GPU 메모리 확장 기술의 성능 및 CPU 사용량 분석*

이원식○ 김장우
서울대학교 전기정보공학부
{wonsik.lee, jangwoo.kim}@snu.ac.kr

Performance and CPU usage analysis of GPU memory extension

Wonsik Lee○ Jangwoo Kim
Dept. of Electrical and Computer Engineering, SNU

요 약

GPU에서 수행되는 어플리케이션들의 메모리 요구량이 점차 높아짐에 따라, GPU의 한정된 메모리 크기가 큰 문제가 되고 있다. 최근까지는 이러한 메모리 요구량에 대한 문제를 해결하기 위해 사용자가 GPU의 한정된 메모리 크기에 따라서 데이터가 이동할 수 있도록 프로그램을 작성하는 방식을 이용하였다. 하지만 이러한 프로그래밍 방식은 사용자에게 큰 부담이 되고, 이렇게 GPU의 한정된 메모리 크기에 따른 사용자의 프로그래밍 부담을 덜어주기 위해서 Unified Virtual Memory (UVM) 기술이 소개되었다. 이 기술은 GPU에서 수행되는 어플리케이션이 큰 메모리를 요구할 때 interrupt와 page fault 방식을 활용하여 호스트 메모리를 사용할 수 있도록 한다. 따라서 메모리가 부족했던 기존 어플리케이션들도 추가적인 어플리케이션 수정없이도 호스트 메모리까지 활용하여 동작을 수행할 수 있게 된다. 본 논문에서는 이러한 UVM 기술을 활용할 때에 발생하는 성능 병목 현상과 호스트 CPU 사용량을 분석하였다.

1. 서 론

최근 딥러닝, 그래프 처리 등 다양한 분야에서 그래픽 처리 장치 (Graphic Processing Unit)가 활용되고 있다. 그래픽 처리 장치, 즉 GPU는 쓰레드 단위의 병렬 처리를 통해 메모리 접근에 대한 높은 지연 시간이 드러나지 않도록 한다. 또한, 사용자의 편의를 위해 병렬 프로그래밍을 위한 언어와 프로그래밍 모델이 제공되고 있다. GPU는 호스트 메모리에서 GPU 메모리로 데이터를 복사 후 처리하는 과정을 기반으로 동작한다. 따라서 사용자들은 GPU의 병렬 처리 및 호스트 메모리와 GPU 메모리 간 데이터 이동 과정을 직접 프로그래밍하여 어플리케이션들을 수행한다. 이러한 프로그래밍 모델의 큰 문제는 어플리케이션이 필요로 하는 메모리 요구량과 GPU의 메모리 용량에 대한 정보를 기반으로 워크로드의 특성에 따라 사용자가 직접 프로그래밍을 해야 한다는 점이다.

AMD와 Nvidia에서는 이 문제를 해결하기 위해 GPU 메모리와 호스트 CPU 메모리를 하나의 메모리처럼 동작하게 하는 Unified Virtual Memory (UVM) 기술을 소개하였다 [1], [3]. 이 기술은 큰 워크로드 수정없이도 사용이 가능하다는 장점 때문에 높은 메모리 사용량을 필요로 하는 워크로드들에서 주목받고 있다.

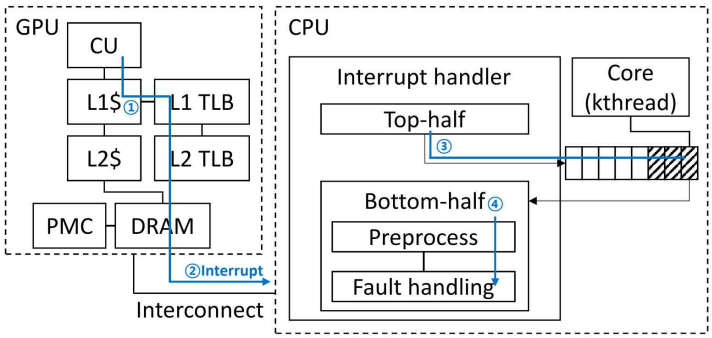

그림 1 Unified Virtual Memory (UVM)의 동작

UVM은 GPU와 호스트 CPU가 동일한 메모리 영역을 볼 수 있도록 포인터를 공유하고, 각 메모리에 있는 데이터가 사용자의 간섭 없이 GPU와 호스트 CPU의 메모리 사이를 이동할 수 있도록 제어한다. 따라서 사용자들은 별도의 복잡한 프로그래밍 과정 없이도 높은 메모리 사용량을 요구하는 워크로드들을 수행할 수 있게 된다. 하지만 UVM은 사용자의 프로그래밍 부담을 덜어주는 반면 이 기술을 활용하여 워크로드를 수행할 경우, GPU와 호스트 CPU간 반복적인 인터럽트에 따른 성능 오버헤드와 page fault 처리에 따른 높은 호스트 CPU 사용량을 요구하게 된다.

본 논문에서는 이러한 UVM 기술에서 발생하는 성능 오버헤드와 높은 호스트 CPU 사용량을 분석한다.

* 본 연구는 IDEC의 EDA Tool을 지원받아 수행하고, 서울대학교 자동화시스템연구소와 반도체공동연구소의 지원을 받음.

2. 기존 모델 분석

그림 1은 Unified Virtual Memory (UVM) 기술이 동작하는 형태를 보여주고 있다. 먼저 GPU가 워크로드를 수행할 때, 접근하는 데이터가 GPU 메모리에 없는 경우 GPU는 디바이스 드라이버로 인터럽트를 발생시킨다. 이 때 디바이스 드라이버에서는 인터럽트 처리에 대한 높은 지연 시간을 방지하기 위해 전체 처리 과정을 top-half와 bottom-half로 구분하고, 주로 높은 지연 시간을 필요로 하는 작업은 bottom-half에서 처리하게 된다. 따라서 디바이스 드라이버는 이 bottom-half에 대한 작업을 처리하기 위한 별도의 쓰레드 (kthread)를 생성하고 이를 특정 CPU 코어에 할당하여 처리한다. 이렇게 생성된 별도의 쓰레드는 (1) GPU에서 발생한 page fault에 대한 요청들을 batch 단위로 가져오고 (2) 이를 가상 주소 기반으로 정렬한 뒤 (3) 각 가상 주소에 대한 page fault 요청들을 처리하고 (4) 특정 조건에 따라 이후에 접근 가능성이 있는 특정 데이터들을 GPU 메모리로 미리 prefetch하고 난 뒤 인터럽트 처리를 완료한다. 하지만 호스트 CPU에서 발생하는 page fault와 다르게 GPU에서 발생하는 page fault는 디바이스 드라이버에서 수행되는 interrupt 처리에 따른 성능 오버헤드가 발생한다.

따라서 기존 UVM 기술에서도 page fault 처리에 따라 발생할 수 있는 성능 오버헤드를 줄이기 위해 여러 가지 최적화 방식들을 적용한다. 그 예로는, (1)과 같이 각 page fault마다 처리하지 않고 여러 page fault를 특정 batch 단위로 한꺼번에 처리하는 방식, (2)와 같이 동일한 page fault들을 중복하여 처리하지 않기 위해 page fault가 발생한 요청들을 가상 주소로 정렬한 뒤 동일한 page에 속하는 요청들을 제거한 뒤 처리하는 방식 그리고 (4)와 같이 미래에 발생할 가능성이 있는 page fault를 미리 방지 하기 위해 접근 패턴에 따라 특정 데이터들을 GPU 메모리로 미리 prefetch하는 방식이 있다.

이후 3.에서는 GPU에서 발생하는 page fault에 대한 전체 처리 시간이 성능에 미치는 영향과 각 page fault 처리에 따른 오버헤드 그리고 전체 처리 과정 중 병목이 되는 지점을 분석한다.

3. 기존 모델의 성능 분석

그림 2는 12GB의 메모리를 가진 Nvidia GeForce Titan X GPU에서 UVM을 활용하여 Breadth-First Search (BFS)를 수행할 경우 나타나는 성능 변화를 보여주고 있다.

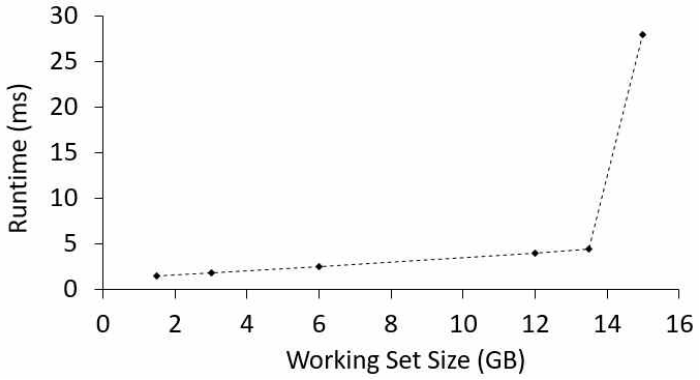

그림 2 워크로드의 메모리 요구량에 따른 성능 변화

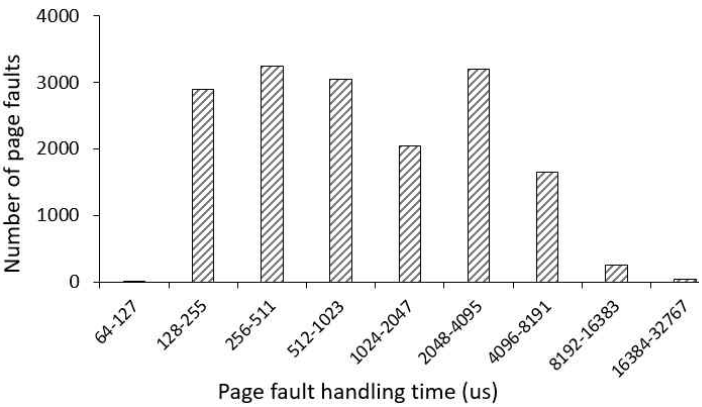

그림 3 각 page fault 별 요구되는 처리 시간 분포

이 실험은 BFS에서 요구하는 최대 메모리 크기를 15GB까지 변화시키며 수행하였다. 이 실험 결과에서 워크로드가 요구하는 메모리 사용량이 GPU 메모리의 크기를 벗어나지 않는 경우, 성능이 선형적으로 증가하는 것을 알 수 있다. 반면, 워크로드의 메모리 요구량이 GPU 메모리 크기를 초과하는 경우 성능이 급격하게 감소하는 것을 알 수 있다. 따라서, 이 실험 결과를 통해 GPU의 메모리 크기보다 더 높은 메모리 요구량을 처리하기 위해 UVM 기술이 적용되는 시점부터 성능이 감소한다는 것을 알 수 있다.

따라서 성능이 급격하게 감소하는 원인을 분석하기 위해 (1) 각 page fault가 발생할 때마다 수행되는 page fault 처리 시간과 (2) page fault 처리 과정에서 병목이 되는 지점을 분석하였다. 그림 3은 워크로드의 전체 수행 시간에서 발생하는 page fault 처리에 소요되는 시간에 대한 분포를 보여준다.

이 결과를 통해 대부분의 page fault가 처리를 위해 최소 64us부터 최대 32ms정도까지 걸린다는 것을 알 수 있다. 일반적으로 호스트 CPU에서 발생하는 page fault의 처리 시간인 약 20us인 반면 GPU에서 발생하는 page fault는 더 높은 처리 시간을 요구하는 것을 알 수 있다 [2].

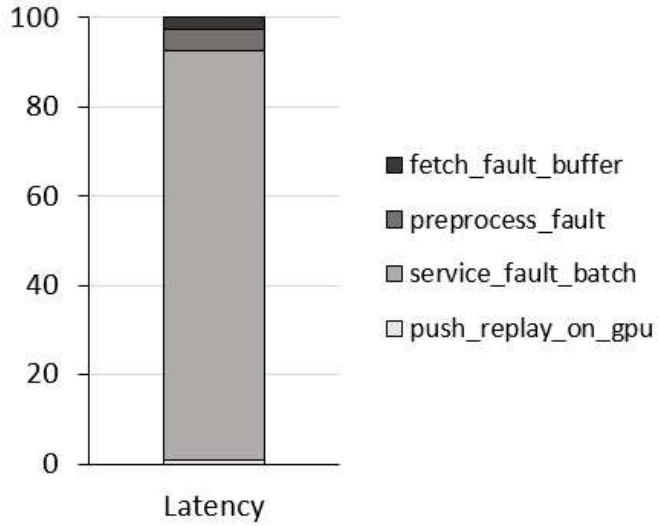

그림 4 전체 처리 시간에서 각 과정이 차지하는 비율

그림 4는 page fault 처리 과정에서 가장 병목이 되는 지점을 분석하였다. 그림 4를 통해 각 과정 중에서 실제로 page fault 요청을 처리하는 과정(service_fault_batch)이 전체 수행 시간에서 가장 많은 비중을 차지하는 것을 알 수 있다.

그 이유는 각 GPU마다 특정 호스트 CPU 코어에서 모든 page fault에 대해 처리하기 때문이다. 따라서, 각 page fault가 발생한 가상 주소에 따라서 독립적으로 처리가 가능하더라도 특정 CPU 코어 내에서 순서대로 처리된다.

4. 기존 모델의 호스트 자원 사용량 분석

그림 5는 UVM 기술에서 사용되는 호스트 CPU 사용량을 보여주고 있다. 그림 5를 보면 워크로드의 메모리 요구량이 GPU 메모리를 초과하여 호스트 메모리를 사용하는 시점부터 호스트 CPU 자원 사용량이 크게 증가하는 것을 관찰할 수 있다.

2. 에서 설명한 것처럼 interrupt를 통해 page fault 처리는 interrupt 처리 시간을 줄이기 위해 top-half와 bottom-half로 구분하여 동작을 수행하고, page fault에 대한 주요 처리 과정은 bottom-half에서 수행된다. 이를 위해 GPU 디바이스 드라이버는 각 GPU마다 특정 호스트 CPU 코어를 할당하여 해당 과정을 수행하는 쓰레드를 생성한다.

따라서 해당 GPU가 호스트 메모리에 접근하는 시점부터 특정 호스트 CPU 코어에 bottom-half를 수행하는 쓰레드가 생성되고, 해당 GPU에서 발생하는 page fault에 대한 요청들은 모두 해당 CPU 코어에서 처리된다.

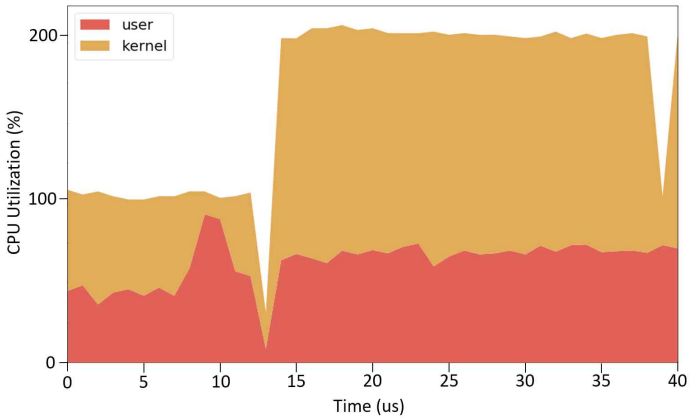

그림 5 UVM의 시간에 따른 CPU 사용량 변화

5. 결론 및 향후 연구 계획

본 연구에서는 호스트 CPU 메모리를 GPU 메모리로 사용하는 Unified Virtual Memory (UVM) 기술을 분석하였다.

UVM 기술은 GPU의 메모리보다 메모리 요구량이 큰 워크로드를 수행할 때 발생할 수 있는 사용자의 프로그래밍 부담을 줄이기 위해 소개되었다. 하지만 UVM 기술은 사용자의 프로그래밍 부담을 줄이는 반면 사용에 따른 성능 오버헤드와 호스트 CPU 자원 사용량을 크게 요구한다. 따라서, 이 기술을 효과적으로 사용하기 위해서는 이러한 성능 오버헤드와 호스트 자원 사용량을 개선할 수 있는 방향이 고려되어야 한다.

따라서 향후에는 본 논문에서 분석한 결과를 바탕으로 GPU의 메모리 확장과 관련된 기존 기술에서 발생하는 성능 및 호스트 자원 사용량에 대한 오버헤드를 감소시킬 수 있는 기술을 개발하여 평가하도록 한다.

참 고 문 헌

[1] Kim, Hyojong, et al. “Batch-aware unified memory management in GPUs for irregular workloads.“ Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020.

[2] Lee, Gyusun, et al. “A case for hardware-based demand paging.“ 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA) . IEEE, 2020.

[3] Milic, Ugljesa, et al. “Beyond the socket: NUMA-aware GPUs.“ Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 2017.