# メモリーアクセスのパフォーマンス・ボトルネックを特定

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「Finding your Memory Access performance bottlenecks」の日本語参考訳です。

---

アプリケーションがメモリーにアクセスする方法は、パフォーマンスに大きく影響します。スレッド化とベクトル化により、アプリケーションの並列性を高めるだけでは十分ではありません。メモリー帯域幅もまた重要ですが、ソフトウェア開発者にはあまり理解されていません。メモリーのレイテンシーの最小化と帯域幅の増加に役立つツールは、パフォーマンスのボトルネックを特定して、その原因を診断するのに有用です。

今日の最新プロセッサーでは、さまざまな階層および種類のメモリーアクセスが行われます。例えば、L1 キャッシュヒットのレイテンシーは、すべてのメモリー階層のキャッシュにミスして DRAM をアクセスする必要がある場合のレイテンシーとは大きく異なります。さらに、NUMA アーキテクチャーによる複雑性も加わります。

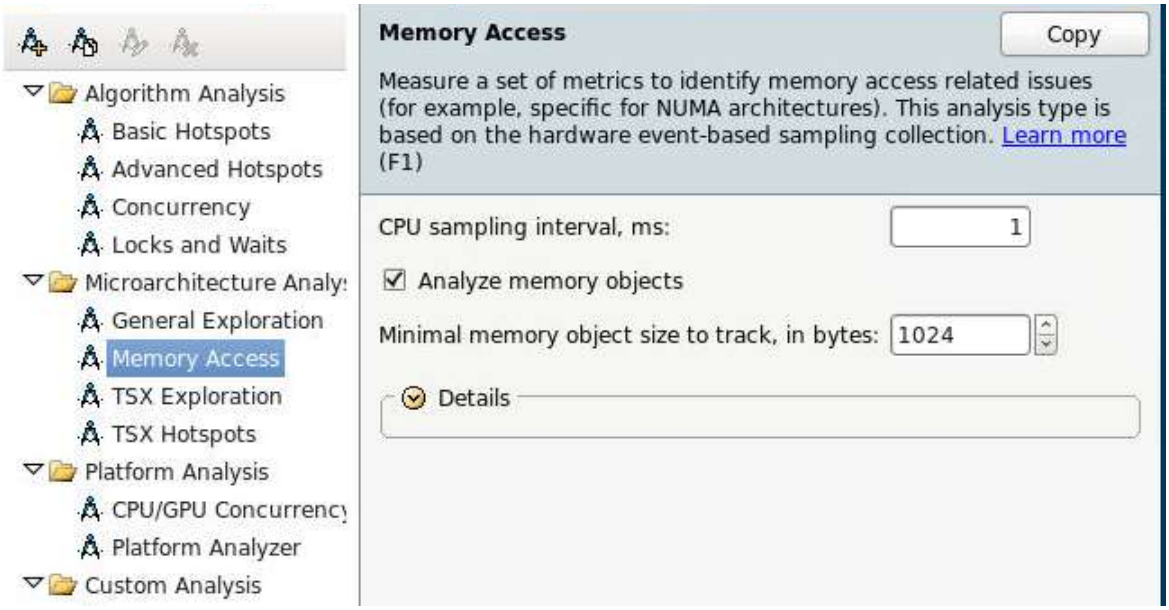
インテル® VTune™ Amplifier は、多くのメモリーアクセス解析機能を備えたパフォーマンス・プロファイラーです。これらの機能は、新しい「メモリーアクセス」解析タイプに含まれています。新しい解析タイプを使用して次のことができます。

1. メモリー階層 (L1、L2、LLC、DRAM 依存) による**パフォーマンスの問題を特定**します。
2. **メモリー・オブジェクトを追跡**し、そのオブジェクトが引き起こすレイテンシーをコードおよびデータ構造に関連付けます。
3. **帯域幅が制限されるアクセス** (DRAM およびインテル® QuickPath インターコネクト [インテル® QPI] 帯域幅を含む) を解析し、プログラム実行の時系列全体の帯域幅を示す DRAM とインテル® QPI のグラフと分布図を表示します。
4. パフォーマンスの低下に直結する **NUMA 関連の問題を特定**します。

この記事では新しい「メモリーアクセス」解析の機能を紹介し、この機能を使用していくつかの困難なメモリーの問題を特定し、アプリケーションのパフォーマンスを大幅に高める方法を示します。

**概要**

インテル® VTune™ Amplifier のメモリーアクセス解析機能を利用するには、[メモリーアクセス] 解析タイプを選択して、[開始] をクリックします。




**帯域幅利用率**

DRAM とインテル® QPI の帯域幅がどのくらい有効利用されているか確認できます (図 13)。高い帯域幅の利用率を考慮する必要があります。これを解決するため、帯域幅に関連するコード内の場所を特定できます。(図 14)

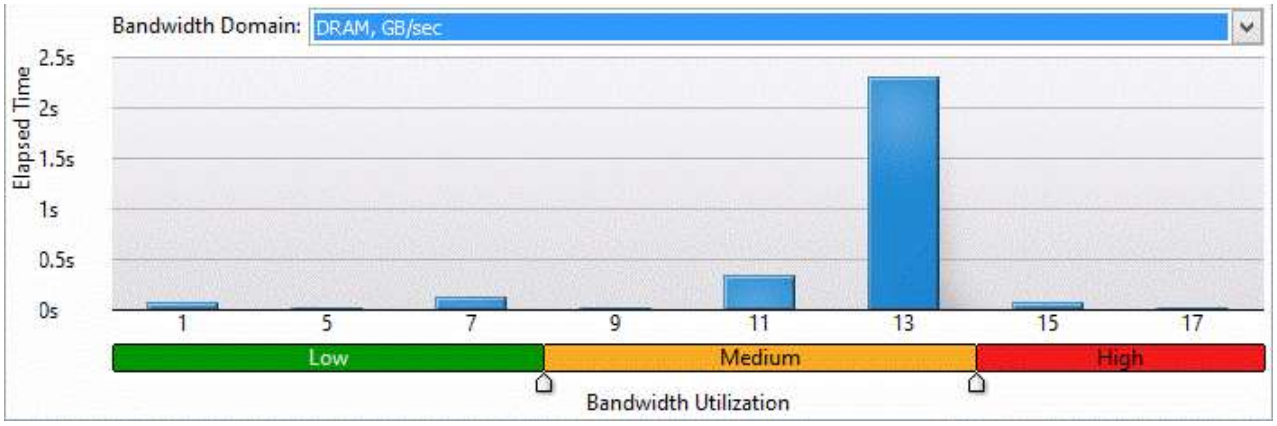
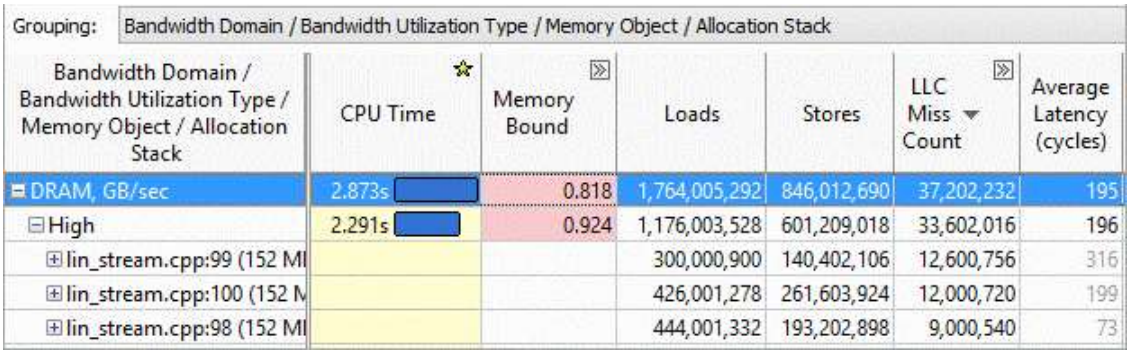



図 13.帯域幅ヒストグラム

**帯域幅を含むメモリー・オブジェクトを表示**

帯域幅を含むコードのソースとメモリー・オブジェクトを特定します。帯域幅ドメインをグループ化することにより、メモリー帯域幅に最も関連するメモリー・オブジェクトを特定できます (図 14)。DRAM やインテル® QPI などの問題がある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 |

図 14.メモリー帯域幅

**アプリケーションの実行期間全体にわたるメモリー帯域幅のグラフ**

メモリー帯域幅は、通常、プログラムの実行によって異なります。読み取り/書き込み帯域幅を示す GB/秒単位のグラフを表示することで、アプリケーションのメモリー使用量の変化を確認して、アプリケーションが追加のメモリー使用を行うコード領域を見つけることができます。その後、急激なメモリー使用が発生したタイムライン領域を選択して、フィルター処理を行い、その期間中にアクティブであるコードのみを表示できます。

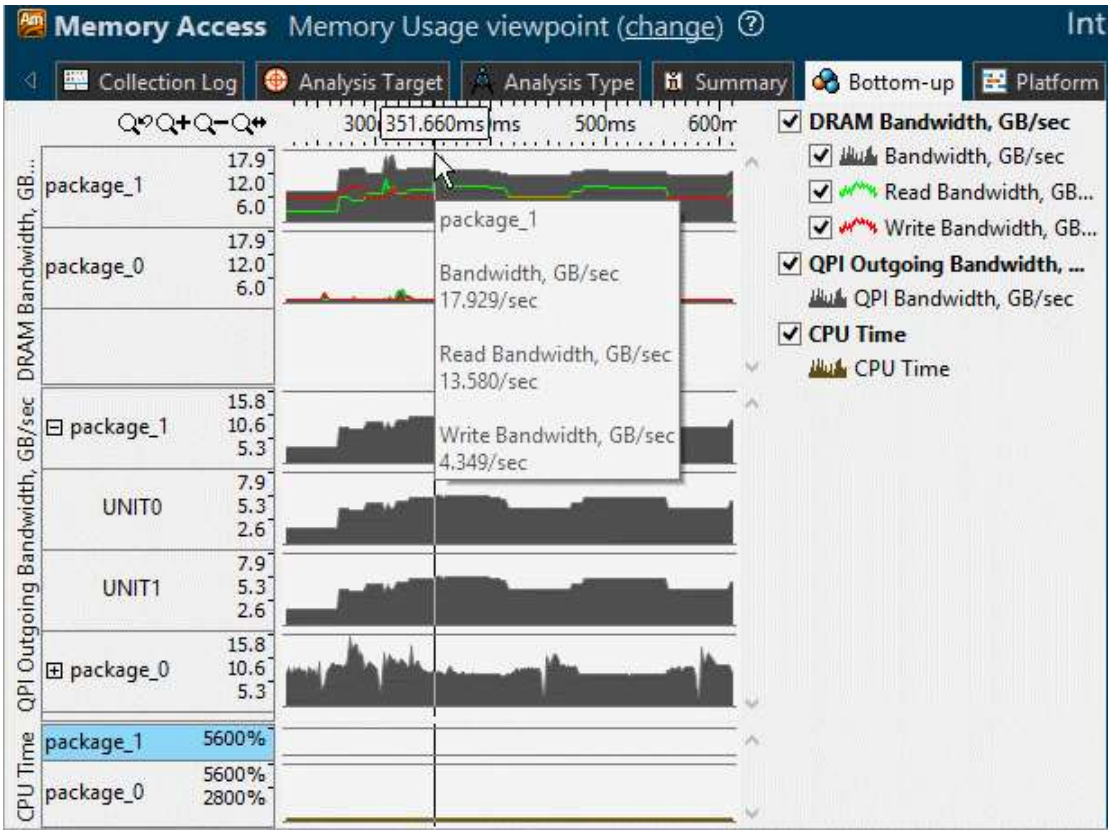



図 15.メモリー使用

メモリー帯域幅の増加に関連するアプリケーションのコード領域を追跡する機能は強力です。メモリーアクセスをチューニングする場合、平均レイテンシーも重要になります。タイムラインに表示される帯域幅グラフで、アプリケーションの実行中にメモリー使用量の概要を理解することができます。インテル® VTune™ Amplifier

の最新バージョンでは、帯域幅グラフはプラットフォームが提供できる最大値の相対値として示されるため、残されているパフォーマンスを明確に確認できます。

**インテル® VTune™ Amplifier を使用して解決できるメモリー・オブジェクトの問題**

**困難な問題 1 - フォルス・シェアリング**

最初にいくつかの簡単な定義を確認しましょう。複数スレッドが同じメモリーにアクセスする場合、それらはメモリーを「共有」する、と言われます。現代のコンピューターのアーキテクチャーと構成により、この共有はあらゆる種類のパフォーマンス上のペナルティーにつながる可能性があります。異なるすべてのスレッド/コアは、メモリーアドレスに格納されている内容を一致させ、競合するキャッシュ階層をすべて同期する必要があるため、パフォーマンスのペナルティーが生じます。

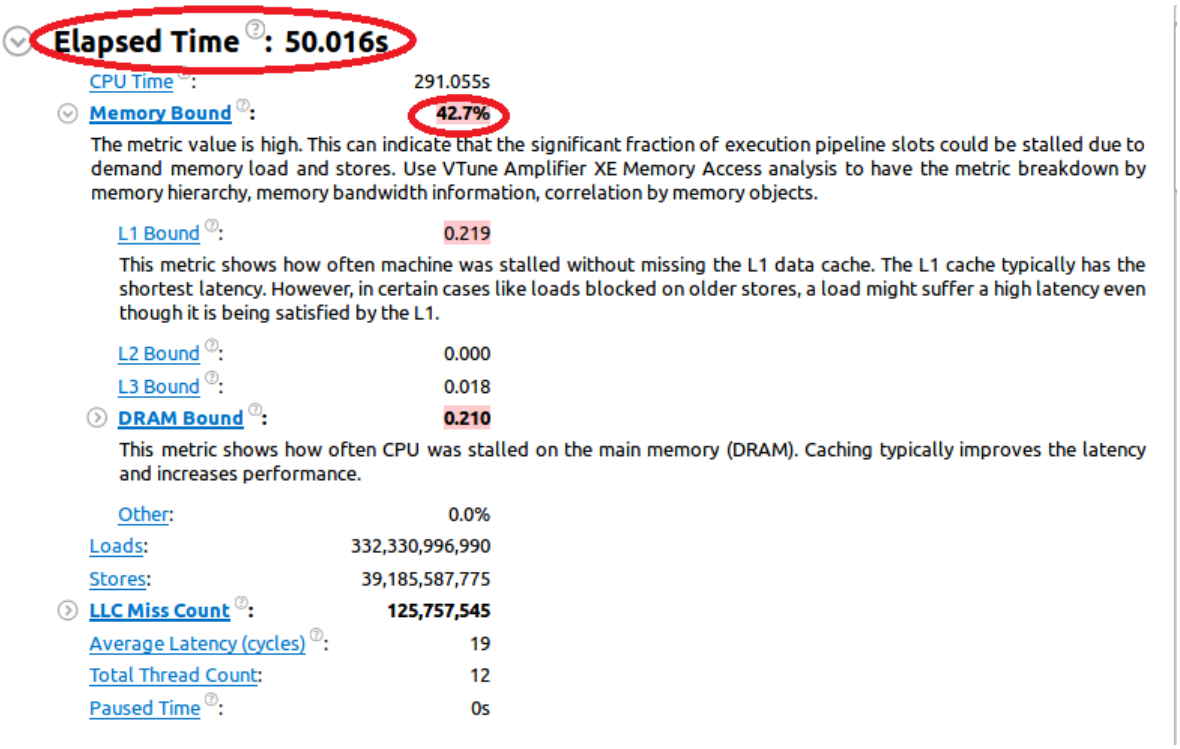
フォルス・シェアリング (偽りの共有) とは、異なるスレッドが同じキャッシュラインにあるデータにアクセスする際に、実際には同じデータを共有しませんが、メモリー参照が隣接するため同一キャッシュラインに格納されることを意味します。複数のスレッドがフォルス・シェアリング関係にある場合、同じメモリー位置を共有するのと同じ種類のペナルティーが生じますが、これは不要なペナルティーです。

ここでは、phoenix スイート (http://csl.stanford.edu/~christos/sw/phoenix (英語)) の linear_regression アプリケーションを調査します。

**ステップ 1 - メモリーアクセス解析を実行して潜在的なメモリーの問題を明確にする**

[メモリー・オブジェクト解析] オプションを有効にして、メモリーアクセス解析を実行します。すべてのメモリー位置を取得するには、オブジェクト・サイズのしきい値を 1 に設定します。

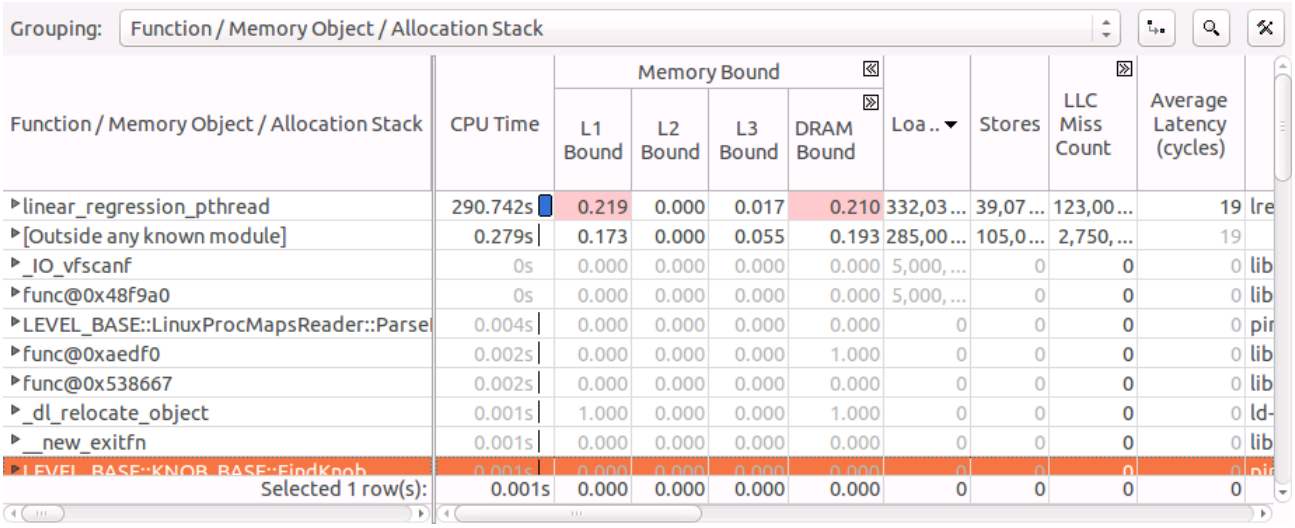
サマリービューにはいくつかのメトリックが表示されます。



最初の実行の「経過時間」は 50 秒でした。また、アプリケーションは「メモリー依存」であり、CPU リソースの 42% 以上がメモリー操作の完了を待機していることが分かります。「メモリー依存」メトリックがピンク色で表示されていることに注目してください。これは、潜在的なパフォーマンスの問題に対処する必要があることを示しています。

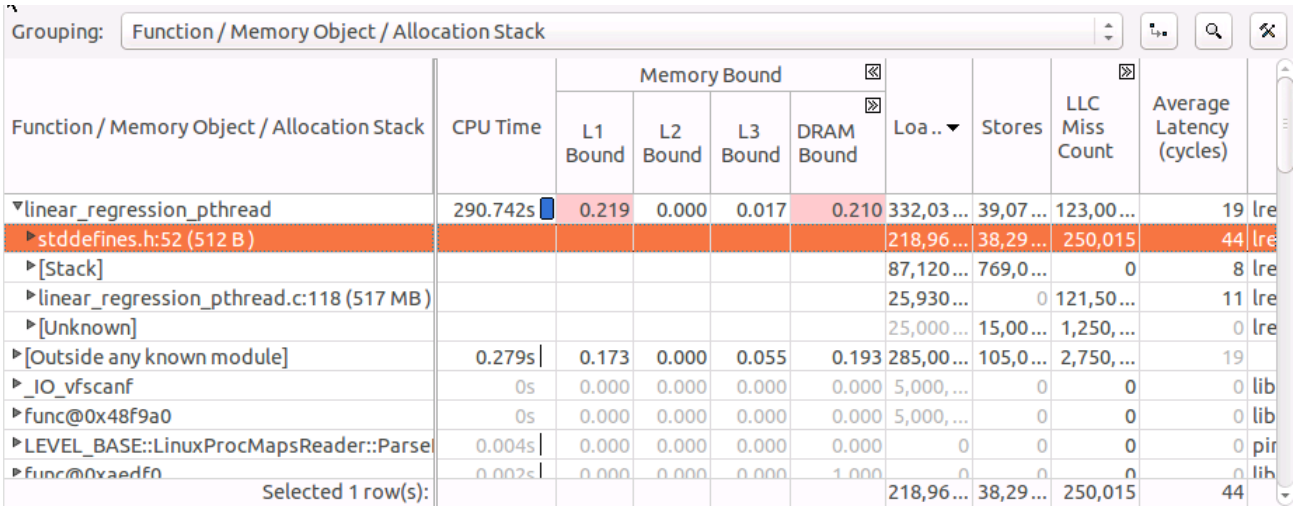
**ステップ 2 - 特定したメモリーの問題を調査する**

[ボトムアップ] タブに切り替えて詳細を表示します。




ほとんどの時間が、linear_regression_pthread 関数で費やされていることが分かります。この関数は、L1 と DRAM 依存であることも示されています。

linear_regression_pthread 関数のグリッド行を展開して、アクセスするメモリー・オブジェクトを確認し、ロードでソートします。




最もホットなオブジェクト ('stddefines.h:52 (512B)') は非常に小さく 512 バイトしかないため、L1 キャッシュに収まると考えられますが、「平均レイテンシー」メトリックは 44 サイクルとなっています。これは、L1 アクセス・レイテンシー 4 サイクルをはるかに上回っています。これには、真の共有またはフォルス・シェアリングによる競合の可能性があります。'stddefines.h:52 (512B)' オブジェクトの割り当てスタックを確認すると、オ

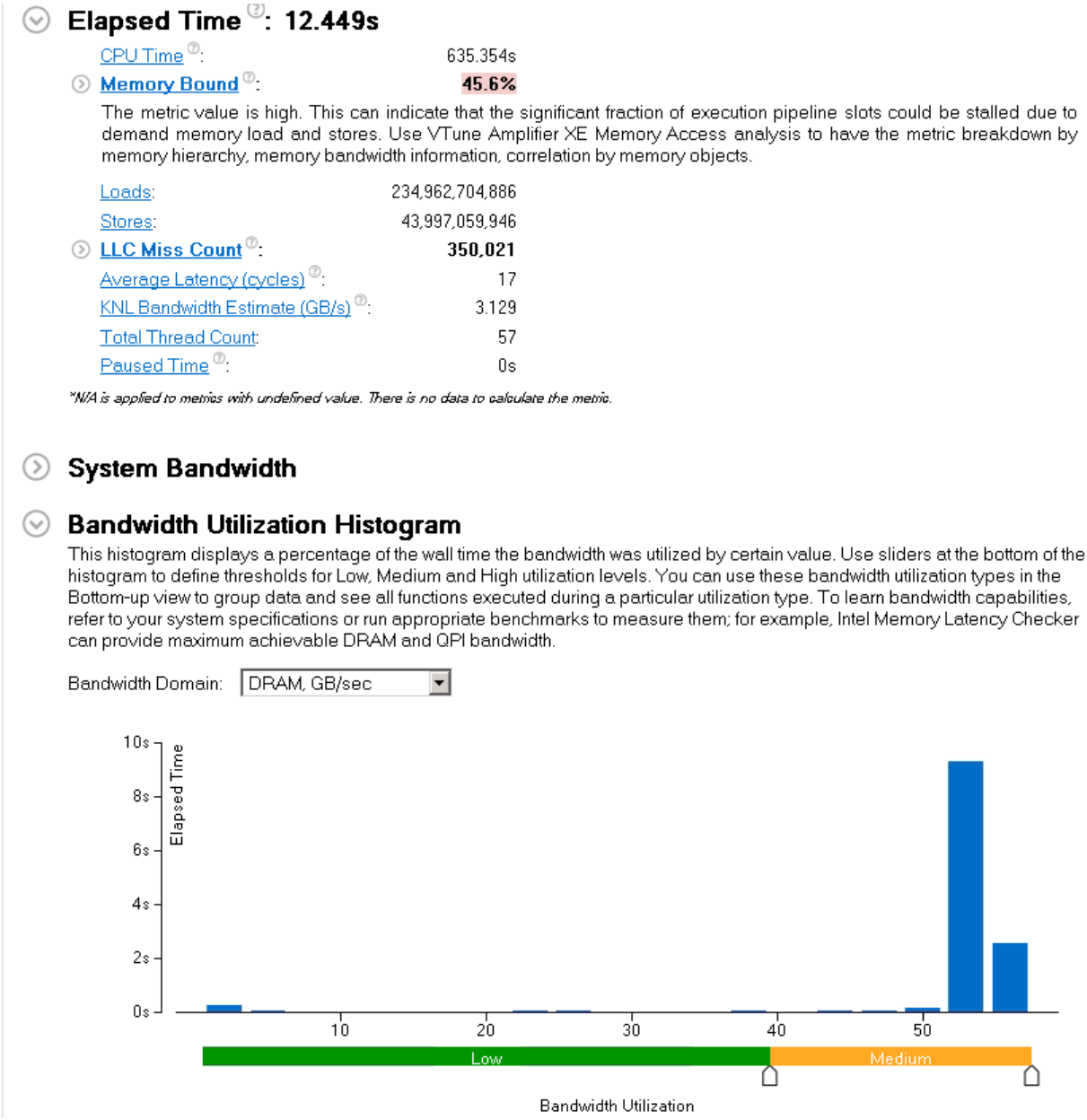
ブジェクトが割り当てられたソース行が分かります。各スレッドが配列内の要素に個別にアクセスしているため、これはフォルス・シェアリングであると考えられます。

### ステップ 3 - フォルス・シェアリングを回避するようにコードを修正する

スレッドが常に異なるキャッシュラインをアクセスするよう、配列にパディングを追加することで、容易にフォルス・シェアリングを回避できます。

### ステップ 4 - メモリーアクセス解析を再度実行する

以下は再実行後の結果です。



経過時間は 12 秒に短縮されました。構造体をパディングする 1 行のコードを追加するだけで、アプリケーションのパフォーマンスをおよそ 4 倍向上できました。「メモリー依存」メトリックもまた低下し、「L1 依存」の問題は排除されました。

**困難な問題 2 - NUMA の問題**

Non-Uniform Memory Access (NUMA) をサポートするプロセッサーでは、アプリケーションが実行されるプロセッサーのキャッシュミスを認識するだけでは不十分です。NUMA アーキテクチャーでは、別のソケットの CPU キャッシュと DRAM を参照することもできます。このタイプのアクセス・レイテンシーは、ローカルのレイテンシーより桁違いに大きくなります。リモート・メモリー・アクセスを識別して、最適化する必要があります。

このケースでは、Haswell†ベースのデュアルソケットのインテル® Xeon® プロセッサー上で実行される、OpenMP* の機能を使用して並列化された簡単な Triad アプリケーションを使用します。

以下にコードを示します。

```
static double   a[N];
static double   b[N];
static double   c[N];

void doTriad(double x)
{
#pragma omp parallel for
    for (int i = 0; i < N; i++)
        a[i] = b[i] + x*c[i];
}

int main() {

    for (int i = 0; i < N; ++i)
    {
        a[i] = 1.0;
        b[i] = 2.0;
        c[i] = 0.0;
    }

    for (int n = 0; n < NTIMES; ++n)
    {
        doTriad(3.0);
    }

    return 0;
}
```

最初に配列を初期化し、omp parallel for を使用する doTriad 関数を呼び出します。

**ステップ 1 - メモリーアクセス解析を実行する**

このアプリケーションのメモリーアクセス解析を実行します。DRAM 帯域幅に依存することが予想されるため、システム帯域幅を最大限に活用する必要があります。

## Elapsed Time: 12.449s

| | |
|---|---|
| CPU Time: | 635.354s |
| **Memory Bound**: | **45.6%** |

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use VTune Amplifier XE Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

| | |
|---|---|
| Loads: | 234,962,704,886 |
| Stores: | 43,997,059,946 |
| **LLC Miss Count**: | **350,021** |
| Average Latency (cycles): | 17 |
| KNL Bandwidth Estimate (GB/s): | 3.129 |
| Total Thread Count: | 57 |
| Paused Time: | 0s |

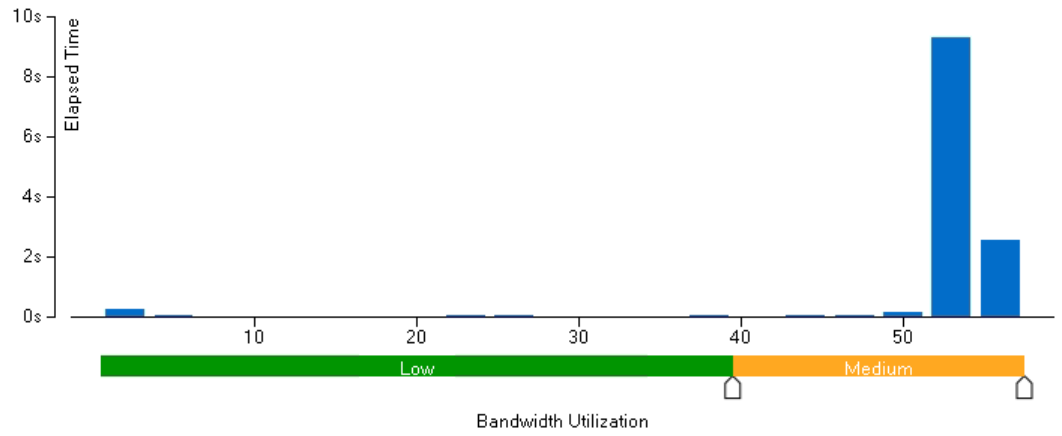
**N/A is applied to metrics with undefined value. There is no data to calculate the metric.*

## System Bandwidth

## Bandwidth Utilization Histogram

This histogram displays a percentage of the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.

Bandwidth Domain: DRAM, GB/sec



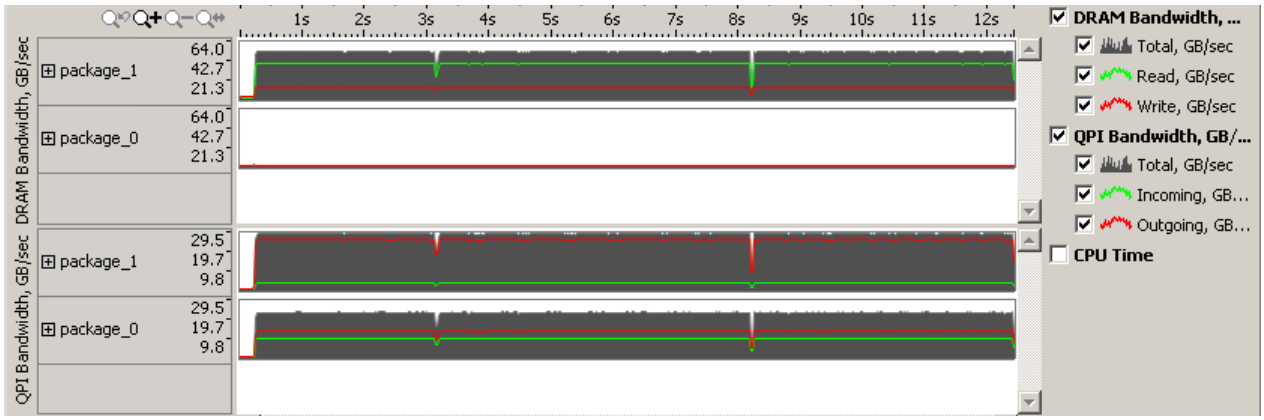

サマリーレポートでは、いくつかの有用なメトリックが示されます。経過時間は 12.449 秒でした。予想通り、「メモリー依存」メトリックは高くハイライト表示されています。不可解なことは、「帯域幅利用率」分布図が中程度の DRAM 帯域幅利用率レベル (50 - 60GB/秒) を示していることです。これは調査の必要があります。

その他の有用なメトリック:

- 平均レイテンシー - メモリーアクセスにかかる平均サイクル数。
  注意: L1 キャッシュアクセスはわずか 4 サイクルで済みますが、リモート DRAM へのアクセスには最大 300 サイクル必要です。
- KNL†帯域幅予測 - これは、インテル® Xeon Phi™ プラットフォーム (開発コード名 Knights Landing) で実行した場合のコアあたりの帯域幅の予測値です。これは、KNL†への移行を計画する利用者にとって、コードのメモリーアクセスが適しているかどうか推測するのに有用です。

**ステップ 1 - 帯域幅利用率を調査する**

[ボトムアップ] タブに切り替えて詳細を表示します。




[タイムライン] グラフから、DRAM 帯域幅は package_1 のソケットのみを示していることが分かります。さらに、最大 30GB/秒の高いインテル® QPI (ソケット間) トラフィックが見られます。これは、1 つのソケットにメモリーが割り当てられ、ワークが複数のソケットに分割される NUMA マシンの典型的な問題の 1 つです。これにより、一部のワークはインテル® QPI リンクを介してリモートメモリーからデータをロードする必要があり、ローカルメモリーにアクセスするよりはるかに低速です。

**ステップ 2 - リモート・メモリー・アクセスを回避するようにコードを修正する**

リモートメモリーへのアクセスを回避し、ローカルメモリーだけをアクセスするようにコードを変更できれば、さらに高速に実行できます。Linux* では、メモリーページは最初のアクセス時に割り当てられるため、解決策は容易に実装できます。ページを操作するソケットと同じソケットでメモリーを初期化する必要があります。これは、初期化を行うループに "omp parallel for" ディレクティブを追加することで達成できます。

**ステップ 3 - KMP_AFFINITY 環境変数を設定してメモリーアクセス解析を再度実行する**

経過時間は 12.449 秒から 6.69 秒に短縮され、およそ 2 倍のスピードアップが達成できました。また、「DRAM 帯域幅」利用率は、期待通り高いレベルに移行しました。

## Elapsed Time: 6.692s

| | |
|---|---|
| CPU Time: | 343.239s |
| **Memory Bound**: | **35.7%** |

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use VTune Amplifier XE Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

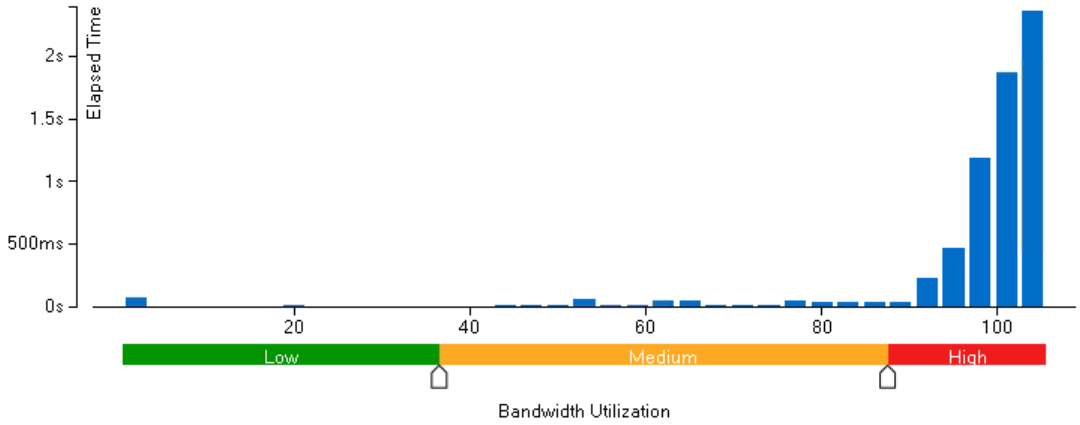
| | |
|---|---|
| Loads: | 210,196,630,588 |
| Stores: | 42,156,032,331 |
| **LLC Miss Count**: | **700,042** |
| Average Latency (cycles): | 12 |
| KNL Bandwidth Estimate (GB/s): | 3.517 |
| Total Thread Count: | 57 |
| Paused Time: | 0s |

## System Bandwidth

## Bandwidth Utilization Histogram

This histogram displays a percentage of the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.

Bandwidth Domain: DRAM, GB/sec




帯域幅はソケット間で均等に分散され、インテル® QPI トラフィックは 1/3 になっています。

NUMA アーキテクチャーでは、その複雑性によりメモリーアクセスに注意を払う必要があります。最大のレイテンシーを被るアプリケーションのメモリーアクセスを最適化することで、潜在的なパフォーマンスを最大限に高めることができます。

**まとめ**

プログラムのメモリーアクセスを最適化することはパフォーマンス向上には重要です。インテル® VTune™ Amplifier のようなツールを使用して、プログラムがどのようにメモリーをアクセスするか理解することは、ハードウェアのリソースを最大限に活用することにつながります。

ここでは、インテル® VTune™ Amplifier の新しいメモリーアクセス解析機能の概要を紹介しました。また、この機能を利用することで、いくつかの発見が難しいメモリーの問題を解決できることも示しました。

比較的小規模なメモリー・オブジェクトに対する高い平均レイテンシー値を見つけることで、フォルス・シェアリングの問題を特定する方法を示しました。これには、構造体をパディングする 1 行のコードを追加するだけで、アプリケーションのパフォーマンスを 4 倍向上できました。

さらに、大量のリモート・メモリー・アクセスを行う NUMA 固有の問題を特定する方法を紹介しました。リモートアクセスを排除することで、アプリケーションのパフォーマンスが 2 倍向上しました。

開発コード名
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。