Journal of The Korea Society of Computer and Information
Vol. 21 No. 2, pp. 9-16, February 2016

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.2.009

An IPC-based Dynamic Cooperative Thread Array Scheduling Scheme for GPUs

Dong Oh Son*, Jong Myon Kim**, Cheol Hong Kim***

Abstract

Recently, many research groups have focused on GPGPUs in order to improve the performance of computing systems. GPGPUs can execute general-purpose applications as well as graphics applications by using parallel GPU hardware resources. GPGPUs can process thousands of threads based on warp scheduling and CTA scheduling. In this paper, we utilize the traditional CTA scheduler to assign a various number of CTAs to SMs. According to our simulation results, increasing the number of CTAs assigned to the SM statically does not improve the performance. To solve the problem in traditional CTA scheduling schemes, we propose a new IPC-based dynamic CTA scheduling scheme. Compared to traditional CTA scheduling schemes, the proposed dynamic CTA scheduling scheme can increase the GPU performance by up to 13.1%.

▸ Keyword : General Purpose computation on the Graphics Processing Unit, Cooperative Thread Array Scheduling Schemes, Performance, Parallelism

I. Introduction

최근 컴퓨팅 시스템의 성능을 향상시키기 위해서 중앙처리장치(CPU: Central Processing Unit)의 처리량을 증가시키는 방법은 전력소비의 증가와 온도 문제의 발생으로 인해 제한되고 있다[1]. 이를 해결하기 위한 방법 중 하나로 CPU에서 작업을 처리할 때, 그래픽처리장치(GPU: Graphics Processing Unit)의 자원을 활용하여 CPU의 작업을 분담하여 컴퓨팅 시스템의 성능을 향상시키는 방법이 있다. GPU는 강력한 하드웨어 자원을 활용하여 병렬 연산을 수행할 수 있는 프로세서로써, 다수의 스레드들을 병렬적으로 수행하여 응용프로그램들의 데이터 병렬성을 활용할 수 있다[2]. 이를 위해서는 기존의 GPU에서 처리하던 그래픽 연산뿐만 아니라 CPU에서 전달된 작업의 범용 연산이 가능하여야 한다. GPU에서 범용 연산을 가능하게 하는 방법을 그래픽 처리장치 병렬컴퓨팅(GPGPU: General Purpose computation on the Graphics Processing Unit)이라 부른다. GPGPU는 GPU를 활용하여 그래픽 관련 연산뿐만 아니라 범용 연산을 수행 할 수 있으며, 최근에는 컴퓨팅 시스템 성능 향상을 위해서 GPGPU에 대한 많은 연구가 진행되고 있다. GPU 개발 업체들은 최신 GPU들의 GPU 병렬성을 손쉽게 활용하기 위해서 CUDA, OpenCL, ATI Streaming 등의 응용프로그램 인터페이스(API: Application Programming Interface)를 제공하고 있으며 응용프로그램 개발자들은 이를 사용하여 손쉽게 GPU의 병렬성을 활용할 수 있다[3-5]. 제공하는 API를 활용한다면 GPU에서 범용 작업 수행의 활용범위가 넓어지고 수천 개에 달하는 스레드들을 병렬적으로 처리가

• First Author: Dong Oh Son, Corresponding Author: Cheol Hong Kim
*Dong Oh Son(sdo1127@gmail.com), School of Electronics and Computer Engineering, Chonnam National University
**Jong Myon Kim(jmkim07@ulsan.ac.kr), School of Electrical Engineering, University of Ulsan
***Cheol Hong Kim(chkim22@chonnam.ac.kr), School of Electronics and Computer Engineering, Chonnam National University
• Received: 2015. 08. 18, Revised: 2015. 09. 14, Accepted: 2015. 11. 23.
• This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2015R1D1A3A01019454).

가능하여, 컴퓨팅 시스템의 성능을 크게 향상시킬 수 있다. 이처럼 최신 GPU는 GPGPU를 사용하여 그래픽 관련 작업뿐만 아니라 범용 작업 수행이 가능하다[6]. GPU는 단일 명령어 다중 데이터(SIMD: Single Instruction Multiple Data) 구조를 사용하며, 스레드들의 병렬성을 향상시키기 위하여 스레드 수준에서 스레드들을 SIMD 구조로 집단화한다. 이러한 스레드들의 집합을 워프(warp)로 부르며, 32개의 스레드가 모여서 하나의 워프를 구성한다. 이러한 워프는 GPU에서 연산을 수행할 때 사용되는 가장 기본적인 단위이다. 이러한 워프들은 스레드 블록이라는 단위로 구성할 수 있다. 스레드 블록은 CTA(Cooperative Thread Array)와 같은 의미로 사용되며, 워프들로 구성되어 있다[7]. 강력한 하드웨어 자원을 가진 GPU는 다수의 코어를 프로세서 내부에 포함하고 있으며, GPU 내부에 있는 코어, 즉 스트리밍 멀티프로세서(SM: Streaming Multiprocessors)는 워프와 CTA의 스케줄링 기법을 통하여 범용 작업 처리가 가능하다. 이것은 워프, CTA 이중 구조로 스케줄링이 가능하다[8]. GPGPU의 성능을 향상시키기 위해서 제안된 다양한 스케줄링 기법들은 GPU 캐쉬와 메모리 사이에서 발생하는 메모리 대기시간을 줄이기 위한 스케줄링 기법이나, GPU에 많은 워프들을 할당하여 GPU 연산 자원을 극대화하는 워프 스케줄링 기법 등이 연구되고 있다. 이처럼, 워프 스케줄링 기법을 활용한 다양한 연구가 진행되어 왔으나 CTA를 활용하여 스케줄링을 수행하는 CTA 스케줄링 기법에 대한 연구는 아직 미비한 수준이다[9-10]. CTA 스케줄링 기법도 GPGPU에서 성능향상에 많은 영향을 미치고 있으며, CTA 스케줄링 기법을 활용하여 컴퓨팅 시스템의 성능을 크게 향상시킬 수 있다. 따라서, 본 논문에서는 워프 스케줄링 기법과 비교하여 상대적으로 많은 연구가 진행되지 않는 CTA 스케줄링 기법에 대해서 연구의 초점을 맞춰 연구를 진행하고자 한다. 본 논문에서는 CTA 할당 개수가 성능에 미치는 영향을 파악하여 각 벤치마크 프로그램에 맞는 최적의 CTA 할당 개수를 구하며, 이를 동적으로 적용이 가능한 IPC 기반 동적 CTA 스케줄링 기법을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 논문의 대상인 GPU 구조와 CTA 대해서 자세히 설명한다. 또한, 3장에서는 CTA 스케줄링 기법에 대해서 기술하고, 본 논문에서 사용하는 정적 CTA 스케줄링 기법과 IPC 기반 동적 CTA 스케줄링 기법에 대해서 설명한다. 4장에서는 실험 환경과 실험에 사용된 다양한 벤치마크 프로그램에 대해서 기술한다. 5장에서는 SM에 할당된 CTA 개수에 따른 성능 분석과 동적 CTA 스케줄링 기법을 적용한 벤치마크 프로그램의 성능 비교 및 분석을 수행한다. 마지막으로 6장에서 본 논문의 결론을 맺는다.

II. GPU architecture and CTA scheduling scheme

1. GPU architecture

기존의 GPU는 그래픽 관련 연산을 전담하여 수행함으로써 CPU에서 발생하는 워크로드를 분담하여 수행하였다. 하지만 최신 GPU가 그래픽 관련 연산뿐만 아니라 범용 작업의 수행도 가능하다. GPU 발전에 따라 현재의 GPU는 수백 TFLOPS의 연산 능력을 가지고 있으며, 이것은 CPU의 연산 능력을 뛰어넘는다[11]. 이와 같은 장점으로 인해 GPU는 다양한 구조로 연구 및 개발되고 있다. GPU는 제조회사에 따라 다양한 구조가 개발되어 각 명칭 및 연산 방법 등이 상이하지만, 그래픽 처리를 위한 일반적인 구조는 거의 유사하다.

본 논문에서는 NVIDIA사의 Fermi 구조를 대상으로 GPU 구조에 관해서 설명하고자 한다[12]. GPU는 단일 명령어로 복수 데이터에 대한 연산을 동시에 수행하는 SIMD(Single Instruction Multiple Data) 방식을 사용한다. 이와 같은 구조는 하나의 명령으로 다수의 연산을 수행할 수 있으므로 병렬성이 극대화 된다. 그림 1은 NVIDIA사의 Fermi 구조를 나타낸 그림이다. 그림 1의 왼쪽 부분은 Fermi 구조의 전체적인 구조를 나타내며, 이 구조는 대용량의 데이터를 저장하기 위한 메모리가 GPU 내부에 내장되어 있으며, GPU 내부의 작은 GPU라고 불리는 스트리밍 멀티프로세서, 즉 SM은 중앙의 L2 캐쉬(L2 Cache)를 기준으로 위와 아래쪽에 분포되어 있다. SM은 실제 작업을 수행하는 유닛으로 그림 1의 오른쪽 부분에서 확인할 수 있다. SM은 실제로 연산을 수행하는 다수의 스트리밍 프로세서(SP: Streaming Processor)들과 레지스터 파일(Register File), 공유 메모리(Shared Memory)와 L1 캐쉬(L1 Cache)를 포함하고 있다. 또한, 32개의 스레드 집합인 워프(Warp)를 스트리밍 프로세서에 할당하기 위한 워프 스케줄러(Warp Scheduler)와 디스패치(Dispatch)를 포함하고 있다.

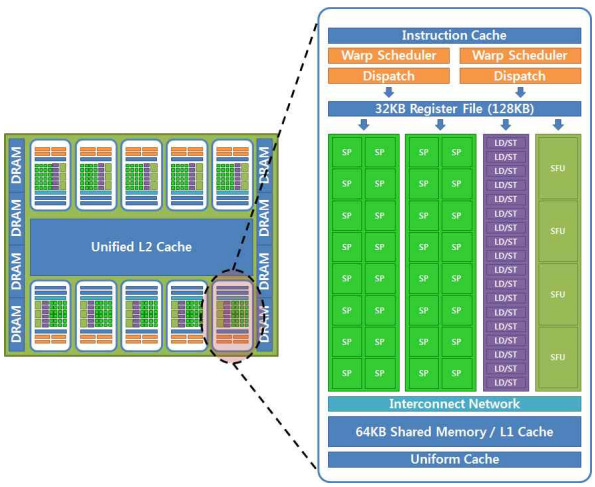

Fig. 1. NVIDIA's GPU architecture(Fermi)

2. Cooperative Thread Array

GPU에 작업을 할당하면 GPGPU구조는 할당된 작업을 세분화하여 스케줄링 기법을 통해 처리한다. 작업의 가장 큰 단위인 응용프로그램(Application)은 GPGPU에서 수행할 수 있도록 커널(Kernel) 단위로 변경된다. 생성된 커널은 다수의 CTA를 포함하고 있으며, 각 CTA는 SM에 독립적으로 할당이 가능하다. 따라서 생성된 커널에서 보유하고 있는 CTA를 각 SM에 할당하게 되는데, CTA 할당 개수의 기준은 SM에서 사용 가능한 연산 자원에 따라서 정해지게 된다. 각 SM은 CTA를 할당받기 전에 유휴 자원을 평가하며, 다수의 CTA 할당을 허용하는 FGMT 구조를 활용하기 때문에 자원 활용률의 향상과 병렬성 향상의 이점이 있다[13]. CTA는 Warp들의 모임으로 구성되어 있으며, GPGPU는 단일 명령어 다중 스레드(SIMT: Single Instruction Multiple Threads) 방식을 사용해서 32개의 스레드들을 동시에 수행하는데 이것은 하나의 워프 단위를 의미한다[14]. 그림 2는 이러한 GPU 프로그래밍 구조에 대한 그림을 보여주고 있다.

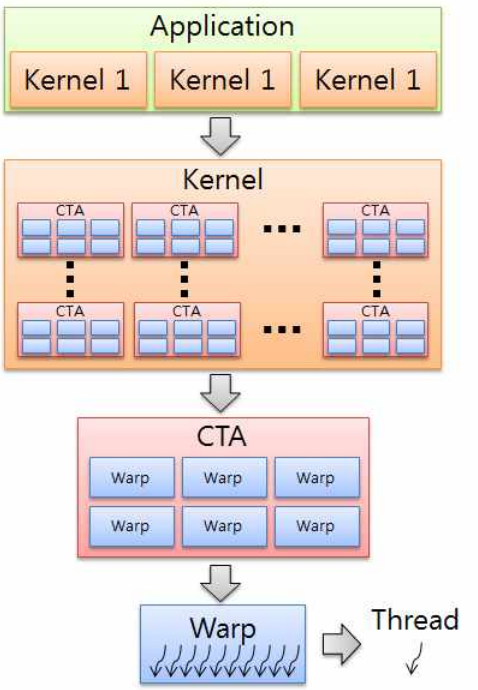

Fig. 2. GPU programming architecture (CUDA)

III. CTA scheduling scheme

일반적으로 CTA는 각 SM에 할당이 가능한 최대한 개수로 할당되어 높은 병렬성과 강력한 하드웨어 자원을 효율적으로 활용하고 있다[15]. 이러한 CTA 할당 개수는 GPGPU가 보유하고 있는 연산 자원들과 각 스레드들이 요구하는 연산 자원량에 따라서 달라지게 된다. GPGPU 구조의 성능 향상을 위해서는 CTA 스케줄링 기법을 통하여 각 SM에 최적의 CTA 개수를 할당하는 것이 중요하다. 기존의 CTA 관련 연구에서는 SM의 자원 활용과 병렬성 향상을 위해서 각 SM에 최대한의 개수로 CTA를 할당하는 것에 대한 연구가 진행되었다[8, 15]. 하지만, 각 SM에 최대한의 CTA를 할당하는 방법이 GPGPU 구조에 적합한 최적의 방법인지는 확인되지 않았다. GPGPU 구조에 적합한 CTA 스케줄링 기법을 개발하기 위해서는 CTA가 SM에 순차적으로 할당됨에 따라 발생하는 성능 분석이 선행적으로 이뤄져야 한다. 성능 분석을 통해 SM에 할당된 CTA 개수에 따른 성능이 CTA 개수 증가에 따라 선형적으로 증가하지 않는다면, 효율적인 CTA 스케줄링 기법을 활용하여 최적의 CTA 개수를 지정하여야 한다. 따라서 본 논문에서는 각 SM에 CTA 개수 할당에 따른 성능을 우선적으로 분석하고, 동적으로 CTA 개수를 조절하여 GPGPU에 적합한 최적의 CTA 개수를 설정하는 IPC 기반 동적 CTA 스케줄링 기법을 개발하고자 한다. 이와 같은 실험을 위해서는 우선적으로 최신 GPGPU 구조를 모델링하고, 다양한 벤치마크 프로그램들을 실행하여 SM에 할당된 CTA 개수를 1~8개까지 적용하여 성능 분석을 수행한다. 성능 분석에 따라 각 벤치마크 프로그램에 적합한 최적의 CTA를 동적으로 변경할 수 있는 IPC 기반 동적 CTA 스케줄링 기법을 제안한다. 본 논문에서 사용하는 IPC(Instructions Per Cycle)는 단위 사이클당 수행되는 명령어 수로 본 실험에서 성능의 지표로 사용하며, 수치가 높을수록 사이클당 처리되는 명령어의 수가 많음을 의미한다.

1. Static CTA Scheduling Scheme

본 논문에서 GPU 모델링의 대상으로 선택한 NVIDIA사의 Fermi 구조는 Giga Engine을 사용하여 CTA 스케줄링 기법을 선택한다. Giga Engine에서 사용하는 CTA 스케줄링 기법은 라운드 로빈 방식을 사용하며, 해당 스케줄링 기법을 통해 각 SM에 순차적으로 CTA가 할당된다. SM에 할당이 가능한 최대 CTA 개수인 8개를 라운드 로빈 방식으로 SM에 할당하고 나면, 나머지 CTA의 연산이 완료될 때까지 나머지 CTA들은 할당을 대기한다. 특정 SM에서 CTA 연산이 완료되면 다시 CTA를 특정 SM에 할당하여 최대 CTA 할당 개수인 8개를 유지한다.

본 논문에서는 기존의 정적 CTA 스케줄링 기법(CTA-8)을 포함하여, 각 SM에 할당되는 최대 CTA 개수를 1개부터 8개까지 고정하여 CTA 개수 변화에 따른 성능을 비교 및 분석을 수행하고자 한다. 따라서, 각 벤치마크 프로그램을 수행할 때 SM에 할당 가능한 최대 CTA 개수를 1~8개까지 고정하여 실험을 수행한다.

2. IPC-based Dynamic CTA Scheduling Scheme

본 논문에서 사용하고자 하는 IPC 기반 동적 CTA 스케줄링 기법을 사용하기 위해서는 Giga Engine을 수정하여 라운드 로

빈 스케줄링 기법이 아닌 GPU의 성능을 실시간으로 파악하여 CTA를 할당하는 방법을 사용해야 할 것으로 예상된다.

본 논문에서 제안하는 IPC 기반 동적 CTA 스케줄링 기법은 각 SM에 우선적으로 CTA 개수를 1개로 할당하여 벤치마크 프로그램을 수행하며, 10,000 사이클마다 SM에 할당되는 CTA 개수를 증가시킨다. 해당 주기마다 성능(IPC)을 측정하여 성능이 지속적으로 증가한 경우 SM에 할당되는 CTA 개수를 증가시키고 해당 주기에서 기존보다 성능이 하락한 경우 SM에 할당하는 CTA 개수를 감소시킨다. 이처럼 동적으로 SM에 할당되는 CTA 개수를 변경할 수 있다면, CTA 개수 증가에 의한 메모리 충돌과 SM 스톨 문제를 해결할 수 있다. IPC 기반 동적 CTA 기법은 주기별로 성능을 측정하여 SM에 할당되는 CTA 개수를 변경시킬 수 있으므로 다양한 벤치마크 프로그램에 적용이 가능하다.

실험을 위해서 CTA 할당 스케줄러를 수정한다. 처음 벤치마크 프로그램이 수행되면 CTA 스케줄러는 SM에 할당 가능한 최대 CTA 개수를 1개로 고정한다. 각 SM에 CTA를 할당하며 10,000 사이클마다 현재까지 수행된 명령어 개수와 사이클 수를 확인하여 IPC를 확인하고 측정한 IPC를 저장한다. 다시 10,000 사이클이 수행되면 기존에 저장된 IPC와 현재 IPC를 비교하여 기존의 IPC보다 현재 IPC가 높다면 SM에 할당 가능한 최대 CTA 개수를 1개씩 증가시키며 현재 IPC 값을 저장한다. 이처럼 10,000 사이클마다 기존의 IPC와 현재 IPC를 비교하며 IPC가 증가할 경우 CTA 할당 개수를 최대 8개까지 증가시킨다. 기존의 IPC와 현재 IPC가 같을 경우 최대 CTA 할당 개수를 증가시키지 않으며, 이전에 고정된 CTA 할당 개수에 맞춰 벤치마크 프로그램을 수행한다. 기존의 IPC보다 현재 IPC가 낮은 경우에는 SM에 할당 가능한 최대 CTA 개수를 1개 감소시키며, 고정된 CTA 할당 개수에 맞춰 벤치마크 프로그램을 수행한다.

이처럼 본 논문에서 제안하는 기법은 특정 주기마다 수행된 명령어 수와 사이클을 이용하여 IPC를 구하며, 구해진 IPC를 기반으로 동적으로 최대 할당 가능한 CTA 개수를 변경한다. 제안하는 기법은 Giga Engine에서 할당되는 최대 CTA의 개수만을 변경하기 때문에 별도의 하드웨어 공간이 필요치 않으며 연산 복잡도를 증가시키지 않는다.

IV. Experiments

1. Experiment environment

이 장에서는 실험 환경에 대해서 간략하게 설명하고자 한다. 각 SM에 CTA를 할당하기 위해서, GPU의 성능을 평가하는 시뮬레이터로 신뢰성이 검증된 GPGPU-SIM[16]과 전력 측정 시뮬레이터는 McPAT[17]을 통합한 시뮬레이터인 GPUWattch[18]를 활용하여 GPU를 모델링하였다. GPU 모델링 대상으로는 NVIDIA사의 Fermi 구조 GPU인 TeslaC2050을 모델링하여 실험을 수행하였다[12]. 본 논문에서 GPU 모델링 대상으로 사용한 Tesla2050 뿐만 아니라 최근에 개발된 대부분의 GPU는 API를 통해서 제안된 기법의 적용이 가능하다. 표 1과 2는 GPU 모델링에 따른 하드웨어 구성 변수와 내부 연결망의 구성 변수를 나타낸다.

본 실험에서 사용된 구성변수는 SM이 14개이며 각 SM에 할당 가능한 최대 CTA 개수는 8개이다. 따라서, 실험에서 사용할 수 있는 최대 CTA 할당 개수는 112개이다.

Table 1. Hardware parameters

실험 인자	값
Number of SM	14
Warp Size(SIMD Width)	32
Number of Threads/SM	1024
Shared Memory/SM	16KB
Constant Cache/SM	8KB, 2-way 64byte lines, Read-only
Texture Cache/SM	12KB, 24-way 128byte lines, Read-only
L1 Data Cache	16KB, 4-way, 128byte lines
Unified L2 Cache	64KB, 8-way, 128byte lines
Clock (Core: Interconnection: DRAM)	575MHz: 575MHz: 750MHz
Number of Memory Controller	6
Number of Memory Chip/Controller	2
Memory Channel Bandwidth	4 bytes
GDDR3 Memory Timing	tCL=10, tRP =10, tRC=35, tRAS=25, tRCD=12, tRRD=8
Warp Formation	Post Dominator
CTA&Warp Scheduler (Scheduling Scheme)	Two-level scheduler (Round-Robin)

Table 2. Interconnection network parameters

실험 인자	값
Topology	Crossbar
Routing Mechanism	Destination Tag
Routing Delay	0
Virtual Channels	1
Virtual Channel Delay	0
Virtual Channel Buffers	8
Virtual Channel Allocator	iSLIP
Input Speedup	2
Output Speedup	1
Internal Speedup	1
Flit Size	32 bytes

2. Benchmark programs

본 실험에서 사용되는 GPUWattch 시뮬레이터는 다양한 벤치마크 프로그램을 사용하여 벤치마크 프로그램의 수행 결과와 함께 GPU의 성능 및 전력소모량 출력이 가능하다[18]. 본 논문에서 사용한 벤치마크 프로그램으로는 NVIDIA SDK 벤치마크 프로그램[19], Rodinia 벤치마크 프로그램[20], ISPASS 벤치마크 프로그램[16]을 사용하였다. 다양한 벤치마크 프로그램 사용에 따른 성능 영향을 분석하기 위해서 3종류의 벤치마크 프로그램을 사용하였고, 그중 6개의 벤치마크 프로그램을 선택하였다. NVIDIA SDK에서는 Sobol Quasirandom Number Generator, Simple Templates 벤치마크 프로그램을 선택하였고, Rodinia 벤치마크 프로그램에서는 Heart Wall, K-Means을 선택하였으며, 마지막으로 ISPASS 벤치마크 프로그램에서 Graph Algorithm: Breadth-First Search, MUMmerGPU 벤치마크 프로그램을 선택하여 실험을 수행하였다. 표 3은 사용된 다양한 벤치마크 프로그램의 이름과 약자, 벤치마크 프로그램에 대한 설명을 보여준다.

Table 3. Benchmark programs

벤치마크 명	약자	설명
Sobol Quasirandom Number Generator	SQRNG [19]	Sobol Quasirandom 번호 생성기
Simple Templates	SimTem [19]	동적으로 할당된 공유 메모리 배열을 템플리트 하는 방법을 보여주는 샘플 프로그램
Heart Wall	HW [20]	초음파 영상의 시퀀스에 쥐 심장의 움직임을 추적
K-Means	KM [20]	데이터 마이닝에서 주로 사용되는 클러스터링 알고리즘
Graph Algorithm: Breadth-First Search	BFS [16]	그래프에서 폭우선 검색을 수행하는 알고리즘
MUMmerGPU	MUM [16]	표준 DNA 염기서열로 구성된 쿼리 문자열을 매칭하는 순차 정렬 프로그램

V. Experiment result

1. Performance according to number of CTAs assigned to the SM

본 장에서는 SM에 할당된 CTA 개수에 따른 성능 결과를 분석한다. 실험은 GPUWattch 시뮬레이터를 활용하여 모델링된 NVIDIA사의 TeslaC2050 GPU를 대상으로 다양한 벤치마크 프로그램을 수행하였다. 그림 3은 SM에 할당된 CTA 개수에 따른 GPU 성능 결과를 보여준다. 세로축은 정규화된 성능 값(IPC)을 나타내며, 가로축은 다양한 벤치마크 프로그램의 약자를 의미한다. 정규화 기준 값으로는 SM에 CTA를 8개 할당하는 CTA-8을 기준으로 정규화하였다.

실험 결과를 분석해 보면 세 가지 유형으로 분류할 수 있다. 첫 번째, SM에 할당된 CTA 개수가 증가함에 따라 성능도 증가하는 경우이다. 이와 같은 경우는 SM에 할당된 CTA 개수가 증가함에 따라 연산 자원을 효율적으로 사용이 가능하다. 기존에 연구결과에 의하면 SM에 할당된 CTA 개수가 증가할수록 GPU 연산 자원을 보다 많이 사용할 수 있으므로 자원 활용률과 병렬성이 크게 증가한다. 또한, SM은 CTA에서 발생하는 명령어를 처리하기 위해서 메모리로부터 데이터를 요구하게 되는데, SM에 할당된 CTA 개수가 증가하게 되면 특정 CTA에서 발생한 메모리 요청시간 동안 다른 CTA의 연산을 수행할 수 있으므로, 메모리 요청 대기시간에 의한 스톨을 방지할 수 있다.

두 번째, SM에 할당된 CTA 개수가 증가함에도 불구하고 성능의 변화가 없는 경우이다. 이와 같은 경우는 동시에 수행 가능한 CTA 수가 정해져 있거나 생성된 CTA 개수가 적은 경우로 예상할 수 있다. 특정 벤치마크 프로그램에서 동시에 수행할 수 있는 CTA 개수가 정해져 있는 경우 각 SM에 많은 수의 CTA를 할당하여도 성능은 동일하다. 또한, 생성된 CTA 개수가 적은 경우는 다수의 SM을 가진 GPU에서 SM의 개수보다 생성된 CTA의 개수가 적기 때문에 많은 수의 CTA를 SM에 할당시켜도 성능은 일정하다.

세 번째, SM에 할당된 CTA 개수가 증가하여도 성능이 감소하는 경우이다. 이와 같은 경우는 GPU에 할당된 CTA 개수 증가에 따라 메모리 충돌로 인한 스톨이 발생하거나 SM 스톨 문제가 발생한 경우이다. 메모리 충돌로 인한 스톨은 한정된 메모리 자원에 많은 수의 메모리 접근이 요청되면 메모리 충돌로 인한 스톨 문제가 발생한다. SM에 할당되는 CTA 개수가 증가함에 따라 메모리를 통한 데이터 접근이 더 많이 발생할 수 있으므로 메모리 충돌로 인한 스톨에 의해서 성능 저하가 발생한다. 또한, SM 스톨은 파이프라인이 스톨 되거나 워프가 이슈되지 않는 문제로 발생하는데, SM에 할당되는 CTA 개수 증가에 따라 파이프라인 처리를 위한 하드웨어 자원이 부족하게 되어 파이프라인이 스톨 되거나 워프가 아직 이슈 되지 않아 파이프라인에서 처리할 워프가 부족하여 파이프라인 스톨이 발생한다.

첫 번째, SM에 할당되는 CTA 개수가 증가함에 따라 성능이 증가한 벤치마크 프로그램은 HW가 있다. 두 번째, SM에 할당되는 CTA 개수가 증가함에도 불구하고 성능의 변화가 없는 경우는 HW, KM, SimTem이 있다. HW 벤치마크 프로그램의 경우 SM에 할당되는 CTA 개수가 증가함에 따라 성능이 증가하지만 SM에 할당되는 CTA가 2개 이상이 되면 성능은 고정된다. 마지막, SM에 할당된 CTA 개수가 증가하여도 성능이 감소하는 경우는 BFS, MUM, SQRNG 벤치마크 프로그램이다. BFS 벤치마크 프로그램은 SM에 할당되는 CTA 개수가 1개부

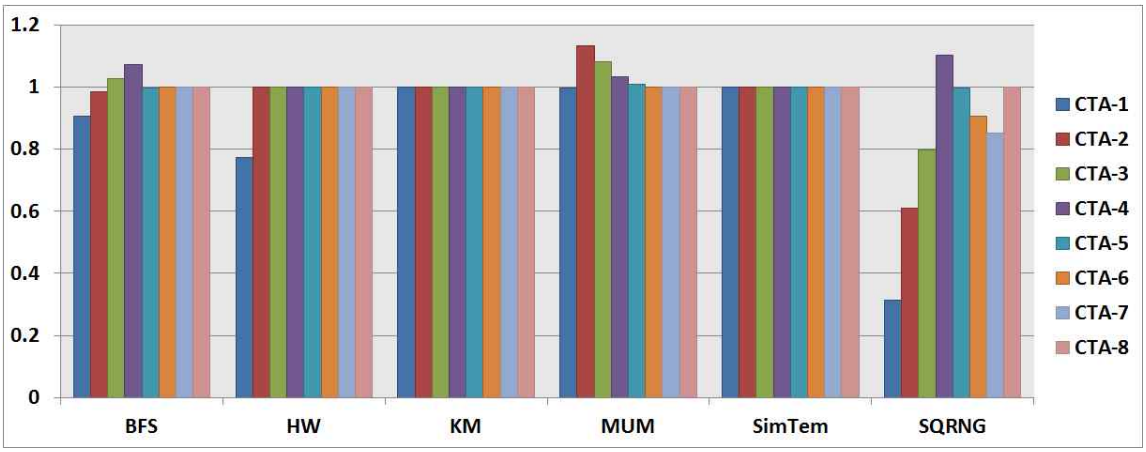

Fig. 3. GPU performance according to number of CTAs assigned to the SM

터 4개까지 성능의 증가가 발생하지만, SM에 할당된 CTA 개수가 5개부터 8개까지는 성능이 감소한다. MUM 벤치마크 프로그램은 GPU에 할당되는 CTA 개수가 2개일 경우 성능이 증가하였지만, SM에 할당되는 CTA 개수가 더 증가함에 따라 성능이 감소하였다. SQRNG 벤치마크 프로그램의 경우 SM에 할당되는 CTA 개수가 4개일 경우 가장 높은 성능을 보이며, SM에 할당되는 CTA 개수가 증가함에 따라 지속적으로 성능이 감소하다가 SM에 할당되는 CTA 개수가 8개일 경우 성능이 다시 증가한다.

이처럼 특정 벤치마크 프로그램에서는 SM에 할당되는 CTA 개수에 따라 성능의 감소가 발생하게 된다. 따라서, SM에 무조건 많은 수의 CTA를 할당하는 것은 GPGPU 구조에서 최적의 성능을 보여주지 않는다. 그러므로 각 벤치마크 프로그램에 맞는 최적의 CTA 할당이 필요하다. 실험결과를 보면 CTA 스케줄링 기법을 통해 SM에 할당되는 CTA 개수 조절을 통한 GPGPU의 성능 향상이 가능하다. 다음 장에서는 이와 같은 문제를 해결하기 위한 IPC 기반 동적 CTA 스케줄링 기법에 대한 실험 결과를 분석한다.

2. Performance according to IPC-based Dynamic CTA Scheduling Scheme

앞 장에서는 SM에 할당되는 CTA 개수에 따른 성능 변화를 분석하였다. 분석 결과 SM에 CTA 개수가 증가하여도 성능이 선형적으로 증가하지 않음을 확인할 수 있었다. 오히려 SM에 할당되는 CTA 개수가 증가함에 따라 성능이 감소하는 경우도 발생하였다. 따라서, 본 논문에서는 SM에 할당하는 CTA 개수를 효율적으로 선택하기 위해서 IPC 기반 동적 CTA 스케줄링 기법을 사용하였다. 그림 4는 벤치마크별 IPC 기반 동적 CTA 스케줄링 기법이 적용된 GPU 성능 결과이다.

각 벤치마크 프로그램에 따른 IPC 기반 동적 CTA 스케줄링 기법을 적용하였다. 실험 결과는 그림 3과 같이 3가지 유형으로 분류할 수 있다. 첫 번째, IPC 기반 동적 CTA 스케줄링 기법을 적용하여 성능이 증가한 경우이다. 이와 같은 경우는 IPC 기반 동적 CTA 스케줄링 기법을 사용하여 벤치마크 프로그램에 맞는 최적의 CTA 수를 유지했기 때문으로 판단된다. BFS, MUM, SQRNG 벤치마크 프로그램에서 기존의 방식인 CTA를 8개 할당하는 것보다 6.7%, 13.1%, 7.0%의 성능향상을 보인다.

두 번째, IPC 기반 동적 CTA 스케줄링 기법을 적용함에도 성능 변화가 없는 경우이다. 이와 같은 경우는 KM, SimTem 벤치마크 프로그램에서 확인할 수 있다. 이러한 벤치마크 프로그램은 SM에 할당되는 CTA 개수가 1개부터 8개까지 변화하여도 성능의 변화가 발생하지 않기 때문에 IPC 기반 동적 CTA 스케줄링 기법을 적용하여도 성능의 변화는 없다.

세 번째, IPC 기반 동적 CTA 스케줄링 기법을 적용하여 성능이 감소한 경우이다. 이와 같은 경우는 SM에 할당하는 CTA가 1개부터 8개까지 증가하기 때문에 가장 높은 성능을 보이는 8개의 CTA 할당까지 성능의 손실이 발생한다. HW 벤치마크 프로그램에서 1.0% 성능의 감소가 발생한다.

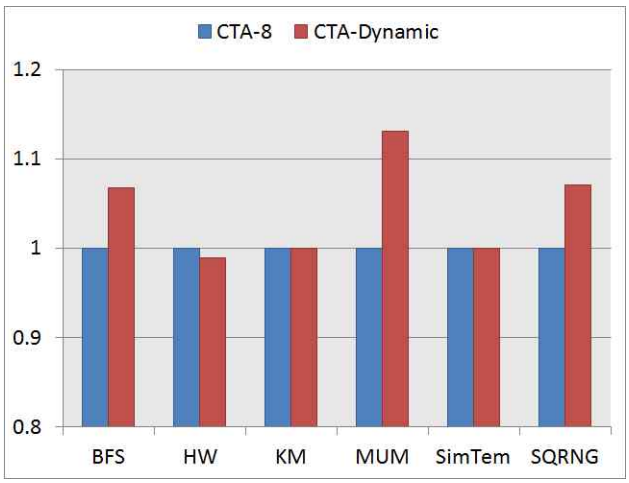

Fig 4. GPU performance according to IPC-based Dynamic CTA Scheduling Scheme

본 장에서는 SM에 할당되는 CTA 개수 변화에 따른 성능 분석과 IPC 기반 동적 CTA 할당기법을 적용하여 기존의 CTA 스케줄링 기법과의 성능 비교를 수행하였다. 앞서 말한 바와 같이 다수의 CTA 할당은 병렬성의 증가와 메모리 접근시간 감소의 장점이 있다. 따라서, CTA 할당 개수의 증가는 작업대기의 감소와 메모리 대기시간 감소의 장점이 있다. 하지만, CTA 할당 개수의 증가에 따라 SM 스톨이 증가하는 단점이 있다. CTA 할당 개수에 따라 발생하는 장단점이 존재함으로 최적의 CTA 할당 개수를 찾는 것은 시스템 성능 향상에 있어서 중요한 부분이다. 본 논문에서 제안된 기법은 이러한 문제를 해결하기 위해서 각 벤치마크 프로그램에 맞는 최적의 CTA 할당 개수를 동적으로 변경하여 최적의 성능을 보여준다.

실험 결과에서 보는 바와 같이 모든 벤치마크 프로그램에 최대한의 CTA를 할당하는 것은 최적을 성능을 보이지 않음을 알 수 있다. 제안된 기법은 6개의 벤치마크 프로그램에서 평균 6.3%의 성능이 향상됨을 확인할 수 있으며, 본 논문에서는 실험을 통하여 IPC 기반 동적 CTA 스케줄링 기법을 통해 성능을 증가시킬 수 있음을 확인하였다.

VI. Conclusions

본 논문에서는 SM에 할당되는 CTA 개수에 따른 성능을 분석하기 위해서 다양한 벤치마크 프로그램을 수행하여 1개부터 8개까지의 CTA를 각 코어에 할당하였다. 시뮬레이션을 통해 SM에 할당된 CTA 개수 변화에 따른 GPU 성능을 분석할 수 있었다. 실험 결과, SM에 할당되는 CTA 개수 변화에 따라 성능의 증가, 고정적인 성능, 성능의 감소를 확인할 수 있었으며, 분석을 통하여 성능 변화의 원인을 확인할 수 있었다. SM에 할당되는 CTA 개수는 각 벤치마크 프로그램별로 다르며, 최대한 많은 수의 CTA를 할당한다고 해서 최적의 성능을 보이지는 않았다. 또한, 각 벤치마크 프로그램별 SM에 적합한 CTA 개수를 적용하기 위해서 IPC 기반 동적 CTA 스케줄링 기법을 적용하여 실험을 수행하였다. 실험 결과, IPC 기반 동적 CTA 스케줄링 기법이 적용된 벤치마크 프로그램은 기존의 정적 CTA 할당 기법과 비교하여 높은 성능을 보임을 알 수 있었다. 제안된 기법은 모든 벤치마크 프로그램에 쉽게 적용이 가능함으로 대부분의 벤치마크 프로그램에서 성능 향상을 보일 것으로 예상된다.

최신 GPU는 많은 연구가 진행되고 있으며, 그중 GPGPU를 활용하는 연구는 컴퓨팅 시스템의 성능을 크게 향상시킬 수 있는 방법이다. 본 논문에서 분석한 SM에 할당되는 CTA 개수에 따른 GPU 성능 변화와 IPC 기반 동적 CTA 스케줄링 기법을 잘 활용한다면 GPU 설계에 있어서 제품경쟁력을 높일 수 있을 것이다. 향후 연구에서는 SM에 할당되는 CTA 개수에 따른 전력 소비량을 분석하고 이를 통해서 성능과 전력 효율성에 있어서 최적의 값을 갖는 CTA 개수를 찾는 동적 CTA 기법을 개발해보고자 한다.

REFERENCES

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, “Clock rate versus IPC: the end of the road for conventional microArchitectures,” In Proceedings of 27th International Symposium on Computer Architecture, pp. 248-259, 2000.

[2] Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, “Brook for GPUs: stream computing on graphics hardware,” In Proceedings of 31th Annual Conference on Computer Graphics, pp.777-786, 2004.

[3] NVIDIA CUDA Programming, available at http://www.nvidia.com/object/cuda_home_new.html

[4] OpenCL, available at http://www.khronos.org/opencl/

[5] ATI Streaming, available at http://www.amd.com/stream

[6] General-purpose computation on graphics hardware, available at http://www.gpgpu.org

[7] I. A. Buck, “Programming CUDA,” In Supercomputing 2007 Tutorial Notes, 2007.

[8] A. Jog, O. Kayiran, N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, “OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance,” In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 395-406, 2013.

[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,” In Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 308-317, 2011.

[10] T. G. Rogers, M. O'Connor, and T. M. Aamodt, “Cache-Conscious Wavefront Scheduling,” In Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 78-85, 2012.

[11] V. W. Lee, C. K. Kim, J. Chhugani, M. Deisher, D. H. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal and P. Dubey, "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," In proceedings of International Symposium on Computer Architecture, pp.451-460, 2010.

[12] NVIDIA’s Next Generation CUDA Compute Architecture: Fermi, available at

www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_ architecture_whitepaper.pdf

[13] J. E. Thornton, "Parallel operation in the control data 6600," In AFIPS Proceedings of FJCC, Part.2, Volume.26, pp.33-40, 1964.

[14] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," In Proceedings of Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp.777-786, 2004.

[15] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling," In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), pp.260-271, 2014.

[16] 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 9th International Symposium on Performance Analysis of Systems and Software, pp.163-174, 2009.

[17] S. Li, J. H Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," In Proceedings of the International Symposium on Microarchitecture, pp.469-480, 2009.

[18] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," In Proceedings of the International Symposium Computer Architecture, pp.487-498, 2013.

[19] CUDA SDK, available at http://developerdownload.nvidia.com/compute/cuda/sdk/website/samples.html

[20] 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.

Authors

Dong Oh Son received the B.S. degree and M.S. in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2010 and 2012 respectively.
Currently, he is pursuing his Ph.D. at Chonnam National University. His research interests include computer architecture, embedded 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 School of Computer Engineering and Information Technology at the University of Ulsan, Ulsan, Korea, where he is currently an Associate 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 an Associate 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.