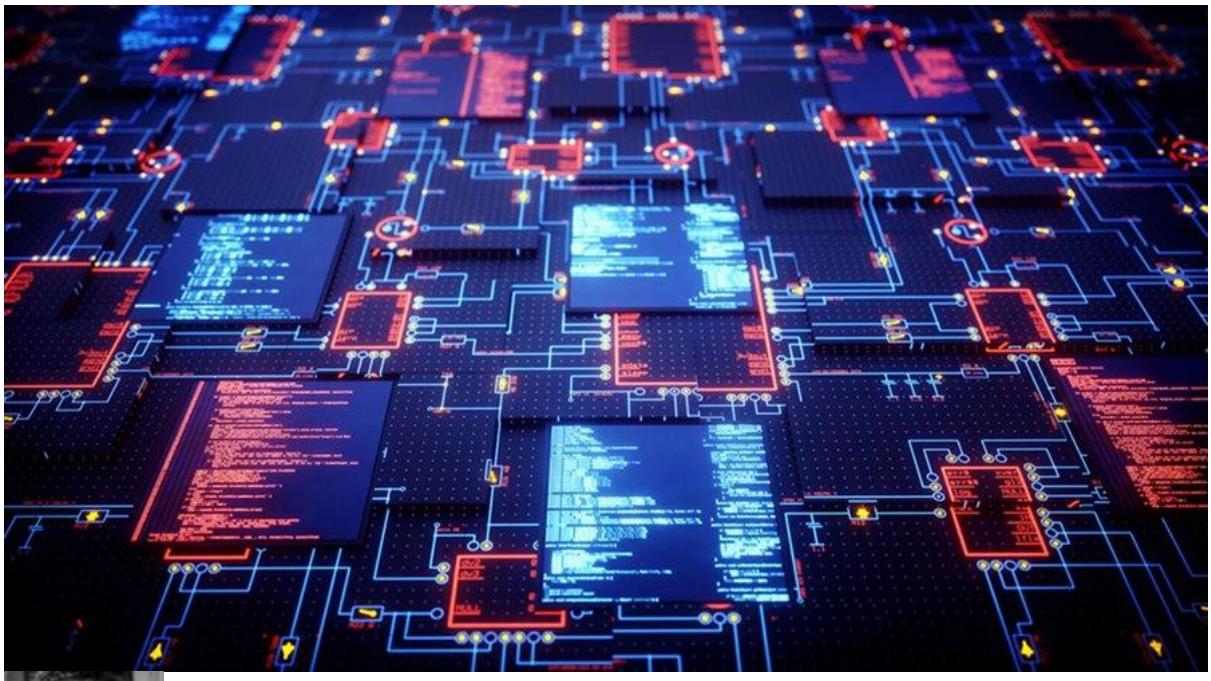
ARM Mobile Studio를 사용하여 Mali GPU 분석 속도 높이기



Peter Harris
2019년 3월 20일

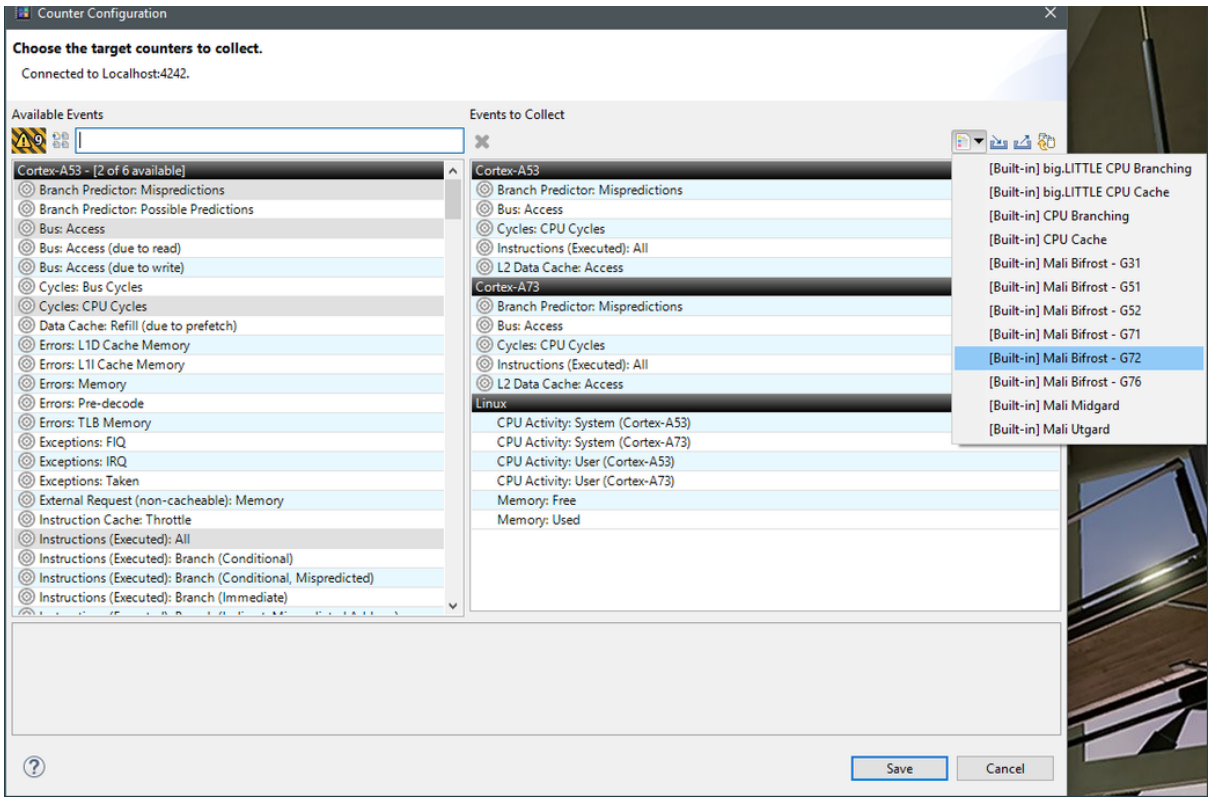
Streamline 성능 분석기는 장치에 있는 ARM CPU와 Mali GPU의 자세한 성능 정보를 제공할 수 있는 샘플 기반 프로파일러입니다. Streamline의 최근 버전에는 사용할 데이터 세트를 쉽게 선택하고 데이터 세트의 시각화 방법을 제어하는 데 사용할 수 있는 사전 정의된 템플릿이 포함되어 있습니다. ARM Mobile Studio 2019.0과 ARM Development Studio 2019.0에 포함된 Streamline 최신 릴리스에는 Mali Bifrost GPU 계열을 위한 Mali GPU 템플릿의 개선 사항이 다수 포함되어 있습니다. 이 글에서는 Mali-G72 GPU용 템플릿의 사용 방법을 살펴봅니다.

이 블로그에서는 독자가 그래픽 용어, 특히 타일 기반 렌더링 GPU 아키텍처와 관련된 용어에 익숙하다고 가정합니다. 이러한 주제에 대한 몇 가지 유용한 빠른 시작 가이드는 아래에서 찾을 수 있습니다.

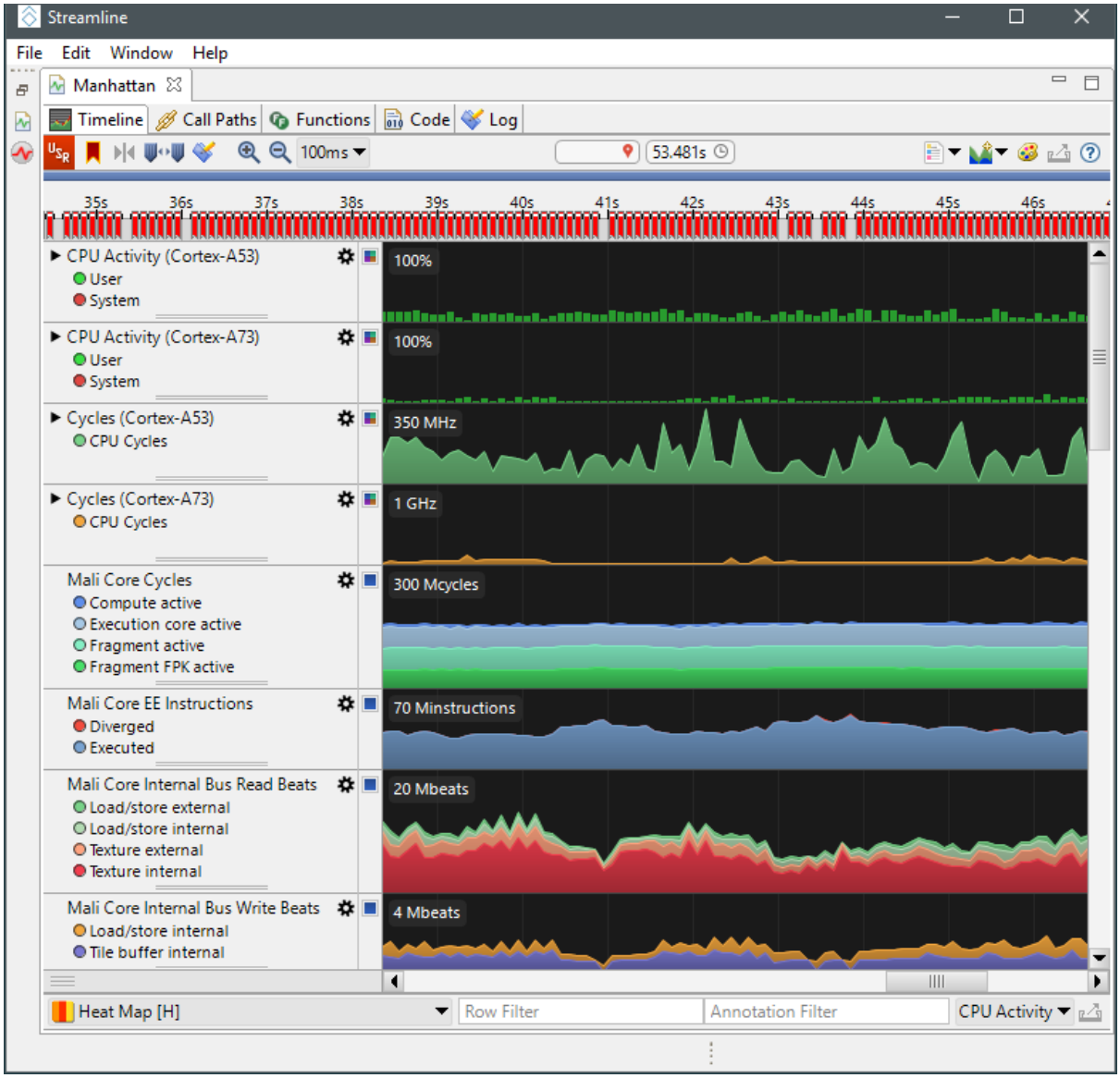
- Understanding Render Passes (렌더 패스의 이해)
- Understanding GPU Pipelining (GPU 파이프라인의 이해)
- Understanding Tile-based Rendering (타일 기반 렌더링의 이해)
- Introduction to the Mali Bifrost Shader Core (Mali Bifrost Shader Core 소개)

카운터 선택

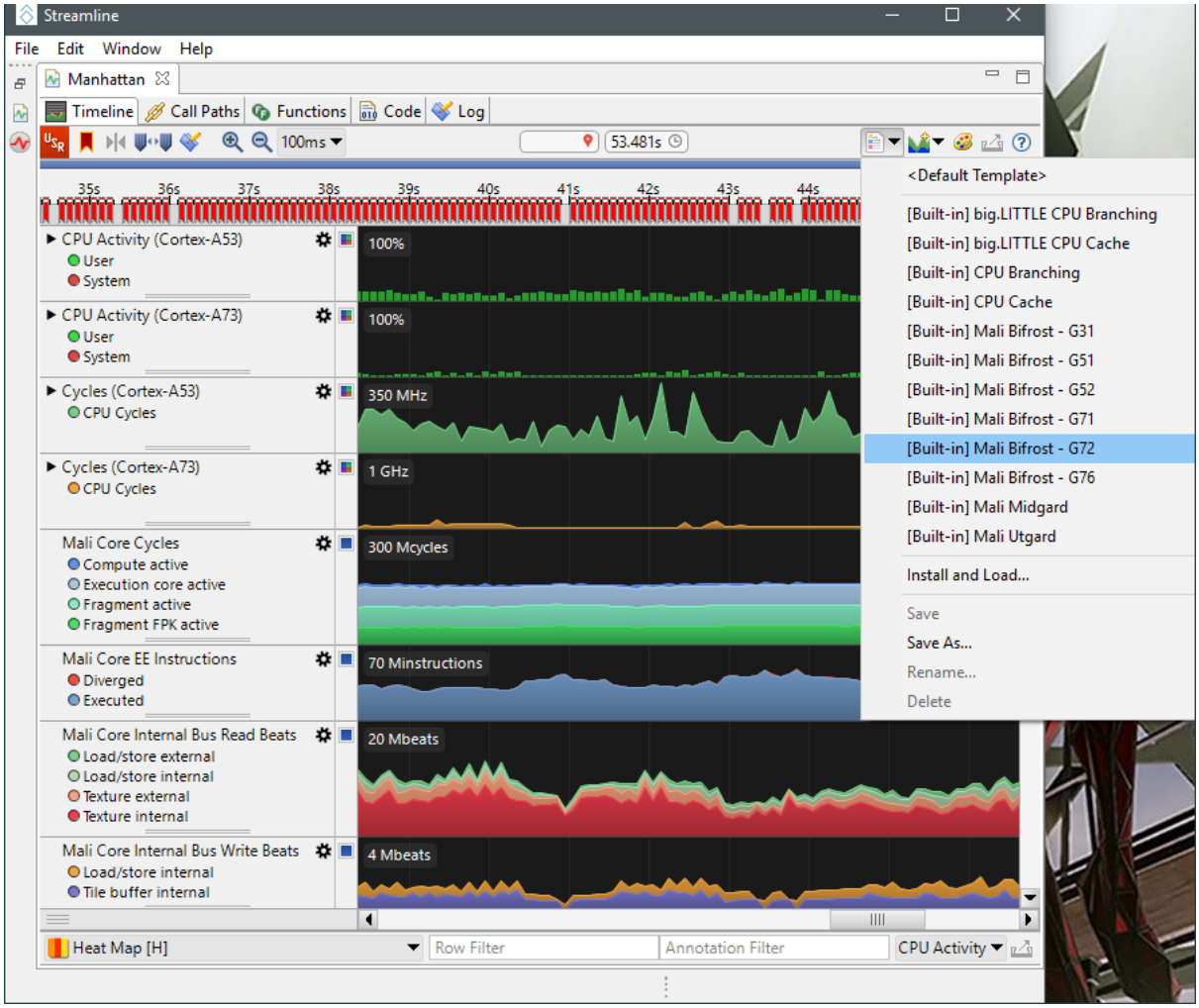
Quick Start Guide 따라 응용 프로그램을 설정하고 타겟에 게이터 데몬(gator daemon)을 설치했다면 이제 몇 가지 데이터 소스를 선택하고 프로파일링을 시작해야 합니다. 장치에 연결하고 Counter Selection 대화 상자를 표시합니다. Counter Selection 대화 상자의 드롭다운 메뉴에서 장치에 적합한 템플릿을 선택합니다.



이렇게 하면 템플릿의 시각화를 렌더링하는 데 필요한 모든 데이터 소스가 자동으로 선택됩니다. Save를 클릭한 다음, 응용 프로그램의 트레이스를 캡처합니다. 초기 데이터 분석이 완료되면 기본 Timeline 시각화가 표시됩니다.



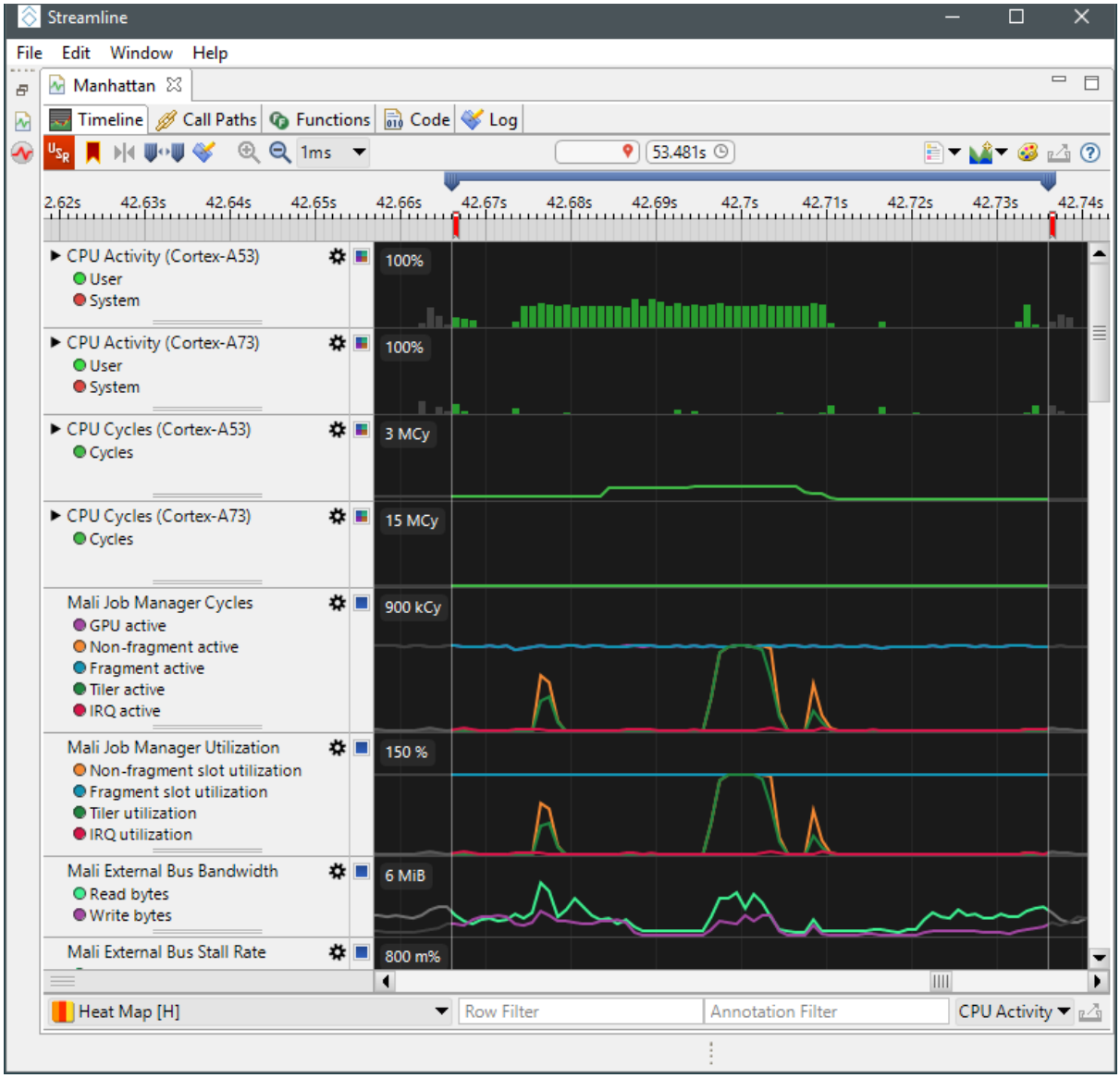
이것은 캡처된 알파벳순 차트 목록과 데이터 계열을 보여 줍니다. 따라서 제일 먼저 할 일은 시각화 캡처에 사용한 것과 동일한 템플릿을 선택하는 것입니다.



이렇게 하면 우리의 성능 분석팀이 설계한 미리 정의된 시각화를 표시하도록 Timeline이 변경됩니다. 그러면 차트의 순서가 더 체계적으로 지정되며, 수학 표현식을 활용하여 여러 원시 카운터를 조합해 기능 유닛의 사용률 같은 더 읽기 쉬운 메트릭이 도출됩니다.

프레임 찾기

Timeline에 표시되는 초기 뷰가 제공하는 시간은 화면 샘플당 1초입니다. 우리의 가장 큰 관심사는 일반적으로 길이가 16~32밀리초 사이인 프레임을 얼마나 잘 처리하고 있는지 보는 것이므로 1초 단위는 그래픽 콘텐츠를 디버깅하기에는 시간 간격이 너무 큽니다. 따라서 분석의 첫 단계는 단일 프레임을 구별할 수 있을 때까지 뷰를 확대하는 것입니다.



샘플에 표시된 응용 프로그램에서는 응용 프로그램이 eglSwapBuffers()를 호출할 때마다 Streamline 마커 주석이 생성되도록 소스 코드에 주석 (annotation)이 추가됐습니다. 이것은 차트 위 타임 트랙에 빨간색 눈금으로 표시됩니다.

개별 프레임을 볼 수 있게 되면 현재 시스템 동작의 초기 평가가 가능합니다.

- 프레임 간 시간을 측정하여 달성된 프레임률을 확인합니다.
- CPU 스레드 부하를 측정하여 CPU 바운드 여부를 확인합니다.
- GPU 스레드 부하를 측정하여 GPU 바운드 여부를 확인합니다.
- CPU 및 GPU 워크로드의 파이프라인을 조사하여 응용 프로그램 논리가 버블 스케줄링 없이 그래픽 파이프라인을 최적으로 피딩하는지 확인합니다.

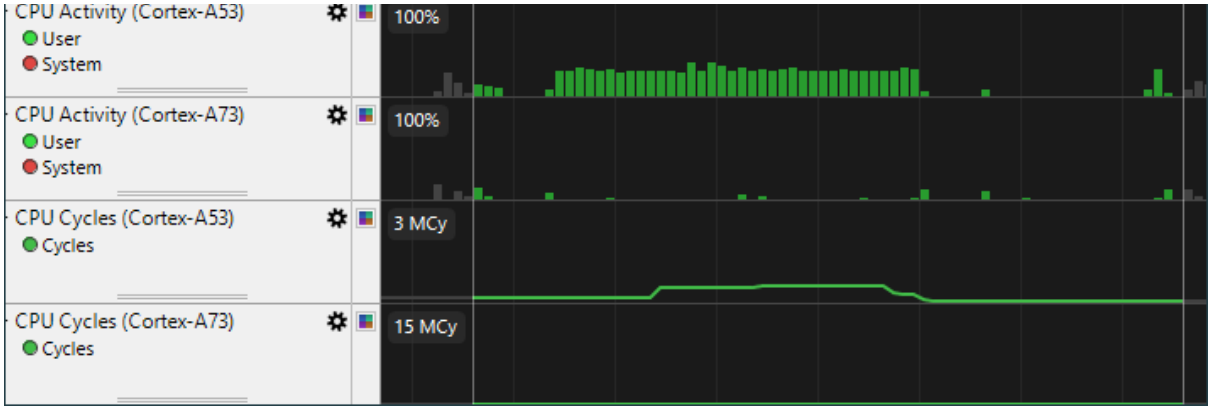
위의 예에서는 프레임의 상당 부분에 대해 CPU가 모두 유휴 상태가 되는 것을 볼 수 있으므로 CPU 바운드가 아닙니다. 또한 GPU가 항상 활성 상태인 것도 볼 수 있으므로 GPU가 이 응용 프로그램의 성능을 제한하는 프로세서일 가능성이 매우 높습니다.

GPU 워크로드를 더 세분하면 프래그먼트 쉐이딩 큐가 항상 활성 상태이며, 모든 지오메트리 및 컴퓨팅 처리에 사용되는 비 프래그먼트 큐는 프레임 대부분 동안 유휴 상태가 되는 것을 볼 수 있습니다. 따라서 성능을 개선하고 싶다면 이 응용 프로그램의 조각 워크로드를 최적화해야 합니다.

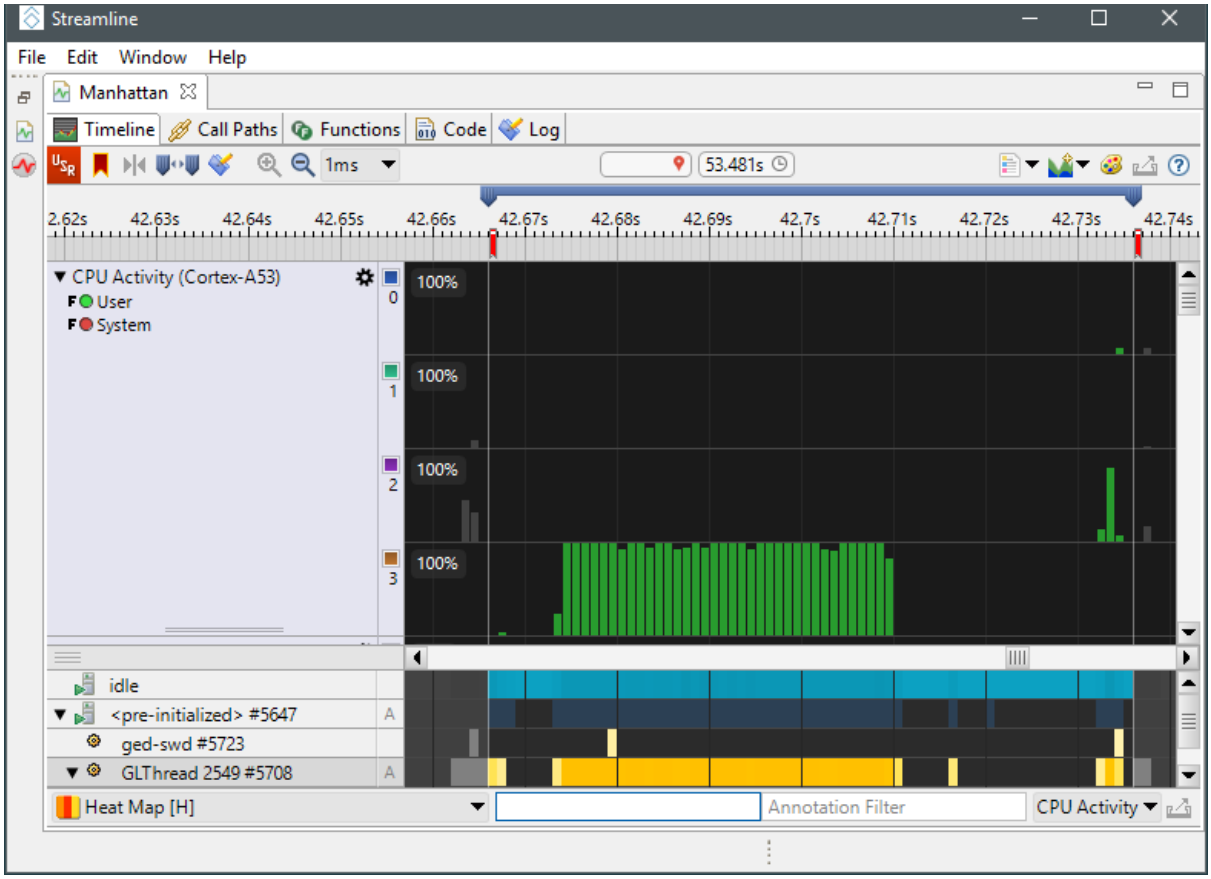
이 자습서의 이후 섹션에서는 템플릿의 각 차트를 살펴보고, 그 의미가 무엇이며 차트의 변화가 성능 개선을 모색하는 응용 프로그램 개발자에게 무엇을 시사하는지 설명합니다.

CPU 워크로드

CPU 차트는 시스템에 있는 CPU의 전체적 사용량을 보여 줍니다.



CPU Activity 차트는 big.LITTLE 클러스터링이 있는 경우, 프로세서 유형으로 나누어 CPU가 활성 상태였던 시간의 비율로 계산한 CPU별 사용률을 보여 줍니다. 이것은 OS 스케줄링 이벤트 데이터에 기반합니다. *CPU Cycles* 차트는 CPU 성능 모니터링 유닛(PMU)을 사용하여 측정된, 각 CPU가 활성 상태였던 사이클 수를 보여 줍니다. 이 두 가지를 함께 고려하면 전체적인 응용 프로그램 소프트웨어 부하를 평가할 수 있습니다. 높은 사용률과 높은 CPU 사이클 수는 CPU가 매우 바쁘며 높은 클록 주파수로 실행되고 있음을 나타냅니다.

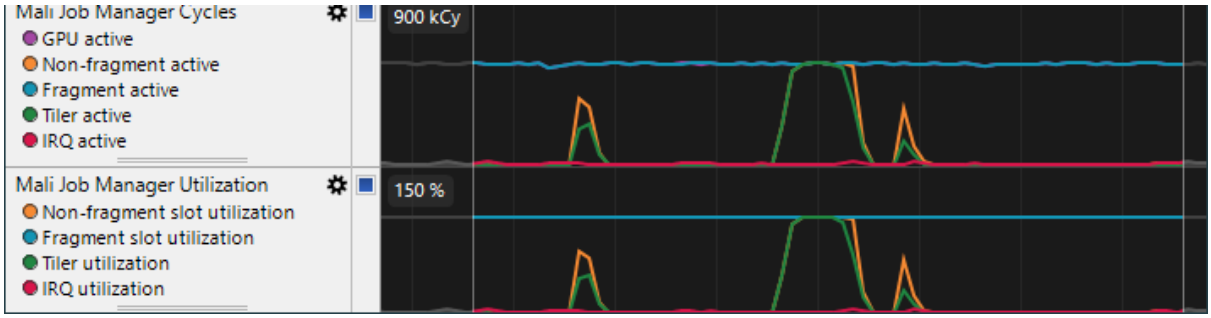


Timeline 탭 하단의 프로세스 뷰는 응용 프로그램 스레드 활동을 보여 주므로 어떤 스레드에서 부하가 발생하는지 파악할 수 있습니다. 목록에서 하나 또는 다수개의 스레드를 선택하면 CPU 관련 차트가 필터링되므로 선택한 스레드의 부하만 표시할 수 있습니다. 스레드 레벨 필터가 활성 상태이면 차트 제목 배경이 파란색으로 바뀌어 측정된 부하 일부를 현재 볼 수 없음을 나타냅니다.

응용 프로그램이 성능 목표에 도달하지 못하고 항상 활성 상태인 CPU 스레드가 하나라면 CPU 바운드일 가능성이 높습니다. 프레임 시간을 개선하려면 이 스레드의 워크로드 비용을 줄이는 소프트웨어 최적화가 필요합니다. Streamline은 성능 카운터 뷰 외에도 프로그램 카운터 샘플링을 통해 네이티브 소프트웨어 프로파일링을 제공합니다. 소프트웨어 프로파일링은 이 튜터리얼의 범위를 벗어나므로 자세한 내용은 Streamline 사용 설명서를 참조하십시오.

GPU 워크로드

GPU 워크로드 차트는 GPU의 전체 사용량을 보여 줍니다.



Mali Job Manager Cycles 차트는 GPU 전체에서 비프래그먼트 및 프래그먼트 작업을 위한 두 개의 병렬 하드웨어 작업 큐를 실행하는 작업에 사용된 GPU 사이클 수를 보여 줍니다. *Mali Job Manager Utilization* 차트는 *GPU 활성 사이클(GPU Active Cycle)*에 대한 백분율로 정규화된 동일한 데이터를 보여 줍니다.

GPU 바운드 콘텐츠의 경우, 주된 작업 큐는 항상 활성 상태여야 하고 다른 큐는 이 큐에 병렬로 실행되어야 합니다. GPU 바운드 응용 프로그램의 병렬 처리가 좋지 않은 경우, glFinish()와 같이 렌더링 파이프라인을 드레이닝하는 API 호출 또는 glReadPixels()의 동기화 사용 또는 여러 렌더 패스의 단계 중첩(프레임 간 중첩 포함)을 허용하기에는 너무 보수적인 Vulkan 종속성을 확인하십시오.

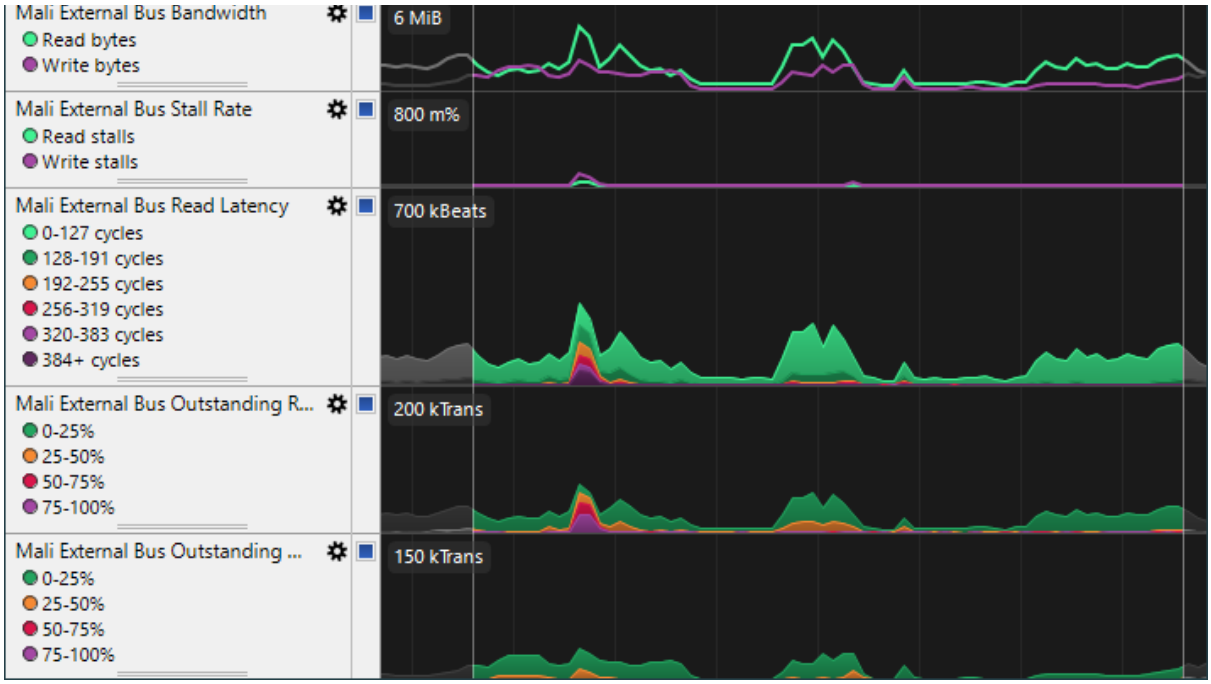
이 차트의 *Tiler active* 카운터는 지오메트리 처리 전체 기간 동안 일반적으로 타일러가 활성 상태이므로 직접적으로 유용하지 않을 수도 있지만 컴퓨트 쉐이딩이 얼마나 있는지 보여 줄 수 있습니다. *Non-fragment active*와 *Tiler active*의 격차가 크다면 응용 프로그램 컴퓨트 쉐이더가 원인일 수 있습니다.

IRQ active 카운터는 보류 중인 CPU와의 인터럽트가 있는 GPU의 사이클 수를 보여 줍니다. GPU 사이클 2%까지의 IRQ 보류율은 정상이지만 응용 프로그램이 많은 수의 작은 렌더 패스 또는 컴퓨팅 디스패치를 큐에 넣어 인터럽트 비율이 높아질 수 있습니다.

참고: 높은 IRQ 오버헤드는 권한 있는 커널 동작에 의해 장시간 CPU 인터럽트가 마스킹되는 경우와 같은 시스템 통합 문제를 나타낼 수도 있습니다. 응용 프로그램 변경을 사용하여 높은 IRQ 오버헤드를 수정하기는 일반적으로 불가능합니다.

GPU 메모리 시스템

메모리 시스템 차트는 GPU에 의해 생성된 메모리 트래픽과 시스템이 이 트래픽을 얼마나 효과적으로 처리하는지를 기준으로 GPU 메모리 인터페이스에 나타나는 동작을 보여 줍니다.



Mali External Bus Bandwidth 차트는 응용 프로그램에 의해 생성된 총 읽기 및 쓰기 대역폭을 보여 줍니다. 외부 DDR 메모리 액세스가 매우 에너지 집약적이므로 메모리 대역폭 감소는 효과적인 응용 프로그램 최적화 목표가 될 수 있습니다. 이후의 차트는 응용프로그램의 어떤 리소스가 트래픽의 원인인지를 파악하는 데 도움이 됩니다.

Mali External Bus Stall Rate 차트는 버스 정지가 있는 GPU 사이클의 비율을 보여 주며, GPU가 외부 메모리 시스템으로부터 얼마나 많은 배압을 받고 있는지 나타냅니다. 5%까지의 정지 비율은 정상으로 간주되며, 이보다 훨씬 높은 정지 비율은 메모리 시스템이 처리할 수 있는 것보다 많은 트래픽을 생성하는 워크로드를 나타냅니다. 전체 메모리 대역폭을 줄이거나 액세스 지역(locality)을 개선하면 정지 비율을 줄일 수 있습니다.

Mali External Bus Read Latency 차트는 외부 메모리 액세스 응답 지연의 스택형 히스토그램을 보여 줍니다. Mali GPU는 GPU 사이클 170회까지의 외부 메모리 지연을 고려하여 설계되었으므로 더 느린 빈(bin)에서의 높은 읽기 비율은 메모리 시스템 성능 문제를 나타낼 수 있습니다. DDR 성능은 일정하지 않으며 DDR이 높은 부하를 받을 때는 지연이 증가하므로 대역폭을 줄이는 것이 지연을 줄이는 효과적인 방법이 될 수 있습니다.

참고: DDR은 공유 리소스이고 시스템의 다른 부분에서 오는 경합 트래픽이 있기 때문에 메모리 액세스 중 적은 부분은 더 느린 빈에 있을 것으로 예상됩니다.

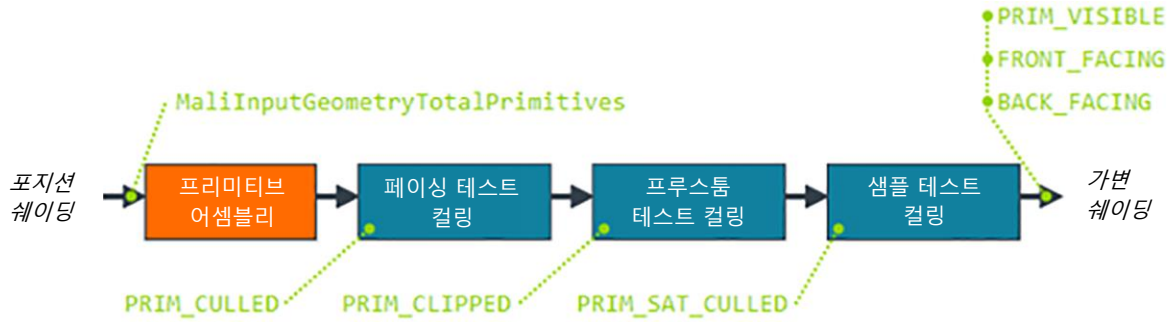
Mali External Bus Outstanding Reads/Writes 차트는 또 다른 스택형 히스토그램 세트를 보여 주는데, 이번에는 GPU가 메모리 시스템 큐에 넣은 허용된 메모리 액세스의 비율을 보여 줍니다. 히스토그램이 75~100% 빈에 있는 비율이 높을 경우, GPU에 트랜잭션이 부족할 수 있습니다. 이렇게 되면 이전 메모리 요청이 사용 중지될 때까지 새 메모리 요청이 정지됩니다. DDR에서 메모리 대역폭을 줄이거나 액세스 지역(locality)을 개선하면 성능을 높일 수 있습니다.

GPU 지오메트리

지오메트리 차트는 GPU가 처리 중인 지오메트리의 양과 프리미티브 컬링 유닛의 동작을 보여 줍니다.

Mali Primitive Culling 차트는 처리되는 프리미티브의 절대 수, 각 컬링 단계에 의해 소멸되는 프리미티브 수, 보이는 프리미티브 수를 보여 줍니다. 단일 정점은 메모리 대역폭 요구 사항이 높기 때문에 단일 프래그먼트보다 처리에 훨씬 더 많은 비용이 듭니다. 따라서 프레임당 총 프리미티브 수를 최대한 줄이는 것을 목표로 해야 합니다.

Mali Primitive Culling Rate 차트는 각 컬링 단계에 진입하여 해당 단계에 의해 소멸되는 프리미티브의 비율과 보여지는 프리미티브의 비율을 보여 줍니다. 컬링 파이프라인은 일련의 처리단계로 실행됩니다.



3D 장면의 경우, 프리미티브의 50%까지는 후면이며 페이싱 테스트 컬링 유닛에 의해 소멸될 것으로 예상됩니다. *Culled by facing test* 비율이 이보다 훨씬 낮다면 페이싱 테스트가 올바르게 활성화되어 있는지 검토하십시오.

CPU에서 프루스툼 밖에서 그리는 드로우 콜들이 컬링되게 하는 것이 응용프로그램의 표준 모범 사례이므로 *Culled by frustum test* 비율은 최대한 낮춰야 합니다. 입력 프리미티브의 10% 이상이 이 단계에서 소멸되는 경우, CPU 측 컬링의 효과를 검토하십시오. 또한 지나치게 큰 객체 배치(batch)는 컬링 효율을 떨어뜨릴 수 있으므로 배치 크기를 검토하는 것이 좋습니다.

마지막 컬링 비율인 *Culled by sample test*는 너무 작아서 래스터화 샘플 포인트에 도달하지 못해 소멸되는 프리미티브의 비율을 측정합니다. 밀집된 지오메트리는 직접 정점 처리 비용이나 프래그먼트 쉐이딩 효율 감소 측면에서 비용이 매우 많이 들기 때문에 이 수치는 가능하면 0%에 가깝게 억제해야 합니다. 여기서 소멸되는 프리미티브가 많을 경우, 정적 메쉬 밀도와 동적으로 lod(level-of-detail)를 선택하는 효과를 검토하십시오.

Mali Geometry Threads 차트는 Mali의 인덱스 기반 정점 쉐이딩 알고리즘에 의해 생성된 쉐이딩 요청의 절대 수를 보여 줍니다. 이 설계는 응용 프로그램의 버텍스 쉐이더를 위치를 계산하는 부분과 다른 varying 변수를 계산하는 부분으로 양분합니다. 가변 쉐이더는 클리핑과 컬링에서 살아남는 프리미티브에 속한 버텍스에 대해서만 실행됩니다. 이 시점에서는 여러 가지를 검토할 수 있습니다.

- 총 포지션 쉐이더 호출 수를 응용 프로그램 인덱스 버퍼와 비교합니다. 응용 프로그램이 제출한 것보다 많은 인덱스를 GPU가 쉐이딩하는 경우, 인덱스 지역(locality)이 좋지 않은 것일 수 있으며, 이로 인해 포지션 캐시 스래싱과 강제 리쉐이딩이 발생할 수 있습니다.

- 총 포지션 쉐이더 호출 수를 총 입력 프리미티브 수와 비교합니다. 대부분의 콘텐츠의 경우, 비용을 최대한 분할하기 위해서는 여러 인접 프리미티브가 단일 버텍스를 사용해야 하므로 프리미티브당 평균 정점 1개 미만을 목표로 하십시오.

GPU 쉐이더 프런트 엔드

쉐이더 프런트 엔드 차트는 프리미티브를 쉐이딩할 프래그먼트 스레드들로 바꾸는 고정 함수 유닛의 동작을 보여 줍니다.

Mali Core Primitives 차트는 래스터화를 위해 로드되는 프리미티브의 수를 보여 줍니다. Mali는 타일당 한 번씩 큰 프리미티브를 로드하므로 교차하는 타일마다 이 숫자에 단일 프리미티브가 한 번씩 포함된다는 점을 명심하십시오.

Mali Early ZS Testing Rate 차트는 깊이(Z) 및 스텐실(S) 테스트와 프런트 엔드의 컬링 비율을 보여 줍니다. Early ZS 테스트는 Late ZS 테스트보다 비용이 훨씬 적게 들므로 거의 모든 프래그먼트를 *Early ZS 테스트*하는 것을 목표로 하십시오. 그 방법은 shader discard, alpha-to-coverage, 쉐이더가 생성한 깊이 값의 사용을 최소화하는 것입니다. *FPK killed* 카운터는 Mali의 Forward Pixel Kill 은면 제거(hidden surface removal) 체계에 의해 소멸되는 쿼드들(quad)의 비율을 보고합니다. FPK에 의해 소멸되는 쿼드 비율이 높은 것은 뒤에서 앞으로의 렌더링 순서를 나타내며, 이를 앞에서 뒤로의 렌더링 순서로 역전하면 Early ZS 테스트 중에 더 일찍 쿼드가 소멸되어 에너지 소비가 줄어듭니다.

Mali Late ZS Testing Rate 차트는 깊이 및 스텐실 테스트와 조각 쉐이딩 후 백 엔드의 컬링 비율을 보여 줍니다. Late ZS 테스트 중에 소멸되는 쿼드 비율이 높다는 것은 쉐이딩된 후 조각이 소멸되는 것이므로 잠재적인 효율 문제를 나타냅니다.

참고: 투명 색상보다 기존 깊이 또는 스텐실 어태치먼트를 시작 상태로 사용하는 렌더 패스는 리로드 프로세스의 일환으로 Late ZS 연산을 트리거합니다. 이것은

불가피할 수도 있지만 모든 어태치먼트의 삭제 없이 시작하는 렌더 패스의 수를 최소화하는 것을 목표로 하십시오.

Mali Core Warps 차트는 컴퓨팅 프런트 엔드(모든 비 프래그먼트 워크로드 포함)와 프래그먼트 프런트 엔드에 의해 생성된 워프의 수를 보여 줍니다. 워프 폭은 제품마다 다를 수 있습니다.

마지막 두 개의 차트는 스레드당 평균 쉐이더 코어 처리 비용을 보여 줍니다. GPU 바운드 콘텐츠의 경우, 쉐이더 워크로드에는 다음과 같은 가능한 두 가지 최적화 목표가 있습니다.

- 장면 콘텐츠를 단순화하여 생성되는 워프의 수를 줄이거나
- 쉐이더 프로그램을 최적화하여 스레드당 비용을 줄이는 것입니다.

GPU 쉐이더 프런트 엔드 픽셀

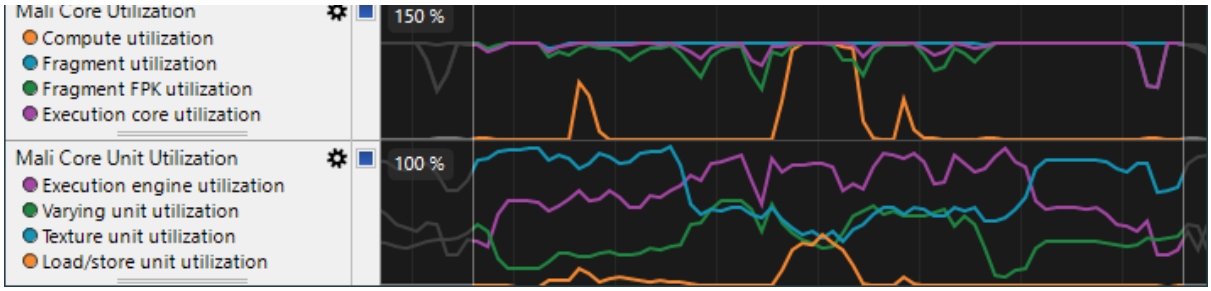
이 차트 세트는 쉐이더 코어가 픽셀을 생성하는 속도를 살펴봅니다.

Mali Pixels 차트는 모든 쉐이더 코어에 의해 쉐이딩된 총 픽셀 수를 보여 주므로 프레임 생성에 필요한 총 픽셀 수를 평가할 수 있습니다.

Mali Overdraw 차트는 출력 픽셀당 쉐이딩된 평균 프래그먼트 수를 보여 줍니다. 오버드로우 레벨이 높으면 조각당 비용이 낮더라도 성능이 줄어들 수 있습니다. 오버드로우를 줄이려면 사용 중인 투명 프래그먼트의 레이어 수를 최소화하는 것을 목표로 하십시오.

GPU 쉐이더 코어

쉐이더 코어는 GPU의 핵심이므로 쉐이더 코어 워크로드를 검사할 수 있는 카운터가 많은 것에 놀라서는 안 됩니다. 첫 번째 차트 세트의 목표는 전체적인 쉐이더 코어 사용률을 한눈에 볼 수 있는 뷰를 제공하는 것입니다.



참고: 쉐이더 코어 "컴퓨팅" 데이터 경로는 모든 비 프래그먼트 워크로드 처리에 사용되므로 컴퓨팅 관련 카운터에는 정점 쉐이딩 워크로드도 포함됩니다.

Mali Core Utilization 차트는 쉐이더 코어의 3개 주요 부분의 사용률을 보여 줍니다.

- *Compute utilization* 계열과 *Fragment utilization* 계열은 래스터화 및 타일 라이트백(writeback) 같은 고정 함수 논리에 사용된 시간을 포함하여 쉐이더 코어가 해당 유형의 워크로드를 처리하는 시간의 비율을 보여 줍니다.
- *Execution core utilization* 계열은 프로그래밍 가능한 코어 자체가 활성 상태인 시간의 비율을 보여 줍니다. 이 비율이 장기간 100% 미만인 경우, 쉐이더 코어 (programmable shader core)를 계속 일할 수 있도록 해야 하는 문제가 있음을 나타낼 수 있습니다.
- 이 차트의 *Fragment FPK utilization* 계열은 쿼드를 큐에 넣어 프래그먼트 스레드로 변환되기를 기다리는 시간의 비율을 보여 줍니다. 이 비율이 장기간 100% 미만인 경우, 쉐이더 코어를 위한 새 프래그먼트를 충분히 빠르게 생성하지 못하고 있음을 나타낼 수 있습니다. 이 현상의 원인은 프리미티브당 소수의 프래그먼트를 생성하는 마이크로트라이앵글이 많아서일 수도 있지만 일반적인 그림자 맵 유형처럼 지오메트리를 전혀 포함하지 않는 빈 타일 수가 많은 워크로드를 나타낼 수도 있습니다.

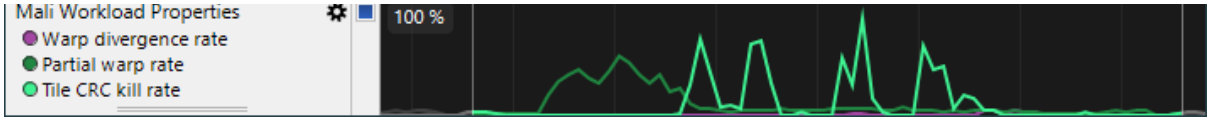
Mali Core Unit Utilization 차트는 실행 코어 내부의 주요 파이프라인 사용률을 보여 줍니다.

- *Execution engine utilization* 계열은 쉐이더 코어 산술 유닛이 활성 상태인 시간의 비율을 보여 줍니다.
- *Varying unit utilization* 계열은 고정 함수 보간 유닛이 활성 상태인 시간의 비율을 보여 줍니다.
- *Texture unit utilization* 계열은 고정 함수 텍스처 샘플링 및 필터링 유닛이 활성 상태인 시간의 비율을 보여 줍니다.
- *Load/store unit utilization* 계열은 범용 메모리 액세스 유닛이 활성 상태인 시간의 비율을 보여 줍니다.

쉐이더 코어 바운드인 쉐이더 콘텐츠의 경우, 이 차트를 사용하여 가장 심한 부하가 걸리는 유닛을 찾는 것이 최적화 대상을 결정하는 좋은 방법입니다.

워크로드 속성

Mali Workload Properties 차트에는 워크로드의 흥미로운 동작을 나타내는 다양한 구성요소 계열이 포함됩니다.



Warp divergence rate 계열은 일부 실행 레인(lane)이 마스킹되도록 하는 제어 흐름 분기가 워프에 있을 때 실행되는 명령어의 비율을 보고합니다. 쉐이더 실행 효율을 급속히 잠식할 수 있는 제어 흐름 분기의 최소화를 목표로 하십시오.

Partial warp rate 계열은 커버리지 없는 스레드 슬롯을 포함하는 워프의 비율을 보고합니다. 이것은 프리미티브 엣지와 교차하는 프래그먼트 쿼드로 인해 발생하며, 결과적으로 히트 샘플이 없는 프래그먼트가 생깁니다. 높은 부분 워프 비율은

마이크로트라이앵글 또는 매우 얇은 삼각형이 많은 응용 프로그램을 나타낼 수 있습니다. 쉐이더 실행 효율을 급속히 잠식할 수 있는 부분 워프 수를 최소화하는 것을 목표로 하십시오.

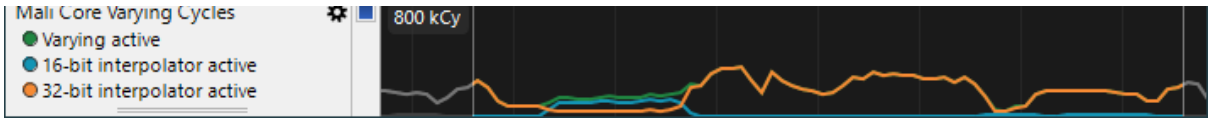
Tile CRC kill rate 계열은 CRC 일치로 인해 소멸되는 타일의 비율을 보고하며, 계산된 색상이 메모리에 이미 있는 색상과 일치함을 나타냅니다. 높은 소멸 비율은 응용 프로그램이 화면의 변경된 부분만 식별하고 드로우할 수 있는 경우, 최적화 기회를 나타낼 수 있습니다.

산술 유닛(Arithmetic Unit)

Execution Engine은 모든 산술 워크로드를 비롯한 모든 쉐이더 명령어를 실행하는 데 사용됩니다. 앞서 설명한 *Mali Core Unit Utilization* 차트의 *Execution engine utilization* 계열을 사용하여 응용 프로그램의 산술 처리가 제한되는지 여부를 확인할 수 있습니다.

가변 유닛(Varying Unit)

Mali Core Varying Cycles 차트는 데이터 정밀도로 나누어 고정 함수 가변 보간 사용량을 보고합니다.

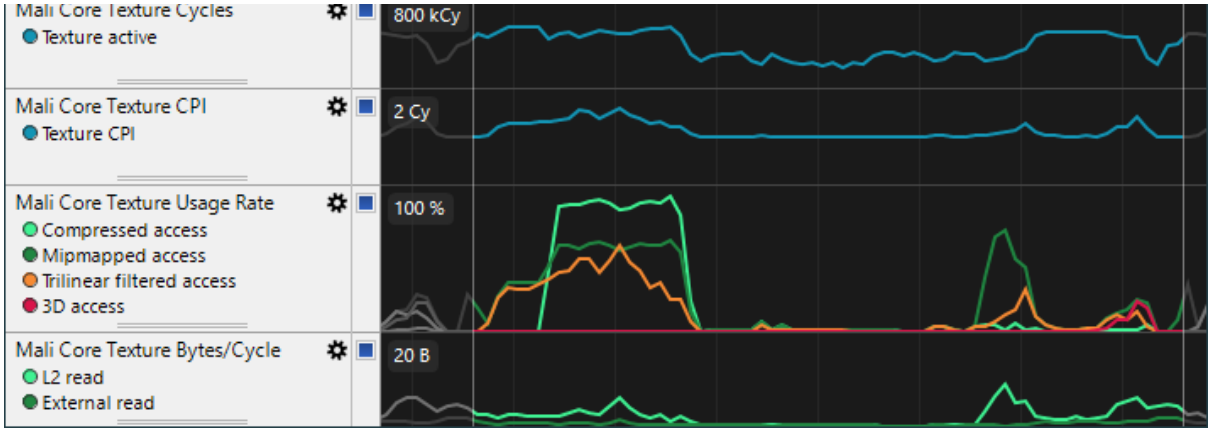


가변 바운드인 콘텐츠의 경우, 다음과 같은 가능한 세 가지 최적화 기회가 있습니다.

- 직접 워크로드 감소: 프레임당 로드되어야 하는 varying 변수의 전체 수 감소.
- 정밀도 감소: 32비트 highp에서 16비트 mediump 가변 매개 변수로 전환하면 보간 요구 사항이 절반이 되며, 이로 인해 종종 쉐이더 로직에서 연쇄적으로 성능개선이 발생합니다.
- varying 패킹: 16비트 varying 변수를 32비트의 배수인 벡터로 패킹하면 사용되지 않는 보간 레인으로 인한 손실 사이클이 최소화됩니다. 예를 들어 패킹된 vec4는 float와 별도의 vec3보다 한 사이클 빨리 보간됩니다.

텍스처 유닛

텍스처 유닛은 모든 텍스처 샘플링과 필터링을 처리하는 복잡한 유닛입니다. 텍스처 유닛은 사용되는 텍스처 형식과 필터링 모드에 따라 성능이 달라질 수 있습니다.



Mali Core Texture Cycles 차트는 고정 함수 텍스처 필터링 유닛의 총 사용량을 보고합니다.

Mali Core Texture CPI 차트는 요청당 평균 텍스처 사이클 수를 보고하며, 더 복잡한 필터링 모드 사용 시 트리거되는 다중 사이클 연산의 수를 파악할 수 있습니다. 텍스처 바운드 콘텐츠의 경우, 단순한 필터링 모드를 사용하여 CPI를 줄이는 것이 효과적인 성능 개선 방법이 될 수 있습니다.

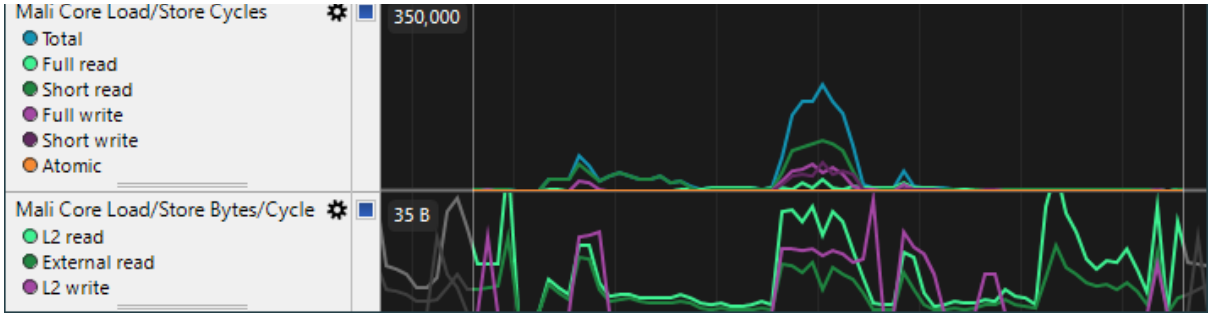
Mali Core Texture Usage Rate 차트는 수행 중인 텍스처 액세스 유형에 대한 통계를 보고합니다.

- *Compressed access* 계열은 ASTC와 ETC 같은 블록 압축 텍스처 형식을 사용 중인 텍스처 액세스의 비율을 보고합니다. 게임 렌더링은 대역폭을 줄이기 위해 블록 압축 텍스처를 최대한 이용해야 합니다.
- *Mipmapped access* 계열은 밉맵 텍스처를 사용 중인 텍스처 액세스의 비율을 보고합니다. 게임 렌더링은 성능 및 이미지 품질 개선을 위해 모든 3D 장면에 밉맵 텍스처를 사용해야 합니다.
- *Trilinear filtered access* 계열은 삼선형 필터링을 사용 중인 텍스처 샘플의 비율을 보고합니다. 이러한 액세스는 이선형 액세스의 절반 속도로 실행됩니다.
- *3D access* 계열은 샘플을 입체적 텍스처로 만드는 텍스처 샘플의 비율을 보고합니다. 이러한 액세스는 2D 액세스 텍스처의 절반 속도로 실행됩니다.

Mali Core Texture Bytes/Cycle 차트는 각 필터링 사이클을 위해 L2와 외부 메모리에서 가져와야 할 바이트 수를 보고합니다. 외부 액세스는 특히 에너지 집약적이므로 압축된 텍스처와 밉맵을 사용하고 캐시 프레셔를 줄이기 위해 좋은 샘플 지역(locality)을 사용하는 것이 좋습니다.

로드/저장 유닛

로드/저장 유닛은 일반적인 읽기/쓰기 데이터 메모리 액세스와 이미지 및 원자적 액세스를 제공합니다.

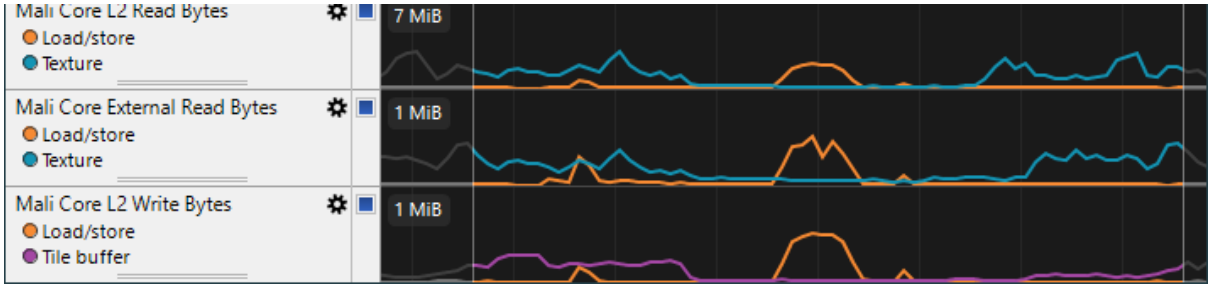


Mali Core Load/Store Cycles 차트는 로드/저장 캐시에 대해 수행된 액세스 유형을 보고합니다. 읽기 및 쓰기는 'full'이거나 'short'일 수 있습니다. short 액세스는 사용할 수 있는 데이터 버스 폭을 모두 사용하지 않습니다. 벡터화된 메모리 액세스를 수행하여 short 액세스 수를 줄이고 연산 쉐이더에서 공간적으로 인접한 스레드에 있는 인접 데이터에 액세스하면 성능 개선에 도움이 됩니다.

Mali Core Load/Store Bytes/Cycle 차트는 읽기당 L2 및 외부 메모리에서 가져올 바이트 수와 쓰기당 쓰이는 바이트 수를 보고합니다. 사용되는 알고리즘에 대한 지식 없이는 이 카운터를 해석하기가 어려울 수 있지만 메모리 액세스 및 데이터 사용 패턴을 알고 있는 경우, 연산 쉐이더 성능을 조사하는 데 유용할 수 있습니다. 예를 들어 대부분의 부하가 L2 캐시가 아닌 외부 메모리에서 발생하는 콘텐츠가 파악되면 작업 세트가 너무 크다는 뜻일 수 있습니다.

메모리 대역폭

마지막 차트 세트는 트래픽을 생성하는 유닛으로 나누어 쉐이더 코어에 의해 생성된 메모리 대역폭을 보여 줍니다.



전체적인 대역폭 문제가 있는 콘텐츠의 경우, 이 카운터는 가장 많은 트래픽을 생성하는 데이터 리소스를 파악하는 데 도움이 될 수 있습니다.

- 로드/저장 유닛 트래픽은 모든 유형의 버퍼 액세스와 image() 접근자를 통한 데이터 액세스와 관련됩니다.
- 텍스처 유닛 트래픽은 첨부가 삭제 또는 무효화되지 않은 경우, 렌더 패스 시작 시 타일 버퍼 콘텐츠를 복원하는 데 필요한 암시적 부하를 포함하여 모든 유형의 쉐이더 texture() 액세스와 관련됩니다.
- 타일 버퍼 트래픽은 렌더 패스 종료 시 메모리에 대한 모든 프레임 버퍼 첨부 나중 쓰기와 관련됩니다.

요약

이 글에서는 ARM CPU와 Mali GPU를 사용하여 그래픽 응용 프로그램의 초기 성능 검토를 수행하는 방법과 성능 카운터를 사용하여 주된 워크로드와 성능 저하의 가능한 원인을 파악하는 방법을 살펴봤습니다. Streamline에 내장된 사전 정의된 템플릿을 사용하면 체계적인 검토 워크플로우 지원에 필요한 카운터를 쉽고 효율적으로 캡처하여 설계에서 다양한 블록과 동작을 단계적으로 검토할 수 있습니다.

ARM Mobile Studio