The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 19, No. 1, pp.111-116, Feb. 28, 2019. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2019.19.1.111
JIIBC 2019-1-15

GPU 작업 배치의 효율화를 위한 자원 이용률 상세 분석

Analyzing Fine-Grained Resource Utilization for Efficient GPU Workload Allocation

박윤주*, 신동희*, 조경운**, 반효경***

Yunjoo Park*, Donghee Shin*, Kyungwoon Cho, Hyokyung Bahn*****

요 약 최근 GPU가 그래픽 처리뿐 아니라 다양한 분야의 병렬 처리로 그 영역을 넓혀가고 있다. 그러나, 현재 GPU는 워크로드의 다양성을 반영하기보다 간결한 제어 구조를 통한 개별 워크로드의 병렬성 극대화에 초점을 맞추고 있다. 본 논문은 워크로드 특성을 반영한 GPU 작업 배치를 위해 GPU에서 수행되는 워크로드의 자원 사용 특성을 컴퓨팅 바운드형, 메모리 바운드형, 실행종속 지연형으로 분류한 후, 각 분류에서 병목점이 되는 세부 자원을 규명한다. 예를 들어 컴퓨팅 바운드형의 경우 단정밀도 연산장치, 배정밀도 연산장치, 특수함수 연산장치 등 병목 자원이 무엇인지 분석한다. 본 논문의 분석 결과는 동일한 컴퓨팅 바운드형 워크로드라도 병목이 되는 세부 자원이 다를 경우 함께 배치하는 것이 성능 충돌을 일으키지 않는다는 점을 규명하여 GPU 작업배치의 효율화에 기여할 것으로 기대된다.

주제어 : GPU, 자원 이용률, 병렬 워크로드, 작업 배치

Abstract Recently, GPU expands application domains from graphic processing to various kinds of parallel workloads. However, current GPU systems focus on the maximization of each workload's parallelism through simplified control rather than considering various workload characteristics. This paper classifies the resource usage characteristics of GPU workloads into computing-bound, memory-bound, and dependency-latency-bound, and quantifies the fine-grained bottleneck for efficient workload allocation. For example, we identify the exact bottleneck resources such as single function unit, double function unit, or special function unit even for the same computing-bound workloads. Our analysis implies that workloads can be allocated together if fine-grained bottleneck resources are different even for the same computing-bound workloads, which can eventually contribute to efficient workload allocation in GPU.

Key Words : GPU, resource utilization, parallel workload, workload allocation

Ⅰ. 서 론

4차산업혁명시대의 도래로 딥러닝, 블록체인, 유전자 분석 등 다양한 분야에서 GPU가 병렬 연산장치로 활용되고 있다[1,2]. 그러나, GPU 관리는 워크로드의 다양성을 반영하기보다 간결한 제어 구조를 통한 개별 워크로드의 병렬성 극대화에 초점을 맞추어왔다. 최근 다양한 분야의 워크로드가 GPU에서 동시에 수행되는 것이 지원되면서 개별 워크로드의 병렬 수행보다 멀티태스킹으로 인한 효율성을 높이는 것이 GPU 관리의 중요한 이슈로 부각

*준회원, 이화여자대학교 컴퓨터공학과
**비회원, 이화여자대학교 임베디드소프트웨어연구센터
***정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일 : 2018년 11월 29일, 수정완료일 : 2019년 1월 8일
게재확정일 : 2019년 2월 8일

Received: 29 November, 2018 / Revised: 8 January, 2019 / Accepted: 8 February, 2019
***Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

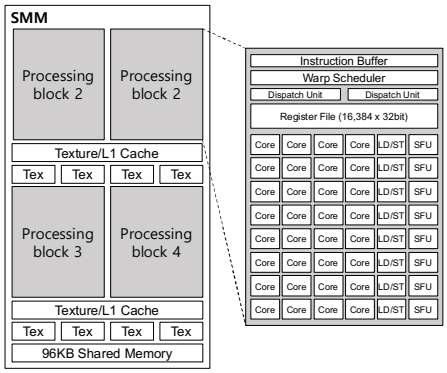


그림 1. Maxwell 아키텍처 기반의 GPU 내부 구조
Fig. 1. Internal architecture of Maxwell-based GPU

되고 있다[3,4].

본 논문은 멀티태스킹을 통한 GPU의 자원이용률 극대화를 위해 GPU 워크로드의 자원 사용 특성을 분석한다. 이를 위해 GPU 워크로드를 컴퓨팅 바운드형, 메모리 바운드형, 실행종속 지연형으로 분류한 후, 각 분류에서 병목점이 되는 세부 자원을 규명한다. 특히, 본 논문은 동일한 분류에 속한 워크로드인 경우에도 병목점에 해당하는 세부 자원이 다를 수 있음을 보여준다. 예를 들어 컴퓨팅 바운드형의 경우 단정밀도 연산장치, 배정밀도 연산장치, 특수함수 연산장치 등 구체적인 병목 자원이 워크로드에 따라 다름을 확인한다.

본 논문의 분석 결과는 동일한 유형의 워크로드라도 병목이 되는 세부 자원이 다를 경우 함께 배치하는 것이 성능 충돌을 일으키지 않는다는 점을 규명하여 GPU 작업배치 및 멀티태스킹의 효율화에 기여할 수 있을 것으로 기대된다.

Ⅱ. GPU 아키텍처

1. GPU의 내부 구조

현대의 매니코어 GPU는 수백에서 수천 개의 코어를 이용한 병렬 처리로 빠른 연산을 수행한다. GPU에서 수행되는 작업은 SIMT(single instruction multiple thread) 모델에 기반하여 여러 코어에서 동일한 종류의 연산이 병렬적으로 수행된다. 이 때, 호스트에서 런칭되는 GPU 프로그램을 커널이라고 부른다. SIMT 프로그래밍 모델에서는 스레드가 실행의 기본 단위이며, 스레드들은 계층적인 스레드 블록과 그리드로 그룹화된다. CPU에서 프로그램이 실행되면, 커널 코드가 GPU로 전달되고 그리드 단위로 생성 및 처리된다.

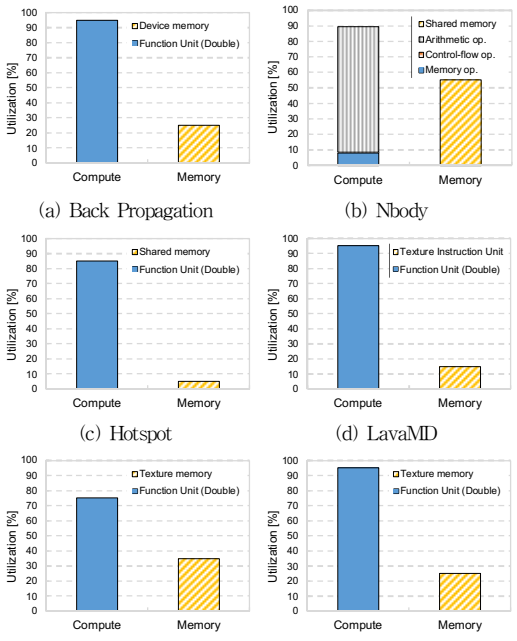
커널은 여러 스레드 블록들로 구성된 그리드이고, 스레드 블록은 다수의 스레드로 구성된다. GPU 하드웨어는 다수의 스트리밍 멀티프로세서(SM: streaming multiprocessor)로 구성되며, 각 SM은 프로세싱 블록들로 세분화된다. 프로세싱 블록은 연산을 수행하는 스트리밍 프로세서 또는 코어라고 불리는 유닛들로 구성된다. 스레드 블록 스케줄러에 의해 각 스레드 블록들은 SM에 할당되며, 자원의 여유가 있으면 8개까지의 스레드 블록을 동시에 실행할 수 있다. 블록 안의 스레드끼리는 의존적이지만, 블록 간 스레드는 독립적이므로 병렬적인 처리가 가능하다.

그림 1은 Maxwell 아키텍처 기반의 GPU 내부구조를 보여주고 있다. Maxwell 아키텍처에서는 SM을 SMM이라고 칭하며, 하나의 SM이 4개의 프로세싱 블록으로 구성된다. 각 프로세싱 블록은 32개의 코어와 특수함수 연산장치(special function unit, SFU), 로드 스토어 유닛과 각종 메모리 자원들로 구성된다. 각 코어에는 32비트 단정밀도 연산장치, 64비트 배정밀도 연산장치, 제어흐름장치 등이 존재한다. SM 내의 메모리 자원들로는 레지스터 파일, 공유 메모리, 통합 캐시(unified cache) 등이 있으며, 통합 캐시는 텍스쳐 캐시와 L1 캐시로 구성된다. 텍스쳐 캐시는 데이터 값을 시각화하는 그래픽 데이터용 저장공간이며, L1 캐시는 SM 외부의 전역 메모리로부터 읽어온 데이터를 저장하는 역할을 한다. 또한, 각 프로세싱 블록 내에는 명령어 처리를 위한 디스패치 유닛과 워프 스케줄러가 존재한다. 디스패치 유닛은 클록마다 명령어 하나를 코어에 할당하는 역할을 하며, 워프 스케줄러는 32개의 스레드로 구성되는 워프 단위로 스케줄링하는 역할을 한다.

2. CUDA 플랫폼 및 메모리 구조

CUDA는 GPU에서 수행하는 병렬 알고리즘을 C 언어를 비롯한 산업 표준 언어와 결합하여 작성할 수 있도록 NVIDIA사가 설계한 병렬플랫폼 기술이다.

CUDA의 메모리는 칩내부 메모리인 온칩 메모리와 칩외부 메모리인 디바이스 메모리로 구성되며, 온칩 메모리는 스레드 블록 간 통신을 위한 공유메모리와 각 스레드가 사용하는 레지스터로 구성된다. 한편, 레지스터



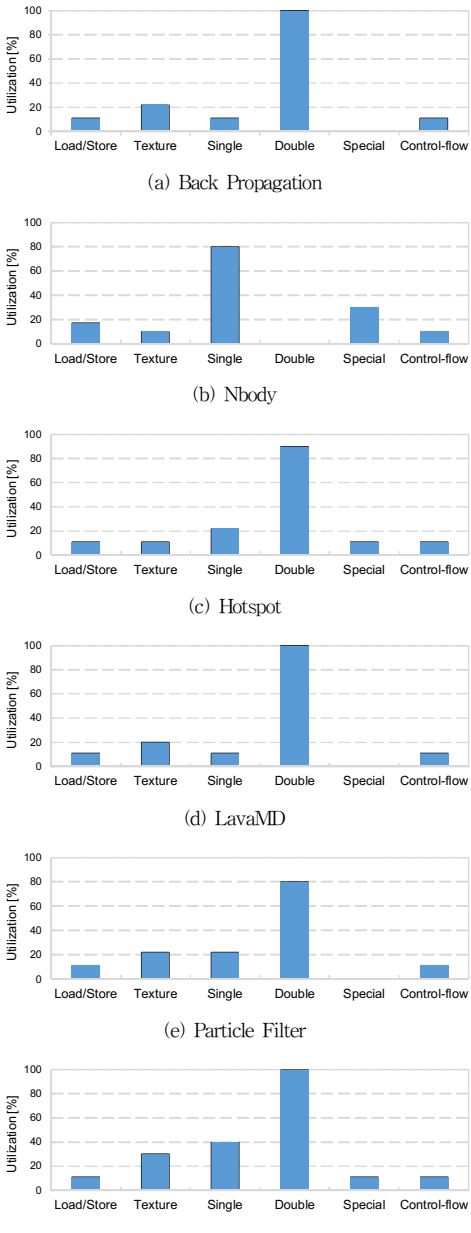
(a) Back Propagation (b) Nbody
(c) Hotspot (d) LavaMD
(e) Particle Filter (f) Srad

그림 2. 컴퓨팅 바운드형 워크로드의 자원 이용률
Fig. 2. Resource utilization of computing-bound workloads

용량이 한정적이므로 온칩 메모리가 부족한 경우 디바이스 메모리인 지역 메모리가 활용된다. 이 외에도 디바이스 메모리에는 모든 스레드가 접근할 수 있는 전역 메모리와 모든 스레드가 공유하지만 읽기전용 공간인 컨스턴트 메모리, 텍스쳐 메모리 등으로 구성된다. 최적의 성능을 위해서는 CUDA의 메모리 구조 및 특성을 충분히 고려하여 병렬 프로그램을 작성해야 한다.

Ⅲ. GPU 워크로드 특성의 상세 분석

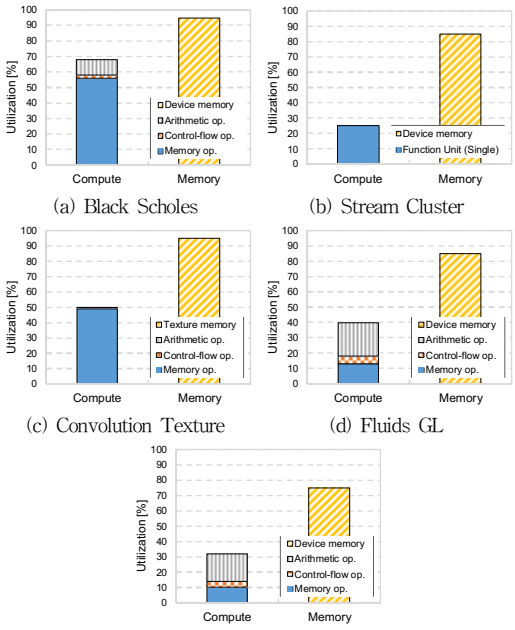
GPU에서 수행되는 병렬작업의 부하를 높이면 보통 컴퓨팅 자원 또는 메모리 자원 중 하나가 성능 상의 병목점에 이르게 된다. 본 논문에서는 자원의 이용률에 근거하여 GPU 워크로드를 컴퓨팅 바운드형과 메모리 바운드형으로 분류하였다. 한편, 워크로드에 따라서는 병렬 작업의 부하를 최대한 높여도 컴퓨팅 자원이나 메모리 자원이 병목점이 되지 않는 경우가 존재한다. 이러한 경우는 워크로드 자체의 실행 종속관계에 의해 GPU 자원 활용이 제한받는 유형으로 본 논문에서는 이러한 유형을 실행종속 지연형으로 분류하였다.



(a) Back Propagation
(b) Nbody
(c) Hotspot
(d) LavaMD
(e) Particle Filter
(f) Srad

그림 3. 연산 장치 종류에 따른 이용률
Fig. 3. Utilization of each computing resource

본 논문에서는 NVIDIA의 프로파일러 툴을 사용하여 GPU 성능평가에 활용되는 대표적인 벤치마크인 NVIDIA SDK[5]와 Rodinia[6] 실행시의 자원 이용률을 측정하였다. 시스템의 구성은 CPU Intel(R) i7-3770, 메모리 8GB DDR3, GPU NVIDIA Geforce GTX750로 구



(a) Black Scholes (b) Stream Cluster

(c) Convolution Texture (d) Fluids GL

(e) HS Optical Flow

그림 4. 메모리 바운드형 워크로드의 자원 이용률

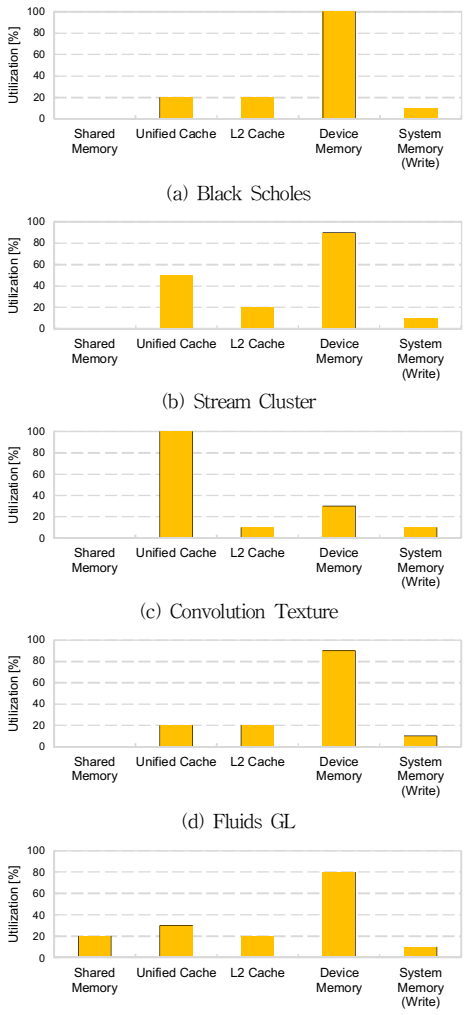
Fig. 4. Resource utilization of memory-bound workloads

성되며, 운영체제는 Ubuntu 16.04 LTS를 사용하였다.

1. 컴퓨팅 바운드형

그림 2는 컴퓨팅 바운드형 워크로드들의 자원별 이용률을 보여준다. 자원의 이용률은 매 클록당 1개의 명령어가 실행되었을 때를 100%로 하였을 때의 상대적인 수치로 표시하였다. 그림에서 보는 것처럼 컴퓨팅 바운드형 워크로드들은 컴퓨팅 자원이 병목점에 이른 반면 메모리 자원은 이용률이 낮은 것을 확인할 수 있다.

한편, GPU 내의 컴퓨팅 자원은 단정밀도 연산장치, 배정밀도 연산장치, 제어흐름 장치, 로드 스토어 유닛, 특수함수 연산장치 등으로 분류할 수 있다. 그림 3은 이러한 상세 자원별 이용률을 표시한 결과이다. 그림에서 보는 것처럼 동일한 컴퓨팅 바운드형이라 하더라도, 병목을 일으키는 상세 자원이 상이한 것을 확인할 수 있다. 6개의 워크로드 중 5개는 배정밀도 연산장치가 80% 이상의 높은 이용률을 보였으며, 이에 비해 단정밀도 연산장치와 SFU는 이용률이 매우 낮은 것을 확인할 수 있다. 이를 통해 이들 워크로드들은 64비트 배정밀도 데이터의 산술연산이 대부분임을 알 수 있다. 그에 반해 Nbody는 확연히 다른 자원 사용 양상을 보인다. 배정밀도 연산장



(a) Black Scholes

(b) Stream Cluster

(c) Convolution Texture

(d) Fluids GL

(e) HS Optical Flow

그림 5. 메모리 종류에 따른 이용률

Fig. 5. Utilization of each memory resource

치의 이용률은 0%인 반면, 단정밀도 연산장치의 이용률은 80%, SFU의 이용률은 30%임을 확인할 수 있다. 따라서, Nbody는 대부분 32비트 단정밀도 연산으로 이루어졌음을 알 수 있다.

2. 메모리 바운드형

메모리 바운드형 워크로드는 GPU 내의 메모리가 커널의 요청 속도로 데이터를 제공하지 못하여 성능 상의 병목점이 되는 경우를 뜻한다. 이는 궁극적으로 컴퓨팅 자원의 이용률을 낮아지게 하는 문제를 발생시킨다. 그림 4는 메모리 바운드형 워크로드들의 컴퓨팅 자원과 메

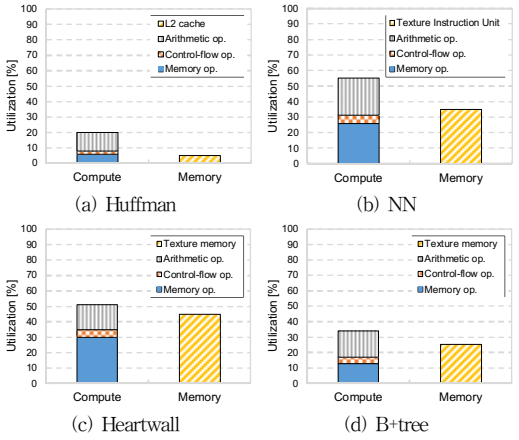


그림 6. 실행종속 지연형 워크로드의 자원 이용률
Fig. 6. Resource utilization of dependency-latency-bound workloads

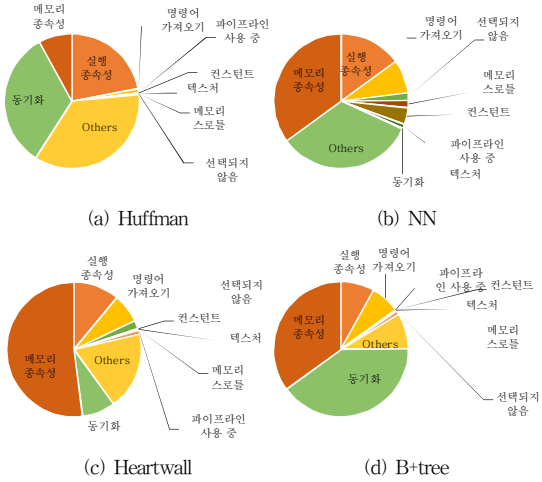


그림 7. 실행종속 지연형 워크로드의 스톨 원인
Fig. 7. Stall reason of dependency-latency-bound workloads

모리 자원 이용률을 보여주고 있으며, 그림 5는 메모리 상세 자원별 이용률을 표시하여 구체적으로 어떤 메모리 자원이 병목을 일으켰는지를 보여준다. 그림에서 보는 것처럼 Convolution Texture를 제외한 모든 워크로드에서 디바이스 메모리의 이용률이 가장 높게 나타났다. 이에 비해 Convolution Texture는 통합 캐시의 이용률이 높은 것을 확인할 수 있다. 이는 Convolution Texture가 텍스쳐 메모리에 대한 읽기 접근이 높아 발생한 현상으로 분석된다. 본 절의 분석을 통해 동일한 메모리 바운드형의 워크로드라 하더라도 GPU 디바이스가 제공하는 서로 다른 종류의 메모리 자원이 병목점이 될 수 있음을 확인할 수 있다.

3. 실행종속 지연형

실행종속 지연형 워크로드란 병렬 작업을 GPU 상에 충분히 배치하여 자원의 이용을 최대한 유도했음에도 컴퓨팅 자원과 메모리 자원의 이용률이 낮은 경우를 의미한다. 이는 워크로드 실행시 종속관계에 의한 멈춤현상(stall) 발생으로 자원이 충분히 활용되지 못함을 뜻한다. 워프의 실행을 막는 이러한 지연의 원인은 다음과 같은 것들이 있다.

- 명령어 인출 지연 - 다음 어셈블리 명령어가 아직 준비되지 않은 경우
- 파이프라인 지연 - 명령어 실행에 필요한 컴퓨팅 리소스를 사용할 수 없는 경우
- 동기화 지연 - 워프가 스레드 동기화를 위해 대기 중인 경우
- 컨스턴트 미스 - 컨스턴스 캐시 미스에 의해 데이터가 아직 로드되지 않은 경우
- 텍스쳐 오버헤드 - 텍스쳐 메모리가 모두 사용 중이거나 너무 많은 요청이 있는 경우
- 메모리 종속 지연 - 필요한 데이터가 아직 사용할 수 없거나 모두 사용중이서 로드/스토어가 불가능하거나 동종 자료형에 대한 요청이 너무 많은 경우
- 실행 종속 지연 - 명령어 실행에 필요한 인풋이 준비되지 않은 경우
- 메모리 스로틀 - 대기 중인 많은 메모리 연산으로 프로그램이 진행되지 않는 경우
- 미선택 지연 - 워프가 실행 준비를 마쳤으나 다른 워프가 이슈된 경우

이러한 지연 발생은 GPU 자원의 효율적인 관리만으로는 해결되지 않으며 병렬 워크로드 작성 시점부터 자원 효율성을 높이는 방향으로 구성할 경우 어느 정도의 개선을 기대할 수 있다.

Ⅳ. 결론

본 논문에서는 GPU에서 수행되는 워크로드를 컴퓨팅

바운드형, 메모리 바운드형, 실행종속 지연형으로 분류하였다. 총 15종의 대표적인 병렬 워크로드에 대한 분석 결과 컴퓨팅 바운드형 6종, 메모리 바운드형 5종, 실행종속 지연형 4종으로 확인되었으며, 동일한 분류에 속한 경우라도 병목 현상을 일으키는 세부 자원이 상이할 수 있음을 발견하였다. 본 논문의 분석 결과는 다양한 워크로드를 GPU 상에 적절히 배치하여 유휴 자원을 최소화하고 시스템 전체의 성능을 극대화하는 관리 기법에 활용될 수 있을 것으로 기대된다. 특히, 동종 분류에 속한 워크로드끼리도 세부적인 병목자원이 다른 경우 공동 배치하는 것이 가능할 것으로 기대된다.

References

[1] M. Abadi et al., "Tensorflow: a system for large-scale machine learning," Proc. OSDI Conf., pp. 265-283, 2016.

[2] D. Shin and H. Bahn, "Management technique of buffer cache for rendering systems," the Journal of the Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 18, No. 5, pp. 155-160, 2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.5.155

[3] A. Jog, et al., "Anatomy of gpu memory system for multi-application execution," Proc. ACM Symp. Memory Systems, pp 223-234, 2015.

[4] J. Choi, "Parallel-Addition Convolution Algorithm in Grayscale Image," The Journal of KIIECT, Vol. 10, No. 4, pp. 288-294, 2017.
DOI: https://doi.org/10.17661/jkiiect.2017.10.4.288

[5] GPU Computing SDK, http://developer.nvidia.com/gpu-computing-sdk.

[6] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," Proc. IEEE Symp. Workload Characterization, pp. 44 - 54, 2009.

저자 소개

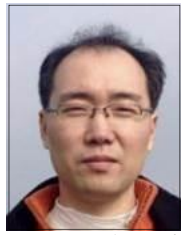
박 윤 주(준회원)

- 2015년 2월: 이화여자대학교 컴퓨터공학과 학사
- 2015년 3월~: 이화여자대학교 컴퓨터공학과 통합과정

신 동 희(준회원)

- 2016년 8월: 이화여자대학교 교육공학과 학사
- 2017년 3월~: 이화여자대학교 컴퓨터공학과 석사과정

조 경 운(비회원)



- 1995년 2월: 서울대학교 계산통계학과 학사
- 1997년 2월: 서울대학교 전산과학과 석사
- 2012년 2월: 서울대학교 컴퓨터공학부 박사
- 2000년~2016년: ㈜클루닉스 연구소장
- 2016년 4월 ~: 이화여자대학교 임베디드소프트웨어연구센터 수석연구원/연구교수

반 효 경(정회원)

- 1997년 2월: 서울대학교 계산통계학과 학사
- 1999년 2월: 서울대학교 전산과학과 석사
- 2002년 2월: 서울대학교 컴퓨터공학부 박사
- 2002년 9월~: 이화여자대학교 컴퓨터공학과 교수.

※ This work was supported by the ICT R&D program of MSIP/IITP (2018-0-00549, Extremely Scalable Order-preserving Operating System for Manycore and Non-volatile Memory).