The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 5, pp.49-54, Oct. 31, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.5.49
JIIBC 2022-5-8

상세 자원 이용률에 기반한 병렬 가속기용 스레드 블록 스케줄링

Thread Block Scheduling for GPGPU based on Fine-Grained Resource Utilization

반효경*, 조경운**

Hyokyung Bahn*, Kyungwoon Cho**

요 약 최근 클라우드 시스템에서 병렬가속기를 사용하는 사례가 늘면서 가속기 내에서 멀티태스킹을 통해 자원 이용률을 높이는 것이 중요한 이슈로 부각되고 있다. 본 논문에서는 병렬가속기 내 자원 사용 패턴을 컴퓨팅 중심과 메모리 중심으로 분류하여 워크로드를 배치하는 방식이 자원 이용률 측면에서 충분한 효과를 나타내지 못함을 보이고, 워크로드별 상세 자원 이용률에 기반한 새로운 스레드 블록 스케줄링 기법을 제안한다. 제안한 기법은 기존 방식과 달리 프로파일링과 스케줄링을 분리하여 스케줄링시의 오버헤드를 줄이고 병목 자원이 일치하지 않는 워크로드들을 최대한 중복 배치하여 자원 이용률을 높인다. 다양한 가상머신 시나리오에 대한 시뮬레이션 실험을 통해 제안한 기법이 병렬가속기의 처리량을 평균 130.6%, 최대 161.4%까지 개선함을 보인다.

Abstract With the recent widespread adoption of general-purpose GPUs (GPGPUs) in cloud systems, maximizing the resource utilization through multitasking in GPGPU has become an important issue. In this article, we show that resource allocation based on the workload classification of computing-bound and memory-bound is not sufficient with respect to resource utilization, and present a new thread block scheduling policy for GPGPU that makes use of fine-grained resource utilizations of each workload. Unlike previous approaches, the proposed policy reduces scheduling overhead by separating profiling and scheduling, and maximizes resource utilizations by co-locating workloads with different bottleneck resources. Through simulations under various virtual machine scenarios, we show that the proposed policy improves the GPGPU throughput by 130.6% on average and up to 161.4%.

Key Words : GPGPU, resource utilization, cloud system, multitasking, thread block scheduler

Ⅰ. 서 론

최근 매니코어 기술의 급격한 발전으로 클라우드 환경에서 이종 워크로드들이 병렬가속기 내에서 동시에 실행되는 사례가 늘고 있다. 병렬가속기는 그래픽 처리, 딥러닝, 유전자 분석 등 다양한 영역에서 컴퓨팅 작업의 빠른 처리를 위해 폭넓게 활용되고 있다[1, 2, 3]. 전통적인 병렬가속기 환경이 단일 워크로드의 빠른 실행에 초점을

*정회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 임베디드소프트웨어연구센터
접수일자 2022년 9월 11일, 수정완료 2022년 9월 29일
게재확정일자 2022년 10월 7일

Received: 11 September, 2022 / Revised: 29 September, 2022 / Accepted: 7 October, 2022
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

맞춘 반면, 최근 클라우드 시스템에 탑재된 병렬가속기들은 다중 워크로드가 가속기 내에서 동시 실행되면서 가속기 내의 자원 이용률을 높이는 것이 중요한 이슈로 부각되었다[4, 5].

그림 1에서 보는 것처럼 클라우드 시스템 내의 병렬가속기는 전통적인 호스트 자원뿐 아니라 가속기 내에서 다양한 워크로드가 함께 실행되며 자원을 공유하고 있다. 이때, 전통적인 자원인 CPU, 메모리, 스토리지 등은 호스트 운영체제 및 가상화 SW에 의해 워크로드 간 자원 플래닝이 효율적으로 이루어지고 있다[6, 7, 8]. 이에 비해 병렬가속기 내의 자원 관리는 하드웨어 로직을 단순화하고 단일 워크로드의 빠른 실행에 최적 설계되어 이종 워크로드 간 자원 공유에 비교적 취약하다. 즉, 병렬가속기 내 일부 자원이 고갈된 상태에서 또다른 자원은 유휴 상태에 머무는 경우도 빈번히 발생한다.

이를 해결하기 위해 최근 병렬가속기 내 다중 워크로드의 실행시 가속기 내 자원들의 부하균형을 추구하는 연구가 주목받고 있다[9, 10]. 가속기 내에는 SM (streaming multiprocessor)이라는 일련의 자원 집합체들이 존재하며, 워크로드를 SM에 배치할 때 메모리 중심 작업과 컴퓨팅 중심 작업을 하나의 SM에 함께 배치하여 SM 내 메모리나 컴퓨팅 자원이 골고루 활용될 수 있도록 하는 것이 이들 연구의 목표이다. 그러나, 모든 워크로드가 컴퓨팅 중심 혹은 메모리 중심인 경우 이러한 자원 이용률 저하 문제는 해결이 불가능하다. 또한, 기존 연구 중 일부는 워크로드 간 간섭 효과를 확인하기 위해 작업을 동일 SM에 시범적으로 배치한 후 실시간으로 성능 프로파일링을 하고 재배치하는 방식을 택하여 오버헤드를 초래한다는 문제점이 있다.

본 연구에서는 워크로드별 자원 이용률이 상세 자원 수준까지 분석될 경우 실시간 프로파일링 없이도 병렬가속기 내 자원들이 효율적으로 공유될 수 있다는 점에 착안하여 새로운 스케줄링 기법을 제안한다. 제안한 기법은 컴퓨팅 중심의 워크로드들(혹은 메모리 중심의 워크로드들)만 실행중인 경우에도 워크로드별 세부 병목 자원이 다를 경우 이들을 동일 SM에 배치하여 자원이용률을 높일 수 있음을 보인다. 예를 들어 컴퓨팅 중심 워크로드 A와 B 중 A는 정수 연산을 수행하고 B는 부동소수점 연산을 수행할 경우 이들을 동일 SM에 배치하더라도 SM 내에 정수 연산장치와 부동소수점 연산장치가 별도로 있어 충돌 없이 동시 실행이 가능하며 본 논문은 이러한 특성을 활용하는 것이다.

자원이용률을 극대화하기 위해 본 논문은 워크로드들을 컴퓨팅 중심형과 메모리 중심형으로만 나누는 대신 병목이 되는 상세 자원에 근거한 세분화 작업을 수행한다. 이에 따라 동종 분류에 속하는 워크로드인 경우에도 세부 병목 자원이 상이할 수 있음을 보이고 그럴 경우 워크로드들을 동일 SM에 배치한다. 일반적으로 SM 내 워크로드 배치 단위가 스레드 블록이므로 본 논문에서는 자원 이용률을 높이는 새로운 스레드 블록 스케줄링 기법을 설계한 후 스레드 블록을 각 SM에 순차 할당하되 상세 자원 이용률이 포화상태인 SM을 피하여 전체적인 자원이용률을 극대화하였다.

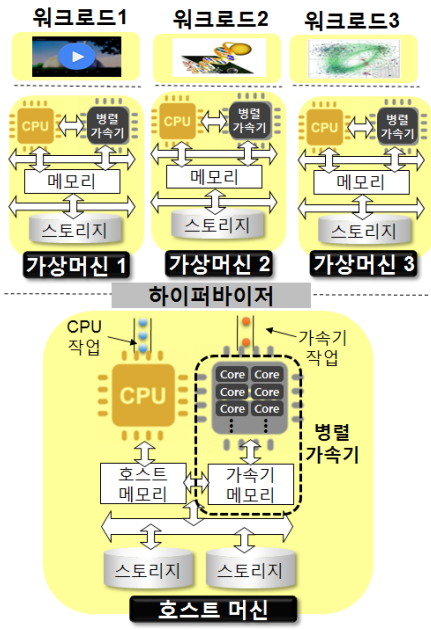


그림 1. 클라우드 환경의 병렬 가속기 구조
Fig. 1. GPGPU architecture in cloud environments.

다양한 가상머신 환경에서의 시뮬레이션 실험을 통해 제안한 기법이 병렬가속기에서 널리 사용되는 라운드 로빈 기법 대비 평균 130.6%, 최대 161.4%의 처리량 개선 효과가 있음을 보인다.

II. 스레드 블록 스케줄링 기법

본 장에서는 병렬가속기의 상세 자원 이용률에 근거한 새로운 스레드 블록 스케줄링 기법을 설명한다. 병렬가속기의 스레드 블록 스케줄링을 위해 가장 널리 사용되는 방법은 라운드 로빈이다[11]. 라운드 로빈은 모든 SM

에 스레드 블록들을 하나씩 번갈아 할당하는 방식으로 알고리즘이 단순하고 하드웨어적인 로직으로 구현이 쉬워 네트워크 패킷 전송이나 CPU 스케줄링시 널리 활용되고 있다[11, 12, 13]. 그러나, 라운드 로빈은 워크로드의 자원 이용률을 고려하지 않아 자원들 간에 부하균형을 이루는 스케줄링 방법은 아니다. 예를 들어 특정 SM에 할당된 모든 스레드 블록들이 FP64 연산만을 수행하는 경우 해당 SM의 FP64 유닛에 과부하가 발생하지만, FP32 유닛은 유휴상태에 머물러 있을 수 있으며, 다른 SM에서는 상황이 정반대일 수 있다.

본 논문에서 제안하는 스케줄링 기법은 라운드 로빈과 유사하게 대기중인 스레드 블록들을 큐에서 하나씩 꺼내어 SM에 순차적으로 할당한다. 이때, 만약 현재의 할당으로 인해 SM 내 특정 자원의 이용률이 그 한계치를 넘어설 경우 해당 SM에의 할당을 건너뛰고 다음 SM을 탐색한다. 과거의 연구에서도 자원 사용 특성이 상이한 스레드 블록들을 동일 SM에 배치하여 자원 이용률을 높이려는 시도가 있었다[9]. 그러나, 기존 방식은 컴퓨팅 중심, 메모리 중심 등 자원 사용 패턴의 큰 흐름에 기반해서 배치할 뿐 본 논문의 방식처럼 상세 자원의 이용률에 근거하지 않아 그 한계가 뚜렷하다.

기존 방식과 본 논문이 제안한 방식의 정확한 동작 차이를 예를 통해 살펴보자. 4개의 SM으로 구성된 병렬가속기가 있고 3개의 스레드 블록이 큐에 대기 중이라고 하자. 표 1과 2는 현재 각 SM의 자원 이용률과 대기중인 스레드 블록들의 자원 수요를 보여주고 있다. 본 예시에서는 컴퓨팅 자원이 FP32와 FP64의 2종만 존재하고 메모리 자원은 디바이스 메모리와 공유 메모리의 2종만 존재한다고 가정한다. 표에서 자원 이용률은 동일 자원 분류에 속한 상세 자원이용률 칼럼에서 가장 큰 값으로 계산된 결과이다.

제안하는 기법은 TB1을 SM1에 할당할 경우 모든 상세 자원 이용률이 여전히 100% 이하이므로 할당을 수락한다. 그 결과 SM1의 상세자원이용률은 (90, 10, 0, 10)에서 (100, 100, 0, 20)로 업데이트된다. 이와 유사하게, TB2와 TB3은 각각 SM2와 SM3에 할당된다. 반면, 기존의 스케줄링 방식은 현재 SM의 자원이용률과 대기중인 스레드 블록의 자원이용률을 비교해서 스케줄링 여부를 결정한다. 따라서, TB1이 SM1이나 SM2에 할당될 경우 컴퓨팅 자원의 이용률이 100%를 넘기 때문에 TB1은 SM1과 SM2 어디에도 할당되지 못한다. 한편, SM3에 할당될 경우 과부하가 발생하지 않으므로 TB1은 최종적으로 SM3에 할당된다. 이와 유사한 방식으로 TB2는

표 1. SM별 현재 자원이용률
Table 1. Current resource utilizations of SMs

	상세자원이용률				자원이용률	
	컴퓨팅 (FP32)	컴퓨팅 (FP64)	메모리 (Device)	메모리 (Shared)	컴퓨팅	메모리
SM1	90	10	0	10	90	10
SM2	50	80	0	10	80	10
SM3	10	10	0	10	10	10
SM4	10	80	0	10	80	10

표 2. 대기중인 스레드 블록의 자원 요구량
Table 2. Resource demand of pending thread blocks

	상세자원이용률				자원이용률	
	컴퓨팅 (FP32)	컴퓨팅 (FP64)	메모리 (Device)	메모리 (Shared)	컴퓨팅	메모리
TB1	10	90	0	10	90	10
TB2	10	20	80	10	20	80
TB3	80	0	10	10	80	10

SM4에 할당된다. 그러나, 이후 TB3의 할당을 위해 SM1부터 SM4까지를 순차적으로 검색한 결과 기존 방식은 모든 SM에서 컴퓨팅 자원의 과부하로 인해 더 이상 할당이 불가능한 것을 확인할 수 있다. 이 경우, TB3은 가용 SM이 생길 때까지 큐에서 대기해야 한다. 라운드 로빈 스케줄링의 경우 TB1, TB2, TB3이 각각 SM1, SM2, SM3에 할당되며, 이는 라운드 로빈이 자원 이용률이 100%를 넘어서더라도 추가적인 스레드 블록의 할당이 하드웨어적인 한계를 넘어서지 않는 이상 SM에 순차적인 할당을 진행하기 때문이다.

한편 본 논문이 활용하는 상세 자원 이용률의 정보는 워크로드에 대한 사전 프로파일링 결과를 통해 획득되므로 스케줄 시점에 온라인으로 오버헤드를 발생시키지 않는다. 또한, 워크로드별 프로파일링 작업에는 30 ms 이하의 시간이 소요되어 실제 워크로드의 실행과 비교할 때 그 오버헤드는 미미한 수준이다. 따라서, 제안한 기법이 비록 워크로드에 대한 사전지식을 필요로 하지만 워크로드의 최초 실행시에도 짧은 프로파일링 후 이를 곧바로 활용하는 것이 가능함을 의미한다. 따라서, 제안하는 기법의 실행시 발생하는 유일한 오버헤드는 각 SM별 자원이용률을 파악하는 일일 것이다. 그러나, 이 작업은 스레드 블록 할당 및 완료 시 스레드 블록 스케줄러가 SM별 자원 이용률을 파악하는 정도로 사실상 오버헤드를 발생시키지는 않는다.

Ⅲ. 성능 평가

제안한 기법의 효과를 입증하기 위해 15개의 가상머신으로 구성된 4가지 시나리오에 대해 시뮬레이션 실험을 진행하였다. 표 3은 각 시나리오의 워크로드 구성을 보여주고 있다. 본 논문의 실험은 호스트가 20개의 CPU 코어와 64GB의 메모리로 구성되며, Nvidia Tesla V100 모델에 근거해 80개의 SM과 16GB의 메모리를 가지는 2개의 GPGPU 장치로 구성된다. 본 논문에서는 클라우드 상에서 이종 워크로드가 동시수행됨에 따른 효과를 확인하기 위해 각 가상머신에서 주어진 워크로드가 반복수행되는 것으로 가정했다.

제안한 기법인 상세자원이용률 기반 라운드 로빈의 비교 대상으로는 라운드 로빈과 이용률 기반 라운드 로빈을 사용했다. 그림 2는 제안한 기법과 2가지 비교 대상의 호스트 및 각 가상머신 처리량을 보여주고 있다. 그림에서 보는 것처럼 제안한 기법은 모든 시나리오와 워크로드에 대해 라운드 로빈 대비 우수한 성능을 나타내었으며, 성능 개선 효과는 평균 130.6%, 최대 161.4%에 이르렀다. 라운드 로빈의 성능이 좋지 않은 이유는 SM 간의 부하 균형을 이루지 못했기 때문으로 볼 수 있다. 비록 라운드 로빈이 각 SM마다 스레드 블록을 하나씩 할당하며 부하균형을 추구하지만 좋은 결과가 도출되지 않은 이유는 워크로드의 특성을 고려하지 않기 때문이다. 이는 대부분의 병렬가속기가 라운드 로빈을 채택하고 있다는 점을 미루어 볼 때 스레드 블록 스케줄러의 성능 개선 여지가 많이 있음을 뜻한다.

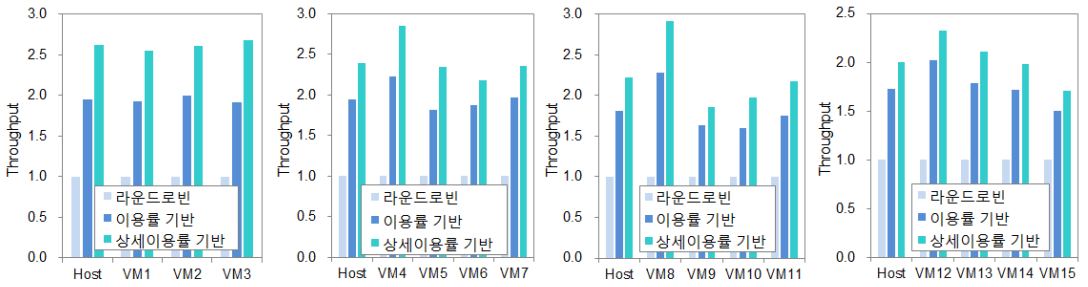
이용률 기반 라운드 로빈 역시 라운드 로빈에 비해 우수한 성능을 나타내었으나, 제안한 기법이 그보다도 평균 23.7% 최대 34.1%의 우수한 성능 개선을 나타내었다. 이는 제안한 기법이 각 응용별로 병목을 일으키는 상세 자원을 정확하게 분석하고 동일 SM에 동일 자원이 병목인 워크로드를 함께 할당하지 않기 때문이다. 이에 반해 이용률 기반 라운드 로빈은 상세 자원 수준까지 고려하지 않아 이러한 효과를 얻지 못한 것으로 볼 수 있다. 가장 큰 성능 개선은 시나리오 1에서 나타났으며 제안한 기법의 성능 개선은 34.1%였다. 이는 시나리오 1의 세 워크로드가 모두 메모리 중심으로 이용률 기반 라운드 로빈이 이들을 동일 SM에 배치하는 데에는 한계가 있지만 제안한 기법은 이들 가상머신의 상세 병목 자원이 다른 경우 동일 SM에 배치해서 병렬성 효과를 극대화시킬 수 있기 때문이다. 가상머신 1과 2가 디바이스 메모리를 많이 사용하는 반면 가상머신 3은 디바이스 메모리의 이용률이 낮고 통합 캐시의 이용률이 높아 제안하는 기법은 충돌 없이 가상머신 3의 워크로드를 가상머신 1 또는 2와 동일 SM에 중복 배치하여 자원 이용률을 극대화하나 이용률 기반 라운드 로빈은 그런 효과를 누리지 못한다.

표 3. 실험에 사용된 가상머신 시나리오
Table 3. Virtual machine scenarios used in experiments.

시나리오	가상머신	워크로드	분류
Scenario 1	VM1	Black Scholes	메모리 중심
	VM2	Stream Cluster	메모리 중심
	VM3	Convolution Texture	메모리 중심
Scenario 2	VM4	Nbody	컴퓨팅 중심
	VM5	Hotspot	컴퓨팅 중심
	VM6	LavaMD	컴퓨팅 중심
	VM7	Srad	컴퓨팅 중심
Scenario 3	VM8	Stream Cluster	메모리 중심
	VM9	Convolution Texture	메모리 중심
	VM10	Fluids GL	메모리 중심
	VM11	HS Optical Flow	메모리 중심
Scenario 4	VM12	Particle Filter	컴퓨팅 중심
	VM13	Hotspot	컴퓨팅 중심
	VM14	Convolution Texture	메모리 중심
	VM15	Black Scholes	메모리 중심

시나리오 2는 세 워크로드가 모두 컴퓨팅 중심으로 이용률 기반 라운드 로빈은 서로 다른 두 워크로드를 동일 SM에 배치하지 못한다. 이 경우, SM 내 일부 자원의 이용률 저하가 불가피하다. 예를 들어 하나의 워크로드가 FP32 연산만을 집중적으로 수행하고 다른 워크로드가 FP64 연산을 집중 수행할 경우 이용률 기반 라운드 로빈은 이들을 동일 SM에 배치하지 못하므로 SM 내의 연산 유닛 중 한 종류는 유휴 상태에 머물 수밖에 없다. 이와 달리 제안한 기법은 이러한 컴퓨팅 중심 워크로드 중에도 세부적인 이용 자원이 다를 경우 이들을 동일 SM에 배치하여 FP32와 FP64 연산장치의 이용률을 동시에 높이며, 그 결과 호스트 머신과 워크로드의 처리량을 개선한다. 시나리오 2에서 제안한 기법의 성능 개선 효과는 라운드 로빈 대비 139.3%, 이용률 기반 라운드 로빈 대비 22.7%임을 확인할 수 있었다.

시나리오 3은 메모리 중심 워크로드 4종으로 구성되며, 그 중 가상머신 9는 통합 캐시를 주로 사용하는 반면, 나머지 3종의 가상머신은 디바이스 메모리를 집중적으로 사용한다. 따라서, 제안한 기법은 가상머신 9를 다른 가상머신과의 자원 충돌 없이 어떤 SM에든 배치하는 것이 가능하다. 그러나, 다른 가상머신들은 병목 자원이 동일하므로 하나의 SM에 같이 배치하는 것이 불가능하다. 따라서, 제안한 기법의 성능 개선 효과는 앞선 시나리오들에 비해 다소 적은 것을 확인할 수 있다.



(a) 시나리오 1 (b) 시나리오 2 (c) 시나리오 3 (d) 시나리오 4

그림 2. 호스트 및 가상머신 처리량 비교
Fig. 2. Comparison of throughput in host and virtual machines.

끝으로 시나리오 4에서는 컴퓨팅 중심 워크로드 2종과 메모리 중심 워크로드 2종이 가상머신에서 수행되는 시나리오이다. 이 경우 이용률 기반 라운드 로빈이 컴퓨팅 중심과 메모리 중심 워크로드를 동일 SM에 배치하는 것이 가능하다. 그러나, 제안한 기법은 메모리 중심 워크로드인 가상머신 14와 15의 상세 병목자원이 달라 동일 SM에 여러 가상머신을 함께 배치할 수 있는 가능성을 더욱 높일 수 있다. 비록 성능 차이가 앞선 시나리오들보다는 적지만 제안한 기법은 시나리오 4에서 이용률 기반 라운드 로빈 대비 15.5%의 성능 개선을 나타낸 것을 확인할 수 있었다.

Ⅳ. 결 론

클라우드 시스템에서 여러 가상머신이 동시에 실행되면서 워크로드 사이의 병렬가속기 내 자원 공유 효과가 중요해지고 있다. 본 논문에서는 워크로드별 상세 자원 이용률에 기반해서 동일한 특성의 워크로드인 경우에도 상세 병목 자원이 상이할 수 있음을 분석하고 이에 근거한 새로운 스레드 블록 스케줄링 기법을 제안하였다. 기존 방식과 달리 본 논문은 프로파일링과 스케줄링을 분리하여 스케줄링시의 오버헤드를 줄이고 상세 병목 자원이 일치하지 않는 경우 워크로드들을 동일 SM에 최대한 중복 배치하여 병렬가속기 내 전체 자원 이용률의 극대화를 추구하였다. 다양한 가상머신 시나리오에 대한 시뮬레이션 실험을 통해 제안한 기법이 병렬가속기의 처리량을 평균 130.6%, 최대 161.4% 개선함을 보였다.

References

[1] A. Fonseca and B. Cabral, “Prototyping a GPGPU neural network for deep-learning big data analysis,” Big Data Research, vol. 8, pp. 50-56, 2017.
DOI: https://doi.org/10.1016/j.bdr.2017.01.005

[2] G. Hwang and Y. Kim, “Implementation of a CUDA C matrix multiplication program for multiple GPUs,” Journal of KIIT, vol. 18, no. 11, pp. 47-54, 2020.
DOI: http://doi.org/10.14801/jkiit.2020.18.11.47

[3] D. Shin, K. Cho, and H. Bahn, “File type and access pattern aware buffer cache management for rendering systems,” Electronics, vol. 9, no. 1, article 164, 2020.
DOI: https://doi.org/10.3390/electronics9010164

[4] S. Pai, M.J. Thazhuthaveetil, and R. Govindarajan, “Improving GPGPU concurrency with elastic kernels,” ACM SIGARCH Computer Architecture News, vol. 41, pp. 407-418, 2013.
DOI: https://doi.org/10.1145/2490301.2451160

[5] K. Cho and H. Bahn, “Performance analysis of thread block schedulers in GPGPU and its implications,” Applied Sciences, vol. 10, no. 24, article 9121, 2020.
DOI: https://doi.org/10.3390/app10249121

[6] O. Kwon, H. Bahn, and K. Koh, “Popularity and prefix aware interval caching for multimedia streaming servers,” Proc. IEEE CIT Conference, pp. 555-560, 2008.
DOI: https://doi.org/10.1109/CIT.2008.4594735

[7] H. Bahn and J. Kim, “Separation of virtual machine I/O in cloud systems,” IEEE Access, vol. 8, pp. 223756-223764, 2020.
DOI: https://doi.org/ 10.1109/ACCESS.2020.3044172

[8] J. Park and E. Park, “Performance evaluation of IoT cloud platforms for smart buildings,” Journal of the Korea Academia-Industrial cooperation Society(JKAIS), vol. 21, no. 5 pp. 664-671, 2020.
DOI: https://doi.org/10.5762/KAIS.2020.21.5.664

[9] J. Park, Y. Park, and S. Mahlke, “Dynamic resource

management for efficient utilization of multitasking GPUs," Proc. ACM ASPLOS Conference, pp. 527-540, 2017.
DOI: https://doi.org/10.1145/3037697.3037707

[10] Y. Park, D. Shin, K. Cho, and H. Bahn, "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

[11] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," IEEE/ACM Trans. Networking, vol. 4, no. 3, pp. 375-385, 1996.
DOI: https://doi.org/10.1109/90.502236

[12] S. Yoo, Y. Jo, and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," IEEE Trans. Industrial Informatics, vol. 18, no. 1, pp. 631-641, 2022.
DOI: https://doi.org/10.1109/TII.2021.3067714

[13] S. Yoon, H. Park, K. Cho and H. Bahn, "Supporting Swap in Real-Time Task Scheduling for Unified Power-Saving in CPU and Memory," IEEE Access, vol. 10, pp. 3559-3570, 2022.
DOI: https://doi.org/10.1109/ACCESS.2021.3140166

저 자 소 개

반 효 경(정회원)

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

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

※ This work was supported by the ICT R&D program of MSIT/IITP (2018-0-00549, Extremely Scalable Order Preserving OS for Manycore and Non-volatile Memory) and (2020-0-00121, development of data improvement and dataset correction technology based on data quality assessment).