프로그래머를 위한 GPU 구조

김영석 김동규 이재원 김장우
포항공과대학교 고성능컴퓨팅연구실

연구 목적

현재 GPU 프로그래밍 모델들은 GPU의 DRAM 크기보다 더 큰 메모리를 할당할 수 없도록 제한한다. 이로 인한 대규모 GPU 프로그램 구현의 프로그래머 부담과 성능 손실을 분석하고, 이를 해결할 수 있는 컴퓨터 아키텍쳐 기법을 제시한다.

요약

✓ GPU의 DRAM 크기보다 더 큰 메모리를 필요로 하는 대규모 GPU 프로그램의 구현 부담을 분석함.
✓ 새로운 구현 방식으로 인한 프로그램 성능 저하를 분석함.
✓ GPU 프로그램의 특징에 기반한 DRAM 캐시, prefetching 및 pre-flushing 기법을 제안하고, 구현 부담 및 성능 손실을 해결할 수 있음을 보임.

연구 배경

✓ 대규모 GPU 프로그래밍을 위한 **프로그램의 재정의** [1]

소규모	대규모

`// Perform C = A * B on GPU`
`MemcpyCPUtoGPU(matA);`
`MemcpyCPUtoGPU(matB);`
`CalcMatMultAll(matA, matB, matC);`
`MemcpyGPUtoCPU(matC);`
	`// Perform C = A * B on GPU`
	`stream_t strm[rows * cols];`
	`for (i = 0; i < rows; i++) {`
	`  for (j = 0; j < cols; j++) {`
	`  int idx = i * cols + j;`
	`  CreateStream(&streams[idx]);`
	`  MemcpyCPUtoGPU(matA[i][...], strm[idx]);`
	`  MemcpyCPUtoGPU(matB[...][j], strm[idx]);`
	`  CalcMatMultOne(matA, matB, matC, i, j, strm[idx]);`
	`  MemcpyGPUtoCPU(matC[i][j], strm[idx]);`
	`  PerformStream(strm[idx]);`
	`  }`
	`}`
	`for (i = 0; i < rows; i++)`
	`  for (j = 0; j < cols; j++)`
	`  SynchronizeStream(strm[i * cols + j]);`

✓ 데이터 전송으로 인한 대규모 프로그램의 **성능 저하**

- **Small-scale**

- **Large-scale**

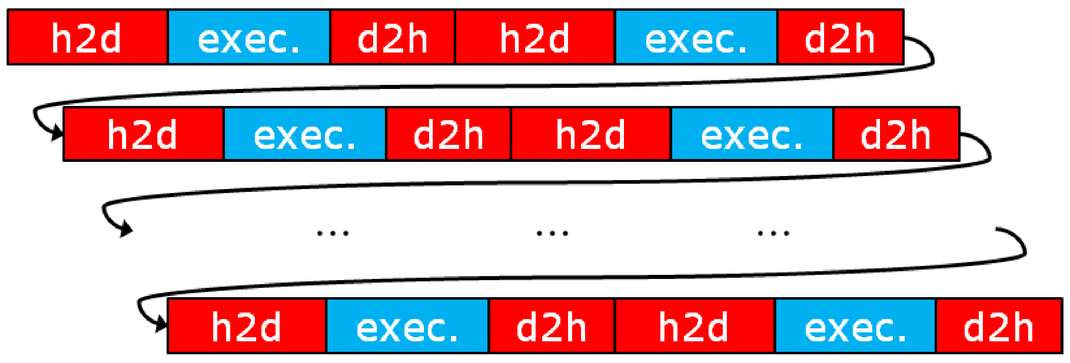

그림 1. 대규모 프로그램을 소규모 프로그램으로 변환을 하더라도 전송해야 할 데이터의 크기는 같기 때문에 성능 저하가 발생한다.

문제점

✓ 대규모 프로그램을 작은 용량의 메모리만 사용하는 소규모 프로그램으로의 재정의가 필요하다. [2]
✓ GPU 호출 및 메모리 복사 횟수 증가로 인해 성능 저하가 발생한다. [3]

목표

✓ 프로그램의 재정의 및 코드 수정이 없이도 대규모 프로그램을 실행할 수 있는 GPU 구조를 제안한다.
✓ 새로운 GPU 구조에서 실행되는 대규모 프로그램의 성능 저하를 최소화 하는 GPU 구조 및 소프트웨어적 기법을 제안한다.

연구 결과

✓ 대규모 GPU 프로그래밍을 위한 새로운 GPU 구조

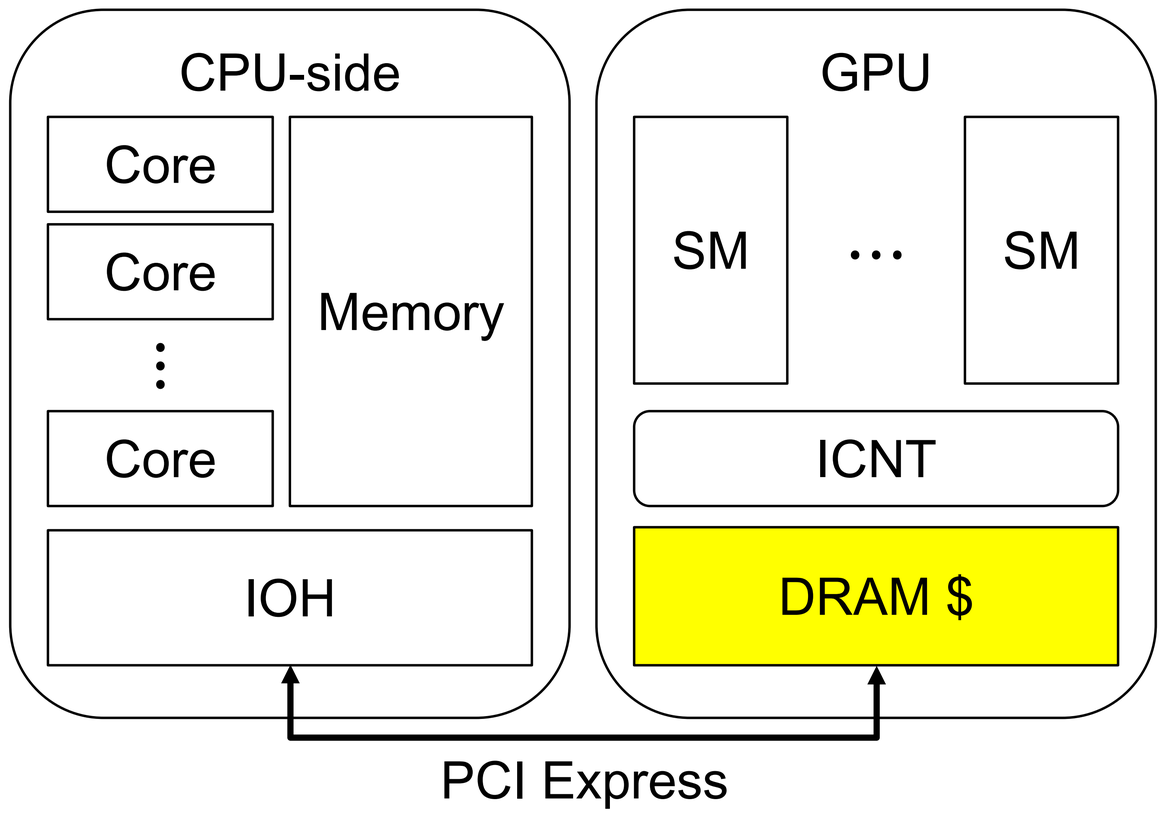

그림 2. 전형적인 CPU-GPU 아키텍쳐. CPU와 GPU는 PCI Express 버스를 통해서 데이터를 주고 받는다.

✓ DRAM 캐시
- PCI Express 링크는 접근하는 주소가 **순차적**이며, **최소한 1MB를 한 번에** 접근할수록 효율이 좋다.

✓ Prefetching
- GPU 쓰레드의 행동은 **쓰레드의 번호에 의존**한다. 정적 분석을 활용하면 각 쓰레드가 접근할 메모리 영역을 계산해낼 수 있다.

✓ Pre-flushing
- GPU는 write의 횟수가 CPU에 비해 상대적으로 적다. 이 특성에 기반한 pre-flushing 기법을 사용하면 성능을 높일 수 있다.

특장점

✓ DRAM 캐시로 인해 GPU의 DRAM 크기와 무관한 크기의 메모리를 사용할 수 있다.
✓ Prefetching과 pre-flushing으로 DRAM 캐시의 효율을 높이고, 메모리 복사로 인한 성능 손실을 줄일 수 있다.
✓ GPU 프로그램들의 특징에 기반한 구조이기에 일반적인 컴퓨터 아키텍쳐 기법을 적용하는 것 보다 더 적은 하드웨어 비용이 든다.

결론 및 향후 계획

✓ DRAM 캐시로 인해 GPU의 메모리 크기와 무관한 크기의 메모리를 사용할 수 있다.
✓ Prefetching과 Pre-flushing을 이용하여 DRAM 캐시의 효율과 메모리 전송으로 인한 시간을 감소시킬 수 있다.
✓ 정적 분석으로 알아내기 힘든 포인터와 간접 배열 접근 지원을 위한 컴퓨터 아키텍쳐 기법을 고안한다.

참고 문헌

[1] NVIDIA Corporation, *CUDA C Programming Guide*.
[2] R. Whu, B. Zhang, and M. Hsu, *Clustering Billions of Data Points Using GPUs*. In *UCHPC-MAW'09*.
[3] C. Gregg and K. Hazelwood, Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *ISPASS'11*.