等方性 3 次元有限差分波動方程式における NUMA の影響を理解する

この記事は、インテル® デベロッパー・ゾーンに公開されている「Understanding NUMA for 3D isotropic Finite Difference (3DFD) wave equation code」の日本語参考訳です。

この記事は、最新のインテル® ソフトウェア開発ツールを利用して、アプリケーションの NUMA 関連のパフォーマンスの問題を見つけ、修正するための手法を示します。

目次

- はじめに
- ISO3DFD アプリケーションのコンパイルと実行手順
- NUMA 関連のパフォーマンスの問題を見つける
- リモート・メモリー・アクセスを軽減するためのコード変更
- 変更したコードのメモリーアクセス解析
- 全体的なパフォーマンスの比較
- システム構成
- 参考文献

1. はじめに

NUMA (Non-Uniform Memory Access) は、マルチプロセッサーで使用されているコンピューター・メモリー設計です。メモリーアクセス時間は、メモリー位置からプロセッサーへの距離に依存します。プロセッサーは、ローカルメモリーのほうがリモートメモリー (ほかのプロセッサーのローカルメモリーや複数のプロセッサー間で共有されるメモリー) よりも高速にアクセスできます。

この記事は、インテル® VTune™ Amplifier XE の最新のメモリーアクセス機能を使用して、アプリケーションで NUMA 関連の問題を見つける方法を紹介します。インテル® デベロッパー・ゾーン (IDZ) で公開されている、インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサー上で実行する等方性 3 次元有限差分アプリケーションの開発とパフォーマンス比較に関する記事を拡張したものです。NUMA 環境でアプリケーションが一貫してハイパフォーマンスを達成できるように、ソースコードの変更に関する推奨事項も示します。

NUMA の問題に注目するため、この記事ではインテル® Xeon® プロセッサー向けに最適化されたバージョンについてのみ述べます。コードは、こちらからダウンロードできます。ISO3DFD アプリケーションのソースコードから dev06 バージョンを使用して、主要メトリックを比較し、アプリケーションで NUMA を認識する利点について考えます。

2. ISO3DFD アプリケーションのコンパイルと実行手順

makefile[1] を使用してアプリケーションをコンパイルします。

```
make build version=dev06 simd=avx2
```

ソースコードに同梱の *run_on_xeon.pl* スクリプトを使用してアプリケーションを実行します。

```
./run_on_xeon.pl executable_name n1 n2 n3 nb_iter n1_block \
    n2_block n3_block kmp_affinity nb_threads
```

```
説明:
    -executable_name: 実行ファイルの名前
    -n1: N1 //X 次元
    -n2: N2 //Y 次元
    -n3: N3 //Z 次元
    -nb_iter: 反復回数
    -n1_block: x 次元のキャッシュブロックのサイズ
    -n2_block: y 次元のキャッシュブロックのサイズ
    -n3_block: z 次元のキャッシュブロックのサイズ
    -kmp_affinity: スレッドの分割
    -nb_threads: OpenMP* スレッドの数
```

3. NUMA 関連のパフォーマンスの問題を見つける

最近の NUMA アーキテクチャーは複雑です。メモリーアクセスを調査する前に、NUMA のアプリケーション・パフォーマンスへの影響を確認すると良いでしょう。numactl[2] ユーティリティーを使用して確認できます。レイテンシーが大きいメモリーアクセスを見つけることが重要です。それらのメモリーアクセスを最適化することで、より大きなパフォーマンスの向上が見込めます。

3.1 numactl

NUMA のアプリケーションへの影響を調べる手っ取り早い方法の 1 つは、1 ソケット/NUMA ノードでアプリケーションを実行し、複数の NUMA ノードで実行した場合のパフォーマンスと比較することです。NUMA による影響がない理想的なシナリオでは、ソケット間で適切にスケーリングし、ほかにスケーリングの妨げとなる要因がない限り、パフォーマンスは 1 ソケットの 2 倍 (2 ソケットのシステムの場合) になるはずです。次に示すように、ISO3DFD では、1 ソケットのパフォーマンスのほうがすべてのノードを使用する場合よりも良いため、NUMA によるパフォーマンスへの影響が見受けられます。この方法は、NUMA のアプリケーション・パフォーマンスへの影響は分かりますが、アプリケーションのどの領域が問題かは分かりません。インテル® VTune™ Amplifier XE のメモリーアクセス解析機能を利用することで、NUMA の問題を詳しく調査できます。

- numactl を使用しない場合の 2 ソケットのパフォーマンス (1 ソケットあたり 22 スレッド):

```
./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe 448 2016 1056 10 448 24 96 compact 44

n1=448 n2=2016 n3=1056 nreps=10 num_threads=44 HALF_LENGTH=8
n1_thrd_block=448 n2_thrd_block=24 n3_thrd_block=96
allocating prev, next and vel: total 10914.8 Mbytes
------------------------------
time:           3.25 sec
throughput:  2765.70 MPoints/s
flops:        168.71 GFlops
```

- numactl を使用した場合の 1 ソケットのパフォーマンス:

```
numactl -m 0 -c 0 ./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe 448 2016 1056 10 448 24
96 compact 22

n1=448 n2=2016 n3=1056 nreps=10 num_threads=22 HALF_LENGTH=8
n1_thrd_block=448 n2_thrd_block=24 n3_thrd_block=96
allocating prev, next and vel: total 10914.8 Mbytes
-------------------------------
time:           3.05 sec
throughput:  2948.22 MPoints/s
flops:        179.84 GFlops
```

- 2 プロセスを別々のソケットで実行した場合のパフォーマンス (1 ソケットあたり 22 スレッド):

```
numactl -c 0 -m 0 ./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe \
         448 2016 1056 10 448 24 96 compact 22 & \
numactl -c 1 -m 1 ./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe \
         448 2016 1056 10 448 24 96 compact 22 &

n1=448 n2=2016 n3=1056 nreps=10 num_threads=22 HALF_LENGTH=8
n1_thrd_block=448 n2_thrd_block=24 n3_thrd_block=96
allocating prev, next and vel: total 10914.8 Mbytes
-------------------------------
time:           2.98 sec
throughput:  2996.78 MPoints/s
flops:        180.08 GFlops
n1=448 n2=2016 n3=1056 nreps=10 num_threads=22 HALF_LENGTH=8
n1_thrd_block=448 n2_thrd_block=24 n3_thrd_block=96
allocating prev, next and vel: total 10914.8 Mbytes
-------------------------------
time:           3.02 sec
throughput:  2951.22 MPoints/s
flops:        179.91 GFlops
```

3.2 インテル® VTune™ Amplifier XE – メモリーアクセス解析の使用

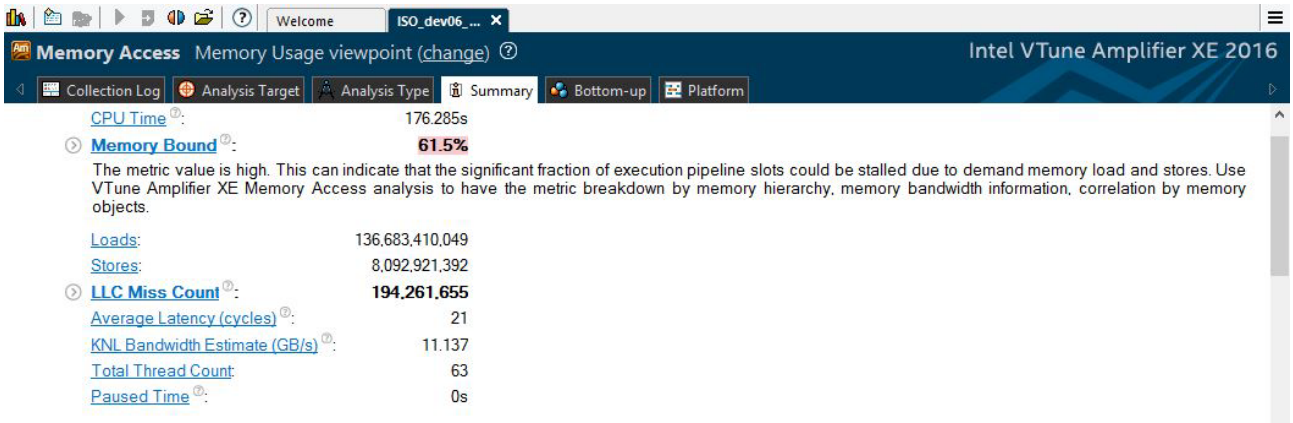
NUMA をサポートするプロセッサーでは、実行中の CPU のキャッシュミスだけではなく、リモート DRAM や別の CPU のキャッシュへの参照も調査することが重要です。これらの詳細を得るため、次のように、アプリケーションのメモリーアクセス解析を行います。

```
amplxe-cl -c memory-access –knob analyze-mem-objects=true \
     -knob mem-object-size-min-thres=1024  -data-limit=0 \
     -r ISO_dev06_MA_10 ./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe \
     448 2016 1056 10 448 24 96 compact 44
```

以下は、NUMA 関連のメトリックです。

3.2.1 メモリー依存 – アプリケーションはメモリー依存か? メモリー依存の場合、帯域幅の使用ヒストグラムで DRAM 帯域幅の使用率が高いか? 計算負荷の高い作業はソケット間で等分されるため、ソケット間で帯域幅がバランスよく使用されていることが重要です。

[Summary (サマリー)] ウィンドウを使用して、アプリケーションがメモリー依存かどうか判断できます。



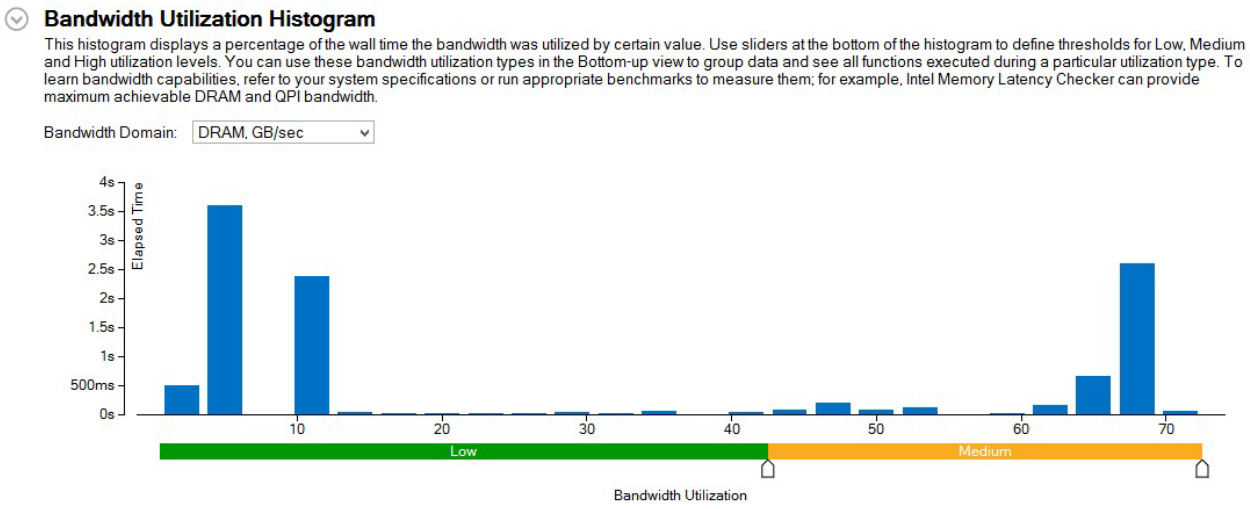


図 1: Memory Bound (メモリー依存) メトリックと DRAM 帯域幅ヒストグラム

Memory Bound (メモリー依存) メトリックは高く、ハイライトされていますが、**DRAM 帯域幅の使用状況** は、予想に反して低～中です。これはさらに調査する必要があります。

3.2.2 インテル® QuickPath インターコネクト (インテル® QPI) の帯域幅 アプリケーションのパフォーマンスは、ソケット間のインテル® QPI リンクの帯域幅により制限されることがあります。インテル® VTune™ Amplifier XE は、このような帯域幅の問題につながるソースとメモリー・オブジェクトを特定するためのメカニズムを備えています。

[Summary (サマリー)] ウィンドウの **[Bandwidth Utilization Histogram (帯域幅の使用ヒストグラム)]** で、**[Bandwidth Domain (帯域幅ドメイン)]** ドロップダウンから **QPI** を選択します。

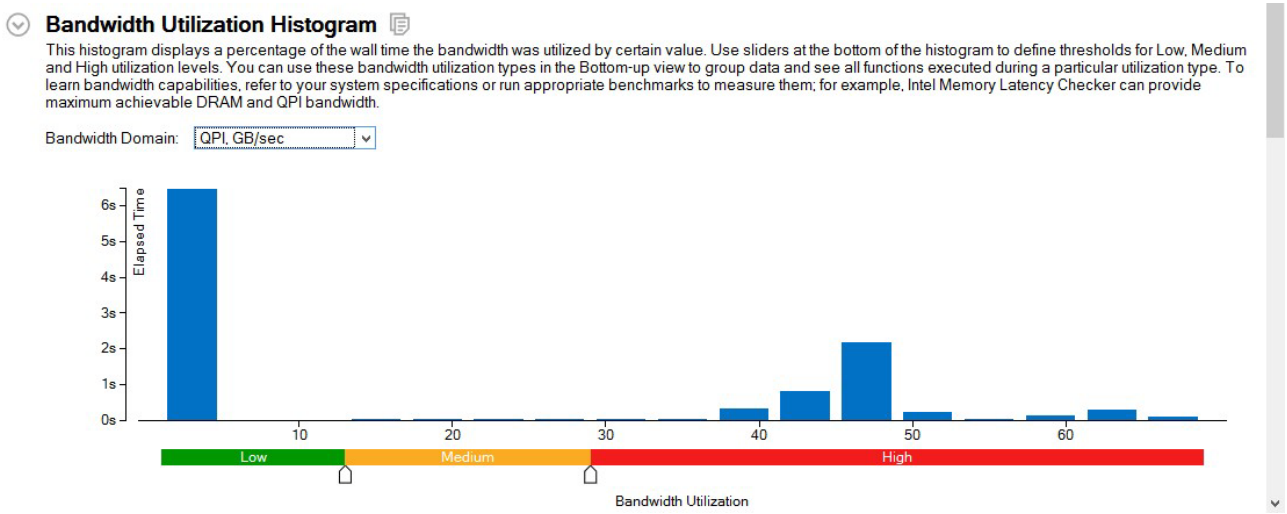


図 2: インテル® QuickPath インターコネクトの帯域幅使用ヒストグラム

[Bottom-up (ボトムアップ)] ビューに切り替えて、タイムライン・ビューで QPI 帯域幅の使用率が高い領域を選択して、フィルターできます。

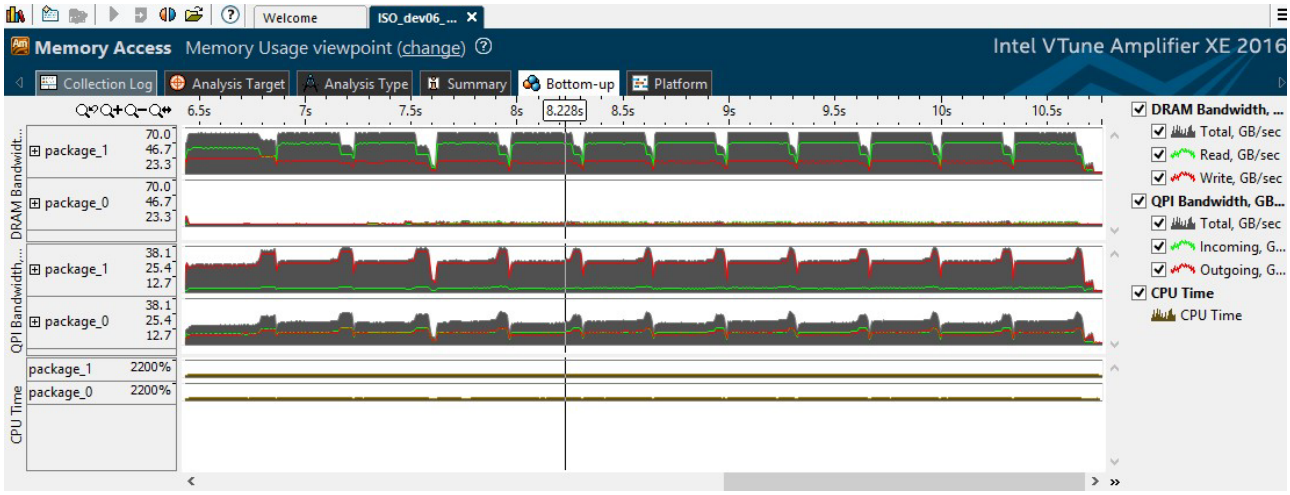


図 3: 帯域幅の使用のタイムライン・ビュー

フィルターを適用すると、タイムライン・グラフから、DRAM 帯域幅が 1 つのソケットでのみ使用されており、QPI 帯域幅が最大 38GB/秒と高いことが分かります。

[Bottom-up (ボトムアップ)] ビューで、タイムライン・ペインの下のグリッドは、その時間範囲に実行されたものを示します。QPI 使用率の高い関数を確認するには、**[Bandwidth Domain / Bandwidth Utilization Type / Function / Call Stack (帯域幅ドメイン / 帯域幅の使用状況 / 関数 / コールスタック)]** グループを選択し、**QPI** ドメインの **[High (高)]** を展開します。

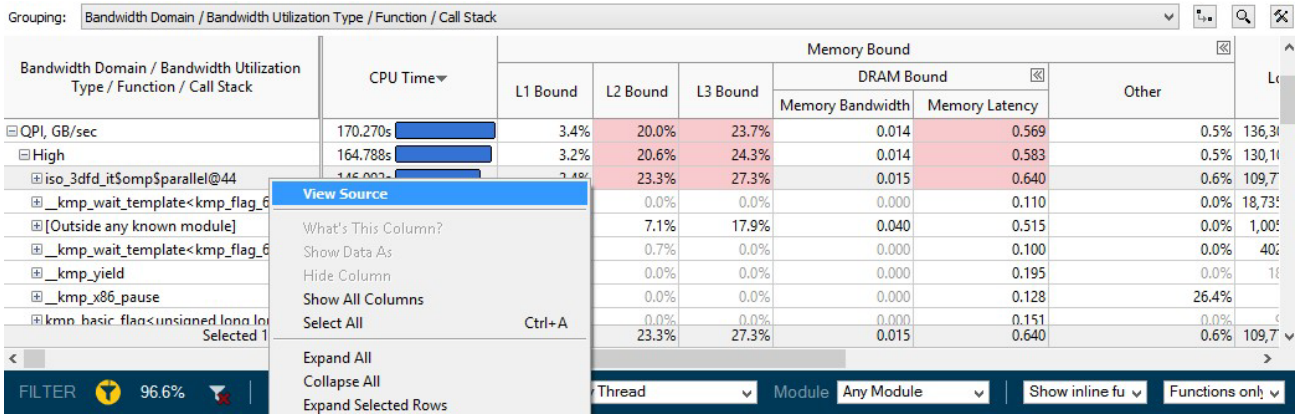


図 4: インテル® QuickPath インターコネクトの使用率が高い関数 - [Bottom-up (ボトムアップ)] グリッドビュー

これらは、NUMA マシンでよく見られる典型的な問題で、OpenMP* スレッドのメモリーが 1 つのソケットに割り当てられ、スレッドがすべてのソケットにわたってスポーンされます。そのため、一部のスレッドはインテル® QPI リンクを利用して、ローカルメモリーよりもアクセスに時間のかかるリモート DRAM やリモートキャッシュからデータをロードしなければなりません。

4. リモート・メモリー・アクセスを軽減するためのコード変更

NUMA の影響を抑えるため、各ソケットで実行中のスレッドは、ローカルメモリーにアクセスしてインテル® QPI トラフィックを軽減すべきです。これは、first-touch (最初のタッチ) ポリシーにより有効にできます。Linux* では、最初のアクセス時にメモリーページが割り当てられます。つまり、データは、最初に書き込まれるまでメモリーにマップされません。最初に書き込みを行う (タッチする) スレッドは、実行中の CPU の近くにデータを配置することができます。これを達成するためには、計算に使用されるのと同じ OpenMP* ループ順でメモリーを初期化す

る必要があります。*src/dev06/iso-3dfd_main.cc* (ISO3DFD ソースコード に同梱) にある初期化関数を、first-touch を有効にする initialize_FT に置換します。これにより、計算負荷の高い地震波が伝搬される *iso_3dfd_it* 関数で使用するデータブロックのアクセスと初期化が、スレッドのローカルメモリーで行われる可能性が高まります。また、初期化と計算の両方で、OpenMP* スケジュールを static に設定することで、より高いパフォーマンスを達成できます。

```
void initialize_FT(float* ptr_prev, float* ptr_next, float* ptr_vel, Parameters* p, size_t
nbytes, int n1_Tblock, int n2_Tblock, int n3_Tblock, int nThreads){

    #pragma omp parallel num_threads(nThreads) default(shared)
    {
        float *ptr_line_next, *ptr_line_prev, *ptr_line_vel;
        int n3End = p->n3;
        int n2End = p->n2;
        int n1End = p->n1;
        int ixEnd, iyEnd, izEnd;
        int dimn1n2 = p->n1 * p->n2;
        int n1 = p->n1;
        #pragma omp for schedule(static) collapse(3)
        for(int bz=0; bz<n3End; bz+=n3_Tblock){
            for(int by=0; by<n2End; by+=n2_Tblock){
                for(int bx=0; bx<n1End; bx+=n1_Tblock){
                    izEnd = MIN(bz+n3_Tblock, n3End);
                    iyEnd = MIN(by+n2_Tblock, n2End);
                    ixEnd = MIN(n1_Tblock, n1End-bx);

                    for(int iz=bz; iz<izEnd; iz++) {
                        for(int iy=by; iy<iyEnd; iy++) {
                            ptr_line_next = &ptr_next[iz*dimn1n2 + iy*n1 + bx];
                            ptr_line_prev = &ptr_prev[iz*dimn1n2 + iy*n1 + bx];
                            ptr_line_vel = &ptr_vel[iz*dimn1n2 + iy*n1 + bx];

                            #pragma ivdep
                            for(int ix=0; ix<ixEnd; ix++) {
                                ptr_line_prev[ix] = 0.0f;
                                ptr_line_next[ix] = 0.0f;
                                ptr_line_vel[ix] = 2250000.0f*DT*DT; // v² と dt² の積分
                            }
                        }
                    }
                }
            }
        }
    }

    float val = 1.f;
    for(int s=5; s>=0; s--){
        for(int i=p->n3/2-s; i<p->n3/2+s;i++){
            for(int j=p->n2/4-s; j<p->n2/4+s;j++){
                for(int k=p->n1/4-s; k<p->n1/4+s;k++){
                    ptr_prev[i*p->n1*p->n2 + j*p->n1 + k] = val;
                }
            }
        }
        val *= 10;
    }
}
```

5. 変更したコードのメモリーアクセス解析

ここで注目すべきメトリックは、DRAM 帯域幅の使用状況と QPI 帯域幅です。

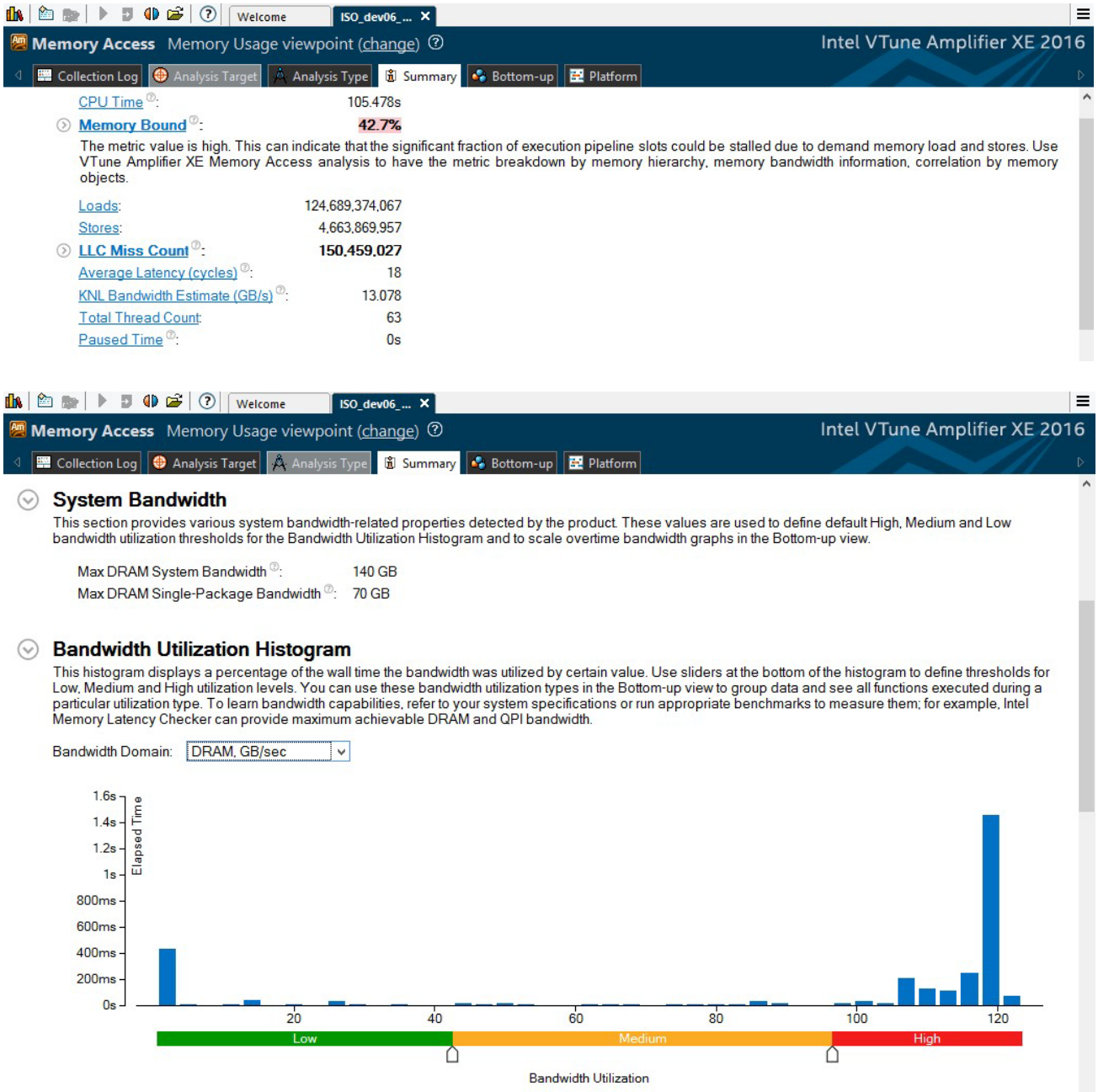


図 5: メモリー依存メトリックと DRAM 帯域幅の使用状況 - コード変更後

[Summary (サマリー)] ウィンドウから、アプリケーションがまだメモリー依存であり、帯域幅の使用率が高いことが分かります。

[Bandwidth Utilization Histogram (帯域幅の使用ヒストグラム)] で **[Bandwidth Domain (帯域幅ドメイン)]** ドロップダウンから **QPI** を選択すると、QPI 帯域幅の使用状況が低または中に軽減されたことが確認できます。

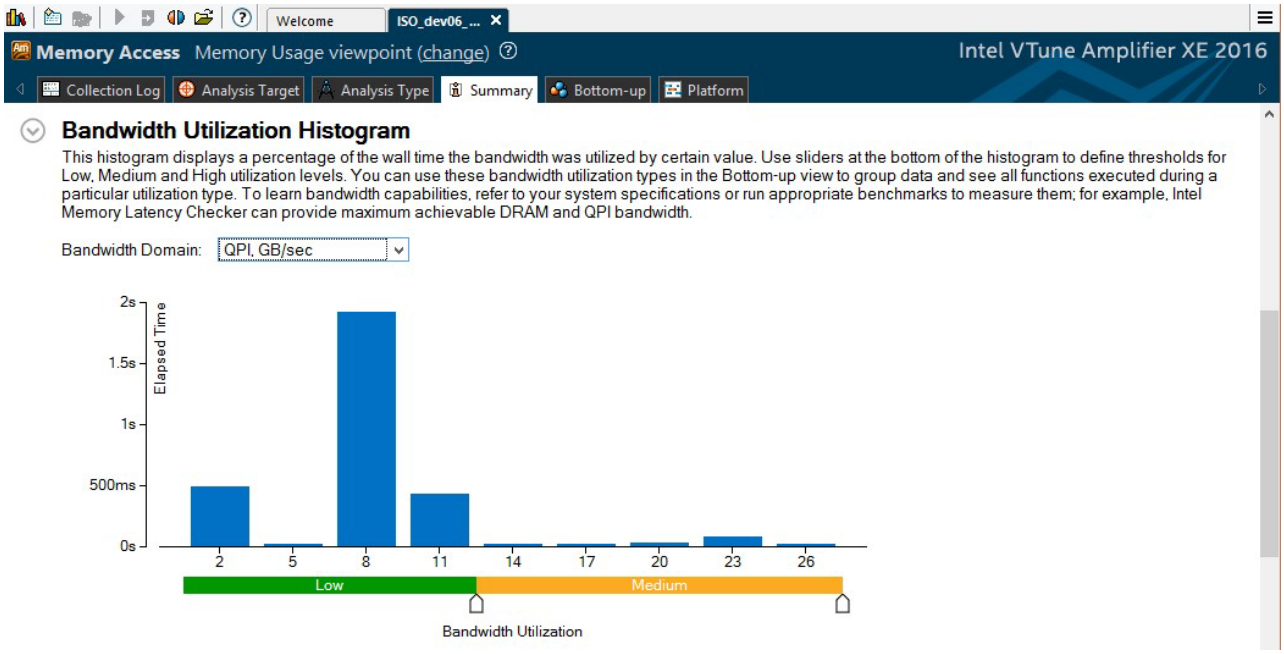


図 6: first-touch 適用後の QPI 帯域幅ヒストグラム

[Bottom-up (ボトムアップ)] ビューに切り替えてタイムラインを確認すると、DRAM 帯域幅がバランスよく使用され (2 つのソケット間で分散され)、QPI トラフィックも 1/3 になったことが分かります。

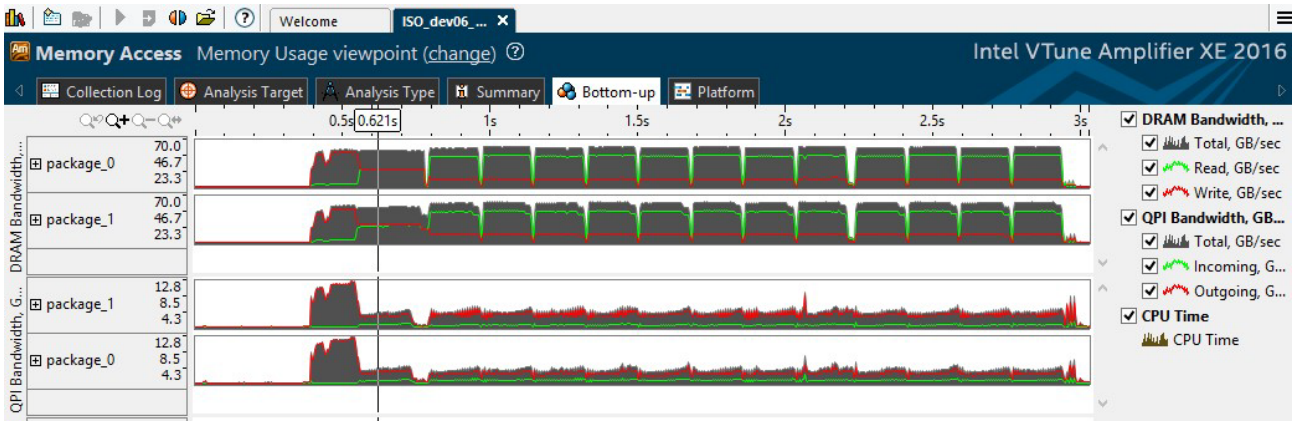


図 7: QPI トラフィックの軽減とバランスのよい各ソケットの DRAM 帯域幅

6. 全体的なパフォーマンスの比較

コード変更後のアプリケーションを次のように実行しました。

```
./run_on_xeon.pl bin/iso3dfd_dev06_cpu_avx2.exe 448 2016 1056 \
         10 448 24 96 compact 44

n1=448 n2=2016 n3=1056 nreps=10 num_threads=44 HALF_LENGTH=8
n1_thrd_block=448 n2_thrd_block=24 n3_thrd_block=96
allocating prev, next and vel: total 11694.4 Mbytes
--------------------------------
time:            1.70 sec
throughput:   5682.07 MPoints/s
flops:         346.61 GFlops
```

メモリーアクセスが改善され、アプリケーションのスループットが 2765 MPoints/秒から 5682 MPoints/秒に向上し、約 2 倍高速になりました。

一貫したパフォーマンスの向上が得られることを確認するため、コード変更前と変更後のバージョンをそれぞれ 10 回ずつ、1 回の実行につき 100 反復ずつ実行しました。OpenMP* スケジュールの変更によるパフォーマンスの向上と first-touch の適用により得られるパフォーマンスを比較するため、オリジナルの dev06 を OpenMP* スケジュールを static と dynamic に設定して実行しました。変更後の NUMA を認識するコードは、static 設定で実行したほうがより高いパフォーマンスが得られました。dynamic 設定では、各 OpenMP* スレッドとキャッシュブロック/秒 (このアプリケーションの OpenMP* スレッドの作業単位) の間のマッピングが反復ごとに非決定的となりました。その結果、dynamic 設定では first-touch の効果が見られませんでした。

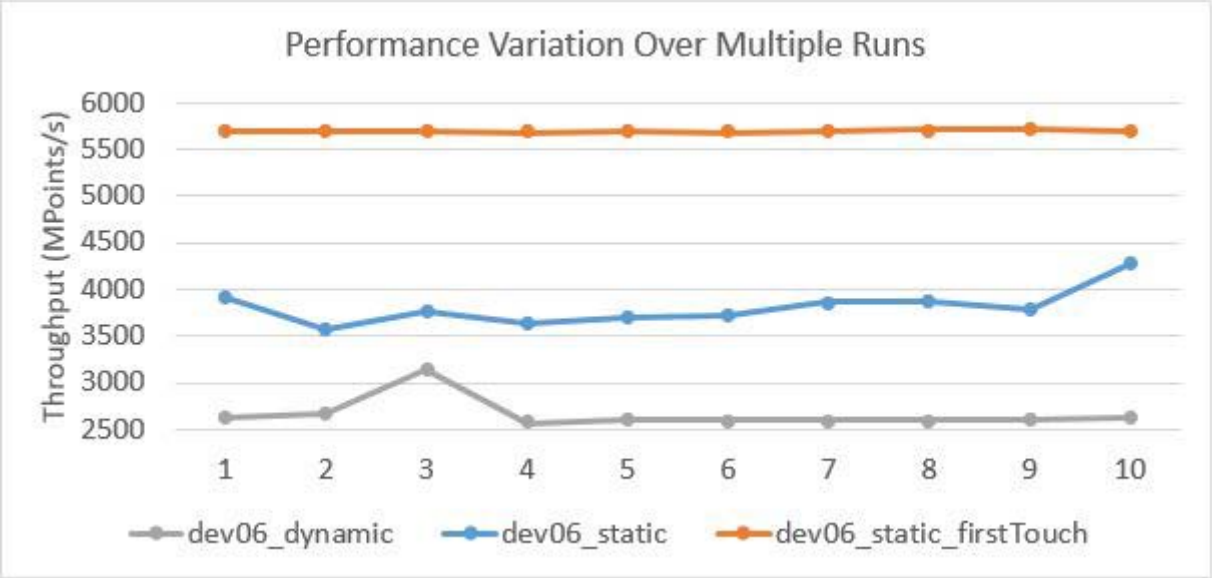


図 8: パフォーマンスの変化

7. システム構成

この記事で紹介したパフォーマンスは、次のシステム構成での測定値です。詳細については、http://www.intel.com/performance (英語) を参照してください。

コンポーネント	仕様
システム	2 ソケットのサーバー
ホスト・プロセッサー	インテル® Xeon® プロセッサー E5-2699 V4 @ 2.20GHz
コア/スレッド	44/44
ホストメモリー	64GB/ソケット
コンパイラー	インテル® C++ コンパイラー 16.0.2
プロファイラー	インテル® VTune™ Amplifier XE 2016 Update 2
ホスト OS	Linux* カーネル 3.10.0-327.el7.x86_64

8. 参考文献

等方性 3 次元有限差分コード向けの 8 つの最適化 (**https://software.intel.com/en-us/articles/eight-optimizations-for-3-dimensional-finite-difference-3dfd-code-with-an-isotropic-iso**) (英語)

インテル® VTune™ Amplifier XE 2016 (**http://www.isus.jp/intel-vtune-amplifier-xe/**)

インテル® VTune™ Amplifier XE - メモリー使用状況の解釈 (**https://software.intel.com/en-us/node/544170**) (英語)

NUMA (Non-uniform Memory Access) (**https://ja.wikipedia.org/wiki/NUMA**)

numactl - Linux* man ページ (**http://linux.die.net/man/8/numactl**) (英語)

インテル® Xeon Phi™ コプロセッサー上での等方性倍精度 3 次元有限差分ステンシル・アルゴリズムの実行パフォーマンスの最適化 (**http://www.isus.jp/products/vtune/optimizing-execution-performance-for-isotropic-dp/**)

マルチコア・プロセッサーにおける 3 次元有限差分法の実装 (**http://www.isus.jp/products/psxe/3d-finite-differences/**)

著者紹介

Sunny Gogar
ソフトウェア・エンジニア

フロリダ大学ゲインズビル校で電子情報工学の修士号、およびインド ムンバイ大学で電気通信学の学士号を取得しています。現在は、インテル コーポレーションのソフトウェア & サービスグループのソフトウェア・エンジニアで、並列プログラミング、マルチコア/メニーコア・プロセッサー・アーキテクチャー向けの最適化に取り組んでいます。

---

[1] この記事で比較したバージョンはすべて -fma などの新しいインテル® プロセッサー向けのコンパイラー・オプションを使用しています。

[2] numactl - プロセッサーや共有メモリーに対する NUMA ポリシーを制御します。

著作権と商標について

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。実際の性能はシステム構成によって異なります。詳細については、各システムメーカーまたは販売店にお問い合わせいただくか、http://www.intel.co.jp/ を参照してください。

本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、知的財産権の非侵害性への保証、およびインテル製品の性能、取引、使用から生じるいかなる保証を含みますが、これらに限定されるものではありません。

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

本資料で説明されている製品およびサービスには、不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

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

Intel、インテル、Intel ロゴ、Xeon、Intel Xeon Phi、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。

© 2016 Intel Corporation.

このサンプルコードは、インテル・サンプル・ソース・コード使用許諾契約書 (英語) の下で公開されています。

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