Journal of The Korea Society of Computer and Information
Vol. 24 No. 4, pp. 1-9, April 2019
www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.04.001

Latency Hiding based Warp Scheduling Policy for High Performance GPUs

Gwang Bok Kim*, Jong Myon Kim**, Cheol Hong Kim*

Abstract

LRR(Loose Round Robin) warp scheduling policy for GPU architecture results in high warp-level parallelism and balanced loads across multiple warps. However, traditional LRR policy makes multiple warps execute long latency operations at the same time. In cases that no more warps to be issued under long latency, the throughput of GPUs may be degraded significantly. In this paper, we propose a new warp scheduling policy which utilizes latency hiding, leading to more utilized memory resources in high performance GPUs. The proposed warp scheduler prioritizes memory instruction based on GTO(Greedy Then Oldest) policy in order to provide reduced memory stalls. When no warps can execute memory instruction any more, the warp scheduler selects a warp for computation instruction by round robin manner. Furthermore, our proposed technique achieves high performance by using additional information about recently committed warps. According to our experimental results, our proposed technique improves GPU performance by 12.7% and 5.6% over LRR and GTO on average, respectively.

▸Keyword: GPUs, Warp Scheduler, Latency Hiding, Thread Level Parallelism, Data Locality

I. Introduction

GPU와 같은 처리량 향상을 위해 고안된 프로세서들은 최근 강력한 연산 자원을 이용하여 범용 프로그램을 빠르게 수행할 수 있다. CUDA[1]와 같은 프로그래밍 모델이나 OpenCL[2]은 기존 멀티코어에 비해 더 많은 동시 처리가 가능한 환경을 제공함으로써 점점 그 활용 범위가 커지고 있다.

GPU에서는 최소 연산 처리 단위인 스레드로부터 SM(Streaming Multiprocessor)에서 동시 수행 가능한 스레드의 집한인 워프(Warp)와 스레드간의 통신이 가능한 집합인 CTA(Cooperative Thread Array)까지의 계층을 가지고 있다. SM에 있는 워프 스케줄러는 어떤 워프가 선택되어 실행될지를 선택하는 역할을 한다. 워프 스케줄러는 서로 다른 명령어를 수행하는 워프들의 실행 순서를 결정하기 때문에 데이터 지역성이나 병렬성을 결정한다. 따라서 워프 스케줄링 기법과 관련한 연구로 GPU 성능 향상을 위해 많은 관련 기법들이 제안되고 있다[3-8, 10, 11].

일반적으로 GPU 아키텍처에 널리 적용되고 있는 워프 스케줄링 정책은 모든 워프에게 공평한 선택 순서를 적용하는 LRR(Loose Round Robin)이나 일부 워프에게 실행 우선순위를 높게 갖도록 하는 GTO(Greedy Then Oldest) 정책이 있다. 두 워프 스케줄링 정책은 구현 방법이 간단하면서도 강력한 성능을 보이지만 GPU에서 수행되는 다양한 워크로드의 특성에 모두 최적화될 수 없기 때문에 수행 벤치마크에 따라 다른 성능 양상을 보인다.

최신 워프 스케줄링 정책들은 과도한 TLP(Thread Level

• First Author: Gwang Bok Kim, Corresponding Author: Cheol Hong Kim
*Gwang Bok Kim (loopaz63@gmail.com), School of Electronics and Computer Engineering, Chonnam National University
**Jong Myon Kim (jmkim07@ulsan.ac.kr), IT Convergence Department, University of Ulsan
*Cheol Hong Kim (chkim22@chonnam.ac.kr), School of Electronics and Computer Engineering, Chonnam National University
• Received: 2019. 01. 22, Revised: 2019. 04. 03, Accepted: 2019. 04. 11.
• This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A2B6005740) and This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2019-2016-0-00314) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation)

Parallelism)으로 인한 역효과를 줄이기 위해 일부 워프들만 우선적으로 이슈될 수 있도록 하는 GTO 정책에 기반하는 경우가 많다. 또한 일반적으로 많은 벤치마크에서 동일 워프에서 발생하는 지역성(Intra-warp locality)이 서로 다른 워프 사이에서 발생하는 지역성(Inter-warp locality)보다 크게 나타난다는 기존 연구 결과[3, 4]를 바탕으로 GTO 정책이 적용되는 경향이 있다. 다시 말해, 애플리케이션의 코드가 반복적으로 수행되면서 데이터 블록이 동일한 워프로부터 다시 참조되는 경우가 많이 발생하는 것이다. 그럼에도 불구하고 일부 벤치마크에서는 공평한 우선순위 정책인 LRR 정책이 훨씬 높은 성능을 보이는 경우도 있다. GTO 정책은 스톨이 발생하지 않는 이상 동일한 워프가 순차적으로 명령어를 계속 수행하도록 하고, 스톨이 발생한다면 가장 할당된 시간이 오래된 워프를 우선적으로 이슈하기 때문에 SM에서 할당된 시간이 오래된 워프들이 계속적으로 이슈되는 결과를 초래한다. 반대로 LRR 정책은 모든 워프를 이슈 대상으로 공평하게 이슈하는 우선순위 정책이기 때문에 GTO 정책에 비해 스레드 수준의 병렬성이 높다. 다수의 워프는 동일한 명령어 순서를 가지는 경우가 많기 때문에 LRR 정책을 적용한다면 거의 비슷한 시간 내에서 동일한 자원에 대한 경쟁을 발생시킬 수 있다. 특히 긴 지연 시간을 발생시키는 메모리 명령어 실행을 동시에 발생시킬 수 있기 때문에 메모리 관련 자원에 대한 경합이 발생하고 그동안 실행시킬 활성화 워프가 없어 일부 자원을 활용하지 못할 수 있다[9]. 본 논문에서는 지연시간 숨김을 적극적으로 활용하기 위해 메모리 명령어를 우선적으로 이슈하고 GTO와 LRR 정책의 이점만을 활용하는 정적 워프 스케줄링 기법을 제안함으로써 GPU의 성능을 향상시키고자 한다.

본 논문은 구성은 다음과 같다. 2장에서는 기존 GPU 구조와 제안하는 기법의 연구 배경에 대해 설명한다. 3장은 제안된 워프 스케줄링 기법에 대해 구현 방법을 자세히 기술한다. 4장에서는 제안된 기법과 기존의 GPU 구조와의 성능 비교 및 분석을 수행한다. 마지막으로 5장에서 본 논문의 결론을 맺는다.

II. Background

1. GPU Architecture

최신 GPU는 다수의 SM(Streaming Multiprocessor)로 구성된다. 각 SM은 내부 연결망(Interconnection Network)을 통해 각 별도의 메모리 파티션(Memory Partition)에 연결된다. 각 메모리 파티션은 또한 각자의 DRAM chip와 L2를 가지고 있다. SM은 주소를 가지고 어떤 메모리 파티션도 접근할 수 있다. 그림 1은 SM의 주요 구성요소를 포함하여 마이크로아키텍처로 표현한 그림이다. 각 SM은 SIMD(Single Instruction Multiple Data) 구조를 갖는 파이프라인을 갖으며 크게 명령어 캐쉬, 워프 스케줄러, 레지스터 파일, L1 데이터 캐쉬로 구성되어있다. 또한 SM에는 실행 중인 모든 스레드의 컨텍스트(Context)를 저장하는 워프 풀(Warp Pool)이 있다. 워프 스케줄러는 해저드가 발생하지 않도록 명령어 버퍼를 확인하여 적절한 실행 유닛에 명령어를 전달하는 역할을 한다.

GPGPU 응용 프로그램은 커널이라는 동일한 코드 부분을 수행하는 스레드의 집합으로 이루어진다. 실행 시간 동안 여러 스레드가 CTA(Cooperative Thread Array) 혹은 스레드 블록(Thread Block)이라고 불리는 단위로 스레드 집합을 형성한다. 스레드 블록 내의 모든 스레드는 SIMD의 동시 처리가 가능한 폭(SIMD-width) 만큼인 워프(Warp) 단위로 수행되고 관리된다. 워프 스케줄러는 워프 풀에서 파이프라인 해저드를 발생시키지 않을 워프를 선택하여 매 사이클마다 이슈한다. 멀티 스레딩 기술과 큰 레지스터 파일을 사용하면 스레드/워프가 스톨을 발생시킬 때마다 문맥 전환(Context Switching)을 빠르게 수행하여 불필요한 지연시간을 최소화한다. 따라서 GPU는 기존 CMP 설계와 비교하여 평균 파이프라인 사용률과 처리량을 높일 수 있다[10].

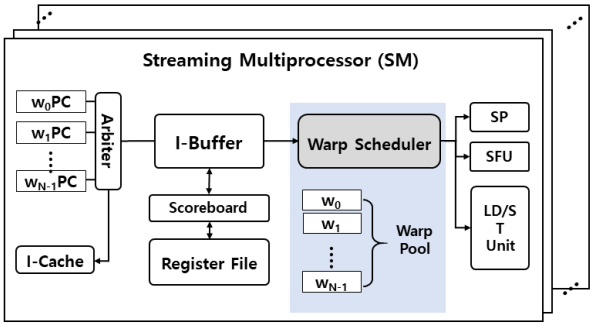

Fig. 1. Microarchitecture of SM

기존 GPU에서는 구현에 용이하고 높은 성능을 보이는 단일 워프 스케줄링 기법들이 적용되어왔다. 따라서 기존 GPU 워프 스케줄러 관련 연구들은 LRR 또는 GTO 정책과 함께 비교한다. LRR 정책은 워프 ID를 기준으로 워프들이 순차적으로 이슈할 수 있도록 하고, 이슈 사이클에 이슈될 준비가 되지 않은 워프는 무시한다. 모든 워프를 이슈할 수 있는 후보 워프로 보기 때문에 워프/CTA 관점에서 균등하게 이슈되는 경향이 있고, 병렬성을 극대화할 수 있는 정책이다. 하지만 활성화 스레드의 풀 크기가 특정 이상으로 증가하면 작을 때보다 성능이 저하될 수 있다. 근접한 워프는 동일한 명령어 세트를 처리하는 경향이 있기 때문에 동일 자원을 거의 유사한 사이클 내에서 많은 워프가 동일 유형의 명령어를 수행할 수 있다. 따라서 일반적으로 TLP(Thread Level Parallelism)의 증가에 따라 구조적 해저드가 빈번하게 발생할 수 있다. GTO 정책은 스톨이 발생하지 않는다면 동일 워프를 계속 수행한다. 만약 스톨이 발생한다면 할당된 시간이 오래된 워프를 우선적으로 이슈한다. 따라서 모든 워프가 전반적인 긴 대기시간을 갖지 않는다면 할당된 시간이 오래된 워프들이 계속적으로 수행된다. GTO와 같은 정책은 워프별 수행률이 불균형해져 워프 사이의 지역성을 저해할 수도 있다. 하지만 기존 연구들[3-5]과 GTO의 경우 동일 워프가 코드 재사용의 특성에 따라 빠른 시간

내에 재사용 될 수 있는 워프 내 지역성을 활용하는 것이 전체적인 GPU 성능에 유리하다. GPU가 수행하는 다양한 범용 프로그램들은 지역성 특성이 상이하기 때문에 전혀 다른 정책 방향을 갖는 LRR과 GTO 스케줄러는 서로 다른 성능을 보인다. 따라서 GTO와 LRR 정책의 장점을 결합한 연구들이 GPU의 성능을 크게 향상시킨다[11-12].

CCWS[3]은 L1 데이터 캐쉬에서의 워프 내 지역성을 제대로 활용하지 못하는 문제점을 완화하고자 제안된 워프 스케줄링 기법이다. L1 데이터 캐쉬에서 교체된 태그 정보를 추가된 캐쉬에 저장하고 워프별로 스레싱(Thrashing) 발생 빈도를 모니터링한다. 만약 한 워프가 지역성을 활용하지 못한다면 해당 워프를 다른 워프에 비해 우선적으로 이슈하여 요청 데이터가 교체되기 전에 재사용할 수 있도록 한다. 따라서 캐쉬의 지역성을 향상시킬 수 있지만 스레드 수준의 병렬성이 감소될 수 있다. 또한, 실행 시간 동안 워프별로 스레싱 관련 정보를 갱신하고 계산하기 위한 하드웨어 복잡도와 공간이 요구된다.

iPAWS[11]는 공평한 우선 정책인 LRR과 일부 특정된 워프들이 계속 우선적으로 이슈되도록 하는 GTO 정책을 동적으로 적용하는 기법이다. 제안된 기법은 LRR 정책에 유리한 워크로드는 워프별 스톨이 유사하게 발생하여 소수 워프가 계속 이슈되기 어려운 패턴을 보인다는 관찰 결과를 기반으로 실행 시간 동안 워크로드의 이슈 패턴을 모니터링하고 LRR과 GTO 정책 중에서 선택적으로 하나의 정책만을 적용을 한다. CCWS 정책과 결합하여 서로 다른 특성을 갖는 벤치마크들에 대해 평균적으로 우수한 성능을 보인다.

CAWA[10]에서는 명령어와 스톨 기반의 임계성(Criticality) 예측기를 제안하고 이에 따라 스레드 블록에서의 각 워프들을 분류한다. 또한 임계성 기반의 워프 스케줄러는 임계성 워프들을 더 자원에 할당하도록 하고 캐쉬 재사용성을 예측한다. CAWA는 작업부하에 대한 자원 활용률을 향상시키기 위해 실행 시간 차이를 제거한다.

2. Motivation

그림 2는 LRR 정책과 GTO 정책이 적용된 SM에서 동시에 수행하는 워프의 수를 정량화하여 비교하기 위해 이슈된 후 완료 전까지의 워프를 주기적으로 측정한 결과이다. 실행 중인 워프의 수를 그림 2와 같이 나타낸다. LRR 정책과 GTO의 정책은 극명한 정책 방향의 차이를 보인다. LRR 정책은 모든 활성화 워프에게 동등한 이슈 순서를 부여하기 때문에 훨씬 큰 범위의 워프 관점에서 작업 분배가 가능하며 이로 인해 병렬성이 상대적으로 높다. 반면 GTO 정책은 일부 워프가 지속적으로 명령어를 수행하도록 하기 때문에 워프 내 시간적 지역성 활용 측면에서 유리하지만 병렬성이 상대적으로 낮을 수밖에 없다. 그림 2와 같이 각 벤치마크의 실행시간 동안 동시에 이슈 상태에 있는 워프 수를 구간별 측정하여 평균을 도출한 결과 LRR 정책이 GTO 정책에 비해 평균 28% 더 높은 것을 확인할 수 있다.

GPU에서의 작업 분배 문제는 주로 SM 수준에서의 CTA 작업 분배를 중점으로 하는 연구를 포함하여 단일 CTA 내부의 워프 사이와 CTA 사이에서 발생하는 작업 불균형(Load Imbalance)도 심각한 차이를 보일 수 있다. 또한 서로 독립적인 작업을 수행하는 CTA는 동시에 수행이 가능하다. 따라서 워프 사이의 스케줄링은 SM에 할당된 다수의 CTA에 대한 스케줄링을 포함한다.

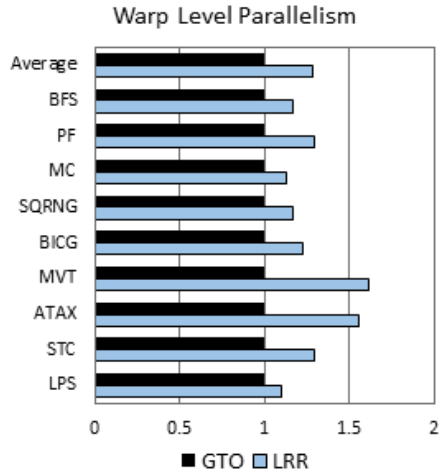

Fig. 2. Warp Level Parallelism

LRR 정책의 높은 병렬성은 자원에 대한 경합(Contention) 또는 모든 워프가 스톨 상태로 더 이상 이슈할 워프가 없는 상태를 발생시킨다. 워프를 이슈할 때 일부 워프를 우선적으로 이슈하는 GTO 정책과 모든 워프가 균형있는 이슈가 되는 LRR 정책은 메모리 접근에 있어서도 차이를 보인다. 메모리 명령어를 이슈하는 순서가 바뀌면 메모리에 접근하는 순서 또한 변경되므로 지역성에 영향을 끼친다. 일반적으로 애플리케이션에 따라 데이터 재사용 패턴은 달라질 수 있다. 동일한 스레드가 동일한 데이터를 사용하거나 접근하는 메모리의 일정한 간격(stride)을 가지는 경우 또는 반복(loop)의 크기에 따라 같은 데이터를 사용하는 스레드가 달라질 수 있다. 워프의 관점에서 데이터 지역성은 크게 동일한 워프가 명령어를 순차적으로 처리하면서 동일한 데이터를 요청하는 워프 내 지역성(Intra-warp locality)과 서로 다른 워프가 동일한 데이터를 요청하는 워프 사이 지역성(Inter-warp Locality)로 분류될 수 있다. 이러한 애플리케이션 별 특징에 따라 워프를 이슈하는 동적 스케줄링 정책들은 성능을 크게 향상시킬 수 있다. 하지만 대다수의 워프는 반복적인 명령어 처리를 하는 대다수의 커널 특성에 따라 워프 사이 지역성보다는 워프 내 지역성이 더 강하게 나타난다. 따라서 오직 워프 내 지역성을 탐지하고 활용하는 연구들이 존재한다[3, 4].

GPU는 명령어를 처리할 때 긴 지연시간을 발생시키더라도 다른 명령어를 수행함으로써 긴 지연시간을 숨길 수 있다. 하지만 워프를 이슈할 때 짧은 레이턴시를 가지는 워프를 우선적으로 이슈하고 긴 레이턴시를 발생시키는 워프를 이후에 이슈한다면 긴 지연시간을 충분히 숨길 수 없으며, 처리량 중심의 프로세서의 경우 성능이 크게 감소된다. 따라서 긴 지연시간을 발

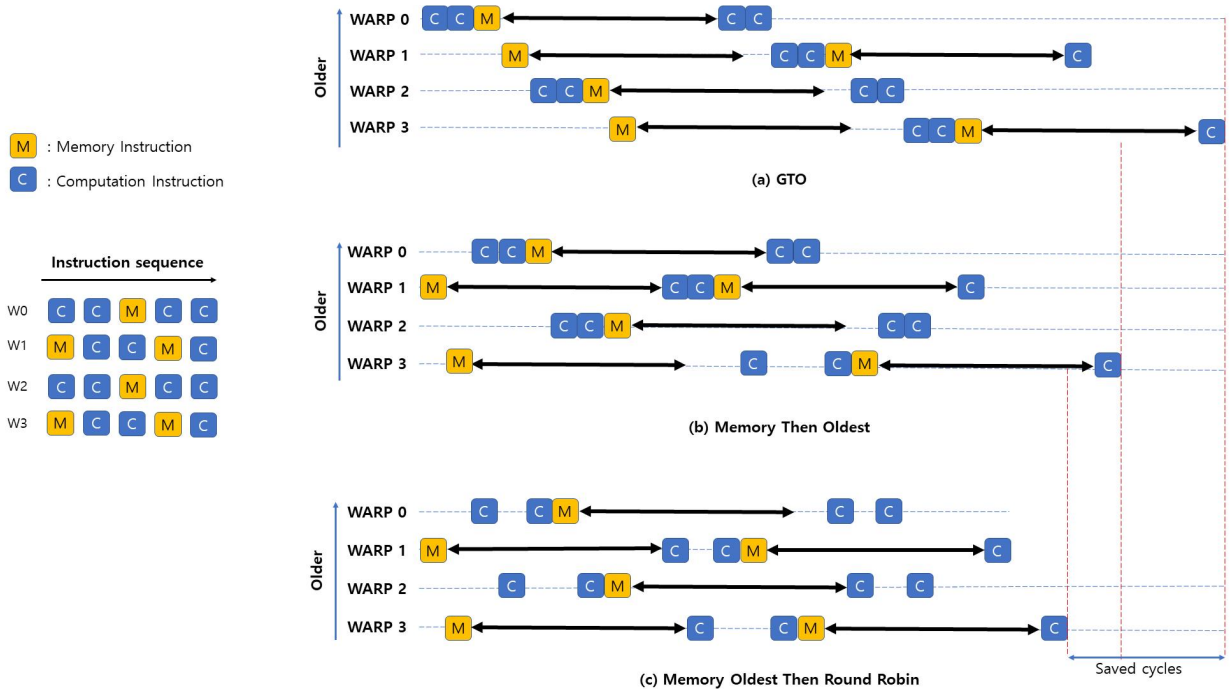

Fig. 3. Warp Scheduling Policy Comparison

생시킬 수 있는 명령어를 우선적으로 이슈하고 짧은 실행 시간을 갖는 명령어를 이슈하면 성능이 증가될 수 있다.

하지만 단순히 메모리 명령어를 우선적으로 처리하는 정책만 사용한다면 메모리 자원에 대한 병목현상(Bottleneck)을 발생시킬 수 있다. 본 논문에서는 이러한 병목현상을 일으키지 않으면서 지연시간 숨김과 메모리 자원 활용에 유리한 정적 워프 스케줄링 기법을 제안한다. 기존의 동적 워프 스케줄링 기법은 실행하는 애플리케이션의 특성을 하드웨어 또는 소프트웨어 수준에서 탐지하여 사이클 단위 또는 기간(Period) 단위로 정책을 적용한다. 하지만 여러 가지 자원 상태를 고려하거나 실행 시간(run time) 동안 측정하기 위해서는 하드웨어적인 복잡도와 공간이 추가적으로 필요하다. CCWS 기법은 워프별 캐쉬 태그를 저장하고 각 워프별 지역성 정도를 계산하기 때문에 추가적인 하드웨어 공간이 요구된다.

본 논문에서는 다양한 특성을 가지는 벤치마크에 대해 높은 성능을 보장할 수 있도록 서로 다른 특성을 가진 워프 스케줄링 정책들을 결합하여 GPU의 성능을 결정하는 지연시간 숨김 효과와 메모리 자원을 적극적으로 활용할 수 있는 기법을 제안하고자 한다.

III. Warp Scheduling Policy for High Performance GPUs

1. Maximizing Latency Hiding

메모리 명령어는 메모리 계층에서 데이터를 불러오거나 쓰기 연산을 수행한다. 만약 온 칩 메모리에서 데이터 블록을 찾지 못하면 긴 지연시간이 추가적으로 발생한다. 오프 칩의 L2 캐쉬에 접근하는 경우 100 클럭 사이클 이상의 긴 지연시간이 발생한다. 또한 오프칩인 DRAM에 대한 접근 레이턴시는 400~600 클럭 사이클에 달한다 [18]. 따라서 긴 지연시간 동안 다른 명령어를 수행할 수 있도록 스케줄러에 의한 지연시간 숨김이 적극 활용되어야 한다. 또한, 스레드 수준의 병렬성(TLP; Thread Level Parallelism) 크기는 GPU의 자원을 효과적으로 활용하는 주 요인이라고 할 수 있다. 스레드 집합인 워프가 메모리 명령어를 수행할 때 스레드 스케줄러에 의해 다른 워프를 실행할 수 있도록 컨텍스트 스위칭(Context Switching)을 수행한다. 만약 SM에 이슈될 준비가 된 워프들이 충분하고 다수의 워프가 동시 수행 가능하도록 지원할 수 있는 관리 자원 또한 필요하다.

LRR 정책은 앞서 언급한 바와 같이 모든 워프에 대해 거의 같은 처리율을 보이기 때문에 동시에 같은 자원을 점유하는 경향이 크다. 특히 긴 지연시간을 발생시키는 메모리 명령어가 동시에 이슈된다면 캐쉬 자원에 대한 병목현상이 발생함으로써 성능을 감소시킬 수 있다. 또한, 긴 지연시간 동안 다른 연산 유닛을 사용하는 명령어가 준비되지 않는다면 일부 실행 유닛의 활용률 저하로 이어지게 된다. 그림 3은 자원을 무한정으로 가정하여 메모리 명령어와 연산 명령어가 워프 스케줄러에 의해 이슈될 때 서로 다른 워프 스케줄링 정책에 따라 다른 결과를 보여준다. 그림 3의 (a)는 GTO 정책에 따라 워프가 이슈되는 경우이고 (b)는 메모리 명령어를 우선적으로 이슈하되 동일한 명령어 타입에 대해서는 할당시간이 오래된 워프를 우선적으로 이슈한다. (c)는 메모리 명령어에 대해서 할당시간 순서를 따르되 그 외 연산 명령어를 수행하는 워프는 LRR 정책을 따르는 경우다. (b)와 (c)는 모두 메모리 명령어를 우선적으로 이슈함으로써 지연시간 숨김 효과를 통해 전체 실행 시간이 감소됨을 알 수 있다. 각 SM에서 제안하는 하드웨어 워프

스케줄러를 구현하기 위해서는 각 워프가 실행하는 명령어의 타입에 따라 메모리 명령어와 그 외 연산 메모리 명령어로 나누어 저장한다. 하지만 각 명령어 타입에 해당하는 워프들을 어떤 우선순위로 이슈할지 또한 결정해야 한다. 본 논문에서는 GPU의 구조에서 널리 사용되는 LRR 정책과 GTO 정책을 적용한다. LRR 정책과 GTO 정책은 구현 복잡도가 낮고 워크로드의 특성에 따라 각각 우수한 성능을 보인다.

본 논문에서는 명령어를 수행할 때 각각 다른 처리 시간을 가지는 명령어들을 효과적으로 병렬 실행하도록 스케줄링함으로써 긴 지연시간 숨김 효과를 높이고자 한다. 따라서 제안하는 기법을 적용하는 GPU는 명령어를 처리하는 전체 사이클 수가 감소한다. 본 논문에서 제안하는 효과적인 지연 시간 활용 방법을 크게 두 가지로 제안한다. 긴 지연시간을 발생키는 메모리 명령어(Memory Instruction)를 우선적으로 이슈하여 메모리 자원을 최대한 활용하는 방법과 연산 명령어(Computation Instruction)를 수행하는 워프에서는 LRR 방식을 적용하여 워프 수준의 병렬성을 높이는 기법이다. LRR 정책은 동시에 많은 워프가 동일 자원을 사용함으로써 성능 저하가 발생한다. 제안하는 기법은 메모리 명령어를 최대한 빨리 이슈하기 때문에 동시에 메모리 명령어가 수행되는 현상을 완화할 수 있다.

제안된 기법에 의해 워프 풀에 있는 이슈 준비 워프(Ready Warp)들은 수행할 명령어의 타입에 따라 크게 두 가지 그룹으로 분류되어 저장된다. 분류된 워프들은 각각 다른 큐에 삽입될 때 서로 다른 우선순위 정책에 따라 다시 정렬된다. 메모리 명령어를 우선적으로 이슈할 때 전체 실행 사이클은 그림 3에서와 같이 감소된 것을 볼 수 있다. 그림 3의 (b)와 (c)는 모두 GTO 정책에 비해 전체 실행 클럭 사이클이 줄어든다. 메모리 명령어를 우선적으로 이슈하고 연산 명령어에 대해서 LRR 정책을 사용하는 경우 그림 3의 (c)와 같이 (b) 정책에 비해 지연시간 숨김이 효과적으로 이루어진 것을 볼 수 있다.

2. Utilizing Intra-warp Locality

제안하는 워프 스케줄링 기법은 LRR 정책과 GTO 정책의 이점을 모두 활용하여 고정적으로 적용할 수 있는 기법을 제안한다. 특히 GTO 정책이 가지는 성능 측면의 이점은 워프 내 지역성을 적극적으로 활용하고 자원에 대한 경합 완화라고 할 수 있다. 제안 기법은 메모리 명령어에 대해 GTO 방식과 유사하게 SM에 할당된 시간이 오래된 워프 순서로 정렬한다. 따라서 각 워프가 수행할 다음 명령어는 메모리 명령어 타입 여부에 따라 할당 시간이 오래된 워프가 계속적으로 메모리 자원을 이용할 수 있도록 우선적으로 이슈한다.

명령어 버퍼는 워프 당 최대 2개의 명령어를 명령어 캐쉬로부터 인출하고 해석하여 정보를 저장할 수 있다. 따라서 현재 명령어 버퍼에 저장된 명령어를 처리해야만 다음 수행할 명령어를 인출 및 해석할 수 있다. 만약 GTO 정책에 따라 할당 시간이 오래된 워프가 메모리 명령어를 우선적으로 이슈하도록 설계하더라도 LRR 정책에 따라 연산 명령어를 이슈하는 정책에 영향을 받을 수밖에 없다. 다시 말해, 부분적인 LRR 정책의 영향으로 메모리 명령어에 대한 이슈를 할 때 명령어를 할당 시간이 오래된 워프들을 그렇지 않은 워프들보다 이슈할 기회를 항상 보장하진 못한다. 제안 기법은 최근 메모리 명령어를 완료하여 캐쉬에 데이터를 적재한 상태로 추정되는 워프들에 대해 이슈 후보 워프로 결정한다. 최근 메모리 명령어를 완료한 워프는 요청된 데이터가 L1 데이터 캐쉬에 존재할 확률이 매우 높다. 또한, 메모리 접근을 불특정 워프 그룹으로 한정하여 워프 내 지역성을 향상시킬 수 있다.

활성화 워프에 대해 명령어 처리 여부를 실시간으로 측정하여 최근 이슈된 워프로부터 순서대로 활성화 워프 수/2 개만 별도 그룹으로 분류한다. 본 논문에서는 이러한 워프 그룹을 최근 이슈 완료 워프라고 정의한다. 워프 스케줄러가 연산 명령어를 이슈할 때 최근 이슈 완료 워프를 구분하기 위해 각 워프별 최근 이슈 완료를 나타내는 레지스터가 요구된다. SM에서의 최대 할당 가능한 활성화 워프 수는 48개로 고정되어있다. 각 활성화 워프 슬롯(Warp Slot)에서 Load/Store 와 같은 메모리 명령어가 완료될 때 해당 워프 해당하는 카운터를 0으로 리셋한다. 또한, 다른 워프의 카운터는 모두 1을 증감시킨다. 또한, 활성화 워프 수(W_Active)를 실시간으로 카운트하여 W_Active/2개의 최근 워프 그룹이 유지되도록 한다. 만약 최근 워프의 수가 W_Active/2 개 이상이 된다면 카운터의 값이 가장 큰 워프를 최근 워프 그룹에서 제외 시킨다. 각 워프별 최근 워프 여부는 최근 비트(Recency Bit)를 통해 쉽게 관리할 수 있다. 활성화 워프의 수는 각 벤치마크마다 상이하다. 따라서 실제 워프가 할당된 후 활성화된 워프 수를 측정하고, 이러한 정보를 기반으로 최근 완료된 메모리 워프 수를 측정한다. 따라서, 제안하는 기법은 메모리 명령어를 우선적으로 이슈하여 SM에 할당된 시간이 오래된 워프를 우선적으로 이슈하고, 더 이상 이슈할 수 있는 메모리 명령어가 없거나 메모리 파이프라인이 스톨이 발생해서 해당 사이클에 이슈가 불가능하다면 연산 명령어를 수행하는 워프에서 이슈할 워프를 선택한다.

그림 4는 제안하는 기법을 위한 아키텍처를 간단히 나타낸다. GTO Ordering Logic은 메모리 명령어를 수행하는 워프에 대해서 할당 시간 정보를 기반으로 정렬하여 Memory Warp Queue에 삽입한다. 연산 명령어를 수행하는 워프들 또한, 최근 이슈된 워프 ID 값을 기준으로 라운드 로빈 방식으로 정렬된다. 그와 동시에 연산 명령어를 수행하는 워프들은 최근 레지스터(Recency Register)에 있는 최근 비트를 참조하여 최근 메모리 명령어를 이슈한 워프를 대상으로 우선적으로 정렬한다. 이와 같은 워프들 또한 Computation Warp Queue에 삽입된다. 그림 4에서 최근 레지스터는 총 3개의 필드로 구성된다. 인덱스가 워프 ID로 설정되며, 최대 48개의 엔트리를 포함한다. 카운터는 최근 메모리 명령어를 완료한 워프들을 관리하기 위해 워프별 6-bit 카운터로 구성한다. 카운터 값을 참조하여 최근 이슈 완료 워프는 R 비트를 1로 변경한다. 제안하는 기법은 각 워프가 수행하는 명령어의 타입에 따라 각각 다른 우선순위 기법을 적용하여 Memory Warp Queue와 Computation Warp Queue에 워프들을 정렬하여 저장한다.

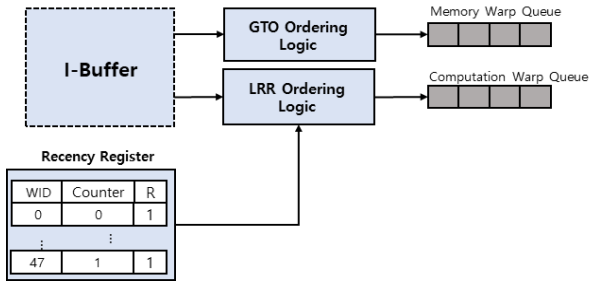

Fig. 4. Microarchitecture of Proposed Unit

알고리즘 1은 제안한 워프 스케줄링 기법 중 워프 풀에서 각 워프별 우선순위를 위한 값을 부여하는 과정을 pseudo 코드로 나타낸다. 제안하는 워프 스케줄링 기법은 메모리 명령어 타입을 수행하는 워프를 우선적으로 이슈하며, 이 워프들을 다시 GTO 방식과 동일하게 우선순위를 부여한다. 이 외 명령어 타입은 최근 메모리 명령어가 이슈되었는지를 판단하고 우선순위 값을 세트한다. 최근에 메모리 명령어가 이슈된 기록이 없는 워프는 그렇지 않는 워프보다 더 낮은 우선순위를 설정하기 위해 MAX_NUM_WARP 만큼 더 작은 값을 설정한다. 본 논문에서는 각 워프가 순서대로 우선순위 값을 1간격으로 부여받을 때 최근 비트가 설정된 워프를 그렇지 않은 워프보다 우선적으로 이슈하도록 보장해야 한다. 따라서 두 분류의 워프들은 최대 워프 수만큼의 우선순위 값 차이를 발생시킴으로써 항상 최근 비트가 설정된 워프를 우선적으로 이슈한다. 제안된 알고리즘에 따라 서로 다른 우선순위 값이 설정된 워프들은 두 개의 큐에 삽입되어 서로 다른 정책에 따라 스케줄링된다.

Algorithm 1. Proposed Warp Scheduling Algorithm

```
1: function WarpPrioritization(warp)
2: /* A warp with higher Prior value is prioritized */
3:   if warp.InstType == MemoryInst then
4:     warp.Prior ← GTO_prior;
5:   else
6:     if warp.Is_recency_bit then
7:       warp.Prior ← LRR_prior;
8:     else
9:       warp.Prior ← LRR_prior-MAX_NUM_WARP;
```

IV. Experimental Evaluation

1. Methodology

본 논문에서는 최신 GPU를 모델링하고 수정이 가능한 시뮬레이터로서 실제 GPU에서의 성능 측정 결과와 매우 유사한 시뮬레이터인 GPGPU-SIM[13]을 사용한다. GPGPU-SIM은 사이클 수준의 성능 측정이 가능한 시뮬레이터로서 신뢰성이 검증된 시뮬레이터이다. GPGPU-SIM은 CUDA와 OpenCL 응용프로그램을 입력받아 GPU 구조의 성능을 IPC와 자원 활용도 등 다양한 측면에서 측정할 수 있다. 본 실험에서는 GTX480 아키텍처를 기반으로 GPU 모델을 구성하고 각 구성요소는 표 1을 따른다. 벤치마크로는 NVIDIA SDK[14], Rodinia[15], ISPASS[13], Polybench[16], Parboil[17]에서 선별하여 총 9개의 프로그램을 사용한다. 벤치마크들은 GPU에서 병렬처리되기 위해 CUDA(Compute Unified Device Architecture) 플랫폼에 맞게 작성되고 컴파일된다. 실험에 사용된 벤치마크 프로그램은 표 2와 같다.

Table 1. Baseline Configurations

Parameter	Value
# of SMs	15
Warp Size	32 threads
# of threads/SM	1024
# of registers/SM	32768
L1 Data-Cache/SM	16KB, 4-way, 128 B line
L1 Inst-Cache/SM	2KB, 4-way
# of Warp Scheduler/SM	2
CTA Scheduler	Round Robin
Unified L2 Cache	768 KB, 128 KB/bank, 6 banks, 8-way, 128 B line
Interconnect	Fly, 32 B channel width, 1.4GHz

Table 2. Benchmarks

Benchmark	Description
LPS [13]	Laplace discretization on a 3D structured
STC [17]	Jacobi stencil operation on a 3-D grid
ATAX [16]	Matrix transpose and vector multiplication
MVT [16]	Matrix-vector product and transpose
BICG [16]	BiCG sub kernel of BiCGStab linear solver
SQRNG [14]	Sobol sequence generator
MC [14]	evaluation for fair call price using Monte-Carlo
PF [15]	finding a path on a 2-D grid
BFS [13]	Breath first search algorithm

2. Experimental Results

본 장에서는 제안하는 워프 스케줄링 기법을 기존 GPU 구조에서 사용되는 LRR 정책, GTO 정책과 성능을 비교하고 성능 차이에 대한 분석을 하고자 한다. 제안하는 기법은 LRR 정책과 GTO 정책의 각 이점을 활용하여 효과적인 지연시간 숨김과 병렬성 그리고 메모리 자원에 대한 효율성을 높임으로써 처리량을 향상시킨다.

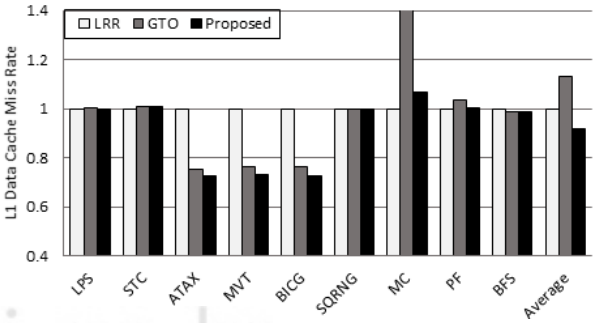

Fig. 5. L1 Data Cache Miss Rate

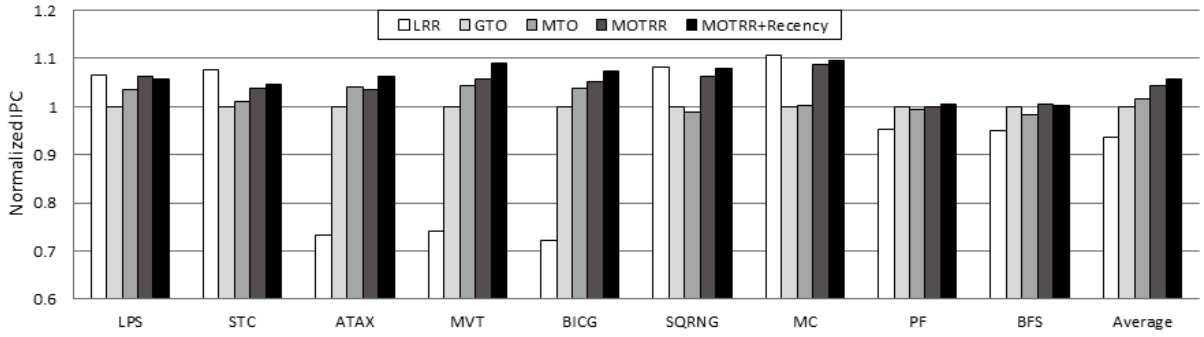

Fig. 8. IPC Comparison with Different Warp Scheduling Policy

그림 5는 L1 데이터 캐쉬에서의 미스율을 보인다. GTO 정책은 ATAX, MVT, BICG 벤치마크에서 낮은 미스율을 보이지만 MC 벤치마크의 미스율이 LRR에 비해 2.8배 높다. 제안하는 기법은 전체 벤치마크에서 낮은 미스율을 보여 L1 데이터 캐쉬에서의 지역성을 유지하는 것으로 나타난다. 또한, 그림 6에서 보이는 바와 같이 L1 데이터 캐쉬 관련 자원인 MSHR 또는 miss queue의 점유 가능한 엔트리가 부족하거나 캐쉬 세트에 대한 접근이 빈번한 경우 모든 블록이 데이터를 대기하는 경우 LD/ST 유닛에 대한 접근이 불가하다. 이런 경우 Reservation Fail 사이클로 정의하며 그 수치를 LRR, GTO 정책과 제안하는 기법의 결과를 비교한다. 일부 벤치마크에서는 이러한 스톨 사이클이 증가하지만 전체 벤치마크에 대한 평균 Reservation fail cycle은 낮은 것으로 나타난다. 따라서 캐쉬 효율성과 캐쉬 자원에 대한 과도한 경합도 측면에서 제안하는 기법이 우수한 것으로 나타난다.

그림 7은 이슈 단계에서 연산 자원을 사용할 수 없는 상태로 인해 단 하나의 워프도 이슈를 하지 못하는 스톨 사이클(Stall Cycle)을 측정하고 비교한 그래프이다. 보이는 바와 같이 제안하는 기법이 적용된 GPU 구조는 LRR 정책을 적용한 구조에 비해 23% 감소한다. 따라서, 제안하는 기법에 의해 지연 시간을 효과적으로 숨겨 성능을 향상시킨 것으로 분석된다. 앞서 분석한 요소인 캐쉬 효율성, LD/ST 유닛 스톨 수와 함께 모든 워프가 파이프라인 스톨이 되어 긴 지연시간 동안 워프를 이슈하지 못하는 사이클 수가 줄어든 것을 확인할 수 있다.

LRR 정책의 높은 병렬성과 워프들에 대한 수행 진행률을 균등하게 만든다. 제안하는 기법은 이러한 LRR 정책의 특성을 반영하면서 메모리 접근에 대하여 일부 워프만 주로 허용하도록 할당 시간을 기준으로 메모리 명령어를 이슈한다.

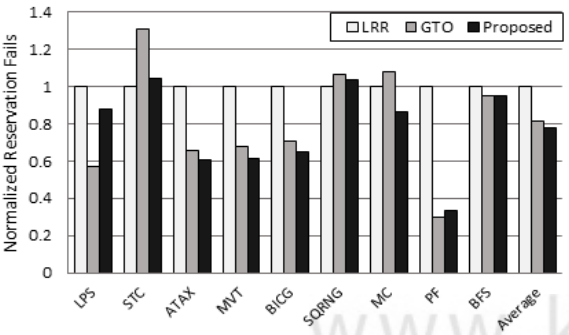

Fig. 6. Reservation Fails on L1 Data Cache

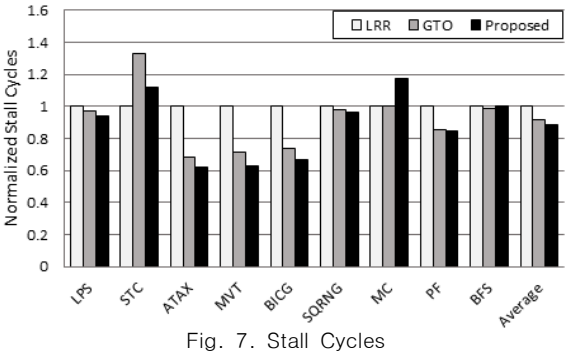

Fig. 7. Stall Cycles

그림 8은 LRR 정책과 GTO 정책을 적용한 구조와 함께 제안하는 다양한 정책들과의 성능을 비교한다. 메모리 명령어를 우선적으로 이슈하되 그 다음 할당 시간을 기준으로 오래된 워프를 이슈하는 MTO(Memory Then Oldest) 정책, 메모리 명령어 중 할당 시간이 오래된 워프를 이슈하되 다른 명령어 타입을 수행하는 워프는 라운드 로빈 방식으로 이슈 순서를 결정하는 MOTRR(Memory Oldest Then Round Robin), MOTRR 정책과 함께 최근 메모리 명령어 수행한 워프 정보를 반영하는 정책인 MOTRR+ Recency에 대해서 성능을 비교한다. 성능 비교는 사이클 당 처리 명령어 수인 IPC(Instruction Per Cycle)을 각각 동일한 벤치마크에 대해 측정하고 GTO 정책의 결과에 대해 정규화한다. MOTRR 정책은 MTO 정책과 다르게 연산 명령어에 대해 공평하게 이슈함으로써 더 높은 성능을 보인다. 따라서 단순히 LRR 정책과 GTO정책을 별도로 사용하는 것보다 명령어 유형별 다른 정책이 적용된 경우 더 효과적임을 보인다. LPS, STC, SQRNG, MC 벤치마크의 경우 GTO 정책과 MTO 정책 성능결과가 LRR 성능보다 훨씬 낮은 IPC를 보인다. 하지만 MOTRR과 MOTRR+ Recency 정책은 부분적으로 라운드 로빈 방식이 적용되기 때문에 급격한 성능 차이를 보인다. 실험 결과, MOTRR은 LRR 정책에 비해 평균 11.5% 그리고 GTO 정책에 비해 평균 4.4% 높은 성능을 보인다. 또한, 최근 메모리 완료 워프 정보를 활용하는 기법인 MOTRR+ Recency의 IPC는 LRR 정책과 GTO 정책에 대해 각각 12.7%, 5.6% 의 평균적인 성능 향상을 보인다. 다양한 벤치마크에 대해 LRR 정책과 GTO 정책은 서로 다른 성능 양상을 보인다. 따라서 일부 벤치마크에서는 LRR 정책이 GTO에 비해 높은 성능을 보인다. RMOTRR+ Recency 정책은 ATAX, MVT, BICG에서 상당한 성능 향상을 보인다. 특히 MVT 벤치마

크의 경우 RMOTRR+ Recency 정책으로 인한 성능이 MOTRR에 비해 3.1% 향상된다.

V. Conclusions

본 논문에서는 GPU의 병렬성과 자원 활용률을 결정하는 워프 스케줄러를 변경하여 GPU 성능을 향상시키고자 한다. 모든 워프를 대상으로 공평한 스케줄링 정책을 제시하면서 자원을 보다 효율적으로 활용할 수 있는 스케줄링 기법을 제안하였다. 기존 GPU 구조에서의 LRR 정책은 높은 병렬성과 비교적 균등한 작업 분배가 가능하지만 동일한 자원에 대한 병목현상을 발생시킬 수 있다. GTO 정책 또한 한정된 워프가 계속 우선적으로 이슈되는 탐욕적 알고리즘으로써 워프 사이의 병렬성을 감소시킨다는 단점이 있다. 본 논문에서 제안하는 기법은 긴 지연시간을 발생시키는 메모리 명령어를 우선적으로 이슈하기 위해 이슈단계에서 각 워프를 수행 명령어 타입에 따라 두 개의 큐에 분류한다. 메모리 명령어는 워프 내 지역성을 적극 활용하기 위해 GTO 정책에 따라 큐에 저장된다. 또한 메모리 이 외 명령어를 수행하는 워프는 LRR 정책에 따라 워프 들 사이에서 균등하게 이슈됨에 따라 병렬성을 향상시킨다. 제안하는 기법은 LRR 정책에 비해 12.7%, GTO 정책에 비해 5.6%의 성능 향상을 보였다. 또한 제안하는 기법은 다양한 애플리케이션에 대해 높은 성능 향상과 추가적인 하드웨어 오버헤드가 기존 연구들에 비해 작다는 장점을 가진다. 향후 연구를 통해서는 워프 사이 지역성과 워프 내 지역성을 모두 고려하는 워프 스케줄링 기법을 개발함으로써 GPU의 처리량을 더욱 향상시키고자 한다.

REFERENCES

[1] NVIDIA, “CUDA C Programming Guide,” 2012.

[2] Khronos OpenCL Group, “The OpenCL Specification,” 2011.

[3] T. G. Rogers., M. O'Connor., and T. M. Aamodt, “Cache-conscious wavefront scheduling,” Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture pp. 72-83, 2012.

[4] T. G. Rogers, M. O’Connor, and T. M. Aamodt, “Divergence-aware Warp Scheduling,” Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 99-110, 2013.

[5] Kim, G. B. Kim, J. M., & Kim. C. H., "Dynamic Selective Warp Scheduling for GPUs Using L1 Data Cache Locality Information," International Conference on Parallel and Distributed Computing: Applications and Technologies. Springer, Singapore, pp. 230-239, 2018.

[6] Zhang, Y., Xing, Z., Liu, C., Tang, C., & Wang, Q., “Locality based warp scheduling in GPGPUs,” Future Generation Computer Systems, 82, pp. 520-527. 2018.

[7] ElTantawy, A., & Aamodt, T. M., “Warp scheduling for fine-grained synchronization,” In 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 375-388, 2018.

[8] Oh, Yunho, et al. "Adaptive Cooperation of Prefetching and Warp Scheduling on GPUs." IEEE Transactions on Computers 68.4 (2019): 609-616.

[9] 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,” Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 308-317, 2011.

[10] S. Y. Lee, A. Arunkumar, and C. J. Wu, “CAWA: Coordinated warp scheduling and Cache Prioritization for critical warp acceleration of GPGPU workloads,” ACM SIGARCH Computer Architecture (ISCA), pp. 515-527, 2015.

[11] M. Lee, G. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, “iPAWS: Instruction-issue pattern-based adaptive warp scheduling for GPGPUs,” High Performance Computer Architecture (HPCA), IEEE International Symposium on. pp. 370-381, 2016.

[12] M. K. Yoon, Y. Oh, S. Lee, S. H. Kim, D. Kim, and W. W. Ro, “Draw: investigating benefits of adaptive fetch group size on gpu,” In Performance Analysis of Systems and Software (ISPASS), pp. 183-192, 2015.

[13] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, “Analyzing CUDA Workloads Using a Detailed GPU Simulator,” Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp. 163-174, 2009.

[14] “NVIDIA CUDA SDK Code Samples,” http://developer.nvidia.com/cuda-downloads, 2015.

[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Shadron, “Rodinia: A Benchmark Suite for Heterogeneous Computing,” 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, C. Rodrigues, J. I. Sung, et al. “Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing,” Center for Reliable and High-Performance Computing, 2012.

[18] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler,

W. J. Dally, E. Lindholm, and K. Skadron, "A hierarchical thread scheduler and register file for energy-efficient throughput processors," ACM Transactions on Computer Systems (TOCS), Vol. 30, No. 2, April 2012.

Authors

Gwang Bok Kim received the B.S degree and M.S in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2013 and 2015 respectively. He is currently a Ph.D student at Chonnam National University. His research interests include computer architecture, low power systems, and GPGPU.

Jong Myon Kim received the B.S. degree in electrical engineering from the Myong-Ji University, Yong-In, Korea, in 1995, the MS degree in electrical and computer engineering from University of Florida, Gainesville, in 2000, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2005. He was a senior research staff in the Chip Solution Center of Samsung Advanced Institute of Technology from 2005 to 2007. Since 2007, he has been with the IT Convergence Department at the University of Ulsan, Ulsan, Korea, where he is currently a Professor. His research interests include embedded systems, application-specific processors, and parallel processing.

Cheol Hong Kim received the B.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 1998 and M.S. degree in 2000. He received the Ph.D. in Electrical and Computer Engineering from Seoul National University in 2006. He worked as a senior engineer for SoC Laboratory in Samsung Electronics, Korea from Dec. 2005 to Jan. 2007. Now is working as professor at School of Electronics and Computer Engineering, Chonnam National University, Korea. His research interests include embedded systems, mobile systems, computer architecture, SoC design, low power systems, and multiprocessors.