

GPU와 PIM 구조의 메모리 주소 매핑 방식에 따른 행렬-벡터곱 연산 성능 분석

(Analysis of GEMV Kernel Computations by Address Mapping Approaches Based on GPU and PIM Architectures)

신 지원* 구 건 재**
(Jiwon Shin) (Gunjae Koo)

요약 Processing-in-Memory(PIM)은 프로세서와 오프칩(off-chip) 메모리 간의 대역폭 한계를 극복하기 위해 제안된 구조로, 메모리 내부의 높은 대역폭과 병렬 처리 능력을 활용해 데이터 연산 성능을 향상시킬 수 있다. 이러한 특성 덕분에 PIM을 기존 GPU와 같은 고성능 프로세서에 결합할 경우 전체 시스템의 성능 향상을 기대할 수 있다. 그러나, PIM과 GPU의 구조적 특성 차이에 따라 두 구조간 메모리 주소 매핑 방식에는 차이가 존재한다. 이로 인해 PIM의 주소 매핑 방식을 GPU에 그대로 적용할 경우 성능 저하가 발생할 가능성이 있다. 본 논문에서는 메모리 집약적 워크로드인 행렬-벡터곱(GEMV) 연산을 통해 PIM의 주소 매핑 방식이 GPU 성능에 미치는 영향을 분석하였다. 실험 결과, PIM의 주소 매핑 방식을 GPU에 적용한 경우 성능과 대역폭이 모두 감소하는 현상이 나타났으며, 주소 매핑 방식의 차이가 GPU-PIM 구조에서의 성능 저하의 원인을 확인할 수 있었다.

키워드: GPU, processing-in-memory (PIM), 메모리 주소 매핑, 행렬-벡터곱

Abstract Processing-in-Memory (PIM) is an architectural approach that can overcome bandwidth limitations between host processors and off-chip memory. PIM can improve data computation performance by exploiting high internal bandwidth and parallel computations within a memory module. Therefore, it is expected that PIMs can be paired with high-performance processors such as GPUs to achieve overall performance improvements. However, due to differences in memory address mapping schemes between PIM and GPU architectures, applying PIM's address mapping method directly to GPUs may result in a decrease in overall performance. In this paper, we analyze the performance impact of PIM's address mapping schemes on GPU using memory-intensive general matrix-vector product (GEMV) kernels. Our evaluation results exhibit that PIM's address mapping schemes degrade performance and memory bandwidth on GPUs, indicating that differences in mapping schemes could potentially cause performance degradation in GPU-PIM architectures.

Keywords: GPU, processing-in-memory, address mapping, general matrix-vector product (GEMV)

- 이 성과는 2025년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2021R1C1C1012172)
· 이 논문은 2024 한국컴퓨터종합학술대회에서 'GPU와 PIM 구조의 메모리 주소 매핑 방식에 따른 GEMV 커널 성능 분석'의 제목으로 발표된 논문을 확장한 것임

* 비회원 : 고려대학교 컴퓨터학과 학생
ddr2671@korea.ac.kr

** 정회원 : 고려대학교 컴퓨터학과 교수(Korea Univ.)
gunjaekoo@korea.ac.kr
(Corresponding author)

논문접수 : 2024년 10월 16일
(Received 16 October 2024)
논문수정 : 2025년 4월 20일
(Revised 20 April 2025)
심사완료 : 2025년 4월 24일
(Accepted 24 April 2025)

Copyright©2025 한국정보과학회 : 개인 목적이거나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지 제52권 제6호(2025. 6)

1. 서론

그래픽스 처리 장치(Graphics Processing Unit, GPU)는 현재 그래픽스 작업 뿐만 아니라 대규모 병렬처리를 요구하는 데이터 분석, 신경망 처리, 언어 모델 처리 등 주요 애플리케이션 구동에 핵심적으로 활용되고 있다. GPU는 수천에서 수만 개의 스레드(thread)를 다수의 연산 코어를 통해 병렬로 처리함으로써 데이터 연산 성능을 크게 향상시킨다. 그러나 현재 GPU는 오프칩(off-chip) 메모리에 접근할 때 발생하는 높은 지연 시간과 메모리 채널 대역폭의 한계로 인해, 메모리 시스템이 병목현상을 일으켜 성능에 제한을 받고 있다[1]. 최근에는 이러한 메모리 구성 요소의 성능 한계로 인한 전체 시스템 성능 저하를 극복하기 위한 방안으로 Processing-in-Memory (PIM) 기술이 주목받고 있다[2]. PIM은 메모리 내부에 연산 유닛(processing element, PE)을 배치하여 메모리 모듈 내부에서 데이터를 처리함으로써 높은 메모리 내부 대역폭을 활용할 수 있다는 장점을 가진다. 이러한 특성 덕분에 특히 메모리 집약적 (memory-intensive) 연산에서 PIM 구조를 적용한다면 큰 성능 향상을 기대할 수 있다. 최근 머신 러닝 등 다양한 분야에서 메모리 집약적 연산이 증가함에 따라, PIM 기술에 대한 연구 또한 활발히 이루어지고 있다. 실제 연구되어 발표된 PIM 구조로는 HBM-PIM[3]과 Accelerator-in-Memory(AiM)[4,5] 등이 있다.

GPU의 디바이스 메모리에 PIM 기술을 적용한 GPU-PIM 구조는 GPU와 오프칩 메모리 간의 데이터 이동을 줄여, 메모리 집약적이고 정형적인 연산을 수행하는 작업에서 높은 성능 향상을 기대할 수 있다. 기존 GPU-PIM 관련 연구는 연산 스케줄링 기법[6], 데이터 오프로딩 전략[7], PIM 연산 구조 설계[8], 그리고 특정 애플리케이션 최적화[9,10]에 주로 집중되어 왔다. 본 연구는 이러한 선행 연구를 바탕으로, GPU-PIM 통합 구조에서 상대적으로 간과되어 온 메모리 주소 매핑 방식이 성능에 미치는 영향을 실험적으로 분석한다. GPU와 PIM은 구조적 특성상 메모리 주소 매핑 방식에서 차이를 보인다. GPU는 채널 수준 병렬성(channel-level parallelism)을 극대화 하기 위해 인접 데이터를 여러 메모리 채널에 분산하여 저장한다. 반면에 PIM은 연산 유닛이 사용하는 데이터가 단일 메모리 모듈 내부에 집중될 수 있도록 뱅크 수준 병렬성(bank-level parallelism)을 보장해야 높은 내부 대역폭을 활용할 수 있다. 본 연구에서는 PIM의 주소 매핑 방식을 GPU에 적용한 후, 행렬-벡터곱(GEMV) 연산의 성능 변화를 비교 분석하여 향후 GPU-PIM 구조에 미칠 수 있는 잠재적 영향을 탐구한다.

본 논문의 구성은 다음과 같다. 먼저 GPU와 PIM의

구조와 특성에 대해 소개한 후 다음 장에서 자세한 주소 매핑 방식을 설명한다. 이후 실험 단계에서 실험 구성과 함께 연구 결과를 분석한다. 마지막 장에서 결론과 향후 연구 방향을 이야기하며 끝맺는다.

2. 배경

이 장에서는 GPU와 PIM의 구조와 특성에 대해 설명한다.

2.1 GPU

GPU는 대규모 병렬 처리에 특화된 구조로, 수천 개의 작은 연산 유닛을 이용해 데이터를 동시에 처리한다. 연산 유닛들은 대용량의 데이터를 처리하기 위해 수많은 메모리 요청을 동시에 발생시키는데, 이때 채널 수준 병렬성을 활용하여 메모리 대역폭을 극대화한다.

GPU의 메모리 시스템은 여러 개의 독립적인 메모리 채널로 구성되어 있으며, 각 메모리 채널은 데이터 버스와 제어 회로를 갖추고 있어 동시에 데이터를 전송하고 처리할 수 있다. 이러한 구조 덕분에 GPU는 여러 채널로부터 동시에 데이터를 받아와 처리함으로써 전체 메모리 대역폭을 효과적으로 활용할 수 있다.

2.2 PIM

PIM은 기존 컴퓨터 아키텍처에서 발생하는 메모리 병목 문제를 해결하기 위해 제안된 새로운 기술이다. 기존 컴퓨터 구조에서는 프로세서가 메모리에서 데이터를 가져오는 과정에서 제한된 메모리 대역폭과 데이터 전송 속도가 병목으로 작용한다. PIM은 메모리 내부에 연산 유닛을 포함하여 메모리 모듈 외부로의 데이터 이동을 최소화하고 메모리 모듈 내에서 직접 연산을 수행한다.

GPU와 달리, PIM은 뱅크 수준 병렬성이 보장되어야 높은 내부 대역폭을 활용할 수 있다. 현재 개발되어 발표된 PIM에서는 각 메모리 뱅크에 연산 유닛이 위치하여 해당 뱅크의 데이터를 내부에서 직접 처리한다. 뱅크 수준 병렬성이 확보되면 뱅크 간 데이터 이동이 줄어들고, 각 뱅크가 독립적으로 데이터를 처리할 수 있어 메모리 내부 대역폭 활용을 극대화할 수 있다.

3. GPU와 PIM의 주소 매핑 방식

이 장에서는 GPU와 대표적인 PIM 구조인 HBM-PIM과 AiM의 실제 주소 매핑 방식에 관해 설명한다. 연산에 사용된 행렬은 모두 열 우선 배열로 표현하였으며, 행렬의 각 요소의 크기는 트랜잭션 (transaction) 크기인 32 바이트로 설정하였다.

3.1 GPU의 주소 매핑 방식

그림 1은 NVIDIA RTX2060 GPU의 주소 매핑 방식을 나타낸 그림이다[11]. RTX2060 GPU는 오프칩 메모리로

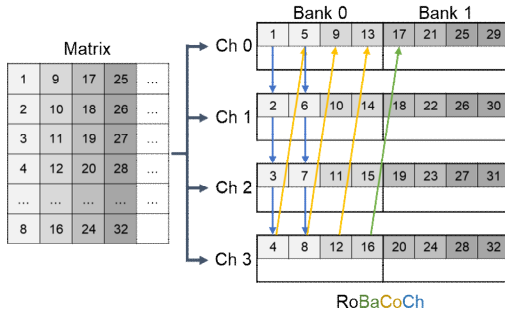


그림 1 GPU의 주소 매핑 방식

Fig. 1 Address mapping scheme of GPU

GDDR6을 사용하며, 주소 매핑 방식으로 RoRaBaCoCh (행-랭크-뱅크-열-채널)을 채택하고 있다. 여기서 Ro는 행(row), Ra는 랭크(rank), Ba는 뱅크(bank), Co는 열(column), Ch는 채널(channel)을 의미한다. 이 매핑 방식에서는 하위 비트부터 상위 비트 순서로 데이터를 여러 채널에 분산하여 저장한다. 즉, 이 방식에선 모든 채널의 첫 번째 뱅크, 첫 번째 열에 행렬 데이터를 먼저 저장하고, 다음 열로 이동하여 데이터를 이어서 저장한다. 각 채널의 한 뱅크 내 하나의 행이 모두 채워지면, 다음 뱅크로 넘어가 동일한 방식으로 데이터를 저장한다.

위 방식은 독립적인 메모리 채널로 구성된 GPU의 병렬 처리를 극대화하기 위한 설계로, 최신 NVIDIA 아키텍처를 비롯한 현대 GPU에서도 이러한 구조적 특성을 유지하고 있다. 따라서 본 연구에서 분석한 메모리 주소 매핑 특성은 최신 GPU 환경에서도 유효하게 적용할 수 있다.

3.2 HBM-PIM의 주소 매핑 방식

HBM-PIM[3]은 HBM2[12] 기술을 기반으로 한 PIM 아키텍처로, 각 연산 유닛은 두 개의 메모리 뱅크와 연결되어 있다. 각 유닛 내 레지스터 파일(register file) 중 행렬-벡터곱 연산에는 입력 벡터를 저장하는 GRF_A(global register A)와 출력 벡터를 저장하는 GRF_B(global register B)를 사용한다. 각 레지스터 파일은 32 바이트 용량으로, FP16 기준 16개의 데이터를 저장할 수 있는 8개의 레지스터로 구성된다.

HBM-PIM은 RaRoBaCoCh(랭크-행-열-뱅크-채널)의 주소 매핑 방식을 사용하고 있어, 겉보기에는 GPU 처럼 채널 수준 병렬성을 활용하는 방식으로 보인다[3]. 그러나 이 할당 방식으로 행렬-벡터곱 연산을 수행할 때는 데이터가 채널별로 분산되어 PIM 내부의 높은 데이터 대역폭을 효율적으로 활용하지 못하는 문제가 발생할 수 있다. 이 문제를 해결하기 위해 HBM-PIM은 소프트웨어 스택에서 가중치 행렬의 데이터 레이아웃을 재구성한다. 이 과정에서 연산의 효율성을 위해 행렬을

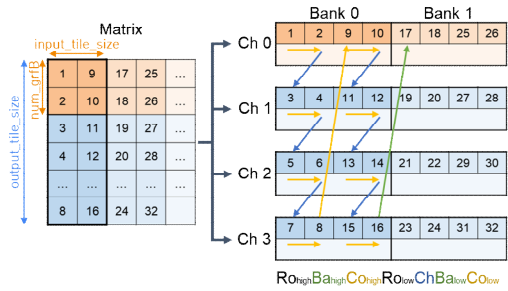


그림 2 HBM-PIM의 주소 매핑 방식

Fig. 2 Address mapping scheme of HBM-PIM

타일링(tiling)하여 각 뱅크에 데이터를 분산 저장한다.

그림 2는 HBM-PIM의 가중치 행렬 주소 매핑 방식을 시각적으로 표현한 그림이다. 타일링을 적용하면 최종적인 행렬 매핑 방식은 하드웨어의 구성과 행렬의 크기에 따라 달라진다. 위 그림에서는 4개의 채널, 채널당 2개의 뱅크를 가진 DRAM에서 GRF_A와 GRF_B의 레지스터 수는 각 2개씩 있다고 가정한다. 가중치 행렬의 매핑 방식은 $Ro_{high}Ba_{high}Co_{high}Ro_{low}ChBa_{low}Co_{low}$ 의 순서로 이루어진다. 여기서 Co_{low} 는 GRF_B 레지스터 수에 대한 로그 값, Ba_{low} 는 뱅크 수에 대한 로그 값에서 1을 뺀 값, Ro_{low} 는 행렬의 열 길이를 외부 타일 길이로 나눈 값의 로그로 정의된다. 열 우선 배열을 가정하였기에 Ro_{low} 연산에서는 행렬의 열 길이를 사용하며, 행 우선 배열에서는 행렬의 행 길이를 사용한다.

소프트웨어 스택의 동작은 다음과 같다. 먼저 내부 타일 길이는 GRF_A의 레지스터 수로, 외부 타일 길이는 GRF_B의 레지스터 수와 연산 유닛의 총 개수를 곱한 값으로 설정한다. 이후 행렬의 각 열을 GRF_B 레지스터 수만큼 짝수 뱅크의 열에 연속적으로 저장한다. 이 작업을 모든 유닛의 짝수 뱅크에 대해 반복하면 외부 타일의 길이만큼 데이터가 저장된다. 그런 다음 DRAM 내에서 행렬의 크기에 따라 뱅크의 다음 행, 또는 다음 열로 이동하여 동일한 작업을 반복한다. 짝수 뱅크의 데이터 저장에 완료되면 동일한 작업을 홀수 뱅크에 대해서도 수행한다.

3.3 AiM의 주소 매핑 방식

AiM[4,5]은 각 뱅크가 하나의 연산 유닛과 연결된 구조를 가진다. 이 구조는 GDDR6를 기반으로 설계되었으며, RTX 2060 GPU와 동일한 RoRaBaCoCh (행-랭크-뱅크-열-채널)의 주소 매핑 방식을 따른다. 그러나 HBM-PIM과 마찬가지로, 이러한 매핑 방식은 뱅크 수준 병렬성을 제대로 활용하지 못해 성능 저하가 발생할 수 있다. 이를 해결하기 위해서는 소프트웨어의 지원이

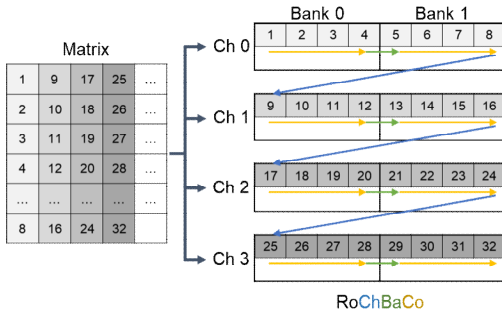


그림 3 AiM의 주소 매핑 방식
Fig. 3 Address mapping scheme of AiM

표 1 GPGPU-Sim 실험 환경
Table 1 Simulation configuration of GPGPU-Sim

GPU configuration	NVIDIA RTX2060
No. of SMs	34
L2 cache size	4 MB
DRAM	GDDR6, 4 GB
No. of DRAM channels	16
No. of DRAM banks / channel	16

필요하다.

그림 3은 AiM 구조에 유리한 데이터 할당을 예시로 보여주며, 이 경우 RoChBaCo(행-채널-뱅크-열)의 주소 매핑을 사용한다[13]. 이 방식에서는 행렬의 각 행이 동일한 채널 내에 집적되도록 데이터를 할당한다. 먼저 한 채널의 모든 뱅크에 한 행의 행렬 데이터를 저장한 후, 다음 채널로 넘어가 같은 작업을 반복한다.

4. 주소 매핑 방식에 따른 GPU 성능 분석

본 연구는 GPU에 다양한 주소 매핑 방식을 적용하여 IPC(instructions per cycle)와 평균 메모리 대역폭을 기준으로 성능을 비교 분석한다. 실험은 GPU 구조 시뮬레이터인 GPGPU-Sim v4.2[11]를 기반으로 진행되었으며, 성능 분석에 사용된 실험 환경은 표 1에 제시되어 있다. HBM-PIM의 레지스터 개수는 원 논문의 설정에 따라 8개로 설정한다. 실험에 사용한 연산은 트랜스포머 모델[14]에서 사용되는 행렬-벡터곱 연산 중 쿼리, 키, 밸류 프로젝션(query, key, value projection) 부분을 대상으로 하였다. 해당 연산은 쿼리, 키, 밸류 연산을 한번에 수행해 트랜스포머 모델의 차원에 해당하는 d_{model} 크기의 벡터를 입력으로 받아 $3 \times d_{model}$ 크기의 벡터를 출력으로 생성한다. 실험에 사용한 d_{model} 크기는 GPT-3[15]의 매개변수를 참조하여 768부터 12288까지 다양한 크기로 설정되었다.

그림 4는 주소 매핑 방식에 따른 d_{model} 크기 별 GPU

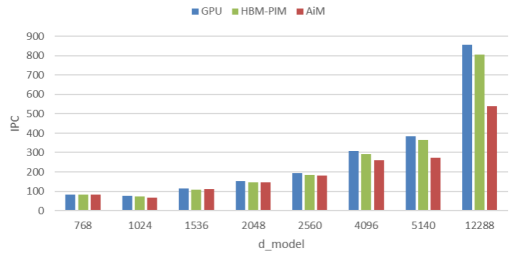


그림 4 주소 매핑 방식에 따른 IPC 성능 비교
Fig. 4 IPC performance under different address mapping schemes

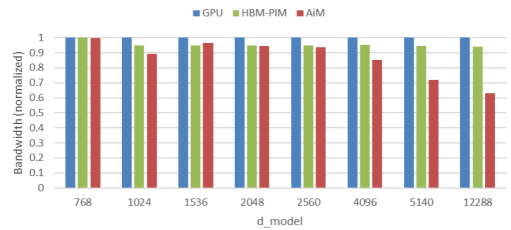


그림 5 주소 매핑 방식에 따른 대역폭 성능 비교
Fig. 5 Memory bandwidth under different address mapping schemes

의 IPC 변화를 보여준다. HBM-PIM과 AiM의 주소 매핑 방식을 적용한 경우, 기존 GPU의 주소 매핑 방식과 비교하여 성능이 각각 평균적으로 4.5%, 13.4% 감소한 것으로 나타났다. 이는 뱅크 수준 병렬성이 보장될 때 높은 성능을 보이는 PIM의 특성이 GPU의 구조에는 적합하지 않음을 나타낸다. 특히 채널 수준 병렬성이 가장 낮은 AiM의 주소 매핑 방식을 따를 때 성능 저하가 가장 두드러지게 나타났다. 다만, d_{model} 이 768일 경우 입력 데이터의 총 크기가 3.37 MB로 GPU의 L2 캐시 크기보다 작아 성능의 차이가 거의 나타나지 않았다.

그림 5는 주소 매핑 방식에 따른 d_{model} 크기 별 GPU의 대역폭을 GPU 주소 매핑 방식을 기준으로 정규화한 결과를 보여준다. 행렬-벡터곱 연산은 메모리 집약적인 특성상 대역폭과 성능이 밀접하게 연관되어, 대역폭 변화와 IPC 성능이 비례한 경향을 보였다. HBM-PIM의 주소 매핑 방식을 적용한 경우 대역폭이 기존보다 5.2% 감소하였고, AiM의 주소 할당 방식에서는 15.1% 감소하였다. 앞선 IPC 결과와 마찬가지로 d_{model} 이 768일 때는 데이터 크기가 작아 대역폭의 차이가 나타나지 않았다.

5. 결론 및 향후 연구

본 논문에서는 행렬-벡터곱 연산을 활용하여 PIM 구

조의 주소 매핑 방식이 GPU 성능에 미치는 영향을 실험적으로 분석하였다. 연구 결과, PIM 방식의 주소 매핑을 GPU에 적용하였을 때 성능과 대역폭이 모두 감소하는 경향을 보였다. 본 연구는 향후 GPU-PIM 아키텍처의 최적화 과정에서, 주소 매핑 방식의 차이에 따른 성능 저하의 가능성을 고려해야 함을 시사한다. 향후 연구에서는 이러한 성능 저하를 최소화하기 위해 GPU-PIM 아키텍처에 최적화된 데이터 매핑 방안을 탐구할 예정이다.

References

- [1] N. Vijaykumar, et al. "A case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling flexible data compression with assist warps," *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, Portland, OR, USA, pp. 41-53, 2015.
- [2] Elliott, Duncan, et al. "Computational RAM: Implementing Processors in Memory." *IEEE Design & Test of Computers*, Vol. 16, No. 1, pp. 32-41, 1999.
- [3] Lee, Sukhan, et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product." *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021.
- [4] He, Mingxuan, et al. "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning." *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020.
- [5] D. Kwon et al., "A 1nm 1.25V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application." *IEEE Journal of Solid-State Circuits*, Vol. 58, No. 1, pp. 291-302, Jan. 2023.
- [6] Pattnaik, Ashutosh, et al. "Scheduling techniques for GPU architectures with processing-in-memory capabilities." *Proc. of the 2016 International Conference on Parallel Architectures and Compilation*, pp. 31-44, 2016.
- [7] Hsieh, Kevin, et al. "Transparent offloading and mapping (TOM) enabling programmer-transparent near-data processing in GPU systems." *ACM SIGARCH Computer Architecture News* 44.3, pp. 204-216, 2016.
- [8] Zhang, Dongping, et al. "TOP-PIM: Throughput-oriented programmable processing in memory." *Proc. of the 23rd international symposium on High-performance parallel and distributed computing*, pp. 85-98, 2014.
- [9] Choi, Jungwoo, Hyuk-Jae Lee, and Chae Eun Rhee. "ADC-PIM: Accelerating Convolution on the GPU via In-Memory Approximate Data Comparison." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12.2, pp. 458-471, 2022.
- [10] Wang, Tianyu, Zhaoyan Shen, and Zili Shao. "CNN Acceleration with Joint Optimization of Practical PIM and GPU on Embedded Devices." *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, pp. 377-384, 2022.
- [11] Khairy, Mahmoud, et al. "Accel-Sim: An extensible simulation framework for validated GPU modeling." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.
- [12] K. Sohn et al. "A 1.2 V 20 nm 307 GB/s HBM DRAM With At-Speed Wafer-Level IO Test Scheme and Adaptive Refresh Considering Temperature Distribution." *IEEE Journal of Solid-State Circuits*, Vol. 52, No. 1, pp. 250-260, Jan. 2017.
- [13] Kwon, Yongkee, et al. "Memory-Centric Computing with SK Hynix's Domain-Specific Memory." *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 2023.
- [14] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*, 2017.
- [15] Brown, Tom, et al. "Language models are few-shot learners." *Proc. of the 34th International Conference on Neural Information Processing Systems (NIPS '20)*, pp. 1877-1901, 2020.

신 지 원

2023~2025 고려대학교 컴퓨터학과 (석사)
2016~2021 중앙대학교 전자전기공학부 (학사). 관심분야는 GPU 아키텍처, PIM (Processing-in-Memory)



구 건 제

2020~현재 고려대학교 컴퓨터학과 부교수
2018~2020 홍익대학교 전자전기공학부 조교수. 2003~2011 LG전자 책임연구원
2011~2018 University of Southern California EE (박사). 2001~2003 서울대학교 전기컴퓨터공학부 (석사). 1997~2001 서울대학교 전기공학부 (학사). 관심분야는 GPU 구조, 가속기 구조, PIM, 스토리지 시스템

