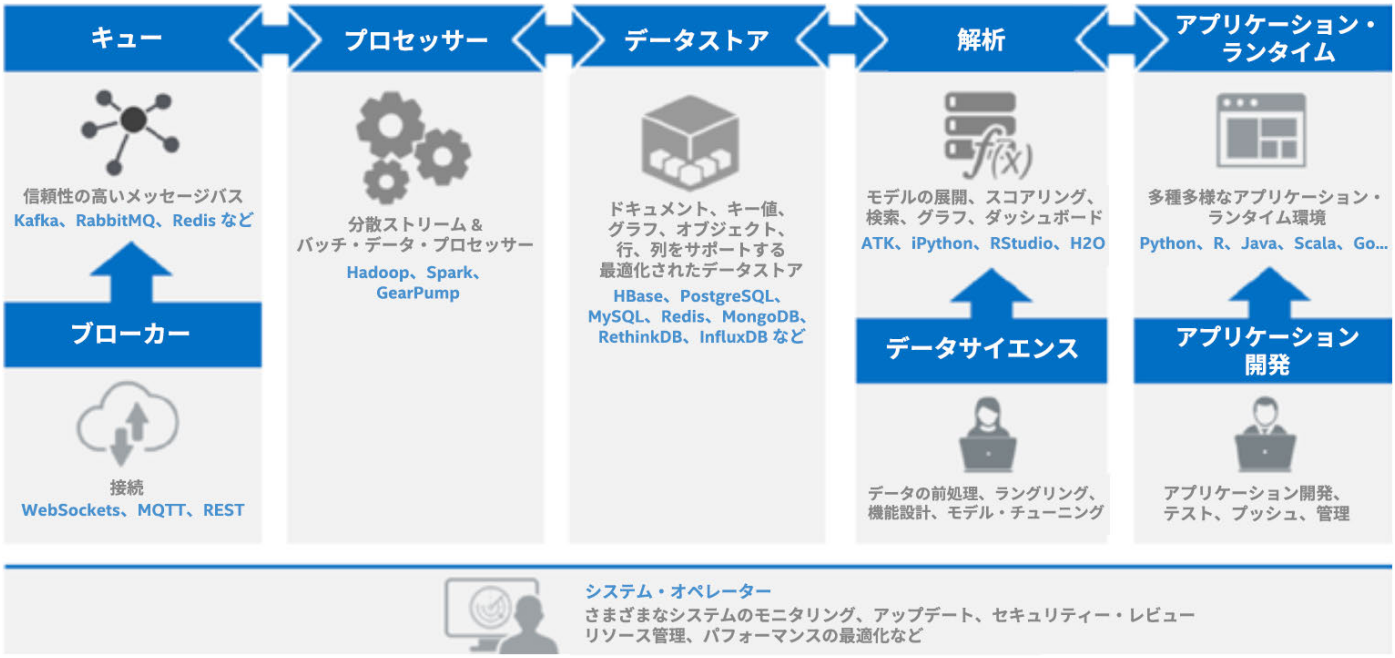
これらのツールはアプリケーション開発に有効なだけではなく、データセットの変更によって生じるパフォーマンスの問題を特定したり、一見問題のない拡張の正当性とパフォーマンスへの影響を示すことにより、保守コストも軽減します。

ビッグデータはどうか？

これらのインテル® ソフトウェア・ツールはすべて、インテルのコードの現代化**開発フレームワーク**に含まれており、ワークフローやフレームワークでアプリケーションを開発、提供、保守するのに役立ちます。しかし、Bob Rogers (インテル コーポレーション、ビッグデータ・ソリューションのチーフ・データ・サイエンティスト) によれば、「強力な解析機能をすべてのビジネス意思決定者に提供する」こともインテルのミッションです。

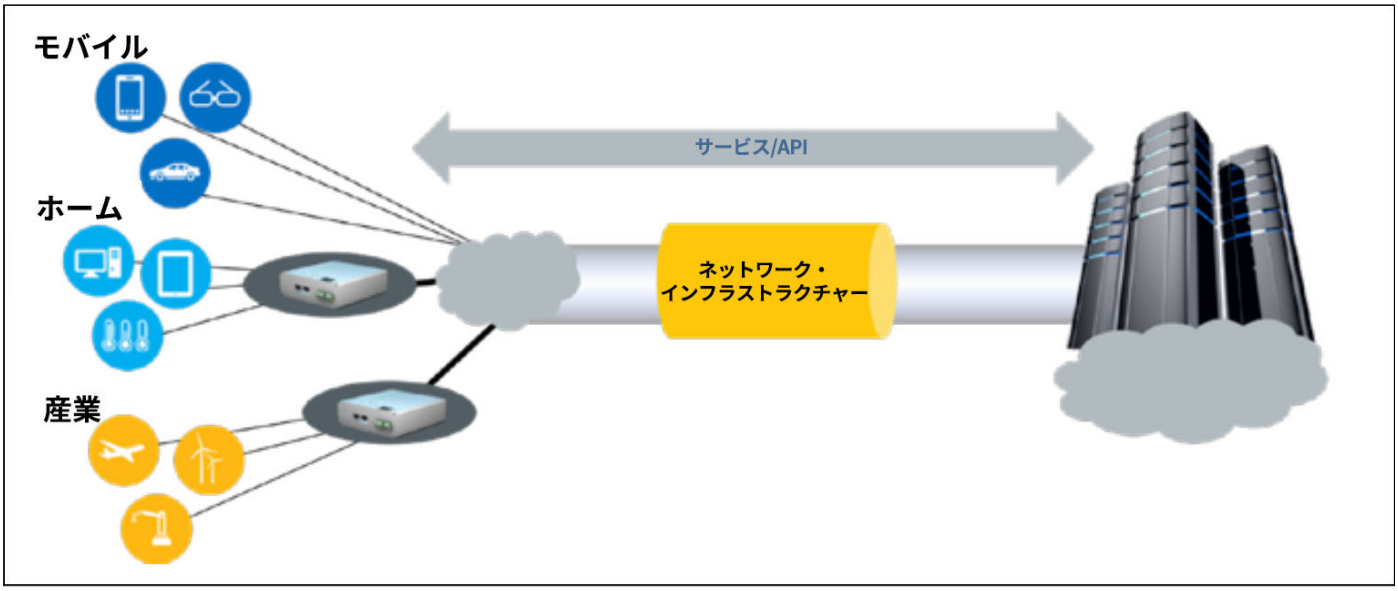
HPC 関連のドメイン・スペシャリスト向けに、インテルでは仮想実行に関連する MPI レイテンシーを軽減し、クラウド環境で実行できるように取り組んでいます。非常に複雑なモデリングとシミュレーションが可能なインテルの MPI 対応のジョブは、Apache* Hadoop* プラットフォームと密に統合して実行することができます。そのため、ドメイン・スペシャリストは、高速、高スループットで、信頼性の高い解析が求められるデータ・サイエンティストの世界に足を踏み入れることができます。

現代のデータ・サイエンティストに要求される高スループット解析の最も大きな障害はサイロ化です。**Trusted Analytics Platform** (TAP) は、コンポーネントが連携して動作できるようにすることでサイロ化を防いでいます (**図 12**)。



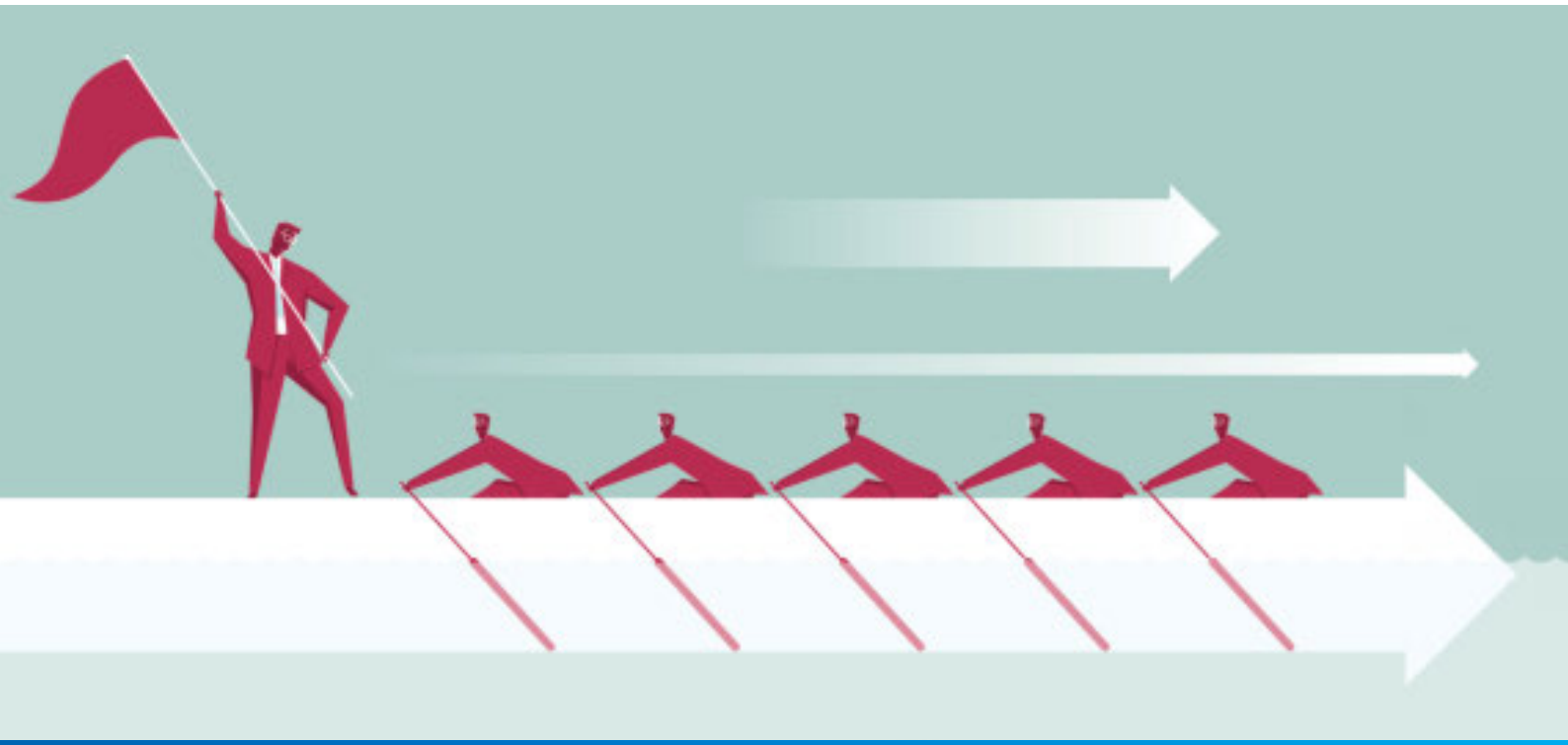
12 コンポーネントが連携して動作できる論理的な方法

インテル® プロセッサーは、異なる市場向けにさまざまなパフォーマンスと機能を提供しています。インテルのソフトウェア開発ツールは、この記事で紹介した機能のサブセットを提供します。テクニカル・ソフトウェア開発者、ドメイン・スペシャリスト、データ・サイエンティストは、これらのツールを利用して、将来のデータ解析や機械学習のさまざまな要件 (**図 13**) を満たすアプリケーションを提供することができます。



13 将来のハイパフォーマンス・データ解析環境

詳細は、**インテル® Modern Code の Web サイト**を参照してください。HPC 手法が Hadoop* の世界で採用されるようになり、Trusted Analytics Platform のようなソリューションは、将来の問題に取り組んでいる開発者とデータ・サイエンティストのコミュニティー全体と関係を持つことにより、サイロ化を防ぎます。



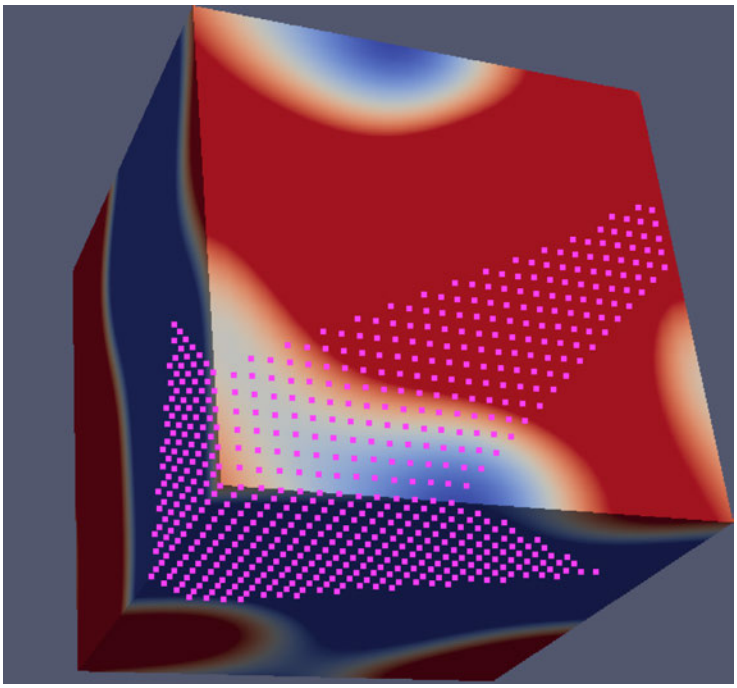
ベクトル化アドバイザーのヒントの活用

Zakhar A. Matveev インテル コーポレーション ソフトウェア・アーキテクト
Stephen Blair-Chappell 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

新しい**ベクトル化**アドバイザー (インテル® Advisor XE の機能) をまだ試していない方は、この記事が、ツールを利用するきっかけとなることでしょう。ベクトル化アドバイザーを利用することは、信頼できる知り合いにコードを調べてもらい、助言を受けることに似ています。

飛行機の機体面、インクジェット印刷、油 / 水混合物の分離、新しいシャンプーで使用されているプロテイン・ソリューションに共通の項目は何でしょうか？計算化学者は、これらすべての分野において、凝縮物質のメゾスコピック・シミュレーションが必要であると言うでしょう。「メゾスコピック」とは、シミュレーションで原子より少し大きな大量の物質を扱う必要があることを意味し、「凝縮物質」には、液体や固体の状態をモデル化することが含まれます。

広範なメゾスケール関連の科学および業界の要求に応えるため、イギリスの STFC Daresbury Laboratory の研究者は、DL_MESO と呼ばれるメゾスコピック・シミュレーション・パッケージを開発しました。DL_MESO は、シャンプー、粉体洗剤、農薬、石油添加剤の最適な化学式の計算 (コンピューターを利用した式化) にメゾスケールのシミュレーションを使用する、Unilever、Syngenta、Infineum などの企業によりヨーロッパ市場で採用されています。



1 3D_PhaseSeparation ベンチマークの視覚化

コンピューターを利用した式化シミュレーション・プロセスには大量の時間とリソースが必要になることが多いため、当初から Daresbury のエキスパートは、最近のプラットフォームにおけるパフォーマンス重視の設計と DL_MESO の最適化に関心を持っていました。これが Hartree Centre とインテルによるインテル® Parallel Computing Center (IPCC) 共同プロジェクトの 1 つとして DL_MESO が選ばれた理由です。[1] IPCC プロジェクトは、最新の手法を利用して最新のシステムに対するコードの現代化を行っています。IPCC プロジェクトでコードの現代化を支援する新しいテクノロジーを採用するのはごく自然なことです。

DL_MESO のエンジニアは、2015 年の初めに、ベクトル化アドバイザー・ツールの早期プレリリース・バージョンを利用しました。(ベクトル化アドバイザーは現在、インテル® Parallel Studio XE に含まれるインテル® Advisor XE のコンポーネントとして提供されています。)

新しいテクノロジーへの関心は、最近のインテル® プラットフォームのベクトル並列化機能を最大限に活用するという DL_MESO の開発者の意図によるものでした。マルチコア インテル® Xeon® プロセッサーやメニーコア インテル® Xeon Phi™ プラットフォームでは、次の**両方の**並列化を行った場合のみ、コードは優れたパフォーマンスを発揮します。

- マルチコア並列化
- ベクトルデータ並列化

512 ビット幅の SIMD (Single Instruction Multiple Data) 命令では、効率的にベクトル化されたコードの浮動小数点演算のパフォーマンスは、ベクトル化されていないコードの 8 倍 (倍精度) または 16 倍 (単精度) になります。DL_MESO の開発者は、このパフォーマンスを最大限に活用することにしました。

この記事では、Daresbury Lab の計算科学者である Michael Seaton と Luke Mason が、ベクトル化アドバイザーをどのように使用して DL_MESO の格子ボルツマン方程式 (LBE) コード [2] を解析したか説明します。

新しいマルチコア インテル® Xeon® プロセッサーやメニーコア インテル® Xeon Phi™ コプロセッサーでは、アプリケーションでマルチコア並列化とベクトルデータ並列化を併用することで、最適なパフォーマンスを実現できます。

アプリケーションのベクトル化手法はさまざまです。次にいくつかの例を示します。

- インテル® MKL のように、すでにベクトル化されているライブラリーを使用する。このアプローチの利点は、最適化されたライブラリー関数を使用することにより、コードのベクトル化に必要なプログラミング労力のほとんどを省略できる点です。
- コンパイラーの自動ベクトル化機能を使用する。これはインテル ® コンパイラーを使用する多くの開発者が利用していた従来の方法です。
- OpenMP* SIMD プラグマ / ディレクティブのようなプラグマまたはディレクティブを明示的に追加する。自動ベクトル化を使用する場合に比べて、低水準プログラミングを行うことなく、何をベクトル化するかより細かく制御できるため、このオプションを選択する開発者が増えています。
- ベクトル組込み関数、C++ ベクトルクラス、アセンブリー命令を使用して、ベクトル対応のコードを挿入する。ベクトル化をサポートする関数と命令に関する豊富な知識が必要です。この方法で記述されたコードは前述したほかの手法よりも移植性が大幅に低下します。

どの方法を選んだ場合でも、プロセッサーのベクトルユニットを効率的に利用することが重要です。DL_MESO ライブラリーの場合、Daresbury のプログラマーは OpenMP* 4.x の機能を使用してベクトル化のパフォーマンスを向上しました。

ベクトル化アドバイザー

ベクトル化アドバイザーとスレッド化アドバイザーは、インテル® Advisor XE の主要な機能です。ベクトル化アドバイザーは、次のような状況でユーザーを支援する解析ツールです。

- ベクトル化されなかったループについて、コードのベクトル化を妨げている原因を特定し、コードをベクトル化する方法についてのヒントが得られます。
- 最近の **SIMD** 命令を使用してベクトル化されたループについて、パフォーマンス効率を測定し、効率を向上する方法についてのヒントが得られます。
- ベクトル化されたループとベクトル化されなかったループの両方について、より効率的にベクトル化できるようにメモリーレイアウトとデータ構造を見直します。

ベクトル化アドバイザーは、任意のコンパイラーとともに使用できますが、インテル® コンパイラーと組み合わせたときに最高の結果が得られます。ベクトル化アドバイザーは、インテル® コンパイラーが生成したさまざまなレポートを見やすく表示するだけでなく、コンパイル時解析の結果、関連するバイナリーの統計解析、CPU の hotspot やループのトリップカウントのようなランタイム・ワークロード・メトリックも提供します。

また、これらのスタティック解析とダイナミック解析とともに、最適化に役立つ推奨事項が提供されます。ベクトル化アドバイザーを利用することで、インタラクティブなフィードバックと豊富なダイナミック・バイナリー・プロファイルにより、コンパイル時 (スタティック) からランタイム (ダイナミック) までの詳細な情報が得られます。[3]

BLOG HIGHLIGHTS

コードの現代化エキスパートによる IDF プレゼンテーションのレポート

KATHY FARREL >

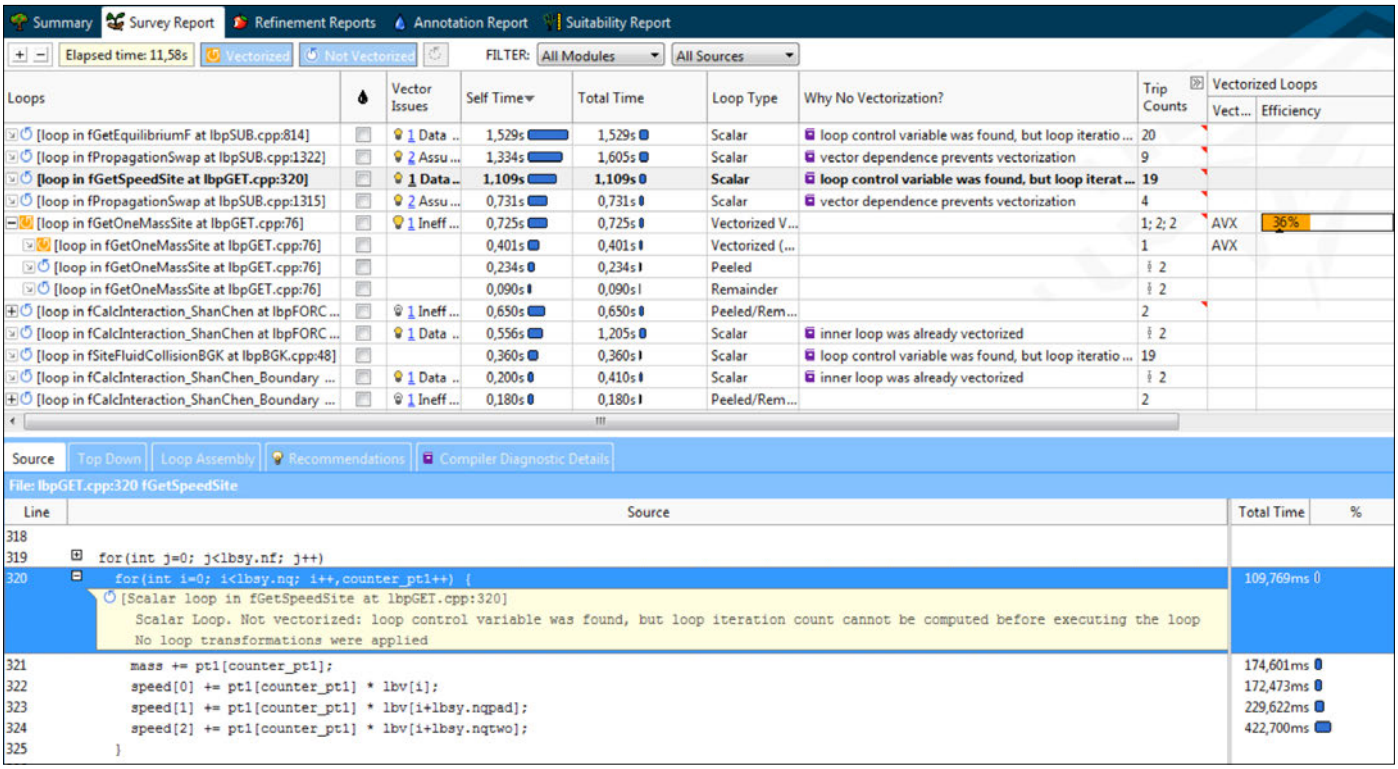
2015 年 8 月 18 日に、私は IDF 2015 でインテルの主席エンジニア (Robert Geva) と 3 人のソフトウェア開発者 (Clay Breshears、Tom Murphy、Gaston Hillar) による、コードの現代化エキスパート討論会の司会を行いました。このプレゼンテーションの目的は、現在、そして今後、ハードウェアに搭載されている多くのコアを活用するには、並列プログラミング手法を利用する必要があることを開発者に理解してもらうことでした。

この記事の続きはこちら (英語) でご覧になれます。 >

アドバイザーの調査 : DL_MESO パフォーマンスの概要を一目で表示

ベクトル化アドバイザーのユーザー・インターフェイスは、主要なベクトル化機能が 1 個所にまとめられています。**図 2** は、インテル® Advisor XE のベクトル化調査とトリップカウント機能による格子ボルツマンの初期解析結果です。

調査結果から、合計実行時間の約半分が上位 10 の hotspot で費やされており、特に目立つ hotspot がないことがわかります。時間を費やしているループをすべて合計しても、プログラム時間の 12% 未満に過ぎません。このようなプロファイルは、比較的フラットとして分類されます。ワークロードを大幅にスピードアップするには多くの hotspot を個別にプロファイルして最適化する必要があるため、フラット・プロファイルは通常、ソフトウェア開発者にとって良い傾向ではありません。ソフトウェア・ツールを利用しないと、この作業には時間がかかります。



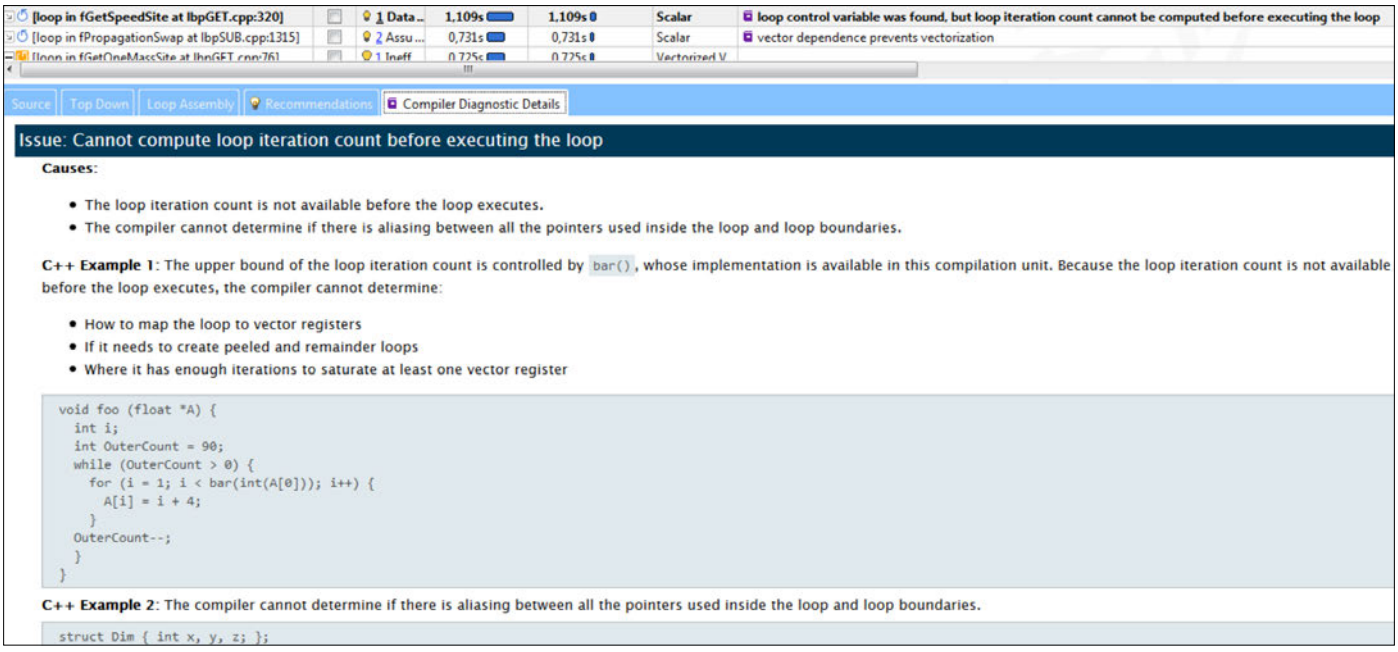
2 調査とトリップカウント

ベクトル化アドバイザーは、hotspot を次のカテゴリーに分類します。

- ベクトル化可能だがベクトル化されなかったループで、コンパイラーによる SIMD 並列処理を有効にするために最小限のプログラム変更 (ほとんどは OpenMP* 4.x の機能を利用) が必要なもの。調査リストの上位 4 つの hotspot は、このカテゴリーに属します。
- ベクトル化されたループで、簡単な最適化手法によりパフォーマンスを向上できるもの。
- ベクトル化されたループで、データレイアウトによりパフォーマンスが制限されているもの (パフォーマンスを向上するにはコードのリファクタリングが必要)。後述するように、上記の 2 つのカテゴリーに対応する手法を適用すると、1 つ目と 2 つ目の hotspot はこのカテゴリーに移行します。
- ベクトル化されたループで、適切に実行されているもの。
- 上記以外の場合 (ベクトル化できないカーネルを含む)。

ベクトル化アドバイザーは、ループに関する情報を提供するだけでなく、[Recommendations (推奨)] タブと [Compiler Diagnostic Details (コンパイラー診断詳細)] タブに特定の問題および問題の解決方法に関する詳細な情報を表示します。

このケースでは、3 つ目の hotspot、**`fGetSpeedSite`** は、コンパイラーがループの反復回数を判断できなかったためベクトル化されませんでした。**図 3** に、アドバイザーの [Compiler Diagnostic Details (コンパイラー診断詳細)] ウィンドウを示します。この問題の解決方法の例と推奨事項が表示されています。表示された推奨事項に従うことで、ループを簡単にベクトル化でき、ループはカテゴリー 2 からカテゴリー 4 に移行しました。



3 アドバイザーの調査の [Compiler Diagnostics Details (コンパイラー診断詳細)] ウィンドウ

コードをベクトル化できる場合でも、ベクトル化が必ずしもパフォーマンスの向上につながるとは限りません (ベクトル化されたループがカテゴリー 2 および 3 の場合)。そのため、すでにベクトル化されているループを調べて適切に動作していることを確認することが重要です。次のセクションでは、適切にベクトル化されていないループにベクトル化アドバイザーを利用して Daresbury Lab が達成した最適化について簡単に説明します。

簡単な方法 : 平衡分布計算カーネルのパディング手法

DL_MESO プロファイルの最もホットなループのコードを**図 4** に示します。

配列 **lbv** は、各次元の格子の速度とループカウント変数 **lbsy.nq** (速度の数) を格納します。このケースでは、モデルは 3 次元で 19 速度の格子 (D3Q19) を表しているため、**lbsy.nq** の値は 19 です。結果の平衡は配列 **feq[i]** に格納されます。

最初の解析では、このループはスカラーループとしてレポートされました。つまり、コードはベクトル化されませんでした。ループの先頭に **#pragma omp simd** を追加するだけで、ループはベクトル化され、合計実行時間は 13% から 9% に減りました。この追加を行った後も、最適化の余地はまだ残されています。

```
int fGetEquilibriumF(double *feq, double *v, double rho)
{
 double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
 double uv;

 for(int i=0; i<lbsy.nq; i++)
 {

 uv = lbv[i*3] * v[0]
 + lbv[i*3+1] * v[1]
 + lbv[i*3+2] * v[2];

 feq[i] = rho * lbw[i]
 * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);
 }
 return 0;
}
```

4 平衡分布を計算しているループのコード

ベクトル化アドバイザーは、コンパイラーが 1 つではなく 2 つのループを生成したことを示しました。

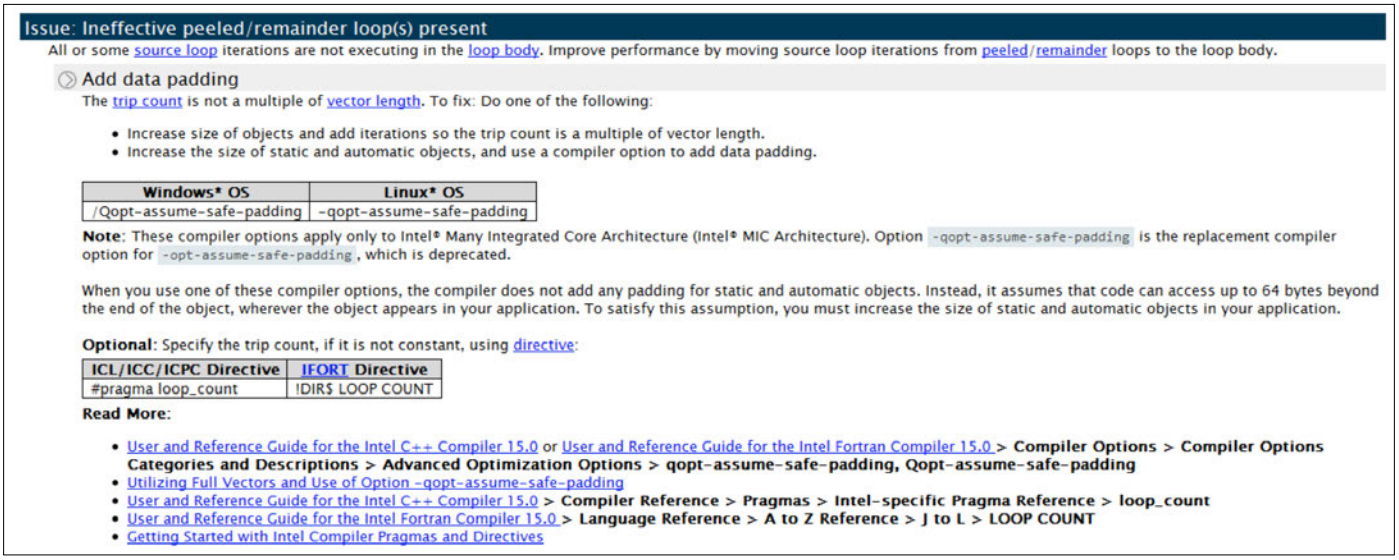
- ベクトル長 4 の (256 ビット幅のインテル® AVX レジスターに 4 つの倍精度浮動小数点を保持する) ベクトル化されたループ本体。
- ループ時間の約 30% を費やしているスカラー・リマインダー。

スカラー・リマインダーは不要なオーバーヘッドです。このリマインダー・ループは、並列化の効率 (つまり、達成できる最大のスピードアップ) を下げます。大きなリマインダー・ループのオーバーヘッドは、ベクトル長 (VL) の倍数でないループ (トリップ) カウントにより引き起こされます。ループをベクトル化すると、コンパイラーはベクトル化したループ本体 (このケースでは、ループ反復 0 から 15 を実行) を生成します。残りの 3 つの反復 (16 から 18) はスカラー・リマインダー・コードで実行されます。合計ループカウントが非常に少ないため、残りの 3 つの反復がループの経過時間の大部分を占めます。理想的に最適化されたトリップカウントが低いループには、リマインダー・コードはありません。

このコードに適用できる 1 つの手法は、VL の倍数 (このケースでは 20) になるようにループの反復回数を増やすことです。データパディングと呼ばれるこの手法は、まさにこのループの [Recommendations (推奨)] ウィンドウ (**図 5**) でベクトル化アドバイザーが推奨しているものです。データパディングを行うには、(使用していない) 20 番目の場所にアクセスしたときにセグメンテーション・フォルトなどの問題を引き起こさないように、配列 **`feq[]`**、**`lbv[]`**、**`lbw[]`** のサイズを増やす必要があります。

図 11 の 2 列目に必要な変更の例を示します。値 lbsy.nqpad は、オリジナルループのトリップカウント (lbsy.nq) とパディング値 (NQPAD_COUNT) の合計です。

DL_MESO の開発者は **`#pragma loop count`** ディレクティブも追加しています。ループカウントをコンパイラーに知らせることにより、コンパイラーはカウントがベクトル長の倍数であることを理解し、スカラー・リマインダー呼び出しがランタイムに省略されるように特定のトリップカウント値向けにコードを最適化します。



Issue: Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Add data padding

The trip count is not a multiple of vector length. To fix: Do one of the following:

- Increase size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

Windows* OS	Linux* OS
/Qopt-assume-safe-padding	-qopt-assume-safe-padding

Note: These compiler options apply only to Intel® Many Integrated Core Architecture (Intel® MIC Architecture). Option `-qopt-assume-safe-padding` is the replacement compiler option for `-opt-assume-safe-padding`, which is deprecated.

When you use one of these compiler options, the compiler does not add any padding for static and automatic objects. Instead, it assumes that code can access up to 64 bytes beyond the end of the object, wherever the object appears in your application. To satisfy this assumption, you must increase the size of static and automatic objects in your application.

Optional: Specify the trip count, if it is not constant, using directive:

ICL/ICC/ICPC Directive	IFORT Directive
#pragma loop_count	!DIR$ LOOP COUNT

Read More:

- User and Reference Guide for the Intel C++ Compiler 15.0 or User and Reference Guide for the Intel Fortran Compiler 15.0 > **Compiler Options > Compiler Options Categories and Descriptions > Advanced Optimization Options > qopt-assume-safe-padding, Qopt-assume-safe-padding**
- Utilizing Full Vectors and Use of Option -qopt-assume-safe-padding
- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference > loop_count**
- User and Reference Guide for the Intel Fortran Compiler 15.0 > **Language Reference > A to Z Reference > J to L > LOOP COUNT**
- Getting Started with Intel Compiler Pragmas and Directives

5 ベクトル化アドバイザーはデータパディングを推奨

DL_MESO コードには、同じ方法で変更できる同様の平衡分布コード構造が多く含まれています。この例では、同じソースファイルの 3 つのほかのループを変更して、各ループで 15% のスピードアップを達成しました。

オーバーヘッドと最適化のトレードオフのバランス

最初の 2 つのループに使用したパディング手法には、パフォーマンスとコード保守のコストがかかります。

- パフォーマンスの点から見ると、パディングによりスカラー部分のオーバーヘッドは回避されますが、ベクトル部分に余計な計算が追加されます。
- コード保守の点から見ると、データ構造の割り当てを見直す必要があります。ワークロード別のプラグマ定義も必要になるでしょう。

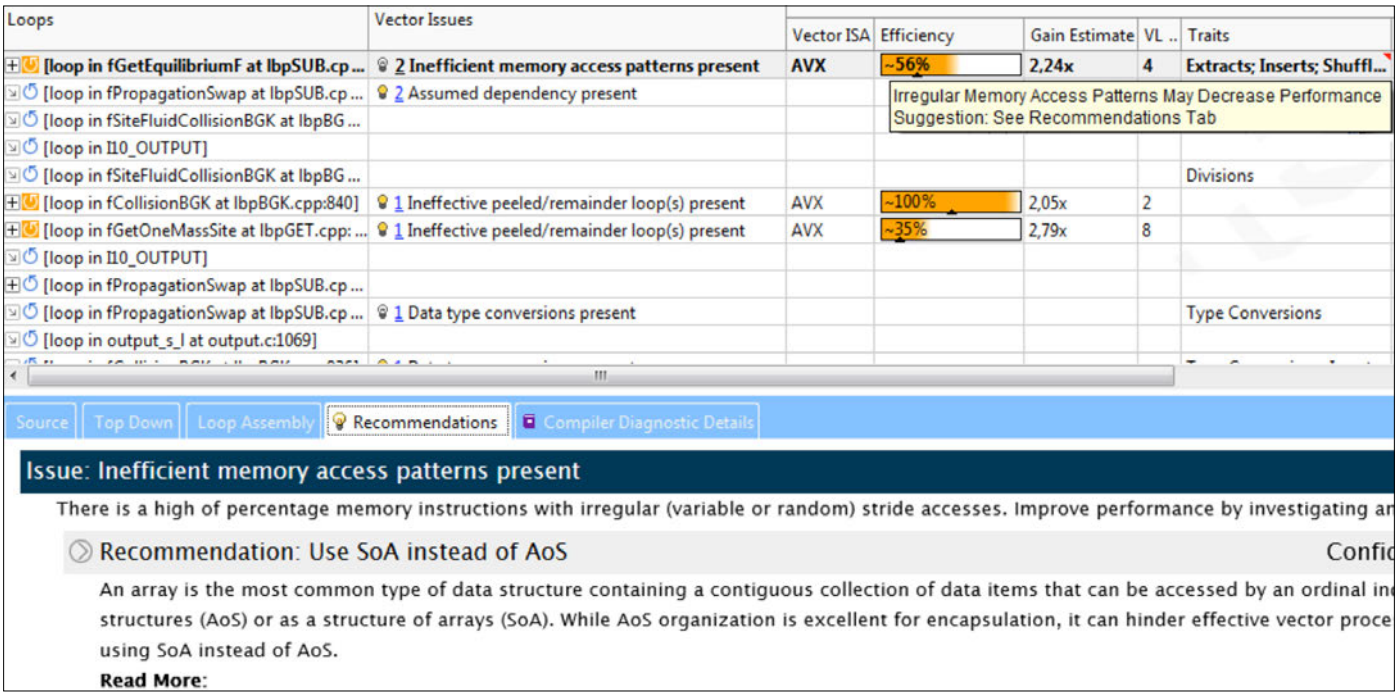
幸い、このケースでは、パフォーマンスの向上がパフォーマンスの低下よりも大きく、コード保守のコストもそれほどかかりません。

MAP は、非効率なメモリー・アクセス・パターンを細かく識別して分類できる、より詳細なベクトル化アドバイザーの解析です。

データレイアウトの変換

ベクトル化、ループのパディング、データ・アライメント手法により 1 つ目の hotspot のパフォーマンスは 25 ～ 30% 向上し、ベクトル化アドバイザーに表示されるベクトル並列化の効率[4] は 56% になりました。

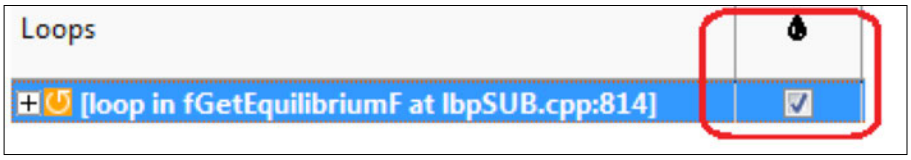
56% は理想 (100%) にはまだほど遠いため、Daresbury Lab の開発者は、ループ効率の向上を妨げている原因をさらに調査することにしました。[Vector Issues (ベクトル問題)] と [Recommendation (推奨事項)] の内容を再確認したところ、[Vector Issues (ベクトル問題)] 列に新しい問題 (非効率なメモリー・アクセス・パターンの可能性) がハイライトされていました。そして、メモリー・アクセス・パターン (MAP) 解析の実行が推奨されていました。低水準命令セット・アーキテクチャー・レベルの特性解析でも、同様の推奨事項が表示されました (**図 6**)。



6 非効率なメモリー・アクセス・パターン問題、特性、推奨事項

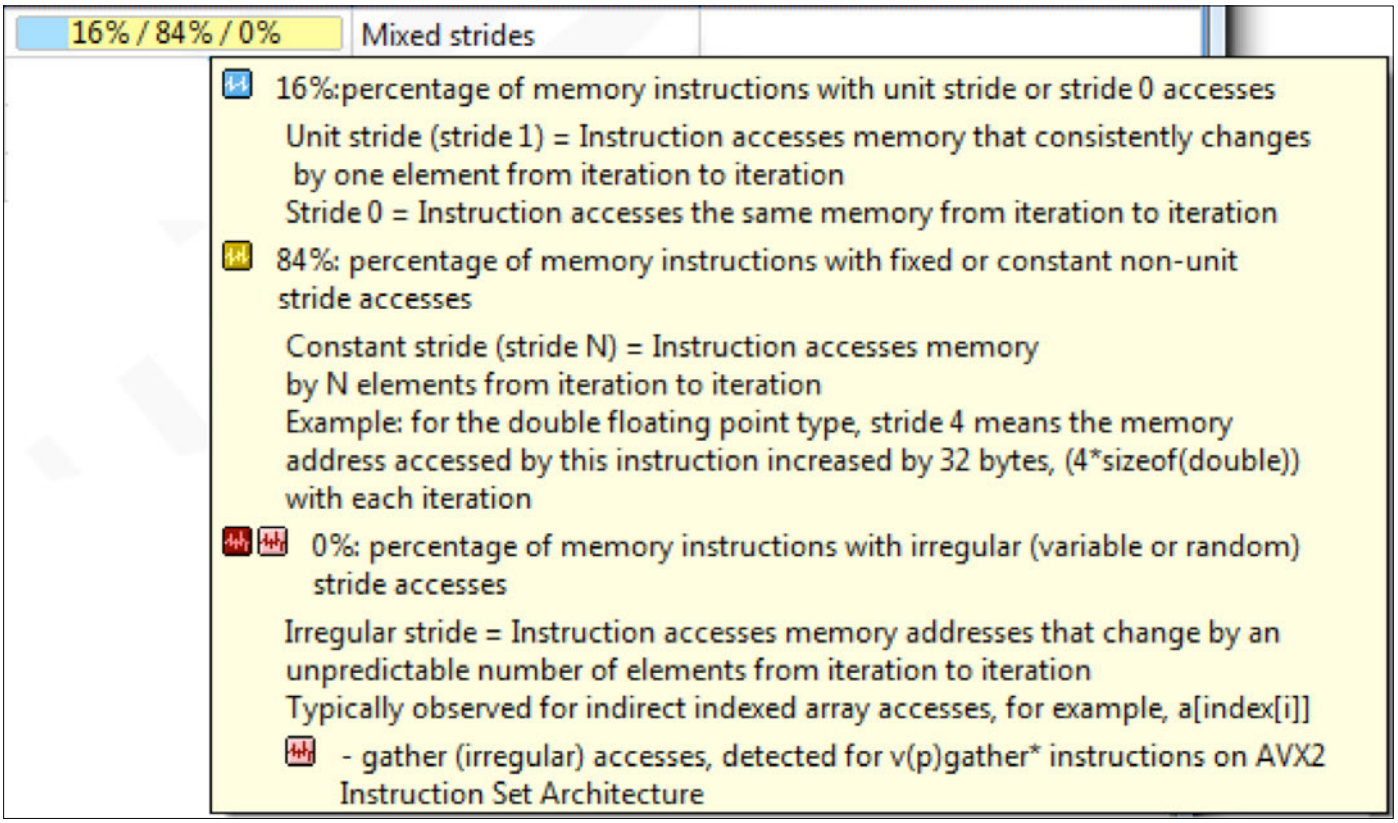
MAP は、非効率なメモリー・アクセス・パターンを細かく識別して分類できる、より詳細なベクトル化アドバイザーの解析です。DL_MESO の最適化では、次の GUI ベースの方法で MAP ツールを実行しました。

- 最初に、調査の 2 つ目の列のチェックボックスをオンにして、730 行目のループをマークしました (**図 7**)。
- 次に、ワークフロー・パネルを使用して MAP コレクションを実行しました。



7 MAP 解析の結果

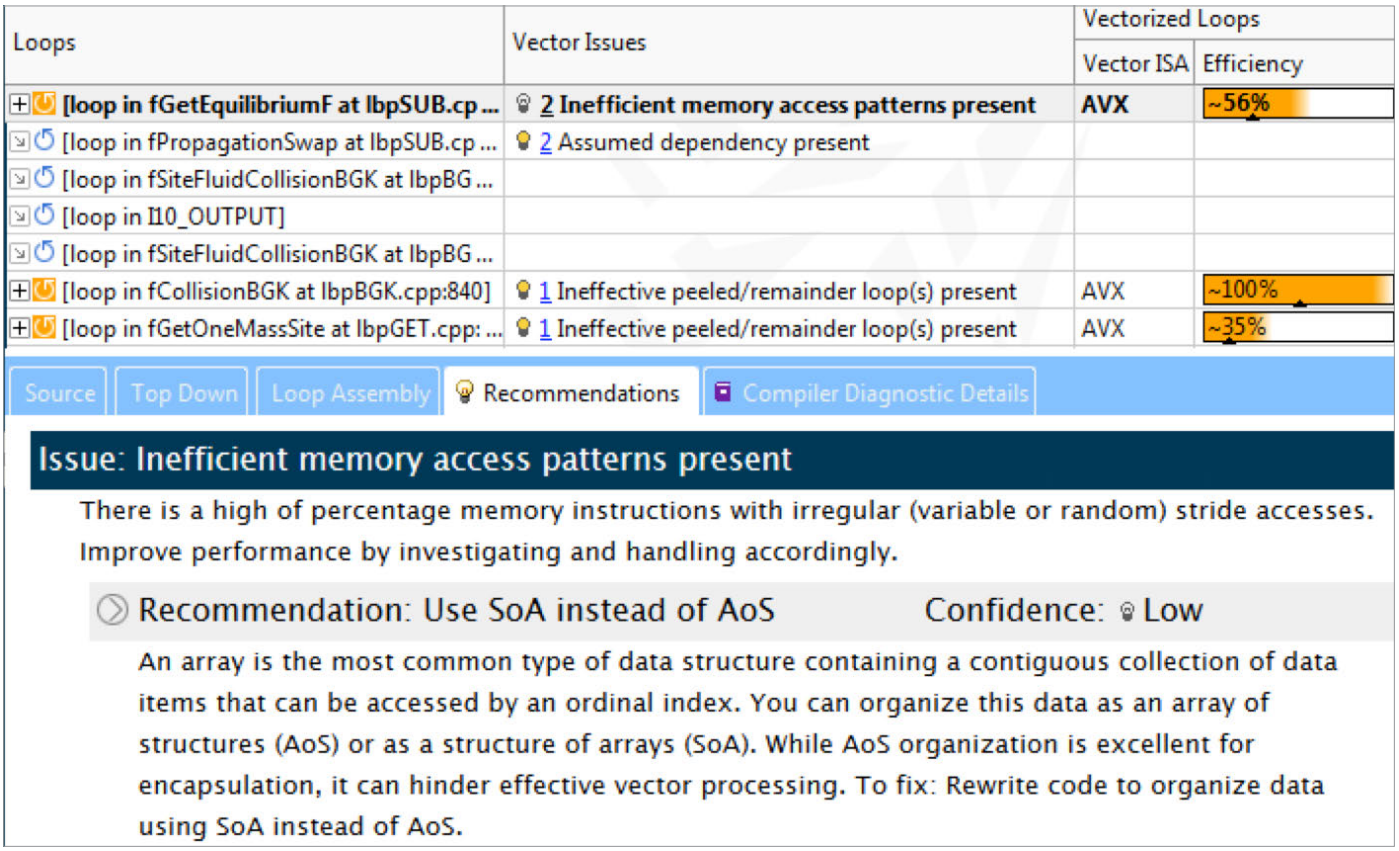
MAP 解析の結果、測定された高レベルのストライド分布は、ユニットストライド形式と非ユニット定数ストライド形式のアクセスがループで行われていることを示しました (**図 8**)。MAP の [Problems (問題)] ビューと [Source (ソース)] ビューを詳細に調べると、lbv 配列の操作に Stride-3 アクセス (オリジナル・スカラー・バージョンの場合) や Stride-12 アクセス (パディングを使用してベクトル化されたループの場合) が存在することが分かりました。



8 ストライド分布ループ解析と対応するツールヒント (ストライド分類の説明を含む)

定数ストライドは、反復から反復で配列要素へのアクセスを予測できるものの、非線形の方法でシフトされます。このケースでは、整数要素の lbv 速度配列への Stride-3 アクセスは、次の反復で lbv 配列へのアクセスが 3 つの整数要素分シフトされることを意味します。対応する式は `lbv[i*3+X]` であるため、3 という値は意外なものではありません。

連続していない定数ストライドが存在すると、ベクトル化されたコードバージョンでは、単一のパックドメモリー移動命令ですべての配列要素をベクトルレジスターにロードできないため、連続していない定数ストライドはベクトル化に適していません。[5] 一方、定数ストライドアクセスは、多くの場合、[構造体配列] から [配列構造体] への (AoS → SoA) 変換手法により、ユニット (つまり、連続) ストライドアクセスに変換できます。[6] MAP 解析を実行すると、`fGetEquilibriumF` のループに対する推奨事項が AoS → SoA 変換の適用に自動的に変わります (**図 9**)。



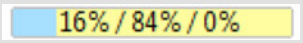
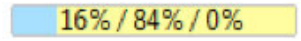
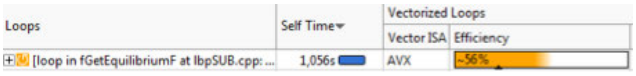
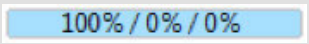
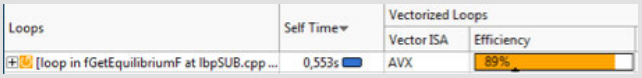
9 非効率なメモリー・アクセス・パターン問題と推奨事項

Daresbury Lab のエンジニアは、**`lbv`** 配列のデータレイアウトを最適化することにしました。これが **`fGetEquilibriumF`** のループに適用された最後の最適化でした。この変換を行うため、(X、Y、Z 次元の速度を含む) 単一の lbv 配列を 3 つの配列 (**`lbvx`**、**`lbvy`**、**`lbvz`**) に置換しました。

図 10 (ベクトル化アドバイザーの効率と MAP メトリックを含む) および **図 11** は、パディングと AoS → SoA 変換を適用した DL_MESO のすべてのループとデータ構造変換の要約です。

リファクタリングは時間がかかり (パディングのように) 簡単な作業ではありませんでしたが、その結果得られたスピードアップは、それだけの価値があるものでした、とエンジニアは述べています。

DL_MESO のエンジニアは、リファクタリングは時間がかかり (パディングのように) 簡単な作業ではありませんでしたが、その結果得られたスピードアップは、それだけの価値があるものでしたと述べています。**`fGetEquilibriumF`** のループは、すでに最適化されていたバージョンから、さらに 2 倍スピードアップしました。**`lbv`** 配列を操作するほかのループでも、同様のスピードアップが見られました。

実装バージョンと最適化手法	所要時間 (秒)	アドバイザーのベクトル化効率	非ユニットストライド方式のメモリーアクセス命令の比率 アドバイザーのストライド分布	速度配列のアクセスパターン (アドバイザーのソースビュー)
#1: なし	1.529	N/A	73% 	Stride-3
				
#2: 明示的なベクトル化	1.150	38%	84% 	Stride-12
#3: 明示的なベクトル化 + パディング/アライメント	1.056	56%	84% 	Stride-12
				
#4: 明示的なベクトル化 + パディング/アライメント+ 配列構造体	0.553	89%	0% 	Stride-1 (ユニットストライド)
				

10 パディングとデータレイアウトの変換 (AoS → SoA) が **`fGetEquilibriumF`** の for ループに与える影響

実装バージョンと最適化手法	データレイアウト (割り当て)	ループの実装

#1: 明示的なベクトル化	`lbv = new double[3*lbsy.nq];`	`for(int i=0; i<lbsy.nq; i++)`
		`{`
		`uv =`
		`lbv[i*3] * v[0] +`
		`lbv[i*3+1] * v[1] +`
		`lbv[i*3+2] * v[2];`
		`feq[i] = rho * lbw[i] * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);`
		`}`

Line	Source	Stride	Operand Type	Vector Length
733	uv = lbv[i*3] * v[0] + lbv[i*3+1] * v[1] + lbv[i*3+2] * v[2];	3;	float64;int;int32	1;

#3: 明示的なベクトル化 + パディング/アライメント	`Lbsy.nqpad = lbsy.nq + NQPAD_COUNT;`
	`lbv = (int*) _mm_malloc(3*lbsy.nqpad*sizeof(double), ALIGNMENT);`
		`__assume_aligned(lbv, ALIGNMENT);`
		`#pragma vector aligned`
		`#pragma simd private(uv,rho,modv)`
		`#pragma loop count (NQPAD_COUNT)`
		`for(int i=0; i<lbsy.nqpad; i++)`
		`{`
		`uv =`
		`lbv[i*3] * v[0] +`
		`lbv[i*3+1] * v[1] +`
		`lbv[i*3+2] * v[2];`
		`feq[i] = rho * lbw[i] * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);`
		`}`

Line	Source	Stride	Operand Type	Vector Length
733	uv = lbv[i*3] * v[0] + lbv[i*3+1] * v[1] + lbv[i*3+2] * v[2];	12; 12;	float64;int32;int8	1;4

#4: 明示的なベクトル化 + パディング/アライメント+ 配列構造体	`lbsy.nqpad = lbsy.nq + NQPAD_COUNT;`
	`lbvx = (int*) _mm_malloc(lbsy.nqpad*sizeof(double),ALIGNMENT);`
	`lbvy = (int*) _mm_malloc(lbsy.nqpad*sizeof(double),ALIGNMENT);`
	`lbvz = (int*) _mm_malloc(lbsy.nqpad*sizeof(double),ALIGNMENT);`
		`__assume_aligned(lbvx, ALIGNMENT);`
		`__assume_aligned(lbvy, ALIGNMENT);`
		`__assume_aligned(lbvz, ALIGNMENT);`
		`#pragma vector aligned`
		`#pragma simd private(uv,rho,modv)`
		`#pragma loop count (NQPAD_COUNT)`
		`for(int i=0; i<lbsy.nqpad; i++)`
		`{`
		`uv =`
		`lbvx[i] * v[0] +`
		`lbvy[i] * v[1] +`
		`lbvz[i] * v[2];`
		`feq[i] = rho * lbw[i] * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);`
		`}`

Line	Source	Stride	Operand Type	Vector Length
733	uv = lbvx[i] * v[0] + lbvy[i] * v[1] + lbvz[i] * v[2];	[1]	float64;int32	4

11 `fGetEquilibriumF` のループのパディングおよびデータレイアウト変換 (AoS → SoA) に使用したデータ割り当て、ループ実装、アドバイザーのストライドデータ

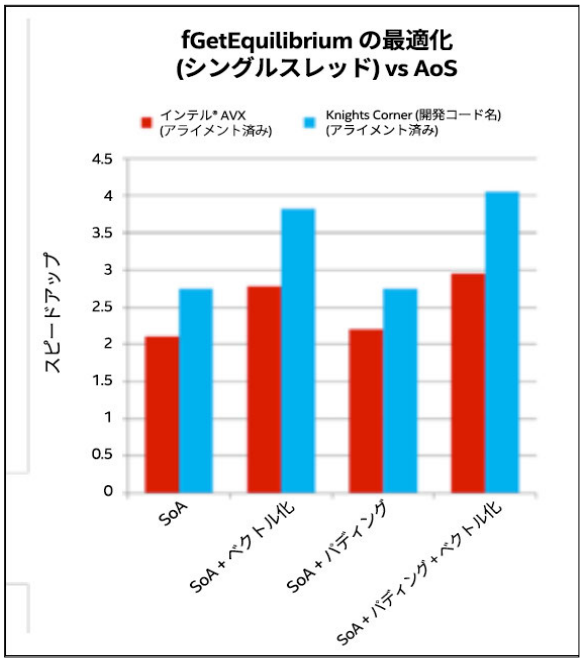
まとめ

ベクトル化アドバイザーを使用してコードにプラグマを追加することにより、Hartree Centre は上位 3 つの hotspot から 10 ～ 19 % の時間を減らすことができました。すべての最適化は、ベクトル化アドバイザーの推奨事項に基づいて行われました。作業には、ベクトル化を有効にすることと、パディング手法によりループのパフォーマンスを向上することが含まれていました。同様の手法をほかのいくつかの hotspot に適用することにより、アプリケーション全体でベクトル化効率が 18% 向上しました。

また、ベクトル化アドバイザーの推奨事項に基づいて一部の変数のデータレイアウトを AoS から SoA に変更することにより、さらにパフォーマンスが向上しました。

この作業を行った時点では、ベクトル化アドバイザーはインテル® Xeon® プロセッサーでのみ利用可能でしたが、同じ最適化をインテル® Xeon Phi™ コプロセッサーで実行するコードに適用したところ、同様のスピードアップが得られました。

図 12 は、**`fGetEquilibriumF`** (主な hotspot 関数の 1 つ) にさまざまな最適化を適用した場合の、インテル® Xeon® プロセッサー・ベースのサーバー (図のインテル® AVX) とインテル® Xeon Phi™ コプロセッサー (図の Knights Corner [開発コード名]) におけるスピードアップを示しています。これらの最適化によるスピードアップは、インテル® Xeon® プロセッサーで 2.5 倍、インテル® Xeon Phi ™ コプロセッサーで 4.1 倍になりました。

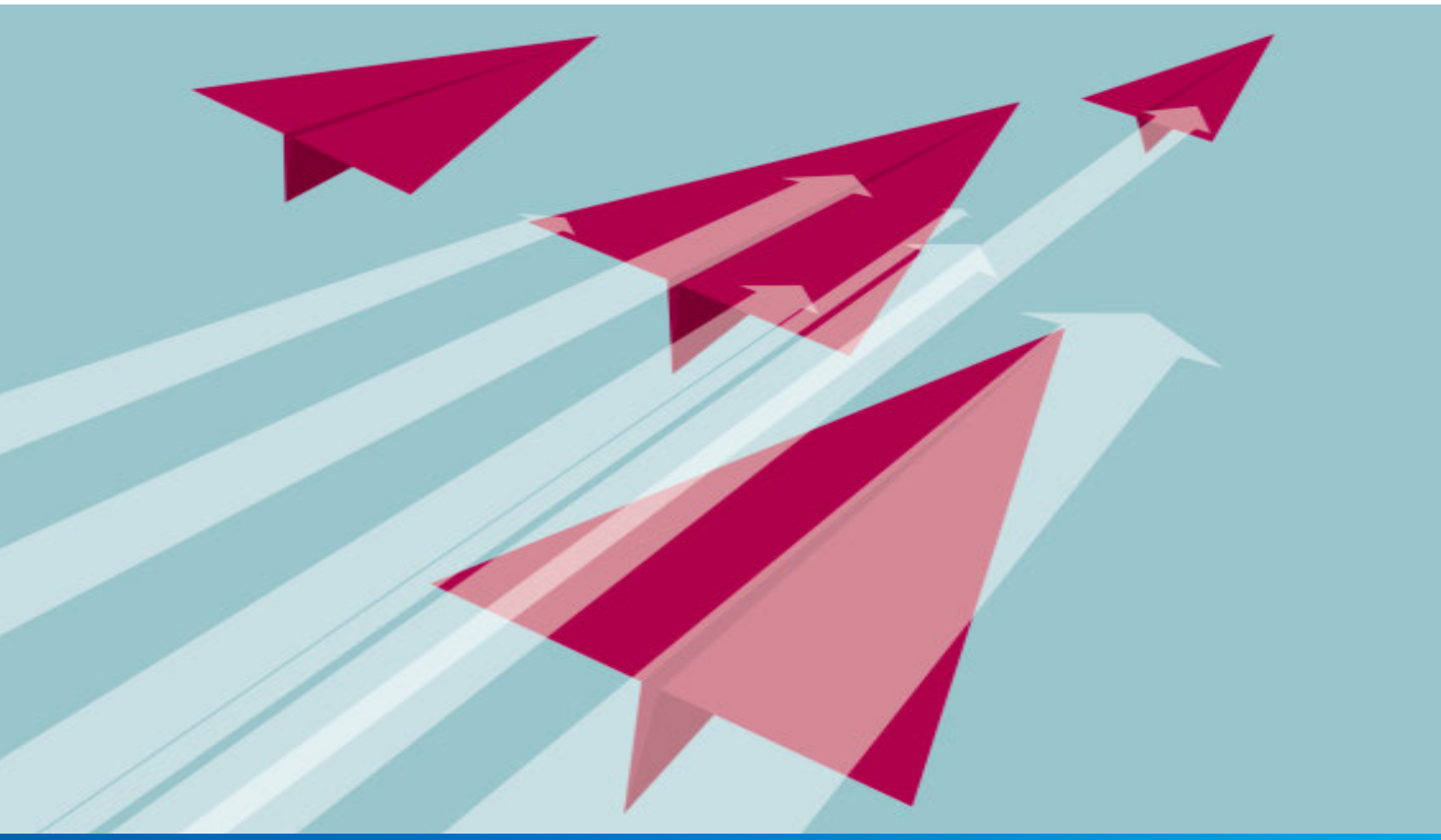


12 さまざまな最適化を適用したときのスピードアップ (大きいほうが良い)

エンジニアたちは、ベクトル化アドバイザーが DL_MESO コードの大幅なスピードアップに役立ったことを喜びました。中心となった開発者の 1 人は、「このツールを非常に気に入りました。今後も、このツールはインテル® Xeon Phi™ コプロセッサーでの作業に大いに役立つでしょう。」と述べています。

参考文献

1. **https://software.intel.com/en-us/articles/intel-parallel-computing-center-at-hartree-centre-stfc** (英語)
2. DL_MESO は、格子ボルツマン方程式 (LBE) および散逸粒子動力学 (DPD) 法を実装する 2 つのパッケージから構成されます。LBE パッケージは、複数の流体成分、溶質、連動熱伝達を含む格子ガス系のシミュレーションをサポートします。
3. ベクトル化アドバイザーでフルセットの解析データを収集するにはインテル® コンパイラーが必要です。しかし、ほかのコンパイラーでビルドされたバイナリーでもメトリックのサブセットを利用することができます。
4. アドバイザーのベクトル化効率メトリックは、現在、インテル® コンパイラー 16.x (2016) リリースを使用してコンパイルしたコードをプロファイリングする場合にのみ利用できます。
5. ストライドについては、**https://software.intel.com/en-us/videos/memory-access-101** (英語) および **https://software.intel.com/en-us/videos/stride-and-memory-access-patterns** (英語) のビデオを参照してください。
6. 配列構造体については、**https://software.intel.com/en-us/articles/a-case-study-comparing-aos-arrays-of-structures-and-soa-structures-of-arrays-data-layouts** (英語) の記事を参照してください。



画像処理の最適化

インテル® C++ コンパイラーとインテル® IPP により JD.com の画像処理が 17 倍にスピードアップ

Yanfeng Mu インテル China アプリケーション・エンジニア
Feilong Huang インテル APAC R&D テクニカル・コンサルティング・エンジニア
Ying Hu インテル APAC R&D テクニカル・コンサルティング・エンジニア

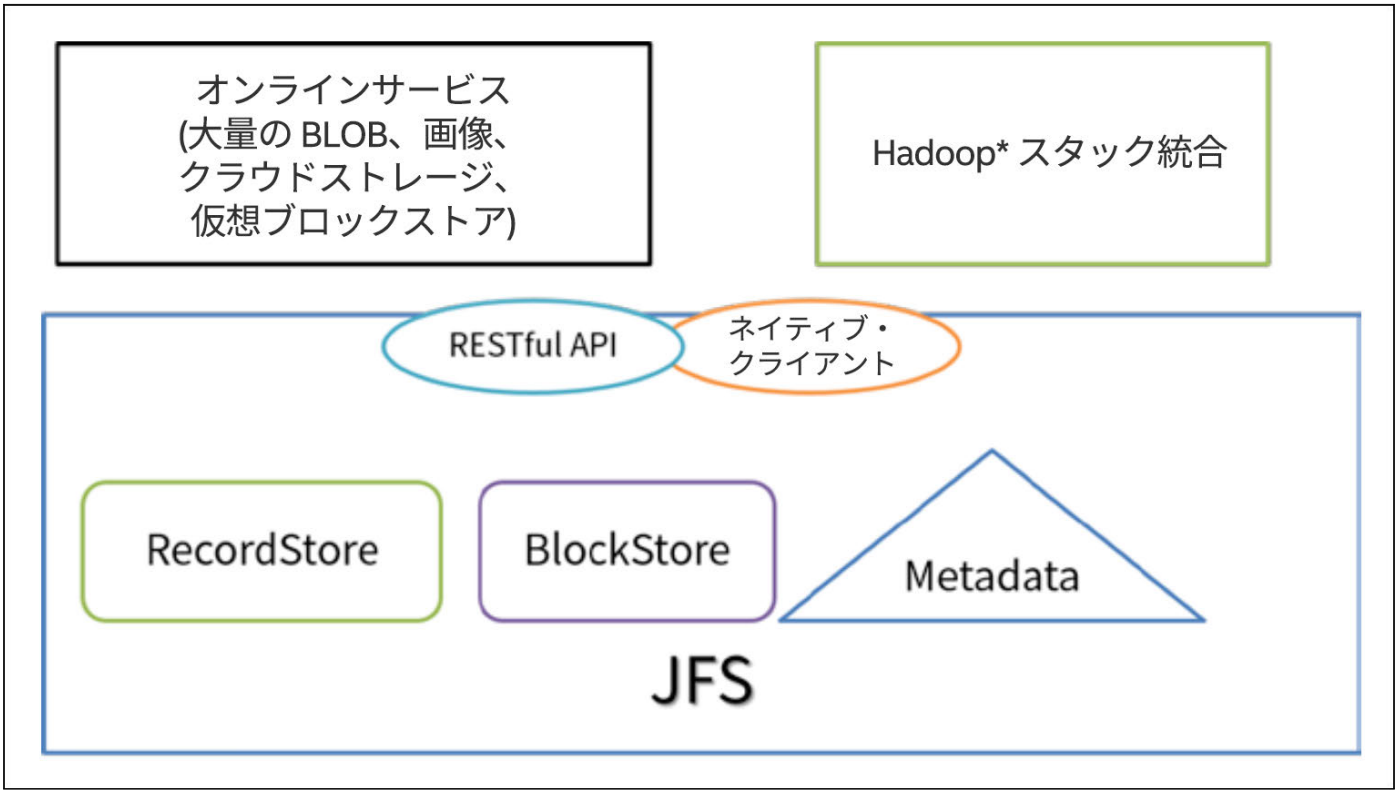
中国最大のオンライン直販企業である JD.com では、毎日数十億の製品画像を処理しています。この膨大な処理を行うために、JD.com では独自の分散型ファイルシステム、JD File System (JFS) を開発しました。インテル® C++ コンパイラーとインテル® インテグレーテッド・パフォーマンス・プリミティブ (**インテル® IPP**) により、JD.com の画像処理速度は 17 倍 に向上 (30 万画像の処理を 2,800 秒から 162 秒に短縮) しました。

ビジネス要件

JD.com は、中国のほかのどの企業よりも多くのオンライン・トランザクションを処理しており、サードパーティーの市場調査会社 iResearch によれば、2014 年第 2 四半期の市場シェアは 54.3% に上ります。JD.com のビジネスは急成長しており、提供する SKU の数は 2011 年の約 150 万から 2013 年には約 2570 万へと大幅に増加しました。現在では、JD.com が扱うデータはペタバイトの領域に達しており、効率的で強固な分散型ファイルシステムが求められています。

画像システムの概要

膨大なデータを処理するため、JD.com では JFS (急成長している e コマースビジネスの強固な基盤を形成する分散型キャッシュシステムおよび高速キー値ストレージシステム) を開発しました。優れたパフォーマンス、信頼性、スケーラビリティーを実現するように設計された分散型ストレージシステムとして、JFS は 3 種類のインターフェイス (バイナリー・ラージ・オブジェクト・ストレージ、ファイル・システム・ストレージ、ブロックストレージ) を提供しています。JFS は、パブリック / プライベート・クラウド、画像システム、ロジスティクス・エクスチェンジ・プラットフォーム、インスタント・メッセージ・ファイル共有ストレージを含む、JD.com のコアサービスの多くをサポートしています。**図 1** に全体のアーキテクチャーを示します。



1 JFS のアーキテクチャー

製品画像は、JD.com のような e コマース Web サイトにおいて最も重要な項目の 1 つです。Web サイトでは実際に製品に触れたり、手に取ったり、匂いをかいだり、味わったりすることはできないため、画像が顧客と対話する唯一の手段となります。JD.com では、さまざまなフォーマット (JPG、GIF、PNG など) の高 / 中 / 低解像度画像を使用して製品を表示、説明しています。毎日、何十億もの製品画像を処理して格納するため、優れたパフォーマンスと処理能力が不可欠です。ストレージコストを削減するため、JD.com では通常、大きな画像を圧縮して小さくしています。

JD.com の大きな課題は、画像のサイズ変更、鮮明化、減色、特殊効果の追加のような、画像処理要件にいかに対応するかです。元々、すべての画像処理には GraphicsMagick (GM) と GNU* C コンパイラー (GCC) が使用されていましたが、大量の CPU 時間を費やすため JD.com のビジネスリソースに深刻な影響を与えていました。

インテルは、JD.com のエンジニアと協力して、インテル® アーキテクチャー・ベースのプラットフォームで画像処理アプリケーションのボトルネックを解析し、アプリケーションの最適化を行いました。一連の作業には、インテル® C++ コンパイラーとインテル® IPP を使用しました。

「インテルのエンジニアとの共同作業で、弊社の画像処理アプリケーションにインテル® C++ コンパイラーとインテル® IPP ライブラリーを採用しました。その結果、アプリケーションのパフォーマンスが大幅に向上し、運用コストも大幅に削減されました。これはテクノロジーによって価値が産み出される良い例と言えるでしょう。」

JD.com システム・テクノロジー部ディレクター
クラウド・プラットフォーム・チーフ・アーキテクト
Liu Haifeng 氏

インテルのツールで画像処理を高速化

インテル® C++ コンパイラーは、互換プロセッサーでも強力なパフォーマンスを提供しますが、インテル® Xeon® プロセッサーやインテル® Xeon Phi™ コプロセッサーを含む、最新のインテル製プロセッサー向けにさまざまな最適化機能を提供しています。特許取得済みの自動 CPU ディスパッチ機能は、現在動作しているプロセッサー向けにコードを最適化し、アプリケーションの実行時に検出したプロセッサー向けに最適化されたコードを実行します。インテル® C++ コンパイラーは、現在および以前の C/C++ 標準規格に加えて、C++11、C99、OpenMP* 4.0 を含む標準的な拡張を広範にサポートします。

インテル® IPP は、メディアおよびデータ処理向けの豊富なソフトウェア関数群を含むライブラリーです。画像処理を含む、広範な分野でよく使用される多くの関数を提供します。これらの関数は、インテルの SIMD 命令セットを使用してパフォーマンスを引き出せるように高度に最適化されています。

パフォーマンス・チューニング

GM は、88 を超える形式で画像の読み取り、書き込み、操作をサポートする強固で効率的なツールとライブラリーのコレクションを含む、オープンソースの画像処理システムです。JD.com の画像処理アプリケーションの基盤として、GM は常に大量の画像を処理しています。インテルとの共同作業で、JD.com は**インテル® ソフトウェア開発ツール**を用いて、GM と関連画像処理ライブラリーの最適化に大変な労力をかけました。

JD.com のエンジニアは、インテル® Xeon® プロセッサー E5-2620 ベースのシステムで実行する画像処理アプリケーションのパフォーマンス・ボトルネックと最適化が可能な場所の特定方法を求めていました。画像のサイズ変更はアプリケーションで一般的で最も時間を費やす操作の 1 つであるため、ベンチマークを使用してインテル® Xeon® プロセッサーにおけるアプリケーションのパフォーマンスを測定しました。このベンチマークは、500 の同時インスタンスで 30 万の画像ファイルのサイズ変更を行います。記録されるメトリックには、サイズ変更完了までの所要時間、CPU 使用率、1 秒あたりのサイズ変更された画像ファイル数、平均応答時間が含まれます。

JD.com のエンジニアは、GCC とインテル® C++ コンパイラーで GM と一部の画像処理ライブラリー (libturbojpeg、libpng、libwebp を含む) を再コンパイルしました。そして、インテル® Xeon® プロセッサー・ベースのシステムでアプリケーションのベンチマークを実行しました。ベンチマークの結果、インテル® C++ コンパイラーで生成されたライブラリーは GCC で生成されたライブラリーよりも 12 倍高速であることが示されました。インテル® C++ コンパイラーで生成されたライブラリーの CPU 使用率は (100% から) 94% に減りました。平均応答時間は 62% 減り、応答時間は 2,943 ミリ秒から 378 ミリ秒に減りました。QPS (1 秒あたりのクエリー数) は 100 から 1,322 に増えました。

JD.com のエンジニアは、画像のサイズ変更が GM の hotspot であることを突き止め、新しい関数 ScaleImage_ipp (画像を指定のサイズに変更する GM の ScaleImage 関数をインテル® IPP で最適化したバージョン) を実装しました。その機能は ScaleImage と本質的に同じです。オリジナルのコードの一部をいくつかの IPP 関数呼び出しに置き換えました。JD.com で GM により処理される画像の 95% 以上は BMP、JPEG、PNG であるため、ScaleImage_ipp 関数の現在の実装は、これら 3 つの形式のみサポートしています。サイズを変更する画像の形式を判断する ScaleImage 関数の呼び出し元に条件付きジャンプを追加し、BMP、JPEG、PNG 画像の場合は ScaleImage_ipp 関数を実行し、ほかの形式の場合は ScaleImage 関数を実行するようにしました。**表 1** は、ScaleImage 関数と ScaleImage_ipp 関数のコードを並べて示したものです。

オリジナルバージョン

```
MagickExport Image *ScaleImage(const Image *image,const
unsigned long columns,
              const unsigned long rows,ExceptionInfo
*exception)
{
 for (y=0; y < (long) scale_image->rows; y++)
  {
   q=SetImagePixels(scale_image,0,y,
      scale_image->columns,1);
   if (q == (PixelPacket *) NULL)
    break;
   if (scale_image->rows == image->rows)
    {
     /*
      新しいスキャン行を読み取る
     */
     p=AcquireImagePixels(image,0,i++,
      image->columns,1,exception);
     if (p == (const PixelPacket *) NULL)
      break;
     for (x=0; x < (long) image->columns; x++)
      {
       x_vector[x].red=p->red;
       x_vector[x].green=p->green;
       x_vector[x].blue=p->blue;
       x_vector[x].opacity=p->opacity;
       p++;
      }
    }
    ...
  }
 ...
 return(scale_image);
}
```

インテル® IPP を使用して最適化したバージョン

```
MagickExport Image *ScaleImage_ipp(const Image
*image,const unsigned long columns,
              const unsigned long rows,ExceptionInfo
*exception)
{
 ...

 /* インテル® IPP を使用してサイズ変更 */
 /* RGB 形式のイメージのみ処理*/
 IppStatus st;
 int channel = 4;
 blobSrc = ippsMalloc_8u(channel*(image->columns) *
   image->rows);
 dstBuf = ippsMalloc_8u(channel*columns * rows);

 DispatchImage(image, 0, 0, image->columns,
   image->rows, "RGBA", CharPixel, blobSrc, &srcExp);
 ...
 int interpolation =IPPI_INTER_NN;
 /* 補間 */
 st = ippiResizeGetBufSize(srcRoi, dstRoi,
channel,interpolation , &bufSize);
 pBuffer = ippsMalloc_8u(bufSize);
         ...
         st= ippiResizeSqrPixel_8u_C4R(blobSrc, srcSize,
srcWidthStep, srcRoi, dstBuf, dstWidthStep, dstRoi, x_
factor, y_factor, 0.0, 0.0,interpolation, pBuffer);
         scale_image = ConstituteImage(dstSize.width,
dstSize.height, "RGBA", CharPixel, dstBuf, &dstExp);
 ...
 return scale_image;
}
```

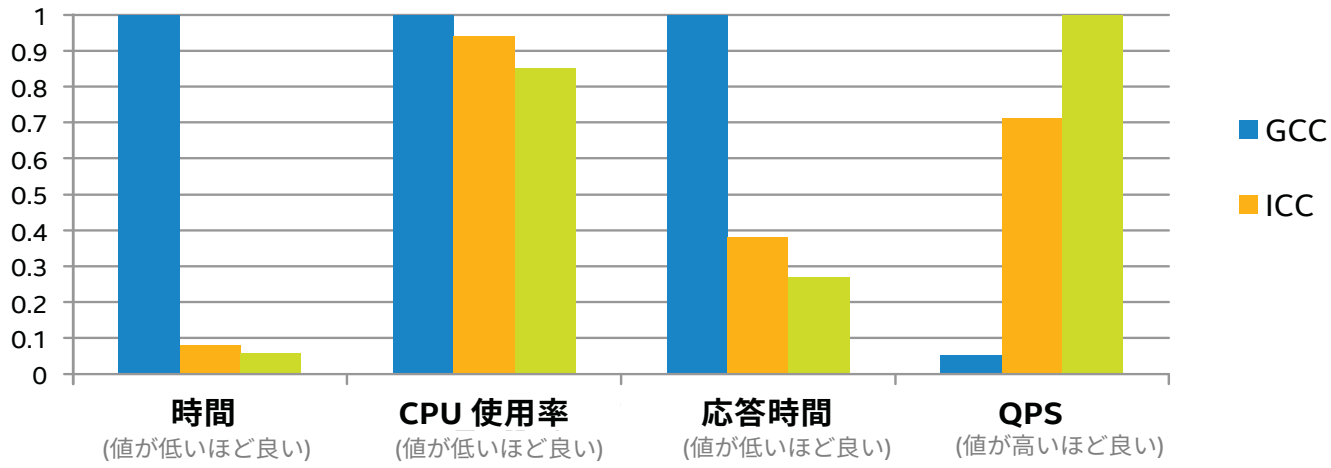
表 1. ScaleImage 関数と ScaleImage_ipp 関数のコードの比較

JD.com のベンチマーク結果 (**表 2**) から、ScaleImage_ipp 関数の使用によりパフォーマンスが大幅に向上したことが分かります。CPU 使用率は 85% に低下し、QPS は 1,856 に向上しました。これは、JD.com が同じ数のサーバーでより多くのリクエストを処理できるようになったことを意味します。また、JD.com の顧客は、Web ブラウザーでサイズ変更された画像をより速く表示できるようになります。

	GCC	インテル® C++ コンパイラー	インテル® C++ コンパイラーおよびインテル® IPP
時間 (秒)	2,800	226.867	161.589
CPU 使用率	100%	94%	85%
応答時間 (ミリ秒)	2,943	378.112	269.315
QPS	100	1,322	1,856

表 2. JD.com のベンチマーク結果

図 2 は、インテル® Xeon® プロセッサー・ベースのシステムでインテル® C++ コンパイラーとインテル® IPP を使用したソリューションを実行した場合のパフォーマンス向上率を示しています。



システム構成: ハードウェア: Inspur NF5270M3、インテル® Xeon® プロセッサー E5620 @ 2.40GHz、2 ソケット、64GB RAM、ハイパースレッディング有効。ソフトウェア: インテル® C++ コンパイラー 14.0.2、インテル® IPP 8.1、GCC 4.4.7。Linux* OS: CentOS* リリース 6.5 (Final)

性能に関するテストや評価は、特定のコンピューター・システム、コンポーネント、またはそれらを組み合わせて行ったものであり、このテストによるインテル製品の性能の概算の値を表しているものです。システム・ハードウェア、ソフトウェアの設計、構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。システムやコンポーネントの購入を検討される場合は、ほかの情報も参考にして、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。インテル製品の性能評価についてさらに詳しい情報をお知りになりたい場合は、**www.intel.com/performance/resources/benchmark_limitations.htm** (英語) を参照してください。
* 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。

最適化に関する注意事項: 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

2 インテル® C++ コンパイラーとインテル® IPP を使用した場合のパフォーマンス向上率

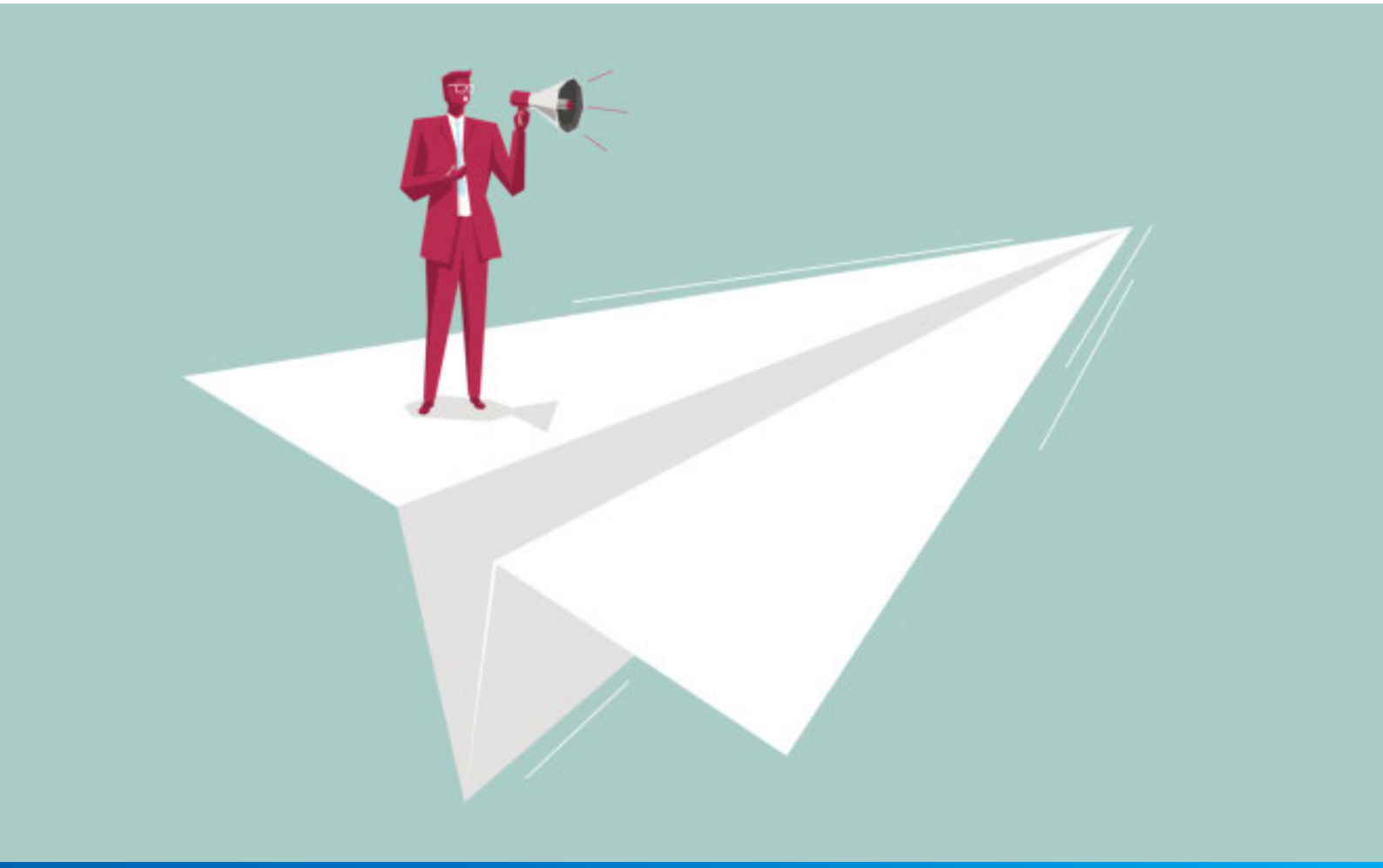
インテル® IPP ライブラリーを追加した後、アプリケーションのパフォーマンスは GCC で生成されたオリジナルコードの 17 倍に向上しました。

まとめ

JD.com は、インテル® C++ コンパイラーを利用することにより、画像処理アプリケーションのパフォーマンスを大幅に向上することができました。JD.com の画像サイズ変更コードのパフォーマンスは、インテル® Xeon® プロセッサー・ベースのサーバーで 12 倍に向上しました。インテル® IPP ライブラリーを追加した後、アプリケーションのパフォーマンスは GCC で生成されたオリジナルコードの 17 倍に向上しました。

パフォーマンスが向上したことにより、JD.com は、より多くの画像をより速く低レイテンシーで処理できるようになりました。JD.com は、効率的なシステム運用とリソース集約型コードの高速化により、大幅なコスト削減に成功しました。

2015 年 8 月に、インテルは、インテル® IPP を含む、インテル® パフォーマンス・ライブラリーのコミュニティー・ライセンスを発表しました。このライセンスの下、個人、企業、組織は、実績のある強力なパフォーマンス・ライブラリーを無料で利用し、高速で信頼性の高い優れたソフトウェア・アプリケーションを作成できます。詳細は、**software.intel.com/free_tools_and_libraries** (英語) を参照してください。



音声認識パフォーマンスの向上

インテル® MKL による Qihoo360 Euler の最適化

Ying Hu インテル APAC R&D テクニカル・コンサルティング・エンジニア
Yanfeng Mu インテル China アプリケーション・エンジニア

Euler プラットフォームは、ビジネス向けの機械学習関連の計算モデルをサポートする重要な分散型オフライン計算プラットフォームです。中国のインターネット・セキュリティー企業 Qihoo360 Technology Co., Ltd. にとって、インテル® Xeon® プロセッサー・ベースのサーバーにおける Euler プラットフォームのパフォーマンスは非常に重要です。Qihoo360 は、インテルとの共同作業により、インテル® アーキテクチャー向けにアプリケーションを最適化しました。両社のエンジニアは、ともに協力して**インテル® マス・カーネル・ライブラリー** (インテル® MKL) を使用した Euler プラットフォームの音声認識モジュールの最適化に取り組み、パフォーマンスを 5 倍向上することに成功しました。

ビジネス要件

Qihoo360 には、1 カ月で約 5 億人のアクティブ・インターネット・ユーザーと 6 億 4000 万人を超えるモバイルユーザーがいます。Qihoo360 は、すべてのインターネット・ユーザーとモバイルユーザーにとってセキュリティーは不可欠なものであると認識し、マルウェアや悪意のある Web サイトからユーザーのコンピューターとモバイルデバイスを保護する、包括的で効率良く使いやすいインターネット / モバイル・セキュリティー製品とサービスを提供することにより、ユーザーベースを構築してきました。360 Safe Guard などのインターネット・セキュリティー製品に加えて、Qihoo360 は、360 Cloud、360 Browsers、360 Search、360 Mobile Assistant を含む新製品をさらにリリースしました。

モバイル・インターネット時代のビジネスとユーザーベースの急速な増加に伴い、Qihoo360 は、その増加するユーザーベースとビジネスをサポートしつつ総保有コストを削減するという難題に直面しました。Qihoo360 のデータセンターでは、ほとんどのサービスとアプリケーションをインテル® アーキテクチャー・ベースのプラットフォームで実行しています。そのため、サービスとアプリケーションのパフォーマンスを向上し、インテル® アーキテクチャーで効率的に実行することが差し迫った課題となりました。

Qihoo360 Euler の概要

Qihoo360 の Euler プラットフォームは、完全なインメモリー計算モデルをメインとする、大規模データ処理用の分散型オフライン計算プラットフォームです。Euler プラットフォームは、機械学習のような多くの反復が必要な計算モデルを効率良く処理するように設計されています。機械学習は、データから学習できるアルゴリズムの構築と研究を行う科学分野です。このようなアルゴリズムは、単に明示的にプログラムされた命令に従うのではなく、入力に基づいてモデルを構築し、そのモデルを使用して予測や決定を行います。Euler プラットフォームは、Qihoo360 の機械学習の基盤となるプラットフォームで、会社のビジネスで非常に重要な役割を果たしています。**図 1** に Euler プラットフォームのアーキテクチャーを示します。

1 Qihoo360 の Euler プラットフォームのアーキテクチャー

Euler プラットフォームは、次のようなさまざまな機能を統合します。

- リソース管理
- タスク・スケジュール
- データ・パーティション
- 並列計算
- データ依存性
- ネットワーク通信
- データバリア
- フォールトトレラントの分散型フレームワーク
- 分散型データ構造

Euler は、Qihoo360 のサービス・デベロッパーが機械学習関連の開発に取り組めるように、API、ライブラリー (C++ のみ)、フレームワークを提供します。

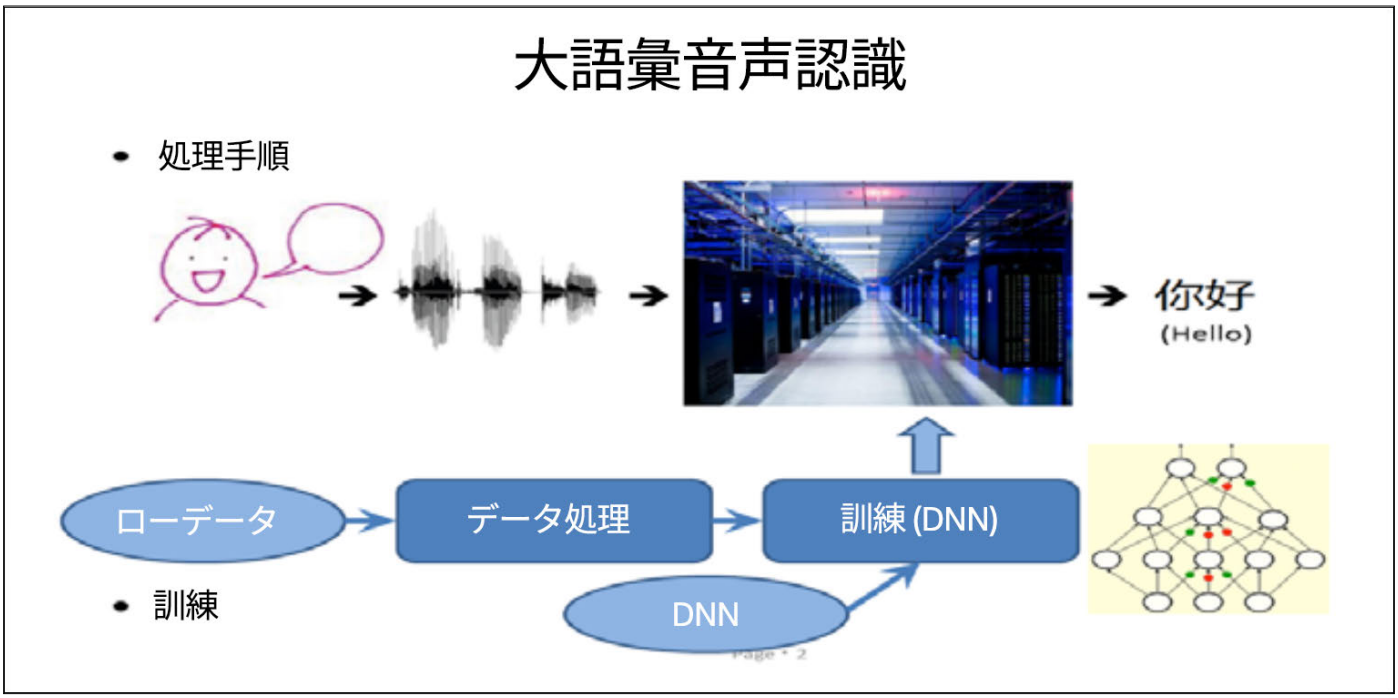
Euler プラットフォームは、クリックスルー率推定モデル、ページランク、ホワイトリスト、モバイル・アシスタント・レコメンデーション、画像検索の画像類似度計算、ネットワーク攻撃行動解析モデル、word2vec モデル、音声認識を含む、Qihoo360 のビジネスのオフライン・トレーニング・モデルとして利用されています。Euler プラットフォームでは、入出力 (I/O) データは分散型ファイルシステム (HDFS* など) で処理されます。計算集約型ワークロードでよく見られるほかの I/O オーバーヘッドはありません。

インテル® Xeon® プロセッサー・ベースのサーバーで Euler の計算パフォーマンスを向上することが Qihoo360 にとって大きな課題でした。Euler プラットフォームの機械学習モデルのトレーニング速度 (つまり、音声認識モデルのトレーニング速度) は Qihoo360 のビジネスに直接影響します。

Qihoo360 とインテルのエンジニアは、ともに協力して、Qihoo360 の Euler プラットフォームの重要な音声認識モジュールの最適化を行いました。音声認識は、話し言葉を文字列に変換する機能で、自動音声認識 (ASR) と呼ばれることもあります。音声認識は、Qihoo360 の音声検索サービスで使用されています。

音声認識の最適化

ASR は、Qihoo360 の Euler プラットフォームにおける、大規模な計算集約型モジュールです。タスクには、ディープ・ニューラル・ネットワーク (DNN) が使用されています。**図 2** に示すように、大量の音声データを前処理した後、すべての浮動小数点データが DNN に送られ、反復を繰り返した後、音響モデルを確立していました。この処理中に、プロジェクト全体のボトルネックになった、膨大な行列ベクトル演算が行われていました。



2 Euler プラットフォームの大語彙音声認識

計算速度が Qihoo360 の ASR モジュールのボトルネックになりましたが、単にサーバープールを拡張することはコスト効率が悪いため Qihoo360 では最適化されたソフトウェア・ソリューションを求めていました。

インテル® Parallel Studio XE は、インテル® アーキテクチャーの機能を最大限に引き出し、高度なテクノロジーを簡単に導入できる**ソフトウェア開発ツール**スイートです。すべてのインテル® アーキテクチャー・ベースのクラスター / サーバー・プラットフォーム向けに、インテル® C/C++ コンパイラーとインテル® MKL を含むツール群を提供し、Windows*、Linux*、OS X*、Android* に対応しています。

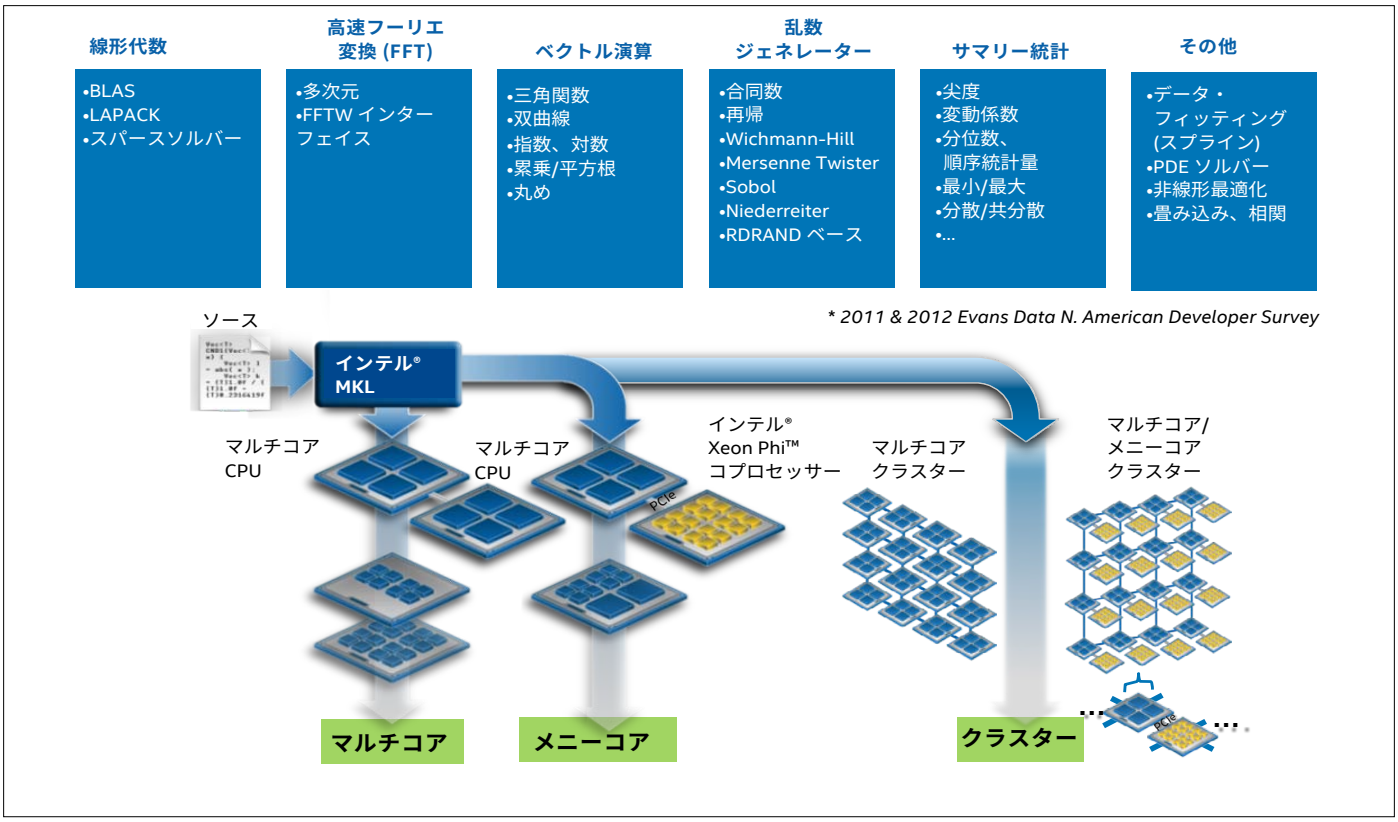
インテル® C/C++ コンパイラーの概要

インテル® C/C++ コンパイラーは、インテル製プロセッサーと互換プロセッサーの両方で優れたパフォーマンスを提供するように設計されており、新しいマルチコア・プロセッサーやマルチプロセッサー・システムを利用するための機能が備わっています。すべてのインテル® アーキテクチャー・ベースのプラットフォーム向けに、自動ベクトル化 / 並列化、メモリー / キャッシュラインのチューニング、高レベルの最適化を含む、最適化されたコードを自動的に生成します。最小限の CPU サイクルでタスクを完了する方法も調べます。また、Microsoft* Visual C++* (Windows*) および GNU* コンパイラー・コレクション (GCC) (Linux*) と互換性があります。OpenMP*、自動並列化、インテル® Xeon Phi ™ コプロセッサーのサポートにより、マルチコア、メニーコア、マルチプロセッサー・システムでスケーリングすることもできます。すべてのコンパイラー機能は、開発者が開発時間、コスト、リスクを短縮 / 削減して、インテル® アーキテクチャーで最高のパフォーマンスを達成できるように支援します。(詳細は、**こちらから**。)

インテル® MKL の概要

インテル® MKL (**図 3**) は、科学、工学、金融系アプリケーション向けの豊富な機能を備えたハイパフォーマンス数学ライブラリーです。開発者は、最小限の労力で優れたパフォーマンスを達成できます。インテル® MKL は、インテルのマルチコア・プロセッサーやメニーコア・プロセッサーを含む、最新のインテル製プロセッサー向けに最適化されています。インテル® Xeon® プロセッサーとインテル® Xeon Phi™ 製品ファミリーの計算および並列処理能力を最大限に引き出すことができます。(サポートしているプロセッサーの一覧は、インテル® MKL **リリースノート** (英語) を参照してください。)

インテル® MKL は、基本線形代数サブプログラム (**BLAS**) および線形代数パッケージ (LAPACK) ルーチン、高速フーリエ変換、ベクトル演算関数、乱数生成関数、その他の機能を含む、豊富な関数群を提供します。C++、Fortran、C#、Java*、Python*、R を含む標準 API へのインターフェイスも提供しています。



3 インテル ® MKL

インテル® MKL の BLAS ライブラリーはインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX およびインテル® AVX2) のような SIMD (Single Instruction Multiple Data) 命令で最適化されています。マルチスレッド・バージョンも用意されているため、多くの数学ライブラリーおよびアプリケーションでパフォーマンス・ビルディング・ブロックとして使用することができます。BLAS 自体は、一般的な数学計算ルーチンのセットと線形代数ライブラリーの標準な API で、ベクトル・スケーリング、ベクトルドット積、行列ベクトル乗算、行列乗算のような一般的な線形代数演算 (**表 1**) を実行します。

演算	CBLAS ルーチン (S は単精度浮動小数点)	例	計算の複雑さ (ワーク)
ベクトル x スカラー	cblas_SSCAL	x = a*x	O(N)
ベクトル x ベクトル	cblas_SAXPY	y = y + α x	O(N)
行列 x ベクトル	cblas_SGEMV	y = αAx + βy	$O(N^2)$
行列 x 行列	cblas_SGEMM	C = αA * B + βC	$O(N^3)$

表 1. DNN モデリングの CBLAS 関数

BLAS は **netlib** により 1979 年に Fortran ライブラリーとして最初に実装されました。現在は、CBLAS (BLAS の C インターフェイス) のような別の実装や、インテル® MKL、AMD ACML、GotoBLAS、**ATLAS** (移植性の高い、自己最適化 BLAS) のような高度に最適化された実装など、さまざまなバリエーションがあります。BLAS ライブラリーは、高水準数学プログラミング言語 / ライブラリー (LINPACK、LAPACK、**MATLAB***、GNU* Octave、**Mathematica***、**NumPy**、R など) やあらゆる種類の計算アプリケーションで、ビルディング・ブロックとして広く利用されています。開発者は、任意の BLAS ライブラリーをインテル® MKL の BLAS ライブラリーに置換して、インテル® アーキテクチャーで BLAS 計算を簡単に高速化できます。インテル® MKL の BLAS ライブラリーは、MATLAB*、NumPy、Scipy、R、Armadillo、Boost uBLAS を含む、多くのサードパーティー・ライブラリーと統合することができます。

「インテルと協力してインテル® MKL とインテル® C/C++ コンパイラーを利用することにより、インテル® アーキテクチャーで Euler ASR モジュールの計算速度を大幅に向上できたことを喜んでいます。この取り組みは弊社が引き続き業界のリーダーとしてビジネスを成長させる新しい機会を得る助けとなることでしょう。」

Qihoo360
アーキテクト
Bai Ming 博士

インテル® コンパイラーとインテル® MKL ベースのソリューション

前述のように、Qihoo360 では計算集約型アプリケーションを最適化する方法を必要としていました。元々、Qihoo360 は、計算レイヤーの基本コンポーネントとして ATLAS CBLAS ライブラリーと、デフォルト・コンパイラーとして GCC を使用していました。CBLAS ライブラリーは BLAS ライブラリーの C インターフェイスであり、上位レベルのアプリケーションをサポートする基本計算ブロックとしてよく使用され、標準 API を備えています。そのため、開発者は、コードを変更することなく、CBLAS ライブラリーを最適化された BLAS ライブラリーに変更することができます。インテル® MKL には、標準 CBLAS ライブラリーと同じ API を備えた BLAS ライブラリーと CBLAS ライブラリーがあります。インテル® コンパイラーはインテル® アーキテクチャーのパフォーマンスを最大限に引き出すことができるため、Qihoo360 の Euler チームは、GCC コンパイラーと CBLAS ライブラリーの代わりに、インテル® コンパイラーとインテル® MKL を使用して ASR モジュールをビルドすることにしました。

図 2 に示すように、DNN は ASR モジュールの最も重要な部分です。DNN は、CBLAS API でサポートされるさまざまな線形代数演算 (**表 1**) により構築されています。

例えば、行列乗算関数 **cblas_sgemm** を見てみましょう。**表 2** の左側のコードは機能のプロトタイプを、右側は CBLAS 実装を示しています。

アルゴリズムの C プロトタイプ

```
void func_a_1(const int row, const int col, const
float* const in_vec1, const float* const in_vec2,
float* const out_vec)
{
memset(out_vec, 0, sizeof(float) * row*col);
float sum;
for (int i = 0; i < row; ++i)    {
for (int j = 0; j < col; ++j)     {
         sum = 0.0;
 for (int k = 0; k < col; k++)
sum += in_vec1[i*col + k] * in_vec2[k*col +j];
        out_vec[offset +j] =sum;
                }
        }
}
```

CBLAS 実装

```
void func_a_2(const int row, const int col, const
float* const in_vec1, const float* const in_vec2,
float* const out_vec)
{
memset(out_vec, 0, sizeof(float) * row*col);
float alpha = 1.0, beta = 0.0;
 cblas_sgemm(CblasRowMajor,CblasNoTrans,CblasNoTr
ans, row, col, col, alpha, in_vec1, col, in_vec2,
col, beta, out_vec, col);
}
```

ビルドのコマンドライン:

オープンソース:

```
$g++ -O2 test_sgemm.cc -I./CBLAS/
include /usr/lib64/atlas/libptcblas.so.3 -fopenmp
-lm -lpthread
```

インテル® MKL :

```
$g++ -O2 test_sgemm.cc -I./CBLAS/
include -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_
core -liomp5 -lm -lpthread*
```

*** 実際のプロジェクトでは `$icc test_sgemm.cc -mkl` を使用します。**

表 2. ATLAS BLAS およびインテル® MKL で cblas_sgemm 関数をビルド

OS で (Red Hat yum リポジトリーから) 提供される ATLAS CBLAS とインテル® MKL の CBLAS ライブラリー (インテル® Parallel Studio XE 2013 SP1 の一部) を使用して **cblas_dgemm** 関数を作成しました。オリジナルのコードは変更していません。唯一の変更点は、**表 2** のように、ATLAS CBLAS ライブラリーの代わりにインテル® MKL BLAS ライブラリーをリンクするようにリンク行を変更したことです。インテル® MKL は新しいインテル® AVX 命令向けに最適化され、マルチコア・プラットフォーム向けに最適なチューニングが行われているため、libptcblas.so のマルチスレッド・バージョンでも OS で提供される ATLAS CBLAS のパフォーマンスを圧倒的に上回っています。**図 4** の左のグラフは、インテル® MKL の BLAS を使用した場合の **cblas_sgemm** のスピードアップを、OS で提供されるオリジナルの ATLAS BLAS と比較した結果です。パフォーマンス・メトリックはスピードアップです。**図 4** から、テストマシン (**インテル® Xeon® プロセッサー E5-2680** 2.70GHz、2 ソケット x 8 コア、インテル® AVX 対応、Red Hat Enterprise Linux* Server 6.3) でインテル® MKL の BLAS はオリジナルの ATLAS BLAS より 7 倍高速であることが分かります。

インテル® MKL はインテル製プロセッサー向けに最適なチューニングが行われているため、Qihoo360 がインテル® AVX2 のような新しいハードウェア・ソリューションにアップグレードした場合、インテル® MKL の CBLAS ライブラリーで新しいハードウェアの機能が自動的に利用され、その時点で最高のパフォーマンスを引き出すことができます。

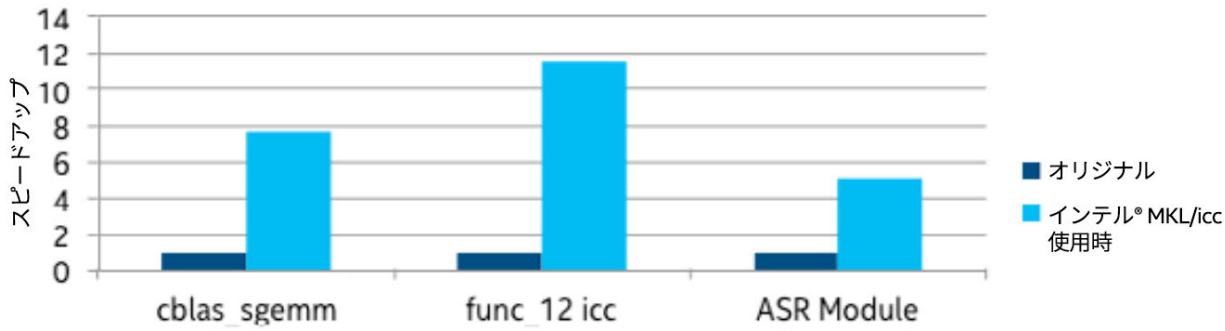
CBLAS 関数に加えて、ASR モジュールにも多くのベクトル演算が含まれています。例えば、**func_12()** 関数には、ベクトルの最大値を求め、ベクトルの指数を計算し、それらを合計する、といったさまざまな種類の計算が含まれています。当初は、この関数を GCC でビルドし、CBLAS 関数を適用しようとしました。しかし、パフォーマンスが多少向上したものの、ほとんどの計算は O(N) であり、マルチスレッド化が困難であったため、速度は向上しませんでした。インテル® コンパイラーは、自動ベクトル化や並列化、メモリーやキャッシュラインのチューニングを含む、最適化されたコードを自動的に生成できるため、インテル® コンパイラーでビルドすることにより、パフォーマンスが 11 倍に向上しました。(**図 4** の真ん中のグラフを参照。)

```
void func_12(const int rows, const int cols, const
float* const in_vec, float* const out_vec)
{// バッチループ開始
for (int ridx=0; ridx<rows; ridx++)
{
const float* in_buf = in_vec;
 float* out_buf = out_vec;
float max_val = -FLT_MAX;
float exp_sum = 0.0f;
for (int cidx=0; cidx<cols; cidx++)
{
max_val = max_val > in_buf[cidx]?
max_val : in_buf[cidx];
}
for (int cidx=0; cidx<cols; cidx++)
{
out_buf[cidx] = exp(in_buf[cidx]-max_val)
exp_sum += out_buf[cidx];
}
float scale = 1.0f / exp_sum;
for (int cidx=0; cidx<cols; cidx++)
{
        out_buf[cidx] *= scale;
}
        in_buf += cols;
        out_buf += cols;
} // バッチループ終了
 }

/* GNU* G++ およびインテル® コンパイラーでビルド
$g++ -O2 test_main.cc -o test_gcc -fopenmp -lm
-lpthread
$icc test_main.cc -o test_icc -openmp -lm -lpthread
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
-liomp5
 */
```

表 3. GCC およびインテル® C++ コンパイラーで func_12 関数をビルド

これらのテスト結果に感銘を受け、Qihoo360 は、インテル® コンパイラーとインテル® MKL をモジュール全体に採用しました。インテル® MKL の CBLAS ライブラリーと ベクトル・マス・ライブラリー、インテル® コンパイラー、いくつかの最適化手法 (メモリー・アライメント、CPU KMP アフィニティー、ベクトル化、インテル® SSE 命令を含む) を利用することにより、ASR のパフォーマンスは大幅に向上し、以前の実装の約 5 倍になりました (**図 4** の右のグラフ)。



構成情報 - バージョン: インテル® MKL 11.1; ハードウェア: インテル® Xeon® プロセッサー E5-2680 1.70GHz、2 ソケット x 8 コア、インテル® AVX 対応。8MB キャッシュ、8GB メモリー; オペレーティング・システム: RHEL 6.3 GA x86_64; ベンチマーク出典: Qihoo360 Corporation.

性能に関するテストや評価は、特定のコンピューター・システム、コンポーネント、またはそれらを組み合わせて行ったものであり、このテストによるインテル製品の性能の概算の値を表しているものです。システム・ハードウェア、ソフトウェアの設計、構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。システムやコンポーネントの購入を検討される場合は、ほかの情報も参考にして、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。インテル製品の性能評価についてさらに詳しい情報をお知りになりたい場合は、**www.intel.com/performance/resources/benchmark_limitations.htm** (英語) を参照してください。

* 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。

最適化に関する注意事項: 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

4 ASM 最適化の結果

まとめ

オリジナルの CBLAS ライブラリーをインテル® MKL の CBLAS ライブラリーに変更するだけで、Qihoo360 ではインテル製プロセッサーのパフォーマンスを引き出すことができました。インテル® MKL は、現在および将来のインテル製プロセッサー向けに最適化されているため、アプリケーションのコードを変更することなく、新しいプロセッサーのパフォーマンスを自動的に引き出すことができます。インテル® コンパイラーは、簡単かつ柔軟にさらなるパフォーマンスの向上を図る上で役立ちます。これらのツールを利用することで、開発者は、より少ない労力でパフォーマンスを向上し、アプリケーション開発の効率を改善し、保守にかかる時間を減らし、インテル® アーキテクチャーで優れたスケーラビリティーを実現することができます。Qihoo360 はこの結果に満足しており、ほかの計算集約型モジュールの将来の開発にもこのソリューションを利用することを計画しています。

サブモジュールによる Fortran 開発者の生産性の向上

Udit Patidar インテル コーポレーション 開発製品部門 プロダクト・マーケティング・エンジニア
Steve Lionel (情報提供) インテル コーポレーション 開発製品部門 テクニカル・コンサルティング・エンジニア

インテル® Parallel Studio XE (インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー、互換プロセッサーの多数のコアと広いベクトルレジスター幅を利用して、アプリケーションのパフォーマンスを大幅に向上する**ソフトウェア開発ツール**スイート) では、最新の Fortran 言語規格のサポートが拡張されました。この拡張には、C との互換性の向上 (Fortran 2015 ドラフト仕様で提案) と BLOCK からの EXIT 機能 (Fortran 2008) が含まれます。

この記事では、インテル® Fortran コンパイラー 16.0 (最新のインテル® Parallel Studio XE のコンポーネント) でサポートされた、Fortran 2008 のサブモジュール機能について取り上げます。サブモジュールのいくつかの重要な機能を説明した後、多くの Fortran 開発者が生産性を大幅に向上できると信じている理由を述べます。

Fortran 90: モジュールの採用 (ただし再コンパイルが必要)

Fortran 90 では、ほかのプログラム単位で参照可能な別々にコンパイルされた宣言とプロシージャーのコレクションである、モジュールの概念が採用されました。その結果、参照結合でモジュール内に含まれるすべての変数に適切にアクセスできるようになり、開発者の効率が向上しました。しかし、モジュールにわずかな変更を加えた場合でも、そのモジュールを使用するすべてのソースファイルを再コンパイルする必要がありました。

モジュール・プロシージャーの実装と宣言を切り離して、1 つのファイルのみを再コンパイルして再リンクできるようになれば、どれだけのコンパイル時間を短縮できるか想像してみてください。Fortran 2008 のサブモジュールは、この想像を現実のものにします。

「サブモジュールは、プログラマーの生産性を高め、特に大規模プロジェクトでプログラムの保守コストを削減します。」

Cray User Group 2005 Proceedings
Bill Long 氏

Fortran 2008: 大規模な再コンパイルの回避

ほかの Fortran ソースのようにサブモジュールをビルドして、そのオブジェクト出力をアプリケーションのリンクに含めます。makefile や依存関係を考慮してビルドスクリプトを作成するときに、唯一新しいことは、サブモジュールはその親モジュールに依存し、(子サブモジュールを除いて) 何もサブモジュールに依存しないことです。

サブモジュールについて詳しく見てみましょう。次の (Fortran 2008 の) 例について考えてみます。

```
module points
  type :: point
     real :: x, y
  end type point
contains
  function point_dist(a, b) result(distance)
  type(point), intent(in) :: a, b
    real :: distance
    distance = sqrt((a%x - b%x)**2 + (a%y - b%y)**2)
  end function point_dist
end module points
```

このモジュールをサブモジュールに変換するには、**point_dist()** の実装をサブモジュールにします。具体的には、**contains** セクションを削除して、モジュール・プロシージャーのインターフェイス・ブロックを追加します。

```
module points
  …
interface
    module function point_dist(a, b) result(distance)
       type(point), intent(in) :: a, b
       real :: distance
     end function point_dist
  end interface
end module points
```

モジュール・プロシージャー **point_dist()** の実装を含むサブモジュールも必要です。このサブモジュールは別のファイルに記述します。次のサブモジュール **points_a** について考えてみます。

```
submodule (points) points_a
end submodule points_a
```

括弧の中の名前は、このサブモジュールの親です。次の選択肢があります。

- インターフェイスのサブルーチン/関数ステートメントおよびすべての宣言を繰り返す
- モジュール・プロシージャーを使用してインターフェイスを継承する

宣言の繰り返しのほうが保守は簡単ですが、モジュール・プロシージャーを使用するほうが簡潔な実装になります (宣言が 1 回で済むため)。両方のコードを次に示します。

```
submodule (points) points_a
contains
  module function point_dist(a, b) result(distance)
    type(point), intent(in) :: a, b
    real :: distance
    distance = sqrt((a%x - b%x)**2 + (a%y - b%y)**2)
  end function point_dist
end submodule points_a
```

または

```
submodule (points) points_a
contains
  module procedure point_dist
    distance = sqrt((a%x - b%x)**2 + (a%y - b%y)**2)
  end procedure point_dist
end submodule points_a
```

この時点で、サブモジュールの実装はモジュールとは別のものになります。サブモジュールを変更し、コンパイルして、アプリケーションと再リンクできます。オリジナルモジュールを再コンパイルする必要はありません。

「サブモジュールによりコードが高速に (あるいは低速に) なることはありませんが、扱いやすい単位に分割し、ビルド時間を短縮できるため、プログラマーの生産性が大幅に向上します。」

Steve Lionel インテル コーポレーション 開発製品部門
Dr. Fortran ブログシリーズ

サブモジュールにより、扱いやすい単位に分割し、ビルド時間を短縮できるため、プログラマーの生産性が大幅に向上します。親モジュールのソースコードのサイズを減らすことも可能でしょう。

古くからの Fortran ユーザーグループとサポーターの中には、プログラマーの生産性を向上する重要なステップとして、サブモジュールを賞賛している人々もいます。

1 人でも多くの Fortran 開発者が、この新しいサブモジュール機能に興味を持ってくれることを期待しています。

THE PARALLEL UNIVERSE