The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 2, pp.71-76, Apr. 30, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.2.71
JIIBC 2022-2-10

다중 워크로드 환경을 위한 GPGPU 스레드 블록 스케줄링

Thread Block Scheduling for Multi-Workload Environments in GPGPU

박소연*, 조경운, 반효경*****

Soyeon Park*, Kyung-Woon Cho**, Hyokyung Bahn***

요 약 대규모 병렬 워크로드를 GPGPU의 연산 유닛에 할당하기 위한 스케줄링으로 라운드 로빈 방식이 널리 사용되고 있다. 라운드 로빈은 작업을 각 연산 유닛에 순차적으로 할당하여 구현이 쉽다는 장점이 있으나, 클라우드와 같은 다중 워크로드 환경에서는 연산 유닛 간 부하 균형이 잘 이루어지지 않는 문제점이 발생한다. 본 논문에서는 이러한 문제를 해결하기 위해 새로운 스레드 블록 스케줄링을 제안한다. 제안하는 방식은 다양한 GPGPU 워크로드가 만들어낸 스레드 블록들을 그 작업량에 근거해 다중큐로 관리하고 각 연산 유닛의 잔여 자원을 가장 잘 활용할 수 있는 큐에서 스레드 블록을 선택하여 연산 유닛들의 자원 이용률을 극대화시키고 부하균형을 유도한다. 다양한 부하 환경에서의 시뮬레이션 실험을 통해 제안하는 방식이 라운드 로빈 대비 평균 24.8%의 성능개선 효과가 있음을 보인다.

Abstract Round-robin is widely used for the scheduling of large-scale parallel workloads in the computing units of GPGPU. Round-robin is easy to implement by sequentially allocating tasks to each computing unit, but the load balance between computing units is not well achieved in multi-workload environments like cloud. In this paper, we propose a new thread block scheduling policy to resolve this situation. The proposed policy manages thread blocks generated by various GPGPU workloads with multiple queues based on their computation loads and tries to maximize the resource utilization of each computing unit by selecting a thread block from the queue that can maximally utilize the remaining resources, thereby inducing load balance between computing units. Through simulation experiments under various load environments, we show that the proposed policy improves the GPGPU performance by 24.8% on average compared to Round-robin.

Key Words : GPGPU, Resource Utilization, Load Balancing, Thread Block Scheduling

Ⅰ. 서 론

최근 매니코어 컴퓨팅 기술의 발전으로 GPGPU (General Purpose Graphics Processing Unit)가 PC, 워크스테이션뿐 아니라 클라우드 환경에서도 널리 사용되고 있다. GPGPU는 그래픽 연산부터 빅데이터에 이르는 다양한 영역의 병렬 처리에 이용되고 있으며, 4차산업혁명시대의 도래로 머신러닝 및 AI 응용에서도 폭넓게

*비회원, 이화여자대학교 컴퓨터공학과
*정회원, 이화여자대학교 임베디드소프트웨어연구센터
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자 2022년 2월 19일, 수정완료 2022년 3월 19일
게재확정일자 2022년 4월 8일

Received: 19 February, 2022 / Revised: 19 March, 2022 / Accepted: 8 April, 2022
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

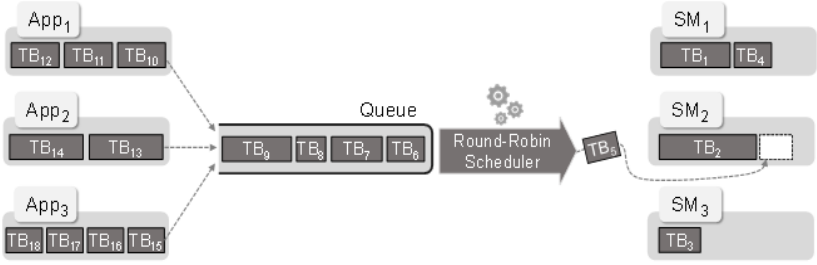

그림 1. 라운드 로빈 스케줄링에서 SM에 스레드 블록(TB)을 할당하는 예
Fig. 1. An example of Round-Robin scheduling that allocates Thread Blocks (TB) to SM.

활용되고 있다[1, 2, 3].

한편, 클라우드 환경에서는 여러 응용들이 가상머신 형태로 하드웨어 자원을 공유하면서 실행되므로 GPGPU 역시 다양한 워크로드들을 동시에 실행하면서 발생하는 이종 작업 간 스케줄링 문제가 발생한다[4, 5]. 전통적인 스토리지 I/O의 경우 호스트 OS를 통해 캐싱, 스케줄링 등의 관리가 다중 응용의 특성을 고려해 효율적으로 이루어질 수 있다[6, 7]. 이해 비해 GPGPU는 단일 워크로드의 병렬적 처리에 유리하도록 설계되었고 하드웨어 관리가 호스트 OS의 통제 권한 밖에 있으므로 다양한 워크로드들이 동시에 실행될 때 이를 효율적으로 관리하는 데에 어려움이 있다.

GPGPU 내에는 수천 개의 연산 유닛이 존재하며 각 유닛들은 동시에 서로 다른 스레드를 실행할 수 있다[8]. 일반적으로 GPGPU에서 실행되는 작업은 일련의 스레드들의 집합인 스레드 블록 단위로 스케줄링이 이루어진다. 한편, GPGPU 내의 연산 자원은 SM(Stream Multiprocessor)이라 불리는 일련의 연산 유닛들로 구성되며, 각 스레드 블록을 어떤 SM에 할당할 것인지를 스케줄링 시 결정해야 한다. 큐에 대기 중인 스레드 블록들을 각 SM에 분배할 때에는 자원의 용량을 초과하지 않는 범위에서 전체 SM이 골고루 활용될 수 있도록 스케줄링하는 것이 중요하다. 즉, 스레드 블록 스케줄러의 가장 주요한 목표는 스레드 블록들을 SM에 적절히 잘 분배하여 SM 간 부하 균형을 이루도록 하는 것이다[9].

현재 GPGPU 내의 스레드 블록 스케줄러는 라운드 로빈(Round-Robin) 방식을 가장 널리 사용하고 있다. 라운드 로빈은 CPU 디스패칭, 네트워크 패킷 전송 등 다양한 스케줄링 도메인에서 널리 활용되는 알고리즘이다[10, 11, 12]. 라운드 로빈은 큐에 대기 중인 스레드 블록들을 각 SM에 하나씩 번갈아가며 할당하는 방식으로, 워크로드가 동종일 경우 부하 균형을 기대할 수 있다. 그러나, 이종 워크로드가 동시에 실행되는 경우, 각 스레드 블록의 연산량이 균일하지 않으므로 라운드 로빈이 SM 별 부하의 편향을 유발할 수 있다[13].

본 논문은 다양한 GPGPU 워크로드가 공존하는 클라우드 환경을 위한 새로운 스레드 블록 스케줄링 기법을 제안한다. 제안하는 기법은 기존 기법들처럼 SM 간의 부하 균형을 추구하지만, 스케줄링 방식에 있어 스레드 블록들의 특성을 고려하여 SM에 워크로드를 효율적으로 배치한다. 이를 위해 큐에 대기 중인 스레드 블록들을 작업의 양에 따라 다중 큐에 배치한 후, SM의 잔여 자원을 가장 잘 활용할 수 있는 스레드 블록을 배치하는 방식을 사용한다. 본 논문에서는 스레드 블록 스케줄링 문제를 빈 패킹 문제로 매핑하여 SM 내의 연산 자원에 발생하는 단편화(fragmentation) 문제를 최소화하도록 스케줄링하였다. 다양한 워크로드 부하 환경에서의 시뮬레이션 실험을 통해 제안하는 방식이 라운드 로빈 알고리즘 대비 평균 24.8%의 성능 향상을 나타냄을 보였다.

II. 스레드 블록 스케줄링

본 장에서는 제안하는 스레드 블록 스케줄링 알고리즘을 자세히 설명한다. SM에 스레드 블록을 할당하기 위해 GPGPU는 스레드 블록들을 도착 시간 순서에 따라 큐에 넣어 관리한다. 스케줄링이 필요할 때 스케줄러는 큐의 제일 앞단에 있는 스레드 블록을 꺼내어 이를 배치할 SM을 선택한다. 상용 GPGPU 시스템에서 가장 널리 사용되고 있는 라운드 로빈 기법은 대기 중인 스레드 블록들을 순차적으로 SM들에 배치하며 그림 1은 라운드 로빈이 어떻게 큐에 있는 스레드 블록을 SM에 배치하는지를 보여주고 있다. 라운드 로빈은 구현이 간단하지만 SM 간의 부하균형 측면에서는 워크로드의 종류가 다양할 때

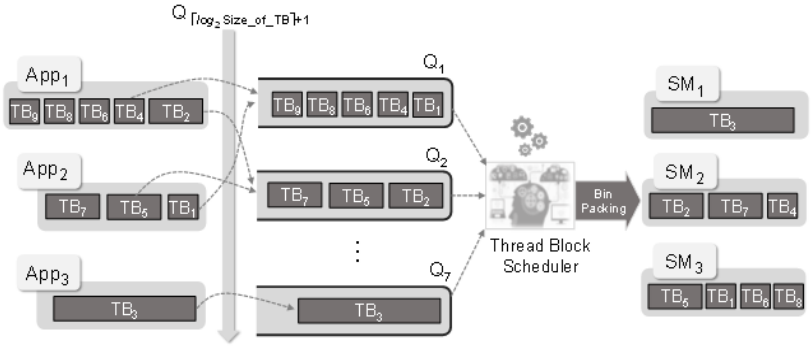

그림 2. 제안하는 기법이 SM에 스레드 블록을 할당하는 예시
Fig. 2. An example of allocating Thread Blocks (TB) to SM in the proposed policy.

그다지 효과적이지 못하다. 이는 스레드 블록의 작업량이 균일하지 않은 환경에서 SM의 잔여 부하와 큐에 대기 중인 스레드 블록의 작업량을 고려하지 않고 하나씩 기계적으로 배치하기 때문이다.

이러한 상황에 대처하기 위해 BFA(Breadth First Allocation)와 DFA(Depth First Allocation) 방식이 제안되었다[13]. BFA는 가장 부하가 적은 SM에 우선적으로 스레드 블록을 배치하는 정책으로 어느 정도의 부하 균형을 이룰 수 있지만, 시간이 흐름에 따라 모든 SM의 가용 자원이 점점 줄어 연산량이 많은 스레드 블록은 아무 곳에도 배치할 수 없는 상황에 이르게 된다. 연산량이 많아 어느 SM에도 할당될 수 없는 스레드 블록이 큐의 제일 앞에 위치한 경우 오랜 시간을 대기해야 해당 스레드 블록을 스케줄링할 수 있는 SM이 생기므로 큐에 있는 작업 전체의 대기시간이 길어지게 된다. DFA는 BFA와는 반대로 큐의 제일 앞에 대기 중인 스레드 블록을 수용 가능한 SM 중 부하가 가장 높은 SM에 할당하는 정책이다. 부하 균형과 반대되는 부하 편향의 방식은 일반적인 상식과 배치되는 측면이 있으나 다양한 분야에서 그 효용성이 입증되고 있다[14]. 이는 다시 말해 현재 작업을 할당받은 SM을 최대한 채우고 난 후 비어 있는 SM을 채워나가려는 정책으로 큰 스레드 블록을 할당하기 위한 잔여 SM을 비워놓는 전략이라 할 수 있다. GPGPU 내의 작업 부하가 높은 상황에서는 DFA가 자원의 이용률을 높일 수 있는 장점이 있다. 그러나, DFA는 워크로드의 작업 부하가 낮은 환경에서 SM 간 부하 균형이 이루어지지 않아 유휴 자원이 발생하고 그 효율성이 크게 떨어지는 것으로 알려져 있다[13].

라운드 로빈, BFA, DFA 등 기존 알고리즘들의 근본적인 문제점은 매 스케줄링 시점마다 큐의 제일 앞에 위치한 스레드 블록 하나를 스케줄링 대상으로 고려하고 SM에 배치한다는 점이다. 즉, 큐의 뒤쪽에 어떤 스레드 블록들이 대기 중인지를 모르는 상태에서 스케줄링이 이루어지기 때문에 현재의 결정이 이후의 스케줄링에 비효율적인 영향을 초래할 수 있다는 뜻이다. 이미 큐에 진입해 있는 스레드 블록의 종류나 수가 알려져 있음에도 이를 고려하지 않아 SM 간의 부하균형을 어렵게 하고 전체적인 스케줄링의 효율성을 저해하게 되는 것이다.

이러한 문제를 해결하기 위해 본 논문에서는 스레드 블록을 큐에서 하나씩 스케줄링하는 대신 일련의 스레드 블록들을 한꺼번에 스케줄링하는 플랜을 미리 세우는 방안을 도입한다. 이를 위해 본 논문에서는 스레드 블록의 대기 큐를 연산요구량(이후 스레드 블록의 크기라고 지칭함)에 따라 다중 큐로 관리하는 방안을 제안하고, 크기 기반의 SM 배치를 통해 SM 내부의 유휴자원(이후 잔여 공간이라 지칭함)을 최소화하는 배치를 수행한다. 그림 2에서 보는 것처럼 제안하는 기법은 스레드 블록을 그 크기에 따라 분류하고 이를 관리할 큐를 두어 크기 N인 스레드 블록이 ($\lceil \log_2 N \rceil + 1$)번째 큐에 진입하도록 한다. 예를 들어 스레드 블록의 최대 크기가 64인 경우 7개의 큐를 두게 된다.

실제 상황에서는 제안하는 스레드 블록 스케줄링이 빈 패킹 문제와 매우 유사하다. 모든 SM이 비어있는 초기상태에서 제안하는 기법은 스레드 블록 하나씩을 각 SM에 번갈아 라운드 로빈 방식으로 배치하며 이때 배치할 스레드 블록은 다중 큐 중 해당 SM이 수용가능한 가장 큰 크기의 큐에서 선택한다. 이를 통해 SM의 잔여 공간을 가장 잘 채울 수 있는 크기의 스레드 블록을 배치하게 되

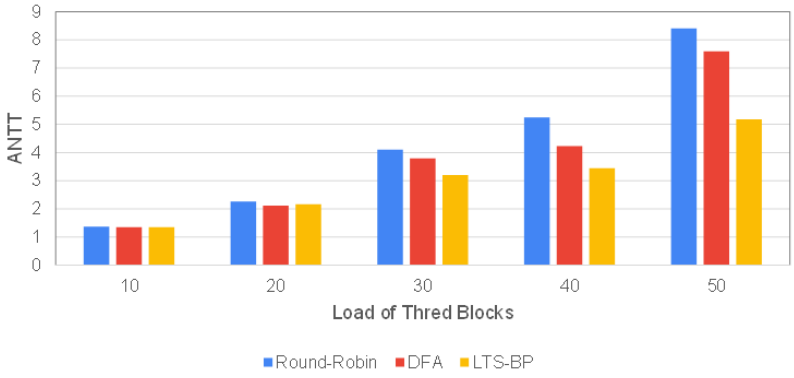

그림 3. 제안하는 기법과 기존 기법 간의 실행 시간 비교
Fig. 3. Comparison of the proposed policy and existing policies with respect to the execution time.

어 SM 간의 부하균형을 이룰 수 있게 된다. 시간이 흐름에 따라 작업이 끝나는 스레드 블록이 발생하면 SM 내의 잔여 공간이 추가로 생기게 되며 제안하는 기법은 해당 SM의 잔여 공간을 제일 잘 채울 수 있는 큐에서 스레드 블록을 추출하여 이를 해당 SM에 배치한다.

III. 성능 평가

스레드 블록 스케줄링 알고리즘의 성능 평가를 위해 본 논문에서는 다양한 워크로드 부하 환경에서 시뮬레이션 실험을 수행하였다. 본 논문에서 제안한 알고리즘인 LTS-BP(Load-Based Time Sliding Bin Packing)의 성능을 Round-Robin, DFA와 비교했으며, 워크로드 부하는 GPGPU 내 모든 SM 자원이 최대로 사용되는 부하를 100%라고 했을 때, 그 10%부터 50%까지 변화시키며 실험을 하였다. 성능 척도로는 다른 연구에서 널리 사용되고 있는 ANTT(Average Normalized Turnaround Time)를 사용하였다[5, 13]. ANTT는 스레드 블록 하나만 단독 실행될 때의 실행시간 대비 해당 스케줄링 알고리즘에 의해 소요된 상대적인 시간을 뜻한다. 예를 들어 ANTT가 3인 경우 해당 워크로드가 단독으로 실행될 때보다 3배의 시간이 소요되었다는 것을 의미한다.

그림 3은 세 알고리즘의 ANTT를 비교해서 보여주고 있다. 그림에서 보는 것처럼 부하가 10%인 상황에서는 알고리즘 간 성능 차이가 크지 않았으며, 이는 스케줄링 알고리즘에 따른 차별화된 결과를 나타내기에 부하가 충분하지 않기 때문이다. 워크로드의 부하가 늘어남에 따라 세 알고리즘의 성능 격차는 명확히 나타났으며, LTS-BP는 라운드 로빈 대비 평균 24.8%, DFA 대비 평균 15.9% 우수한 성능을 부하 구간 20-50%에서 나타내었다. 이는 기존 스케줄링 알고리즘이 부하가 높은 상황에서 비효율적임을 드러낸 결과라고 할 수 있다.

그림 4는 세 알고리즘의 SM 이용률을 보여주고 있다. SM 이용률이란 전체 시간 중 SM이 작업을 수행한 시간의 비율을 나타내며, 알고리즘이 SM을 얼마나 유휴상태로 두지 않고 지속적으로 작업을 수행시켰는지를 나타내어 높을수록 우수한 의미를 가지는 성능 척도이다. 그림에서 보는 것처럼 LTS-BP는 라운드 로빈에 비해 부하가 10%를 넘는 상황에서 한결같이 우수한 SM 이용률을 나타낸 것을 확인할 수 있다. 특히 이 구간에서 LTS-BP의 SM 이용률 개선 효과는 평균 37.9%, 최대 52.9%에 이르렀다. 라운드 로빈의 SM 이용률이 낮은 것은 SM 간의 부하 분배에 실패했기 때문으로 이는 작업의 크기나 SM 내의 유휴 자원을 스케줄러가 고려하지 않았기 때문에 초래된 결과이다.

한편, 그림 4에서 보는 것처럼 DFA의 SM 이용률은 부하가 높은 경우 LTS-BP와 어느 정도의 격차를 나타내었으나 그 차이는 ANTT에 비해서는 크지 않은 것으로 나타났다. 이는 부하가 높은 환경에서 DFA가 SM 이용률을 높이는 것에 최적화된 알고리즘이기 때문에 나타난 결과이다. 하지만, LTS-BP는 다양한 크기의 스레드 블록들을 SM들의 부하 상태를 고려해 적절히 배치함으로서 DFA보다 더 좋은 결과를 나타낸 것을 확인할 수 있다.

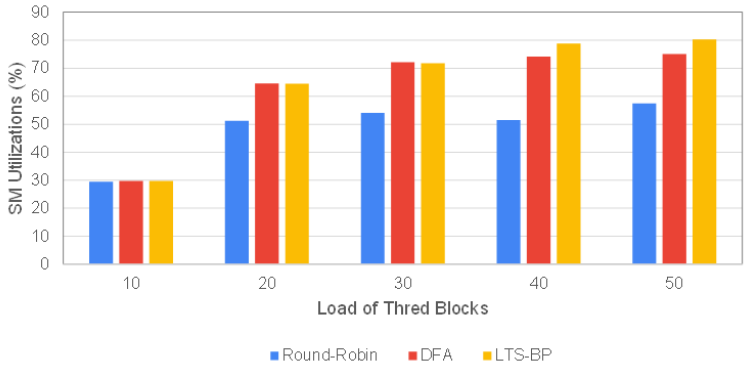

그림 4. 제안하는 기법과 기존 기법 간의 SM 이용률 비교
Fig. 4. Comparison of the proposed policy and existing policies with respect to SM utilization.

IV. 결 론

본 논문에서는 다양한 GPGPU 워크로드가 공존하는 환경에서 큐에 대기 중인 스레드 블록들을 그 작업량에 근거해 각 SM에 효율적으로 배치하는 스레드 블록 스케줄링 알고리즘을 제안하였다. 제안한 알고리즘은 각 SM의 내부 단편화를 최소화하기 위해 스레드 블록들을 작업량에 근거해 다중큐로 관리하고 각 SM의 잔여 자원을 가장 잘 채울 수 있는 큐에서 스레드 블록을 선택하여 모든 SM의 자원 이용률을 극대화시켰고, 이는 궁극적으로 SM 간의 부하균형을 이루는 효과를 얻을 수 있었다. 다양한 부하 환경에서의 시뮬레이션 실험을 통해 제안한 스케줄링 알고리즘이 스레드 블록의 실행시간을 라운드 로빈과 DFA 알고리즘 대비 각각 24.8%와 15.9% 개선할 수 있음을 확인하였다.

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] 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

[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, 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] E. Lee, J. Whang, U. Oh, K. Koh, and H. Bahn, "Popular channel concentration schemes for efficient channel navigation in internet protocol televisions," IEEE Trans. Consumer Electronics, vol. 55, no. 4, pp. 1945-1949, 2009.
DOI: https://doi.org/10.1109/TCE.2009.5373754

저 자 소 개

박 소 연(비회원)

- 2021년 2월 : 이화여자대학교 컴퓨터공학과 학사
- 2021년 3월 ~ : 이화여자대학교 컴퓨터공학과 대학원생

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 MSIT/IITP (2018-0-00549, extremely scalable order preserving OS for manycore and non-volatile memory).