GPU와 PIM 구조의 메모리 주소 매핑 방식에 따른 GEMV 커널 성능 분석

신지원[O] 구건재

고려대학교 컴퓨터학과

ddr2671@korea.ac.kr, gunjaekoo@korea.ac.kr

Performance Analysis of GEMV Kernels by GPU and PIM Memory Address Mapping Approaches

Jiwon Shin[O] Gunjae Koo

Department of Computer Science and Engineering, Korea University

요 약

Processing-in-Memory(PIM)은 프로세서와 오프칩(off-chip) 메모리 사이의 대역폭 한계를 극복하기 위한 구조로서 제안되었으며 메모리 내부의 높은 대역폭과 병렬성을 이용하여 정규적인 데이터 연산 성능을 높일 수 있는 구조로 여겨지고 있다. 그렇기 때문에, PIM을 기존의 GPU와 같은 고성능 프로세서에 연결하여 전체적인 성능 향상을 얻을 것으로 기대된다. 그렇지만, PIM과 GPU 구조의 특성에 따른 메모리 주소 매핑 방식의 차이로 인해 메모리 내부의 병렬성을 우선하는 PIM의 주소 매핑 방식을 GPU에 그대로 적용할 경우 전체적인 성능이 하락할 수 있다. 본 논문에서는 메모리 집약적인 행렬-벡터곱(GEMV) 커널 연산을 통해 PIM에 적용된 주소 매핑 방식에 따른 GPU 성능을 분석한다. 분석 결과 PIM의 주소 매핑 방식이 적용된 GPU에서 성능과 대역폭이 하락하여 해당 매핑 방식이 GPU-PIM 구조에서 잠재적 성능 하락의 원인임을 밝혔다.

1. 서 론[1]

그래픽 처리 장치(Graphic Processing Unit, GPU)는 많은 수의 단순 연산 코어를 사용하여 수천에서 수만 개의 쓰레드(threads)를 병렬적으로 처리하여 데이터 연산의 성능을 획기적으로 높일 수 있다. 그렇지만, 오프칩(off-chip) 메모리를 통해 데이터에 접근하는 데에 높은 지연시간이 요구되고, 메모리 채널의 대역폭이 쓰레드 처리 용량 대비 한계가 있어서 메모리 시스템의 성능 한계에 따른 병목현상이 발생한다. 이러한 메모리 성능 한계에 따른 전체 시스템 성능 저하를 극복하기 위해 Processing-in-Memory(PIM) 구조 방식이 제안되었다. PIM은 메모리 내부에 연산 유닛(processing unit, PE)을 위치시켜서 메모리 내부의 높은 데이터 대역폭을 활용할 수 있는 구조다. PIM 구조를 적용할 경우 메모리 집약적인 커널에서 성능에 크게 이득을 볼 수 있다. 최근 머신 러닝 등 다양한 분야에서 메모리 집약적인 연산이 증가하면서 PIM에 관한 연구가 활발히 진행되고 있다. 현재 실제 개발된 PIM에는 HBM-PIM[1]과 Accelerator-in-Memory(AiM)[2]이 있다.

GPU의 디바이스 메모리에 PIM 구조를 적용한 GPU-PIM 구조를 구현한다면 GPU와 오프칩 메모리 간의 데이터 이동이 줄기 때문에 높은 성능향상이 있을 것으로 기대된다. 하지만 GPU와 PIM은 각 구조의 특성에 의해 메모리 주소 매핑 방식에 차이가 존재한다. GPU의 메모리는 여러 개의 독립적인 채널로 구성되어 있으며 각 채널은 여러 메모리 파티션에 동시 접근이 가능하도록 병렬적으로 동작한다. 즉, GPU에서는 메모리 대역폭을 향상시키기 위해 채널 수준 병렬성(channel-level parallelism)을 이용한다. PIM에서는 한 PE에서 사용하는 데이터가 다른 메모리 칩에 저장되어 있으면 채널 버스를 통한 데이터 이동이 필요하기에 뱅크 수준 병렬성(bank-level parallelism)을 보장해야 높은 내부 대역폭을 활용할 수 있다. 본 연구에서는 PIM의 주소 매핑 방식을 적용한 GPU에서 행렬-벡터곱(GEMV) 커널의 성능 변화를 비교하여 향후 GPU-PIM 구조에 미칠 잠재적 영향을 분석한다.

* 이 성과는 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임
(NRF-2021R1C1C1012172)

2. 배 경

이 장에서는 GPU와 HBM-PIM, AiM의 주소 매핑 방식에 대해 설명한다. 연산에 사용된 행렬은 모두 열 우선 배열로 표현하였으며, 행렬의 각 요소의 크기는 트랜잭션 크기인 32 바이트로 설정하였다.

2.1. GPU의 기본 주소 매핑 방식

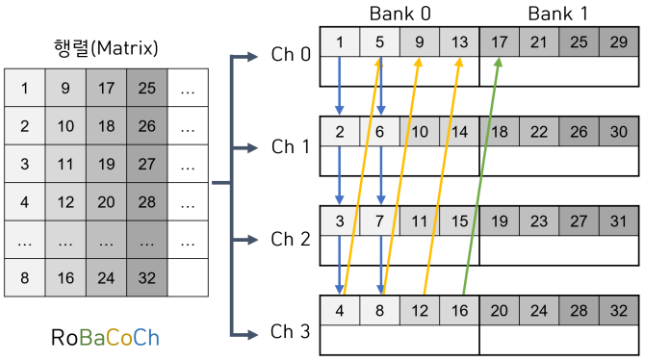

그림 1. GPU의 주소 매핑 방식

그림 1은 NVIDIA RTX2060 GPU의 주소 매핑 방식을 나타낸 그림이다[3]. RTX2060는 GDDR6을 오프칩 메모리로 사용하고 있으며 RoRaBaCoCh(행-랭크-뱅크-열-채널)의 주소 매핑 방식을 가진다. Ro는 행(row), Ra는 랭크(rank), Ba는 뱅크(bank), Co는 열(column), Ch는 채널(channel)로, 하위 비트부터 데이터가 여러 채널에 흩어져서 저장된다. 즉, 모든 채널의 첫 번째 열에 행렬 데이터가 저장되고, 이후 다음 행에 데이터가 저장된다. 그렇게 모든 채널의 한 뱅크, 한 행을 전부 채우면 다음 뱅크로 넘어가 같은 동작을 반복한다.

2.2. HBM-PIM의 주소 매핑 방식

HBM-PIM[1]은 HBM2를 기반으로 설계된 PIM 구조로 하나의 PE당 2개의 뱅크와 연결되어 있다. 각 PE 내 레지스터 파일(register file) 중 GEMV 연산에 사용하는 레지스터 파일은 입력 벡터를 저장하는 GRF_A(global register A)와 출력 벡터를 저장하는 GRF_B(global register B)를 사용한다. 두 레지스터 파일은 각각 32 바이트(16×FP16) 크기의 레지스터 8개로 이루어져 있다. HBM-PIM은 RaRoCoBaCh(랭크-행-열-뱅크-채널)의 주소 매핑 방식을 가진다[1]. 그러나 위 할당 방식으로 GEMV 연산을 처리할 경우 가장 먼저 채널 별로 데이터가 흩어져 내부 대역폭을 활용할 수 없는 문제가 발생한다. 따라서 HBM-PIM은 소프트웨어 스택에서 가중치 행렬의 데이터 레이아웃을 재정렬하며, 이 때 연산의 효율성을 위해 행렬을 타일링(tiling)한 뒤 각 뱅크에 저장한다.

그림 2는 HBM-PIM의 가중치 행렬 주소 매핑 방식을 표현한 그림이다. 타일링으로 인해 매핑 구성이 하드웨어의 구성과 행렬의 크기에 따라 달라지며, 위

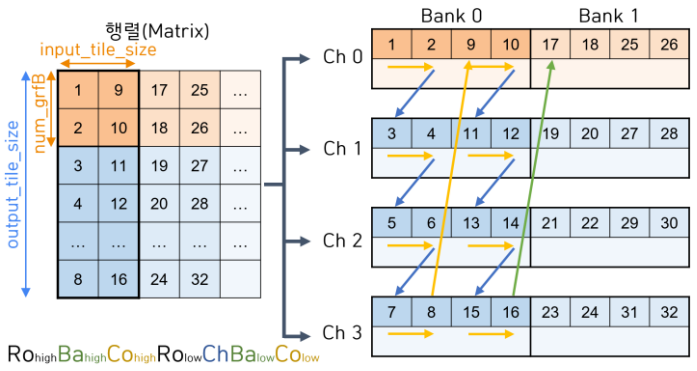

그림 2. HBM-PIM의 가중치 행렬 주소 매핑 방식

그림은 4개의 채널, 채널 당 2개의 뱅크를 가진 DRAM에 GRF_A와 GRF_B의 레지스터 수는 각 2개를 가정한다. 가중치 행렬은 $Ro_{high}Ba_{high}Co_{high}Ro_{low}ChBa_{low}Co_{low}$의 매핑 방식을 가진다. 이 때 Co_{low}는 $\log_2$(GRF_B 레지스터 수), Ba_{low}는 $\log_2$(뱅크 수)-1, Ro_{low}는 $\log_2$(행렬의 열 길이/외부 타일 길이)다. 열 우선 배열로 가정했기 때문에 Ro_{low} 연산에 행렬의 열 길이를 사용하며, 행 우선 배열에선 행렬의 행 길이를 사용한다. 소프트웨어 스택의 동작은 다음과 같다. 먼저 내부 타일 길이를 GRF_A의 레지스터 수, 외부 타일 길이를 GRF_B의 레지스터 수×PE의 총 개수로 설정하여 행렬을 타일링한다. 그 다음 행렬의 한 열을 GRF_B의 레지스터 수만큼 짝수 뱅크의 열에 연속적으로 저장한다. 이를 모든 PE의 짝수 뱅크에 대해 반복하면 외부 타일의 길이 만큼 데이터가 저장된다. 이후 행렬의 크기에 따라 다음 행 또는 다음 열로 넘어간다. 내부 행렬 길이만큼의 열을 전부 저장하면 홀수 뱅크에 대해 동일한 동작을 반복한다.

2.3. AiM의 주소 매핑 방식

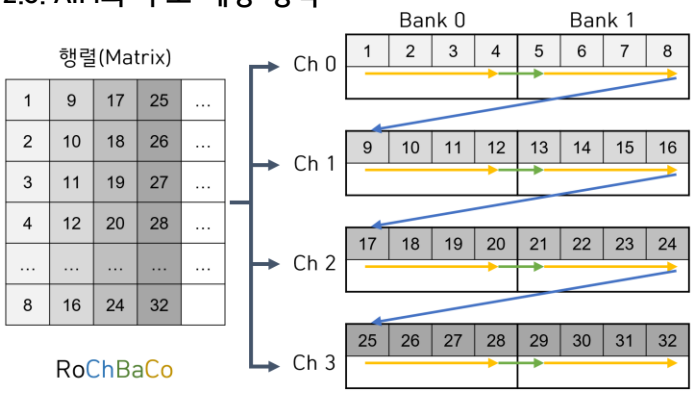

그림 3. AiM의 주소 매핑 방식

AiM[2]은 각 뱅크당 하나의 PE와 연결되어 있으며, GDDR6를 기반으로 설계되어 RTX 2060 GPU과 동일한 RoRaBaCoCh(행-랭크-뱅크-열-채널)의 주소 매핑 방식을 가진다. 하지만 이 경우 HBM-PIM과 마찬가지로 뱅크 수준 병렬성을 갖지 못하여 성능이 떨어지기 때문에 소프트웨어의 지원이 필요하다. 그림3은 AiM 구조에

유리한 데이터 할당을 단순하게 표현한 그림으로 RoCh BaCo(행-채널-뱅크-열)의 주소 매핑 방식을 가진다[4]. 행렬의 한 행이 한 채널에 최대한 모일 수 있도록 한 채널의 모든 뱅크의 한 행에 데이터를 전부 저장한 후 다음 채널로 넘어가는 방식이다.

3. GPU 성능 분석

표 1. GPGPU-Sim 실험 환경

GPU configuration	NVIDIA RTX2060
SM 개수	34
L2 cache size	4 MB
DRAM	GDDR6, 4 GB
DRAM 채널 개수	16
DRAM 채널 당 뱅크 개수	16

본 연구는 GPU에 여러 주소 매핑 방식을 적용하였을 때의 성능을 비교하기 위해 IPC(instructions per cycle)와 평균 대역폭에 대해 분석한다. 실험은 GPU 시뮬레이터인 GPGPU-Sim v4.2[3]을 사용하였으며 성능 분석에 이용한 실험 환경은 표 1과 같다. HBM-PIM[1]의 레지스터 개수는 논문의 구성과 동일한 8개로 설정한다. 커널은 트랜스포머의 GEMV 연산, 그 중에서도 쿼리, 키, 밸류 프로젝션(query, key, value projection)에 해당하는 GEMV 연산 부분을 사용한다. 해당 커널은 쿼리, 키, 밸류 연산을 한 번에 수행하여 트랜스포머 모델의 차원인 $\mathbf{d_{model}}$ 크기의 벡터를 입력으로 하여 $3 \times \mathbf{d_{model}}$ 크기의 벡터를 출력으로 한다. 실험에 사용한 $\mathbf{d_{model}}$ 크기는 GPT-3[5]의 매개변수(parameters)를 따랐으며 그 크기는 768부터 12288까지 다양하다.

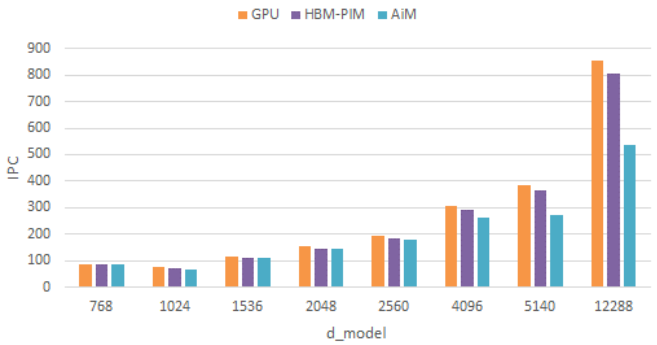

그림 4. 주소 매핑 방식에 따른 GPU 성능 비교

그림 4는 주소 매핑 방식에 따른 $\mathbf{d_{model}}$ 크기 별 GPU의 IPC를 보여준다. HBM-PIM과 AiM의 주소 매핑 방식을 사용한 경우 기존 GPU의 할당 방식을 사용했을 때보다 성능이 평균적으로 각각 4.5%, 13.4% 하락했다. 이는 뱅크 수준 병렬성이 최대한 보장되어야 성능이 좋은 PIM의 특성이 GPU의 구조에는 맞지 않음을 보여준다. 두 가지 PIM 중에는 채널 수준 병렬성이 가장 낮은 AiM의 할당 방식을 따를 때 성능이 가장 떨어졌다. $\mathbf{d_{model}}$이 768일 때에는 입력 데이터의 총 크기가 3.37 MB로 GPU의 L2 캐시 크기보다 작아 성능의 차이를 보이지 않았다.

그림 5. 주소 매핑 방식에 따른 GPU 대역폭 변화

그림 5는 주소 매핑 방식에 따른 $\mathbf{d_{model}}$ 크기 별 GPU의 대역폭을 GPU 주소 매핑 방식을 기준으로 정규화하여 나타낸 차트다. GEMV 커널은 메모리 집약적인 특성에 의해 대역폭과 성능이 거의 비례한 경향을 보였다. HBM-PIM의 주소 매핑 방식은 기존보다 5.2%, AiM의 할당 방식은 15.1% 하락하였다. 마찬가지로 $\mathbf{d_{model}}$이 768일 때에는 데이터 크기가 너무 작아 대역폭의 차이가 없었다.

4. 결론 및 향후 연구

이 논문은 GEMV 커널을 사용하여 PIM 구조 주소 매핑 방식을 적용한 GPU의 성능 변화에 대해 분석하였다. GPU에 PIM 방식의 주소 매핑을 사용할 경우 성능과 대역폭이 하락했다. 이번 연구를 통해 향후 GPU-PIM 아키텍처의 최적화를 위해서는 주소 매핑 방식의 차이에 의한 잠재적 성능 하락을 고려해야함을 밝혔다. 이후에는 GPU-PIM 아키텍처를 위한 데이터 매핑 최적화 방안에 대해 연구할 계획이다.

참 고 문 헌

[1] 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.

[2] He, Mingxuan, et al. "Newton: A DRAM-maker's accelerator-inmemory (AiM) architecture for machine learning." *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020.

[3] 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.

[4] 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.

[5] Brown, Tom, et al. "Language models are few-shot learners." *Advances in neural information processing systems, 33*, 1877-1901, 2020.