최신 GPU 아키텍처의 데이터 캐시 성능 분석*

정종현 O1 오윤호 2 구건재 3
1,3 고려대학교 컴퓨터학과
2 성균관대학교 전자전기공학부
dida1245@korea.ac.kr, yunho.oh@skku.edu, gunjaekoo@korea.ac.kr

Analyzing Data Cache Performance of Modern GPU Architecture

Jong Hyun Jeong O1 Yunho Oh2 Gunjae Koo3
1,3Department of Computer Science and Engineering, Korea University
2Department of Electronic and Electrical Engineering, SungKyunKwan University

요 약

Graphics processing unit (GPU)는 그래픽 어플리케이션의 처리 뿐만 아니라 최근 machine learning, big data analytics 등의 대규모 병렬처리를 요구하는 어플리케이션의 처리에 널리 사용되고 있다. GPU 는 많은 수의 쓰레드를 동시에 실행하는 병렬처리 구조를 가지고 있다. 이 때문에 많은 메모리 요청이 단시간 내에 발생되어 메모리 계층구조의 자원이 소진되고 데이터 캐시가 비효율적으로 사용되는 문제가 있었다. 최신 GPU 아키텍처에서는 이러한 문제를 해결하기 위해 streaming cache 라고 불리는 새로운 캐시 구조를 적용하여 데이터 캐시의 성능 저하 요소를 줄였다. 본 연구에서는 최신 GPU 시뮬레이터를 사용하여 streaming cache 의 성능을 기존 캐시와 자세히 비교하여 streaming cache 의 특성을 밝히고 있다. 본 연구에서는 streaming cache 가 데이터 캐시의 congestion 은 해결하지만 memory congestion 이 interconnection network 단으로 옮겨갈 수 있음을 밝혀냈다. 본 연구를 통하여 streaming cache 가 최신 GPU 메모리 시스템에서 미치는 영향을 분석하여 최신 GPU 아키텍처의 메모리 시스템에서 발생할 수 있는 성능 문제점들에 대해서 제시한다.

1. 서 론

Graphics processing unit (GPU)는 그래픽 어플리케이션의 처리 뿐만 아니라 최근 machine learning, big data analytics 등의 대규모 병렬처리를 요구하는 어플리케이션의 처리에 널리 사용되고 있다. GPU 는 효과적인 병렬 처리를 위하여 하나의 커널에서 수백에서 수천 개의 쓰레드를 발생시켜서 이를 병렬적으로 처리하는 구조를 가지고 있다. 이를 위하여 GPU 는 하나의 streaming multiprocessor (SM) 내부에 수십개의 연산 엔진을 가지고 있으며 또한 fine-grained multithreading 기법을 사용하여 파이프라인 지연을 최소화하는 방식을 사용하고 있다. 다만 이렇게 많은 수의 쓰레드에서 생성되는 메모리 요청들을 GPU 의 메모리 계층구조에서 직렬화되어 처리가 될 수 밖에 없기 때문에 GPU 의 메모리 자원들은 빈번하게 자원 부족 현상을 겪으며 이는 GPU 에서 심각한 성능 저하 요소로 작용하고 있다.

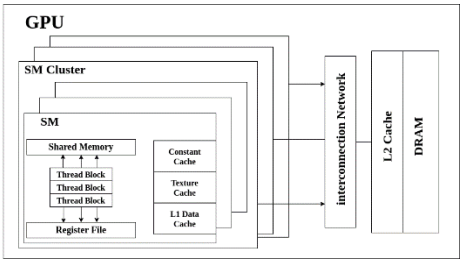

그림 1. GPU 아키텍처와 GPU 메모리 시스템

GPU 의 독특한 병렬처리 방식을 효과적으로 지원하기 위해 GPU 는 CPU 와는 다른 방식의 메모리 계층구조를 가지고 있다. GPU 는 하나의 SM 에 L1 데이터 캐시를 가지고 있으며 각각의 SM 은 interconnection network 를 통하여 L2 공유 캐시로 연결되어 있다. 또한 GPU 의 독특한 데이터 형태를 지원하기 위해 texture 캐시와 constant 캐시와 같은 캐시 메모리를 SM 이 포함하고 있다. 그리고 GPU 는 사용자가 직접 프로그래밍하여 쓰레드 사이에 데이터 공유에 사용할 수 있는 shared 메모리를 SM 내부에 지원하고 있다. 그렇지만 범용 프로그램에서는 GPU 의 데이터 캐시가 빈번하게 사용되고 있기 때문에 데이터 캐시에서의 자원 활용 방식은 GPU 의 성능에 큰 영향을 미친다.

GPU 의 데이터 캐시는 원래의 GPU 구조에서는 사용되지 않다가 GPU 를 이용해서 범용 어플리케이션을 처리하면서 사용되기 시작하였다. 이전의 GPU 구조에서는 CPU 와 비슷한 구조의 데이터 캐시 구조를 사용하였으나 캐시가 수십 개의 쓰레드에 공유되기 때문에 비효율적으로 사용되는 문제가 있다. 최근 NVIDIA 에서는 새로운 Volta 아키텍처에서 streaming cache 라고 불리는 새로운 캐시 구조를 적용하여 데이터 캐시에서의 성능 저하 요소를 줄이고자 하였다 [1]. Streaming cache 는 기본적으로 작은 크기의 메모리 요청을 지원할 수 있는 sector cache 형태를 가지고 있으며, 수많은 메모리 요청에 따른 자원 부족 현상을 피하기 위하여 miss status handling register (MSHR)를 크게 확장한 특징을 가지고 있다. 이를 통하여서 비정형적 특성을 가지고 있는 메모리 요청에 의한 자원 점유 현상을 줄일 수 있으며, 다량의 cache miss 로 인한 데이터 캐시의 자원 부족으로 인한 파이프라인 지연 현상을 완화할 수 있을 것이라 예상되고 있다.

본 연구에서는 cycle-accurate 시뮬레이터인 GPGPU-Sim [2] 최신 버전을 사용하여 최신 GPU 아키텍처에 적용된 streaming cache

* 이 성과는 2021 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2021R1C1C1012172)

구조의 특성을 밝히고자 한다. GPGPU-Sim 은 이전 아키텍처인 Fermi 아키텍처를 기반으로 개발이 되었으며 최근에 새로운 버전을 공개하면서 NVIDIA 의 최신 아키텍처 구조를 반영할 수 있도록 설계되었다 [3]. 본 연구에서는 최신 아키텍처에 적용된 streaming cache 의 성능을 기존의 캐시 구조의 성능과 자세하게 비교하여 그 특성을 밝히고 새로운 GPU 아키텍처의 메모리 구조에서 전체 성능에 크게 영향을 줄 수 있는 부분을 밝히고자 한다.

2. 실험 환경

본 논문에서는 이전 아키텍처의 기존 캐시 (normal cache)와 Volta 아키텍처부터 적용된 streaming cache 의 성능을 비교하였다. 이를 위하여 GPU 의 코어 및 L2, 메모리 부분은 아래와 같이 Turing 아키텍처 기반의 RTX2060 과 유사한 설정을 적용하였다.

표 1. GPU 공통 기본 구성 변수

Core	30 SMs, 64 CUDA cores / SM
Warp	32 Warps / SM
CTA	32 CTAs / SM
Register file	64KB / SM
Shared memory	64 KB / SM
L2 Cache	3MB, 128B line, 16 way, 192 MSHR entries
DRAM	GDDR6
ICNT topology	52 × 1 butterfly
ICNT peak BW	786.24GB/s

표 2. Normal cache 와 streaming cache 의 구성 변수

	Normal Cache	Streaming Cache
L1D cache	64 KB, 128set, 128B line 4 way	64 KB, 1set, 128B line, 4 sector 512 way
L1D latency	27	20
MSHR entry	256	2048
MSHR max merge	8	32
Allocation policy	on fill	on fill

표 1 은 GPGPU-Sim 설정에 사용한 GPU 하드웨어 설정을 보여준다. 이번 연구에서는 SM, interconnection network, L2 캐시 및 메모리는 동일하게 설정한 상태에서 L1 데이터 캐시를 기존의 캐시 구조 (normal cache)와 새로운 캐시 구조 (streaming cache)로 다르게 설정하여 성능을 비교하였다. Streaming 캐시는 하나의 캐시 블록이 4 개의 sector 로 나누어진 sector cache 구조를 가지고 있다. 또한 shared memory 와 통합된 구조를 사용함으로서 낮은 latency 를 가지고 있다. Streaming cache 는 각각의 sector 에 대해서 개별 MSHR entry 를 가지고 있으며 총 2048 개의 MSHR entry 를 지원한다. 또한 이미 cache miss 에 대응한 요청이 MSHR entry 에 존재할 경우 최대 32 개의 동일한 메모리 주소에 대한 요청을 통합할 수 있다. (이는 하나의 warp 에서 수행하는 쓰레드의 숫자와 같다.) 서로 다른 캐시 구조의 성능을 측정하기 위해서 우리는 널리 사용되는 GPU 벤치마크인 CUDA SDK [4], Rodinia [5], Polybench [6] 및 Parboil [7]에서 41 개의 어플리케이션을 실험에 사용하였다.

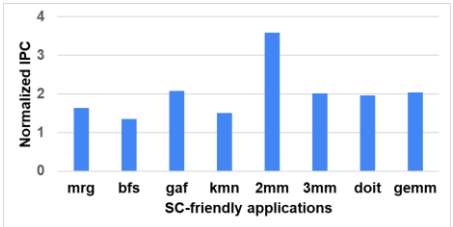

그림 2. streaming cache 적용에 따른 성능 변화

그 중에서 streaming cache 를 적용했을 때에 더 좋은 성능을 보여주는 8 개의 어플리케이션 (SC-friendly)을 선정하여 더욱 자세하게 분석하였다. 그림 2 는 normal cache 에서의 성능을 1 로 정규화했을 때, streaming cache 에서 성능이 향상되는 8 개의 어플리케이션의 성능 변화를 보여주고 있다.

3. 실험 결과

그림 3. Streaming cache (S)와 normal cache (N)에서 요청 처리 분포

그림 3 은 streaming cache 와 normal cache 구조를 적용했을 때 데이터 캐시에 대한 요청 처리 결과 분포를 보여준다. 앞에서 언급한대로 streaming cache 는 하나의 캐시 블록을 4 개의 sector 로 나누어서 요청되는 데이터의 크기가 작을 경우 작은 크기의 데이터 요청을 지원한다. 캐시 미스가 발생할 경우 온전한 블록 크기 만큼의 데이터를 요구하는 normal cache 에 비해서 streaming cache 는 보다 작은 단위인 sector 단위의 데이터를 요구하기 때문에 사용하는 데이터 대역폭을 줄일 수 있는 장점이 있다. 위의 실험 결과를 보면 streaming cache 를 적용했을 때와 normal cache 를 사용했을 때의 cache miss rate 는 큰 차이를 보이지 않는 것을 알 수 있다. 도리어 gaf 와 kmn 을 제외하고 나머지 어플리케이션에 대해서는 streaming cache 를 적용할 경우 miss rate 가 더 증가하는 것을 볼 수 있다. 그렇지만 streaming cache 를 적용했을 때 위와 같은 어플리케이션에서 성능이 향상되는 이유는 대량의 메모리 요청이 있을 때에 streaming cache 가 더 효율적으로 자원을 할당할 수 있기 때문이다. 즉, 데이터 캐시의 자원 이용이 불가능해져 발생하는 reservation fail 의 빈도가 streaming cache 를 사용할 경우 크게 줄어들게 되므로 reservation fail 에 따른 파이프라인 지연이 줄어들게 된다. 이에 대한 분석 결과는 이후 자세하게 소개할 것이다.

표 3. 데이터 캐시의 reservation fail 빈도 비교

	Normal cache	Normal cache + larger MSHR	Stream
ICNT injection queue full	30151 (0.01%)	297013 (3.72%)	461462 (100%)
MSHR entry full	96771331 (0.18%)	0	0
MSHR merge full	422051937 (99.82%)	7678751 (96.28%)	0
Fail per access	1.71	0.03	0.0017

표 3 은 실험에 사용한 어플리케이션에 대해서 normal cache, streaming cache 및 normal cache 에서 MSHR 의 entry 개수를 streaming cache 만큼 확장시켰을 때의 reservation fail 원인 별 비율과 발생 횟수, 그리고 메모리 요청 당 reservation fail 의 빈도를 비교한 것이다. Reservation fail 은 데이터 캐시에서 miss 요청을 할당할 수 있는 자원이 부족할 때 발생하며 데이터 캐시가 더 이상 메모리 요청을 받지 못하게 되므로 파이프라인 지연을 발생시키다. Streaming cache 에서는 모든 sector 에 대한 개별 MSHR entry 가 존재하므로 MSHR entry 의 부족에 따른 reservation fail 이 발생하지 않는다. 또한 normal cache 보다 MSHR merge 가 가능한 요청의 수가 증가하였기 때문에 이에 따른 reservation fail 또한 발생하지 않음을 알 수 있다. 반면에 normal cache 구조에서는 MSHR merge 가 적은 수만 허용되기 때문에 이에 따른 reservation fail 이 빈번하게 발생함을 알 수 있다. 그 결과 streaming cache 를 적용할 경우 normal cache 에 비해 메모리 요청당 reservation fail 의 발생이 99.9% 감소한다. 그렇지만 이는 데이터 캐시에서 매우 많은 수의 메모리 요청이 interconnection network 로 전달된다는 것을 의미하고, 그렇기 때문에 interconnection network 의 혼잡도를 증가시키게 된다. 즉, 표 3 에서 보는 바와 같이 streaming cache 를 적용할 경우

interconnection network 의 injection queue 가 가득차서 발생하는 reservation fail 이 15 배 증가하게 된다. 또한 이로 인하여 interconnection network 의 bandwidth 가 증가할 것이라고 예상할 수 있다.

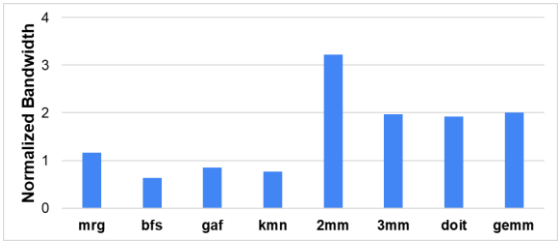

그림 4. Interconnection network bandwidth

그림 4 는 normal cache 에서의 interconnection network bandwidth 를 1 로 정규화했을 때 streaming cache 를 적용할 경우의 bandwidth 를 보여주고 있다. bfs, kmn 및 gaf 와 같이 비정형적인 메모리 접근 특성을 가진 어플리케이션의 경우 sector cache 구조를 가지고 있는 streaming cache 에서 보다 작은 크기의 데이터 이동이 가능하기 때문에 bandwidth 가 감소하는 경향을 보인다. 그렇지만 나머지 어플리케이션의 경우 데이터 캐시에서의 reservation fail 이 감소되면서 메모리 요청이 단시간 내에 모두 interconnection network 로 몰리면서 전체적인 bandwidth 가 증가하는 것을 볼 수 있다.

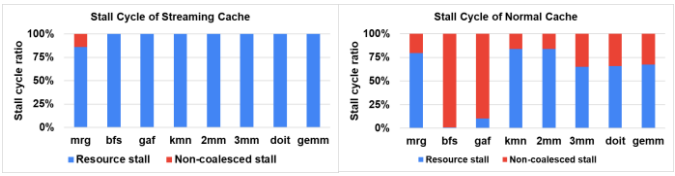

그림 5. 원인 별 stall cycle 분포

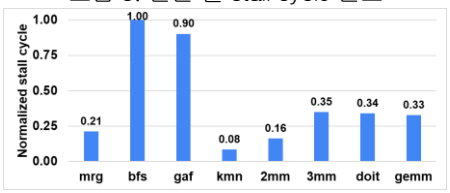

그림 6. Streaming cache 에서 stall cycle

그림 5 는 streaming cache 와 normal cache 에서 발생하는 memory stage 파이프라인 지연의 원인 별 분포를 보여준다. Resource stall 은 메모리에 데이터가 없어서 발생하는 파이프라인 지연이고 non-coalesced stall 은 메모리 요청이 합병하기 위해 대기하면서 발생하는 stall 이다. Streaming cache 의 sector 구조로 인해 보다 작은 단위로 메모리 요청을 할 수 있기 때문에 메모리 요청의 합병을 기다리지 않아도 되어 non-coalesced stall 은 대부분 사라지고 메모리에 데이터가 없어서 발생하는 resource stall 이 대부분의 지연 발생 원인이 된다. 그림 6 은 normal cache 에서 발생한 stall cycle 을 1 로 정규화 했을 때, streaming cache 의 stall cycle 을 보여준다. bfs 와 gaf 는 non-coalesced stall 이 줄어든 만큼 resource stall 이 증가하여 전체 memory stage stall cycle 에 큰 차이가 없지만 나머지 어플리케이션들은 streaming cache 로 인해 non-coalesced stall 이 제거되어 stall cycle 자체가 감소하면서 성능 이득을 얻을 수 있다.

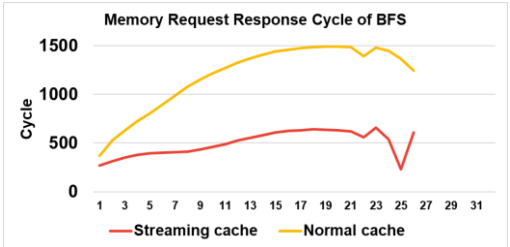

그림 7. 메모리 요청 latency 비교

그림 7 은 bfs 어플리케이션의 메모리 요청 수에 따른 메모리 요청 응답 대기시간의 변화를 보여준다. bfs 는 비정형적인 메모리 요청 특성으로 인해 하나의 warp 에서 매우 많은 수의 메모리 요청이 발생한다. streaming cache 의 sector cache 구조는 보다 작은 단위로 메모리 요청이 가능하여 이와 같이 비정형적인 메모리 요청에서 유리하게 작용한다. 이로 인해 normal cache 보다 훨씬 낮은 메모리 요청 응답 시간을 가지게 되고 이는 비슷한 특성을 가지는 어플리케이션의 성능 향상으로 이어진다.

4. 결론 및 향후 연구

Volta 아키텍처부터 새롭게 적용된 데이터 캐시 디자인인 streaming cache 는 sector 구조로 이루어져 보다 작은 단위로 메모리 요청이 가능하여 비정형적인 메모리 요청이 많은 어플리케이션에서 유리하게 동작한다. 또한 확장된 MSHR 로 데이터 캐시의 reservation fail 을 99.9% 제거하면서 cache congestion 문제를 해결하여 normal cache 보다 평균적으로 더 성능을 향상시킬 수 있음을 확인하였다. 하지만 interconnection network 로 전달되는 메모리 요청이 급격하게 증가하면서 interconnection network 의 자원 부족 현상이 빈번하게 발생하여 이로 인한 reservation fail 이 빈번하게 발생하고, 일부 어플리케이션에서는 interconnection network 의 bandwidth 가 급증할 수 있다는 것이 확인되었다. 따라서 기존에 데이터 캐시로 인해 발생하던 congestion 이 완전히 해결된 것이 아니라 interconnection network 단으로 이동하였다고 할 수 있다. streaming cache 로 인해 가중된 Interconnection network 의 congestion 을 해결하기 위한 캐시 구조와 메모리 시스템 구조는 앞으로 연구할 과제이다.

참 고 문 헌

[1] J. Choquette, O. Giroux and D. Foley, "Volta: Performance and Programmability," in IEEE Micro, vol. 38, no. 2, pp. 42-52, Mar./Apr. 2018.

[2] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 163-174, 2009.

[3] M. Khairy, Z. Shen, T. M. Aamodt and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 473-486, 2020.

[4] CUDA C/C++ SDK Code Samples, 2015, [online] Available: http://developer.nvidia.com/ cuda- cc- sdk- code- samples.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, et al., "Rodinia: A Benchmark Suite for Heterogeneous Computing", Proceedings of IEEE international symposium on workload characterization (IISWC), pp. 44-54, 2009.

[6] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula and J. Cavazos, "Auto-tuning a High-Level Language Targeted to GPU Codes", Innovative Parallel Computing (InPar), pp. 1-10, 2012.

[7] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, et al., "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing", Center for Reliable and High-Performance Computing, vol. 127, 2012.