The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.11, pp.590~596

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.11.590

Potential of LUT-based PIM for DNNs

Junguk Hong† · Jinho Lee††

ABSTRACT

Processing-in-memory (PIM) is an accelerator that enables data to be processed closer to the stored memory. Due to the nature of PIM specialized in data movement, PIM can be used as an efficient accelerator in transformer-based generative model and recommendation system, which have recently garnered attention. In this paper, we examine the latest research trends of PIM that enhance the recommendtation system and generative language model, which are suitable applications for PIM usage. Additionally, we discuss the research direction of PIM based on previous studies. Lastly, we verify the effectiveness of LUT in PIM systems through experiments.

Keywords : Processing-in-memory, Lookup Table, Transformer-based Generative Model, Recommendation System

DNN 모델 추론을 위한 Lookup Table 기반 PIM의 가능성

홍 정 욱† · 이 진 호††

요 약

Processing-in-memory (PIM)은 데이터가 저장된 메모리 근처에서의 연산을 통해 데이터를 더 가까운 곳에서 연산할 수 있게 하는 가속기이다. 데이터 이동에 특화된 PIM의 특성상 최근 주목받고 있는 생성형 언어모델과 추천 시스템에서는 PIM이 효율적인 가속기로 이용될 수 있다. 본 논문에서는 PIM을 사용하기 적합한 어플리케이션인 추천시스템과 생성형 언어모델을 PIM으로 가속화한 최신 연구 동향을 살핀다. 또한, 선행 연구들을 기반으로 PIM의 연구 방향성에 대해 토의한다. 마지막으로, PIM 시스템에서의 LUT의 효용성을 실험으로 성능을 확인하고 검증한다.

키워드 : 프로세싱 인 메모리, 룩업 테이블, 트랜스포머 기반 생성형 모델, 추천시스템

1. 서 론

Processing-in-memory (PIM)은 데이터가 저장되어 있는 메모리 근처에서 연산을 함으로써 데이터 이동을 더 효율적으로 할 수 있는 가속기이다. AI 모델을 실행시킬 때 가장 흔히 사용되고 있는 가속기인 GPU는 연산 처리 능력이 뛰어나지만 그에 반해 데이터 이동이 병목 현상을 보이는 경우가 있다[1, 2]. 특히 최근에 각광받고 있는 생성형 언어모델이나 추천 시스템같은 경우, 데이터 이동이 주요 병목 지점으로 널리 알려져있다[1-4]. 이런 어플리케이션들같은 경우, 데이터 이동에 특화된 가속기인 PIM을 이용하는 것이 효율적이다.

PIM을 이용한 가속화로 큰 성능 향상을 얻기 위해서는 PIM에 적합한 연산들을 선택해서 PIM에서 처리하게끔 설계하는 것이 중요하다. PIM은 데이터 이동을 더 효율적으로 할 수 있는 가속기로 memory-intensive 연산들을 처리하는데 효과적이다. 반면, 연산에 특화된 GPU같은 가속기에 비해서는 연산 능력이 비교적 낮다. 이런 PIM의 특징을 고려하여 생성형 언어모델이나 추천시스템같은 타겟 어플리케이션들의 구조를 분석하여 PIM에 적합한 연산들을 찾고, 이를 PIM에서 효율적으로 수행하게 하는 것이 중요하다.

Fig. 1은 생성형 언어모델의 주요 연산들을 나타낸 것으로 크게 attention layer와 fully-connected layer로 구성되어 있다. Fully-connectedlayer과 같은 경우, activation 행렬과 weight 행렬간의 general matrix matrix multiplication (GEMM)을 수행한다. GEMM은 대표적인 연산 집적도가 높은 연산으로써 필요한 데이터 양에 비해 연산량이 많으므로 GPU와 같이 연산능력이 좋은 가속기에 적합하다. 반면, attention layer에서는 query 등과 같은 activation 벡터와 key/value 행렬간의 general matrix vector multiplication(GEMV) 연산을 수행한다. GEMM과 다르게 GEMV는 연산 집적도가 낮은 연산으로써 필요한 데이터 양에 비해 연산량이 비교적 적다. 생성형 언어모델에서는 이 GEMV 연산이 흔히 병목 현

※ 이 논문은 대한한국 과학기술정보통신부(MSIT)가 지원하는 정보통신기획평가원(IITP)의 연구비(2024-00395134)에 의하여 연구되었음.
† 준 회 원 : 서울대학교 전기정보공학부 석사과정
†† 비 회 원 : 서울대학교 전기정보공학부 교수
Manuscript Received : October 8, 2024
Accepted : November 14, 2024
* Corresponding Author : Jinho Lee(leejinho@snu.ac.kr)

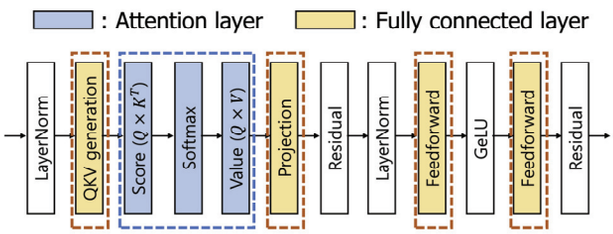

Fig. 1. Operations in Transformer-based Generative Model

상을 보이므로 이는 곧 데이터 이동에 특화된 PIM에 적합하다는 것을 의미한다.

Fig. 2는 추천시스템의 구조로 각 아이템을 하나의 row로 가지는 임베딩 테이블들 여러개와 bottom/top multi-layer perceptron(mlp)로 구성되어 있다. 각 임베딩 테이블에서는 고객들이 동시에 선택한 아이템들을 입력값으로 받고 이들에 해당하는 임베딩 벡터들을 더하는 연산인 embedding lookup을 통해 해당 임베딩 테이블의 대표 임베딩 벡터를 구한다. 임베딩 테이블들의 크기가 크고, 임베딩 벡터들을 더하는 과정인 vector addition은 연산 집적도가 굉장히 낮은 연산으로써 주요 병목현상으로 꼽힌다. 그렇기 때문에 추천시스템을 PIM을 효율적으로 사용할 수 있는 대표적인 어플리케이션으로 볼 수 있다. 하지만 추천시스템의 연산은 전체 실행시간의 70% 이상으로 연산의 가속화가 필요한 상황이다[24].

추천시스템과 생성형 언어모델을 가속화하는 방안으로는 시스템적인 기법, Lookup Table(LUT)를 이용한 소프트웨어적인 기법, GPU/PIM을 동시에 사용하는 heterogeneous system 설계 기법이 있다. 본 논문에서는 PIM을 사용하기 적합한 어플리케이션들에 대한 가속화 기법 선행 연구, PIM에 대한 LUT, GPU-PIM heterogeneous system 측면 연구들을 다루고 이에 따른 PIM 연구의 방향성을 제시한다. 특히, 실제 PIM 시스템 상에서 AI 모델에서 많이 사용되는 low-bit multiply-accumulation(MAC) 연산을 미리 계산한 LUT를 구현해서 성능 향상을 비교하고, 검증한다.

2. PIM에 적합한 어플리케이션에 대한 선행 연구

2.1 추천시스템

1) TensorDIMM

TensorDIMM [5]에서는 효율적인 추천시스템 추론을 위해 mlp 연산은 GPU를 통해 빠르게 계산하고 병목 현상을 보이는 embedding lookup을 PIM과 유사한 near-data processing을 통해 해결한다. 먼저 저장된 임베딩 벡터들 근처에서 embedding lookup을 수행한다. 이후, 기존에 GPU가 임베딩 테이블들의 대표 임베딩 벡터를 구하기 위해 필요한 임베딩 벡터들을 전부 가져오는 대신 메모리 근처에서 reduced된 임베딩 벡터들만을 가져온다. 이러면 병목현상으로 지적되던 GPU와 메모리간 통신 시간을 효과적으로 줄일 수 있다. 이를

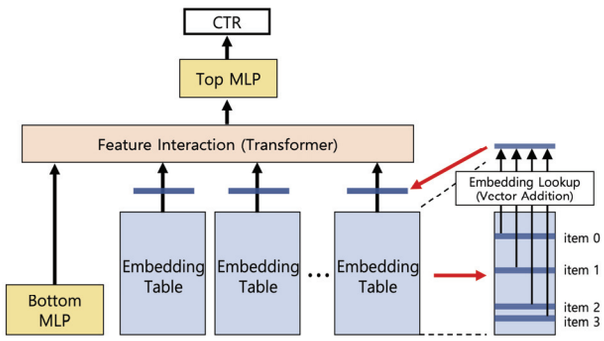

Fig. 2. Structure of Recommendation System

위해 TensorDIMM에서는 column-wise partitioning [6]을 사용해서 각 임베딩 벡터들을 나눠서 서로 다른 near-memory processing 코어들이 독립적으로 처리하게 하고 GPU에서 대표 임베딩 벡터들을 가져오도록 요청하는 명령어를 정의하였다.

2) RecNMP

RecNMP [7]는 [5]의 column-wise partitioning 대신 row-wise partitioning을 사용한다. 저장된 임베딩 테이블 근처에서 local embedding lookup을 수행하고 임베딩 벡터들의 부분합들을 다시 더해서 온전한 대표 임베딩 벡터들을 구하는 global embedding lookup을 수행한다. 또한 시스템적으로도 최적화를 하기 위해 자주 접근되는 임베딩 벡터들을 캐시에 저장하였으며, 임베딩 벡터들을 가져올 때 임베딩 테이블의 temporal locality를 높이기 위해서 각 연산기들이 같은 임베딩 테이블에서 데이터를 가져오는데에 우선순위를 두는 스케줄링을 적용해서 latency를 줄였다.

3) TRiM

TensorDIMM과 RecNMP와 달리 TRiM [8]은 임베딩 테이블에 row-wise partitioning을 적용하였다. 이는 DRAM 기반 PIM의 특성상 임베딩 테이블을 서로 다른 메모리 뱅크에 저장하기 위해서 column-wise partitioning을 적용하면 internel memory bandwidth가 낭비되므로 효율을 높이기 위해서이다. 추천시스템에서의 임베딩 벡터 하나의 크기는 크지 않아서 memory bank의 하나의 row에 대해 activation을 수행하고 데이터를 가져오는 DRAM 구조 상, 온전한 임베딩 벡터에 접근하는 것과 부분적인 임베딩 벡터에 접근하는 것이 비효율적이다. Fig. 3에서 메모리 근처에서 연산기는 rank(주황색), bank-group(파란색), bank(연두색) 단위로 배치되어 있다. rank 단위 연산기는 연산기의 개수가 적기 때문에 연산기간 workload imbalance가 적으며 cost가 낮지만 PIM에 의한 성능향상이 비교적 적다는 단점이 있다. 반면, bank 단위 연산기는 연산기의 개수가 많기 때문에 PIM에 의한 성능향상이 가장 높지만 workload imbalance가 발생하며 cost가 높다. TRiM은 효율을 위해 cost 대비 vector addition 연산 수가 가

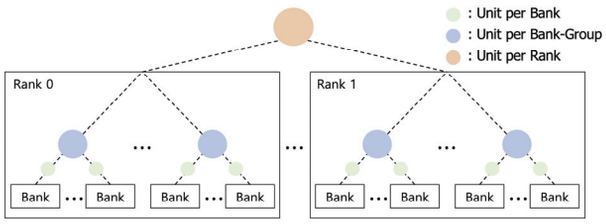

Fig. 3. Processing Element Unit Placement

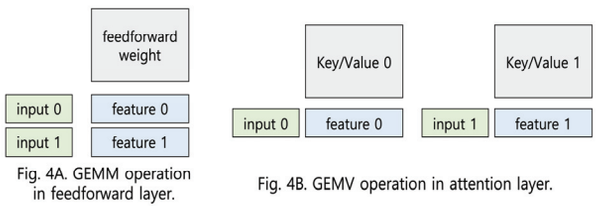

Fig. 4A. GEMM operation in feedforward layer.

Fig. 4B. GEMV operation in attention layer.

Fig. 4. Comparison Between Fully-connected and Attention Layers

장 높게 하기 위해 연산기를 bank-group 단위로 배치하였다.

추천시스템의 특성상, 하나의 아이템을 표현하는 임베딩 벡터의 접근은 power-low distribution 형태를 띈다. 즉, 자주 접근되는 임베딩 벡터들이 존재하며 이로 인해 PIM을 이용한 추천시스템에서는 workload imbalance가 발생한다. TRiM은 자주 접근되는 임베딩 벡터들을 메모리 뱅크에 중복되게 저장함으로써 연산기간의 workload imbalance를 해결하였다.

2.2 생성형 언어모델

1) AttAcc

AttAcc [1] 에서는 생성형 언어모델에서 흔히 병목현상으로 꼽히는 attention layer를 PIM을 이용해 해결하였다. 연산능력이 좋은 GPU의 특성상 GEMM 연산같은 computation-intensive operation에 적합하므로 생성형 언어모델을 실행시킬 때 GPU의 utilization을 높이기 위해 가장 널리 사용되는 기법 중 하나가 batching이다. Fig. 4A에서 batching을 통해 들어온 입력값들은 같은 모델 weight를 공유함으로써 weight 행렬의 reuse를 높이는 GEMM 연산 형태를 보인다. 반면, Fig. 4B의 attention layer에서 batching 기법을 사용해도 GEMM 연산이 아닌 여러개의 GEMV 연산을 하는 것을 알 수 있다. Attention layer에서는 batching 기법을 사용해도 입력값들이 key/value 값을 공유하지 않기 때문에 key/value 행렬을 reuse할 수 없으므로 비효율적이다. 또한, 이 key/value 행렬들의 크기가 배치 사이즈에 비례하기 때문에 GPU의 메모리 제한을 일으킨다.

AttAcc은 computation-intensive한 GEMM 연산은 GPU에서, memory-intensive한 GEMV 연산은 PIM에 처리하게 함으로써 효율성을 높였다. 성능, 에너지, 디램 내의 공간 등을 고려하여 뱅크 레벨로 GEMV 연산기를 배치하였으며 cost를 고려해서 HBM buffer die마다 softmax 연산기를 배치하였다. 메모리 뱅크에 대해서는 column-wise partitioning을 적용해서 GEMV 연산을 통해 얻는 결과값의 데이터 이동을 최소화하였다. 또한, attention layer와 fully-connected layer 사이에는 dependency가 존재하므로 GPU와 PIM에서 연산을 동시에 하기 위해 기존의 batch를 2개로 나눠서 GPU와 PIM이 각각 다른 batch에 대한 연산을 하게 설계하였다.

2) NeuPIMs

NeuPIMs [2]도 [1]과 같이 GEMM/GEMV 각 연산에 대해 효율적인 가속기인 GPU/PIM을 사용하였다. 또한, attention layer와 fully-connected layer간의 dependency를 batch를 2개로 나눠서 pipelining하는 형식으로 사용하였다. 또한 실험에서는 GPU에서 널리 사용되는 가속화 기법인 Orca [9]와 메모리 관리 기법인 vLLM [10]을 사용하였다.

NeuPIMs는 메모리 공간을 공유해서 GPU와 PIM을 동시에 실행시킬 수 없는 GPU-PIM 가속기의 문제를 해결하기 위해 double buffering을 제안하였다. 기존 DRAM에서 어떤 데이터에 접근할 수 있는 row buffer는 1개이지만 row buffer를 통해 PIM row buffer와 memory row buffer 중 하나로 데이터를 전송함으로써 PIM과 GPU가 제대로 데이터를 얻을 수 있게 설계하였다.

3. LUT 기반 PIM System의 소프트웨어적 기법

3.1 Lookup Table

연산능력이 뛰어난 GPU에서는 데이터의 이동이 병목현상이 발생할 수 있는 것과 반대로, 데이터 이동에 강점을 가진 PIM에서는 메모리 근처에 위치시킨 연산기의 연산이 병목현상으로 지목되는 경우가 많다 [11]. 연산기의 배치에는 cost, power, area 오버헤드 등이 고려되어 성능이 좋은 연산기를 배치하기 쉽지 않기 때문이다. 연산 능력을 해결하기 위한 대표적인 방법은 Lookup Table(LUT)를 사용하는 것이다. LUT는 가능한 연산에서 나올 수 있는 결과값들을 미리 계산해서 테이블 형태로 저장하는 것을 의미한다. 어플리케이션 런타임에 연산을 계속해서 수행하지 않고, 실행할 연산에 해당하는 LUT 인덱스를 이용해서 LUT에서 미리 계산한 결과값을 가져온다. 중복되는 연산이 많다면, 계속 실제로 연산을 하는 것이 아닌 LUT를 구성하고 LUT를 사용하는 것이 효율적이다 [12-14].

AI 모델 실행에서 LUT는 주로 multiplication이나 multiply-accumulation (MAC) 연산을 저장하는데 사용된다. 크기가 너무 큰 LUT는 실제로 사용하는데 적합하지 않기 때문에 적당한 크기의 LUT를 사용하거나 quantization한 값들에 대한 MAC 결과값을 저장함으로써 search space를 줄여서 LUT의 크기를 줄이는 방법이 많이 사용된다. T-MAC [15]에서는 weight 행렬을 n-bit로 quantization한 뒤, 블록 단위로 자른 activation과의 곱연산의 결과값을 미리 계산한 LUT에서 참

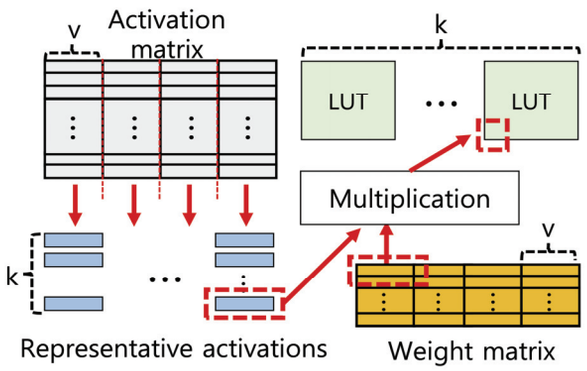

Fig. 5. LUT Generation in PIM-DL

조한다. 참조한 중간 결과값에 대해 shifting과 addition을 통해 최종 결과값을 얻을 수 있다. 추가적으로 LUT의 사이즈를 줄이기위해 LUT가 대칭적이라는 특성을 이용하여 LUT의 사이즈를 반으로 줄이고 추후에는 부호를 고려해서 참조한다.

PIM-DL [16]에서는 PIM에서의 언어모델 추론을 가속화를 위해 LUT를 사용하였다. Fig. 5와 같이 LUT(보라색)을 미리 구성하기 위해 훈련데이터에서 일부를 실행시켜서 얻은 activation matrix(회색)들을 sub-vector단위로 자른다. 추후 추론 과정에서는 대표 sub-vector들 중에서 실제 activaiton과 가장 가까운 vector를 선택하기 때문에 정확도를 위해 모델 추론 속도 하락이 크지 않은 선에서 최대한 작게 자른다. 이후 sub-vector들에 대해 K-means 클러스터링을 수행해서 얻은 대표적인 activation pattern(파란색)과 weight 행렬(주황색)과의 곱연산을 LUT 형태로 저장한다. 이후 추론에서는 대표 sub-vector들과 실제 모델을 실행시켜서 나온 activation과 가장 가까운 것을 선택한 뒤, 이에 해당하는 weight와의 곱연산 결과값을 LUT로부터 참조해서 가져온다. 마지막으로 LUT에서 가져온 중간 결과값들을 더해서 최종 결과값을 얻는다.

LUT를 사용하면 연산능력이 부족한 PIM의 문제를 줄여줄 수 있으며, 이를 위해 PIM 하드웨어에 적합한 LUT의 크기 선택이 중요하다. LUT의 크기가 클수록 더 많은 값들에 대한 MAC 연산 결과를 저장할 수 있어서 연산 횟수가 더욱 줄어드는 효과를 얻을 수 있지만, LUT의 크기가 너무 크면 memory/area 오버헤드로 인해 비효율적이다. 이런 LUT의 활용은 PIM의 떨어지는 연산능력을 보완해줄 수 있으므로 관련 연구가 활발히 필요하다.

3.2 Heterogeneous system

생성형 언어모델과 추천시스템의 주요 병목지점인 memory-intensive 연산들은 PIM을 통해 가속화가 가능하지만 이 어플리케이션들의 동작과정에는 연산 집적도가 높은 computation-intensive 연산도 있다. memory-intensive 연산들로 주로 구성되어 있는 데이터베이스 시스템 [17]과 달리 두 어플리케이션에서는 PIM만을 사용하는 것은 비효율적이다. 따라서 GPU-PIM heterogeneous system을 제시하는 [1, 2]는 생성형 언어모델에서 fully-connected layer의 GEMM 연산을, [5]는 추천 시스템에서 bottom/top mlp의 GEMM 연산을 모두 GPU에서 수행한다.

하지만 현재 GPU-PIM heterogeneous system은 이전에 GPU에서 연구되었던 다양한 AI 모델에 대한 가속화 기법 적용이 부족하다. 먼저 multi GPUs에 대해 제대로 고려되지 않았다. 기존 GPU 연구에서는 성능 향상과 GPU 메모리 제한을 해결하기 위해 multi GPUs를 사용하는 경우가 많다. GPU들 간의 통신을 줄이거나 다양한 방식의 병렬화를 통해 GPU의 메모리를 효율적으로 사용하는데 PIM을 사용하게 되면 다른 부분에서 병목현상이 나타날 가능성이 존재한다. 또한 GPU-PIM heterogeneous system에 대한 시스템적 고려가 부족하다. 현재 GPU-PIM heterogeneous system은 quantization [18-20] 등 GPU에서 꾸준히 연구되왔던 기법들이 지원되지 않기 때문에 GPU를 이용한 선행 연구들을 GPU-PIM system에 적용하려는 시도가 필요하다.

4. PIM System 상에서의 LUT 효용성 검증 실험

4.1 실험 환경 설정

LUT의 성능향상을 분석/검증하기 위해 실제 상용화된 PIM인 DDR4 기반의 UPMEM PIM system [21]에 LUT를 구현하였다. 공정성을 위해 랜덤 데이터셋을 사용하였으며 호스트 프로세서로는 Intel Xeon Gold 5215 CPU를 사용하였다. PIM이 내장되어있는 DIMM을 사용하였으며, 실험에 사용된 UPMEM-enabled DIMM은 4개의 채널과 채널당 4개의 랭크가 사용되었다. UPMEM PIM system은 DDR4 기반이기 때문에 랭크당 8개의 칩, 칩당 8개의 메모리 뱅크로 구성이 되어 있다. UPMEM은 뱅크 당 하나의 연산기가 배치되어 있기 때문에 실험해서 사용된 총 연산기의 개수는 4(채널) × 4(랭크) × 8(칩) × 8(뱅크) = 1024개이다.

4.2 LUT 실험 구성

UPMEM PIM system은 메모리 뱅크마다 연산기가 배치되어 있으며 그 밖에 데이터가 저장되어 있는 Main Ram, 스크래치패드 메모리인 Working Ram (WRAM)으로 구성되어 있다. 연산기에서 Main Ram으로부터 데이터를 가져와서 연산을 수행하기 위해서는 반드시 WRAM을 거쳐야하는 구조이며, WRAM은 유저가 자유롭게 사용할 수 있기 때문에 LUT를 생성하여 WRAM에 저장하고 연산기에서 연산대신 LUT를 참조할 수 있게 구현하였다.

LUT는 low-bit MAC 연산 결과값을 저장하였으며 실제 PIM system 상에서 2bit나 4bit 원소같은 low-bit MAC 연산을 수행할때와 LUT를 참조할때의 성능 차이를 비교하였다. 2-bit LUT는 2-bit 원소 두 개의 multiplication 값들을 저장

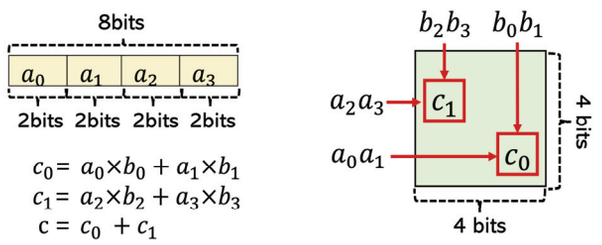

Fig. 6. Examples of 4-bit LUT for a 2-bit Element MAC Operation

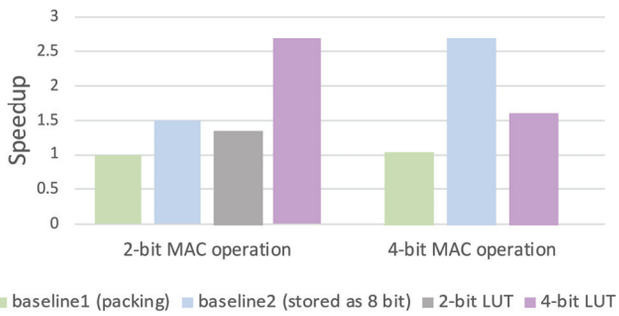

Fig. 7. Comparison of LUT and Naive MAC Computation in Real PIM System

한 테이블이며 LUT를 참조할 때 인덱스로 2-bit 원소 두 개를 받는다. 마찬가지로 4-bit LUT는 참조 시 인덱스로 4-bit 원소 두 개를 받는다. 4-bit 원소간의 MAC 연산 실험에서 4-bit LUT에는 4-bit 원소 두 개의 multiplication 값이 저장되어 있다. 반면, 2-bit 원소간의 MAC 연산 실험에서 4-bit LUT는 2-bit 원소의 MAC 연산 값이 저장되어 있다. Fig. 6은 2-bit 원소간의 MAC 연산 실험에서 4-bit LUT가 참조되는 방법으로 UPMEM의 하나의 뱅크 메모리 용량은 64MB로 큰 편은 아니기 때문에 메모리를 아끼기 위해서 2-bit 원소 $a_0 - a_3$(노란색)이 결합되어 8-bit로 packing이 되어 있다. 2-bit 원소 $a_0 - a_1$와 $b_0 - b_1$간 MAC 연산을 하는 대신 a_0a_1와 b_0b_1를 결합된 상태 그대로 LUT의 인덱스로 사용하면 MAC 연산 값인 c_0을 얻을 수 있다. 마찬가지로 a_2a_3과 b_2b_3을 인덱스로 사용하여 LUT를 참조하면 c_1를 얻을 수 있으며 c_0와 c_1을 더해서 최종적으로 $a_0 - a_3$과 $b_0 - b_3$의 MAC 연산값인 c 를 얻을 수 있다.

첫 번째 베이스라인으로는 메모리 뱅크를 2~4배 효율적으로 사용하기 위해 Fig. 6의 $a_0 - a_3$(노란색)과 같이 2bit 혹은 4-bit 원소 값들이 결합되어 8-bit 값으로 packing되었다고 가정하였다. MAC 연산을 위해서는 WRAM으로 데이터를 가져와서 unpacking을 한 뒤, MAC 연산을 수행한다. 두 번째 베이스라인으로는 2-bit나 4-bit 원소들이 전부 8-bit형태로 저장되어 있어 메모리 뱅크를 비효율적으로 사용하는 대신 unpacking에 의한 오버헤드없이 MAC 연산을 수행하는 경우이다. 첫 번째로 제시하는 2-bit LUT는 2-bit와 2-bit 값들끼리의 multiplication 값을 저장한다. 반면, 4-bit LUT는 2-bit 원소간의 실험에서는 MAC 연산 값을, 4-bit 원소간의 실험에서는 multiplication 값을 저장한다. 즉, 4-bit LUT를 이용한 4-bit 원소간의 MAC 연산은 4bit multiplication 값을 가져와서 이를 연산기에서 더하는 방식이다.

4.3 실제 PIM에서 구현한 LUT의 성능 향상

Fig. 7에서 baseline1(연두색)에 비해 baseline2(하늘색)가 더 빠른 것을 알 수 있다. 이는 baseline2는 메모리를 많이 사용하는 대신 unpacking 과정을 거치지 않기 때문이다. 또한 2-bit MAC 연산에 비해 4-bit MAC 연산의 speedup이 훨씬 컸는데 이는 baseline1에서 2-bit나 4-bit 원소들이 8-bit로 packing되어 있기 때문이다. 만약 LUT의 사이즈가 더 커서 8-bit보다 더 크게 packing이 되어있으면 둘 사이의 speedup 차이는 줄어들 것으로 보인다.

2-bit LUT(회색)은 4bit끼리의 MAC multiplication 결과값을 저장할 수 없기 때문에 2bit끼리의 MAC 연산에서만 실험을 하였다. 2-bit LUT를 사용했을때는 baseline1에 비해 1.3배의 성능 향상이 있었다. 이는 multiplication과 accumulation 연산을 연이어 수행하는 것이 아닌 미리 계산된 LUT에서 참조해온 값들을 더함으로써 multiplication 연산 수를 줄였기 때문이다.

4-bit LUT(연보라색)은 2-bit 원소 실험에서는 2.7배, 4-bit 원소 실험에서는 1.6배의 성능 향상을 보였다. 2-bit LUT는 접근 1번당 multiplication 1번을 덜 할 수 있지만 4-bit LUT는 LUT의 크기가 크기 때문에 더 많은 결과값을 저장할 수 있어서 접근 1번당 multiplication 2번과 addition 1번을 안 할 수 있다. 접근 횟수 대비 줄어드는 연산 수가 크기 때문에 2.7배라는 성능 향상을 얻을 수 있었던 것으로 보인다. 4-bit 원소간 MAC 연산 실험에서 baseline2가 4-bit LUT에 비해 더 좋은 성능을 보였지만 unpacking 과정을 피하기 위해 4-bit 원소를 8-bit 단위로 저장함으로써 Main Ram을 2배 더 많이 사용하게 된다. 또한, LUT의 사이즈가 커질수록 줄어드는 연산 수가 많아지므로 충분히 큰 사이즈의 LUT에 대해서는 더 좋은 성능을 보일 것으로 생각된다.

4.4 실제 어플리케이션에서 LUT의 활용

LUT는 PIM에서 GEMM 연산이나 GEMV 연산을 수행할 때 LUT의 연산을 가속화하는 역할을 한다. [1, 2]는 PIM에서 생성형 언어모델의 GEMV 연산을 수행하며 [16]은 PIM에서 GEMM 연산을 수행한다. PIM에서 언어모델의 low-bit quantization 선행 연구들에서 제시한 기법들을 적용하면 LUT의 이점을 충분히 얻을 수 있을 것으로 생각된다 [18-20]. 추천시스템 또한 bottom/top mlp에서 quantization을 적용할 수 있으며 임베딩 벡터들에 대해서도 quantization을 적용할 수 있으므로 LUT를 충분히 활용 가능하다 [22, 23].

5. 결 론

PIM은 memory-intensive 연산에 성능/파워 측면에서 효율적인 가속기로 실생활에 널리 사용되는 추천시스템과 생성형 언어모델과 같이 memory-intensive 연산을 수행해야하는 어플리케이션들에 효과적인 가속기이다. 본 논문에서는 PIM을 이용해서 추천시스템과 생성형 언어모델을 가속화한 선행 연구들의 전략을 살펴보고, LUT와 GPU-PIM heterogeneous system 측면에서 선행 연구들과 PIM이 연구되어야 할 방안에 대해 토의하였다. 또한, 실제 상용화된 PIM system에서 LUT의 효과를 분석하고 검증하였다.

References

[1] J. Park et al., "AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*, La Jolla CA USA: ACM, pp.103-119, Apr. 2024. doi: 10.1145/3620665.3640422.

[2] G. Heo et al., "NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3*, La Jolla CA USA: ACM, Apr. 2024.

[3] L. Ke et al., "Near-Memory Processing in Action: Accelerating Personalized Recommendation With AxDIMM," *IEEE Micro*, Vol.42, No.1, pp.116-127, Jan. 2022. doi: 10.1109/MM.2021.3097700.

[4] Z. Wang, Y. Wang, B. Feng, D. Mudigere, B. Muthiah, and Y. Ding, "EL-Rec: Efficient Large-Scale Recommendation Model Training via Tensor-Train Embedding Table," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA: IEEE, pp.1-14, Nov. 2022. doi: 10.1109/SC41404.2022.00075.

[5] Y. Kwon, Y. Lee, and M. Rhu, "TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Columbus OH USA: ACM, pp.740-753, Oct. 2019. doi: 10.1145/3352460.3358284.

[6] D. Mudigere et al., "Software-hardware co-design for fast and scalable training of deep learning recommendation models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, pp.993-1011, 2022.

[7] L. Ke et al., "RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain: IEEE,, pp.790-803 May 2020. doi: 10.1109/ISCA45697.2020.00070.

[8] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, Virtual Event Greece: ACM, pp.268-281, Oct. 2021. doi: 10.1145/3466752.3480080.

[9] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A Distributed Serving System for Transformer-Based Generative Models," in *OSDI: 16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, pp.521-538, 2022.

[10] W. Kwon, et al, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp.611-626, 2023.

[11] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," *arXiv preprint arXiv:2105.03814*, 2021.

[12] Y. Jeon, B. Park, S. J. Kwon, B. Kim, J. Yun, and D. Lee, "BiQGEMM: Matrix Multiplication with Lookup Table for Binary-Coding-Based Quantized DNNs," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp.1-14, 2020.

[13] D. C. Ganji et al., "DeepGEMM: Accelerated Ultra Low-Precision Inference on CPU Architectures using Lookup Tables," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Vancouver, BC, Canada: IEEE, pp.4656-4664, Jun. 2023. doi: 10.1109/CVPRW59228.2023.00491.

[14] D. C. Ganji et al., "DeepGEMM: Accelerated Ultra Low-Precision Inference on CPU Architectures using Lookup Tables," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Vancouver, BC, Canada: IEEE, pp.4656-4664, Jun. 2023. doi: 10.1109/CVPRW59228.2023.00491.

[15] J. Wei et al., "T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge," Jun. 25, 2024, arXiv: arXiv:2407.00088. Accessed: Jul. 22, 2024. [Online]. Available: http://arxiv.org/abs/2407.00088.

[16] C. Li et al., "PIM-DL: Expanding the Applicability of Commodity DRAM-PIMs for Deep Learning via Algorithm-System Co-Optimization," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Volume 2, La Jolla CA USA: ACM, pp.879-896, Apr. 2024.
[17] C. Lim et al., "Design and analysis of a processing- in-dimm join algorithm: A case study with upmem dimms," in *Proceedings of the ACM on Management of Data*, Volume 1, No. 2, ACM, June 2023, p. 1-27.
[18] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa, "QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks," Jun. 04, 2024, arXiv: arXiv:2402.04396. Accessed: Jul. 22, 2024. [Online]. Available: http://arxiv.org/abs/2402.04396
[19] S. Shen et al., "Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT," AAAI, Vol. 34, No. 05, pp. 8815-8821, Apr. 2020, doi: 10.1609/aaai.v34i05.6409.
[20] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT Quantization," in *International Conference on Machine Learning (PMLR)*. pp.5506-5518, 2021.
[21] "UPMEM SDK," https://sdk.upmem.com/
[22] M. Anderson et al., "First-generation inference accelerator deployment at facebook," *arXiv preprint arXiv:2107.04140*, 2021.
[23] H. Guan et al., "Post-training 4-bit quantization on embedding tables," *arXiv preprint arXiv:1911.02079*, 2023.
[24] S. Noh. et al., "PID-Comm: A Fast and Flexible Collective Communication Framework for Commodity Processing-in-DIMM Devices," in *2024 ACM/IEEE 51th Annual International Symposium on Computer Architecture (ISCA)*, 2024.

홍 정 욱

https://orcid.org/0009-0001-4004-7714
e-mail : junguk16@snu.ac.kr
2023년 연세대학교 컴퓨터과학과(학사)
2023년 ~ 현 재 서울대학교 전기정보공학부 석사과정
관심분야 : Processing-in-Memory (PIM)

이 진 호

https://orcid.org/0000-0003-4010-6611
e-mail : leejinho@snu.ac.kr
2009년 서울대학교 전기공학부(학사)
2011년 서울대학교 전기컴퓨터공학부(석사)
2016년 서울대학교 전기컴퓨터공학부(박사)
2022년 ~ 현 재 서울대학교 전기정보공학부 교수
관심분야 : 인공지능 시스템 및 최적화, 컴퓨터 구조