インテル® VTune™ Amplifier XE 2017

マイクロアーキテクチャーのトップダウン解析法を使用して、アプリケーションをチューニングする

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、インテル® VTune™ Amplifier XE 2017 のオンラインヘルプの一部「Tuning Applications Using a Top-down Microarchitecture Analysis」の日本語参考訳です。

目次

アプリケーションが CPU マイクロアーキテクチャーの利点を活用するには、アプリケーションが利用可能なハードウェア・リソースをどのように使用しているか知る必要があります。1 つの手段として、オンチップのパフォーマンス・モニタリング・ユニット (PMU) を使用する方法があります。PMU は、システム上で発生した特定のハードウェア・イベントをカウントする CPU コア内部の専用ロジックです。これらのイベントには、キャッシュミスや分岐予測ミスなどがあります。これらを組み合わせることで、命令ごとのサイクル数 (CPI) などの高レベルのメトリックを構成して監視できます。

特定のマイクロアーキテクチャーでは、PMU を介して数百ものイベントを利用できることがあります。しかし、特定のパフォーマンスの問題を検出して解決する際に、どのイベントが有用であるかを判断することは容易ではありません。生のイベントデータから意味のある情報を知るには、マイクロアーキテクチャーの設計と PMU 仕様の両方に関する深い知識が求められます。しかし、事前定義されているイベント、メトリック、トップダウン特性化方法論を使用して、データを実用的な情報に変換することができます。

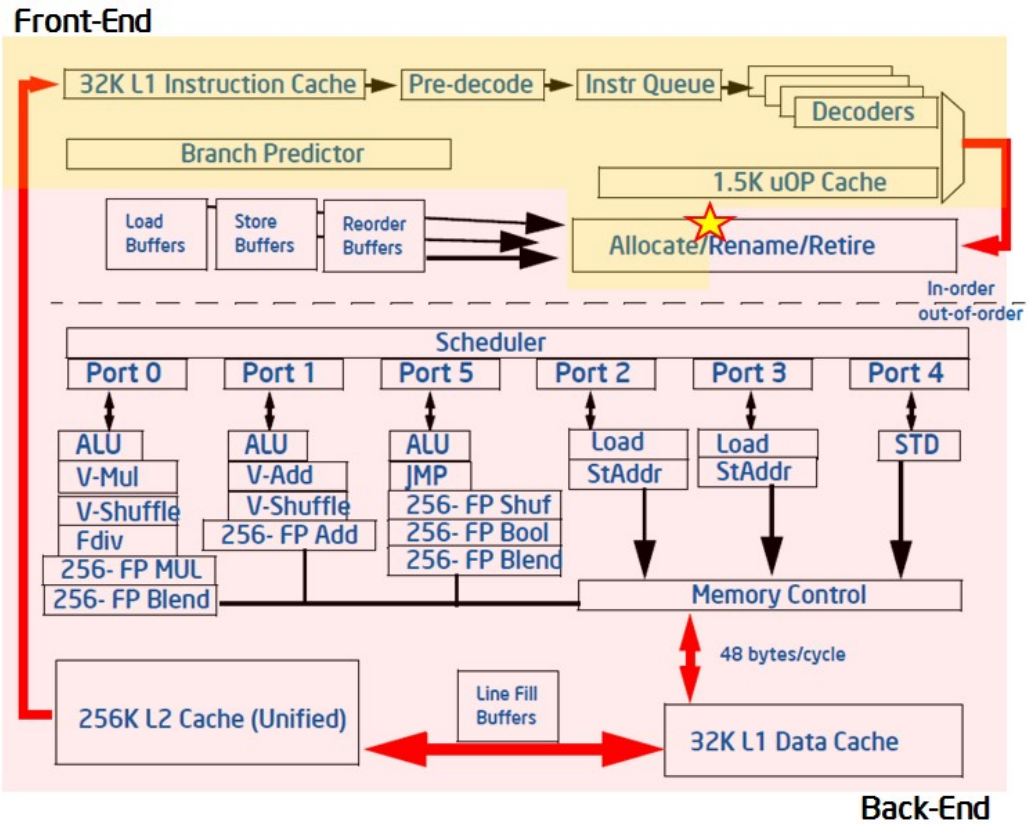
トップダウン・マイクロアーキテクチャー解析法の概要

現代の CPU はリソースを可能な限り有効に活用するため、ハードウェアによるスレッド化、アウトオブオーダー実行、命令レベルの並列性などの技術とともにパイプラインを採用しています。しかし、ソフトウェア・パターンとアルゴリズムの中には、依然として非効率なものがあります。例えば、リンクデータ構造はソフトウェアで良く利用されますが、これはハードウェア・プリフェッチャーの有効性を損ねる間接アドレスの原因となります。多くの場合、このような振る舞いはパイプラインに無益な隙間 (パイプライン・バブル) を作り、データが取得されるまで実行する命令がない状態となります。リンクデータ構造は、ソフトウェアの問題に対しては適切なソリューションですが、非効率な結果となるでしょう。これ以外にも、CPU パイプラインに関連して重要な意味を持つ多くのソフトウェア・レベルの例があります。トップダウン・マイクロアーキテクチャー解析法は、トップダウン特性化方法論をベースとして、適切なアルゴリズムとデータ構造を選択しているかどうか、詳しい情報を提供することを目指しています。トップダウン・マイクロアーキテクチャー解析法のさらに詳しい情報については、『Intel® 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1』(英語) をご覧ください。

トップダウン特性化は、アプリケーションのパフォーマンス・ボトルネックを特定するイベントベースのメトリックを階層的に構成します。これは、CPU がアプリケーションを実行する際のパイプラインの利用率の平均を示します。イベントを解釈する以前のフレームワークは、CPU のクロックティックのカウントに依存していました。どの要素の CPU のクロックティックが、どの操作 (L2 キャッシュミスによるデータ取得など) に費やされたかで判別します。このフレームワークの代わりに、パイプラインのリソースをカウントする方法を使用します。トップダウン特性化を理解するため、高レベルのマイクロアーキテクチャーの概念を調査します。マイクロアーキテクチャーの詳

細の多くはこのフレームワークで抽象化されており、ハードウェアのエキスパートでなくても理解して、使用することができます。

現代のハイパフォーマンス CPU のパイプラインは、非常に複雑です。以下の図に示すように、パイプラインは概念的にフロントエンドとバックエンドの 2 つに分割できます。フロントエンドは、アーキテクチャーの命令を表すプログラムコードをフェッチして、それらを µop (マイクロオペレーション) と呼ばれる 1 つ以上の低レベルの操作にデコードします。µop は、パイプラインのアロケーションと呼ばれる過程でバックエンドへ送られます。アロケーションが行われると、バックエンドは µop のデータオペランドが利用可能になるのを監視し、利用可能な実行ユニットで µop を実行する役割を持ちます。µop の実行完了はリタイアと呼ばれ、µop の実行結果がアーキテクチャー状態にコミットされます (CPU レジスターやメモリーへの書き戻し)。通常、ほとんどの µop はパイプラインを通過してリタイアしますが、投機的にフェッチされた µop はリタイアする前に取り消される場合があります (分岐予測ミスのようなケース)。



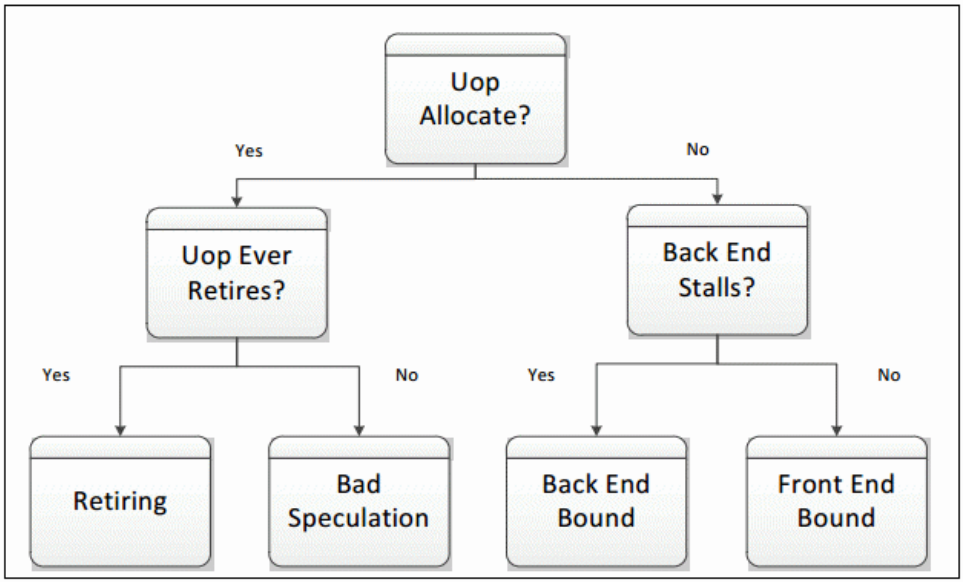
最近のインテル® マイクロアーキテクチャーにおけるパイプラインのフロントエンドは、サイクルごとに 4 つの µop をアロケートでき、バックエンドはサイクルごとに 4 つの µop をリタイアでき

ます。これらの機能を考慮して、パイプライン・スロットの抽象化の概念を定義できます。パイプライン・スロットは、1 つのマイクロオペレーションを操作するために必要なハードウェアのリソースを表します。トップダウン特性化では、それぞれの CPU コアには、各クロックサイクルで利用可能な 4 つのパイプライン・スロットがあると仮定します。そして、これらのパイプライン・スロットがどの程度効率良く利用されているか測定するため、特殊な PMU イベントを使用します。パイプライン・スロットの状態は、アロケーションの時点で取得され (上記の図で ★ 印が記された場所)、µop はフロントエンドからバックエンドへ移ります。アプリケーションの実行中に利用可能な各パイプライン・スロットは、前述したパイプライン・ビューの単純化によって 4 つのカテゴリーのいずれかに分類されます。

すべてのサイクルにおいて、パイプライン・スロットは空であるか、µop で埋められているかのどちらかです。スロットが 1 クロックサイクルの間空であるならば、それはストールしていると考えられます。このパイプライン・スロットを分類するための次のステップは、パイプラインのフロントエンドまたはバックエンドのどちらがストールの原因かを明確にすることです。これは、指定された PMU イベントと式を使用して行われます。トップダウン特性化の目的は、重要なボトルネックを特定することなので、ストールがフロントエンドもしくはバックエンドのどちらによって引き起こされたかは、重要な考慮点です。一般に、ストールの原因がフロントエンドによる µop 供給不足であれば、このサイクルは Front End Bound (フロントエンド依存) として分類され、パフォーマンスはフロントエンド依存カテゴリーにあるいくつかのボトルネックにより制限されます。また、フロントエンドは µop を準備しているけれども、バックエンドがまだそれを受け入れることができない場合、Back End Bound (バックエンド依存) の空のパイプライン・スロットとして分類されます。バックエンド・ストールは、一般にバックエンドがいずれかのリソース (ロードバッファーなど) を使い果たしていることにより生じます。そして、フロントエンドとバックエンドの両方がストールしている場合、そのスロットはバックエンド依存として分類されます。これは、そのような場合にフロントエンドのストールを解決しても、アプリケーションのパフォーマンス改善に直結することが少ないためです。バックエンドが障壁となるボトルネックであるなら、フロントエンドの問題を解決する前にそれを排除する必要があります。

プロセッサーがストールしていない場合、アロケーションでパイプライン・スロットに µop が満たされます。この場合、スロットをどのように分類するかを決定する要因は、µop が最終的にリタイアするかどうかです。リタイアすれば、そのスロットはリタイアに分類されます。フロントエンドによる不適切な分岐予測、もしくは自己修正コードによるパイプライン・フラッシュなどのクリアイベントによってリタイアしない場合、そのスロットは投機の問題に分類されます。これら 4 つの

カテゴリーが、トップダウン特性化のトップレベルを形成します。アプリケーションの特性を決定するため、それぞれのパイプライン・スロットはこれら 4 つのカテゴリーのいずれか 1 つに分類されます。



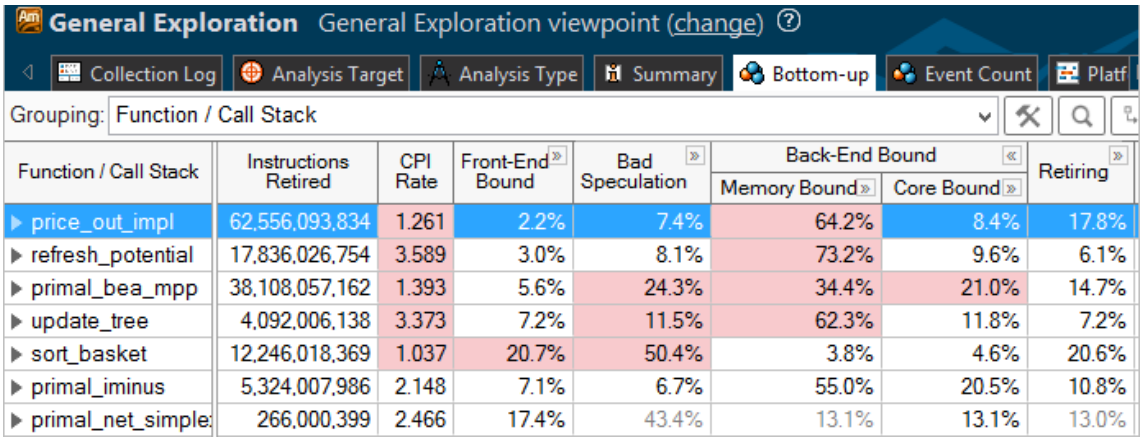
これら 4 つのカテゴリーにおけるパイプライン・スロットの分布は非常に有用です。イベントベースのメトリックは長年使われてきましたが、この特性化以前はパフォーマンスに最も影響を与える問題を識別する手法はありませんでした。このフレームワークにパフォーマンス・メトリックが配置されると、最初に取り組むべき問題が明らかになります。4 つのカテゴリーに分類されたパイプライン・スロットのイベントは、インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge (第 2 世代インテル® Core™ プロセッサー・ファミリーとインテル® Xeon® プロセッサー E5 ファミリー) から適用されます。以降のマイクロアーキテクチャーでは、これらハイレベルのカテゴリー分割はより細かなパフォーマンス・メトリックになる可能性があります。

インテル® VTune™ Amplifier XE によるトップダウン解析法

インテル® VTune™ Amplifier XE 製品は、General Exploration analysis (一般解析) タイプ (英語) を提供し、これにはインテル® マイクロアーキテクチャー開発コード名 Ivy Bridge 以降のトップダウン特性化で定義されたイベント収集のための事前定義が含まれます。一般解析はまた、その他多数の

有用なパフォーマンス・メトリックの計算に必要なイベントを収集します。一般解析の結果は、デフォルトでは [General Exploration viewpoint (一般解析ビューポイント)] (英語) に表示されます。

一般解析の結果は、トップダウン特性化の特徴を補足する階層的な列で表示されます。[Summary (サマリー)] ウィンドウ (英語) は、アプリケーション全体のそれぞれのカテゴリーにパイプライン・スロットの比率を表示します。結果はさまざまな方法で調査できます。最も一般的な方法は、関数レベルで表示されるメトリックを調査することです。



Function / Call Stack	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound		Retiring
					Memory Bound	Core Bound	
price_out_impl	62,556,093,834	1.261	2.2%	7.4%	64.2%	8.4%	17.8%
refresh_potential	17,836,026,754	3.589	3.0%	8.1%	73.2%	9.6%	6.1%
primal_bea_mpp	38,108,057,162	1.393	5.6%	24.3%	34.4%	21.0%	14.7%
update_tree	4,092,006,138	3.373	7.2%	11.5%	62.3%	11.8%	7.2%
sort_basket	12,246,018,369	1.037	20.7%	50.4%	3.8%	4.6%	20.6%
primal_iminus	5,324,007,986	2.148	7.1%	6.7%	55.0%	20.5%	10.8%
primal_net_simple	266,000,399	2.466	17.4%	43.4%	13.1%	13.1%	13.0%

それぞれの関数で、各カテゴリーのパイプライン・スロットの断片が表示されます。例えば、上記で選択されている関数 price_out_impl は、2.2% がフロントエンド依存カテゴリー、7.4% が Bad Speculation (投機の問題)、64.2% が Memory Bound (メモリー依存)、8.4% が Core Bound (コア依存)、そして 17.8% が Retiring (リタイア) カテゴリーのパイプライン・スロットであることが分かります。各カテゴリーは、そのカテゴリー配下のメトリックを展開表示できます。自動ハイライト機能により、開発者の注意を促すため潜在的な問題領域が強調されます。ここでは、price_out_impl のメモリー依存のパイプライン・スロットの比率がハイライトされています。

マイクロアーキテクチャー・チューニングの方法論

パフォーマンス・チューニングを行う際には、常にアプリケーションの上位のホットスポットに注目します。*Hotspots* (ホットスポット) は、最も CPU 時間を消費する関数です。これらのスポットに注目し、アプリケーションのパフォーマンス全体に影響する最適化を特定します。インテル® VTune™ Amplifier XE には、これに役立つ Basic Hotspots (基本ホットスポット) と Advanced Hotspots (高度なホットスポット) という 2 つの解析タイプがあります。一般解析ビューポイント内

のホットスポットは、CPU クロックティック数を測定したクロックティック・イベントの値が最も高い関数やモジュールを特定することで見つけられます。マイクロアーキテクチャーのチューニングから最大限の利益を得るため、並列処理を加えるなどのアルゴリズムの最適化がすでに適用されていることを確認します。一般にシステムのチューニングが最初に行われ、その後アプリケーション・レベルのアルゴリズムのチューニング、アーキテクチャーとマイクロアーキテクチャーのチューニングへと続きます。この手順は、トップダウンのソフトウェア・チューニング方法論と同様に、"トップダウン" と呼ばれます。ワークロードの選択などその他の重要なパフォーマンス・チューニングの要因については、「謎めいたソフトウェア・パフォーマンスの最適化を紐解く」(英語) の記事で説明されています。

1. ホットスポットとなる関数 (アプリケーションの合計クロックティックの大半を占める) を特定します。
2. トップダウン法と以下に示すガイドラインを使用してホットスポットの効率を評価します。
3. 非効率であれば、最も重要なボトルネックを示すカテゴリーをドリルダウンして (掘り下げて)、原因を特定するためボトルネックのサブレベルを調べます。
4. 問題を最適化します。インテル® VTune™ Amplifier XE のチューニング・ガイド (英語) には、各カテゴリーの多くのパフォーマンスの問題に対するチューニングの推奨事項が含まれます。
5. 上位のホットスポットをすべて評価するまで上記のステップを繰り返します。

インテル® VTune™ Amplifier XE は、ホットスポットが事前定義されたしきい値の範囲を超えている場合、GUI 上のメトリック値を自動的にハイライト表示します。インテル® VTune™ Amplifier XE は、アプリケーション内で取得された総クロックティックの 5% 以上である関数をホットスポットとして分類します。パイプライン・スロットが特定のカテゴリーのボトルネックを構成するかどうかの判断はワークロードに関係していますが、以下の表に一般的なガイドラインを示します。

	各カテゴリーのパイプライン・スロットの期待される範囲 (うまく最適化されたホットスポット向け)		
カテゴリー	**クライアント/ デスクトップ向け アプリケーション**	**サーバー/ データベース/ 分散型アプリケーション**	**ハイパフォーマンス・ コンピューティング (HPC) アプリケーション**
リタイア	20-50%	10-30%	30-70%
バックエンド依存	20-40%	20-60%	20-40%
フロントエンド依存	5-10%	10-25%	5-10%
投機の問題	5-10%	5-10%	1-5%

これらのしきい値は、インテル社内でいくつかのワークロードを解析した結果に基づいています。ホットスポットのカテゴリー (リタイアを除く) に費やされた時間が上位にあるか、上記の範囲よりも大きい場合、調査が有効である可能性があります。1 つ以上のカテゴリーにこれが当てはまる場合、最も多い時間のカテゴリーを最初に調査すべきです。ホットスポットが各カテゴリーで時間を費やしていても、値が通常の範囲内であれば、問題にならない可能性があります。

トップダウン法を実施する際に重要なことは、ボトルネックではないカテゴリーに最適化の時間を費やす必要はないということです。例え行ったとしても、それは顕著なパフォーマンス向上にはつながりません。

バックエンド依存カテゴリーをチューニング

チューニングされていないアプリケーションのほとんどは、バックエンド依存です。バックエンドの問題を解決するのは、多くの場合、必要以上にリタイアに時間のかかるレイテンシーの原因を解決することです。インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge 向けに、インテル® VTune™ Amplifier XE は、高いレイテンシーの原因を検出するためのバックエンド依存メトリックを用意しています。例えば、LLC Miss (ラスト・レベル・キャッシュ・ミス) メトリックは、データを取得するため DRAM をアクセスする必要があるコード領域を特定し、Split Loads and Split Stores (分割ロードと分割ストア) メトリックは、パフォーマンスに影響する複数キャッシュ間のメモリー・アクセス・パターンを指摘します。インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge のメトリックの詳細については、チューニング・ガイドをご覧ください。インテル® マイクロアーキテクチャー開発コード名 Ivy Bridge (第 3 世代インテル® Core™ プロセッサー・ファミリー) 以降では、バックエンド依存に分類されるイベントが、メモリー依存とコア依存のサブカテゴリーに区分されています。上位 4 つのカテゴリーに属するメトリックは、パイプライン・スロット・ドメイン以外のドメインを使用する可能性があります。各メトリックは、PMU イベントの最も適切なドメインを使用します。詳細については、各メトリックまたはカテゴリーのドキュメント (英語) をご覧ください。

メモリー依存とコア依存のサブカテゴリーは、実行ユニットの使用率に対応するイベントを使用して決定されます。これは、トップレベルの分類で使用されるアロケーション・ステージとは対照的です。したがって、これらのメトリックの合計は、必ずしもトップレベルで決定されたバックエンド比率と一致するわけではありません (関連性は高い)。

メモリー依存カテゴリーのストールの原因は、メモリー・サブシステムに関連します。例えば、キャッシュミスとメモリーアクセスは、メモリー依存のストールの原因となります。コア依存のストールは、各サイクルで CPU の実行ユニットが十分に利用されていないことが原因で発生します。例えば、連続する複数の除算命令は、除算ユニットの競合を引き起こしコア依存のストールの原因となります。この分類では、ストールで未完のメモリーアクセスがない場合、そのスロットはコア依存に分類されます。例えば、遅延されているロードがある場合、ロードがまだデータを取得していないことが原因で実行ユニットは待機しているため、サイクルはメモリー依存に分類されます。PMU イベントは、アプリケーションの真のボトルネックを特定するのに役立つブレークダウンを可能にするため、ハードウェアで設計されています。バックエンドの問題のほとんどは、メモリー依存カテゴリーに区分されるでしょう。

メモリー依存カテゴリーのほとんどのメトリックは、L1 からメモリーがボトルネックであるメモリー階層のレベルを特定します。この決定に利用されるイベントは注意深く設計されています。いったんバックエンドがストールすると、メトリックは遅延中の特定のキャッシュレベルへのロード、または実行中のストアのストールを区分しようとします。ホットスポットが特定のレベルで制限されている場合、そのデータの多くはキャッシュやメモリー階層から取得されていることを意味します。最適化では、コアに近い位置にデータを移動することに注目します。Store Bound (ストア依存) は、パイプラインを進行中のロードが直前のストアに依存しているなどの依存性を示すサブカテゴリーとしても利用されます。各カテゴリーには、メモリー依存実行の原因となるアプリケーション固有の動作を特定するメトリックがあります。例えば、Store Forwarding (ストアフォワード) でブロックされたロードと 4K Aliasing (4K エリアス) は、アプリケーションが L1 Bound (L1 依存) となる動作を示すメトリックです。

コア依存のストールは、バックエンド依存ではそれほど多くはありません。これは、計算リソースが効率良く利用されておらず、そして/また、有効なメモリー要求がない場合に発生します。例えば、浮動小数点 (FP) 演算を小さなループで行い、データがキャッシュに収まりきる場合が当てはまります。インテル® VTune™ Amplifier XE は、このカテゴリーの動作を検知するいくつかのメトリックを用意しています。例えば、Divider (除算器) メトリックは、除算器ハードウェアが頻繁に使用されたサイクルを特定し、Port Utilization (ポート使用率) メトリックは実行ユニットの散発的な競合を特定します。

Function / Call Stack	Back-End Bound						
	Memory Bound					Core Bound	
	L1 Bound	L2 Bound	L3 Bound	DRAM Bound	Store Bound	Divider	Port Utilization
price_out_impl	4.7%	4.7%	4.4%	44.9%	0.0%	0.0%	7.7%
refresh_potential	1.7%	0.0%	4.6%	75.9%	0.0%	0.0%	10.6%
primal_bea_mpp	0.0%	1.1%	11.7%	26.0%	0.0%	0.0%	23.6%
update_tree	33.3%	16.5%	22.9%	4.1%	0.0%	0.0%	14.6%
sort_basket	11.4%	0.0%	0.0%	0.0%	0.0%	0.0%	13.8%

注

灰色で反転表示されるメトリック値は、このメトリックで収集されたデータの信頼性が低いことを表します。これは例えば、収集された PMU イベントのサンプル数が非常に少ないことが考えられます。このデータは無視できますが、収集に戻ってデータ収集時間、サンプリングの間隔、またはワークロードを増やして再度収集することもできます。

フロントエンド依存カテゴリーをチューニング

フロントエンド依存カテゴリーは、パイプライン・ストール・タイプのほかの問題をいくつかカバーします。パイプラインのフロントエンド部分がアプリケーションのボトルネックとなることは、それほど多くはありません。例えば、JIT コードとインタープリターで解釈されるコードは、命令ストリームが動的に生成されるため (コンパイラーによるコード配置の利点が得られない)、フロントエンド・ストールの原因となります。フロントエンド依存カテゴリーのパフォーマンスを改善するには、通常コード配置 (ホットなコードと隣接して配置) とコンパイラー手法に関連します。例えば、分岐の多いコードや大きなフットプリントのコードは、フロントエンド依存カテゴリーで警告されるでしょう。コードサイズの最適化やコンパイラーによるプロファイルに基づく最適化 (PGO) などの手法は、多くの場合ストールを軽減するのに有効です。

インテル® マイクロアーキテクチャー開発コード名 Ivy Bridge 以降のトップダウン法では、フロントエンド依存のストールがフロントエンド・レイテンシーとフロントエンド帯域幅の 2 つのカテゴリーに分離されました。フロントエンド・レイテンシーのメトリックは、バックエンドの準備ができているにもかかわらず、フロントエンドによって µop が発行されないサイクルを報告します。フロントエンド・クラスターは、サイクルあたり最大 4 µop を発行できることを思い出してください。フロントエンド帯域幅のメトリックは、4 µop 未満が発行されたサイクルを報告します。これは、フロントエンドの能力を使い切っていないことを意味します。各カテゴリー下にさらにメトリックがあります。

分岐予測ミスは、多くの場合、投機の問題カテゴリーに分類されますが、インテル® マイクロアーキテクチャー開発コード名 Ivy Bridge 以降では、フロントエンドに属する Branch Resteer (分岐方向の変更) のボトルネックのメトリックによって、フロントエンドが非効率となったことが示されます。

Function / Call Stack	Front-End Bound							
	Front-End Latency						Front-End Bandwidth	
	ICache Misses	ITLB ...	Branch Resteers	DSB ...	Length...	MS ...	Front-End Band...	Front-End ...
price_out_impl	0.0%	0.0%	0.9%	0.0%	0.0%	0.1%	3.8%	0.0%
refresh_potential	0.0%	0.0%	1.4%	0.2%	0.0%	0.1%	5.7%	0.1%
primal_bea_mpp	0.0%	0.0%	2.9%	0.1%	0.0%	0.0%	12.2%	0.1%
update_tree	0.0%	0.0%	2.2%	0.6%	0.0%	0.0%	7.2%	1.4%
sort_basket	0.0%	0.0%	11.4%	2.1%	0.0%	0.0%	31.4%	0.6%
primal_iminus	0.0%	0.0%	1.0%	0.7%	0.0%	0.0%	9.0%	0.7%

インテル® VTune™ Amplifier XE は、フロントエンド依存の原因を特定するメトリックの一覧を表示します。これらのカテゴリーのいずれかの結果が顕著であるなら、メトリックをさらに深く調査して原因を特定し、修正する方法を考えます。例えば、ITLB Overhead (命令トランスレーション・ルックアサイド・バッファーのオーバーヘッド) と ICache Miss (命令キャッシュミス) のメトリックは、フロントエンド依存の実行で問題のある領域が分かります。チューニングの推奨事項に関しては、インテル® VTune™ Amplifier XE のチューニングガイドをご覧ください。

投機の問題カテゴリーをチューニング

3 番目のトップレベルのカテゴリーは、投機の問題であり、これはパイプラインがリタイアしない命令のフェッチと実行によりビジーであることを示します。投機の問題のパイプライン・スロットは、リタイアしないまたはマシンが不適切な投機実行から回復する間ストールする µop が発行されることでスロットが浪費されている状態です。投機の問題は、分岐予測ミスとマシンクリアおよび、一般的ではありませんが自己修正コードによって引き起こされます。投機の問題は、プロファイルに基づく最適化 (PGO)、間接分岐の回避、およびマシンクリアを引き起こすエラー条件の排除などのコンパイラー手法により軽減できます。投機の問題を収集することは、フロントエンド依存のストール数を減らすのに役立ちます。特定のチューニング手法については、適切なマイクロアーキテクチャー向けのインテル® VTune™ Amplifier XE のチューニング・ガイドを参照してください。

Function / Call Stack	Bad Speculation		Back-End Bound
	Branch Mispredict	Machine Clears	
price_out_impl	7.4%	0.0%	72.7%
refresh_potential	8.0%	0.1%	82.8%
primal_bea_mpp	24.3%	0.0%	55.5%
update_tree	11.5%	0.0%	74.1%
sort_basket	50.4%	0.0%	8.4%
primal_iminus	6.7%	0.0%	75.4%
primal_net_simpl	0.0%	43.4%	26.1%

リタイアカテゴリーのチューニング

トップレベルの最後のカテゴリーは、リタイアです。これは、パイプラインが通常動作の実行でビジーであることを意味します。可能であれば、このカテゴリーにアプリケーションの多くのスロットが分類されるのが理想的です。しかし、パイプライン・スロットでリタイアする大部分がコード領域であれば、改善の余地はあります。リタイアカテゴリーに分類されるパフォーマンスの問題の

1 つは、マイクロシーケンサーによる高い利用率です。これは、特定の条件が記述された µop の長いストリームを生成することで、フロントエンドをアシストします。この場合、多くの µop がリタイアしますが、それらのいくつかは回避することが可能です。例えば、デノーマルイベントに適用される FP アシストの多くは、コンパイラー・オプション (DAZ や FTZ) によって軽減できます。コード生成の選択もまた、これらの問題を軽減するのに役立ちます。詳細は、インテル® VTune™ Amplifier XE のチューニング・ガイドをご覧ください。インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge では、アシストはリタイアカテゴリーのメトリックとして分類されます。インテル® マイクロアーキテクチャー開発コード名 Ivy Bridge 以降では、リタイアカテゴリーのパイプラインは、General Retirement (一般リタイア) と呼ばれるサブカテゴリーに分割され、マイクロコード・シーケンサーの µop は別に識別されます。

Function / Call Stack	Retiring				
	General Retirement				Microcode Sequencer
	FP Arithmetic			Other	Assists
	FP x87	FP Scalar	FP Vector		
price_out_impl	0.0%	0.0%	0.0%	100.0%	0.0%
refresh_potential	0.0%	0.0%	0.0%	100.0%	0.0%
primal_bea_mpp	0.0%	0.0%	0.0%	100.0%	0.0%
update_tree	0.0%	0.0%	0.0%	100.0%	0.0%
sort_basket	0.0%	0.0%	0.0%	100.0%	0.0%

まだ行っていないのであれば、アルゴリズムの並列化やベクトル化によるチューニングは、リタイアカテゴリーのコード領域のパフォーマンスを改善するのに役立ちます。

まとめ

トップダウン法とインテル® VTune™ Amplifier XE におけるその有効性は、PMU を使用したパフォーマンス・チューニングの新たな方向性を示しています。開発者がこの特性化を習得するための時間は価値あるものです。特性化のサポートは、ここ最近の PMU 向けに設計されており、その階層構造は将来のインテル® マイクロアーキテクチャー上にも拡張可能です。例えば特性化は、インテル® マイクロアーキテクチャー開発コード名 Sandy Bridge とインテル® マイクロアーキテクチャー開発コード名 Ivy Bridge 間でかなり拡張されました。

トップダウン法の目的は、アプリケーション・パフォーマンスのボトルネックの種類を特定することです。そして、インテル® VTune™ Amplifier XE の一般解析と可視化機能の目的は、アプリケー

ションを改善するために適用可能な情報を提供することです。これらを併用することで、アプリケーションのパフォーマンスを大幅に改善できるだけでなく、最適化における生産性を向上させます。

関連情報

- General Exploration (一般解析) データを解釈する (英語)
- チューニングガイドとパフォーマンス解析
- ベクトル化アドバイザー (英語)
- クロックティック vs. パイプライン・スロットベースのメトリック (英語)

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。