インテル® ADVISOR のルーフライン解析

パフォーマンス最適化のトレードオフを視覚化する新しい方法

Kevin O’Leary インテル コーポレーション テクニカル・コンサルティング・エンジニア、
Ilyas Gazizov 同シニア・ソフトウェア・デベロッパー、Alexandra Shinsel 同テクニカル・コンサルティング・エンジニア、
Zakhar Matveev 同製品アーキテクト、Dmitry Petunin 同テクニカル・コンサルティング・エンジニア

現在、そして将来のハードウェアを最大限に活用するためには、ソフトウェアはスレッド化およびベクトル化されていなければなりません。データに基づく**ベクトル化**設計は、長期間にわたって優れたパフォーマンスを達成可能で、少ないリスクで大きな効果が得られます。ベクトルレベルおよびスレッドレベルで完全に並列化されている場合であっても、開発者が CPU/ ベクトル / スレッドの使用状況とメモリー・サブシステムのボトルネックを調整しなければならないことはよくあります。多くの場合、ルーフラインの「依存とボトルネック」パフォーマンス・モデルを使用することで、この最適化に対応できます。

この記事では、**インテル® Advisor 2017** の概要を提供し、新しいルーフライン解析 (Roofline Analysis) 機能を説明します。ルーフライン・モデルは、アプリケーションのパフォーマンスの問題に対応するための最良の方法を、直感的かつ強力に表現します。ケーススタディーでは、実際の例を用いて最適化プロセスを紹介します。

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

ルーフライン・モデル

ルーフライン・モデルは、カリフォルニア大学バークレー校の研究者 Samuel Williams、Andrew Waterman、David Patterson により、2009 年に**「Roofline: An Insightful Visual Performance Model for Multicore Architectures」**(英語) で提案されました。近年、このモデルは、Aleksandar Ilic、Frederico Pratas、Leonel Sousa の論文**「Cache-Aware Roofline Model: Upgrading the Loft」**(英語) で、メモリー・サブシステムのすべてのレベルに対応するように拡張されました。

ルーフライン・モデルは、開発者が次の質問に答えられるように支援することで、アプリケーションの動作に関する詳細を提供します。

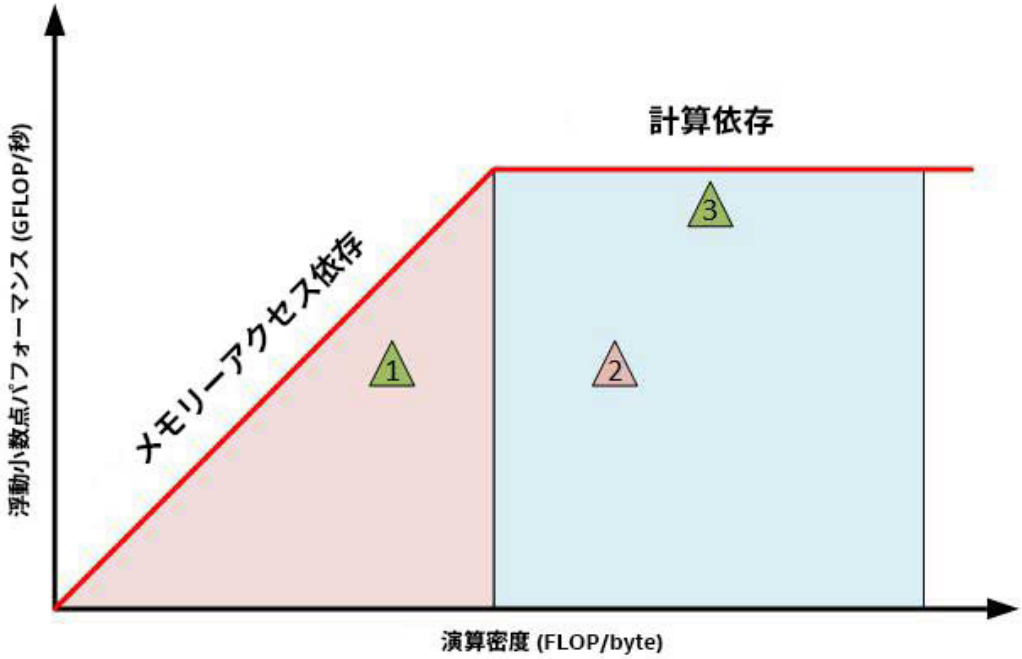
- アプリケーションは現在のハードウェアで最適に動作していますか? そうでない場合、最も活用されていないハードウェア・リソースは何ですか?
- パフォーマンスを制限している要因は何ですか? アプリケーションのワークロードはメモリー依存または計算依存ですか?
- アプリケーションのパフォーマンスを向上する適切な方法は何ですか?

モデルは、次の 2 つのパラメーターを測定し、アプリケーションの計算とメモリー帯域幅の上限を視覚化できるようにデータをグラフ化します。

1. 演算密度 (CPU とメモリー間で転送されたバイトあたりの浮動小数点演算数)
2. 浮動小数点パフォーマンス (GFLOPS)

データポイントがモデルライン (ルーフライン) に近いほど、適切に最適化されていることを示します (**図 1**)。青色の領域にあるカーネルのほうが計算への依存度が高く、Y 軸の上方向にあるカーネルのほうがピーク浮動小数点パフォーマンスに近くなります。これらのカーネルのパフォーマンスは、プラットフォームの計算能力に依存します。カーネル 3 のパフォーマンスを高めるには、**インテル® Xeon Phi™** プロセッサーのような計算能力とメモリー・スループットが高い高度な並列プラットフォームへの移行を検討してください。カーネル 2 は、上限からほど遠く離れているため、ベクトル化によりパフォーマンスを向上できるでしょう。

グラフの赤色の領域にあるカーネルはメモリーに依存します。Y 軸の上方向になるほど、カーネルはプラットフォームの DRAM とキャッシュのピーク帯域幅により制限されます。これらのカーネルのパフォーマンスを向上するには、データ項目ごとの計算量を増やして、パフォーマンスの上限が高いグラフの右側に移動するように、アルゴリズムまたはその実装を再検討してください。これらのカーネルも、インテル® Xeon Phi™ プロセッサーの高いメモリー帯域幅により、高速に実行できる可能性があります。

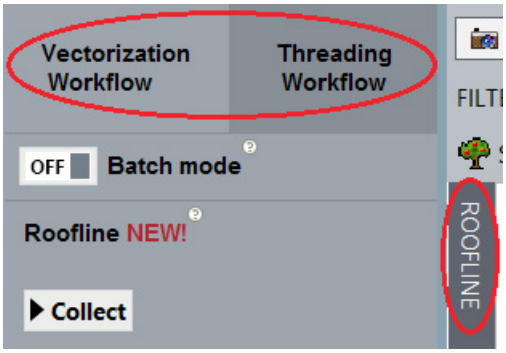


1 ルーフライン・グラフ

インテル® Advisor の概要

インテル® Advisor は、強力なソフトウェア設計とアプリケーション・パフォーマンスの特徴付けを支援するソフトウェア解析ツールです。スレッド並列処理のプロトタイプ生成 (スレッド化アドバイザー)、ベクトル並列処理の最適化 (ベクトル化アドバイザー)、メモリーと計算の特徴付け (ルーフライン・オートメーション) 機能を提供します。

この記事では、主にインテル® Advisor のルーフライン解析とベクトル化解析を取り上げます。インテル® Advisor の GUI では、ワークフロー・セレクターを使用してベクトル化アドバイザーとスレッド化アドバイザーのワークフローを切り替えることができます。ルーフライン・グラフは、[ROOFLINE] サイドバーからアクセスできます (**図 2**)。



2 インテル® Advisor のワークフロー・セレクター

ベクトル化アドバイザーは、次のステップを使用してコードのパフォーマンスを向上できるように支援します。

- **サーベイ (Survey)** は、最も時間を費やしているループと詳細な SIMD 統計を表示します。
- **FLOPS** と**トリップカウント (Trip Counts)** は、各ループと関数の反復回数、呼び出し回数、および正確な FLOPS を測定します。
- **推奨事項 (Recommendations)** は、パフォーマンスの問題を解決するための具体的なアドバイスを示します。
- **依存性解析 (Dependencies Analysis)** は、ループにベクトル化や並列化の妨げとなる反復間の依存性がないか、動的な依存性解析を行います。
- **メモリー・アクセス・パターン解析 (Memory Access Patterns Analysis)** は、ベクトル化に適した方法でメモリーを参照しているか確認します。

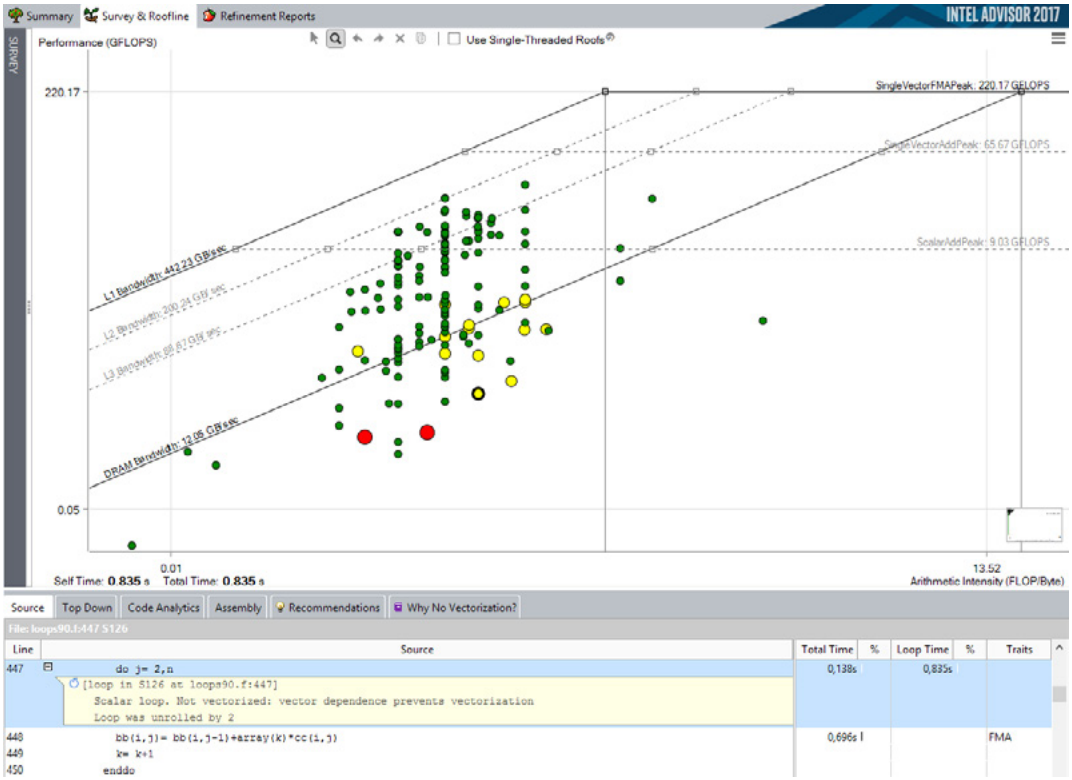
ベクトル化アドバイザーとルーフライン解析を組み合わせることで、パフォーマンスと設計に関する多くの重要な洞察を得ることができます。例えば、ルーフライン・グラフのデータを解釈するには、ベクトル化アドバイザーの [Survey Report (サーベイレポート)] で提供されるベクトル化の効率メトリックを把握しておく必要があります。

インテル® Advisor のルーフライン解析

インテル® Advisor は、「キャッシュを意識した」ルーフライン・モデルを実装し、メモリー / キャッシュ階層のすべてのレベルにわたる詳細を提供します。

- ルーフラインの傾斜部分は、すべてのデータが該当キャッシュに収まる場合のピーク・パフォーマンスを示します。
- 水平線は、ベクトル化やその他の CPU リソースが効率良く利用されている場合に達成可能なピーク・パフォーマンスを示します。

インテル® Advisor は、各ループに対応する点をルーフライン・グラフ上に描画します (**図 3**)。点の大きさと色は、ループの相対的な実行時間を示します。ほとんどのループは、キャッシュメモリーを効率良く利用できるように、さらなる最適化が必要です。いくつかのループ (例えば、グラフ中央の垂直線の右側で、水平の破線 ScalarAddPeak 上にある緑色の点) は、効率良くベクトル化されていません。このように、ルーフライン・グラフを利用することで、アプリケーション・パフォーマンス向上の可能性を簡単に見つけることができます。



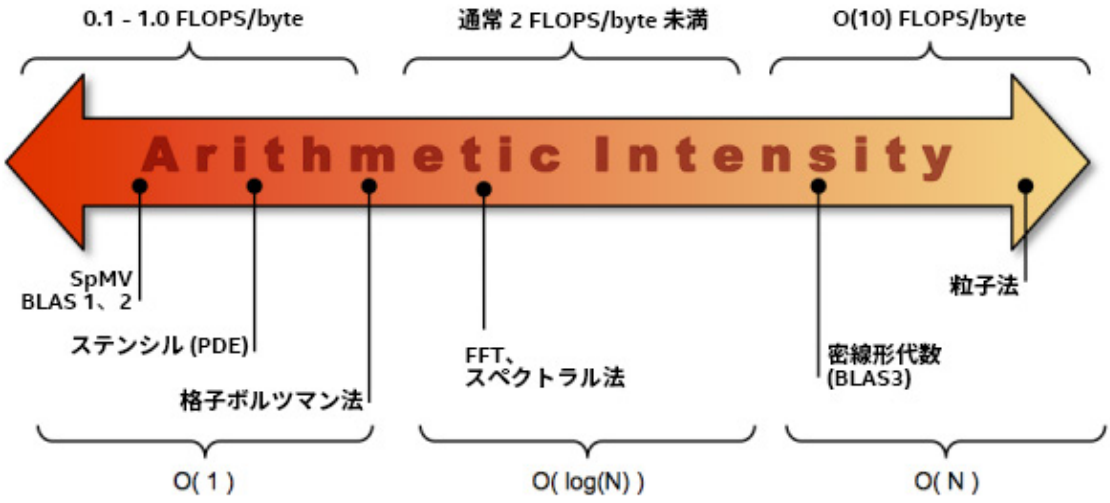
3 インテル® Advisor のルーフライン・グラフと [Source (ソース)] タブ

インテル® Advisor のルーフライン・グラフの解釈方法

ルーフライン・グラフは、役立つ情報を提供しますが、入力と対応する出力を確認するためのリファレンス・テーブルではありません。調査すべき要因を提案するガイドに過ぎないため、その解釈が必要になります。

図 3 に示すようなルーフライン・グラフ上の線は、ホストシステムにおけるベースラインとパフォーマンスの限界を特定するため、インテル® Advisor でベンチマークを実行して得たカーネル・パフォーマンスのハードウェアによる限界を表します。最上部の線 (ルーフ) は、マシンの最大パフォーマンスを示します。**図 3** では、「L1 Bandwidth (L1 帯域幅)」と「Single Vector FMA Peak (単一ベクトル FMA ピーク)」です。すべてのカーネルがこのパフォーマンスを達成できるとは限りません。アルゴリズムの性質によっては、最終的により下のルーフで制限される可能性があります (例えば、ベクトル化できないカーネルは、スカラー計算の最大パフォーマンスに制限されます)。

グラフ上のカーネルの水平位置は、演算密度 (オペランドのサイズで測定された CPU とメモリー間で転送されたバイトあたりの浮動小数点演算数) を示します。これは、主にアルゴリズムによって決まりますが、コンパイル時の最適化によって変わることもあります。**図 4** は、さまざまなアルゴリズムとその演算密度の例です。カーネルのアルゴリズムを再設計して演算密度を上げると、ルーフライン・グラフ上で右側に移動できます。傾斜部分 (メモリー帯域幅のルーフ) に位置している場合は、この移動により最大パフォーマンスが向上する可能性があります。



4 演算密度

カーネルの垂直位置は、さまざまなボトルネックを示します。カーネルがルーフよりも上に位置している場合、そのルーフはパフォーマンスに影響する可能性がありますが、重要なパフォーマンス・ボトルネックではありません。カーネルがルーフよりも下に位置している場合、ボトルネックの可能性があります。ボトルネックの原因は、特定の最適化により解決できます。カーネルがスカラー計算ピークよりも下に位置している場合は、カーネルのベクトル化状況を調査すると良いでしょう。全くあるいは効率良くベクトル化されていない場合、スカラー計算ピークがボトルネックである可能性が高く、慎重にカーネルのベクトル化を改善または実装することを推奨します。効率良くベクトル化されている場合は無視し、カーネルの上にあるほかのルーフの調査に移ります。

インテル® Advisor を使用してパフォーマンスの問題を解決する

このセクションでは、インテル® Advisor に関連したヒントを提供します。

ヒント 1: [Summary (サマリー)] ビューで最も時間のかかるループ (**図 5** と**図 6**) とチューニングの推奨事項を確認します (**図 7**)。

Top time-consuming loops

Loop	Source Location	Self Time	Total Time	Trip Counts
flessparseapngtc	lesSparse.f:387	5.9143s	5.9143s	6
flessparseapg	lesSparse.f:232	4.9664s	4.9664s	1; 2; 4; 1; 1
flessparseapkg	lesSparse.f:286	4.3634s	4.3634s	2; 1; 4; 1
fmtxblkdot2	ftools.f:445	1.0166s	1.0166s	9162
fmtxvdimvecmult	ftools.f:45	0.8999s	0.8999s	572

5 最も時間のかかるループのサマリー

6 最も時間のかかるループのサマリー (1 番上のループはベクトル化されている)

Top time-consuming loops

Loop	Self Time	Total Time	Trip Counts
[loop in flessparseapngtc at lesSparse.f:389]	5.879s	5.879s	1; 2
[loop in flessparseapg at lesSparse.f:232]	5.247s	5.247s	1; 2; 4; 1; 1
[loop in flessparseapkg at lesSparse.f:287]	4.247s	4.247s	2; 1; 4; 1
[loop in fmtxblkdot2 at ftools.f:445]	1.158s	1.158s	9162; 3; 1
[loop in fmtxblkdmaxpy at ftools.f:874]	1.148s	1.148s	9162; 3; 1

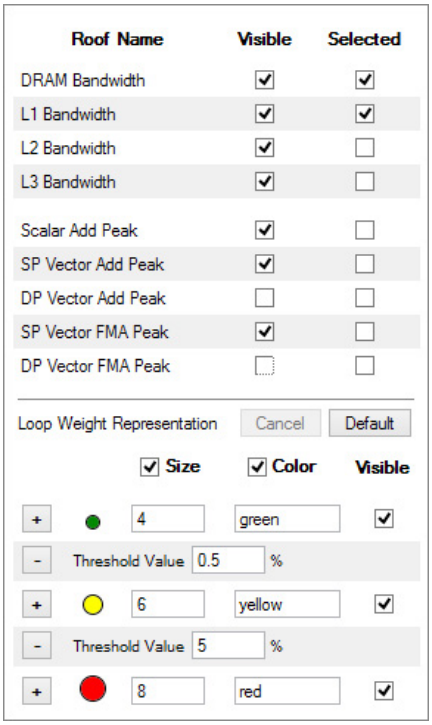
7 最も時間のかかるループに対する推奨事項のサマリー

Recommendations

Loop	Self Time	Recommendations
[loop in flessparseapngtc at lesSparse.f:389]	5.879s	Enforce vectorized remainder
[loop in flessparseapg at lesSparse.f:232]	5.247s	Add data padding
[loop in flessparseapkg at lesSparse.f:287]	4.247s	Enforce vectorized remainder Add data padding
[loop in flessparseapsclr at lesSparse.f:514]	0.261s	Disable unrolling Enforce vectorized remainder Align data Add data padding
[loop in flessparseapfull at lesSparse.f:448]	0.190s	Enforce vectorized remainder Add data padding

ヒント 2: ルーフラインのカスタマイズで不要なルーフを削除します (**図 8**)。

例えば、単精度データのみを扱う場合、ルーフラインから倍精度ピークを削除することができます。

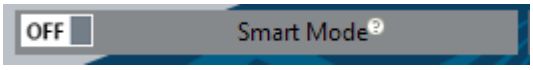


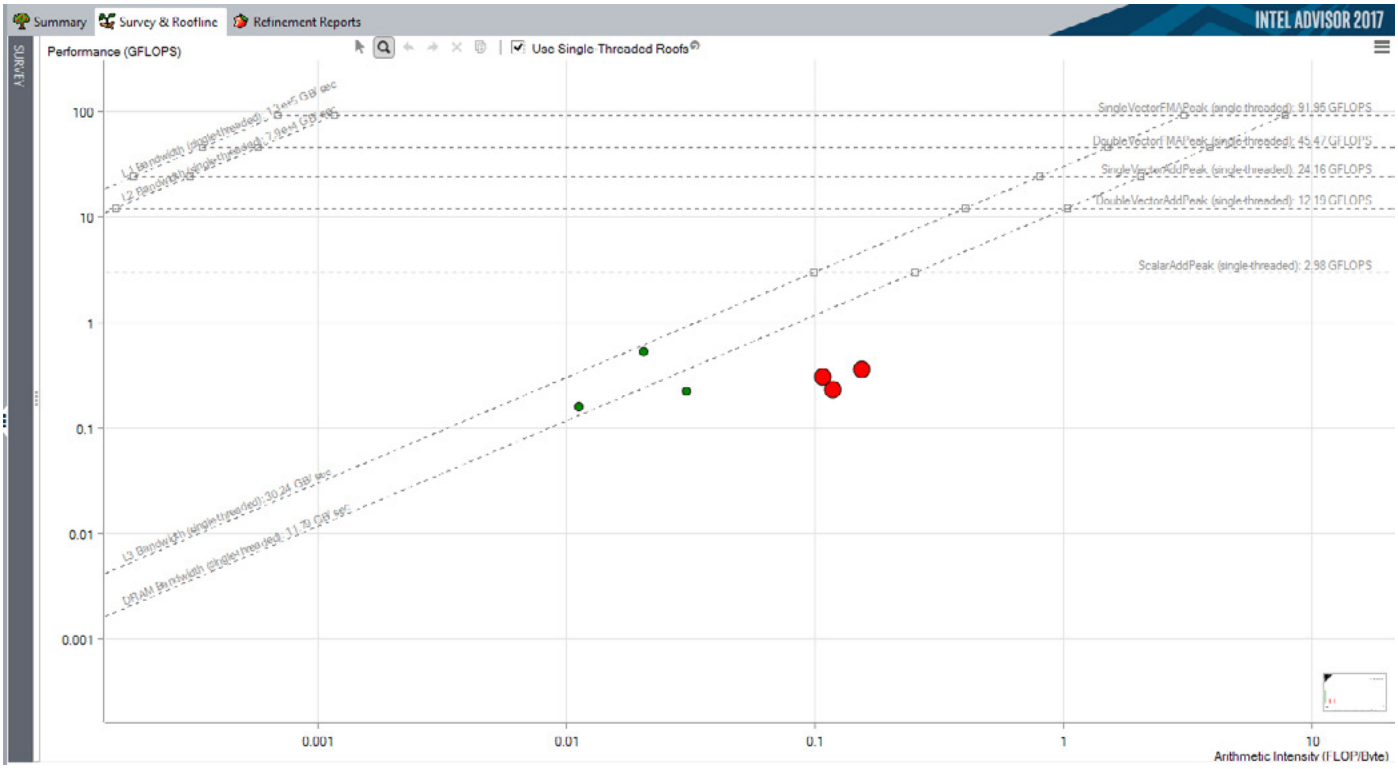
ヒント 3: スマートモード (Smart Mode) を使用して最適化により最も大きな効果が得られる候補を見つけます (**図 9**)。

ルーフライン上では、ループは経過セルフ時間順に表示されますが、スマートモードをオンにすると合計時間が多いループを見つけることができます。合計時間が多いループのほうが、最適化により得られる効果が大きくなります。

8 インテル® Advisor のルーフライン・グラフのカスタマイズ

9a インテル® Advisor の [Smart Mode (スマートモード)] セレクター





9b スマートモードでフィルターされたインテル® Advisor のルーフライン・グラフ

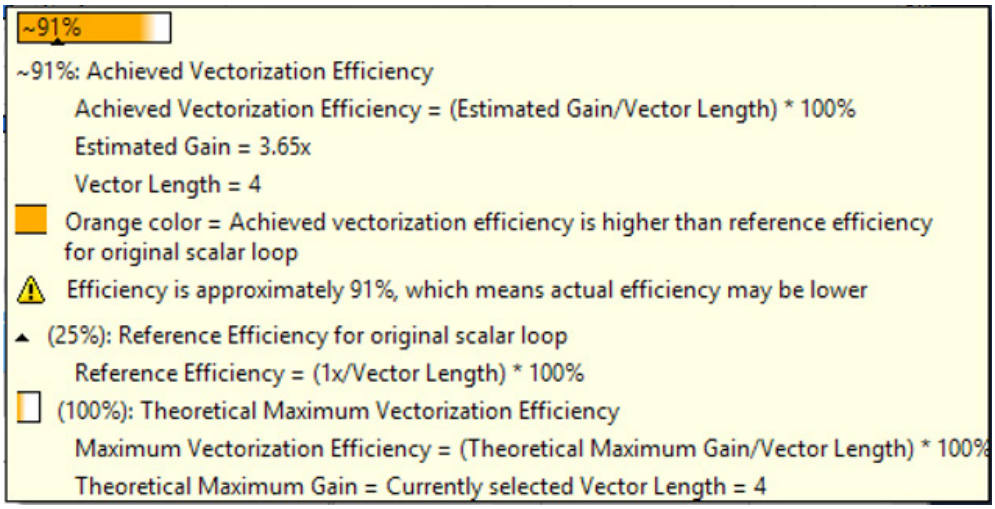
ヒント 4: インテル® Advisor のほかの機能を使用して、ルーフライン・グラフの情報を補完します。

ベクトル化の効率は、ベクトル化の温度計と言えます (**図 10** と**図 11**)。[Instruction Set Analysis (命令セット解析)] の [Traits (特徴)] 列で、ベクトル化に影響している可能性がある要因を確認できます (**図 10**)。ベクトル化に適していない方法でメモリーを参照している可能性がある場合は、インテル® Advisor のメモリー・アクセス・パターン収集の実行を検討してください。[編集者注 : The Parallel Universe 本号の Vladimir Tsymbal の記事**「並列アプリケーションのスケーラビリティーの問題を特定する」**では、いくつかのメモリーアクセスの解析手法を説明しています。]

Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Trip Counts	Instruction Set Analysis		
					Vect...	Efficiency	Gain...	VL (...		Traits	Data T...	Num.
1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4	25; 2	Inserts	Float64	3; 7
1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4	25; 2		Float64	3

10 ベクトル化の効率がハイライト表示されたインテル® Advisor のサーベイ

11 インテル® Advisor のベクトル化の効率に関する説明



ヒント 5: ループセレクターを使用して、ベクトル化されたループとされなかったループを切り替えることができます (**図 12**)。

12 インテル® Advisor のベクトル化された / されていないループのセレクター

ヒント 6: [Source (ソース)] タブとルーフライン・グラフを併用します (**図 13**)。

インテル® Advisor は、ソースコードをパフォーマンス・プロファイルにシームレスに統合します。

BLOG HIGHLIGHTS

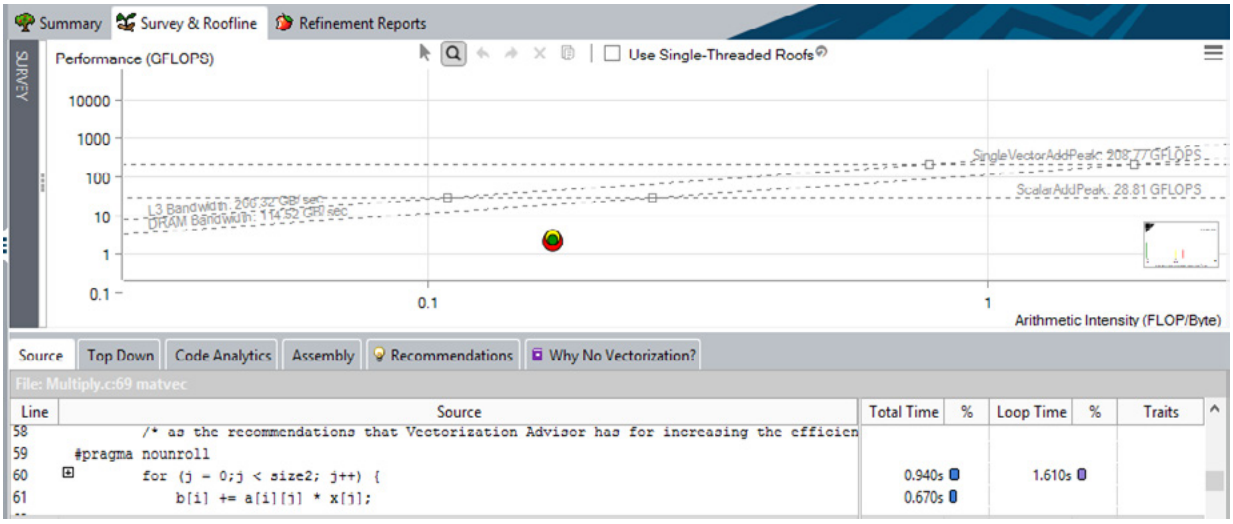
インテル® Xeon Phi™ プロセッサー x200 製品ファミリー (開発コード名 Knights Landing) のユーザーモード (リング 3) の MONITOR と MWAIT

JAMES C. (INTEL) >
BENJAMIN C. (INTEL) >

インテル® Xeon Phi™ プロセッサー x200 製品ファミリー (開発コード名 Knights Landing) には、通常アーキテクチャーでリング 0 (カーネルコード) に制限されている MONITOR および MWAIT[1] 命令を、リング 0 以外のリングで実行可能なモデル固有の機能があります。具体的には、通常のユーザーモードである、リング 3 でこれらの命令の実行を許可します。

この機能を有効にするには、以下に示すように、MSR 140H (MISC_FEATURE_ENABLES モデル固有レジスター) の ビット 1 を設定します。このレジスターを読み取ることで、リング 0 以外で命令が有効になっているかどうか判断することもできます。

この記事の続きはこちら (英語) でご覧になれます。 >



13 インテル® Advisor のルーフライン・グラフと [Source (ソース)] タブ

ケーススタディー : ルーフライン解析を使用した MRI 画像再構成ベンチマークのチューニング

514.pomriq SPEC ACCEL* ベンチマークは、Stone ほか (2008) で説明されている MRI 画像再構成カーネルです。MRI 画像再構成は、収集した信号を傾斜磁場に変換します。収集領域は、傾斜磁場の空間 (k 空間) に位置します。MRI 画像再構成の Q 行列は、信号収集軌道 (K 空間における信号の収集方法) に基づいて事前計算可能な値です。アルゴリズムは、対象の MRI スキャン軌道と収集する信号を表す大きな入力セットを処理します。Q 行列の各要素は、すべての軌道の収集信号の加重によって計算されます。各加重は、入力の 3 要素のベクトルドット積、出力の 3D 位置、いくつかの三角関数により計算されます。出力の Q 要素は複素数ですが、入力は多要素ベクトルです。三角関数は重たい処理であるため、カーネルは根本的に計算依存です。問題の正則性により、メモリー帯域幅は容易に管理できます。そのため、タイリングとデータレイアウトにより人為的なメモリー帯域幅のボトルネックを排除したら、低レベルのシーケンシャル・コードを最適化し、ループアンロールなどにより命令ストリームの効率を改善することが重要です。

514.pomriq への入力は、k 空間の値の数、X 空間の値の数、そして k 空間のサンプルの k 空間座標、X 空間座標、Phi フィールドの複素数値のリストを含む 1 つのファイルです。座標と複素数値の各セットは配列に格納され、各フィールドは隣接して記述されます。514.pomriq の出力は、各行に「実部 , 虚部」形式で複素数値を格納した結果の Q 行列です。このケーススタディーでは、514.pomriq 計算カーネルを解析し、最適化します。**図 14** は、インテル® Xeon Phi™ プロセッサー 7250 上で 514.pomriq を実行して得られたインテル® Advisor の [Summary (サマリー)] ビューです。Q 行列の計算に関連したループが hotspot であることが分かります。

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics

Elapsed Time: 36.93s

Vector Instruction Set: AVX512 | Number of CPU Threads: 136

Total GFLOP Count: 19293.90 | Total GFLOPS: 522.51

Loop metrics

Total CPU time	4267.88s	100.0%
Time in 1 vectorized loop	4206.25s	98.6%
Time in scalar code	61.62s	

Vectorization Gain/Efficiency (Not available)

Top time-consuming loops

Loop	Self Time	Total Time	Trip Counts
[loop in ComputeQCPU at computeQ.c:65]	1957.548s	4206.254s	12500
[loop in ComputeQCPU at computeQ.c:58]	6.963s	4213.216s	15420
[loop in outputData at file.c:70]	0.040s	4.160s	2097152
[loop in start_thread at ?]	0s	49.660s	
[loop in [OpenMP worker at z_Linux_util.c:769]	0s	49.660s	

Refinement analysis data

These loops were analyzed for memory access patterns and dependencies:

Site Location	Dependencies	Strides Distribution
[loop in ComputeQCPU at computeQ.c:66]	No information available	96% / 0% / 4%

Collection details

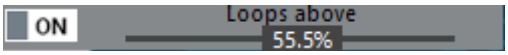
Platform information

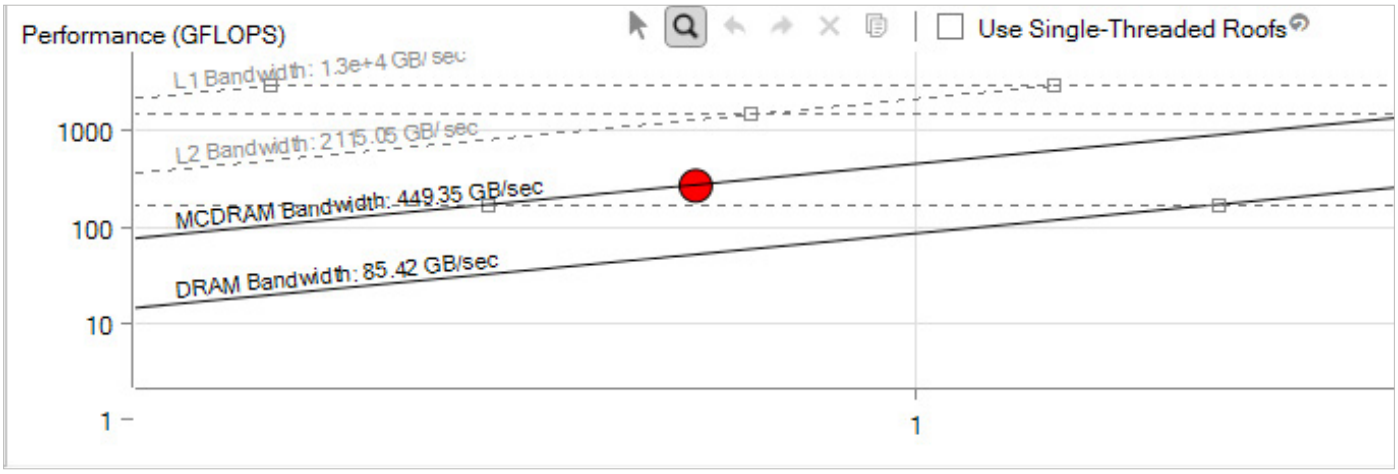
CPU Name:	Intel(R) Xeon Phi(TM) CPU 7250 @000000 1.40GHz
Frequency:	1.40 GHz
Logical CPU Count:	272
Operating System:	Linux

14 インテル® Advisor の [Summary (サマリー)] ビュー

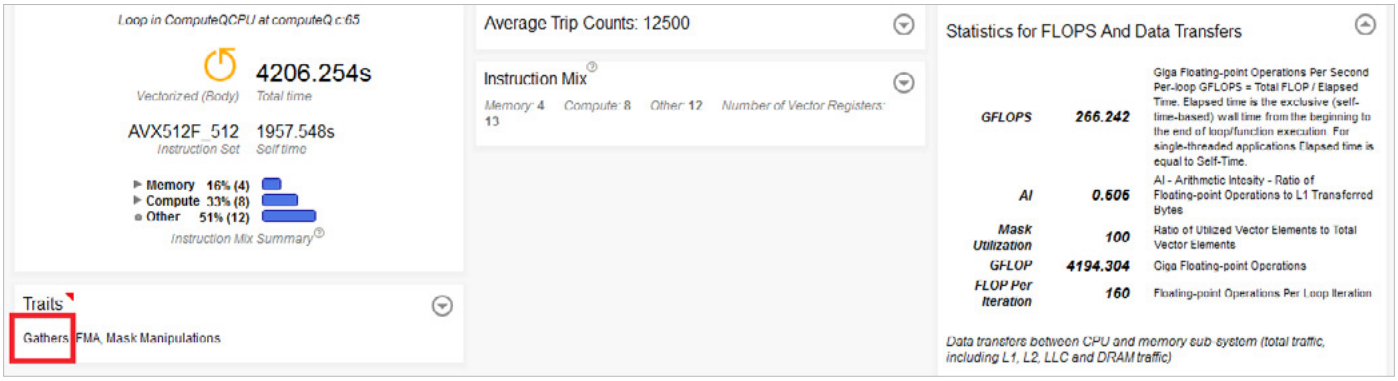
インテル® Advisor のスマートモードを使用して、最適化により最も大きな効果が得られる候補を絞り込みます (**図 15a**)。最も時間のかかるループはベクトル化されていますが、MCDRAM ルーフよりも下にあります (**図 15b**)。これは、メモリーの使用状況に問題がある可能性を示しています。[Code Analytics (コード解析)] タブと [Recommendations (推奨事項)] タブを使用してループを調査します。

15a インテル® Advisor の [Smart Mode (スマートモード)] セレクター





15b スマートモードでフィルターされたインテル® Advisor のルーフライン・グラフ



16 ギャザーがハイライト表示されたインテル® Advisor の [Code Analytics (コード解析)] タブ

負荷の高いギャザー命令があることが分かります (**図 16**)。また、ループのメモリー・アクセス・パターンの調査を勧めるアドバイスが表示されています (**図 17**)。メモリー・アクセス・パターン解析を実行したところ、ギャザーストライドはループあたり 4% であることが分かりました。この情報は、最適でないメモリーアクセスの場所を特定するヒントになります (**図 18**)。詳細を確認すると、ストライドは定数なのでギャザー命令を使用する必要がないことが分かりました (**図 19a**)。インテル® Advisor の [Recommendations (推奨事項)] タブでも同じことが確認できます (**図 19b**)。

17 インテル® Advisor のメモリー・アクセス・パターンに関する推奨事項

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint	Site Name	Recon
[loop in ComputeQCPU at computeQ.c:66]	No information available	96% / 0% / 4%	Mixed strides	3MB	loop_site_17	

Memory Access Patterns Report | Dependencies Report | Recommendations

ID		Stride	Type	Source	Nested Function	Variable references	Access Footprint
P1			Gather stride	computeQ.c:66		block 0x7f0045867010 allocated at main.c:99	3MB
P2			Parallel site information	computeQ.c:66			
P4		0	Uniform stride	omriq_exe_nog2s:0x29a8	__svml_sincosf16	__svml_sincosf16_chosen_core_func	8B
P5		0	Uniform stride	omriq_exe_nog2s:0x29cf	__svml_sincosf16_b3		64B
P6		0	Uniform stride	omriq_exe_nog2s:0x29de	__svml_sincosf16_b3		64B
P7		0	Uniform stride	omriq_exe_nog2s:0x29f3	__svml_sincosf16_b3	__svml_ssincos_data	64B
P8		0	Uniform stride	omriq_exe_nog2s:0x29fa	__svml_sincosf16_b3	__svml_ssincos_data	64B
P9		0	Uniform stride	omriq_exe_nog2s:0x2a07	__svml_sincosf16_b3	__svml_ssincos_data	64B
P10		0	Uniform stride	omriq_exe_nog2s:0x2a14	__svml_sincosf16_b3	__svml_ssincos_data	64B
P11		0	Uniform stride	omriq_exe_nog2s:0x2a1b	__svml_sincosf16_b3	__svml_ssincos_data	64B

18 インテル® Advisor のメモリー・アクセス・パターン・レポート

19a インテル® Advisor のメモリー・アクセス・パターン・レポートの詳細

Details View

Gather (irregular) access

Operand Size (bits): 32
Operand Type: bit*16;float32*16
Vector Length: 16
Memory access footprint: 3MB

Gather/scatter details

Pattern: "Constant (non-unit)"

Instruction accesses values with constant offset from the base:
- stride within instruction = X
- stride between iterations = X*vector length

Horizontal stride (bytes): 16
Vertical stride (bytes): 256

Mask is constant
Mask: [1111111111111111]
Active elements in the mask: 100.0%

Variable references

Names: block 0x7f0045867010 allocated at main.c:99

Issue: Inefficient gather/scatter instructions present

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by alerting the compiler to detected regular stride access patterns, such as:

Pattern	Description
Invariant	The instruction accesses values in the same memory throughout the loop.
Uniform (Horizontal Invariant)	The instruction accesses values in the same memory within the vector iteration.
Vertical Invariant	The instruction accesses the memory locations using the same offset across all vector iterations.
Unit	The instruction accesses values in contiguous memory throughout the loop, and the stride between vector iterations = vector length.

Recommendation: Refactor code with detected regular stride access patterns — Confidence: Low

The Memory Access Patterns Report shows the following regular stride access(es):

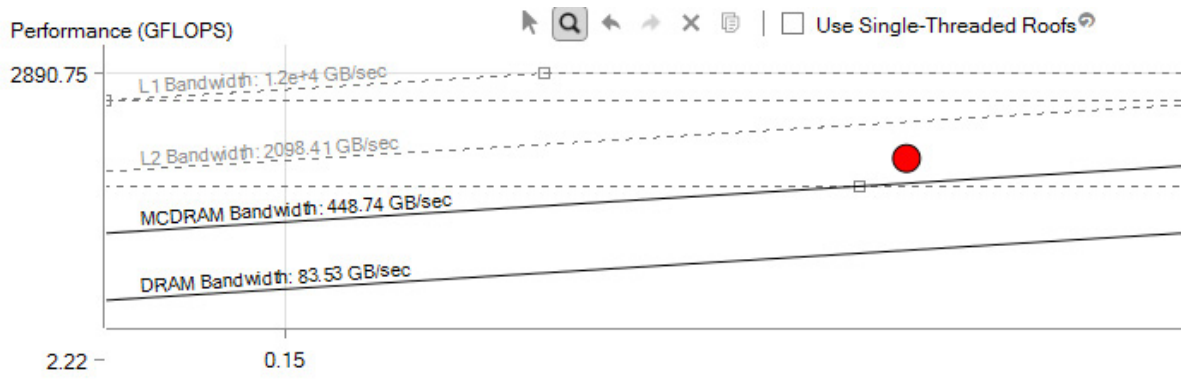
19b インテル® Advisor のギャザー / スキャッターに関する推奨事項

```
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
      for (indexK = 0; indexK < numK; indexK++) {
        expArg = PIx2 * (kVals[indexK].Kx * x[indexX] +
        kVals[indexK].Ky * y[indexX] +
        kVals[indexK].Kz * z[indexX]);

        cosArg = cosf(expArg);
        sinArg = sinf(expArg);

        float phi = kVals[indexK].PhiMag;
        QrSum += phi * cosArg;
        QiSum += phi * sinArg;
      }
```

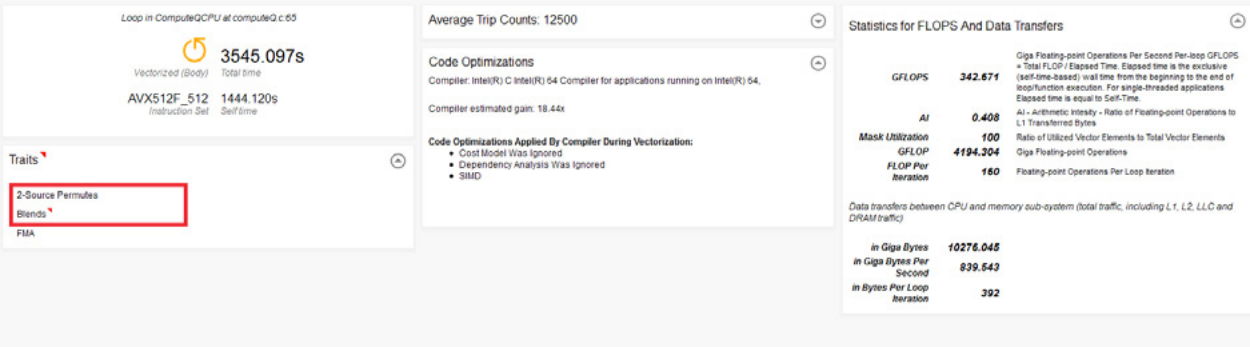
20 速度を制限しているカーネルのソースコード

カーネルのソースコードを確認したところ、原因が分かりました (**図 20**)。このコードは、ベクトル化すると「ギャザー」される構造体配列を使用しています。新しいバージョンのインテル® コンパイラーは、このアクセスパターンを認識し、より軽量な命令を使用できるようにギャザーを排除する最適化を適用します。「ギャザー」の置換は、「ギャザーシャッフル / 並べ替え」コンパイラー変換により実行され、最近の CPU (特にインテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) をサポートするプラットフォーム) では、多くの場合、これにより利点が得られます。[編集者注 : The Parallel Universe 本号の Martyn Corden の記事**「インテル® AVX-512 で向上したベクトル化のパフォーマンス」**では、インテル® コンパイラー 2017 でインテル® AVX-512 を利用して新たなループベクトル化を可能にする方法について説明しています。]

「ギャザーシャッフル / 並べ替え」をサポートする新しいインテル® コンパイラー (インテル® Parallel Studio XE 2017 Update 1 に含まれるインテル® コンパイラー 17.0 Update 1 など) で再コンパイルして、ルーフラインを見てみましょう。カーネルを示す点が MCDRAM の上に移動し、ギャザー命令がなくなり (インテル® AVX-512 の「2 ソースの並べ替え」に置換され)、FLOPS が向上したことが分かります (**図 21** と **図 22**)。



21 インテル® Advisor のルーフライン・グラフ : MCDRAM 帯域幅の上にループが移動



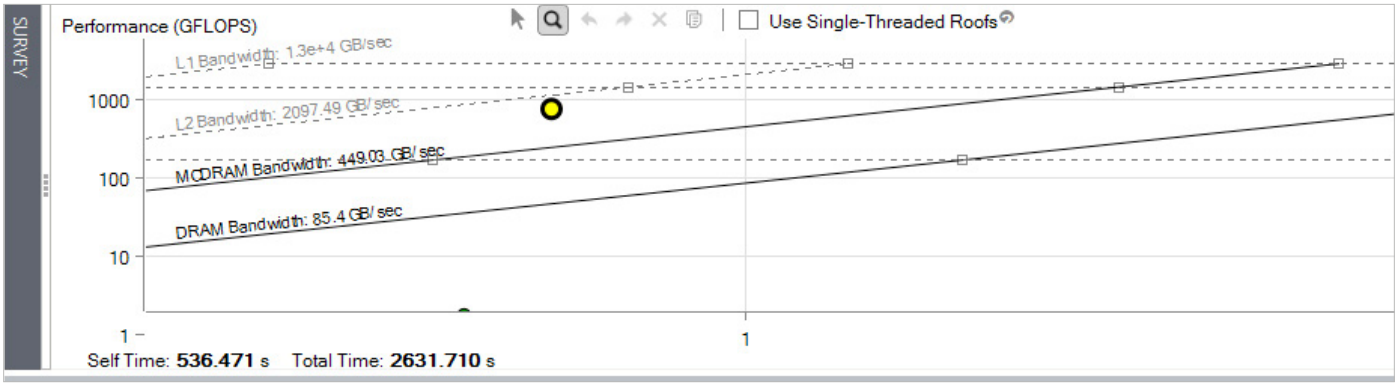
22 インテル® Advisor の [Code Analytics (コード解析)] タブ

しかし、AOS (構造体配列) から SOA (配列構造体) への変換を利用することで、この問題を効率良く解決できます。この最適化は、より扱いやすいデータコンテナーを使用することで、ベクトル処理の効率を向上します。これまでは、この変換を行う場合、データ構造を手動で変更する必要がありました。しかし、SIMD Data Layout Templates (SDLT) (**図 23**) を使用することで、**kValues** 構造体の宣言、初期化、k 値の計算位置に数行のコードを追加するだけで、パフォーマンスを向上できるようになりました。

```
#include <sdlt/sdlt.h>
struct kValues {
  float Kx;
  float Ky;
  float Kz;
  float PhiMag;
};

SDLT_PRIMITIVE(kValues, Kx, Ky, Kz, PhiMag)


sdlt::soa1d_container<kValues> inputKValues(numK);
auto kValues = inputKValues.access();

  for (k = 0; k < numK; k++) {
    kValues [k].Kx() = kx[k];
    kValues [k].Ky() = ky[k];
    kValues [k].Kz() = kz[k];
    kValues [k].PhiMag() = phiMag[k];
  }


auto kVals = inputKValues.const_access();
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
      for (indexK = 0; indexK < numK; indexK++) {
        expArg = PIx2 * (kVals[indexK].Kx() * x[indexX] +
        kVals[indexK].Ky() * y[indexX] +
        kVals[indexK].Kz() * z[indexX]);

        cosArg = cosf(expArg);
        sinArg = sinf(expArg);

        float phi = kVals[indexK].PhiMag();
        QrSum += phi * cosArg;
        QiSum += phi * sinArg;
      }
```

23 SIMD Data Layout Templates ライブラリーの使用

新しいルーフライン・グラフを確認してみましょう (**図 24**)。この最適化を適用後、カーネルを示す点が赤色ではなくなりました。これは、実行時間が短くなり、GFLOPS が向上し、L2 ルーフに近付いたことを示しています。さらに、ループでユニット・ストライド・アクセスを利用することで、特別なメモリー操作がなくなりました (**図 25a** と**図 25b**)。最終的には、カーネルでは 3 倍、アプリケーション全体では 1.5 倍のパフォーマンス向上を達成できました。



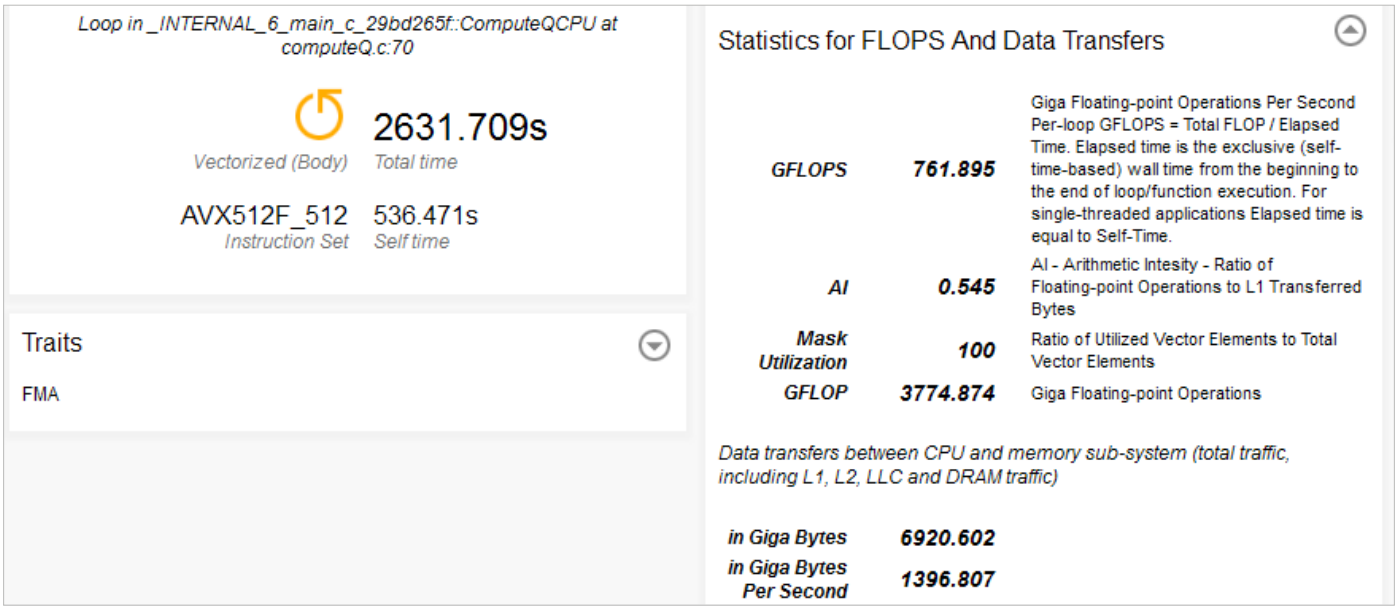
24 最終的な最適化後のインテル® Advisor のルーフライン・グラフ

Site Location	Loop Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint	Site Name	Recommendations
[loop in _ZN4sd ...	No information available	100% / 0% / 0%	All unit strides	769KB	loop_site_27	

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Nested Function	Variable references	Access Footprint	Modules	Site Name
P1		Parallel site information	rvalue_binary_operator_proxy_other_xmacro.h:50				omriq_exe_soa	loop_site_27
P3	0	Uniform stride	omriq_exe_soa:0x2d28	__svml_sincosf16	__svml_sincosf16_chosen_core_func	8B	omriq_exe_soa	loop_site_27
P4	0	Uniform stride	omriq_exe_soa:0x2d4f	__svml_sincosf16_b3		64B	omriq_exe_soa	loop_site_27
P5	0	Uniform stride	omriq_exe_soa:0x2d5e	__svml_sincosf16_b3		64B	omriq_exe_soa	loop_site_27
P6	0	Uniform stride	omriq_exe_soa:0x2d73	__svml_sincosf16_b3	__svml_ssincos_data	64D	omriq_exe_soa	loop_site_27
P7	0	Uniform stride	omriq_exe_soa:0x2d7a	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P8	0	Uniform stride	omriq_exe_soa:0x2d87	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P9	0	Uniform stride	omriq_exe_soa:0x2d94	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P10	0	Uniform stride	omriq_exe_soa:0x2d9b	__svml_sincosf16_b3	__svml_ssincos_data	64D	omriq_exe_soa	loop_site_27
P11	0	Uniform stride	omriq_exe_soa:0x2da2	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P12	0	Uniform stride	omriq_exe_soa:0x2db6	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P13	0	Uniform stride	omriq_exe_soa:0x2dcb	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P14	0	Uniform stride	omriq_exe_soa:0x2dd2	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P15	0	Uniform stride	omriq_exe_soa:0x2ddf	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P16	0	Uniform stride	omriq_exe_soa:0x2deb	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P17	0	Uniform stride	omriq_exe_soa:0x2df4	__svml_sincosf16_b3	__svml_ssincos_data	64D	omriq_exe_soa	loop_site_27
P18	0	Uniform stride	omriq_exe_soa:0x2e13	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P19	0	Uniform stride	omriq_exe_soa:0x2e1a	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P20	0	Uniform stride	omriq_exe_soa:0x2e21	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P21	0	Uniform stride	omriq_exe_soa:0x2e28	__svml_sincosf16_b3	__svml_ssincos_data	64D	omriq_exe_soa	loop_site_27
P22	0	Uniform stride	omriq_exe_soa:0x2e41	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P23	0	Uniform stride	omriq_exe_soa:0x2e48	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P24	0	Uniform stride	omriq_exe_soa:0x2e55	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P25	0	Uniform stride	omriq_exe_soa:0x2e62	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P26	0	Uniform stride	omriq_exe_soa:0x2e7b	__svml_sincosf16_b3	__svml_ssincos_data	64B	omriq_exe_soa	loop_site_27
P27	0	Uniform stride	omriq_exe_soa:0x2e96	__svml_sincosf16_b3		64B	omriq_exe_soa	loop_site_27
P28	0	Uniform stride	omriq_exe_soa:0x2e9e	__svml_sincosf16_b3		64D	omriq_exe_soa	loop_site_27
P29	1	Unit stride	computeQ.c:72	ComputeQCPU		769KB	libiomp5.so; omriq_exe_soa	loop_site_27
P30	1	Unit stride	computeQ.c:73	ComputeQCPU		769KB	libiomp5.so; omriq_exe_soa	loop_site_27
P31	1	Unit stride	computeQ.c:79	ComputeQCPU		769KB	libiomp5.so; omriq_exe_soa	loop_site_27
P32	1	Unit stride	computeQ.c:80	ComputeQCPU		769KB	libiomp5.so; omriq_exe_soa	loop_site_27
P33	1	Unit stride	rvalue_binary_operator_proxy_other_xmacro.h:50			769KB	libiomp5.so; omriq_exe_soa	loop_site_27

25a 最終的な最適化後のインテル® Advisor のメモリー・アクセス・パターン・レポート



25b 最終的な最適化後のインテル® Advisor の [Code Analytics (コード解析)] タブ

まとめ

ルーフライン・モデルは、新しく、視覚的に直感的で、強力な方法でアプリケーションのパフォーマンスを表示します。ルーフライン・グラフ上のアプリケーションの位置に応じて適切な最適化手法を利用することで、パフォーマンスに影響しない最適化に貴重な時間を費やさないで済みます。ルーフライン・モデルは、次の質問に対する答えを提供します。

- パフォーマンスを向上できますか？
- 主なパフォーマンス・ボトルネックの原因はメモリーですか？それとも CPU ですか？
- 特定のボトルネックを最適化したら、どれぐらいのスピードアップが得られますか？
- 別のプラットフォームを使用したら、どれぐらいのスピードアップが得られますか？

システムが大規模になり、複雑さが増す中、これらの質問に対する答えを得ることは重要です。ルーフライン解析は、そのための時間と労力を軽減します。

コードの現代化

- ハードウェアの性能を最大限に引き出すため、ベクトル化とスレッド化によりコードを現代化する必要があります。
- この記事で示したような系統的なアプローチを採用し、**インテル® Parallel Studio XE** の強力なツールを利用することで、現代化の作業を劇的に軽減できます。
- インテル® Parallel Studio XE 2017 Update 1 で利用可能なインテル® Advisor のルーフライン解析が役立ちます。
- 現在開発中の新機能に関する最新情報を受け取りたい方は、**vector_advisor@intel.com** までメールにてご連絡ください。

インテル® Advisor の関連情報 (英語)

インテル® Advisor のルーフライン機能の使い方

セルフ時間ベースの FLOPS 計算と入れ子のループのルーフライン結果の解釈に関する重要な説明

インテル® Advisor でインテル® MPI ライブラリーを使用するアプリケーションを解析する

参考資料 (英語)

Stone, S. S.; J. P. Haldar, S. C. Tsao, W. W. Hwu., Z. Liang, and B. P. Sutton. "Accelerating advanced MRI reconstructions on GPUs." In International Conference on Computing Frontiers, pages 261–272, 2008.

インテル® ADVISOR を評価する

インテル® PARALLEL STUDIO XE に含まれます >