THE PARALLEL UNIVERSE

Issue 11
2012 年 9 月

intel Software

インテル® Parallel Studio XE 2013

パフォーマンスを向上させる 10 項目の機能

James Reinders

目次

編集者からのメッセージ

インテル® Parallel Studio XE 2013 がもたらす「ニューノーマル (新たな常識)」

JAMES REINDERS 4

インテル® Parallel Studio XE 2013: パフォーマンスを向上させる 10 項目の機能

JAMES REINDERS 5

アプリケーションを高速化するインテル® Parallel Studio XE 2013 スイートのコンポーネントと新しい機能およびアプリケーションの可能性について説明します。

インテル® ソフトウェア開発ツールを使用した HMMER の解析

WALTER SHANDS 7

スレッドエラーやメモリーエラーの確認からパフォーマンスやコードの最適化まで、最新世代のマルチコア・プロセッサーで HMMER のようなアプリケーションを開発するテクニックを紹介します。

ポインターチェッカー: 範囲外のメモリーアクセスの迅速な検出

KITTUR GANESH 20

ポインターチェッカーは、メモリーが破壊される前に範囲外のメモリーアクセスをキャッチするように設計されています。ポインターチェッカーを効率的に使用する方法に加えて、セキュリティーとランタイムのトレードオフの適切なバランスを設定する方法を説明します。

インテル® (Visual) Fortran Composer XE の新しい並列プログラミング機能

STEVE LIONEL 21

Fortran に優れた並列化をもたらす新しい 2 つの機能、DO CONCURRENT と Co-Array の概要を説明します。

インテル® マス・カーネル・ライブラリー (インテル® MKL) とインテル® コンパイラーを使用し、実行するたびに数値再現性のある結果を取得

TODD ROSENQUIST および SHANE STORY 25

パフォーマンスの向上と、再現性のある結果およびランタイムの一貫性のバランスをどのように取れば良いでしょうか。これらのテクニックを利用することで、管理可能な制約セットの下、アプリケーションで再現性のある結果を生成できます。

© 2012 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Intel Core、Intel VPro、Pentium、Xeon、Xeon Phi、Cilk、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 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。

James Reinders が並列化ツール・スイート、インテル® Parallel Studio XE 2013 の機能を紹介します。

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

編集者からのメッセージ

インテル® Parallel Studio XE 2013 がもたらす「ニューノーマル (新たな常識)」

並列化で変わったものと変わらないもの

並列化は至るところで行われ、並列プログラミングは「ニューノーマル (新たな常識)」となりました。しかし、アプリケーションの記述、デバッグ、チューニングがプログラマーの仕事であることは変わりません。我々はニューノーマルを作成するためのハードルを上げただけなのでしょうか。そうかもしれません。プログラミングはますます高度なものとなり、ツールに要求される機能も増え続けることが予想されます。

インテルは、このニューノーマルをインテル® Parallel Studio XE の最新エディションの新機能でサポートするために膨大な投資を行いました。

インテル® Parallel Studio XE 2013 ではソフトウェア開発のさまざまな要素にわたり、ニューノーマルに必要なサポートを追加するため、ツールセットを更新しました。

この号では、インテル® Parallel Studio XE 2013 の新しい機能を紹介します。「パフォーマンスを向上させる 10 項目の機能」の記事では、重要な新機能の概要を説明します。一つ一つの機能を詳細に説明しようとすれば、機能ごとに 1 つの号を用意しても足りないかもしれません。

そして、この 10 項目の内容をさらに詳しく紹介する、3 つの記事を取り上げています。ポインターチェッカーに関する記事、Fortran 機能に関する記事、そして実行するたびに (どのプロセッサーで実行しても) 数値再現性のある結果が得られる新しい機能に関する記事です。3 つめの機能は、最新のインテル® Parallel Studio XE の新しいオプションセットにより、演算の順序によって結果が異なる浮動小数点の数値表現の性質に対処します。

また、実際の使用例として、インテル® ソフトウェア開発ツールを使用した HMMER の解析について紹介します。この記事では、インテル® VTune™ Amplifier XE のイベントベース・サンプリング解析とインテル® Composer XE のコンパイラーの最適化機能を使用して、SPECint* のコンポーネントである hmmsearch と hmmbuild のビルドおよび解析を行います。

この号で紹介する新しい機能はどれも興味深いものばかりです。皆さんがニューノーマルの世界に辿り着くのをお待ちしています。

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。2012 年に Morgan Kaufmann から出版された新しい書籍、『Structured Parallel Programming』の共著者。このほかに、『Intel® Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』(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) などの並列化に関する文献を発表しています。

インテル® Parallel Studio XE 2013: パフォーマンスを向上させる 10 項目の機能

並列プログラミング・エバンジェリスト兼ディレクター
James Reinders

インテル® Parallel Studio XE 2013 には、最新の最適化や新しいプロセッサーのサポートだけでなく、非常に革新的なさまざまな機能が含まれています。

このツール・スイートは Microsoft* Visual Studio* および GNU ツールにシームレスに統合されるため、これまでの開発環境への投資が無駄になりません。

インテル® Parallel Studio XE 2013 で再コンパイルするだけで、アプリケーションのパフォーマンスを向上させることができます。最新のコンパイラーでリビルドして最新のライブラリーとリンクすることで、最新のプロセッサーを有効活用できます。

この強力なツール・スイートの特長を表す 10 項目の機能についてまとめました。

1. **最新のインテル® プロセッサーのサポート**
   AVX2、TSX、FMA3 のサポートが追加されました。この更新により、新しくリリースされた第 3 世代インテル® Core™ vPro™ プロセッサー (開発コード名: Ivy Bridge) マイクロアーキテクチャーと次の (開発コード名: Haswell) マイクロアーキテクチャーの両方がサポートされます。以前のインテル® プロセッサーおよび互換プロセッサーとの互換性を保ちながら、最新のインテル製品で強化されたパフォーマンスを活用できます。
2. **インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャーのサポート**
   プロトタイプシステムとプリプロダクション・システムにおける 1 年以上のテストを経て、インテル® MIC アーキテクチャーのサポートが追加されました。最初のインテル® Xeon Phi™ コプロセッサー (開発コード名: Knights Corner) を利用するためにほかのツールは必要ありません。使い慣れたインテル® ソフトウェア開発ツールによってサポートされています。新しいツールを使用することなく、インテル® MIC アーキテクチャー対応のコードを生成、デバッグ、最適化することができます。
3. **高度な数値再現性機能**
   ベータテスターの評判が最も高かった新しい機能です。この革新的な新しい「条件付き数値再現性」機能は、結合則の成り立たない浮動小数点演算を制御する優れた機能で、実行するたびに (どのプロセッサーで実行しても) 数値再現性のある結果が得られる、パフォーマンス・ペナルティーが非常に低いオプションを提供します。インテル® マス・カーネル・ライブラリー (インテル® MKL) を使用した浮動小数点演算の再現性、OpenMP* の特別なサポート、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) 向けに追加されたオプションは新たな可能性を広げます。
4. **プロファイル・データの追加と使いやすさの向上**
   不可解なパフォーマンス・データを理解するために費やしていた時間を削減できる、新しい強力な帯域幅とメモリーアクセス解析がインテル® VTune™ Amplifier XE に追加されました。
5. **ポインターチェッカー**
   割り当てられたアドレスを超えてメモリーアドレスにアクセスするコードを検出する、新しいコンパイラー・ベースの診断ツールです。発見が困難なメモリー破壊問題の特定を支援します。
6. **新しいスレッド化アシスタント: インテル® Advisor XE**
   インテル® Advisor XE は、スケーラブルで保守可能な C、C++、C#、Fortran コードの生成を支援します。アプリケーションへの並列化の実装を単純化し、実装前に別のアプローチを評価できます。
7. **Fortran 規格のサポート**
   インテル® Fortran は、Fortran 2003 規格の機能と Fortran 2008 規格の主要な機能 (Co-Array を含む) を幅広くサポートします。インテルは今後もソフトウェア開発製品で Fortran をサポートしていく予定です。もちろん、VAX Fortran*、Compaq* Visual Fortran、Fortran 95、Fortran 90、Fortran 77、Fortran 66 を含む Fortran のサポートと、BLAS、LAPACK、ScaLAPACK、スパースソルバー、高速フーリエ変換、ベクトルマス、その他のライブラリーのサポートにおける豊富な後方互換性も維持します。
8. **C++ パフォーマンス・ガイド**
   5 ステップのプロセスでパフォーマンスの向上を導く新しい C++ パフォーマンス・ガイドが提供されます。
9. **C および C++ 標準規格のサポート**
   C および C++ のサポートに新しい C++11 と C11 機能の先行サポートが追加されました。C99 を含む従来の標準規格のサポートと、IEEE 754-2008 10 進浮動小数点演算のサポートもこれまでどおり継続されます。
10. **エラーの検出と排除: インテル® Inspector XE**
    インテル® Inspector XE は、C、C++、C#、Fortran のパフォーマンスを保証し、アプリケーションの信頼性を高める効率良い方法を提供します。新しいヒープ拡張解析機能は、メモリーリークを検出する重要な新しい方法です。

30 日間の評価版ダウンロード

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

強力なスイートを構成する 4 つの主要コンポーネント:

1. **C++ /Fortran 最適化コンパイラーとライブラリー:**
インテル® Composer XE は、パフォーマンスの向上に取り組む開発者向けの高度に最適化されたツールで、インテル® C++ コンパイラー、インテル® Fortran コンパイラーに加えて、スレッド化、数値演算、マルチメディア、信号処理などのパフォーマンス・ライブラリーが含まれています。インテル® Cilk™ Plus、インテル® TBB、OpenMP* の並列化モデルをサポートしており、今日および将来のハイパフォーマンス・コンピューティング・システムを容易に活用できます。インテル® MKL とインテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP) には、パフォーマンスの向上と開発時間の短縮に役立つ豊富なルーチンが含まれています。

2. **革新的なスレッド化アシスタント (Linux* および Windows*):**
インテル® Advisor XE は、C、C++、C#、Fortran 開発者向けのスレッド化アシスタントです。並列化によりパフォーマンスが向上する領域の特定や重要な同期問題の検出を支援します。インテル® Advisor XE を使用すると、実際に実装を行う前に別のアプローチを評価して、スピードアップの予測、正当性問題の特定を行い、最も投資効果の高いオプションを選択することができます。この機能の優れている点は、コーディングとデバッグを行う前にアプローチを評価できることです。コードに並列化を実装する方法を検討する際に非常に役立ちます。

3. **シリアルおよび並列パフォーマンスを最適化:**
インテル® VTune™ Amplifier XE は、アプリケーションのパフォーマンスをチューニングするための強力なパフォーマンス/スレッド・プロファイラーです。C、C++、C#、Fortran、アセンブリー・コード、Java コードをプロファイルして、hotspot、スレッド化、ロックと待機、DirectX*、帯域幅、その他の役立つパフォーマンス・データを提供します。

4. **より信頼性の高いアプリケーションを作成:**
インテル® Inspector XE 2013 は、Windows* および Linux* のシリアル/並列アプリケーションのメモリーエラーとスレッドエラーを容易に検出します。C、C++、Fortran 開発者向けのスタティック解析機能は、インテル® Studio XE 製品に含まれています。アプリケーションをリリースする前にアクティブで潜在的な問題をピンポイントで指摘する機能は、インテルの独創的なアイデアを実装したものです。

インテル® Parallel Studio XE の詳細

技術仕様の概要	
プロセッサーのサポート	インテル® Xeon® プロセッサー、インテル® Core™ プロセッサー、インテル® Xeon Phi™ コプロセッサーを含む、さまざまな世代のインテル® プロセッサーおよび互換プロセッサーでの動作を検証済みです。
オペレーティング・システム	Windows* および Linux* をサポートします。コンパイラーおよびライブラリーは、Apple* OS X* 上で XCode* 開発環境のアドオンとして利用することもできます。
開発ツールと環境	各プラットフォームの規格に準拠しているコンパイラー (Microsoft* コンパイラー、GNU コンパイラー、インテル® コンパイラーなど) と互換性があります。GNU ツール、Microsoft* Visual Studio* 2008/2010/2012 などのツールに統合できます。
プログラミング言語	C、C++、Fortran 開発を広範囲にサポートします。Java や C# などの .NET 言語を含むプログラムもサポートします。
サポート	すべての製品アップデート、インテル® プレミアサポート、インテル® サポートフォーラムを 1 年間ご利用いただけます。インテル® プレミアサポートから、テクニカルサポート、テクニカルノート、アプリケーション・ノート、および最新のドキュメントにアクセスできます。
コミュニティー	インテル® サポートフォーラムでは、ソフトウェア開発について活発な情報交換が行われています。 http://software.intel.com/en-us/forums (英語)
システム要件	ハードウェア要件とソフトウェア要件の詳細は、次のページを参照してください。 www.intel.com/software/products/systemrequirements/ (英語)

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

ソフトウェア開発エンジニア
Walter Shands

インテル® ソフトウェア開発ツールを使用した HMMER の解析

この記事では、インテル® Parallel Studio XE 2013 を使用して HMMER (http://hmmer.janelia.org/ (英語)) のビルドおよび解析を行い、ツールの機能を説明します。HMMER は、SPECint の 2 つのコンポーネント、hmmsearch と hmmbuild を含むアプリケーションのセットです。まず、インテル® VTune™ Amplifier XE のイベントベース・サンプリング解析を使用して、hmmsearch のパフォーマンス問題の原因となっているコードパス、コンテキスト・スイッチ、スレッド化部分を特定します。次に、インテル® Composer XE コンパイラーのコード最適化機能を使用して、インテル® Xeon® E5 プロセッサーにおける hmmsearch のパフォーマンスを向上させます。さらに、インテル® Inspector XE を使用して、hmmsearch に含まれるメモリーエラーとスレッドエラーを特定します。

hmmsearch は、隠れマルコフモデルと呼ばれるプロファイルを使用して、タンパク質配列データベースの同族体を検索します。globins4.hmm にはプロファイルが含まれており、uniprot_trembl.fasta は 10GB の配列データベースです。

hmmsearch には MPI バージョンもありますが、今回の実験では非 MPI バージョンを使用しました。8 コアのインテル® Xeon® E5-2680 プロセッサー (2.7GHz、ハイパースレッディング有効) と 23.4GB のメモリーを搭載したシステムで hmmsearch を実行しました。GCC とインテル® C コンパイラーを利用し、両方とも構成スクリプトで提供された設定を使用しました。最初に使用したデフォルトの GCC のオプションは次のとおりです。

```
gcc -std=gnu99 -O3 -fomit-frame-pointer -malign-
double -fstrict-aliasing –pthread –msse2
```

このアプリケーションは SSE2 以降の組込み関数を使用して最適化されたアルゴリズムを利用するため、最低でも SSE2 命令セットをサポートする必要があります。

最初に使用したデフォルトのインテル® コンパイラー・オプションは次のとおりです。

```
icc -O3 -ansi_alias -pthread
```

あるコンパイラーから別のコンパイラーへアプリケーションを移植する場合の課題の 1 つは、アプリケーションのビルドに使用するコンパイラー・オプションがサポートされているかどうかを確認することです。インテル® C コンパイラーは、GCC などのコンパイラーでも有効な多くのオプションをサポートしています。インテル® C コンパイラーは GCC で生成されたオブジェクト・ファイルと互換性のあるオブジェクト・ファイルを生成するため、アプリケーションの一部をインテル® C コンパイラーでコンパイルし、残りの部分を GCC でコンパイルすることも可能です。

インテル® C コンパイラーを使用して -O1、-O2 または -O3 オプションを指定すると、-fomit-frame-pointer オプションが設定されます (このオプションを明示的に指定する必要はありません)。-malign-double オプションは、IA-32 アーキテクチャー・ベースのシステムで最適なパフォーマンスになるように、double、long-double、および long-long 型でアラインします。このオプションはインテル® C コンパイラーで使用できます。

次のコマンドラインを使用してアプリケーションを開始しました。

```
./hmmsearch globins4.hmm ../../
uniprot_trembl.fasta
```

次のステップでは、インテル® VTune™ Amplifier XE を使用してアプリケーションの hotspot を調べます。このプロファイラー・ツールは、オーバーヘッドの低いテクニックを使用して、プロセッサー・アーキテクチャーやアセンブリー・コードの知識がなくても、マルチコア・パフォーマンスのボトルネックをすばやく発見することができます。データを収集するためにアプリケーションにコードを追加する必要がないことに注意してください。

インテル® VTune™ Amplifier XE で hmmsearch のソースコード行を表示するには、リリースビルドにシンボルを含める必要があるため、-g オプションを追加します。ソースビューで該当するコードがすべて表示されるように、-fno-inline-functions オプションも追加しています。

インテル® VTune™ Amplifier XE の hotspot 解析は、アプリケーションで最も CPU アクティビティーが高い部分と、スレッド・オーバー・タイムの CPU アクティビティーの量を表示します。(図 1)

インテル® VTune™ Amplifier XE の hotspot ビューでは、最も CPU 時間を費やしている関数が p7_MSVFilter であることが分かります。関数名をダブルクリックすると、関数のパフォーマンスを最適化するために使用されている SSE 組込み関数の呼び出しが表示されます。アセンブリー・ビューでは、ベクトル命令が使用されているにもかかわらず、インテル® Xeon® プロセッサーの 256 ビット・レジスターや AVX 命令を活用できていないことが分かります。(図 2)

インテル® コンパイラーで p7_MSVFilter のオリジナルの C コードをコンパイルして、ターゲットマシンで利用可能な命令セットに合わせて関数がベクトル化されたため、128 ビット・レジスターの使用が制限されている可能性があります。

スレッド・タイムライン・ビューから、ワーカースレッドで使用されている CPU 時間は少ないにもかかわらず、1 つのスレッドで多くの CPU 時間が費やされていることが分かります。このスレッドは、シーケンス・データベース・ファイルを読み取っているスレッドであることがこの後で判明します。(図 3)

「より大幅にパフォーマンスを向上させるには、ファイルの読み取りによるアプリケーションのシリアル化問題を解決する必要があります。」

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

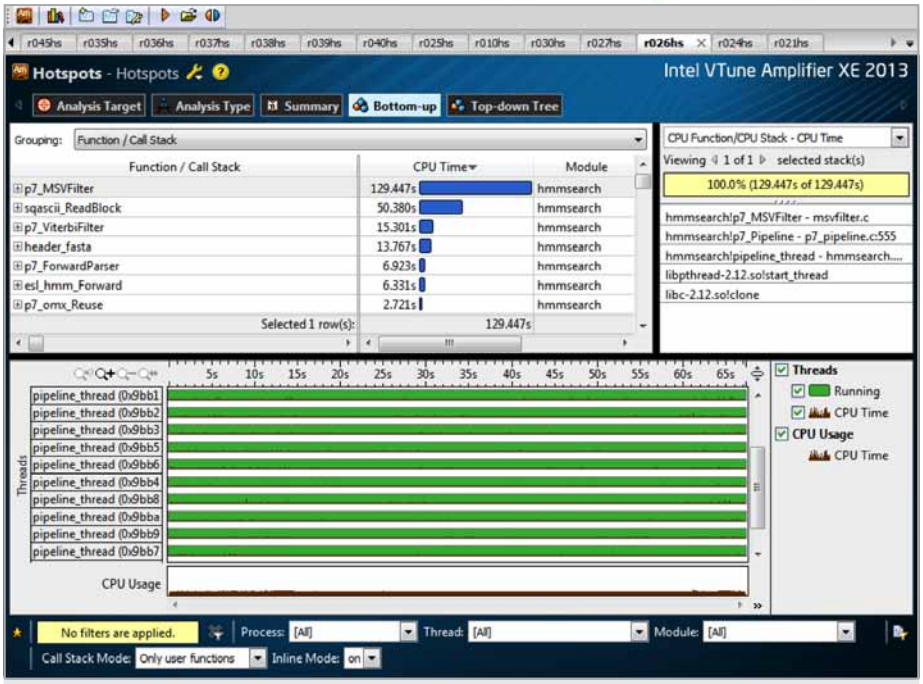

Hotspots - Hotspots
Intel VTune Amplifier XE 2013
Analysis Target
Analysis Type
Summary
Bottom-up
Top-down Tree
Grouping: Function / Call Stack
Function / Call Stack
CPU Time
Module
p7_MSVFilter 129.447s hmmsearch
sqascii_ReadBlock 50.380s hmmsearch
p7_ViterbiFilter 15.301s hmmsearch
header_fasta 13.767s hmmsearch
p7_ForwardParser 6.923s hmmsearch
esl_hmm_Forward 6.331s hmmsearch
p7_omx_Reuse 2.721s hmmsearch
Selected 1 row(s): 129.447s
CPU Function/CPU Stack - CPU Time
Viewing 1 of 1 selected stack(s)
100.0% (129.447s of 129.447s)
hmmsearch!p7_MSVFilter - msvfilter.c
hmmsearch!p7_Pipeline - p7_pipeline.c:555
hmmsearch!pipeline_thread - hmmsearch....
libpthread-2.12.so!start_thread
libc-2.12.so!clone
Threads
Running
CPU Time
CPU Usage
CPU Time
No filters are applied.
Process: [All]
Thread: [All]
Module: [All]
Call Stack Mode: Only user functions
Inline Mode: on

図 1

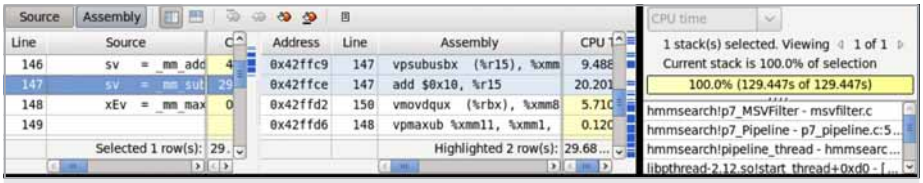

Source
Assembly
Line
Source
Address
Line
Assembly
CPU time
1 stack(s) selected. Viewing 1 of 1
Current stack is 100.0% of selection
100.0% (129.447s of 129.447s)
hmmsearch!p7_MSVFilter - msvfilter.c
hmmsearch!p7_Pipeline - p7_pipeline.c:5...
hmmsearch!pipeline_thread - hmmsearc...
libpthread-2.12.so!start_thread+0xd0 - [...

図 2

Thread
Running
CPU Time
CPU Usage
CPU Time
CPU Usage

図 3

アプリケーションは、マシンのハードウェア・スレッド数 +1 (ハイパースレッド・マシンの場合はハイパースレッド数 +1) のスレッドを生成します。ここでは、17 のスレッドが実行されています。スレッド数を 5 スレッドに制限して hmmsearch を実行すると、17 スレッドの場合よりもアプリケーションが適切にスケールされるようになりました。(図 4)

その証拠に、17 スレッドでは 67.418 秒かかっていた実行時間が 4 スレッドでは 62.561 秒に短縮されています。

5 スレッドの hotspot 表示でスレッド別に結果をフィルターすることにより、最も上位のスレッドはデータファイルを読み取っているスレッドであることが分かりました。(図 5)

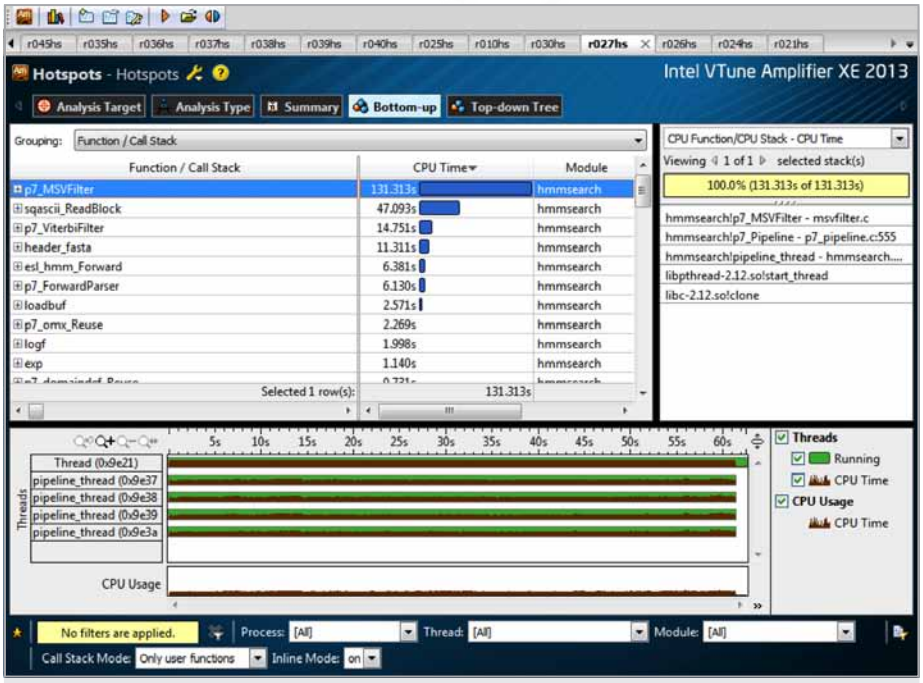


図 4

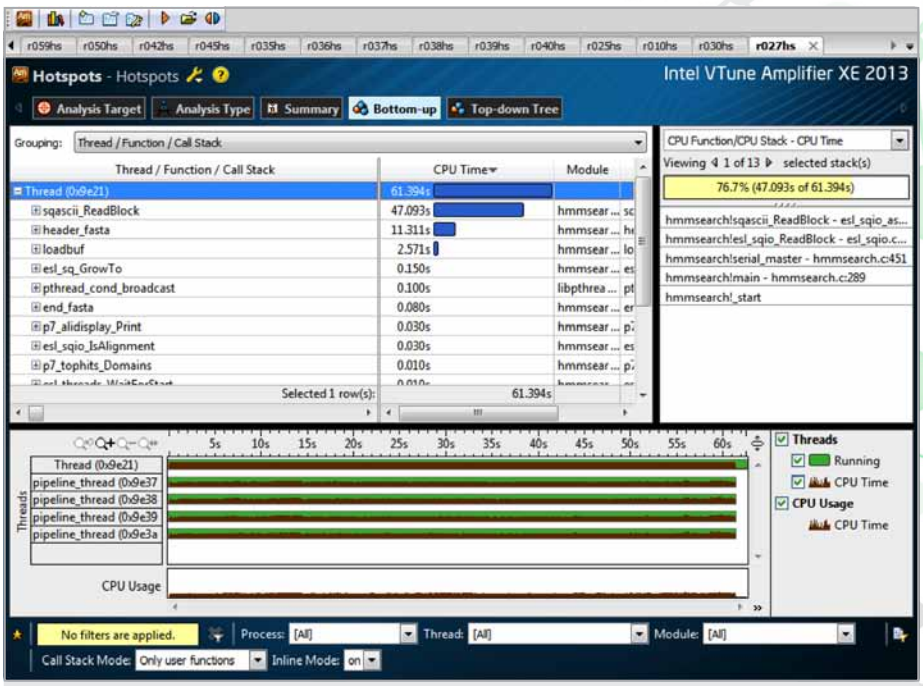


図 5

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

17 スレッドのケースでインテル® VTune™ Amplifier XE のロックと待機解析を行うと、シーケンス・データベース・ファイルを読み取っているスレッドからワーカースレッドに (黄色の線で示された) 非常に多くの移行が行われていることが判明しました。(図 6)

hmmsearch は、生産/消費モデルを使用しており、データをワーカースレッド (図の pipline_thread) のキューに入れた生産スレッド (図の 0xa055) が、ワーカースレッドにブロードキャスト・メッセージを送ったときに (生産スレッドからワーカースレッドへの) スレッドの移行が行われています。

拡大すると、スレッドの実行時間 (深緑) がスレッドの待機時間 (薄緑) よりも少ない、つまり生産的な作業が可能な時間が無駄になっていることが分かります。(図 7)

4 スレッドしか使用していない hmmsearch のスレッドビューを拡大して比較してみましょう。データファイルを読み取っている (図で一番上の) スレッドからの移行は通常、生産スレッドからワーカースレッドへ作業を移動した結果によるものです。(図 8)

しかし、17 スレッドの hmmsearch では、多くのスレッドの移行は作業が完了した結果ではありません。(図 9)

さらに 17 スレッドのケースを拡大すると、これらのスレッドの移行は、データブロックをワークキューで処理する準備ができたことをワーカースレッドに知らせる pthread_cond_broadcast 呼び出しの結果であることが分かります。データブロックを扱えるのは一度に 1 つのスレッドのみで、ほかのスレッドは再度待機しなければなりません。(図 10)

5 スレッドのケースでは、処理するデータブロックを待つスレッドは 2 つのみで、1 つのスレッドを除いてすべてのスレッドが活用されています。(図 11)

つまり、hmmsearch パイプライン・スレッドが 4 つ以上になると、スタベーションが発生することになります。データファイルを読み取るスレッドは、4 つのワーカースレッドが計算を続けられるほどデータを速く提供できないのです。

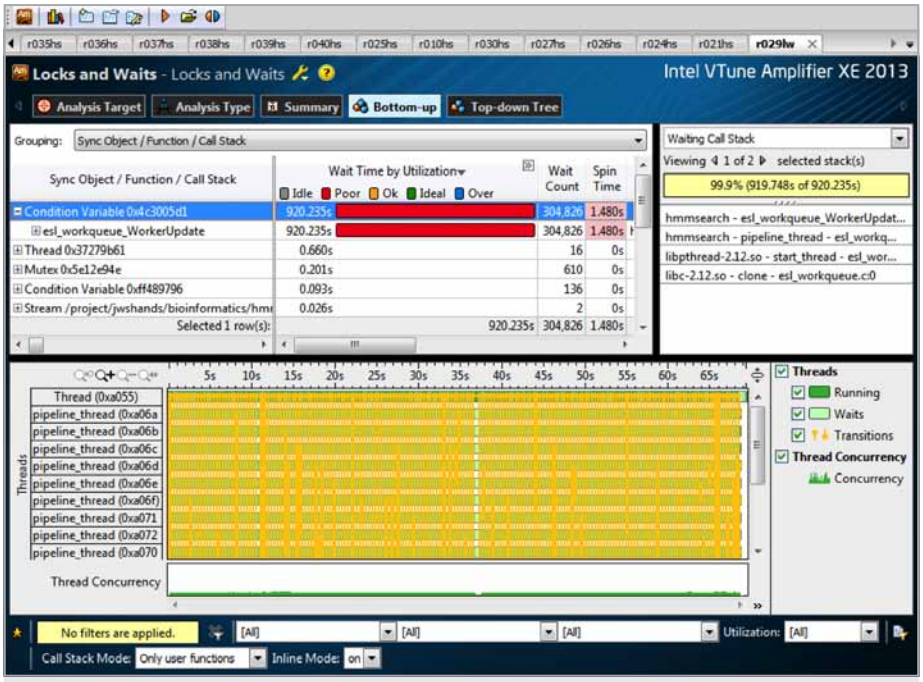


図 6

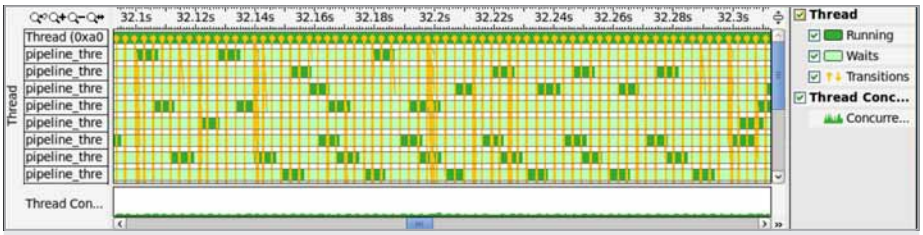


図 7

図 8

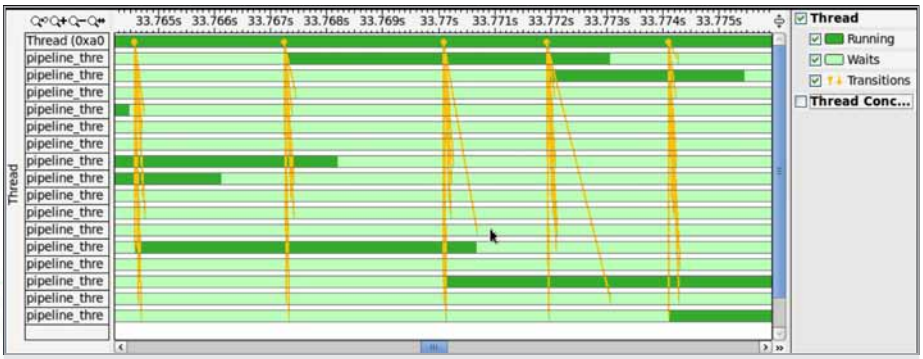


図 9

インテル® VTune™ Amplifier XE の解析により、最も時間を費やしているコードは、msvfilter.c ファイルの p7_MSVFilter で SSE 組込み関数を使用して最適化された MSV アルゴリズムであることが分かりました。組込み関数で最適化されたコードには、より高速になるように、ベクトル化による最適化も含まれています。

インテル® コンパイラーが非組込み関数の最適化されたコードを効率良くベクトル化できるかどうかを確認するため、generic_msv.c ファイルの p7_GMSV 関数で最適化されていない C コードを使用するようにアプリケーションをコンパイルしました。この場合も、MSV アルゴリズムが hotspot であることが表示されました。(図 12)

MSV アルゴリズムの最も時間を費やしている部分が、AVX 命令やインテル® Xeon® プロセッサーの YMM レジスターを活用していない 1 つのループであることも判明しました。(図 13)

このコードを使用した hmmsearch の実行時間は、約 4 分 30 秒です。

```
# CPU time:4137.39u 5.02s
01:09:02.41 Elapsed:00:04:30.08
```

インテル® コンパイラーの `-opt-report` オプションを使用すると、各関数のインライン展開、ループ、メモリー、ベクトル化、並列化に関する最適化レポートが表示されます。p7GMSV 関数では、ループがベクトル化されていないことが示されました。

```
generic_msv.c(80:7-80:7):VEC:p7_GMSV:ループはベ
クトル化されませんでした: ベクトル依存関係が存在しています。
```

コードを再構築して、コンパイラーがループをベクトル化し、インテル® Xeon® アーキテクチャーを活用するコードを生成するようにしたところ、最適化レポートで、2 つのループがベクトル化されたことが示されました。

```
generic_msv.c(88:7-88:7):VEC:p7_
GMSV:ループがベクトル化されました。

generic_msv.c(108:7-108:7):VEC:p7_
GMSV:ループがベクトル化されました。
```

さらに、インテル® VTune™ Amplifier XE のアセンブリー・ビューで、AVX 命令がより大きな YMM レジスターとともに使用されていることも示されました。(図 14)

アプリケーションの実行時間はオリジナルの実行時間の約半分になりました。

```
# CPU time:2207.74u 4.96s 00:36:52.69
Elapsed:00:02:28.16
```

次に、インテル® Inspector XE を使用して hmmsearch のスレッドエラーとメモリーエラーを調べます。インテル® Inspector XE は、アプリケーションの信頼性を高めるために不可欠な、アプリケーションのメモリーとスレッドの動作に関する詳細な情報を提供します。その強力なスレッドチェッカーとデバッガーを使用すると、実行されるコードパスの潜在的なエラーを容易に見つけることができます。たとえエラーが発生しなかった場合でも、断続的なエラーや非決定的なエラーを検出できます。

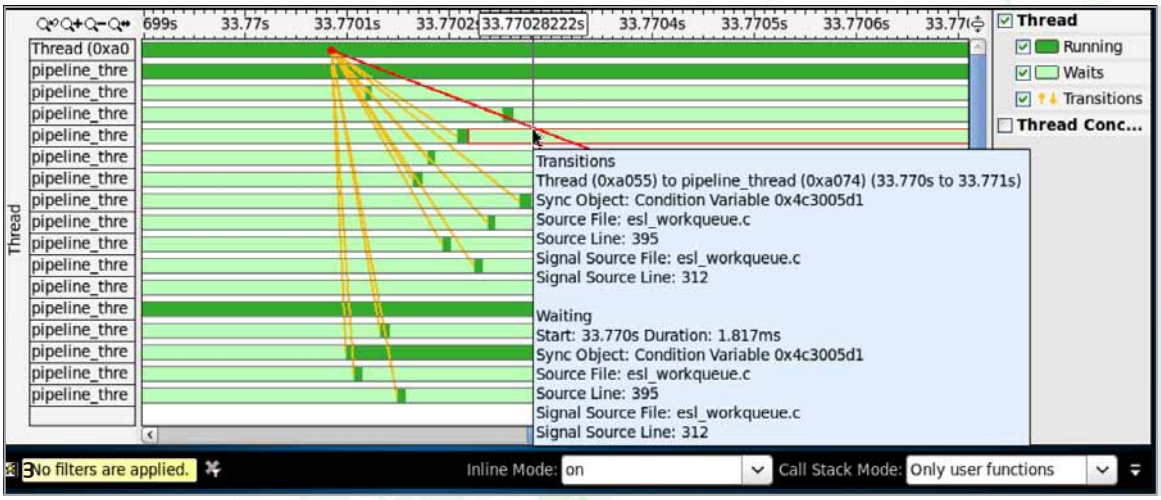


図 10

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

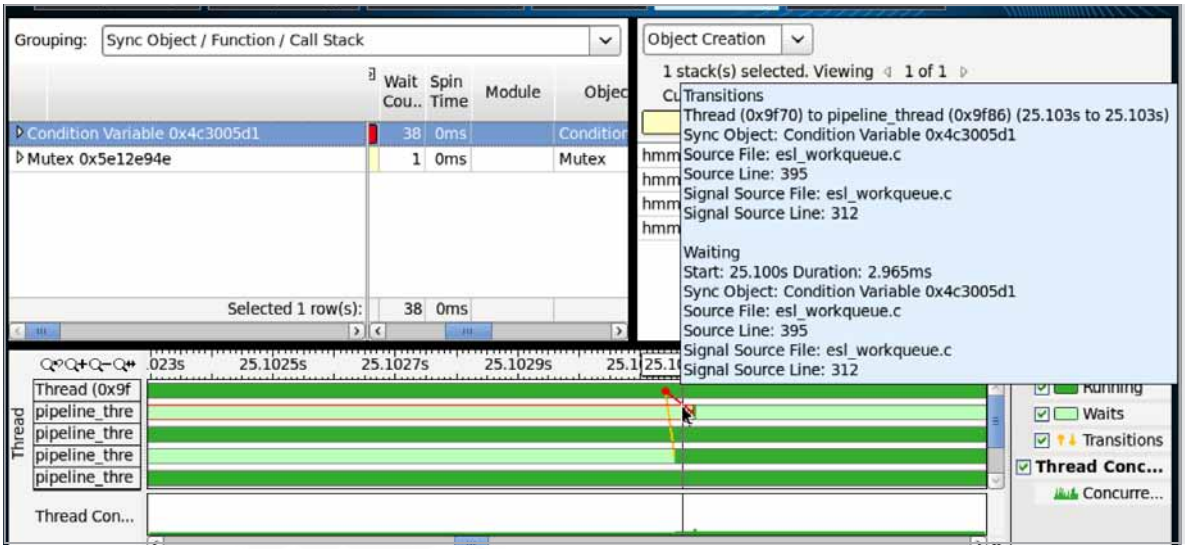


図 11

「インテル® C コンパイラーおよびライブラリーはより高速なコードを生成し、インテル® VTune™ Amplifier XE はボトルネックを検出し、インテル® Inspector XE はエラーが発生する前にメモリーエラーとスレッドエラーを正確に特定します。HMMER のようなアプリケーションを開発する場合、これらのツールはすべて不可欠なものといえるでしょう。」

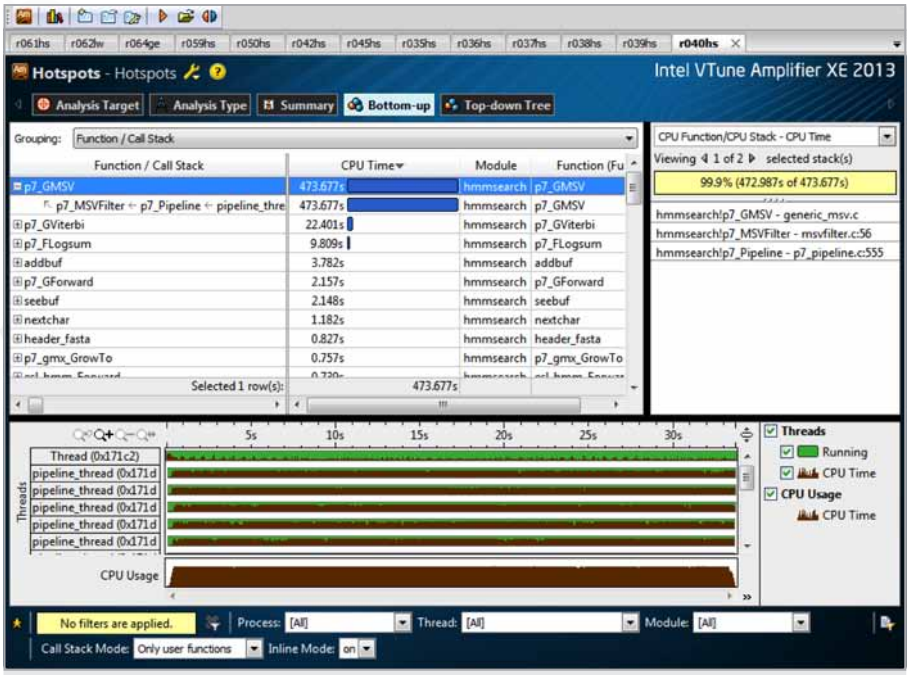


図 12

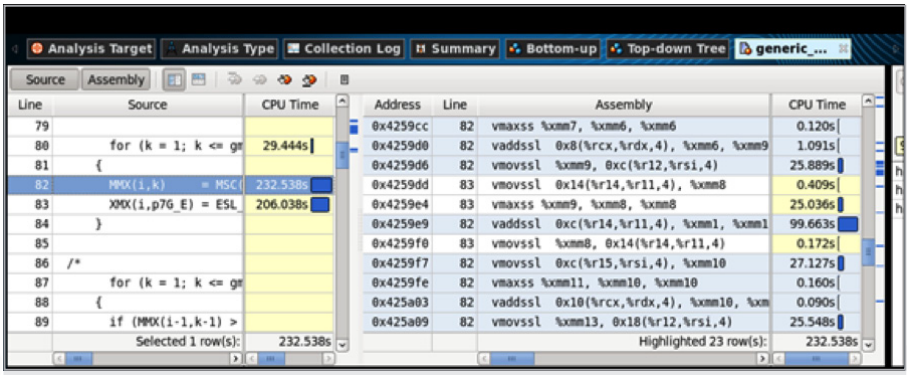


図 13

インテル® Inspector XE は、メモリーリーク、メモリー破壊、一貫性のないメモリー API の使用、データ競合、デッドロック、スレッド間のメモリーアクセスを検出します。

インテル® VTune™ Amplifier XE と同様に、特別なビルドを作成したり、データを収集するためにアプリケーションにコードを追加する必要はありません。

メモリーエラーやスレッドエラーの検出では著しいオーバーヘッドが発生するため、より小さなシーケンス・データベース・ファイルを使用し、スレッド数を減らすオプションを指定して hmmsearch を実行しました。

メモリー問題検出モードでインテル® Inspector XE を実行すると、初期化されていないメモリーへのアクセスがいくつか見つかりました。(図 15)

Locate Memory Problems ペインの行を右クリックすると、初期化されていないメモリーへのアクセス問題に関する説明が表示されます。(図 16)

デッドロックとデータ競合の検出モードでは、問題は見つかりませんでした。(図 17)

問題を修正したら、インテル® コンパイラーに含まれるインテル® Cilk™ Plus を使用してアプリケーションのパフォーマンスを向上させます。インテル® Cilk™ Plus は C および C++ の拡張で、マルチコア・プロセッサーにおいてプログラムのパフォーマンスを向上させる、迅速、容易、かつ信頼性の高い方法を提供します。インテル® Cilk™ Plus はオープン・スタンダードで、近々 GCC 4.7 でも利用できるようになります。インテル® Cilk™ Plus は、3 つのキーワード (`cilk _ for`、`cilk _ spawn`、`cilk _ sync`) を使用して新規および既存の C/C++ プログラムを並列化します。

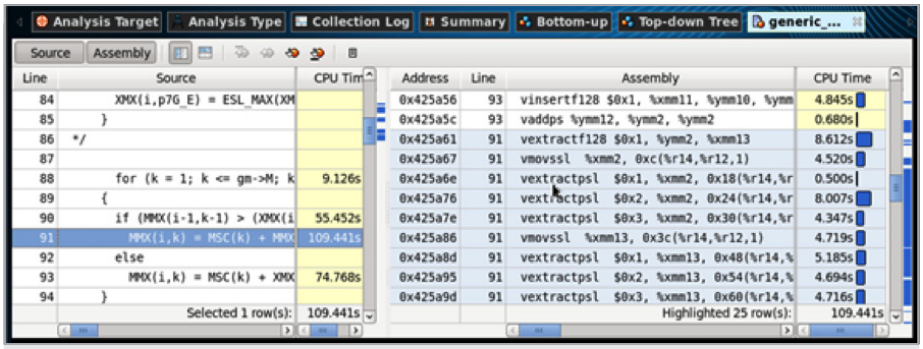


図 14

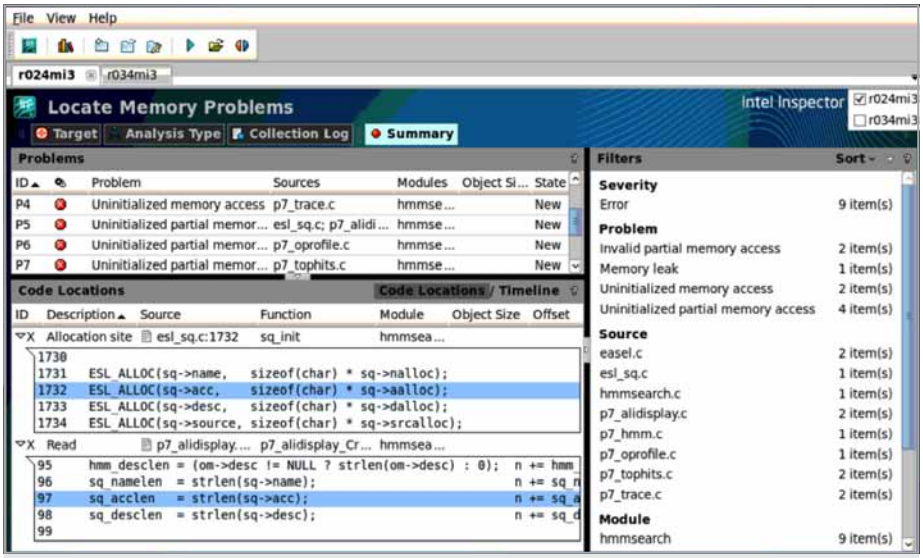


図 15

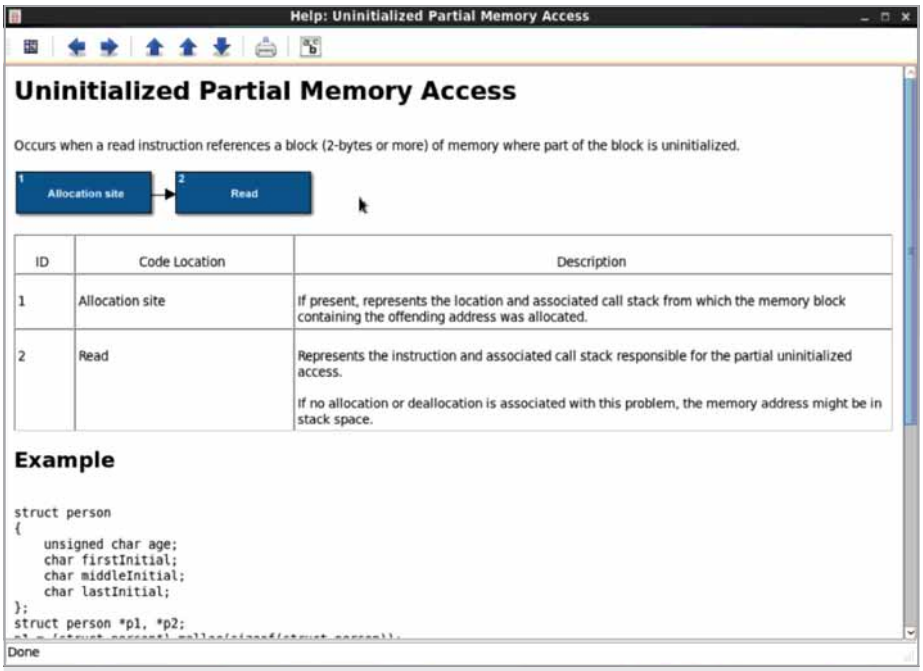


図 16

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

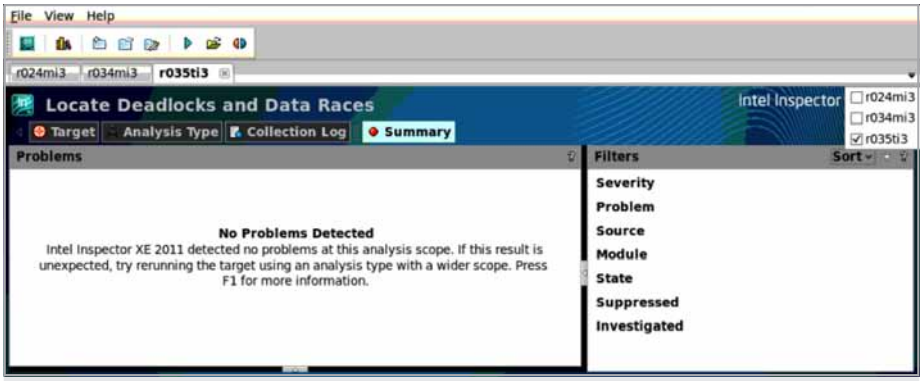


図 17

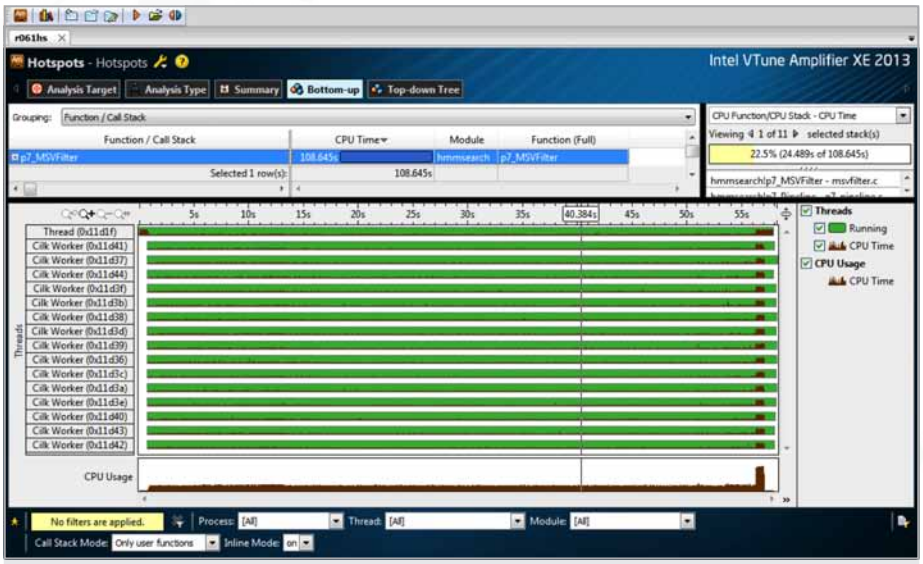


図 18

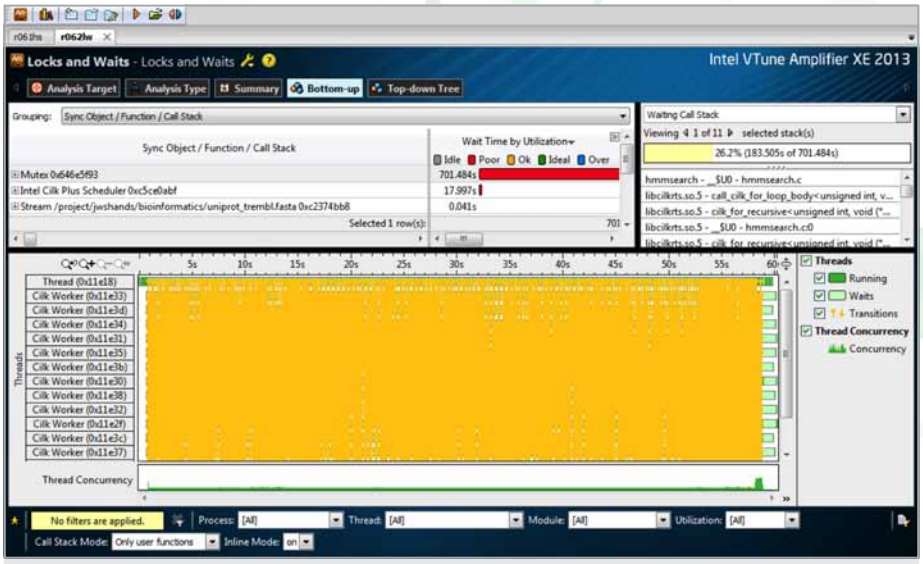


図 19

ここでは、インテル® Cilk™ Plus を使用してスレッド、mutex、条件変数、ワークキューを管理するコードを置換し、より優れたスケジューリングが行われるようにします。しかし、データファイルの読み取りについてはスレッドを同期させる必要があるため、アプリケーションの一部はシリアルになります。

インテル® VTune™ Amplifier XE で hmmsearch の hotspot を表示すると、シーケンス・データベース・ファイルを読み取るコードの周囲で mutex による同期のために、CPU が完全には利用されていないことが分かります。しかし、インテル® Cilk™ Plus 実装バージョンの実行時間は 58.272 秒で、オリジナルの実行時間 67.418 秒よりも短縮されています。(図 18)

再びインテル® VTune™ Amplifier XE のロックと待機解析を行ったところ、まだ多くのスレッドの移行が行われていました。(図 19)

ロックと待機解析のスレッドペインを拡大すると、スレッドの移行はワーカースレッド間で行われ、各ワーカースレッドで実行されるファイルの読み取りを保護する mutex の処理に関連するものであることが判明しました。(図 20)

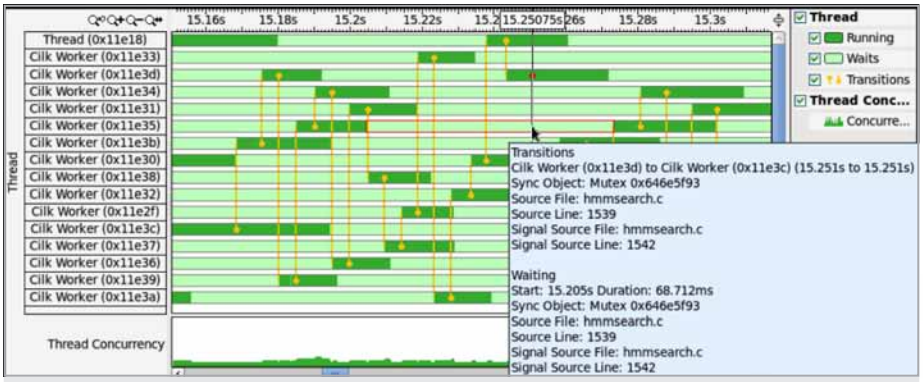


図 20

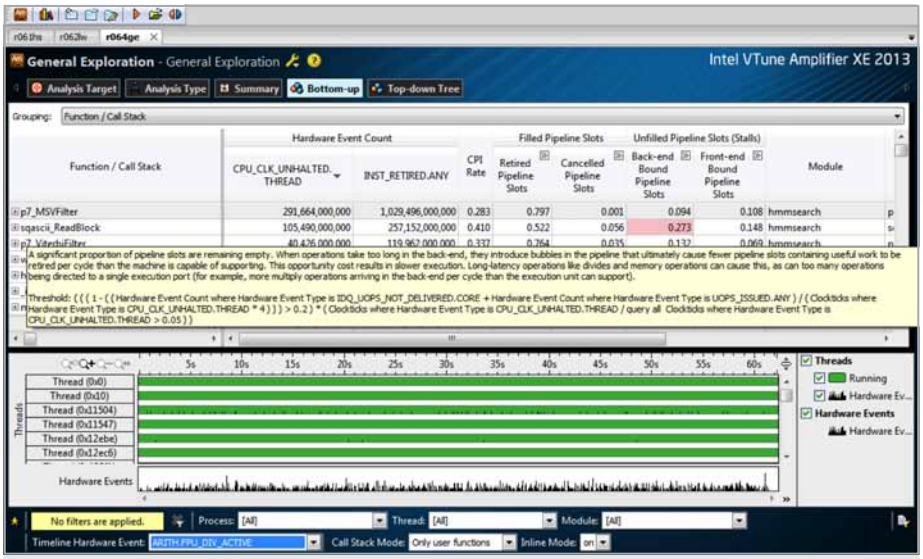


図 21

このほかのインテル® Cilk™ Plus の強力な機能に、配列表記 (アレイ・ノーテーション) があります。配列表記はインテル固有の言語拡張で、C/C++ 配列で宣言されたデータを並列に表現する方法を提供します。データを並列に配列表記することにより、エイリアスや依存性の解析にそれほど頼ることなく、コンパイラーで並列化やベクトル化が可能になります。

より大幅にパフォーマンスを向上させるには、ファイルの読み取りによるアプリケーションのシリアル化問題を解決する必要があります。十分なメモリーが利用可能であれば、計算の前にメモリーにデータを読み込むことで同じデータを使用して後の計算をスピードアップできますが、uniprot_trembl.fasta データファイルを使用するとマシンのメモリー容量をオーバーしてしまうため、この方法は現実的ではありません。

さらなるパフォーマンスの向上は、インテル® コンパイラーのオプションを活用することにより達成できます。インテル® コンパイラーのデフォルトの命令セットは SSE2 で、ターゲットマシンはインテル® Xeon® プロセッサー・システムであるため、コンパイルホストでサポートしている最も高水準の命令セットを生成する -xhost オプションを使用して、AVX 命令と大きなレジスターサイズを活用します。

もう 1 つの重要なコンパイラー・オプションは、ファイル間でプロシージャー間の最適化を有効にする -ipo オプションです。これは、複数ファイル IPO (Interprocedural Optimization) またはプログラム全体の最適化 (WPO: Whole Program Optimization) とも呼ばれます。このオプションを指定すると、コンパイラーは別々のファイル内で定義されている関数への呼び出しについて関数のインライン展開を実行します。

インテル® コンパイラーがループをベクトル化または並列化するために何を行うべきか調べるには、オブジェクトや実行ファイルを生成せずにレポートを出力する -guide オプションを使用します。インテル® コンパイラーのガイド付き自動並列化機能は、シリアルコードのアプリケーション・パフォーマンスを向上させるアドバイスを提供します。通常、アドバイスは 3 つのカテゴリー (ソースコードの修正、プラグマの使用、コンパイラー・オプション) に分けられます。

hmmsearch でこのオプションを使用した後のアドバイスを次に示します。

```
esl_vectorops.c(161):
リマーク #30536:(LOOP) 必要に応じて、-fargument-noalias オプションを追加してコンパイラーによる型ベースの一義化解析を向上できます (オプションはコンパイル全体に適用されます)。
これにより、行 161 のループがベクトル化され最適化が向上します。
```


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

-parallel オプションを追加すると、インテル® コンパイラーは並列に実行できる可能性のある単純な構造のループを検出して、そのループのマルチスレッド・コードを自動的に生成します。ガイド付き自動並列化オプションを -parallel とともに使用すると、さらなる並列化が推奨されることがあります。

```
msvfilter.c(106): リマーク #30525:(PAR)
"#pragma loop count min(1024)" 文をループの
直前 (行 106) に挿入して、このループを並列化します。[確
認] ループの反復回数が 1024 以上であることを確認して
ください。
```

インテル® VTune™ Amplifier XE のハードウェア・イベント・カウンター収集機能を使用して、パフォーマンスに影響を与えるアプリケーション・コードのボトルネックについてアドバイスを得ることもできます。インテル® VTune™ Amplifier XE は、収集したデータで調査すべきパフォーマンス問題が含まれる部分を強調して表示します。hmmsearch を実行した例を示します。(図 21)

まとめ

インテル® ソフトウェア開発ツールは、アプリケーションのパフォーマンスを向上し、ハイパフォーマンス・コンピューティングとエンタープライズ・アプリケーションで必要とされるコードの品質、セキュリティー、信頼性を高められるように開発者を支援します。インテル® C コンパイラーおよびライブラリーはより高速なコードを生成し、インテル® VTune™ Amplifier XE はボトルネックを検出し、インテル® Inspector XE はエラーが発生する前にメモリーエラーとスレッドエラーを正確に特定します。最新世代のマルチコア・プロセッサーで HMMER のようなアプリケーションを開発する場合、これらのツールはすべて不可欠なものといえるでしょう。

インテル® Advisor XE は並列化による生産性の向上をどのように支援しているのか

RAVI VEMURI

宇宙探検、油田 / 天然ガス調査、ハリウッド映画、軍事戦略に共通しているものは? モデリング、シミュレーション、研究、絵コンテ、偵察などのフレーズが思い浮かんだことでしょう。これらはすべて、選択ミス、失敗、過失によるコストを削減するように意図されており、プロジェクトの成功と生産性の向上を支援します。

ソフトウェアの並列化も同様に、コードが並列化に適しているかを評価する並列性調査を行うことで利益が得られます。これまで、並列性調査を行うことができるツールは限られていましたが、インテル® Advisor XE 2013 の登場により、この状況は大きく変わりました。インテル® Advisor XE は、インテル® Parallel Studio XE ツール・スイートの最新のコンポーネントです。

ソフトウェアの並列化は、コードが潜在的に不安定になり、リスクがあり、割高で、さらに複雑です。現在の試行錯誤アプローチは生産的ではなく、デッドエンドの高いリスクがあります。測定したデータ (例えば、hotspot) に基づいてコードの並列化を開始するのは 1 つの方法ですが、コードが適切にスケールされる場合もあれば、適切にスケールされない場合もあります。不適切な並列化による安定性問題はさらに潜在的で、コードが製品化されてしばらく経過した後に表面化するため、修正に多大な費用がかかります。

インテル® Advisor XE は、並列化可能なコード領域の特定を支援するように設計されており、並列化のパフォーマンス (スケーラビリティー) と並列化可能なコード領域のコード / データ共有問題 (正当性) の調査に重点を置いています。インテル® Advisor XE は、並列化のスケーラビリティーと正当性を調査するため、プログラム内の異なる領域をモデル化します。モデル化の結果から、(デッドエンドを回避するために) 並列化しないコード領域と、マルチコアのパフォーマンスを活用するために実際に並列化するコード領域を適切に判断できます。

この手法を使用すると、データ共有問題が**発生する前に**問題を修正することができます。正当性問題を修正した後、並列化のためにコードを準備する場合にも、機能的に変更されていない正しい既存のテスト・フレームワークを使用してプログラムを検証できます。

並列化作業にインテル® Advisor XE を利用することにより、リスクが大幅に減り、より適切に並列化を行うことができます。このツールは、アーキテクトやシニアエンジニアしか生産的な並列化ができない現在の状況を一変させ、ソフトウェア最適化の知識があるプログラマーであれば誰でも生産的な並列化が行えるようになります。

アプリケーションの可能性を知るのはとても楽しいことです。製品の詳細は、インテル® Advisor XE 製品ページ (英語)を参照してください。製品についてお気付きの点がありましたら、是非お知らせください。

ポインターチェッカー: 範囲外のメモリーアクセスの迅速な検出

テクニカル・コンサルティング・エンジニア Kittur Ganesh

この記事では、プログラムの検出が困難なバグを正確かつ容易に検出できる、ポインターチェッカーと呼ばれる強力な新しい機能を紹介します。インテル® C++ Composer XE 2013 製品に含まれるこの機能は、コンパイラーに統合され、ビルドシステムにシームレスで強力な機能を追加します。その優れた実装とエラー報告機能は、プログラムの潜在的な問題に関する正確な情報を提供します。この新しい機能のベータテスト中に、多くのユーザーから、このツールで多数の問題が発見できたというフィードバックが寄せられたことを非常にうれしく思います。

C/C++ ポインターのセマンティクスが適切に定義されている場合でも、多くのアプリケーションは範囲外のメモリーアクセスを行うことがあります。これらのアクセスは検出できないため、データが破壊されたり、脆弱性を利用した悪意のある攻撃を受ける可能性が高くなります。ポインターチェッカーは、ポインターによるすべてのメモリーアクセスを完全にチェックして、メモリーが破壊される前に範囲外のメモリーアクセスをキャッチします。

マルチコア・プロセッサーの出現により、データを頻繁に作成、格納、共有し、ポインターからメモリーにアクセスする、データとスレッドの並列化をプログラムする必要が高まっています。C および C++ 言語ではポインターによるメモリーアクセスの適切なセマンティクスが定義されますが、これらのポインターを制限なく使用することも可能です。その場合、メモリー中のユーザーデータのアクセスや書き込みに対する保護は行われないため、ポインターで任意の操作が行われ、(故意でないデータの修正により結果がランダムに見える、プログラムで非常に検出が困難なエラーである) 検出できない範囲外 (OOB) のメモリーアクセスが発生します。

ポインターの下限と上限が適切に定義されていても、言語 (つまりコンパイラー) によっては、パフォーマンスと速度の関係から範囲チェックを行いません。その結果、アプリケーションのさまざまな部分における (データ破壊、不安定なプログラムの動作、システム・セキュリティー問題などを引き起こす) 潜在的なバッファー・オーバーフローやバッファー・オーバーランが発生したり、多くのソフトウェアの脆弱性を利用した悪意のある攻撃を受ける可能性が高くなります。

インテル® Parallel Studio XE 2013 の新機能であるポインターチェッカーは、範囲チェック (ポインターによるすべてのメモリーアクセスの完全なチェック) を行い、コード範囲外のアクセスを認識します。この記事では、アプリケーションでポインターチェッカーを使用できるように、ポインターチェッカーの概要と使用モデルについて説明します。

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

概要

ポインターチェッカーは、インテル® Parallel Studio XE 2013 の重要な機能です。ポインターチェッカーの主な機能は、Windows* または Linux* オペレーティング・システムの高水準言語 (C および C++) で開発されたアプリケーションで発生するバッファー・オーバーフローとバッファー・オーバーランを検出することです。バッファー・オーバーフローとバッファー・オーバーランは、プログラムがバッファーにデータを書き込むときに、バッファーの範囲を超えて隣接するメモリーに書き込みを行う、メモリー安全性違反の特別なケースです。例えば、図 1 のコードスニペットで示されるようなバッファーの配列を考えてみましょう。

```
char *buf = (char *)malloc(5);
for (int i=0; i<=5;i++) {
 buf[i] = 'A' + i;
}
```

図 1

バッファー・オーバーフローは、配列が保持できる項目よりも多くの項目を配列に格納しようとした場合に発生します。通常、書き込みやストアで発生します。一方、バッファー・オーバーランは、配列の最後を過ぎても読み取りを続けた場合に発生します。通常、読み取りやロードで発生します。さらに、単純なコーディング・エラーは、多くの場合、検出や修正が非常に困難です。例えば、ポインターは void 型のポインターに一度キャストして型をマスクし、その後別の型にキャストして使用されることがよくあるため、アプリケーションでエラーの原因を特定することが非常に困難になります。前述したように、ポインターの下限と上限が適切に定義されていれば、ポインターチェッカーでポインターによるすべてのメモリーアクセスの範囲チェックを行うことで、読み取りや書き込み操作で使用する前にポインターが範囲内にあることが保証されます。

ポインターチェッカーの機能は、**コンパイラー・オプション**を指定して有効にします。ポインターチェッカーを有効にしてアプリケーションをビルドすると、添字付き配列アクセスを含む、アプリケーションで発生するすべての範囲外メモリーアクセスが識別されレポートされます。ポインターチェッカーは、**ダングリング・ポインター** (解放されたメモリーを指すポインター) を検出することもできます。ダングリング・ポインターの検出を有効にしてアプリケーションをビルドした場合、間接アクセスでダングリング・ポインターを使用しても範囲外エラーになります。ポインターチェッカーで提供されるもう 1 つの便利な機能は、**無次元配列**の範囲チェックです。この機能は、extern でデータを共有する複数の開発者により作成された多くの異なるモジュールが統合されているアプリケーションで特に重要です。

この記事の続きを読む (英語)

BLOG highlights

アムダールの法則によるフラストレーションの最小化とチューニング効果の最大化

SHANNON CEPEDA

先日、アプリケーションの hotspot を最適化したのに思ったほどアプリケーション全体のパフォーマンスが向上しなかったというユーザーから質問がありました。この質問は、パフォーマンス・チューニングの反復プロセス中によく寄せられます。通常は、2 つの理由のいずれかによるものです。

1. ある領域を最適化したことで別の領域の効率が低下した。これはパフォーマンス・チューニングではよくあることで、プロセスが**反復的**である理由の 1 つです。ある関数のコードを変更するときに別の部分のパフォーマンスが低下するかどうか予想することは困難であるため、この状況が発生することは避けられません。この状況が発生することを完全に防ぐことはできないかもしれませんが、適切なドキュメントとパフォーマンスの変更を測定できるインテル® VTune™ Amplifier XE のようなツールを使用することにより、この状況がいつ発生しているかを確認することはできます...

この記事の続きはこちら (英語) でご覧になれます。

Go-Parallel.com (英語) でご覧になれます。

Go Parallel では、この他にも次のような関連トピックのブログをご覧いただけます: Translating Multicore Power into Application Performance (マルチコアのパワーでアプリケーション・パフォーマンスを引き出そう)。

インテル® (Visual) Fortran Composer XE

の新しい並列プログラミング機能

開発者向け製品部門
Steve Lionel

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

Fortran プログラマーは長年、自動並列化、OpenMP*、MPI のような Fortran 規格にない手法を使用して並列プログラミングを行ってきました。2010 年後半に国際規格として承認された Fortran 2008 で初めて、並列プログラミングの手法として 2 つの言語機能が採用されました。(1 つの言語機能のみでは並列にできません。)

この記事では、この 2 つの新機能、DO CONCURRENT と Co-Array の概要を説明します。前者は非常に容易に理解できますが、後者は容易ではありません。

DO CONCURRENT

1990 年代初めに、ハイパフォーマンス・コンピューティングおよび並列処理用に Fortran 90 言語を拡張する試みが行われました。High Performance Fortran (HPF) は、配列演算を並列に実行できるようにした手法を用いて Fortran 90 の配列構文の構築を試みました。HPF では、FORALL および WHERE 構文、副作用のない PURE プロシージャー、スキャッター/ギャザーのような演算用の組込みプロシージャーが導入されました。HPF はあまり普及しませんでしたが、その一部の機能は 1997 年に承認された Fortran 95 規格で採用されました。

さまざまな HPF 機能のうち、FORALL ほど誤解されたものはないでしょう。FORALL 構文の例を次に示します。

```
REAL ::A(10, 10), B(10, 10) = 1.0
…
FORALL (I = 1:10, J = 1:10, B(I, J) /= 0)
A(I, J) = REAL(I + J + 2)
B(I, J) = A(I, J) + B(I, J) * REAL(I * J)
END FORALL
```

FORALL キーワードの後の括弧付きのリストは **forall-ヘッダー**と呼ばれ、インデックス名がとる値の範囲を指定する 1 つ以上の **forall-トリプレット**が含まれます。この例では、2 つのインデックス名 I と J があり、それぞれ 1 から 10 の値をとることが指定されています。Fortran 90 の配列表記のように、トリプレットの 3 番目の要素で指定されていない場合、インクリメントは 1 です。最後の部分は、FORALL 構文の本体 (2 つの代入) が実行される条件を決定するマスク式です。

多くの Fortran プログラマーは、FORALL のループ (このケースでは 2 つの入れ子のループ) を目にしたとき、式が false だった場合、次の反復にスキップすると考えるでしょう。しかし、FORALL はループ構文**ではなく**、「マスク配列の代入」です。ループとして見なせるのであれば、各反復で両方の代入が実行され、代入を並列実行できるはずです。しかし、FORALL の定義は異なります。まず、全インデックス名のすべての組み合わせで最初の代入が完全に実行され、マスクでフィルターされます。次に、すべての組み合わせで 2 つ目の代入が完全に実行され、マスクでフィルターされます。FORALL 構文の内部で、関数が PURE の場合、代入文は関数を参照しますが、FORALL に、代入文、WHERE 構文、ほかの FORALL 以外の文を記述することはできません。

FORALL は貴重な試みでしたが、並列に代入を行うための規則に制限が多かったため、Fortran コミュニティーに広く受け入れられることはありませんでした。その結果、Fortran 2008 では、FORALL の改良ともいえる DO CONCURRENT が導入されました。

DO CONCURRENT 構文は、DO と FORALL の組み合わせのように見えます。実は、DO CONCURRENT の先頭は FORALL ヘッダー構文を使用しています。次に例を示します。

```
DO CONCURRENT (I=1:N)
T = A(I) + B(I)
C(I) = T + SQRT(T)
END DO
```

FORALL と同様に、マスクはオプションです。マスクが指定された場合、マスク式が true であるもののみに、インデックス名のアクティブな組み合わせのセットが減らされます。FORALL と異なり、DO CONCURRENT の範囲はそれぞれが反復で、すべてのアクティブなインデックスの組み合わせで個別に実行されます。

DO CONCURRENT にもいくつかの制限があります。例えば、構文外への RETURN や GO TO はできません。また、別の反復で定義された変数、または未定義にされた変数は参照できません。ある反復で書き込んだレコードを別の反復で読み取らない限り、DO CONCURRENT で I/O 処理を行うことができます。FORALL と同様に、構文内から呼び出されるプロシージャーは PURE (副作用がない) でなければなりません。ループ間に依存性がないことを保証するのはプログラマーの責任です。コンパイラーは依存性を確認しません。

DO CONCURRENT はインテル® [Visual] Fortran Composer XE 2011 でサポートされ、自動並列化 (`/Qparallel` または `-parallel`) を有効にすると、コンパイラーは構文を並列に実行しようとします。しかし、DO CONCURRENT が並列実行される保証はなく、反復が実行される順序は予測できません。プログラマーによりループ伝播の依存性がないことが保証されているため、DO CONCURRENT の使用は自動ベクトル化も助けます。

Co-Array

MPI プログラマーの方は、データを収集し、MPI_SEND で収集したデータを別の「ノード」で実行しているプログラムのコピーに送り、MPI_RECV を使用して結果を戻すという基本的な操作をご存知でしょう。(もちろん、この例はかなり単純化されています。)標準の Fortran 構文を使用してプログラムの別のコピーを操作でき、データを移動する呼び出しを追加する心配をしなくてもいいのであれば、この処理は非常に楽になります。

Co-Array Fortran は、1990 年代に Fortran 90 の拡張として最初に提唱され、以前は F- - (F マイナスマイナス) と呼ばれていました。Co-Array Fortran は、Fortran プログラムに並列処理を実装する単純な構文を提供します。(定義と実装は単純ではありませんが、構文は単純です。)2000 年代初めには Cray により T3E および X1 スーパーコンピューター向けに実装され、多少の変更が加えられた後、Fortran 2008 規格に採用されました。インテルは、インテル® [Visual] Composer XE 2011 Linux* 版および Windows* 版で、メインストリーム・コンピューター向けに Fortran 2008 の Co-Array を最初に完全実装したバージョンをリリースしました。

Co-Array Fortran の基本的な概念を次に示します。

- イメージ: アプリケーションの複数のコピーが並列に実行されます。コピーはそれぞれ**イメージ**と呼ばれます。
- Co-Array: CODIMENSION 属性が指定されると、変数は **Co-Array** になります。多少混乱するかもしれませんが、スカラーも Co-Array になります。規格では Co-Array を非ゼロの corank を含むエンティティーとして定義しており、スカラーでも配列でもかまいません。**codimension** (および **coindex**) は角括弧 [] で表示されます。

Co-Array はアプリケーションのすべてのイメージで分割され、各 Co-Array の部分は個々のイメージのローカルメモリーに格納されます。このプロパティーは PGAS (Partitioned Global Address Space) 並列プログラミングの概念と関連付けられます。ここで、Co-Array は共有「アドレス空間」に存在しますが、イメージ固有のセグメントは個々にアドレス可能です。単純な例を見てみましょう。

```
real, dimension(10,20), codimension[*] ::A
```

配列 A を次元 10x20 と 1 つの codimension で定義します。

codimension を追加の次元と見なすと良いでしょう。ちなみに、Fortran 規格では次元と codimension の合計は 15 に制限されています。(インテル® Fortran では拡張して 31 までサポートしています。) codimension の最後の上限 cobound は * でなければなりません。* は実行時にはイメージの数を示します。実行時に 8 つのイメージがある場合、A の cobound は 1:8 です。

次元と同様に、下限と上限を含む複数の codimension を使用でき、最後の 1 つのみ上限として * を含めることができます。次のようになります。

```
integer, codimension[4,2:6,3:*]::B
```

Co-Array は、角括弧で指定する coindex を使用しても、しなくても参照することができます。coindex を使用しない場合は、Co-Array のイメージの部分を参照します。coindex を使用する場合は、参照するイメージの coindex を指定します。

coindex に必要な数のイメージがない場合はどうなるのでしょうか? 通常の配列ではエラーになりますが、それとは異なり、Co-Array の「形状」は規則正しくありません。上記の例 B の場合、Co-Array の各「レイヤー」を満たすには 20 イメージ必要です。39 イメージある場合、coindex は [4,6,4] ではなく [3,6,4] です。(Fortran は左の添え字が最も速く変化する列優先順で処理を行うことに注意してください。)

イメージの数、イメージのインデックス、Co-Array の cobound は組込みプロシージャーを使用して調べることができます。

Co-Array の利点は、Fortran 言語に完全に統合されていることです。次のように、通常の変数が使用できるほとんどの場所で Co-Array を使用することができます。

- 式および代入
- プロシージャー呼び出しの引数
- I/O 文

Co-Array を使用するプログラムが見やすいのはこのためです。例えば、問題をデータのブロックに分割する Jacobian ソルバーを考えてみましょう。ほとんどの計算にイメージのローカルブロックが含まれ、各ブロックの端で、隣接するイメージのチャンクの端にある「halo セル」からの値を考慮する必要があります。このようなコードに Co-Array を使用した場合の例を次に示します。(図 1)

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

「インテル® [Visual] Fortran Composer XE は、単一システムで実行する “共有メモリー” モードで Co-Array をサポートします。-coarray (または /Qcoarray) コンパイラー・オプションを追加して Co-Array プログラムをビルドし、生成された実行ファイルをそのまま実行できます。特別な設定は必要ありません。」

```
my_subgrid( 0, 1:my_M) = my_subgrid( my_N, 1:my_M)[my_north_P,me_Q]
my_subgrid( my_N+1, 1:my_M) = my_subgrid( 1, 1:my_M)[my_south_P,me_Q]

my_subgrid( 1:my_N, my_M+1) = my_subgrid( 1:my_N, 1 )[me_P, my_east_Q]
my_subgrid( 1:my_N, 0 ) = my_subgrid( 1:my_N, my_M )[me_P, my_west_Q]
```

図 1

Fortran は、プログラミングが容易になるように追加の Co-Array の動作を定義します。次に例を示します。

- すべてのイメージが Co-Array を一貫して完全に割り当てるようにするため、すべての割り当てが同期ポイントの ALLOCATABLE Co-Array にすることができます (実際、最も一般的な使用法です)。
- すべてのイメージが I/O を行えます。通常は、各イメージに個別のユニット番号がありますが、すべてのイメージで「標準出力」(ユニット 6) があらかじめ接続されます。他の実装では「ストリームをマージする」必要はありませんが、インテル® Fortran はストリームをマージするため、すべての標準出力はイメージが実行されるコンソールに表示されます。「標準入力」(ユニット 5) はイメージ 1 にのみあらかじめ接続されます。
- すべてのイメージには、最初と最後に暗黙の同期ポイントがあります。

イメージ間を同期する方法はいくつかあります。SYNC ALL 文はすべてのイメージが同じ回数の SYNC ALL を実行するまで待機します。SYNC MEMORY は、継続する前にメモリーの更新がすべて完了していることを確認します。SYNC IMAGES は SYNC ALL に似ていますが、指定したセットのイメージに同期を制限します。

組込みモジュール ISO_FORTRAN_ENV で定義された LOCK_TYPE を使用して宣言し、LOCK および UNLOCK 操作を行うロックもあります。また、ATOMIC_DEFINE および ATOMIC_REF 組込みプロシージャーで整数および論理変数のアトミックな (中断されない) 読み取りと書き込みを行うこともできます (インテル® Fortran Composer XE 2013 で新しくサポートされました)。

インテル® [Visual] Fortran Composer XE は、単一システムで実行する “共有メモリー” モードで Co-Array をサポートします。-coarray (または /Qcoarray) コンパイラー・オプションを追加して Co-Array プログラムをビルドし、生成された実行ファイルをそのまま実行できます。特別な設定は必要ありません。クラスターで「分散メモリー」モデルのサポートを追加するには、(クラスターの構築に加えて) インテル® Cluster Studio のライセンスが必要です。Windows* クラスターの場合も同様です。(Windows* における分散メモリー Co-Array アプリケーションのサポートは、インテル® Visual Fortran Composer XE 2011 Update 6 で追加されました)。

Co-Array および DO CONCURRENT を含む Fortran 2008 規格の詳細は、Fortran 言語標準委員会のドキュメントを参照してください。

次の FORTRAN 規格の CO-ARRAY (英語)

FORTRAN 2008 の新機能 (英語)

FORTRAN 2008 規格 (英語)

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® コンパイラーを使用し、実行するたびに数値再現性のある結果を取得

テクニカル・コンサルティング・マネージャー、インテル® マス・カーネル・ライブラリー　Todd Rosenquist
マネージャー、インテル® MKL テクニカル・ストラテジー　Shane Story

ハリウッドからウォールストリートに至るまで、浮動小数点演算を行うすべてのアプリケーションは、優れたパフォーマンスと、実行するたびに完全に同じになる結果、つまり再現性のある結果の提供という課題に長い間直面してきました。再現性のある結果が得られない主な要因は浮動小数点演算の非結合性によるものですが、ランタイム、選択可能な最適化コードパス、非決定性スレッド化と並列処理、配列のアライメント、根本的なハードウェアの浮動小数点制御設定など、ほかの要因もあります。

この記事では、インテル® ソフトウェア開発ツールのユーザー向けにインテル® MKL とインテル® コンパイラーの機能を使用してパフォーマンスと再現性のある結果のバランスを取る方法を簡単に説明します。インテル® Parallel Studio XE 2013 で追加された新しい数値再現性機能により、実行するたびに一貫した再現性のある結果を得ることができます。

インテル® ソフトウェア開発ツールの数値再現性機能	
インテル® MKL 11.0	mkl_cbwr_set() MKL_CBWR (環境変数)
インテル® Composer XE 2013	-fp-model または /fp KMP_DETERMINISTIC_REDUCTION=yes

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

プロセッサーのクロックの上昇とともにソフトウェアのパフォーマンスが向上した時代が終わり、この数年間はクロックのレートはほとんど上昇しない代わりにマルチコア・システムが増加しています。マイクロプロセッサーの世代が新しくなるにつれ、ソフトウェアのパフォーマンスを高めるには、新しく追加された命令を使用してプロセッサーの性能を引き出すとともに、スレッド化アルゴリズムを使用して多くの計算コアを活用することが必須となっています。この変化に遅れないために、多くの開発者がソフトウェア開発ツールを活用しています。最適化コンパイラーは、命令およびデータレベルで並列化が可能で、プログラムで多くの計算を行う部分を自動的にスレッド化できます。ソフトウェア・ライブラリーは、コードをスレッド化するツールや高度に最適化されたスレッド化関数を呼び出すことにより自動的に並列化を行う手段を提供します。多くのソフトウェア・プログラマーは、これらのハイパフォーマンス・ツールを採用して、より優れたパフォーマンスを引き出しています。このとき、実行するたびに一貫しない結果が生成される可能性が高くなります。

次の 2 つのシナリオを考えてみましょう。アニメーション・スタジオのアーティストは、仮想世界でキャラクターを動かすことができる高度なモデリングツールを使用して毎日作業を行っています。これらのモデリングツールには、衣服、髪の毛、流動物の実世界における動作をシミュレートできる物理エンジンが含まれており、科学アプリケーションや工学アプリケーションで使用されるものに似た浮動小数点モデルを使用します。正確さや精度を常に優先して考える必要はありませんが、特にプロセスの初期段階では、同じ結果が得られることが最も重要です。アーティストが数秒のシーケンスを実行するたびに衣服の軌跡が少しずつ異なる場合、アーティストは創作プロセスをコントロールできず、シーンがさらなるレンダリングや後処理段階を経た後に、どの軌跡が使用されるのか定かではありません。この問題は互いに影響する多くのモデルが含まれるシーンではさらに悪化し、全く予測できない結果につながります。

2 つめのシナリオは、オプションの価格設定からリスク分析まで、さまざまなアプリケーションのアルゴリズムを開発しているウォールストリートの数学者についてです。この分野では、速く結果を得られることが (場合によっては巨額の) 収益につながります。これらのアルゴリズムを開発する「クォンツ」は、速く答えを得ることと、最も信頼性の高い答えを提供するために必要な、シミュレーション時間のバランス調整に日夜取り組んでいます。アルゴリズムのパフォーマンスが向上すれば、同じ時間でより速い決定またはより優れた決定を下すことができるようになり、いずれかのケースで優位に立つことができます。しかし、これらのモデルの一部である最適化された浮動小数点演算では、丸め誤差が発生することがあります。このため、初期の決定を検証するためにモデルを再度実行したときに、結果が以前と異なる可能性があります。プログラマーが避けようとする不確実性は、後から疑問や問題になるものです。

これらは、インテル® MKL のユーザーがこの数年間に遭遇したさまざまなシナリオ[1] の 2 つにすぎません。インテル® MKL は、これまで 15 年以上にわたって多くのアプリケーション分野で使用されてきた、高度に最適化された並列浮動小数点演算関数のライブラリーです。再現性のある結果が必要なアプリケーション・プログラマーにとっては、これまで保証はなく、ライブラリーのシーケンシャル・バージョンを実行するという制限付きのオプションが用意されているだけでした。

「ハリウッドからウォールストリートに至るまで、浮動小数点演算を行うすべてのアプリケーションは、優れたパフォーマンスと、実行するたびに完全に同じになる結果、つまり再現性のある結果の提供という課題に長い間直面してきました。」

ここで、再現性問題についておさらいしておきましょう。この問題は、浮動小数点数の表現方法と、コンピューターで処理される順序によって、丸め誤差が発生することが原因です。IEEE 単精度または倍精度型[2] で表現される一般的な浮動小数点数には、数学的な結合プロパティーがない[3] ことは周知の事実です。例えば、(a + b) + c は a + (b + c) と等しくない場合があります。

特定の例について考えてみましょう。紙に鉛筆で書いた場合、$2^{-63} + 1 + -1 = 2^{-63}$ です。ところが、倍精度浮動小数点数を使用してコンピューターでこの同じ計算を行った場合、$(2^{-63} + 1)$ が 1 に丸められて $(2^{-63} + 1) + (-1) \approx 1 + (-1) = 0$ になります。演算の順序を多少変更して、$2^{-63} + (1 + (-1)) \approx 2^{-63} + 0 = 2^{-63}$ にすることはできます。0 と 2^{-63} が等しくないのは明白であるため、演算の順序は丸め誤差がいつどのように発生するかだけでなく、最終的な計算結果にも影響します。コンパイラーは通常、この順序の曖昧さを再連想と呼びます。

アプリケーション・レベルで並列化を行うと、再現性のない結果が生成される可能性がさらに高くなります。原因は、計算の引数が指定される順序によってキャリーオーバーが発生するためです。作業が複数のスレッドまたはプロセスに分配される場合は常に、計算が依存しているチェーン内で計算の順序を変更すると、中間結果だけでなく、最終的な計算結果が異なる場合があります。配列の要素が複数のスレッドに分配されるときに、配列の合計や積をそのまま計算するリダクション演算が良い例です。一部の合計や積を計算した後、複数のスレッドの値を組み合わせて 1 つの値にします。配列の分配方法、あるいはスレッド固有の合計や積をほかの値と組み合わせる順序を変更すると、最終的に導き出される合計や積に影響します。さらに範囲を広げると、一貫した予測可能な方法で並列処理を制御する方法は、決定性の並列処理[4] の分野に含まれます。

一般的なアプリケーションでは膨大な浮動小数点演算が行われると考えた場合、計算の順序が最終的な計算結果にどのように影響を与えるかは明らかです。

インテル® MKL

インテル® MKL 11.0 では、特定の条件の下で再現性のある浮動小数点演算結果を得ることができる「条件付き数値再現性」機能が追加されました。[5] この新しい機能を使用すると、インテル® MKL 関数は、実行するたびに同じ浮動小数点演算結果を返します。ただし、次のような制限があります。

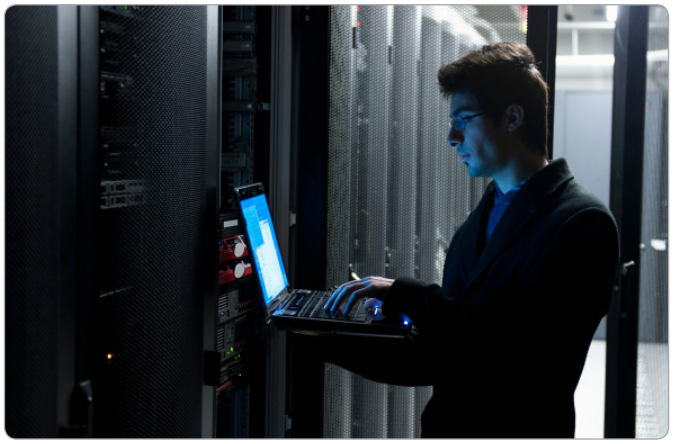
- 関数呼び出しの入出力配列は、SSE/AVX1/AVX2 命令対応のシステムではそれぞれ、16/32/64 ビット境界でアライメントしなければなりません。
- 一貫した結果を得るには、すべての実行でスレッド数の制御が同じでなければなりません。

浮動小数点演算が行われる順序に影響する 1 つの実行プログラム内のアプリケーション関連の要因には、ランタイム・プロセッサー・ディスパッチャーに基づくコードパスの選択、データ配列のアライメント、スレッド数の変化、スレッド化アルゴリズム、内部の浮動小数点制御設定などがあります。これまで、ユーザーがライブラリーのランタイム・ディスパッチや、関数が内部的にスレッド化されるかどうかを制御することはできませんでした。ただし、スレッド数の管理、浮動小数点設定の確認、メモリーを割り当てるときにメモリーのアライメントを設定するステップを実行することは可能でした。

インテル® MKL は、アプリケーションに呼び出されたインテル® MKL 関数で走査する適切な内部コードパスを特定するために、ランタイム・プロセッサー・ディスパッチを行います。選択されるコードパスは、インテル® プロセッサーおよび互換プロセッサーの種類によって異なり、さまざまなレベルのパフォーマンスが提供されます。例えば、インテル® Pentium® 4 プロセッサー上でインテル® MKL 関数を実行すると、SSE2 ベースのコードパスが実行されます。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) 対応の新しいインテル® Xeon® プロセッサー上では、同じライブラリー関数が AVX 命令を使用する異なるコードパスをディスパッチします。この違いは、各コードパスが実行しているプロセッサーで利用可能な機能を使うように最適化されているためです。この機能ベースの最適化アプローチは、その性質により、前述の再現性問題を増幅させます。内部浮動小数点演算が異なる順序で行われたり、関連付けが変更されると、計算結果は異なります。

この記事の続きを読む（英語）

「マイクロプロセッサーの世代が新しくなるにつれ、ソフトウェアのパフォーマンスを向上させるには、新しく追加された命令を使用してプロセッサーの性能を引き出すとともに、スレッド化アルゴリズムを使用して多くの計算コアを活用することが必須となっています。」



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

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

最適化に関する注意事項

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項で対象としている特定の命令セットに関する詳細は、該当製品のユーザーズガイドまたはリファレンス・ガイドを参照してください。

改訂 #20110804