

Arm Mobile Studioを使用したMali GPUの解析の加速



[Peter Harris](#)

2019年3月20日

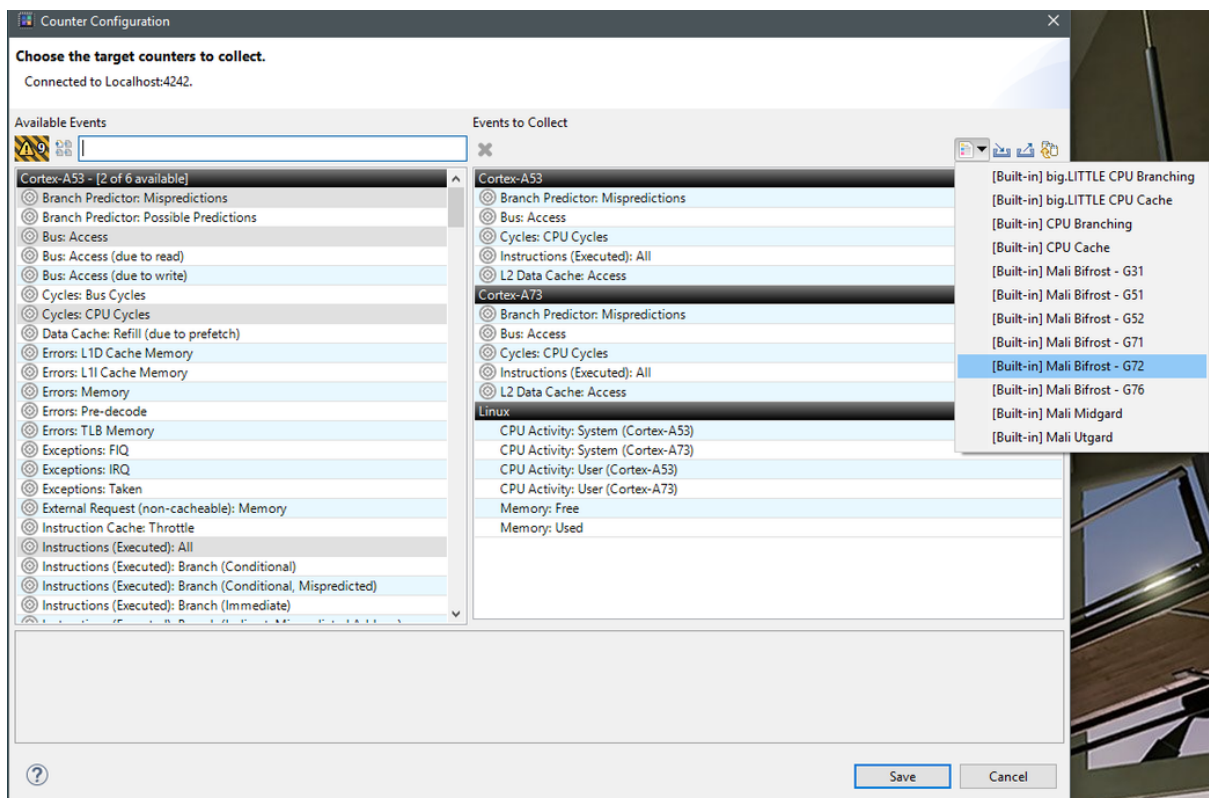
Streamlineパフォーマンスアナライザーは、サンプルベースのプロファイラーで、デバイスに存在するArm CPUとMali GPUに関する詳細なパフォーマンス情報を提示することができます。最近のバージョンのStreamlineには事前定義された一連のテンプレートが付属していて、使用するデータソースを簡単に選択でき、それらをどのように視覚化するかを制御できます。[Arm Mobile Studio](#) 2019.0および[Arm Development Studio](#) 2019.0に付属している最新バージョンのStreamlineには、Mali Bifrost GPUファミリ用のMali GPUテンプレートが含まれています。この記事では、Mali-G72 GPU用のテンプレートを使用して説明することにします。

このブログでは、読者がグラフィックスの用語、特にタイルベースのレンダリングGPUアーキテクチャに関連する用語について理解していることを前提としています。以下のクイックスタートガイドも参考にしてください。

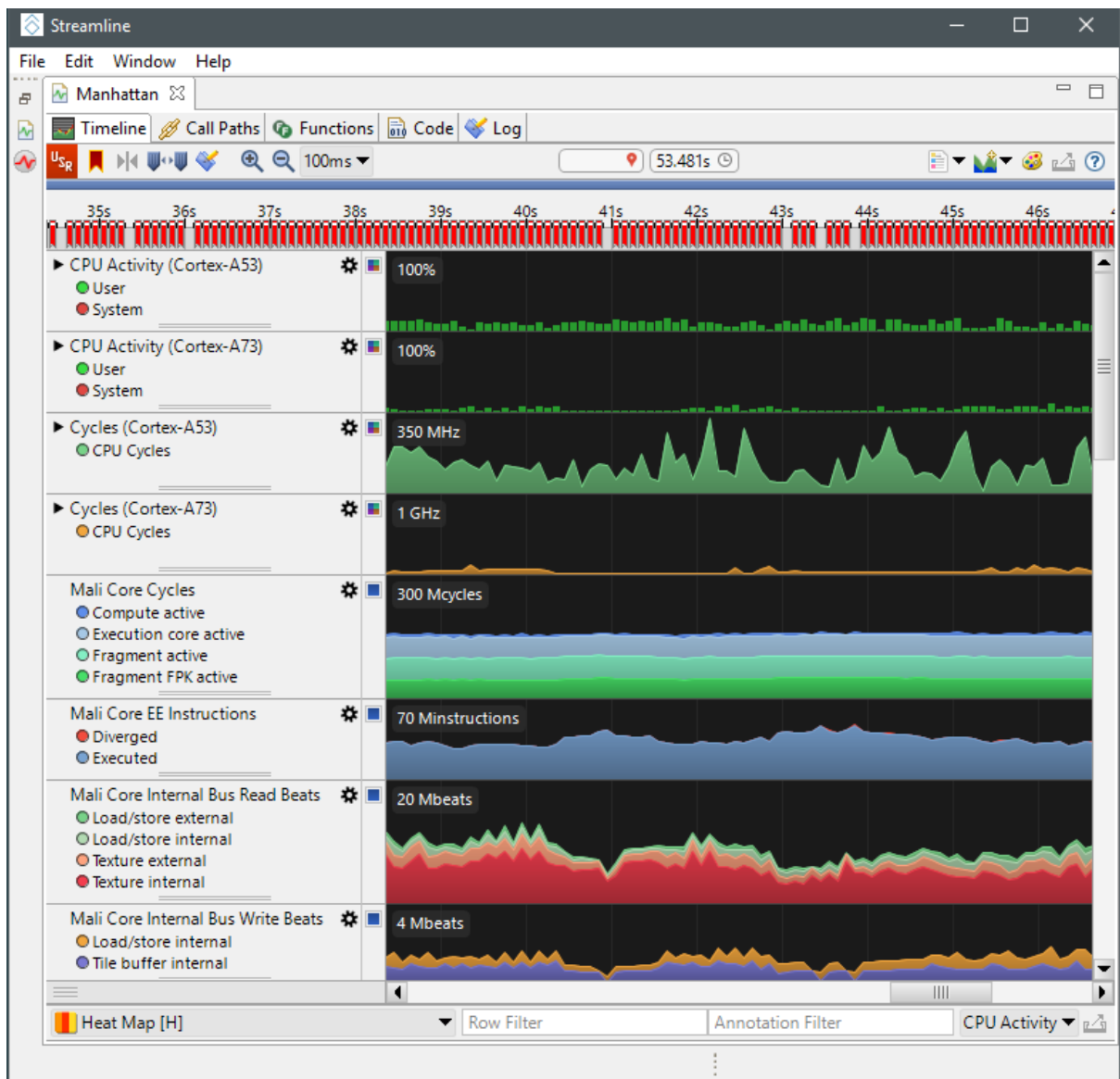
- [Understanding Render Passes \(レンダリングパスについて\)](#)
- [Understanding GPU Pipelining \(GPUパイプラインについて\)](#)
- [Understanding Tile-based Rendering \(タイルベースレンダリングについて\)](#)
- [Introduction to the Mali Bifrost Shader Core \(Mali Bifrostシェーダーコアの概要\)](#)

カウンターの選択

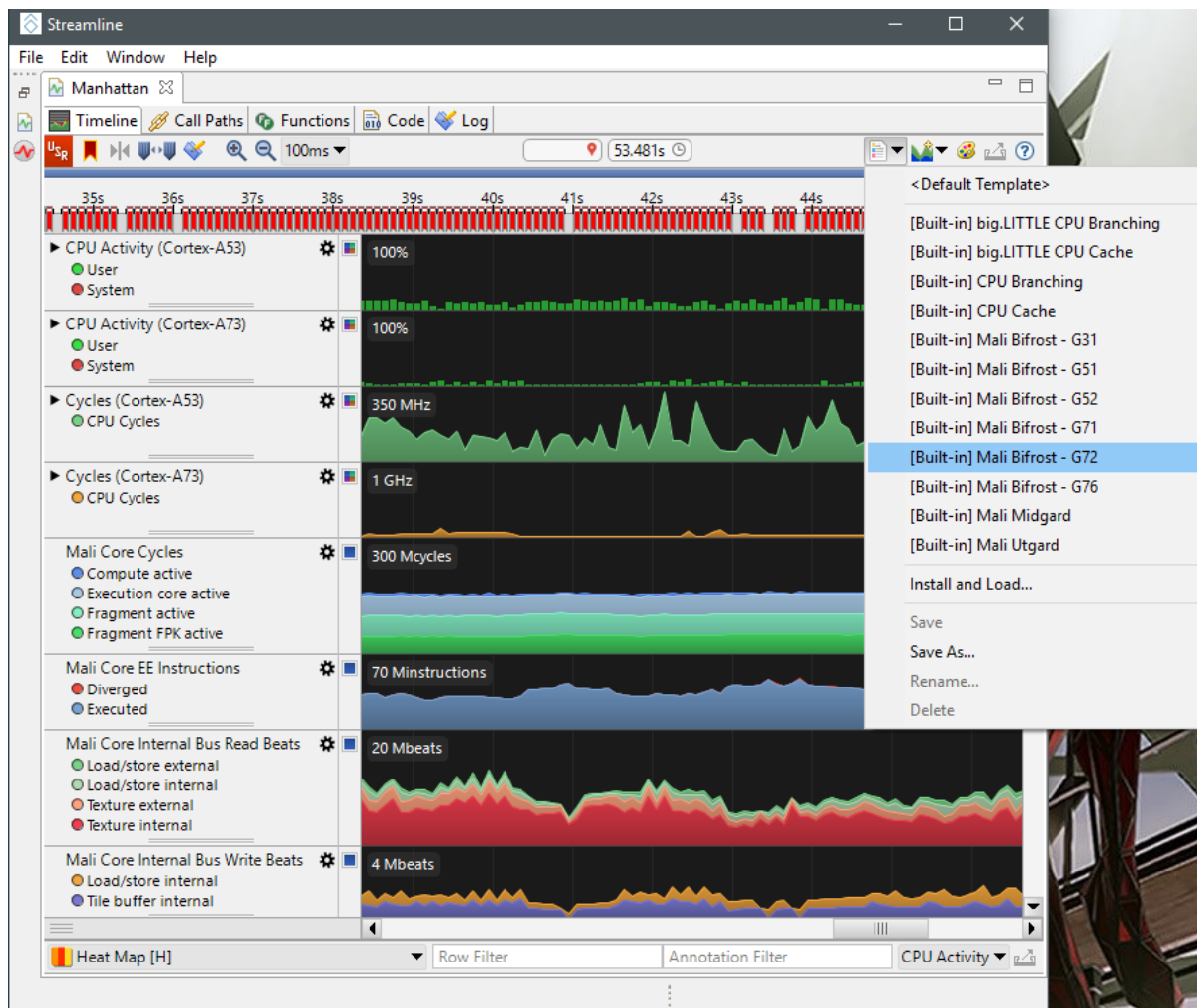
[Quick Start Guide](#)に従ってアプリケーションをセットアップしてgatorデーモンをターゲットにインストールしたら、データソースを選択してプロファイリングを開始します。デバイスを接続すると [Counter Configuration] ダイアログが表示されます。[Counter Configuration] ダイアログで、ドロップダウンメニューからデバイスに適切なテンプレートを選択します。



このように操作することで、そのテンプレートでの視覚化に必要なすべてのデータソースが自動的に選択されます。[Save] をクリックしてから、アプリケーションの追跡をキャプチャします。初回データ解析が完了すると、デフォルトのタイムラインビューが表示されます。



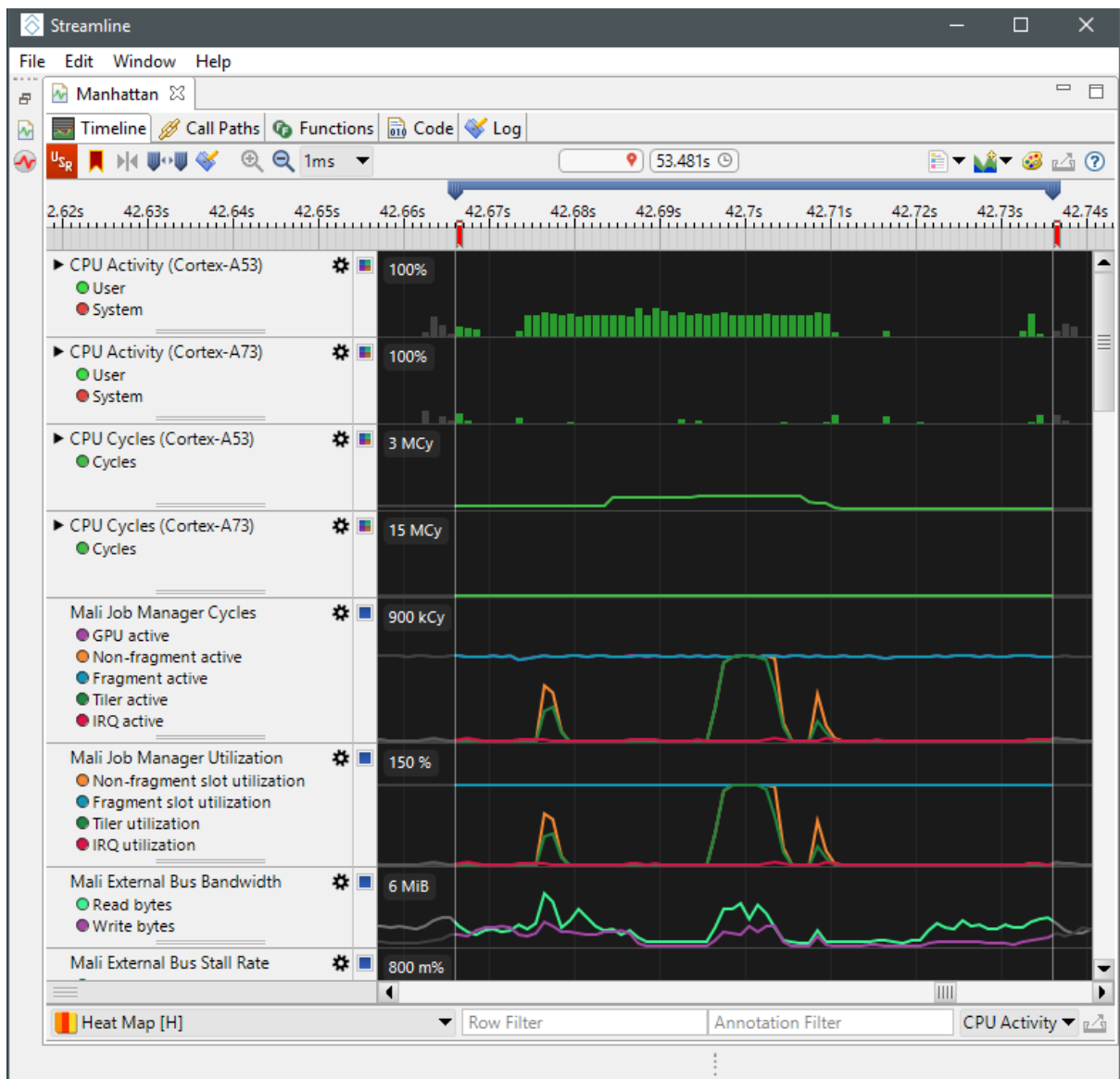
このビューには、キャプチャされたグラフとデータ系列の一覧がアルファベット順に表示されています。まず初めに、キャプチャの視覚化に使用したのと同じテンプレートを選択します。



これにより、タイムラインは、Armのパフォーマンス解析チームによってデザインされた表示に変わります。グラフはより系統だった順に並べ替えられます。複数のカウンターデータを数式を使って組み合わせることで、より読み取りやすいメトリック (機能ユニットの使用率など) となって提示されます。

フレームの特定

タイムラインが表示されている初期ビューでは、画面上のサンプルは1秒間隔で表示されていますが、処理が正しいかを確認するフレームの存続時間は通常16~32ミリ秒であるため、グラフィックスコンテンツをデバッグするには粗すぎます。したがって、解析の最初の手順は、個々のフレームを区別できるようになるまでビューを拡大することです。



この例で示しているアプリケーションには、アプリケーションが`eglSwapBuffers()` を呼び出すごとにStreamlineマーカーアノテーションを生成するための命令をソースコードに追加しています。上記のグラフでは、時間トラック上に赤色のマークとして表示されています。

個々のフレームを確認できるようになったら、システムの現在の挙動の初期評価を行うことができます。

- フレーム間の時間を測定して、達成されたフレームレートを割り出します。
- CPUスレッドの負荷を測定して、CPUバウンドであるかどうかを判断します。
- GPUスレッドの負荷を測定して、GPUバウンドであるかどうかを判断します。

- CPUワークロードとGPUワークロードのパイプラインを調べて、アプリケーションのロジックによってグラフィックスパイプラインにデータが最適に与えられているかどうかを判断します。

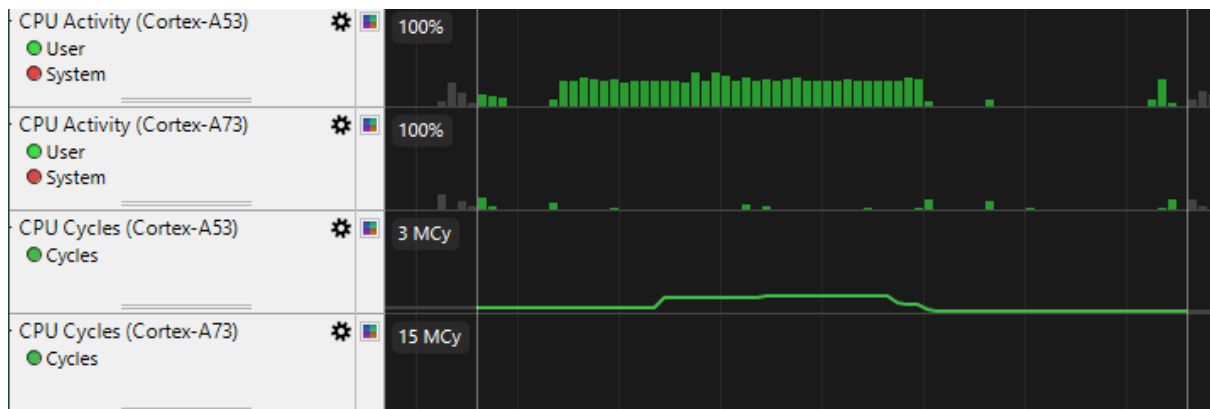
上記の例では、このフレームのかなりの割合で、すべてのCPUが完全にアイドル状態であることがわかるため、CPUバウンドではありません。また、GPUは常にアクティブであることもわかるため、このアプリケーションのパフォーマンスを制限しているプロセッサがGPUである可能性が高いことがわかります。

GPUワークロードをさらに細かく見ると、フラグメントシェーディングキューが常にアクティブである一方で、すべてのジオメトリと演算処理に使用される非フラグメントキューはそのフレームのほとんどでアイドル状態になっていることがわかります。したがって、このアプリケーションのパフォーマンスを改善するには、フラグメントのワークロードの最適化を目指すことになります。

このチュートリアル以降のセクションでは、テンプレートにある各グラフについて、その意味と、パフォーマンスの改善を目指すアプリケーション開発者は必要な変更内容をそこからどう読み取るべきかについて説明しています。

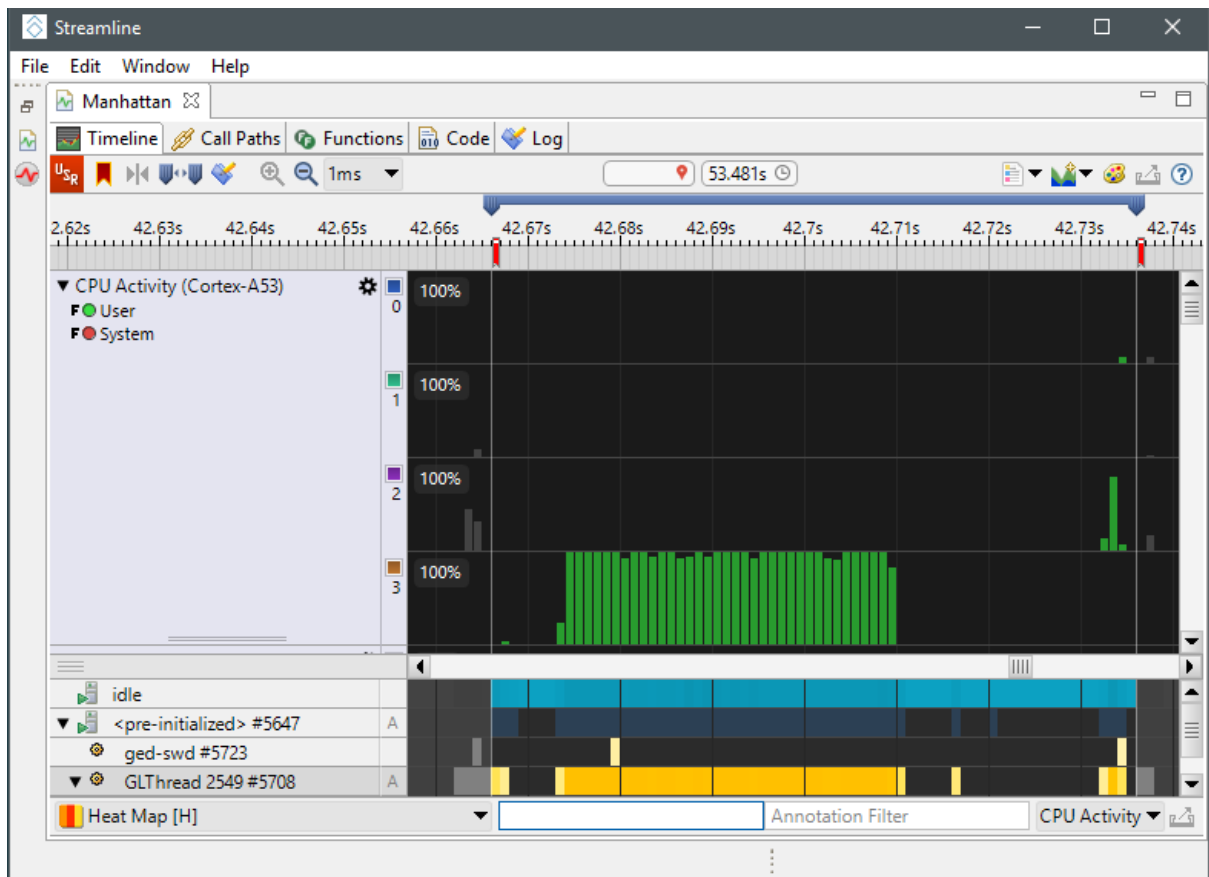
CPUワークロード

CPUのグラフは、システム内のCPUの全体的な使用状況を示しています。



CPU Activity グラフは、CPUがアクティブであった時間の割合として算出されたCPUごとの使用率を示していて、big.LITTLEクラスタリングが存在する場合はプロセッサタイプ別に分けられています。これは、OSのスケジューリングイベントデータに基づいています。*CPU Cycles* グラフは、CPUパフォーマンス モニタ ユニット (PMU) を使用して測定された、各CPUがアクティブであったサイクル数を示しています。これらの両方のグラフを合わせて考えることによって、アプリケーションソフトウェアの全体的な負荷を評価できます。CPUの使用率とサイクルカウント

がどちらも大きい場合は、そのCPUが高度のビジー状態であり、かつ高いクロック周波数で実行されていることを表しています。

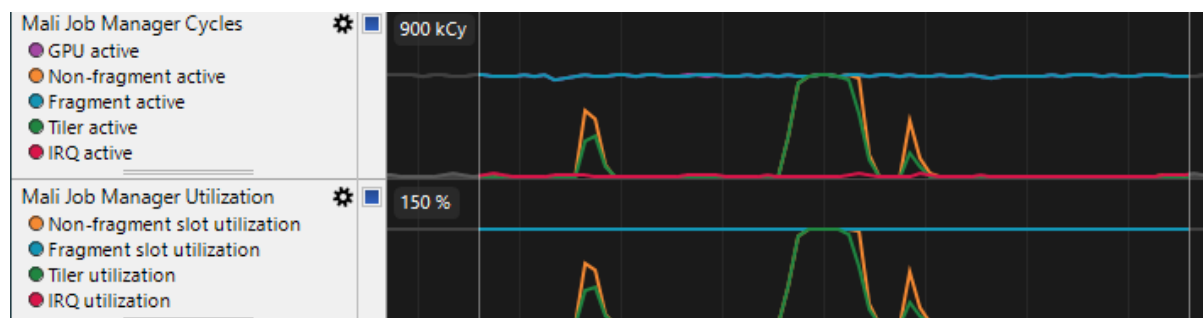


タイムラインタブの下部にあるプロセスビューにはアプリケーションのスレッドのアクティビティが示されているため、測定された負荷がどのスレッドで発生しているかを特定できます。その一覧から1つまたは複数のスレッドを選択すると、CPU関連のグラフがフィルタリングされて、選択したスレッドからの負荷のみが表示されます。スレッドレベルのフィルターがアクティブである場合は、グラフのタイトルの背景が青色で薄く色付けされて、測定された負荷の一部だけが表示されていることが示されます。

アプリケーションのパフォーマンス目標が達成されておらず、1つのCPUスレッドが常にアクティブである場合は、おそらくCPUバウンドです。フレーム時間を改善するには、そのスレッドのワークロードのコストを軽減するようにソフトウェアを最適化する必要があります。Streamlineでは、パフォーマンスカウンタービューだけでなく、プログラムカウンターのサンプリングによるネイティブソフトウェアプロファイリングも提供されています。ソフトウェアプロファイリングについてはこのチュートリアルを超えているため、詳細については『Streamlineユーザーガイド』を参照してください。

GPUワークロード

GPUワークロードのグラフは、GPUの全体的な使用状況を示しています。



*Mali Job Manager Cycles*グラフは、GPU全体と、非フラグメントとフラグメントのための2つの並列ハードウェア作業キューの両方で、作業の実行に費やしたGPUサイクル数を示しています。*Mali Job Manager Utilization*グラフは、GPUのアクティブサイクル数に対する割合として正規化された同じデータを示しています。

GPUバウンドのコンテンツの場合、最も支配的なワークキューは、それと並列実行されている他のキューとともに、常にアクティブである必要があります。GPUバウンドのアプリケーションで良好な並列処理が達成されていない場合は、レンダリングパイプラインを処理するAPIコール (`glFinish()` や `glReadPixels()` の同期使用など) を調べるか、またはVulkanの依存関係が限定的すぎて複数のレンダリングパスのオーバーラップ (フレーム間のオーバーラップも含む) が許可されていないかを確認します。

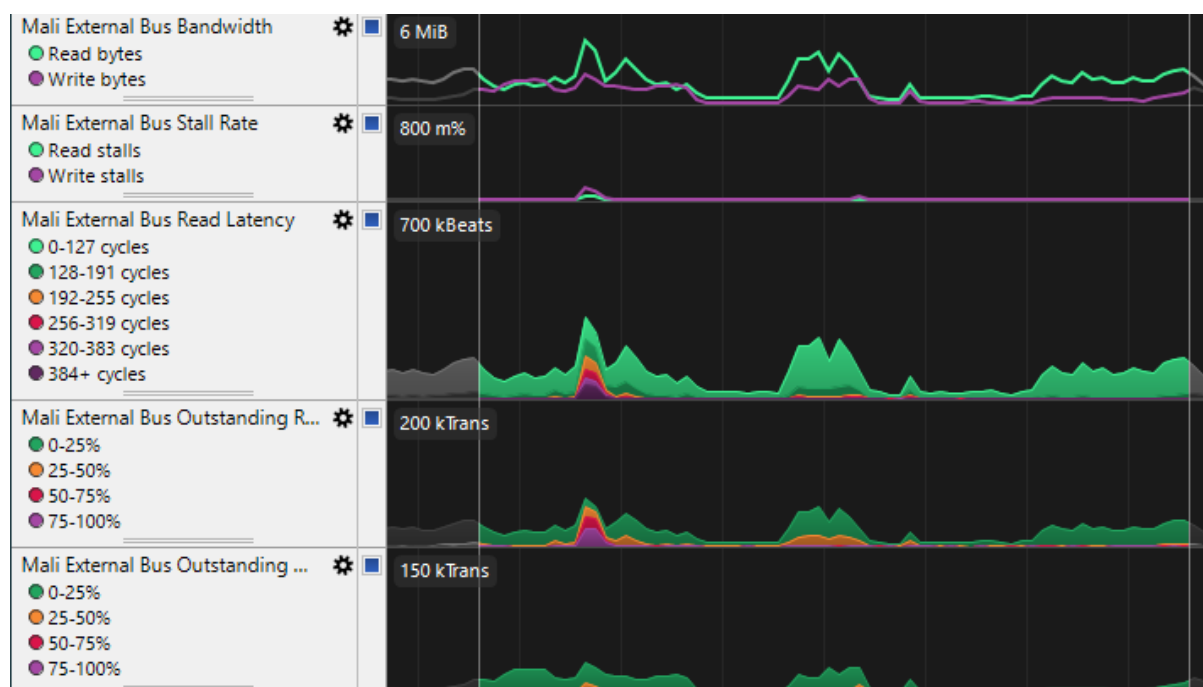
このグラフには *Tiler active* カウンターがありますが、タイラーは通常はジオメトリ処理の間ずっとアクティブであるため、直接的に役立つとは限りませんが、どれだけのコンピュータシェーディングが存在するかの目安になります。*Non-fragment active* と *Tiler active* の値が大きく離れている場合は、アプリケーションのコンピュータシェーダーが原因である可能性があります。

IRQ active カウンターは、CPUで保留中の割り込みがあるGPUのサイクル数を示しています。IRQ保留率がGPUサイクルの2%以内であれば正常ですが、アプリケーションが、多数の小さいレンダリングパスやコンピュータディスパッチをキューに入れることによって、割り込み率が高くなる場合があります。

注: IRQオーバーヘッドが大きい場合は、特権カーネルアクティビティによってCPU割り込みが長時間マスクされているなどの、システム統合の問題を暗示している可能性もあります。IRQオーバーヘッドが大きい場合、通常はアプリケーションを変更しても解決できません。

GPUメモリシステム

メモリシステムのグラフは、GPUによって生成されたメモリトラフィックと、システムがそのトラフィックをどの程度効率的に処理しているかの両方の観点から、GPUのメモリインターフェースでの挙動を示しています。



*Mali External Bus Bandwidth*グラフは、アプリケーションによって生成された読み出しと書き込みの総帯域幅を示しています。外部DDRメモリアクセスはエネルギーを非常に多く消費するため、メモリ帯域幅を小さくすることは、アプリケーション最適化の目標の達成に効果があります。その後のグラフは、アプリケーションのどのリソースタイプでトラフィックが発生しているかを特定するのに役立ちます。

*Mali External Bus Stall Rate*グラフは、バスストールが発生したGPUサイクルの割合を示していて、GPUが外部メモリシステムからどの程度の逆圧を受けているかを表しています。ストール率が5%以下であれば正常とみなすことができますが、ストール率がそれよりはるかに大きい場合は、そのワークロードは、メモリシステムの処理能力を超える量のトラフィックを生成していることを表しています。全体的なメモリ帯域幅を小さくすることや、アクセスの局所性を向上させることによって、ストール率を小さくすることができます。

*Mali External Bus Read Latency*グラフは、外部メモリアクセスの応答レイテンシーの積み上げヒストグラムを示しています。Mali GPUはGPUサイクル数が170以内の外部メモリレイテンシー に対応できるように設計されているため、低速なビンで

の読み出しの割合が高い場合は、メモリシステムのパフォーマンスの問題を表している可能性があります。DDRのパフォーマンスは一定ではなく、DDRが高負荷状態である場合はレイテンシーが大きくなるため、帯域幅を小さくすることは、レイテンシーを小さくするための効果的な方法である可能性があります。

注: DDRは共有リソースであり、システムの他のパーツからのトラフィックと競合するため、低速のビンで行われるメモリアクセスの割合は少ないと想定しています。

*Mali External Bus Outstanding Reads/Writes*グラフは、もう一組の積み上げヒストグラムを示していて、GPUがメモリシステムのキューに配置した許容メモリアクセスの割合を表しています。75~100%のビンでヒストグラムの割合が大きい場合は、そのGPUでトランザクションがなくなりつつある可能性があります。その場合、古い要求が使用されなくなるまで新しいメモリ要求はストールされます。メモリ帯域幅を小さくすることや、DDRでのアクセスの局所性を向上させることによって、パフォーマンスが改善される可能性があります。

GPUジオメトリ

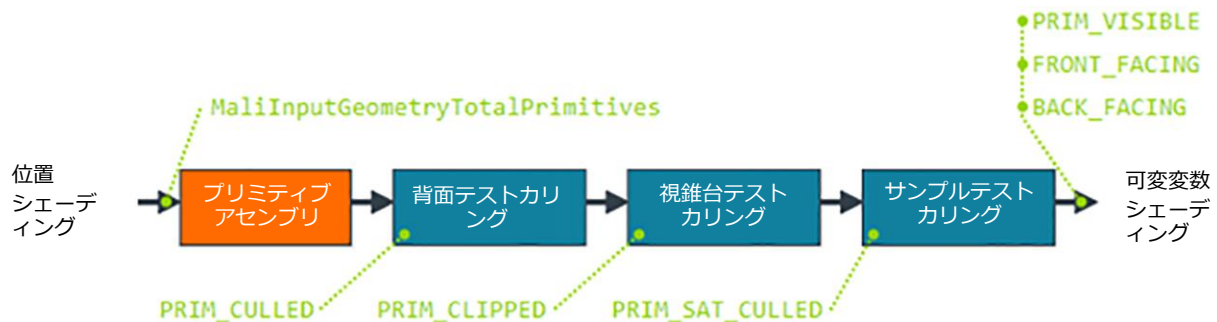
ジオメトリのグラフは、GPUによって処理されたジオメトリの量とプリミティブカリングユニットの挙動を示しています。



*Mali Primitive Culling*グラフは、処理中のプリミティブの絶対数、カリングの各ステージで除外されるプリミティブの数、および表示されているプリミティブの数を示しています。頂点の処理ではメモリ帯域幅の要件が大きいため、1つの頂点の処理は1つのフラグメントの処理に比べてはるかにコストが高くなります。そのため、フレームごとのプリミティブの総数を可能な限り少なくすることを目指す必要があります。

*Mali Primitive Culling Rate*グラフは、カリングの各ステージに入っていてそのステージで除外されるプリミティブの割合と、表示されているプリミティブの割合を

示しています。カリングパイプラインは、以下のようなひと続きのステージとして実行されます。



3Dシーンでは、プリミティブの50%以下が背面側であり、背面テストカリングユニットによって除外されると想定しています。*Culled by facing test*の割合がそれよりはるかに小さい場合は、背面テストが適切に有効化されているかどうかを確認します。

アプリケーションが視錐台外のドロークールをCPUでカリングすることは標準的なベストプラクティスであるため、*Culled by frustum test*率は可能な限り低く抑える必要があります。このステージで入力プリミティブの10%を超えて除外される場合は、CPU側カリングの有効性を見直します。また、オブジェクトの大規模なバッチによってカリングの効率が低くなることもあるため、バッチのサイズを見直すことで効果が出る可能性があります。

最終的なカリング率である *Culled by sample test*では、ラスタライズのサンプル点に達しないほど小さいために除外されるプリミティブの割合が測定されます。高密度のジオメトリは、直接の頂点処理のコストとフラグメントシェーディング効率の減少の両方の観点でコストが非常に高くなるため、この数値は可能な限り0%に近づけておく必要があります。ここで除外されるプリミティブの数が多い場合は、静的なメッシュ密度と、動的なlevel-of-detailの選択の有効性の両方を見直します。

*Mali Geometry Threads*グラフは、Maliのインデックス駆動型の頂点シェーディングアルゴリズムによって生成されるシェーディング要求の絶対数を示しています。この設計では、アプリケーション頂点シェーダーは、位置を計算するピースと他の可変変数を計算するピースの2つに分割されています。可変変数シェーダーは、クリッピングとカリングの後に残ったプリミティブに属する頂点に対してのみ実行されます。この時点で、以下のことを見直すことができます。

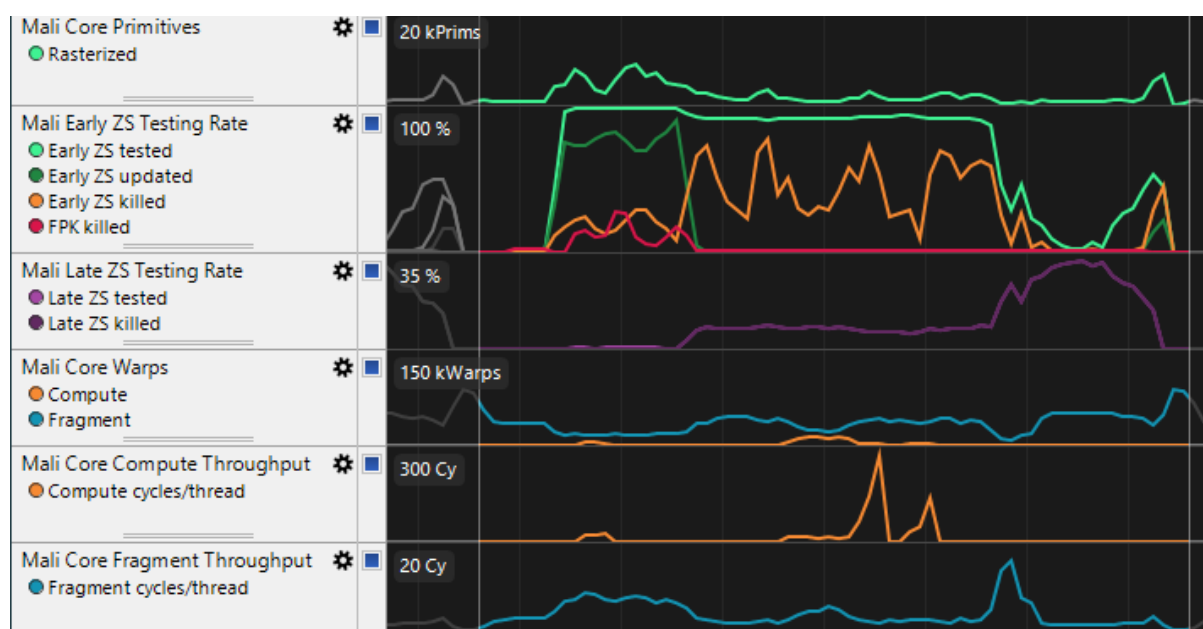
- 位置シェーダーの呼び出し総数を、アプリケーションのインデックスバッファ数と比較します。GPUがアプリケーションによって送信された数より多く

のインデックスをシェーディングしている場合は、インデックスの局所性が低いことを表していて、位置キャッシュのスラッシングや強制的な再シェーディングが行われることになります。

- 位置シェーダーの呼び出し総数を、入力プリミティブの総数と比較します。ほとんどのコンテンツでは、1つの頂点が隣接する複数のプリミティブで使用されて、可能な限り多くのコストが償却されるため、プリミティブあたりの頂点数の平均を1未満にすることを目指します。

GPUシェーダーフロントエンド

シェーダーフロントエンドのグラフは、プリミティブをフラグメントスレッドに変えて共有する固定機能ユニットの挙動を示しています。



Mali Core Primitives グラフは、ラスターライズのためにロードされているプリミティブの数を示しています。Maliは大きいプリミティブをタイルごとに1回ロードするため、このカウントには1つのプリミティブが、交差するタイルごとに1回含まれます。

Mali Early ZS Testing Rate グラフは、フロントエンドでの深度 (Z) とステンシル (S) のテストおよびカリングレートを示しています。早期ZSテストは後期ZSテストよりコストがはるかに低いため、シェーダーのdiscard、Alpha-to-coverage、およびシェーダーが生成する深度値の使用を最小限に抑えることによって、ほぼすべてのフラグメントで早期ZSテストが行われることを目指します。*FPK killed* カウンターは、MaliのForward Pixel Kill (FPK) 隠面消去スキームによって除外されたクワ

ツドの割合を示しています。FPKによって除外されるクワッドの割合が大きい場合は、後ろから前へのレンダリング順序を表しています。それを前から後ろへのレンダリング順序に置き換えることによって、早期ZSテストでクワッドが早めに除外されることになり、エネルギー消費が軽減されます。

*Mali Late ZS Testing Rate*グラフは、フラグメントシェーディングの後のバックエンドでの深度とステンシルのテストおよびカリングレートを示しています。後期ZSテストで除外されるクワッドの割合が大きい場合、それらのフラグメントはシェーディングされたあとに除外されるため、潜在的な効率の問題があることを表しています。

注: 開始状態としてクリアカラーではなく既存の深度またはステンシルのアタッチメントを使用するレンダリングパスでは、後期ZS操作はリロードプロセスの一部としてトリガーされます。これは回避できないこともありますが、すべてのアタッチメントをクリアせずに、開始されるレンダリングパスの数を最小限に抑えることを目指します。

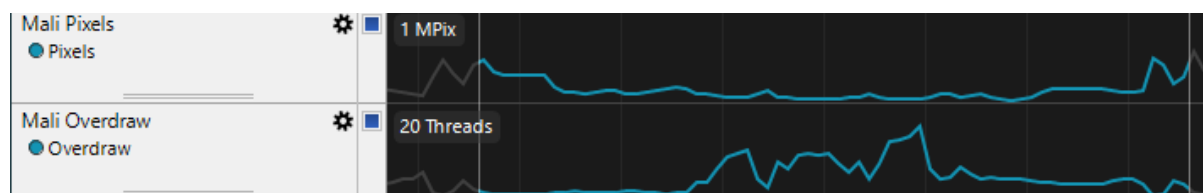
*Mali Core Warps*グラフは、コンピューティングバックエンド (すべての非フラグメントワークロードも含む) とフラグメントフロントエンドによって作成されたワーブの数を示しています。ワーブ幅は製品ごとに異なることがあります。

最後の2つのグラフは、スレッドあたりのシェーダーコアの処理コストの平均を示しています。GPUバウンドのコンテンツの場合は、シェーダーワークロードに対して次の2つの最適化目標が考えられます。

- シーンのコンテンツを簡素化してワーブの数を減らす
- シェーダープログラムを最適化してスレッドあたりのコストを削減する

GPUシェーダーフロントエンド ピクセル

この一組のグラフは、シェーダーコアがピクセルを生成しているレートを示しています。

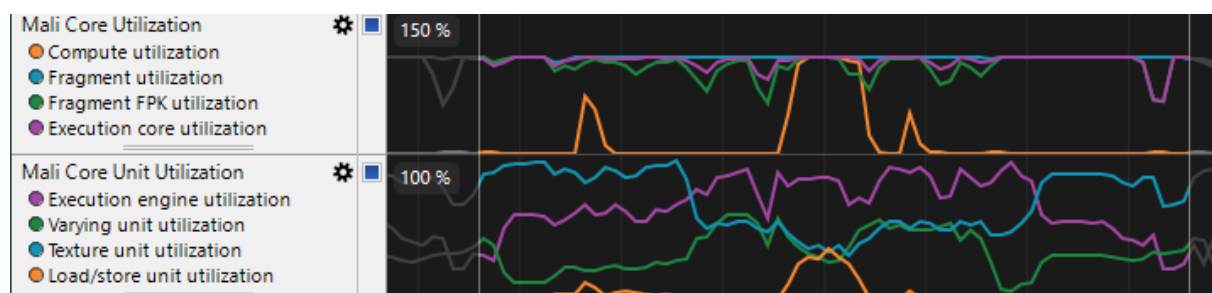


*Mali Pixels*グラフは、すべてのシェーダーコアでシェーディングされたピクセルの総数を示していて、1つのフレームの生成に必要なピクセルの総数を判断できます。

Mali Overdrawグラフは、出力ピクセルあたりの、シェーディングされたフラグメント数の平均を示しています。フラグメントあたりのコストは小さくても、オーバードローの値が大きいとパフォーマンスが低下します。使用する透過フラグメントのレイヤー数を最小限に抑えてオーバードローを減らすことを目指します。

GPUシェーダーコア

シェーダーコアはGPUの中核であるため、シェーダーコアのワークロードを調べることができる多数のカウンターがあるのは当然のことです。最初の一組のグラフでは、シェーダーコアの全体的な使用状況を一目で確認できるようになっています。



注: シェーダーコアの "compute" データパスはすべての非フラグメントワークロードの処理に使用されるため、コンピュート関連のカウンターには頂点シェーディングワークロードも含まれています。

Mali Core Utilizationグラフは、シェーダーコアの3つの主要パーツの使用率を示しています。

- *Compute utilization*系列と*Fragment utilization*系列は、そのタイプのワークロードがシェーダーコアで処理されていた時間の割合を示しています。ラスタライズやタイルライトバックなどの固定機能ロジックでの消費時間も含まれています。
- *Execution core utilization*系列は、プログラム可能なコア自体がアクティブであった時間の割合を示しています。この値が長時間にわたって100%未満である場合は、プログラム可能なコアに作業が供給されないという問題が発生している可能性があります。
- このグラフの*Fragment FPK utilization*系列は、クワッドがキューに入ってフラグメントスレッドに変わるのを待機している時間の割合を示しています。この値が長時間にわたって100%未満である場合は、シェーダーコアに対して新しいフラグメントが十分な速度で生成されていない可能性があります。その原因としては、プリミティブあたり少数のフラグメントを生成する

マイクロトライアングルが多数あることや、ジオメトリが一切含まれていない多数の空のタイルがあるワークロード (よくあるタイプのシャドウマップなど) が考えられます。

Mali Core Unit Utilization グラフは、実行コア内にある主要パイプラインの使用率を示しています。

- *Execution engine utilization* 系列は、シェーダーコアの算術演算ユニットがアクティブであった時間の割合を示しています。
- *Varying unit utilization* 系列は、固定機能の補間ユニットがアクティブであった時間の割合を示しています。
- *Texture unit utilization* 系列は、固定機能のテクスチャサンプリングとフィルタリングのユニットがアクティブであった時間の割合を示しています。
- *Load/store unit utilization* 系列は、汎用メモリアクセスユニットがアクティブであった時間の割合を示しています。

シェーダーコアバウンドであるシェーダーコンテンツの場合は、このグラフを使用して最も負荷が高いユニットを特定することが、どこを最適化の対象とするかを判断するのに効果的な方法です。

ワークロードのプロパティ

Mali Workload Properties グラフには、ワークロードの興味深い挙動を表しているさまざまなコンポーネントが含まれています。



Warp divergence rate 系列は、ワーブ間で制御フローが逸脱していて一部の実行レーンがマスクされているときに実行された命令の割合を示しています。制御フローが逸脱するとシェーダーの実行効率が急激に低下する可能性があるため、逸脱を最小限に抑えることを目指します。

Partial warp rate 系列は、カバレッジがないスレッドスロットが含まれているワーブの割合を示しています。この状況は、プリミティブのエッジと交差しているフラグメントクワッドが原因であり、ヒットサンプルがないフラグメントになります。部分的ワーブの割合が大きい場合は、アプリケーションに多数のマイクロトライアングルまたは非常に薄いトライアングルがあることを表している可能性があります。

す。部分的ワープがある場合もシェーダーの実行効率が急激に低下する可能性があるため、部分的ワープの数を最小限に抑えることを目指します。

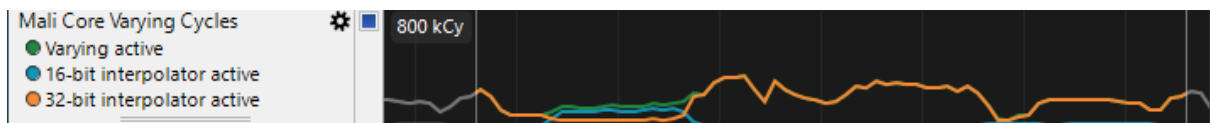
*Tile CRC kill rate*系列は、CRC一致が原因で除外されたタイルの割合を示しています。CRC一致は、算出されたカラーが、既にメモリにあるカラーと一致していることを表しています。除外率が大きい場合、アプリケーションで画面の変化した部分を特定してそこだけを描画できれば、最適化の機会を得られることを表しています。

算術演算ユニット

実行エンジンは、すべての算術演算ワークロードも含めて、すべてのシェーダー命令の実行に使用されます。前に説明した*Mali Core Unit Utilization*グラフにある*Execution engine utilization*系列は、アプリケーションで算術演算処理が制限されているかどうかを判断するために使用できます。

可変変数ユニット

*Mali Core Varying Cycles*グラフは、固定機能の可変変数インターポレーターの使用状況をデータ精度別に示しています。

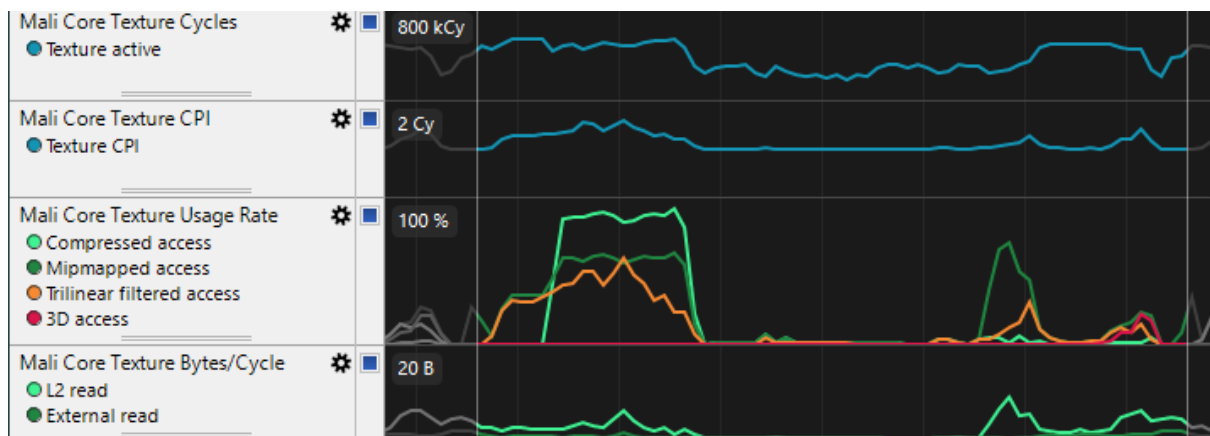


可変変数バウンドであるコンテンツの場合は、次の3つの最適化の機会が考えられます。

- ワークロードの直接的削減: フレームごとにロードする必要がある可変変数の全体的な数を減らします。
- 精度の簡素化: 32ビットのhighpから16ビットのmediump可変変数に切り替えることによって、インターポレーターの要件が半減し、シェーダーロジックで連鎖的に改善されることがよくあります。
- 可変変数のパック化: 16ビットの可変変数を複数の32ビットのベクトルにパックすることで、未使用のインターポレーターレーンのせいで損失されるサイクルが最小限に抑えられます。たとえば、パックされたvec4では、floatや個別のvec3より1サイクル早く補間されます。

テクスチャユニット

テクスチャユニットは、すべてのテクスチャサンプリングとフィルタリングを処理する複合ユニットであり、そのパフォーマンスは使用されるテクスチャ形式やフィルタリングモードに応じて変化する可能性があります。



*Mali Core Texture Cycles*グラフは、固定機能のテクスチャフィルタリングユニットの総使用量を示しています。

*Mali Core Texture CPI*グラフは、要求あたりの平均テクスチャサイクル数を示していて、より複雑なフィルタリングモードを使用することによってトリガーされる複数サイクル操作の数を把握できます。テクスチャバウンズのコンテンツの場合は、より単純なフィルタリングモードを使用してCPIを減らすことが、パフォーマンスを改善するための効果的な方法である可能性があります。

*Mali Core Texture Usage Rate*グラフは、行われたテクスチャアクセスのタイプに関する統計情報を示しています。

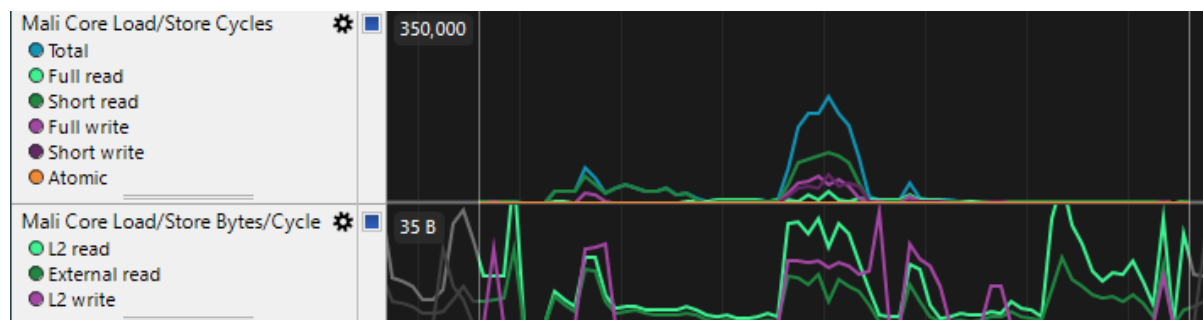
- *Compressed access*系列は、ASTCやETCなどのブロック圧縮テクスチャ形式を使用しているテクスチャアクセスの割合を示しています。ゲームレンダリングでは、帯域幅を減らすために可能な限りブロック圧縮テクスチャを使用する必要があります。
- *Mipmapped access*系列は、ミップマッピングテクスチャを使用しているテクスチャアクセスの割合を示しています。ゲームレンダリングでは、パフォーマンスと画像品質の両方を向上させるために、すべての3Dシーンでミップマッピングテクスチャを使用する必要があります。
- *Trilinear filtered access*系列は、三線形フィルタリングを使用しているテクスチャサンプルの割合を示しています。三線形アクセスは、双線形アクセスの半分の速度で実行されます。

- 3D access系列は、サンプルをボリュメトリックテクスチャにしているテクスチャサンプルの割合を示しています。ボリュメトリックアクセスは、2Dアクセステクスチャの半分の速度で実行されます。

Mali Core Texture Bytes/Cycleグラフは、フィルタリングの各サイクルでL2キャッシュおよび外部メモリからのフェッチが必要なバイト数を示しています。外部アクセスは特にエネルギー消費が多いため、圧縮テクスチャとミップマッピングを使用することをお勧めします。そうすることで、サンプルの良好な局所性も確保されキャッシュの圧力が軽減されます。

ロード/ストアユニット

ロード/ストアユニットでは、データの読み書きの汎用メモリアクセスだけでなく、イメージアクセスやアトミックアクセスも提供されます。

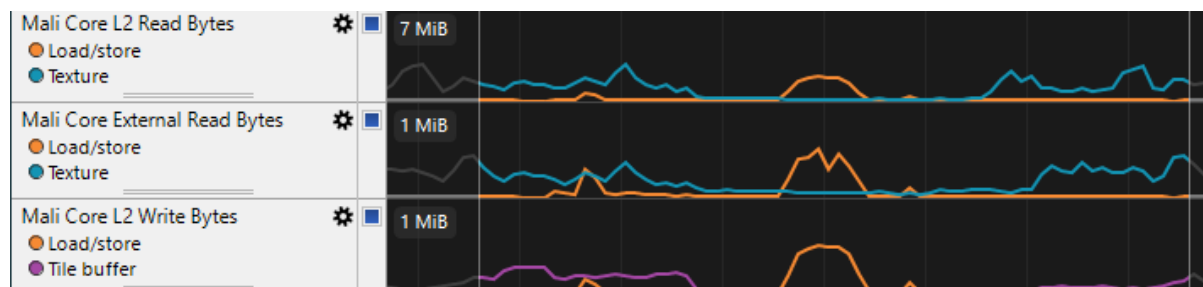


Mali Core Load/Store Cyclesグラフは、ロード/ストアのキャッシュに対して行われたアクセスタイプを示しています。読み出し数と書き込み数は "full" か "short" のいずれかです。"short" アクセスでは利用可能なデータバス幅の一部だけが使用されます。メモリアクセスをベクトル化して "short" アクセスの数を減らし、コンピュートシェーダーで空間的に隣接するスレッドにある隣接データにアクセスすることによって、パフォーマンスが改善されることがあります。

Mali Core Load/Store Bytes/Cycleグラフは、読み出しあたりの、L2キャッシュおよび外部メモリからのフェッチが必要なバイト数と、書き込みあたりの書き込みバイト数を示しています。使用されているアルゴリズムの知識を持たずにこのカウンターを解釈するのは難しいのですが、メモリアクセスとデータ使用のパターンがわかっている場合は、コンピュートシェーダーのパフォーマンスを調べるのに便利です。たとえば、ほとんどのロードがL2キャッシュからではなく外部メモリから行われているコンテンツは、そのワーキングセットが大きすぎることを表していることがあります。

メモリ帯域幅

最後の一組のグラフは、シェーダーコアによって生成されたメモリ帯域幅を、トラフィックを生成したユニット別に示しています。



全体的な帯域幅の問題が発生しているコンテンツの場合は、どのデータリソースで大半のトラフィックが生成されているのかを特定するのに、これらのカウンターが役に立ちます。

- ロード/ストアユニットのトラフィックは、すべてのタイプのバッファアクセスと、`image()` アクセサによるデータへのアクセスと関連しています。
- テクスチャユニットのトラフィックは、すべてのタイプのシェーダーの `texture()` アクセスと関連していて、アタッチメントがクリアされていないか無効である場合に、レンダリングパスの開始時にタイルバッファのコンテンツの復元に必要な暗黙のロードも含まれています。
- タイルバッファのトラフィックは、レンダリングパスの終了時の、すべてのフレームバッファアタッチメントのメモリへの書き戻しと関連しています。

まとめ

この記事では、Arm CPUとMali GPUを使用しているグラフィックスアプリケーションの初回のパフォーマンスレビューを行い、パフォーマンスカウンターの情報を使用して主要なワークロードおよびパフォーマンス低下の考えられる原因を特定する方法を示しています。Streamlineに組み込まれている事前定義テンプレートを使用して、系統だったレビューワークフローに必要なカウンターを迅速かつ効率的にキャプチャできるため、設計でのさまざまなブロックと挙動に対して段階的にレビューを実施できるようになります。

[Arm Mobile Studio](#)