The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 4, pp.87-92, Aug. 31, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.4.87
JIIBC 2020-4-12

SimTBS: GPGPU 스레드블록 스케줄링 시뮬레이터

SimTBS: Simulator For GPGPU Thread Block Scheduling

조경운*, 반효경**

Kyung-Woon Cho*, Hyokyung Bahn**

요 약 GPGPU(General-Purpose GPU)는 수만 단위의 스레드들을 병렬적으로 수행하여 성능을 최대화시킬 수 있지만, 실질적으로는 스레드들을 그룹화하여 스레드블록(Thread Block) 단위로 작업을 정의하고 GPGPU 하드웨어 자원의 할당 단위로 활용한다. 이러한 역할을 담당하는 스레드블록 스케줄러는 GPGPU내에 하드웨어적으로 구현되어 있으며, 스레드블록들을 하드웨어 자원들에게 라운드로빈 방식으로 할당한다. 그런데, 라운드로빈 정책은 단순 순차 할당 방식으로서 GPGPU 하드웨어 자원의 활용도에 최적화되어 있지 않다. 본 논문에서는 다양한 스레드블록 스케줄링 방식의 성능을 정량적으로 분석할 수 있는 스레드블록 스케줄러 모델을 제안하고, 구현된 시뮬레이터의 성능 결과를 통해 기존 GPGPU의 스레드블록 스케줄링 방식이 작업 부하가 높은 경우에는 적합하지 않음을 보이고자 한다.

Abstract Although GPGPU (General-Purpose GPU) can maximize performance by parallelizing a task with tens of thousands of threads, those threads are internally grouped into a thread block, which is a base unit for processing and resource allocation. A thread block scheduler is a specialized hardware gadget whose role is to allocate thread blocks to GPGPU processing hardware in a round-robin manner. However, round-robin is a sequential allocation policy and is not optimized for GPGPU resource utilization. In this paper, we propose a thread block scheduler model which can analyze and quantify performances for various thread block scheduling policies. Experiment results from the implemented simulator of our model show that the legacy hardware thread block scheduling does not behave well when workload becomes heavy.

Key Words : Thread Block, GPGPU, Thread Block Scheduling, Round-Robin, Simulator, SimTBS

Ⅰ. 서 론

GPGPU는 수만 개의 스레드(Thread)들을 병렬적으로 수행하여 성능을 최대화할 수 있다[1]. 일반적으로는 전체 스레드에 의해 수행되는 단위 스레드 작업들을 동일한 수로 분할하여 그룹화된 작업을 정의하고, GPGPU 하드웨어 자원에 할당되는 기본 작업 단위로 활용한다. 대표적 GPGPU 프로그래밍 도구인 CUDA[2][3]에서는 그림 1과 같이 개별 스레드 작업들을 최대 3차원까지의 인덱스로 식별이 가능한 다수의 스레드블록(Thread Block, 이하 TB)에 포함시키며, TB들은 다시 3차원의 그리드(Grid)로 표현하여 전체 GPGPU 작업을 정의한다. 그런데 GPGPU 프로세싱 하드웨어는 많게는 수십 개의 스트림 멀티프로세스(Stream Multiprocessor, 이

*정회원, 이화여자대학교 임베디드소프트웨어연구센터
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자 2020년 5월 30일, 수정완료 2020년 7월 2일
게재확정일자 2020년 8월 7일

Received: 30 May, 2020 / Revised: 2 July, 2020 /
Accepted: 7 August, 2020
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

하 SM)로 구성되므로, 제출된 작업의 TB들을 SM에 할당하는 스케줄러(Thread Block Scheduler, 이하 TBS)가 필요하며, 이러한 TBS는 GPGPU 장치 내에 하드웨어적으로 구현되어 있다.

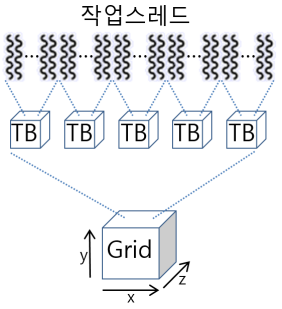

그림 1. GPGPU 작업의 스레드 그룹화 구조
Fig. 1. Thread grouping structure for GPGPU task

TBS의 기본적인 역할은 각 SM에 할당된 TB들의 자원 요구량을 추적하여 SM에서 최대 지원 가능한 자원 보유량을 초과하지 않도록 스케줄링을 수행하는 것이다. 단일 TB를 수행하기 위해 필요한 SM 내자원의 종류나 양은 GPGPU 작업 빌드시에 정적으로 결정되므로, SM의 용량을 초과하지 않도록 스케줄링하는 것은 복잡한 문제가 아니다. 그런데, SM내의 TB를 통해 할당된 스레드 개수가 증가할수록 수행속도는 상대적으로 감소하므로 SM간에 균형있게 할당하는 것이 필수적이다.

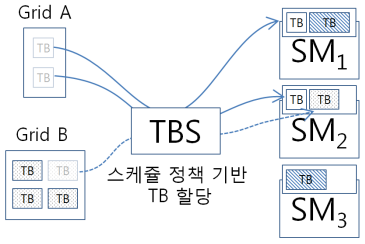

그림 2. SM간의 균형있게 TB를 할당하는 예시
Fig. 2. Example of balanced TB allocation between SMs

기존의 GPGPU 장치내의 스케줄러는 제출된 작업의 TB들을 보유한 SM들에 순차적으로 할당하는 라운드로빈(Round-Robin) 정책을 사용하고 있다. 라운드로빈은 자원 분배 알고리즘으로 널리 사용되는 방식으로서, 단순하여 하드웨어적으로 구현하기 쉬운 장점이 있다. 그러나 SM간의 자원 사용 현황을 고려하지 않아 부하가 많은 경우에 자원 할당의 불균형이 발생할 수 있다.

이처럼 TB 스케줄링은 SM의 활용률에 직접적인 영향을 미치며, 이는 GPGPU 작업의 처리율과 수행 성능을 결정짓게 된다. 따라서 다양한 GPGPU 워크로드에 따라 기본 정책 이외의 좀 더 효율적인 스케줄링 정책의 연구가 필요하다. 본 논문에서는 다양한 스레드블록 스케줄링 방식의 성능을 정량적으로 분석할 수 있는 스레드블록 스케줄러 모델을 제안하고, 구현된 시뮬레이터의 성능 결과를 통해 기존 GPGPU의 스레드블록 스케줄링 방식이 작업 부하가 높은 경우에는 적합하지 않음을 보이고자 한다.

II. 본 론

1. GPGPU 내부 구조

SM은 SIMT(Single Instruction Multiple Thread) 방식으로 단일 스레드 흐름의 프로그램 구조를 유지하면서 다수 스레드 실행을 투명하게 하드웨어적으로 지원한다. SM 당 최대 수행 가능한 스레드의 수는 일반적으로 2,048개로서, TBS는 SM에 TB를 할당시 이 제한을 위배하지 않도록 SM당 허용 가능한 스레드 수를 추적 관리해야 한다[4].

하드웨어적으로 동시에 수행되는 스레드 그룹을 워프(Warp)라고 부르며 최대 32개의 스레드를 동시에 수행할 수 있다. 1개의 TB는 최소 1개 이상의 워프로 구성되며, 32개를 초과할 때 마다 워프의 수가 증가한다. 동시에 수행 가능한 워프의 수는 SM에서 지원하는 다양한 실행 유닛(Execution Unit)의 수에 의존적이다. 이러한 관점에서 SM은 TB내의 모든 스레드를 동시에 수행하는 것이 아니라, TB에 포함된 워프들이 실행 유닛에 할당되어 그 워프에 속한 스레드들을 동시에 수행시키는 것으로 볼 수 있다. SM당 스레드 수와 마찬가지로 워프의 수도 TBS에서 스케줄링시 고려하여야 한다. 이와 같이 TBS는 SM에 TB를 할당할 때 SM의 다양한 자원들을 추적 관리하여 SM에서 허용 가능한 자원 사용량을 초과하지 않도록 관리해야 한다. 관리해야 하는 자원 종류는 다음과 같다.

- 레지스터(Register), 공유메모리(Shared Memory)
- 스레드 정보 및 스레드블록 정보
- 하드웨어 워프(Warp) 정보

동일한 GPGPU 작업을 구성하는 모든 TB들은 자원 종류별 요구량이 동일하지만, 다른 작업들에 속하는 TB들 간에는 자원 요구량이 보통 상이하다. 그러므로 TBS는 SM별로 모든 종류의 자원에 대한 현재 사용량을 관리하고 있어야 한다. 그런데 SM에서 동시에 수행되는 워프

에 의해 모든 실행 유닛이 사용 중이라고 한다면, 실행 유닛을 할당 받지 못한 워프들은 대기하여야 한다. 이와 같이 여러 이유로 워프의 작업이 진행이 되지 못하는 스톨 워프(Stalled Warp)가 발생하게 되면, 수행 성능이 저하될 수 있다. 실행 유닛에 의한 스톨 외에도 메모리 참조로 인한 스톨도 심각한 성능 저하를 야기한다. TB 스레드 간에만 사용되는 공유 메모리의 경우 스레드 수가 증가하면서 점진적으로 100 cycle까지 지연시간이 증가하며, 모든 작업 스레드 간에 공유되는 전역 메모리의 경우는 수백 cycle이 소요된다[5].

2. 스레드 블록 스케줄러 모델

GPGPU의 TBS 기능을 시뮬레이션하기 위해 GPGPU를 구성하는 하드웨어 자원들과 작업들에 대한 정확한 모델링이 필요하다. TBS의 스케줄링 정책에 대한 GPGPU 작업 성능의 영향도를 직접적으로 파악하기 위해서 본 논문에서는 TB를 기준으로 SM과 GPGPU 작업을 정의하고자 한다.

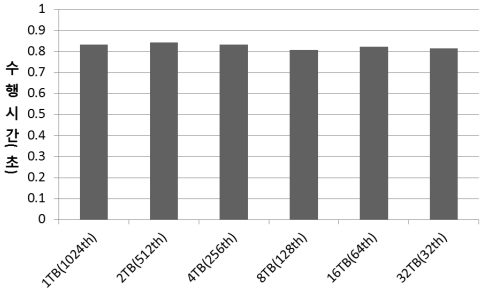

그림 3. 동일한 스레드 개수로 구성된 작업의 수행시간 비교
Fig. 3. Execution time comparison of tasks with same thread count

GPGPU 작업별로 TB내에 포함되는 스레드 수는 자유롭게 정의될 수 있지만, 제안하는 모델에서는 워프에 최대 포함 가능한 32개로 구성되는 마이크로 TB(micro TB, mTB)를 기반으로 작업의 TB와 SM별 수용 가능한 자원을 정의한다. 그림 3은 실제 GPGPU상에서 전체 작업의 스레드 수를 1,024개로 유지하도록 TB의 수와 TB 내의 스레드 수를 다양하게 변화시키면서 수행속도를 측정한 것이다. 결과에서 나타나듯이 실제 작업시 TB 내의 스레드 수에 맞게 복수개의 mTB로 수행하면 성능의 차이는 발생하지 않는 것을 확인할 수 있다. 이는 mTB를 기준으로 TB의 작업량과 SM의 자원보유량을 모델링하여도 실제 수행 성능과 차이가 없음을 의미한다.

GPGPU에는 시뮬레이션 기간 동안 작업들이 지속적으로 요청되며, 요청된 순서대로 TBS가 스케줄링되는 것을 가정한다. TBS에 제출되는 GPGPU 작업은 다음의 속성들로 정의할 수 있다.

- 작업 시작 시각
- 기본 작업 수행 시간
- 작업의 전체 스레드 수에 대응하는 mTB 개수
- mTB별 컴퓨팅 자원 사용량
- mTB별 메모리 자원 사용량

작업 시작 시각은 작업이 제출되는 시각이며, 기본 작업 수행시간은 단일 mTB가 GPGPU를 사용하여 소요되는 시간으로서 다른 mTB와의 자원 경쟁으로 지연되는 시간이 없는 수행 시간이다. 작업의 양은 mTB 개수로 정의된다.

작업은 복수개의 mTB로 구성되며, 작업 수행시 소요되는 자원은 크게 컴퓨팅 자원과 메모리 자원으로 이원화하여 mTB를 기준으로 사용량을 정의한다. SM은 최대 컴퓨팅 자원 보유량을 정의하여, mTB가 해당 SM에 배치될 때 SM에서 컴퓨팅 자원 사용량을 소진하게 되며, 잔여 컴퓨팅 보유량이 부족한 경우 스케줄링이 불가하다.

mTB별 메모리 자원 사용량은 모든 SM의 mTB에서 접근 가능한 전역 메모리 사용량으로서 GPGPU내의 최대 전역 메모리 보유량을 정의하고, mTB가 임의의 SM에 할당될 때, 메모리 보유량을 차감한다. 잔여 메모리 보유량이 부족한 경우, mTB에 대한 스케줄링은 허용되지 않는다.

본 논문의 스케줄러 모델에서 GPGPU의 부하 정도에 따른 작업의 수행시간을 합리적으로 산출하기 위해 기본 작업 수행시간을 보정하는 기능을 제공한다. SM별 컴퓨팅 자원 사용량에 따른 컴퓨팅 오버헤드와 전역 메모리 사용량에 따른 메모리 접근 오버헤드를 정의하여 실제 작업의 수행시간은 오버헤드에 비례하여 수행시간이 증가되도록 설계되었다.

3. TBS 스케줄링 정책

GPGPU에서 전통적으로 사용하는 라운드로빈 방식은 SM의 자원 사용량을 감안하지 않고 단순하게 순차적으로 작업의 TB들을 분배한다. 기존 방식과 달리 BFA(Breadth First Allocation)는 SM의 자원 사용량을 모니터링하여 사용량이 작은 SM에 우선적으로 할당하는 정책이다. 전체적으로 SM간의 자원사용량은 균형적으로 할당될 것이며, 작업 수행시간의 오버헤드를 최소화할 수 있을 것이다.

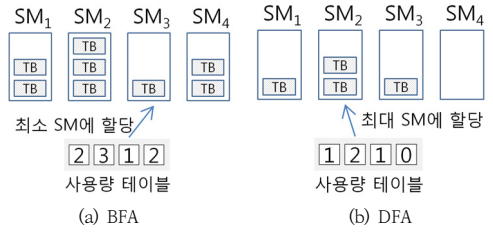

그림 4. BFA와 DFA 정책의 TB 할당 방식 비교
Fig. 4. Comparison of TB allocation methods between BFA and DFA

DFA(Depth First Allocation)는 BFA와는 역으로 자원 사용량이 가장 많은 SM에 할당하는 정책이다. 이는 GPGPU의 부하가 작은 경우에는 유리하지 않지만, 부하가 많으면서 컴퓨팅 요구량이 큰 작업이 요청되는 경우 GPGPU의 활용률을 높일 수 있는 스케줄링 정책이다. 그림 5는 BFA에서는 수용이 불가한 워크로드를 DFA에서는 수용하는 경우를 설명한 예시이다.

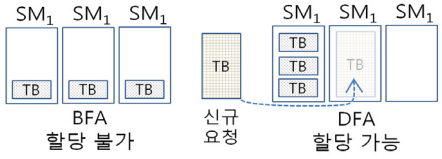

그림 5. 자원 요구량이 큰 TB에 유리한 할당 방식
Fig. 5. Allocation policy advantageous to a TB with high resource requirement

4. SimTBS 구현 결과

제안된 스레드 블록 스케줄러 모델에 따라 SimTBS[6] 시뮬레이터를 구현하였으며, 오픈소스로 공개되어 있다. 본 논문에서 제안된 정책들 이외에도 다양한 스케줄링 정책들을 실험할 수 있도록 스케줄링 정책을 모듈화된 형태로 프레임워크화 하였다. SM의 수나 컴퓨팅 및 메모리 오버헤드, 작업 워크로드 등의 설정 파라미터들은 모두 조정 가능하도록 설계되었으며, 워크로드 자동 생성 도구도 포함되어 있다.

Ⅲ. 성능 평가

TBS 시뮬레이션을 위한 작업 워크로드 생성은 명목적인 활용률을 기준으로 각 작업의 자원 요구량과 수행시간을 임의의 값으로 산출하였다. 명목 활용률은 GPGPU 내의 모든 SM들의 자원보유량 대비 수행 중 작업들의 컴퓨팅 자원 요구량의 비율이다. 명목 활용률은 실제 시뮬레이션에서 관찰되는 SM 활용보다는 훨씬 높게 책정된다.

생성된 다양한 GPGPU 워크로드에 대해서 TBS에 3가지 스케줄링 정책인 라운드로빈(RR), BFA, DFA를 각각 적용하여 성능 평가를 수행하였다. 스케줄링 정책의 평가 측도는 ANTT(Averaged Normalized Turnaround Time)[7]를 사용하였다. ANTT는 단일 GPGPU 작업이 수행된 시간 대비 다른 작업의 간섭으로 인한 컴퓨팅 및 메모리 참조 오버헤드가 반영된 상대적인 작업의 수행시간을 뜻한다. ANTT가 2인 경우는 단독으로 작업이 수행된 경우에 비해 여러 작업이 동시에 수행된 경우 2배의 수행시간이 소요된 것을 뜻하며, 동일한 작업 워크로드를 수행한 TBS의 정책 중 ANTT가 낮을수록 TBS의 성능이 좋다.

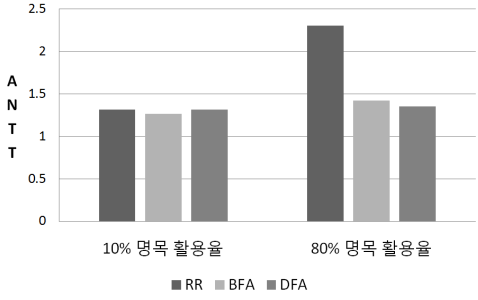

그림 6. 명목 활용률별 스케줄링 정책 성능 비교
Fig. 6. Performance comparison of scheduling policies per nominal utilization

그림 6은 부하가 낮은 워크로드와 높은 워크로드에 대해서 3가지 스케줄링 정책인 RR, BDA, DFA의 ANTT를 비교한 것이다. 낮은 부하(10% 명목 활용률)에서는 3가지 TB 할당 정책간의 성능 차이가 나타나지 않는다. 그러나 높은 부하(80% 명목 활용률)에서는 RR의 ANTT가 약 2.3으로서 다른 정책에 비해 70%정도 작업의 평균 소요시간이 더 늘어난다. 이는 다양한 작업으로 부하가 많은 경우에는 단순한 할당보다는 SM 자원의 활용률을 높일 수 있는 효율적인 스케줄링이 필요함을 나타낸다.

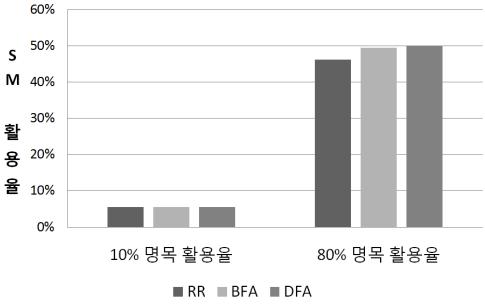

그림 7. 스케줄링 정책별 SM 활용률
Fig. 7. SM utilizations of scheduling policies

그림 7은 스케줄링 정책별로 실제 SM 기준의 활용률을 나타낸 것이다. 명목 활용률에 비해서 3가지 정책들의 실제 SM상의 활용률은 훨씬 낮다. 이는 TB 할당 정책 자체의 비효율적 측면도 있지만, 랜덤하게 작업 부하를 생성하면서 발생하는 내부 단편화의 영향도 크다. 부하가 낮은 경우의 타 정책 대비 RR의 SM 활용율의 차이는 극히 미미하다. 높은 경우의 RR의 SM 활용률은 다른 방식에 비해 약 7.8% 낮다. 활용율의 차이에 비해서 ANTT의 격차는 매우 크므로, 효율적인 TB 스케줄링을 통하여 SM 활용률을 높이는 것이 매우 중요함을 알 수 있다.

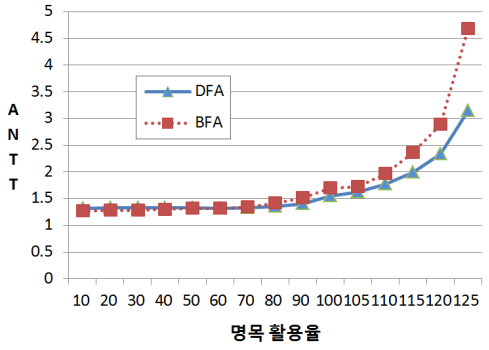

그림 8. 임의의 자원 요구량을 가진 워크로드에서의 성능 비교

Fig. 8. Performance comparison under workload with uniformally randomized resource requirement

그림 8은 BFA와 DFA에 대하여 RR 대비 향상된 성능을 명목적 활용률을 기준으로 나타낸 것이다. 명목적 활용률을 SM의 전체 컴퓨팅 보유량을 넘어서는 125%까지 늘려도 BFA와 DFA의 실제 활용률은 각각 76.1%와 76.4%이다. 그런데, 부하가 극단적으로 높은 구간에서는 DFA의 성능이 BFA보다 좋은 것을 알 수 있다.

그림 9는 작업의 컴퓨팅 자원 요구량을 SM당 최소 1개 혹은 SM의 최대 개수를 요구하도록 극단적인 2개 형태의 작업들로 부하를 생성하여 ANTT를 비교한 것이다. 이 경우는 부하가 높아질수록 자원요구량이 큰 작업이 할당되지 못하는 그림 5와 같은 상황이 자주 발생하게 될 것이다. 명목 활용률이 100%가 넘어가는 경우, BFA의 ANTT는 급격히 증가하지만, DFA는 명목 활용률이 120%(SM 활용률은 86.9%)까지도 ANTT를 4미만으로 유지하고 있다. 이와 같이 DFA는 SM의 활용도를 최대한 높이는 방향으로 TB 할당 정책을 운용하여 부하가 매우 높은 워크로드에서 유리함을 알 수 있다.

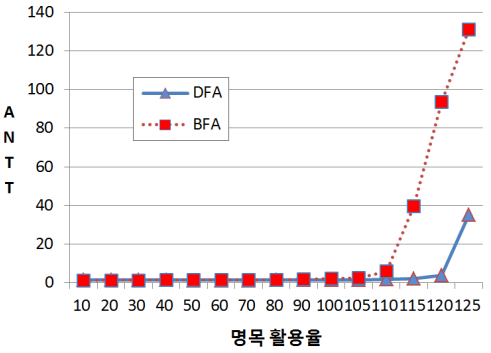

그림 9. 극단적 불균형의 자원 요구량을 가진 워크로드에서의 성능 비교

Fig. 9. Performance comparison under workload with extremely unbalanced resource requirement

Ⅳ. 결 론

본 논문은 스레드 블록 스케줄링을 위한 스케줄러 모델을 제안하고 해당 모델에 따라 구현된 오픈소스 기반의 SimTBS 시뮬레이터를 통하여 기존의 GPGPU 하드웨어 기반의 라운드로빈 방식의 문제점을 지적하였다. 라운드로빈 정책은 하드웨어적으로 구현이 간단한 장점이 있으며, 단일 작업의 성능 위주의 낮은 부하에서는 무난한 스레드 블록 스케줄링 정책이다. 그러나 대량의 GPGPU 작업이 동시에 수행되면서 다수 작업의 처리율과 소요시간이 중요한 경우[8]에는 SM의 활용률을 최대한 끌어올릴 수 있는 효율적인 TB 스케줄링 정책이 필요하다.

제안된 스레드블록 스케줄러 모델은 워프단위의 mTB를 기준으로 작업의 자원 요구량과 SM의 자원보유량을 정의하고, 컴퓨팅 자원과 메모리 사용량에 비례하여 수행시간 오버헤드를 적용하는 방식이다. gpgpusim[9] 처럼 GPGPU 하드웨어를 구성요소 단위에서 정밀하게 모사를 하는 방식이 아닌, 스레드블록 스케줄링과 연관된 GPGPU의 특성을 모사한 모델이다. 해당 모델로부터 구현된 SimTBS의 실험 결과들은 여러 GPGPU 작업으로부터 일정 부하가 발생하는 경우, 라운드로빈 방식 이외의 자원의 사용량을 고려하여 효율적으로 TB를 할당하는 BFA나 DFA가 최소 2배 이상 좋은 성능을 보인다.

References

[1] Hyo-Jeong Lim, et al., "Analysis of Job Scheduling and the Efficiency for Multi-core Mobile GPU", Journal of the Korea Academia-Industrial cooperation Society, Vol.15, No. 7, pp. 4545-4553, 2014. https://doi.org/10.5762/KAIS.2014.15.7.4545

[2] Byoung-Woo Oh, "An Efficient Technique for Clustering of Spatial Data Using GPGPU", The Journal of Korean Institute of Information Technology, Vol.17, No. 4, pp. 21-26, 2019. https://doi.org/10.14801/jkiit.2019.17.4.21

[3] https://docs.nvidia.com/cuda/ Accessed May 2020

[4] Yun-Joo Park, et al., "Analyzing Fine-Grained Resource Utilization for Efficient GPU Workload Allocation", The Journal of The Institute of Internet, Broadcasting and Communication, Vol.19, No. 1, pp. 111-116, 2019. DOI:https://doi.org/10.7236/JIIBC.2019.19.1.111

[5] Jia, Zhe, et al., "Dissecting the NVidia Turing T4 GPU via Microbenchmarking", arXiv preprint arXiv:1903.07486, 2019.

[6] https://github.com/oslab-ewha/simtbs Accessed May 2020

[7] Jason Jong-Kyu Park, et al., "Dynamic resource management for efficient utilization of multitasking gpus", Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, 2017. DOI:https://doi.org/10.1145/3037697.3037707

[8] Yeh Tsung Tai, et al., "Pagoda: Fine-grained gpu resource virtualization for narrow tasks", ACM SIGPLAN Notices, Vol.52, No. 8, pp. 221-234, 2017. DOI:https://doi.org/10.1145/3155284.3018754

[9] Ariel Aaron, et al., "Visualizing complex dynamics in many-core accelerator architectures", 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), IEEE, 2010. DOI: https://doi.org/10.1109/ISPASS.2010.5452029

저 자 소 개

조 경 운(정회원)

- 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)