KIPS Trans. Comp. and Comm. Sys.
Vol.8, No.5 pp.111~118 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2019.8.5.111

MSHR-Aware Dynamic Warp Scheduler for High Performance GPUs

Gwang Bok Kim† · Jong Myon Kim†† · Cheol Hong Kim†††

ABSTRACT

Recent graphic processing units (GPUs) provide high throughput by using powerful hardware resources. However, massive memory accesses cause GPU performance degradation due to cache inefficiency. Therefore, the performance of GPU can be improved by reducing thread parallelism when cache suffers memory contention. In this paper, we propose a dynamic warp scheduler which controls thread parallelism according to degree of cache contention. Usually, the greedy then oldest (GTO) policy for issuing warp shows lower parallelism than loose round robin (LRR) policy. Therefore, the proposed warp scheduler employs the LRR warp scheduling policy when Miss Status Holding Register(MSHR) utilization is low. On the other hand, the GTO policy is employed in order to reduce thread parallelism when MSHRs utilization is high. Our proposed technique shows better performance compared with LRR and GTO policy since it selects efficient scheduling policy dynamically. According to our experimental results, our proposed technique provides IPC improvement by 12.8% and 3.5% over LRR and GTO on average, respectively.

Keywords : GPU, Warp Scheduling, Cache, MSHR, Parallelism

GPU 성능 향상을 위한 MSHR 활용률 기반 동적 워프 스케줄러

김 광 복† · 김 종 면†† · 김 철 홍†††

요 약

GPU는 병렬처리가 가능한 강력한 하드웨어 자원을 기반으로 높은 처리량을 제공한다. 하지만 과도한 메모리 요청이 발생하는 경우 캐쉬 효율이 낮아져 GPU 성능이 크게 감소할 수 있다. 캐쉬에서의 경합이 심각하게 발생한 경우 동시 처리되는 스레드의 수를 감소시킨다면 캐쉬에서의 경합이 완화되어 전체 성능을 향상시킬 수 있다. 본 논문에서는 캐쉬에서의 경합 정도에 따라 동적으로 병렬성을 조절할 수 있는 워프 스케줄링 기법을 제안한다. 기존 워프 스케줄링 정책 중 LRR은 GTO에 비해 워프 수준의 병렬성이 높다. 따라서 제안하는 워프 스케줄러는 L1 데이터 캐쉬 경합 정도를 반영하는 MSHR(Miss Status Holding Register)이 낮은 자원 활용률을 보일 때 LRR 정책을 적용한다. 반대로 MSHR 자원 활용률이 높을 때는 워프 수준의 병렬성을 낮추기 위해 GTO 정책을 적용하여 워프 우선순위를 결정한다. 제안하는 기법은 동적으로 스케줄링 정책을 선택하기 때문에 기존의 고정된 LRR과 GTO에 비해 높은 IPC 성능과 캐쉬 효율을 보여준다. 실험 결과 제안하는 동적 워프 스케줄링 기법은 LRR 정책에 비해 약 12.8%, GTO 정책에 비해 약 3.5% IPC 향상을 보인다.

키워드 : 그래픽 처리장치, 워프 스케줄링, 캐쉬, MSHR, 병렬성

1. 서 론

GPU는 강력한 하드웨어 자원을 활용하여 병렬처리 연산을 지원함으로써 기존 CPU보다 훨씬 높은 처리량을 보여줄 수 있다. 또한 기존의 GPU에서 처리하던 그래픽 연산뿐만 아니라 범용 연산에 활용하는 GPGPU 기술이 보편화되고 있다. GPU는 단일 명령어로부터 다중 데이터를 처리하는 구조인 SIMD(Single Instruction Multiple Data) 구조에서의 실행을 위해 GPU의 하드웨어 자원에서 동시 수행 가능한 수만큼 스레드를 집단화한다. GPU는 다수의 프로세서인 SM(Streaming Multiprocessor)으로 구성된다. SM은 스레드들의 집합 CTA(Cooperative Thread Array)를 CTA 스케줄러로부터 할당받아 명령어를 수행한다. CTA의 스레드들은 동시 수행 가능한 처리 폭(SIMD width) 만큼의 스레드 집합 크기인 워프(Warp) 단위로 구성되어 실행된다. SM에는 명령어를 처리하기 위한 자원이 한정되어 있고, 워프 스케줄러는 클럭 사이클에 따라 일정하게 수행 가능한

※ 본 연구는 한국연구재단 이공분야기초연구사업 중견연구자지원사업의 지원으로 수행되었음(NRF-2018R1A2B6005740). 또한 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음(IITP-2018-2016-0-00314).

† 준 회 원 : 전남대학교 전자컴퓨터공학부 박사과정
†† 정 회 원 : 울산대학교 IT융합학부 교수
††† 종신회원 : 전남대학교 전자컴퓨터공학부 교수

Manuscript Received : November 15, 2018
First Revision : January 4, 2019
Second Revision : January 31, 2019
Accepted : February 12, 2019

* Corresponding Author : Cheol Hong Kim(chkim22@jnu.ac.kr)

워프를 선택하여 이슈한다. 워프를 이슈하기 위한 우선순위 정책은 자원 활용률과 연결되기 때문에 GPU 성능에 큰 영향을 준다. 또한 워프 스케줄링 순서는 메모리 접근 순서로 이어지므로 데이터 지역성에 직접적인 영향을 미친다. 각 스레드는 파이프라인에서의 머무르는 사이클이 서로 다르기 때문에 자원 상황과 메모리 명령어 집중도를 고려한 스케줄링은 다른 동시 수행 가능한 스레드의 불필요한 대기시간을 줄일 뿐만 아니라 여러 가지 하드웨어 자원들의 활용률을 높일 수 있다. 따라서 GPU 성능 향상을 목적으로 CTA/워프 스케줄링에 관한 연구가 진행되고 있다[1-6]. GPU의 성능을 향상시키기 위해서 제안된 스케줄링 관련 연구 중 다수는 GPU 캐쉬와 메모리 사이에서 발생하는 불필요한 대기시간을 줄이거나 대기시간 동안 다른 자원들을 활용함으로써 GPU의 처리량을 높인다. 캐쉬의 관점에서 프로세서를 비교한다면 현대 CPU의 캐쉬 구조는 메모리 레이턴시 감소가 중요한 반면 GPU의 경우 다수의 스레드 경합으로 인한 잦은 블록 교체로 인해 발생하는 캐쉬 비효율성이 성능의 주요 요인 중 하나이다. GPU에서 자주 발생하는 문제로 재사용이 되지 않는 스트리밍 데이터가 캐싱되어 다른 재사용 가능한 데이터를 교체시키거나 워킹 셋(Working Set)의 크기가 캐쉬 크기에 비해 커서 재사용되기 전에 교체되는 문제가 있다. GPU는 다수의 스레드들이 동시에 접근되도록 허용하기 때문에 캐쉬 블록들이 짧은 시간 안에 빈번하게 교체되고, 이에 따라 캐쉬에서의 각 스레드별 캐쉬 저장 시간은 매우 짧아진다. 다수의 워프들 사이에서 발생하는 경합은 지역성을 저해하고 이로 인해 성능 감소가 발생한다. 또한 GPU에서 수행되는 다수의 벤치마크의 워킹 셋은 일반적으로 캐쉬 크기보다 크기 때문에 CPU 관련 연구에서 사용된 캐쉬 교체 정책으로는 GPU에서의 캐쉬 경합 문제를 완화하기 어렵다.

본 논문에서는 SM에서의 동시 수행되는 스레드의 증가로 인한 메모리 집중 정도에 따라 워프의 우선순위 정책을 동적으로 변경하여 캐쉬 효율을 높이고 전체 성능을 향상시킬 수 있는 워프 스케줄링 기법을 제안한다. 각 벤치마크는 메모리 경합(Memory Contention)와 L1 데이터 캐쉬에서의 미스 발생율이 서로 상이하며 성능 향상에 유리한 워프 스케줄링 정책이 다르므로 커널(kernel)이 실행되어 초기 수행되는 명령어의 수행 결과를 통해 이후 사이클 동안 적합한 워프 스케줄링 정책을 적용한다. GPU 구조에서 사용되는 워프 스케줄링 정책 중 비교적 간단한 하드웨어와 함께 높은 성능을 보여주어는 LRR(Loosely Round Robin) 정책과 GTO(Greedy Then Oldest) 정책을 명령어 수행 중에 동적으로 적용함으로써 기존 정책에 비해 GPU 성능을 향상시킬 수 있는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 GPU에서의 워프 스케줄링과 관련된 최신 연구들에 대해 상세히 서술하고 GPU의 기본 구조를 설명한다. 3장에서는 본 논문에서 제안하는 동적 워프 스케줄링 기법과 이를 지원하기 위한 아키텍처에 대해 상세히 기술한다. 4장에서는 실험 환경과 실험에 사용된 다양한 벤치마크에 관해 설명과 함께 제안하는 동적 워프 스케줄러의 성능 결과 분석을 보여준다. 마지막으로 5장에서는 본 논문의 결론을 맺는다.

2. 연구 배경

2.1 관련 연구

SM 수준에서의 각 워프 스케줄러는 32개(NVIDIA Fermi 구조 기준)의 스레드를 가지는 워프를 수행하기 위해 준비된 워프를 선택하여 실행 유닛에 이슈하는 역할을 한다. 워프 실행 순서를 결정하는 워프 스케줄러는 기존 연구들에서도 활용되는 바와 같이 GPU의 성능에 큰 영향을 미치는 요소이다[1-6]. 요구하는 하드웨어가 크지 않으면서도 비교적 간단한 워프 스케줄링 정책으로는 LRR와 GTO가 대표적이다. LRR(Loosely Round-Robin) 스케줄링 정책은 모든 하드웨어적인 워프들의 위치를 기준으로 공평하게 이슈될 수 있는 기회를 갖도록 한다. 워프들은 각 워프 아이디를 이용하여 순차적으로 워프 스케줄러에 의해 선택되고 만약 해당 사이클에서 이슈될 워프가 이슈할 수 없는 상태라면 다음 순서의 워프를 실행하는 정책이다. GTO (Greedy-Then-Oldest) 스케줄링 정책은 스톨을 발생시키지 않는 워프에 대해 우선적으로 이슈하는 정책이다. 또한 스톨 발생 시 준비된 워프들 중 SM에 가장 먼저 할당된 워프를 우선적으로 수행한다. GTO 정책은 GPU에서 수행되는 많은 벤치마크에서 우수한 성능 향상을 보여준다. 일반적으로 CUDA[7]를 기반으로 작성된 벤치마크들은 다수의 워프 사이에서 발생하는 지역성보다 단일 워프 안에서의 지역성을 잘 활용할 수 있는 GTO 방식의 워프 스케줄링 정책이 우수하다. 하지만 일부 벤치마크에서는 상대적으로 쓰레드 병렬성을 높일 수 있는 LRR 정책을 적용할 때 오히려 성능을 향상시킨다.

iPAWS[8]는 GPU가 벤치마크를 수행할 때 커널 단위별 특성을 반영하여 동적으로 워프를 이슈한다. GTO 워프 스케줄링에 적합한 워크로드를 수행할 때 스톨을 발생시키지 않는 일부 워프만이 지속적으로 수행된다. 따라서 전체 워프의 이슈 완료율은 비균일한 접근 패턴을 보이며 워프별 이슈 완료율을 통해 유형을 분류한다. 만약 워프마다 균일한 이슈가 발생한다면 LRR 정책에 유리한 워크로드로 판단하여 데이터 지역성 측정기간 이후에는 새로 적용된 워프 스케줄링 정책을 사용한다. 따라서 측정된 워프 이슈 패턴에 따라 LRR 정책과 GTO 정책 중 최선을 선택함으로써 GPU의 성능 향상을 이끈다. 또한 배리어 명령어 간섭으로 인해 명령어 이슈 패턴 분석이 잘못되는 경우를 보정하기 위한 추가적인 기법을 사용한다.

CCWS[9] 워프 스케줄러는 메모리 시스템으로부터의 피드백에 대한 동적인 워프 스로틀링을 수행한다. CCWS는 캐쉬에서 지역성을 활용하지 못하는 경우 동시 수행 가능한 워프의 수가 과도하다고 판단하여 동시 수행가능한 대기 워프의 수를 제한한다. CCWS는 최근 L1 데이터 캐쉬로부터 교체된 블록의 태그들을 저장하고 교체 태그에서 히트 발생 여부에 따라 지역성 손실도를 측정하고 지역성 손실이 크게 발생한 워프를 다음 사이클부터 우선적으로 이슈한다. 캐쉬에서의 데이터 지역성 측정은 워킹셋 크기와 스레드 경합에 따라 한정된 캐쉬 크기 내에서 측정이 어렵다. 따라서 교체된 태그

정보로부터 각 워프에 대한 지역성 정보를 모니터링해야 하므로 추가적인 하드웨어 오버헤드가 요구된다.

Zhang[10]은 L1 데이터 캐쉬에 요청하는 데이터가 없는 경우 미스 정보를 MSHR에 저장하고 동일한 캐쉬 블록을 요청하는 미스가 추가적으로 발생하는 경우 병합되는 기존 구조를 활용한다. 즉 하위 메모리로부터 데이터가 L1 데이터 캐쉬에 전달되기 전에 다른 스레드가 동일한 캐쉬 블록에 대해 미스를 발생시킨다면 요청의 수는 하나로 결합시켜 내부 연결망 대역폭을 줄이고 하위 메모리에 대한 접근을 줄인다. 이 기법은 MSHR의 엔트리 중 가장 많은 결합(merging) 요청에 우선순위를 두어 하위 메모리로 보내기 위한 요청 순서를 재정렬한다. 따라서 상대적으로 많은 스레드가 데이터를 전달받을 수 있는 데이터를 우선적으로 이슈하므로 전체적인 성능을 향상시킬 수 있다.

2.2 기존 GPU 구조

본 논문에서의 GPU 구조는 NVIDIA사의 Fermi구조를 대상으로 기술한다. GPU는 SIMD 방식의 연산 수행이 가능한 프로세서인 SM(Streaming Multi Processor)으로 구성된다. 각 SM에서는 피연산자가 준비된 워프가 워프 스케줄러에 의해 선택되고 명령어 수행에 필요한 유닛에 전달된다. 스레드들은 32개로 집단화되어 워프 단위로 생성되는데 워프에 속한 각 스레드들은 활성화 마스크를 참조하여 락-스텝(Lock-step) 방식에 따라 한 번에 하나의 명령어에 대해 동시 수행된다.

각 SM은 명령어를 수행하기 위한 다수의 스레드들을 가지고 있고 워프 스케줄러는 이러한 워프들 중 이슈가 가능한 준비된 워프들 중 하나를 선택하여 매 사이클마다 이슈한다. 이러한 SM에서 동시 수행될 수 있는 워프의 개수는 주로 각 SM의 레지스터의 크기뿐만 아니라 각 스레드들의 요청하는 레지스터의 수는 물론 그외 I-buffer, 스코어보드 등 동시 수행 가능한 스레드들의 정보 저장을 지원하는 하드웨어의 크기에 따라 제한된다. 각 워프는 인출 유닛으로부터 PC(Program Counter)를 이용하여 다음에 수행할 명령어를 인출하고, 해석 단계를 거쳐 명령어 버퍼(Instruction Buffer)에 엔트리에 해석된 정보를 저장한다. 명령어 버퍼에 있는 각 슬롯은 명령어의 유효 여부를 나타내는 비트를 가진다. 또한 이외 레지스터를 사용하는데 문제가 없을 경우 준비 상태를 나타내는 비트를 통해 관리하고 있다. 명령어는 데이터 의존성을 발생시키는 명령어가 실행이 완료되고 쓰기가 완료된 레지스터 데이터를 읽을 수 있는 상태가 된다. 따라서 준비 비트를 갱신한다. 워프 스케줄러는 이러한 워프들 중 하나를 선택하여 실행 유닛에 전달하여 실행되도록 한다. 이때 하나의 명령어는 해당 워프의 스택에서 최상위에 해당하는 엔트리로부터 활성화 마스크를 전달받아 활성화 스레드를 구분한다. 이슈된 명령어는 레지스터 파일로부터 피연산자를 인출하여 ALU 또는 메모리 접근 등 필요한 파이프라인에서 연산을 수행하는데 사용한다. 실행이 완료된 명령어는 연산 결과를 레지스터 파일에 쓰고 스코어보드를 갱신한다. 또한 명령어 버퍼에서의 해당하는 명령어의 준비 비트를 갱신하고 이슈 로직(Issue Logic)이 다음 사이클에 수행할 워프를 선택하도록 한다. 하나의 SM은 클럭마다 16개의 스레드들에 대한 데이터를 쓸 주소를 계산한다. Fig. 1은 GPU의 파이프라인을 보여준다. 메모리 명령어를 수행하는 경우 LSU를 통해 처리한다. 메모리로부터 데이이터를 가져오는데 수백 사이클이 소모되므로 이러한 접근 레이턴시를 최소화하기 위해 각 SM이 공유할 수 있는 L2 캐쉬와 각 SM이 글로벌 메모리와 로컬 메모리를 캐싱할 수 있는 L1 데이터 캐쉬를 가진다. 따라서 만약 L1 데이터캐쉬와 같은 온-칩(on-chip) 메모리에 대한 요청이 적중 실패한 경우 상호 연결망(Interconnection Network)을 거쳐 하위 메모리로 데이터 요청이 전달된다.

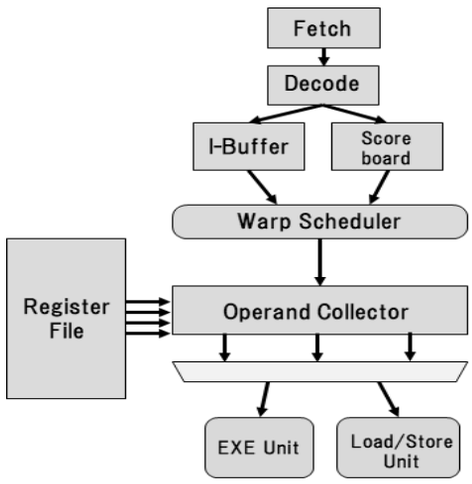

Fig. 1. Streaming Multiprocessor Pipeline

3. MSHR 활용률 기반 동적 워프 스케줄러

3.1 중점 설계 고려사항

GPU는 다수의 SM이 접근할 수 있는 글로벌 메모리를 캐싱하여 레지스터 파일에 빠르게 제공하기 위해 온-칩 L1 데이터 캐쉬를 사용한다. 캐쉬의 효율성은 데이터의 재사용성과 함께 메모리 경합에 따라 달라진다[11, 12]. 캐쉬에 접근하는 메모리 요청은 워프 스케줄러에 의해 이슈된 순서로 LD/ST 유닛의 큐에 전달되어 순차적으로 처리한다. 따라서 발생한 미스 요청을 처리할 수 있는 자원이 부족하다면 다른 메모리 요청을 수행하지 않고 미스 요청이 처리 가능할 때까지 매 사이클 반복적인 요청을 수행한다. 메모리 접근 집중도는 수행되는 워크로드의 특징으로부터 가장 큰 영향을 받지만 워프 스케줄러 및 관련 하드웨어 자원 상태에 따라 메모리 접근 집중도는 조절될 수 있다. 동시 수행되는 스레드 수가 증가하고 이에 따라 메모리 요청도 증가한다면 한정된 캐쉬 공간을 사용하는 각 스레드별 공간과 시간은 줄어들게 된다. 따라서 잦은 교체가 발생하거나 미스를 발생시키는 요청에 대한 처리를 위한 자원들의 한계에 따라 빠른 접근 레이턴시를 위한 온-칩 캐쉬의 이점은 줄어든다. 높은 처리량을

위해 고안된 아키텍처인 GPU는 캐쉬 효율이 낮을 경우 L2 또는 DRAM 에 대한 접근이 빈번해지고 이에 따라 준비된 데이터를 기반으로 연산을 수행하는 명령어 또한 처리가 늦어지게 된다. 따라서 캐쉬 자원인 MSHR이나 miss queue의 자원을 증가시켜 병렬성을 향상되어도 성능은 오히려 감소하기도 한다.

MSHR은 Fermi 구조의 L1 데이터 캐쉬 크기인 16KB에 대하여 발생하는 미스 요청에 대해 하위 메모리로부터 데이터가 전달될 때까지 정보를 기록하여 데이터를 요청한 레지스터 파일의 위치와 캐쉬의 예약 할당된 블록 위치에 전달된다. 또한 캐쉬의 할당 정책 중 하나인 allocation-on-miss 정책인 경우에는 캐쉬에 대한 요청이 미스로 결정되는 즉시 캐쉬 블록의 교체 정책에 따라 할당할 캐쉬 블록을 결정한다. 따라서 하위 메모리로부터 전달된 데이터는 MSHR에 저장된 정보에 따라 L1 데이터 캐쉬에 전달된다. MSHR은 현재 수행 중인 메모리 명령어들로부터 발생한 메모리 요청 경합을 간접적으로 나타내는 자원이면서 L1 데이터 캐쉬에서의 동시에 요청된 캐쉬 블록 수를 직접적으로 나타내는 자원이라고 할 수 있다. 따라서 MSHR의 현재 할당된 엔트리 수를 파악한다면 현재 메모리 집중도와 캐쉬 효율성을 파악할 수 있는 척도가 될 수 있다. 또한 MSHR은 캐쉬에 접근하는 스레드 단위로 엔트리를 구성하면서도 서로 다른 스레드가 동일한 캐쉬 블록에 대해 미스를 발생시킬 때 하위 메모리에 대한 요청 수를 줄일 수 있는 합병을 지원하고 있다. 따라서 단순히 메모리 명령어 수가 아닌 L1 데이터 캐쉬에서 요구되는 캐쉬 공간을 알 수 있는 자원으로 볼 수 있다.

3.2 제안하는 동적 스케줄러 구현

L1 데이터 캐쉬에서 발생하는 미스 정보를 저장하는 MSHR 활용률 정보를 기반으로 동적으로 워프 우선순위 정책을 적용하는 스케줄러링 기법을 제안한다. Fig. 2는 각 벤치마크가 실행시간 동안 보이는 워프 수준의 병렬성(Warp-Level Parallelism)을 나타낸다. 워프 스케줄러에 의해 이슈되어 명령어 수행이 완료되기 전까지 워프를 대상으로 동시에 수행되는 평균 워프 수를 측정하고 비교한다. 일정 사이클 간격으로 동시 수행되는 워프를 측정한 결과 모든 벤치마크에서 LRR 정책을 적용하는 경우 GTO보다 평균 50%의 더 많은 워프를 동시 수행하는 것으로 나타난다. GTO 스케줄링 정책은 스톨이 발생하지 않는 워프의 명령어를 계속 수행하기 때문에 상대적으로 동시 수행 워프 수가 적다. LRR 워프 스케줄러는 모든 워프가 공평한 우선순위를 갖도록 하기 때문에 하나의 워프가 연속적으로 수행되는 것을 방지하고 일부 한정된 자원을 계속 점유하지 않도록 워프간 균일한 이슈를 하는 경향이 있다. 따라서 활성화 워프들에 대해 균일하게 작업분배를 할 수 있다. 하지만 동등한 우선순위 이슈는 각 워프는 유사한 명령어 순서를 가지는 경향에 따라 동일한 시간대에 동일 자원을 점유할 수 있으므로 병목현상을 발생시키기도 한다. 워프 수준의 병렬성 증가는 한정된 SM의 연산 자원에 대한 병목 현상을 증가시키고 캐쉬의 경우 스레드간의 경합을 심화시켜 캐쉬의 효율성 감소로 이어지기도 한다. 따라서 병렬성이 높더라도 각 명령어의 수행시간이 길어지면서 전체 GPU의 성능이 오히려 감소할 수 있다. 본 논문에서는 메모리 경합 정도를 나타내는 MSHR 자원의 활용률을 모니터링하고 과도한 메모리 경합이 발생 시 동적으로 GTO 정책을 적용한다. GTO 정책은 병렬성이 상대적으로 낮고 스톨이 발생하지 않는 이상 소수의 워프만 계속 이슈되는 경향이 있기 때문에 메모리 경합을 감소시킬 수 있다.

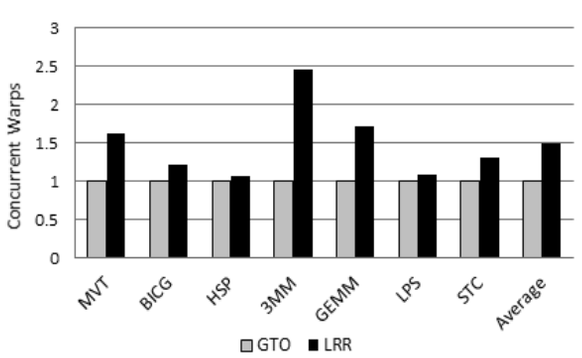

Fig. 2. Warp-Level Parallelism on an SM

GPU에서 SM은 단순히 동시 수행 워프의 정보를 저장하기 위한 관련 하드웨어의 용량과 복잡도를 증가시켜서 병렬성을 향상시킬 수 있다. 또한, 캐쉬 자원 중 하나인 MSHR의 크기를 증가시켜 캐쉬 자원 부족으로 인한 스톨을 줄이고 더 많은 메모리 명령어를 이슈할 수 있도록 허용할 수 있다. 하지만 다수의 워프에 의한 캐쉬 접근이 증가한다면 내부 연결망에 대한 부담도 증가될 뿐만 아니라 한정된 자원에 대한 병목 현상이 발생하거나 캐쉬의 효율이 감소될 수 있다. 기존 연구에서는 스로틀링(Throttling)과 같은 기법들을 통해 동적으로 동시 수행되는 워프의 수를 감소시켜 성능을 증가시킨다[9, 13].

MSHR은 메모리 경합 정도가 높아지거나 메모리 효율성이 낮아질 때 발생하는 Miss 수에 따라 MSHR 엔트리 수가 증가한다. 제안하는 워프 스케줄러는 다수 워프에 의한 경합을 감소시키고자 GTO 워프 스케줄링 정책을 적용하여 이슈 단계에서의 워프 수준의 병렬성을 낮춘다. 반대로 MSHR 사용량이 감소하면 메모리 경합이 완화된 것으로 판단하고 워프 스케줄링 정책을 동적으로 전환하여 워프 수준의 병렬성을 증가시킨다. LRR 정책은 모든 워프에게 동등한 이슈 우선순위를 갖게 하여 GTO 정책에 비해 워프 수준의 병렬성이 높아진다. 따라서 GPU 성능에 큰 영향을 미치는 긴 지연시간 메모리 명령어의 경합 정도가 낮다면 균일한 작업분배를 위한 LRR 정책을 적용함으로써 성능 향상을 기대할 수 있다.

Fig. 3은 제안된 워프 스케줄링을 지원하기 위한 하드웨어 구조를 블록 다이어그램으로 나타낸다. 기존 GPU 구조와 대비하여 제안된 기법을 위해 추가된 유닛은 크게 두 가지로 나뉜다. 두 개의 스케줄링 정책에 따라 워프의 우선순위를 정렬 및 저장하는 로직과 L1 데이터 캐쉬로부터 MSHR 활용률을 기준치와 비교하여 동적으로 우선순위 정책을 선택하는

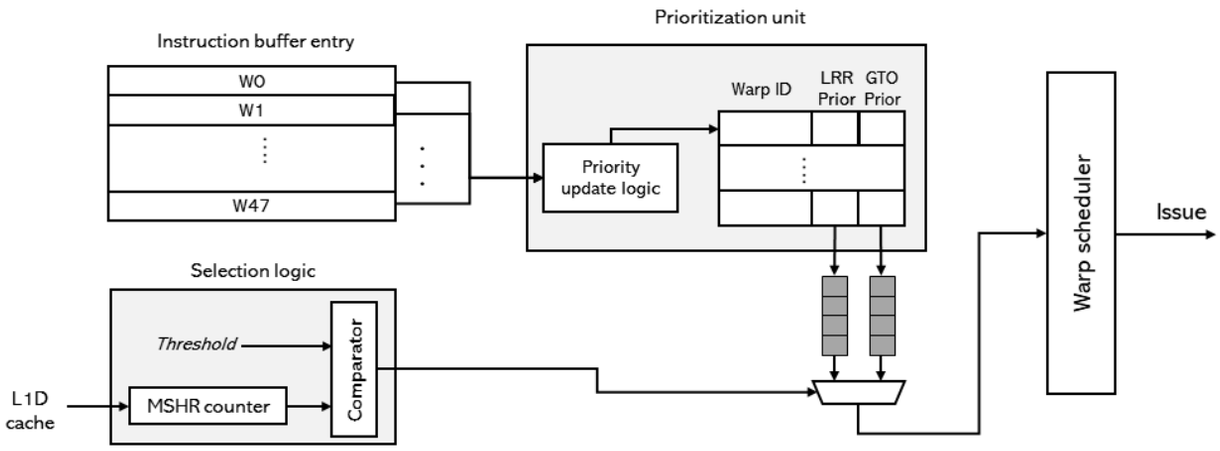

Fig. 3. Block Diagram of Warp Scheduler with Proposed Units

유닛으로 나뉜다. 추가적으로 MSHR의 자원 활용률 정보를 실시간으로 전달받기 위해 MSHR으로부터 엔트리 사용량 정보를 전달하도록 변경한다.

명령어 인출/해석 단계와 이 단계로부터 얻은 해석 결과인 피연산자와 명령어 타입 등의 데이터는 명령어 버퍼(I-Buffer)에 저장되며 각 피연산자에 대한 정보와 목적 레지스터에 대한 데이터 해저드(Data Hazard)는 스코어보드(Scoreboard)를 통해 확인할 수 있다. 제안하는 워프 스케줄러는 최소한의 정보만을 요구하는 워프 스케줄러 정책인 LRR 정책과 GTO 정책을 각각 운용하기 위한 하드웨어 자원으로 구성되며 두 개의 정책 중 하나를 동적으로 선택하기 위한 유닛이 추가적으로 필요하다. LRR 워프 스케줄링 정책은 모든 활성화 워프를 순차적으로 수행하기 위해 워프 식별자만(Warp ID)을 이용한다. GTO 워프 스케줄링 정책은 스레드가 해당 SM에 할당될 때 시간을 기준으로 한다. 따라서 각 워프가 SM에 할당될 때 별도의 증감 카운터를 통해 워프별 할당 순서를 관리할 수 있다. Fig. 3에서 제안하는 우선순위화 유닛(Prioritization Unit)은 워프 풀(Warp Pool)에서 다른 두 워프 스케줄링 정책에 따라 서로 다른 우선순위 비트를 결정 및 갱신하는 Priority Update Logic을 포함한다. LRR 정책은 Warp ID 정보만을 가지고 공평한 이슈 순서를 가지므로 구현 복잡도가 상대적으로 높지 않다. GTO 또한 각 워프의 할당 시간을 기점으로 정렬된다. 각 워프는 고유의 Warp ID를 가지고 있다. 각 SM에서의 워프 할당 시간(Allocation Time)을 포함한 워프 관련 정보는 기존 GPU의 구조의 명령어 버퍼를 통해 쉽게 확인할 수 있다. LRR 정책과 GTO정책을 동적으로 선택함에 있어서 GTO 정책에서 LRR 정책으로 전환될 때 최근 이슈된 워프의 ID의 다음 ID에 해당하는 워프가 이슈된다. 따라서 최근 이슈된 Warp ID를 별도로 저장하고 매 사이클마다 우선순위를 변경한다. 제안하는 우선순위화 유닛은 Warp ID를 인덱스로 하는 엔트리를 갱신하며, 엔트리는 GPU의 각 SM이 할당가능한 최대 엔트리 수인 48개의 엔트리로 구성한다. 또한 각 엔트리는 LRR 정책과 GTO 정책에 따른 우선순위를 나타내는 비트를 가진다.

Fig. 3에서 선택 로직(Selection Logic)은 L1 데이터 캐쉬의 미스 정보를 저장하는 MSHR로부터 엔트리 점유율을 전달받아 우선순위 정책을 동적으로 선택한다. 워프 스케줄러가 매 사이클 워프를 선택할 때는 해당 워프가 피연산자가 준비되어있고 요구 하는 데이터의 다른 명령어의 피연산자에 대해 의존적이지 않아야 한다. 따라서 스코어보드와 명령어 버퍼를 확인하여 연산 준비가 된 명령어인 경우 워프를 이슈한다. 이때 워프들은 우선순위에 따라 큐에 저장되고 매 사이클 동적인 워프 스케줄링 정책을 사용한다면 추가적인 지연시간이 발생할 수 있다. 하지만, MSHR 사용량에 따른 정책 변경이 비교적 몇 사이클 늦게 적용되더라도 제안하는 기법이 사이클 단위의 정확성을 요구하지 않기 때문에 지연을 무시할 수 있다. 따라서 제안하는 워프 스케줄링 정책은 주기적인 사이클마다 MSHR 엔트리의 점유율을 계산하고 적용하여 주기적으로 제안 기법을 적용함으로써 잦은 정책 변환을 하지 않는다.

각 워프가 SM에 할당될 때의 시간 정보는 기존 GTO 정책을 사용하는 구조와 거의 동일한 하드웨어 공간을 사용한다. MSHR 엔트리의 사용량 또한 아주 작은 복잡도를 가지는 유닛을 이용하여 쉽게 구현할 수 있는 하드웨어이므로 전체 추가적인 하드웨어 공간은 무시할 수준이다. 제안하는 워프 스케줄러는 기존의 워프 스케줄링과 마찬가지로 우선순위 정책에 따라 정렬하여 큐 구조로 구현할 수 있다. 또한 GTO와 LRR 정책에 따른 우선순위는 기존 GPU구조와 마찬가지로 매 사이클 갱신이 가능하다.

4. 실험 환경 및 결과

4.1 실험 환경

본 논문에서는 최신 GPU를 모델링하고 수정이 가능한 시뮬레이터로서 실제 GPU에서의 성능 측정과 매우 유사한 결과를 보여주는 GPGPU-SIM v3.2 [14]를 이용하여 실험한다. GPGPU-SIM은 사이클 수준의 성능 측정을 하는 시뮬레이터로서 NVIDIA CUDA로 구현된 애플리케이션을 수행하고 실

제 GPU 구조에서의 실행과 매우 유사한 측정 결과를 보여준다. 본 실험에서는 Fermi의 GTX480 아키텍처를 기반으로 하며 상세한 구조는 Table 1을 따른다.

Table 2는 본 논문에서 실험을 수행한 대상 벤치마크들을 나타낸다. CUDA로 작성되어 성능 검증에 널리 알려진 다양한 벤치마크들로서 RODINIA[15], ISPASS[14], Polybench[16], Parboil[17]로부터 벤치마크를 선정하여 실험을 수행한다. 각 벤치마크는 LRR 정책과 GTO 정책을 각각 적용한 GPU에서 실행함에 따라 성능 차이를 보이며, 5% 이상의 성능 변화를 보이는 경우 상대적으로 우수한 워프 스케줄링 정책에 해당하는 유형으로 분류한다. 만약 벤치마크를 실행할 때 LRR 정책을 적용한 아키텍처가 GTO를 적용한 아키텍처보다 5% 이상의 높은 성능을 보인다면 RF(Round Robin Friendly)으로 분류하고, 반대의 경우 GF(GTO Friendly)으로 분류한다.

4.2 실험 결과 분석

본 장에서는 제안하는 워프 스케줄링 기법을 기존 GPU 구조에 적용하고 기존 LRR 정책과 GTO 정책을 적용한 구조와의 성능을 비교하고 분석한다. 각 벤치마크의 고정된 전체 명령어와 소모된 사이클을 측정하여 IPC (Instruction per Cycle) 값을 계산하고 성능 향상 이유에 대해 분석한다. Fig. 4는 LRR 정책과 GTO 정책과 제안하는 기법의 성능을 비교한다. 제안 기법은 동적으로 LRR 정책과 GTO 정책을 적용함에 있어서 MSHR 엔트리 사용량의 기준점 T(Threshold) 값에 따라 성능을 측정하였다. 사용된 값 T는 전체 엔트리의 수 32개에 대한 사용 비율을 25%, 50%, 75% 적용하여 성능을 측정한다. MSHR-25% 은 전체 MSHR 엔트리의 25%를 사용하기 때문에 L1 데이터 캐쉬에서의 미스 발생에 따라 할당되는 MSHR 엔트리의 사용이 비교적 적게 발생함에도 GTO 정책으로 전환된다. GTO 정책에 유리한 벤치마크인 MVT, BICG, HSP의 경우 좋은 IPC 결과를 보이지만 RF 타입의 벤치마크들에 대해서는 LRR 정책의 성능에 크게 미치지 못한다. MSHR-50%와 MSHR-75%의 경우 LRR 정책에 비해 12.3%, 12.8% 높은 평균 IPC를 보인다. 또한, GTO 정책보다는 각각 3.1%, 3.5% 성능이 높게 나타난다.

제안하는 기법은 L1 데이터 캐쉬에서 미스 발생 요청이

Table 1. System Configuration

Parameter	Value
# of SM	16
Warp Size and SIMD Width	32
# of threads/SM	1024
# of registers/SM	32768
L1 Data-Cache	16KB per SM (32-sets/4-ways)
L1 Inst-Cache	2KB per SM (4-sets/4-ways)
Min. L2 access Latency	120 cycles
# of memory controller	12
Memory channel bandwidth	4KB

Table 2. Benchmarks

Benchmark Name	Addr.	Type	#Inst.
MVT [16]	MVT	GF	268,578,816
BICG [16]	BICG	GF	268,595,200
Hotspot [15]	HSP	GF	103,560,660
3MM [16]	3MM	RF	3,247,964,160
GEMM [16]	GEMM	RF	1,351,876,608
Laplace [14]	LPS	RF	66,673,024
Stencil [17]	STC	RF	849,033,056

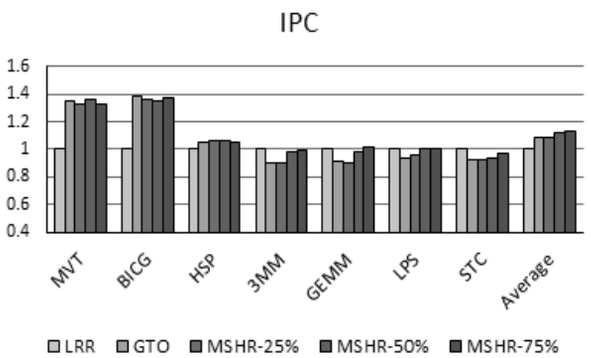

Fig. 4. IPC Comparison with Different MSHR Threshold T

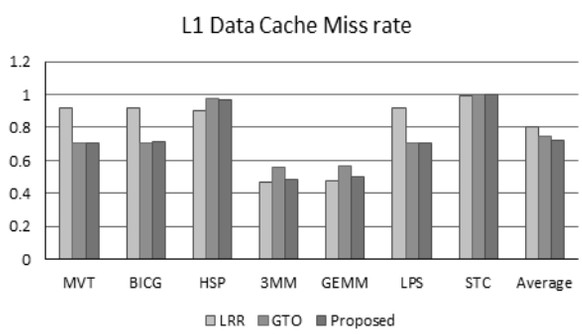

Fig. 5. L1 Data Cache Miss Rate Comparison

과도해지는 워크로드에 대해 워프 수준의 병렬성을 감소시켜 비교적 소수의 워프를 우선적으로 이슈하는 GTO 정책을 사용하여 L1 데이터 캐쉬에 대한 워프간 경합을 줄이고 캐쉬 효율 또한 향상시킨다. Fig. 5는 L1 데이터 캐쉬에서 발생하는 미스율을 보인다. 캐쉬 미스율은 캐쉬의 효율을 나타내는 수치 중 하나로 GPU의 성능에 큰 영향을 끼치는 요소이다. 제안하는 기법은 HSP 벤치마크를 제외하고 가장 낮은 미스율과 유사한 결과를 보이며, 평균적으로 LRR 정책과 GTO 정책보다 낮은 미스율을 보인다. 따라서 제안하는 기법의 가장 큰 성능 향상 원인은 L1 데이터 캐쉬 효율성 향상으로 분석된다.

일반적으로 GPU의 경우 Fig. 5에서 보이는 바와 같이 L1 데이터 캐쉬의 효율이 매우 낮다. L1 데이터 캐쉬의 미스율은 대략 70~80% 수준을 보인다. 또한 동시에 다수의 스레드가 접근하기 때문에 짧은 시간 내에 다수의 미스를 처리하는

MSHR 사용량은 일정 이상의 분포를 보인다. Fig 5.에서 보이는 바와 같이 가장 낮은 미스율을 보이는 3MM 벤치마크 또한 Fig. 6에서와 같이 일정량 이상의 MSHR 사용량을 보인다. Fig. 6은 3MM 벤치마크의 실행 시간에 따라 MSHR 사용량을 측정하여 구간별 평균값을 구하여 나타낸다.

Fig. 7에서의 내부 연결망 스톨 사이클은 내부 연결망 혼잡(Interconnect congestion)으로 인한 DRAM 채널에서의 스톨과 내부 연결망에서 DRAM 채널에서의 요청이 처리되지 않아 스톨이 발생한 사이클의 합을 나타낸다. 각 벤치마크별 측정된 스톨 사이클은 전체 명령어 수를 고려하지 않고 LRR 정책의 스톨 수에 대한 정규화된 값이다. 3MM과 GEMM 벤치마크는 GTO 정책에 대해서 내부 연결망 스톨이 크게 증가하지만 명령어당 스톨의 수는 매우 작기 때문에 영향이 적다. MVT와 BICG 벤치마크는 명령어당 내부 연결망 스톨 사이클이 매우 크다. 메모리 명령어 분포가 크고 미스율이 높기 때문에 IPC 값이 상대적으로 낮은 벤치마크다. 두 벤치마크

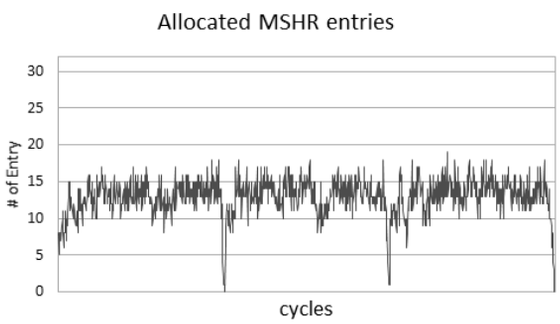

Fig. 6. MSHRs Usage of 3MM

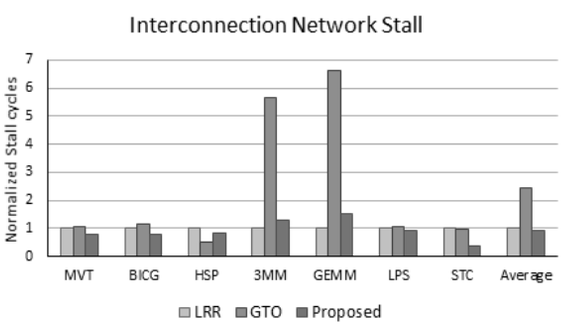

Fig. 7. Interconnection Network Stall Comparison

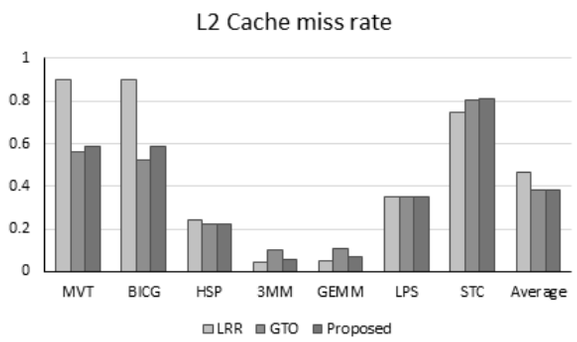

Fig. 8. L2 Cache Miss Rate

에 대해서 제안하는 기법은 LRR과 GTO 정책보다 성능을 크게 감소시켰다. LPS 벤치마크는 GTO 정책이 LRR 정책보다 낮은 미스율을 보임에도 불구하고 IPC가 더 낮다. 또한 내부 연결망 스톨과 L2 캐쉬에서의 미스율이 거의 동일하다. 하지만 DRAM에서의 Row buffer hit 발생율은 LRR 정책을 사용할 때 더 높다. 제안하는 기법은 LPS 벤치마크에 대해 Row buffer hit 발생율이 LRR 정책과 유사하게 나타나고 L1 데이터캐쉬 효율은 GTO와 같이 높기 때문에 결과적으로 우수한 성능을 보이는 것으로 분석된다.

Fig. 8의 L2 캐쉬에서의 미스율을 나타낸다. L2 캐쉬에서 미스 요청은 DRAM 접근으로 이어지기 때문에 데이터를 전달 받을 때까지의 지연시간이 길어진다. 제안하는 워프 스케줄링 기법은 수행 워크로드에 적합한 정책을 LRR과 GTO 중에서 동적으로 선택하므로 가장 최선의 결과값을 얻을 수 있다. Fig 8.에서 보이는 바와 같이 제안하는 기법은 가장 낮은 L2 캐쉬 미스율을 보인다. MVT, BICG 벤치마크에 대해 제안하는 기법은 LRR보다 낮은 접근 수를 보이고 GTO와 유사한 결과를 보인다. 결과적으로 DRAM 접근율이 낮아짐에 따라 메모리 명령어의 평균 레이턴시를 크게 감소한다.

5. 결 론

본 논문은 GPU의 성능에 영향을 미치는 요인 중 하나인 L1 데이터 캐쉬에 대한 미스 발생 정보를 저장하는 MSHR을 모니터링하고 서로 다른 병렬성을 보이는 LRR 정책과 GTO 정책을 동적으로 적용함으로써 전체 성능을 향상시킬 수 있는 기법을 제안하였다. LRR 정책은 다수의 워프에게 동등한 기회로 이슈될 수 있도록 하여 병렬성을 향상시키지만 메모리 경합을 발생시킨다. GTO는 스톨이 발생하기 전까지 동일 워프에 대해 지속적인 높은 우선순위를 주기 때문에 병렬성은 낮지만 캐쉬에 대한 집중도를 낮춰 효율성을 증가시킨다. 실험 결과 L1 데이터 캐쉬에서의 미스율은 LRR과 GTO 정책에 비해 크게 감소하고 내부 연결망 스톨 또는 DRAM 접근이 감소됨을 알 수 있었다. 따라서, 기존의 고정된 정책을 사용하는 것보다 메모리 경합을 고려하여 스케줄링 정책을 동적으로 선택함으로써 높은 성능 향상을 이끌어냈다. 7개의 벤치마크를 대상으로 제안하는 기법을 적용한 결과 LRR 정책과 GTO 정책과 비교하여 평균 약 12.8%, 3.5% 증가하여 제안하는 기법의 우수성을 확인할 수 있었다. 향후에는 MSHR에서의 요청 결합 정보를 기반으로 지역성이 반영된 워프 스케줄링 연구를 수행하고자 한다.

References

[1] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, “Improving GPGPU resource utilization through alternative thread block scheduling,” *High Performance Computer Architecture(HPCA), International Symposium on*. pp.260-271, 2014.

[2] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *Micro-architecture (MICRO), Annual IEEE/ACM International Symposium on. IEEE*, pp.308-317, 2011.

[3] Y. Zhang, Z. Xing, C. Liu, C. Tang, and Q. Wang, "Locality based warp scheduling in GPGPUs," Future Generation Computer Systems. 2017.

[4] B. Wang, Y. Zhu, and W. Yu, "OAWS: Memory occlusion aware warp scheduling," *Parallel Architecture and Compilation Techniques (PACT), 2016 International Conference on. IEEE*, pp.45-55, 2016.

[5] J. Wang, N. Rubin, A. Sidelnik, and S. Yalamanchili, "LaPerm: Locality aware scheduler for dynamic parallelism on GPUs," *ACM SIGARCH Computer Architecture News 44.3*, pp.584-595, 2016.

[6] Y. Liu et al. "Barrier-aware warp scheduling for throughput processors," in *Proceedings of the 2016 International Conference on Supercomputing. ACM*, pp.42, 2016.

[7] NVIDIA CUDA Programming [internet], http://www.nvidia.com/object/cuda_home_new.html

[8] M. Lee et al. "iPAWS: Instruction-issue pattern-based adaptive warp scheduling for GPGPUs," *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. IEEE*, pp.370-381, 2016.

[9] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society*, pp.72-83, 2012.

[10] J. Zhang, Y. He, F. Shen, Q. A. Li, and H. Tan, "Memory Request Priority Based Warp Scheduling for GPUs," *Chinese Journal of Electronics*, Vol.27, No.7, pp.985-994, 2018.

[11] B. Wang, W. Yu, X. H. Sun, and X. Wang, "Dacache: Memory divergence-aware gpu cache management," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, pp.89-98, 2015.

[12] K. Choo, D. Troendle, E. A. Gad, and B. Jang, "Contention-Aware Selective Caching to Mitigate Intra-Warp Contention on GPUs," *Parallel and Distributed Computing(ISPDC)*, pp.1-8, 2017.

[13] X. Chen, L. W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W. M. Hwu, "Adaptive cache manageent for energy-efficient gpu computing," in *Proceedings of the 47th annual IEEE/ACM International Symposium Microarchitecture*, pp.343-355, 2014.

[14] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceedings of International Symposium*, pp.163-174, 2009.

[15] S. Che, M. Boyer, M. Jiayuan, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pp.44-54, 2009.

[16] 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.

[17] J. A. Stratton et al. "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing 127*, 2012.

김 광 복

https://orcid.org/0000-0002-6750-1008

e-mail : loopaz63@gmail.com

2013년 전남대학교 전자컴퓨터공학부 (학사)

2015년 전남대학교 전자컴퓨터공학과 (석사)

2015년~현 재 전남대학교 전자컴퓨터공학과 박사과정

관심분야 : 컴퓨터구조, 저전력 설계, 병렬처리

김 종 면

https://orcid.org//0000-0002-5185-1062

e-mail : jmkim07@ulsan.ac.kr

1995년 명지대학교 전기공학과(학사)

2000년 University of Florida ECE(석사)

2005년 Georgia Institute of Technology ECE(박사)

2005년~2007년 삼성종합기술원 전임연구원

2007년~현 재 울산대학교 IT융합학부 교수

관심분야 : 임베디드 SoC, 컴퓨터구조, 프로세서 설계, 병렬처리

김 철 홍

https://orcid.org/0000-0003-1837-6631

e-mail : chkim22@jnu.ac.kr

1998년 서울대학교 컴퓨터공학과(학사)

2000년 서울대학교 컴퓨터공학부(석사)

2006년 서울대학교 전기컴퓨터공학부 (박사)

2005년~2007년 삼성전자 반도체 총괄 SYS.LSI 사업부 책임연구원

2007년~현 재 전남대학교 전자컴퓨터공학부 교수

관심분야 : 임베디드시스템, 컴퓨터구조, SoC설계, 저전력 설계