http://dx.doi.org/10.5392/JKCA.2013.13.03.009

범용 응용프로그램 실행 시 하드웨어 구성과 분기 처리 기법에 따른 GPU 성능 분석

Analysis of Impact of Correlation Between Hardware Configuration and Branch Handling Methods Executing General Purpose Applications

최홍준, 김철홍
전남대학교 전자컴퓨터공학부

Hong Jun Choi(chj6083@gmail.com), Cheol Hong Kim(chkim22@chonnam.ac.kr)

요약

GPU의 연산 능력과 유연성이 강화됨에 따라, GPU는 그래픽 응용프로그램뿐만 아니라 범용 응용프로그램도 수행한다. 특히, GPU 회사들이 제공하는 API를 활용함으로써 프로그래머들은 보다 쉽게 GPGPU 응용프로그램을 작성할 수 있다. 하지만 대부분의 범용 응용프로그램은 분기 명령어를 많이 포함하고 있기 때문에, 범용 응용프로그램을 수행하는 경우 GPU의 연산 자원을 충분히 활용할 수 없다. 분기 명령어를 처리하기 위해서 다양한 워프 생성 기법들이 제안되었다. GPU 구조에서는 높은 연산 자원 활용률을 보이는 워프 생성 기법이 우수한 성능을 보일 것으로 예상된다. 하지만 예상과는 달리, 실험 결과에 따르면 높은 연산 자원 활용률을 보이는 워프 생성 기법의 성능이 상대적으로 낮은 연산 자원 활용률을 보이는 워프 생성 기법의 성능보다 낮게 나타난다. 높은 연산 자원 활용률을 보이는 워프 생성 기법에서 유발한 많은 메모리 요구로 인한 심각한 메모리 병목 현상이 원인으로 분석된다. 그러므로 적절한 하드웨어 지원이 없는 경우, 높은 연산 자원 활용률이 반드시 우수한 성능을 보장한다고 할 수 없다. 이러한 이유로, 본 논문에서는 하드웨어 자원과 워프 생성 기법사이의 상관관계에 대한 상세한 분석을 수행하고자 한다. 본 논문의 분석 결과는 분기 명령어에 의해 발생된 GPU의 성능 저하 문제를 해결하고자 할 때 중요한 가이드라인이 될 것이다.

■ 중심어 : | 그래픽 처리장치 | GPGPU | 범용 응용프로그램 | 분기 명령어 | 워프 생성 기법 |

Abstract

Due to increased computing power and flexibility of GPU, recent GPUs execute general purpose parallel applications as well as graphics applications. Programmers can use GPGPU by using the APIs from GPU vendors. Unfortunately, computational resources of GPU are not fully utilized when executing general purpose applications because of frequent branch instructions. To handle the branch problem, several warp formations have been proposed. Intuitively, we expect that the warp formations providing higher computational resource utilization show higher performance. Contrary to our expectations, according to simulation results, the performance of the warp formation providing better utilization is lower than that of the warp formation providing worse utilization. This is because warp formation providing high utilization causes serious memory bottleneck due to increased memory request. Therefore, warp formation providing high computation utilization cannot guarantee high performance without proper hardware resources. For this reason, we will analyze the correlation between hardware configuration and warp formation. Our simulation results present the guideline to solve the underutilization problem due to branch instructions when designing recent GPU.

■ keyword : | GPU | GGPPU | General-purpose Application | Branch Instruction | Warp Formation |

* 본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단 기초연구사업(2012R1A1B4003492)의 지원과 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터(NIPA-2012-H0301-12-3005)의 지원을 받아 수행되었음.

접수번호 : #130107-002
접수일자 : 2013년 01월 07일
심사완료일 : 2013년 03월 07일
교신저자 : 김철홍, e-mail : chkim22@chonnam.ac.kr

I. 서 론

공정기술의 지속적인 발달과 트랜지스터 집적도의 증가는 마이크로프로세서의 성능 향상에 주도적인 역할을 담당하기도 하지만, 마이크로프로세서에서 소모하는 전력을 빠르게 증가시키는 주된 원인이 되기도 한다[1]. 그러므로 마이크로프로세서 개발자들은 성능 향상과 더불어 소모 전력을 최소화할 수 있는 다양한 기법들을 개발하고자 연구를 진행한다. 병렬성을 활용하는 기법은 전력 소모량 증가로 인한 문제를 해결함과 동시에 컴퓨팅 시스템의 성능을 향상시키는 대표적인 기법이다. 몇 년 전까지 진행된 병렬성 관련 연구들의 대부분은 명령어 수준의 병렬성(ILP: Instruction Level Parallelism)을 활용한다[2]. 불행히도 명령어 수준의 병렬성은 복잡한 스케쥴링 로직과 큰 캐쉬를 필요로 하기 때문에, 명령어 수준의 병렬성 활용 기법을 통해 얻을 수 있는 성능과 전력 효율성의 이득은 줄어드는 반면 관련 비용은 점점 높아지는 추세이다. 명령어 수준의 병렬성과는 달리, 스레드 수준 병렬성 활용 기법(TLP: Thread Level Parallelism)은 소프트웨어 내의 스레드를 활용하여 연산 처리량을 증가시키기 때문에 성능 향상을 위한 비용이 상대적으로 적게 발생한다. 이에 따라, 최근에는 적은 비용으로 높은 성능 향상을 얻을 수 있는 스레드 수준 병렬성 활용 기법이 관심 받고 있다[3][4].

일반적으로 단일 스레드 또는 소수의 멀티스레드를 수행하는 중앙 처리장치(CPU: Central Processing Unit)와는 달리, 많은 스레드들을 동시에 수행할 수 있는 연산 자원을 보유한 그래픽 처리장치(GPU: Graphics Processing Unit)는 높은 수준의 멀티스레딩을 지원할 수 있는 대표적인 병렬 연산 처리 장치이다[5]. GPU는 CPU가 그래픽 작업을 직접 처리하는 과정에서 발생하는 병목 현상으로 인한 시스템 성능 저하 문제를 해결하기 위하여 개발된 그래픽 처리 특화 프로세서이다. 기존의 GPU는 렌더링 작업을 처리하기 위하여 수행하는 연산을 지원하는 고정된 하드웨어로 구성되었기 때문에, 그래픽 렌더링 작업의 효율성을 높이기 위하여 개발된 그래픽 알고리즘을 적용하고자 하는 경우 알고리즘에 따른 연산을 수행할 수 있는 새로운 하드웨어를 필요로 한다. 새로운 GPU를 개발하는 것은 매우 많은 비용과 시간을 필요로 하기 때문에, 최근의 GPU들은 고정된 연산을 수행하는 하드웨어가 아닌 프로그램 가능한 하드웨어로 구성된다. 프로그램 가능한 하드웨어인 통합형 세이더코어(unified shader core)는 고정된 그래픽 파이프라인 대신에 다양한 그래픽 파이프라인을 수행 할 수 있는 대표적인 GPU의 연산 장치이다[6].

GPU가 프로그램 가능한 연산 장치로 구성됨에 따라, 최신의 GPU는 강력한 병렬 연산 처리 능력과 더불어 유연성 또한 갖추고 있다. GPU의 유연성과 병렬 처리 능력을 활용하기 위하여, 최근 컴퓨팅 시스템에서는 그래픽 분야 외에도 과학, 수학, 신호, 이미지 처리 분야 등과 같이 다양한 분야에서 CPU 대신에 GPU를 사용한다. 이와 같이, GPU를 활용하여 일반 응용프로그램을 수행하는 기술을 그래픽 처리장치 병렬컴퓨팅(GPGPU: General Purpose computation on the Graphics Processing Unit)이라고 한다[7]. 초기의 GPGPU를 활용한 소프트웨어 개발은 그래픽 파이프라인에 대한 높은 이해도를 요구하였기 때문에, 개발자들이 쉽게 이용할 수 없었다. 하지만 최근 들어, GPGPU를 쉽게 이용할 수 있는 응용프로그램 인터페이스기술들을 GPU 회사들이 제공함에 따라, 개발자들이 쉽게 GPU를 활용하여 일반 응용프로그램들을 수행할 수 있다. 대표적인 GPGPU 기술로는 NVIDIA의 CUDA (Compute Unified Device Architecture), Khronos Group의 OpenCL 등이 존재한다[8][9].

일반적으로, GPU는 단일 명령어로 다수의 스레드들을 수행 할 수 있는 구조로 구성되어 있으므로 GPU에서는 다수의 스레드들이 병렬적으로 수행될 수 있도록 스레드들을 단일 명령어 다중 데이터(SIMD: Single Instruction Multiple Data)를 기반으로 집단화한다. 본 논문에서는 집단화된 단일 명령어 다중 데이터 그룹들을 워프(warp)라 부른다. 스레드들의 집합인 워프는 GPU에서 연산을 수행하는 데 있어서 가장 기본적인 단위이다. 세이더코어에서 워프를 수행하는 경우, 워프내의 스레드들은 동일한 명령어로 순차적 명령어 실행 순

서(in-order) 방식으로 락스텝(lock-step)에 맞춰 수행된다[5][10][11]. 앞서 설명한 바와 같이, GPU의 유연성과 연산 능력이 빠르게 향상됨에 따라 GPU를 그래픽 관련 작업뿐만 아니라 범용 작업에도 활용하고자 다양한 연구가 집중적으로 이루어지고 있다. 대표적인 예로서, GPU 회사들이 제공한 인터페이스 기술들은 프로그래머들이 보다 쉽고 효율적으로 GPGPU 기술을 사용할 수 있도록 지원한다.

GPGPU를 활용하여 컴퓨팅 시스템의 성능을 향상시키고자 하는 노력에도 불구하고, 대부분의 범용 응용프로그램을 수행하는 경우의 성능 향상은 크지 않다. 성능 향상이 크지 않은 이유는 분기 명령어가 존재하지 않거나 극히 소수인 그래픽 프로그램과는 달리, 많은 분기 명령어를 포함한 범용 응용프로그램에서는 스레드 수준의 병렬성을 확보하기 어렵기 때문이다[12]. 분기 명령어가 병렬성을 저하시키는 원인은 다음과 같다. 워프는 다수의 스레드로 구성되어 있으며 각 스레드들은 동일한 명령어를 수행한다. 하지만, 분기 명령어를 수행하는 워프내의 스레드들은 분기 결과에 따라 다른 명령어를 수행하게 된다. 다시 말하면, 하나의 워프가 동시에 다른 명령어를 수행해야 한다. 앞서 설명한 바와 같이, GPU는 단일 명령어 다중 데이터 구조로 구성되어 있기 때문에 분기 명령어를 수행하는 워프는 단일 명령어 다중 데이터 구조에 더 이상 적합하지 않게 된다. 즉, 분기 명령어로 인하여 하나의 워프에서 다른 명령어 흐름이 발생하는 경우 단일 명령어로 워프안의 모든 스레드를 수행할 수 없는 문제가 발생한다는 것이다. GPGPU 기술을 지원하기 위해서, 대부분의 GPU는 분기 명령어를 허용하고 있으며 분기 명령어를 처리하기 위한 기법이 적용되어 있다. 하지만, 분기 명령어를 처리하기 위해 적용된 기법들은 GPU 내부의 단일 명령어 다중 데이터 파이프라인에 상당한 손해를 유발하여 성능을 저하시킨다. 그러므로 GPGPU 기술을 활용하고자 하는 경우 분기 명령어로 인해 발생하는 흐름제어(control-flow)를 효과적으로 처리하는 것이 매우 중요하다. 분기 명령어로 인한 흐름제어를 효과적으로 처리할 수 있다면, 연산 밀도가 향상되어 GPU의 연산 자원을 효율적으로 활용할 수 있기 때문이다. 최신 GPU에서는 분기 명령어로 인한 성능 저하 문제를 극복하기 위하여 워프를 생성하는 시점에서 워프 생성(warp formation) 기법을 적용하여 분기를 처리한다. GPU를 위해 개발된 워프 생성 기법에는 단일 명령어 다중 데이터 직렬화(SIMD serialization), 단일 명령어 다중 데이터 재수렴(SIMD reconvergence), 단일 명령어 다중 데이터 동적 워프 생성(SIMD dynamic warp formation) 등이 있다[13-18].

일반적으로 구조적 해저드(structural hazard)가 발생하지 않는 이상적인 GPU 구조에서는 높은 연산 자원 활용률을 보이는 워프 생성기법을 적용하는 것이 높은 성능을 보일 것은 분명하다. 하지만, 실제 상용 GPU의 연산 및 주변 하드웨어 자원에는 한계가 존재하기 때문에 높은 연산 자원 활용률이 반드시 높은 성능을 나타내는 지는 확실하지 않다. 그러므로 본 논문에서 우리는 하드웨어 구성과 워프 생성 기법이 GPU의 성능에 미치는 영향에 대해 분석하고자 한다. 하드웨어 구성과 워프 생성 기법의 상관관계 분석 결과는 GPGPU를 지원하는 GPU를 설계하고자 하는 경우 큰 도움이 될 것이라 기대된다.

본 논문에서는 실험 결과를 평균값을 사용하여 표현한다. 평균값을 표현하는 데 있어서 절대치의 총 합을 사용한다면 효율성을 상대적인 비율로 보여주기 매우 어렵다. 그러므로 본 논문에서는 상대적인 증감을 명확하게 보여주는 기술 방법으로 널리 사용되는 표현 방식인 정규화를 사용한다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로써 GPU 구조와 GPGPU, 워프 생성 기법에 대하여 설명한다. 그리고 3장에서 실험 환경을 상세히 보여주고 4장에서는 실험 결과 및 자세한 분석을 기술한다. 마지막으로 5장은 본 논문의 결론을 맺고 향후 연구 방향을 서술한다.

II. 관련 연구

본 장에서 우리는 GPU의 기본 구조와 GPGPU 그리고 워프 생성 기법에 대하여 설명하고자 한다.

1. GPU 구조

현재, 대표적인 GPU 제조사는 크게 AMD(ATI)와 NVIDIA가 있으며 제조사에 따라 GPU에서 동일한 기능을 수행하는 장치 또는 동일한 정의를 표현하는 용어가 상이한 경우가 존재한다. 본 논문에서는 상이한 용어 사용으로 인한 혼란을 줄이고자, 특별한 언급이 없다면 NVIDIA사에서 정의하는 용어를 사용한다.

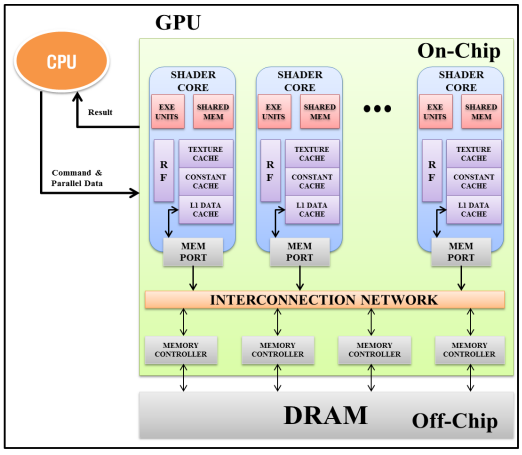

그림 1. 일반적인 GPU 구조

[그림 1]은 일반적인 GPU의 구조를 보여주고 있다. 그림에서 보이듯이, CPU는 GPU에게 작업 명령과 병렬 데이터를 넘겨주고 GPU는 작업을 수행한 결과를 CPU에게 반환한다. GPU는 다수의 세이더코어(SHADER CORE)와 내부 연결망(INTERCONNECTION NETWORK) 그리고 DRAM으로 구성되어 있다. GPU 내부에 존재하는 각 세이더코어는 동일한 코드를 수행하는 다수의 스레드들을 동시에 실행시키며, 텍스쳐 캐쉬(TEXTURE CACHE), 상수 캐쉬(CONSTANT CACHE) 그리고 L1 데이터 캐쉬(L1 DATA CACHE)와 같은 지역 메모리를 가지고 있다. 지역 메모리 중 텍스쳐 캐쉬와 상수 캐쉬는 읽기 전용(read-only)이다.

세이더코어에서 수행되는 스레드들의 명령어는 세이더코어 내부의 연산 처리 장치(EXE UNITS)에서 순차적 명령어 실행 순서(in-order) 방식으로 수행된다. 세이더코어는 실행할 명령어가 선택이 되면, 상응되는 피연산자를 다수의 뱅크로 구성된 레지스터 파일(RF: Register File)로부터 읽어온다. 만약, 레지스터 파일에 원하는 데이터가 존재하지 않으면(miss) 세이더코어는 명령어 수행을 위해 필요한 데이터를 다수의 뱅크로 구성된 지역 메모리에 접근하여 요청한다. 불행히도 지역 메모리에서 메모리 요구(또는 데이터 요구)가 실패(miss)할 경우, 해당 메모리 요구가 하위 레벨 캐쉬 또는 DRAM으로 접근할 수 있도록 메모리 포트(MEM PORT)를 통해 요청한다. 특히, DRAM으로 접근되는 메모리 요구는 내부 연결망을 통해 메모리 컨트롤러(MEMORY CONTROLLER)에 의해 처리된다[19][20].

2. GPGPU

최근에 GPU의 연산 능력이 향상됨에 따라, GPU는 자신의 유휴 연산 자원을 활용하기 위하여 그래픽 연산 처리뿐만 아니라 범용 목적 연산 처리에도 사용되고 있다. 범용 데이터 처리에 GPU의 유휴 연산 자원을 활용할 수 있는 기반에는 크게 두 가지 기술이 존재한다. 하나는 [그림 1]에서 보여주고 있는 GPU를 활용한 프로그램 가능한 파이프라인 구조이다. 다른 하나는 범용 응용 프로그램을 수행하는 데 있어서 GPU를 보다 쉽게 활용할 수 있는 응용프로그램 인터페이스의 등장이다[7].

앞서 설명한 바와 같이, 기존의 GPU 파이프라인은 각각 고정된 역할을 수행하도록 구성되었지만 GPU 설계 기술이 발전함에 따라서 각각의 파이프라인 단계가 프로그래밍에 유연한 구조로 변화하는 중이다. 예를 들어, 최신의 GPU에서는 버텍스(vertex) 처리 단계나 프레그먼트(fragment) 처리 단계를 프로그래밍 가능한 버텍스 프로세서나 프레그먼트 프로세서로 구현하여 프로그래머가 그래픽 처리를 직접 프로그래밍 할 수 있도록 지원하고 있다. 그러므로 현재 생산되는 상용 GPU는 그래픽 작업의 대용량 병렬 데이터를 동시에 병렬적으로 처리할 수 있다[6][21].

기존에도 GPU를 활용하기 위하여 OpenGL 세이딩 언어(OpenGL shading language)나 DirectX와 같은 응용프로그램 인터페이스들이 존재하였으나, 이들을 이용하여 GPU를 범용 연산에 이용하는 프로그램을 작성하기에는 많은 시간과 비용이 요구되었기에 비효율적

이었다. 하지만, 최근 GPU 회사들이 GPU의 활용도와 경쟁력을 향상시키기 위하여 CUDA, ATI Stream, OpenCL, Cg, HLSL, OpenGL 등 GPGPU를 개발 할 수 있는 고급 언어를 제공하는 추세이다[8][9][22-25]. 고급 언어들의 등장으로 개발자들은 GPGPU 응용프로그램을 보다 쉽게 작성할 수 있다. 특히, NVIDIA의 CUDA는 C 기반으로 개발된 데이터 병렬 처리 프로그래밍 언어로써, 범용 연산 처리에 GPU를 활용하는 고급 언어들 중 가장 널리 사용되고 있다[8].

표 1. 대표적인 GPGPU 개발 플랫폼

개발 회사	GPGPU 기술	특징
NVIDIA	CUDA[8]	C 프로그래밍 언어를 포함한 산업 표준 언어를 기반으로 병렬처리 알고리즘을 GPU에서 수행할 수 있도록 프로그램을 작성할 수 있도록 도와주는 대표적인 GPGPU 기술
AMD(ATI)	Stream Technology [22]	CPU나 GPU를 효과적으로 활용하여 동영상 인코딩 및 물리연산 등을 빠르게 처리하여 시스템 성능을 향상시켜 주는 스트림 프로세서 기술
Khronos Group	OpenCL[9]	C99에 기반하여 태스크 병렬성을 갖는 프로그램을 CPU와 GPU 그리고 기타 프로세서로 이루어진 다중 플랫폼에서 구동할 수 있도록 데이터를 작성할 수 있는 개방형 범용 병렬 컴퓨팅 프레임워크

3. 워프 생성 기법

GPU에서 수행 중이던 워프에서 분기 명령어가 실행되는 경우, 분기 명령어의 실행 결과에 따라 워프 내의 스레드들은 서로 다른 명령어를 가진다. 단일 명령어 다중 데이터 파이프라인에서는 동일한 사이클에 하나의 명령어만을 수행할 수 있기 때문에, 분기 명령어를 허용하기 위해서는 분기를 처리하는 기법이 필요하다. 본 절에서는 단일 명령어 다중 데이터 구조에서 분기 명령어를 처리하기 위해 제안된 대표적인 워프 생성 기법들을 소개하고자 한다.

3.1 단일 명령어 다중 데이터 직렬화 (SIMD Serialization)[13][14]

단일 명령어 다중 데이터 직렬화는 분기 명령어를 수행하자마자, 워프 안의 스레드들을 직렬화함으로써 분기를 처리하는 기법이다. 단일 명령어 다중 데이터 직렬화 기법은 구현과 이해가 쉬운 반면에 컴퓨팅 시스템의 성능을 유지할 수 없다. 분기 명령어로 인해 워프안의 스레드들을 직렬화하면 하나의 워프는 다수의 워프들로 분할되고, 분할된 각각의 워프들은 각자 독립적으로 스케쥴링 되어 하나의 명령어를 수행한다. 단일 명령어 다중 데이터 직렬화 기법에서는 한 번 분할된 워프들은 다시 하나의 워프로 모이지 않는다. 다시 말하면, 단일 명령어 다중 데이터 직렬화 기법에는 결합점(merging point, reconvergence point)이 존재하지 않기 때문에, 워프 안의 스레드들이 종료될 때까지 워프는 분기 명령어를 만나면 계속해서 분리된다. 그러므로 단일 명령어 다중 데이터 직렬화 기법은 분기 명령어가 많은 프로그램을 수행하는 경우 연산 자원 활용률을 심각하게 저하시킨다.

3.2 단일 명령어 다중 데이터 재수렴 (SIMD Reconvergence)[15-17]

분기 명령어가 수행되는 경우, 하나의 경로는 테이큰(taken)이고 다른 하나의 경로는 언테이큰(untaken)이다. 단일 명령어 다중 데이터 구조에서 분기를 처리하는 대표적인 방법인 단일 명령어 다중 데이터 재수렴(SIMD reconvergence) 기법에서는 분기 명령어로 인하여 분기 된 경우, 워프를 분기 결과에 따라서 각 경로마다 수행한다. 다시 말하면, 워프 내에서 테이큰(taken)된 스레드들만으로 한 번 수행하고 언테이큰(untaken) 된 스레드들만으로 한 번 수행한다. 그리고 분리되어 명령어를 수행한 워프내의 스레드들은 분기 명령어 수행 후, 결합점에 도달하여 하나의 경로를 갖는 워프로 다시 합쳐진다. 단일 명령어 다중 데이터 재수렴 기법에서 널리 사용되는 Post-dominator 기법은 분기 명령어를 처리하기 위하여 최신 상용 GPU에 대부분 적용된다[17]. Post-dominator 기법에서 지시자(dominance)는 코드 최적화를 위해 컴파일러에서 식별된다. post-dominator 기법에서는 분할된 워프들을 결합점에서 합치기 위해서 필요한 정보들을 분기 스택(divergence stack)을 이용하여 저장 및 사용한다. 분기

스택의 각 엔트리는 PC, 활성화 스레드 정보를 나타내는 활성화 마스크(active mask) 그리고 결합 명령어(reconvergence PC) 등 3가지 정보를 가지고 있다. 활성화 스레드 정보는 해당 PC를 수행하는 경우에 워프 내의 스레드들의 연산 동작 여부를 나타내며 결합 명령어는 결합점의 명령어 정보를 나타낸다.

3.3 단일 명령어 다중 데이터 동적 워프 생성 (SIMD dynamic warp formation)[18]

Post-dominator 기법에서는 워프가 생성되고 난 뒤에 워프가 모든 명령어를 수행해 종료될 때까지, 워프 안의 스레드들은 동일한 명령어를 수행한다. 단일 명령어 다중 데이터 동적 워프 생성기법에서도 생성된 워프 안의 스레드들은 동일한 명령어를 수행한다. 그러나 Post-dominator 기법에서 워프 내의 모든 스레드가 종료될 때까지 유지되는 반면, 단일 명령어 다중 데이터 동적 워프 생성 기법에서는 워프 내의 스레드들이 하나의 명령어를 수행하고 난 다음 스레드 그룹(워프)을 해지한다. 그리고 워프 내의 각 스레드들의 PC 값을 기반으로 동일한 명령어를 가진 새로운 워프를 생성한다. 새롭게 생성된 워프가 스케쥴러에 의해 선택되면, 선택된 워프 내의 스레드들은 동일한 명령어를 동시에 수행한다. 결과적으로 단일 명령어 다중 데이터 동적 워프 생성 기법에서 워프들은 항상 하나의 명령어로 모든 스레드들을 수행하기 때문에, 연산 자원 활용률이 매우 높다.

일반적으로 높은 연산 자원 활용률은 우수한 성능을 보여줄 것으로 예상되기 때문에, 기존의 연구들 중 대부분은 연산 자원 활용률을 증가시키기 위한 내용이다. 하지만 실제 GPU는 제한된 하드웨어 자원을 가지고 있기 때문에, 높은 연산 자원 활용률을 보이는 분기 처리 기법이 반드시 우수한 성능을 보이는지는 확실하지 않다. 그러므로 본 논문에서는 각기 다른 연산 자원 활용률을 보이는 분기 처리 기법의 효율성을 다양한 하드웨어 구성에 따라 상세하게 분석하고자 한다. 본 논문에서 활용되는 GPU에서 분기 명령어를 지원하기 위해 적용된 워프 생성 기법은 본 장에서 소개한 단일 명령어 다중 데이터 직렬화, 단일 명령어 다중 데이터 재수렴, 단일 명령어 다중 데이터 동적 워프 생성 등으로 매우 상이한 연산 자원 활용률을 보여준다.

Ⅲ. 실험 환경

1. 모의실험 환경

본 논문에서는 GPU의 하드웨어 구성과 워프 생성 기법에 따른 성능 변화를 평가하기 위해 신뢰성 있는 시뮬레이터인 SimpleScalar[26]를 기반으로 개발된 GPGPU-SIM을 사용한다. A. Bakhoda et. al.에 의해 개발된 GPGPU-SIM은 GPU와 같이 다수의 코어를 사용하는 매니코어 프로세서의 성능을 사이클 단위의 정확성으로 평가하는 시뮬레이터이다[27]. GPGPU-SIM은 GPU의 성능을 평가하는 시뮬레이터로의 신뢰성을 검증받고 있기 때문에 많은 연구자들에 의해 활용되었으며, 그 결과 또한 저명한 학회의 논문에 발표되었다[18][27].

표 2. GPU 기본 구성 변수

변수	값
Number of Shader Cores	30
Warp Size	32
SIMD Pipeline Width	32
Number of Threads/Core	1024
Shared Memory/Core	16KB
Constant Cache/Core	8KB, 2-way, 64byte lines, Read-only
Texture Cache/Core	8KB, 2-way, 64byte lines, Read-only
L1 Data Cache	32KB, 4-way, 64byte lines
Number of L1 Cache Bank	1
Clock (Core/Interconnection/DRAM)	325/650/800MHz
Number of Memory Controller	12
Number of Memory Chip per Controller	2
Memory Channel Bandwidth	4 bytes
DRAM Request Queue Size	32
GDDR3 Memory Timing	tCL=10, tRP =10, tRC=35, tRAS=25, tRCD=12, tRRD=8

하드웨어 구성 요소와 내부 연결망 구성 요소는 GPU의 성능을 결정하는 주요 요소이기 때문에, GPGPU-SIM은 벤치마크 프로그램을 하드웨어 구성 요소를 기

반으로 실행하는 시스템 시뮬레이터 부분과 GPU 내부의 각 코어들을 연결하는 네트워크 시뮬레이터 부분으로 구성되어 있다. 본 논문에서는 GPGPU-SIM의 하드웨어 구성요소를 NVIDIA사의 QuadroFX5800를 참고하여 모델링한다[28]. QuadroFX5800의 하드웨어 구성요소의 값은 [표 2]에서 보여주고 있다.

표 3. 내부 연결망 구성 변수

변수	값
Topology	Fly
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

GPGPU-SIM은 네트워크 시뮬레이터로 다양한 구조의 내부 연결망 네트워크 구조를 지원하고 다른 시뮬레이터와 협동이 가능한 Booksim[29]을 사용한다. 본 논문에서는 플라이(fly) 구조의 내부 연결망을 사용하며, 상세한 네트워크 변수 설정은 [표 3]에서 보이는 바와 같다.

표 4. 코어 개수와 동작 주파수 변수

연산 자원 구성 변수			
Number of Shader Cores	9	15	30
Warp Size	8	16	32
SIMD Pipeline Width	8	16	32
메모리 자원 구성 변수			
Number of L1 Data Cache Bank	1	2	4
Memory Channel Bandwidth	4bytes	8bytes	16bytes

워프 생성 기법에 따른 성능 변화를 다양한 GPU의 하드웨어 구성에서 측정하기 위하여, 하드웨어 자원들을 변화시키며 실험을 수행한다. 워프 생성 기법은 GPU 내부 자원 활용성(즉, 연산 병렬성)에 큰 영향을 미친다. 그러므로 본 논문에서는 하드웨어 구성 요소에 대한 변수를 설정하는데 있어서 [표 4]에서 보이는 바와 같이 GPU의 병렬성에 크게 영향을 미치는 연산 자원과 메모리 자원을 선택한다. 변수로 선택된 연산 자원의 종류로는 GPU가 동시에 처리할 수 있는 연산량에 크게 영향을 미치는 세이더 코어의 수와 워프의 크기이다. 특히, 워프 내의 스레드 수를 의미하는 워프의 크기는 세이더 코어의 SIMD 파이프라인의 폭을 결정하기 때문에 매우 중요한 요소이다. 그러므로 하드웨어 구성과 워프 생성 기법에 따른 성능 변화를 분석하는 본 논문에서 다양한 워프 크기는 반드시 고려되어야 한다. 뿐만 아니라, 본 논문에서는 동일한 용량을 가진 메모리에서 다양한 병렬성에 따른 워프 생성 기법의 효율성을 분석하고자, L1 데이터 캐쉬의 뱅크 수와 DRAM의 대역폭을 표 4에서 보이는 바와 같이 설정하고 실험을 수행한다. 다수의 워프들이 접근하기 때문에 워프 생성 기법에 의해 크게 영향을 받는 L1 데이터 캐쉬와 DRAM에 비해, 공유 메모리는 하나의 워프내의 스레드들이 공유하는 메모리 자원으로 워프 생성 기법에 의해 큰 영향을 받지 않기 때문에 메모리 자원 구성 변수로 선택하지 않는다. 이를 제외하고도, 세이더 코어의 공유 메모리의 크기와 워프 내의 최대 활성화 스레드의 수에 따라 GPU의 성능은 크게 변화할 것은 분명하다. 하지만 공유 메모리의 크기와 워프 내의 최대 활성화 스레드 수는 워프 생성 기법에는 직접적인 영향을 미치지 않기 때문에, 본 논문에서는 워프 생성 기법에 따른 성능 변화를 보다 상세히 보여주기 위해서 공유 메모리의 크기와 워프 내의 최대 활성화 스레드 수에 따른 실험은 수행하지 않는다.

2. 벤치마크 프로그램

GPGPU-SIM은 그래픽 관련 벤치마크 프로그램뿐만 아니라 일반 벤치마크 프로그램을 입력으로 받아 들여, 벤치마크 프로그램의 수행 결과와 더불어 프로세서 구조의 성능을 IPC와 메모리 접근 지연 시간 및 적중률 등 다양한 측면에서의 측정하여 출력한다[27]. 본 논문에서는 NVIDIA SDK[30] 프로그램들 중에서 분기 명령어를 많이 가지고 있는 응용 프로그램들을 선별하여 총 6개의 프로그램들(Breadth First Search, Fast Walsh Transform, Scan, ScanLargeArray,

MersenneTwister, Ray Tracing)을 이용한다. 선택된 벤치마크 프로그램에 대한 상세한 설명은 [표 5]에 기술한다[31].

표 5. 벤치마크 프로그램

벤츠마크 명	약자	설명
Breadth First Search	BFS	그래프 넓이 우선 평가 알고리즘 프로그램
Fast Walsh Transform	FWT	퓨리에(fourier) 변환을 빠르게 수행하는 알고리즘 프로그램
Scan	SCAN	고정된 크기의 병렬 프리픽스(prefix) 합 연산 능력을 평가하기 위한 프로그램
ScanLargeArray	SLA	임의의 크기의 병렬 프리픽스(prefix) 합 연산 능력을 평가하기 위한 프로그램
MersenneTwister	MT	임의 난수들을 발생시키는 알고리즘을 수행하는 프로그램
Ray Tracing	RAY	실사(real picture)에 근접한 렌더링 작업을 수행하는 프로그램

Ⅳ. 실험 결과

본 논문에서 우리는 GPU의 성능을 보다 다양한 측면에서 측정하고자 [표 5]에서 설명한 6가지의 벤치마크 프로그램들을 사용하여 실험을 수행한다. 하지만 실험 결과는 지면의 제한으로 인하여 평균값만으로 보여준다. 그림에서 NRCV와 PDOM은 단일 명령어 다중 데이터 직렬화 기법과 단일 명령어 다중 데이터 재수렴 방식의 post-dominator 기법을 나타내며, DWF는 단일 명령어 다중 데이터 동적 워프 생성 기법을 나타낸다.

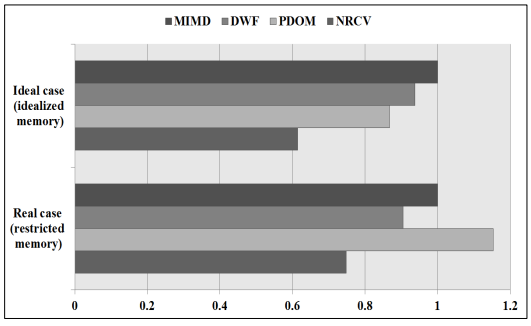

그림 2. 워프 생성 기법에 따른 GPU 성능 변화

[그림 2]는 선택된 벤치마크 프로그램들을 워프 생성 기법에 따라 수행한 경우의 성능을 MIMD를 기준으로 정규화한 IPC 값으로 보여주고 있다. 그림에서 Ideal case는 메모리로 인한 해저드가 발생되지 않는 경우의 성능을 보여주고 있으며, Real case는 QuadroFX5800을 대상[표 2]으로 모델링한 경우의 성능을 보여준다. MIMD는 다중 명령어 다중 데이터(Multiple Instruction, Multiple Data) 구조를 나타내는 것으로 연산 자원을 전부 활용하는 이상적인 워프 생성 기법의 경우를 나타낸다[32]. 그림에서 보이다시피, 이상적인 메모리 시스템을 가진 경우(Ideal case)에서는 높은 연산 자원 활용률을 보이는 워프 생성 기법 순서(DWF, PDOM, NRCV)대로 높은 성능을 나타낸다. 하지만 이상적인 경우(Ideal case)와는 달리 실제 GPU와 같이 제한된 메모리 시스템을 가진 경우(Real case)에서는 NRCV는 여전히 낮은 성능을 보여주는 반면, PDOM의 성능이 DWF의 성능보다 높게 나타난다. 그러므로 실제 GPU에서는 연산 자원을 많이 활용할 수 있는 워프 생성 기법이 반드시 높은 성능을 보인다고 장담할 수 없다. 분석 결과, DWF 기법은 워프를 매번 새롭게 생성하기 때문에 낮은 메모리 지역성을 가진 메모리 요구가 발생한다. 낮은 지역성의 메모리 요구는 많은 메모리 접근을 요구하여 심각한 메모리 시스템의 충돌을 야기하여 성능을 저하시킨다. 그러므로 워프 생성 기법을 통해 연산 자원 활용률을 높여 성능을 향상시키고자 하는 경우 하드웨어 자원과 워프 생성 기법사이의 상관관계는 반드시 고려되어야 한다. 이와 같은 이유로, 본 논문에서는 다양한 하드웨어 구성(연산 자원과 메모리 자원)과 워프 생성 기법에 따른 성능 변화를 상세하게 분석하고자 한다. 본 논문에서는 고정된 메모리 자원에서 다양한 연산 자원 구성과 3가지 워프 생성 기법에 따른 GPU의 성능 변화와, 고정된 연산 자원 구성에서 다양한 메모리 자원 구성과 워프 생성 기법에 따른 GPU의 성능 변화를 상세히 분석하고자 한다. 본 논문에서 우리는 성능에 큰 영향을 미치는 세이더코어의 수와 워프 크기를 연산 자원 구성 요소로 선택하고 메모리 대역폭과 L1 캐쉬 뱅크 수를 메모리 자원 구성 요소로 선택하

여 실험을 수행한다. 본 논문에서는 동일 주파수를 사용하는 프로세서의 성능을 평가하는 경우 널리 사용되는 사이클 당 명령어 수를 나타내는 IPC(Instructions per Cycle)를 사용하여 하드웨어 구성과 워프 생성 기법에 따른 성능 변화를 분석한다. [그림 3-8]에서 성능은 DWF의 IPC를 기준으로 정규화하여 표현한다. 그리고 [그림 3-8]에서 우측의 그래프는 [표 2]의 값으로 구성된 기본 구조(baseline)의 성능을 나타내는 것으로 [그림 2]의 실제 경우(Real case)의 값과 동일하다.

1. 연산 자원 구성과 워프 생성 기법에 따른 성능 변화

[그림 3-5]에서 SC-숫자와 WS-숫자는 세이더코어의 숫자와 워프의 크기를 각각 나타낸다. 예를 들어, SC-15와 WS-16은 15개의 세이더코어를 가진 GPU 구조와 GPU에서 동시에 수행되는 스레드의 개수가 16개임을 의미한다.

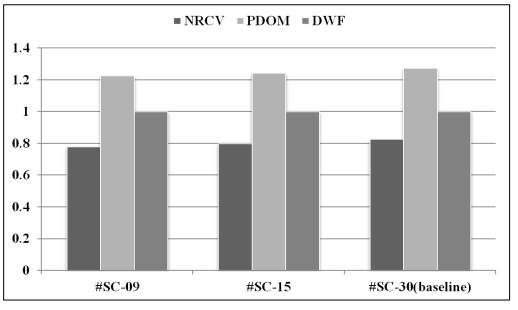

그림 3. 코어 수와 워프 생성 기법에 따른 성능

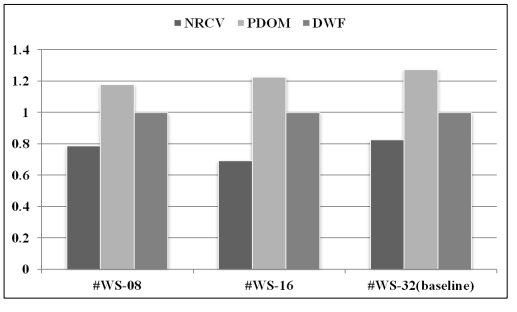

그림 4. 워프 크기와 워프 생성 기법에 따른 성능

[그림 3]은 코어 수와 워프 생성 기법에 따른 GPU의 성능 변화를 보여준다. [그림 3]에서 PDOM과 DWF의 성능 차는 SC-30(baseline)에 비해 SC-15는 3.1% 감소(하고, SC-09는 4.9% 감소한다. 그리고 NRCV와 DWF의 성능 차는 SC-32(baseline)에 비해 SC-15과 SC-09은 각각 2.7%, 4.8% 증가한다.

[그림 4]는 다양한 워프 크기와 워프 생성 기법에 따른 성능 변화를 상세히 보여준다. 그림에서 보이듯이, WS-32(baseline)에 비해 WS-16과 WS-08의 성능 차는 PDOM과 DWF 사이는 각각 4.7%, 9.5% 감소하고, NVRC과 DWF 사이는 13.4%, 4.0% 증가한다.

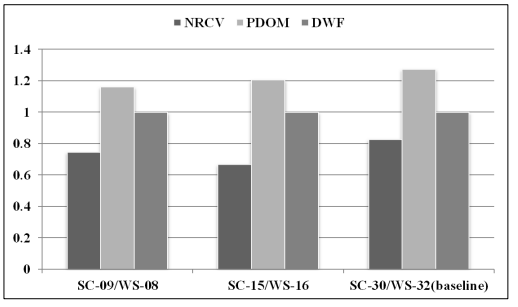

그림 5. 코어 수/워프 크기와 워프 생성 기법에 따른 성능

[그림 5]는 코어 수와 워프 크기 모두를 고려한 연산 자원 구성과 워프 생성 기법에 따른 성능 변화를 보여준다. [그림 5]에서 보이듯이, PDOM과 DWF의 성능 차는 SC-30/WS-32(baseline)에 비해 SC-15/WS-16는 6.9%, SC-09/WS-08는 11.2% 감소한다. 그리고 NRCV와 DWF의 성능 차는 SC-30/WS-32(baseline)에 비해 SC-15/WS-16는 16.0% 증가하고, SC-09/WS-08는 8.1% 증가한다.

연산 자원이 감소함에 따라, 동시에 요청되는 메모리 요구가 줄어들어 메모리 충돌을 감소시킨다. 결과적으로 병목 현상으로 인한 문제점이 완화됨으로써 높은 연산 활용률을 보이는 DWF의 성능이 NRCV/PDOM에 비해 상대적으로 많이 향상된다. 하지만 연산자원의 감소는 병렬성에 부정적인 영향을 미치기 때문에 연산 처리량을 크게 감소시킨다.

2. 메모리 자원 구성과 워프 생성 기법에 따른 성능 변화

[그림 6-8]에서 MB-숫자와 CB-숫자는 세이더코어의 숫자와 워프의 크기를 각각 나타낸다. 예를 들어, MB-16와 CB-02은 메모리 대역폭이 16바이트인 GPU 구조와 L1 캐쉬의 뱅크가 2개인 GPU 구조를 의미한다.

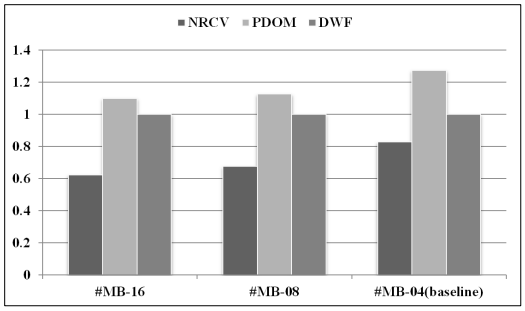

그림 6. 메모리 대역폭과 워프 생성 기법에 따른 성능

[그림 6]은 메모리 대역폭과 워프 생성 기법에 따른 GPU의 성능 변화를 보여준다. [그림 6]에서 PDOM과 DWF의 성능 차는 MB-04(baseline)에 비해 MB-08은 14.8% 감소하고, MB-16는 17.5% 감소한다. 그리고 NRCV와 DWF의 성능 차는 MB-04(baseline)에 비해 MB-08과 MB-16은 각각 15.1%, 20.4% 증가한다.

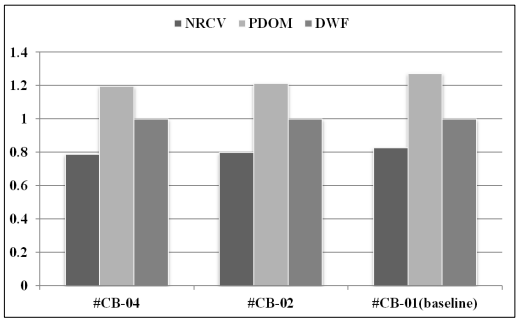

그림 7. L1 캐쉬 뱅크 수와 워프 생성 기법에 따른 성능

[그림 7]는 L1 캐쉬의 뱅크 수와 워프 생성 기법에 따른 성능 변화를 상세히 보여준다. 그림에서 보이듯이, PDOM과 DWF의 성능 차는 CB-01(baseline)에 비해 CB-02와 CB-04는 각각 6.1%, 7.7% 감소한다. NVRC과 DWF의 성능 차는 CB-01(baseline)에 비해 CB-02는 2.8% 감소하고 CB-04는 3.8% 증가한다.

[그림 8]은 메모리 대역폭과 L1 캐쉬 뱅크 수 모두를 고려한 메모리 자원 구성과 워프 생성 기법에 따른 성능 변화를 보여준다. [그림 8]에서 보이듯이, MB-04/CB-01(baseline)에 비해 MB-08/CB-02와 MB-16/CB-04에서 PDOM과 DWF의 성능 차는 각각 18.7%와 21.0% 감소하고, NRCV와 DWF의 성능 차는 각각 17.0%, 22.3% 증가한다.

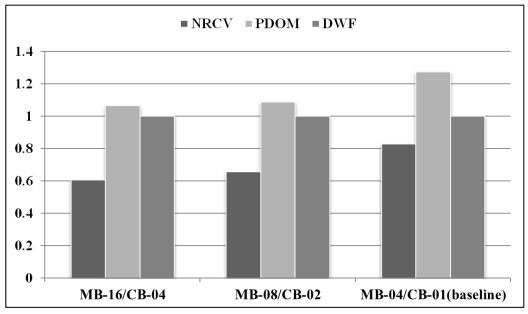

그림 8. 메모리 자원 구성과 워프 생성 기법에 따른 성능

메모리 자원과 워프 생성 기법에 따른 실험 결과는 메모리 자원이 증가할수록 메모리 병목 현상으로 인한 오버헤드를 감소시켜 높은 연산 활용률을 보이는 DWF의 성능이 향상됨을 보여준다. 하지만 메모리 자원을 증가하는 것은 비용을 많이 요구하기 때문에 GPU 설계 비용을 크게 증가시킨다.

V. 결 론

GPU를 활용하여 범용 응용프로그램을 수행하는 경우, GPU의 연산 자원 활용률은 분기 명령어로 인해 저하된다. 연산 자원 활용률 저하 문제를 해결하기 위하여, 최신 GPU들은 분기 명령어를 처리할 수 있는 워프 생성 기법들을 사용한다. 본 논문에서는 대표적인 워프 생성 기법들이 성능에 미치는 영향을 분석하였다. 기존의 예상과는 달리, 높은 연산 자원 활용률을 보이는 워

프 생성 기법의 성능이 상대적으로 낮은 연산 자원 활용률을 보이는 워프 생성 기법의 성능보다 낮게 나타났다. 지역성 원리가 배제된 무작위 메모리 접근으로 인한 심각한 메모리 병목 현상이 효율적인 연산 자원 활용에도 불구하고 심각한 GPU 성능 저하를 유발한 것으로 분석된다. 본 논문에서 우리는 하드웨어 구성에 따라 워프 생성 기법이 GPU의 성능에 미치는 영향에 대해 분석하고자 연산 자원과 메모리 자원을 다양하게 구성하여 성능을 측정하였다. 본 논문에서는 상대적인 효율성을 분석하는 것이 주된 목적이기 때문에, 성능을 평가하는 지표로 상대적인 증감을 명확하게 보여주는 정규화 방식을 사용하였다. 정규화 방식은 편차가 큰 결과 값들을 보다 쉽고 명확하게 보여주기 때문에 널리 사용되는 성능 지표 표현 방식이다. 우리의 실험 결과에 따르면, 연산 자원을 많이 사용하는 워프 생성 기법에서 발생된 문제를 해결하기 위해서는 메모리와 같은 추가적인 자원이 필요하다. 결과적으로 적절한 하드웨어 지원이 없는 경우, 연산 자원을 많이 사용하는 워프 생성기법이 반드시 높은 성능을 발휘한다고 볼 수 없다. 불행히도 하드웨어를 추가하는 것은 비용이 매우 많이 발생한다. 본 논문의 연구결과는 높은 연산 자원 활용률을 보이는 워프 생성 기법에서 발생되는 문제점을 정량적으로 보여주고 있기 때문에, 향후에 추가적인 비용을 최소화하면서 성능을 향상시킬 수 있는 분기 처리 기법에 대한 연구를 진행하는 데 있어서 가이드라인을 제시할 수 있을 것으로 기대된다.

참 고 문 헌

[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] N. P. Jouppi and D. W. Wall, “Available instruction-level parallelism for superscalar and superpipelined machines,” In Proceedings of 3th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.272-282, 1989.

[3] D. M. Tullsen, S. J. Eggers, and H. M. Levy, “Simultaneous multithreading: maximizing on-chip parallelism,” In Proceedings of 22th International Symposium on Computer Architecture, pp.392-403, 1995.

[4] Y. H. Jang, C. Park, J. H. Park, N. Kim, and K. H. Yoo, “Parallel Processing for Integral Imaging Pickup using Multiple Threads,” International Journal of Korea Contents, Vol.5, No.4, pp.30-34, 2009.

[5] I. 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 (SIGGRAPH), pp.777-786, 2004.

[6] E. Lindholm, M. J. Kligard, and H. P. Moreton, “A user-programmable vertex engine,” In Proceedings of 28th Annual Conference on Computer Graphics (SIGGRAPH), pp.149-158, 2001.

[7] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, “A Survey of General-Purpose Computation on Graphics Hardware,” Eurographics 2005, State of the Art Reports, pp.21-51, 2005.

[8] http://developer.nvidia.com/object/cuda_3_1_downloads.html

[9] http://www.khronos.org/opencl/

[10] J. Helin, “Performance analysis of the CM-2, a massively parallel SIMD computer,” In Proceedings of 6th International Conference on Supercomputing, pp.45-52, 1992.

[11] A. Levinthal and T. Porter, “Chap - a SIMD

graphics processor," In Proceedings of 11th Annual Conference on Computer Graphics (SIGGRAPH), pp.77-82, 1984.

[12] S. Che, J. Meng, J. Sheaffer, and K. Skadron, "A performance study of general purpose applications on graphics processors using CUDA," Journal of Parallel and Distributed Computing, Vol.68, No.10, pp.1370-1380, 2008.

[13] R. A. Lorie and H. R. Strong, "Method for conditional branch execution in SIMD vector processors," US Patent 4435758, Vol.6, 1984(3).

[14] S. Moy and E. Lindholm, "Method and system for programmable pipelined graphics processing with branching instructions," US Patent 6947047, Vol.20, 2005(9).

[15] E. Rotenberg, Q. Jacobson, and J. E. Smith, "A study of control independence in superscalar processors," In Proceedings of 5th International Symposium on High-Performance Computer Architecture, pp.115-124, 1999.

[16] B. W. Coon and J. E. Lindholm, "System and method for managing divergent threads in a SIMD architecture," US Patent 7353369, Vol.1, 2008(4).

[17] E. Rotenberg, Q. Jacobson, and J. Smith, "A study of control independence in superscalar processors," In Proceedings of 5th International Symposium on High-Performance Computer Architecture, pp.115-124, 1999.

[18] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," In Proceedings of 40th Microarchitecture, pp.407-420, 2007.

[19] H. J. Choi and C. H. Kim, "Performance Evaluation of the GPU Architecture Executing Parallel Applications," Journal of the Korea Contents Association, Vol.12, No.5, pp.10-21, 2012.

[20] H. J. Choi, H. G. Jeon, and C. H. Kim, "Quantitative Anaysis of the Negative Factors on the GPU Performance," Journal of KIISE : Computing Practices and Letters, Vol.18, No.4, pp.282-287, 2012.

[21] H. J. Choi, S. G. Kang, J. M. Kim, and C. H. Kim, "Analysis of the CPU/GPU Temperature and Energy Efficiency depending on Executed Applications," Journal of The Korea Society of Computer and Information, Vol.17, No.5, pp.9-20, 2012.

[22] http://www.amd.com/stream

[23] https://developer.nvidia.com/cg-toolkit

[24] http://msdn2.microsoft.com/en-us/library/bb509638.aspx

[25] http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

[26] http://www.simplescalar.com

[27] 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.

[28] http://www.nvidia.com/object/product_quadro_fx_5800_us.html

[29] http://nocs.stanford.edu/booksim.html

[30] http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html

[31] http://www.nvidia.com/content/cudazone/

[32] M. J. Flynn, "Very high-speed computing systems," Proceedings of the IEEE, Vol.54, No.12, pp. 1901-1909, 1966.

저 자 소 개

최 홍 준(Hong Jun Choi) 정회원

- 2009년 : 전남대학교 전자컴퓨터공학부 공학사
- 2011년 : 전남대학교 전자컴퓨터공학과 공학석사
- 2011년 ~ 현재 : 전남대학교 전자컴퓨터공학과 박사과정

<관심분야> : 컴퓨팅 시스템, 병렬처리, 컴퓨터구조

김 철 홍(Cheol Hong Kim) 정회원

- 1998년 : 서울대학교 컴퓨터공학사
- 2000년 : 서울대학교 컴퓨터공학부 공학석사
- 2006년 : 서울대학교 전기컴퓨터공학부 공학박사
- 2005년 ~ 2007년 : 삼성전자 반도체총괄 SYS.LSI 사업부 책임 연구원
- 2007년 ~ 현재 : 전남대학교 전자컴퓨터공학부 교수

<관심분야> : 임베디드시스템, 컴퓨터구조, SoC 설계, 저전력 설계