Processing-in-Memory 를 이용한 추천시스템 가속화 및 분석

홍정욱 [1], 이진호 [2]
[1] 서울대학교 전기정보공학부 석사과정
[2] 서울대학교 전기정보공학부 교수

junguk16@snu.ac.kr, leejinho@snu.ac.kr

Accelerating and analyzing the Recommendation System using Processing-in-Memory

Jung-uk Hong[1], Jin-ho Lee[2]
[1]Dept. of Electrical and Computer Engineering, Seoul National University
[2]Dept. of Electrical and Computer Engineering, Seoul National University

요 약

추천 시스템(Recommendation System)은 인터넷 쇼핑몰, 넷플릭스, SNS 등 여러 분야에서 유저에게 맞는 타겟 광고를 추천하는 시스템을 말한다. 추천 시스템을 가속하기 위해서는 추천 시스템 모델에서 불규칙적이고 잦은 데이터 이동으로 인해 병목현상을 일으키는 임베딩 레이어를 타겟하는 것이 중요하다고 알려져 있다. 이 논문에서는 데이터 이동이 잦은 어플리케이션에 효과적인 Processing-in-Memory 를 이용하여 추천 시스템을 가속하고 분석한다.

1. 서론

Deep Learning Recommendation Model(DLRM)[1]은 사용자의 성별, 나이와 같은 정보와 사용자가 선호하는 아이템 목록을 기반으로 특정 아이템의 클릭 확률(Click Through Rate)을 예측하는 모델이다. AI 기반 추천 시스템 추론을 위한 DLRM 은 메타의 데이터센터의 대부분의 자원을 사용하는 것으로 알려져 있다.[2]

그림 1 은 Click Through Rate(CTR)을 구하는 DLRM 의 전체적인 구조를 나타낸 모습이다. DLRM 은 사용자의 정보를 이용한 Multi-Layer Perceptron(MLP) 2 개와 사용자가 선호하는 아이템 정보를 포함하는 임베딩 레이어(Embedding Layer)로 구성되어 있다. DLRM 에서 병목현상을 일으키는 임베딩 레이어는 각 임베딩 테이블(Embedding Table)에서 데이터셋에 의해 선택되는 임베딩 벡터(Embedding Vector)들을 더해서 해당 임베딩 테이블을 대표하는 임베딩 벡터를 계산한다. 임베딩 벡터들은 Bottom MLP 에서 계산된 벡터와 결합되고 추가적인 연산을 거쳐서 Top MLP 의 입력이 된다. 하지만 임베딩 테이블의 크기가 굉장히 크고, 데이터셋 특성상 임베딩 벡터가 랜덤하게 선택되기 때문에 불규칙적이고 잦은 데이터 이동이 일어나게 된다.

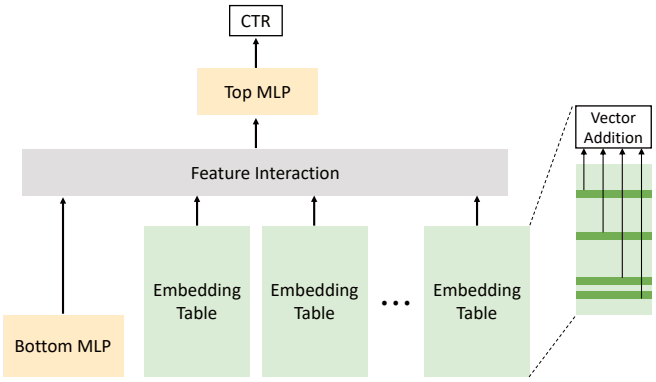

그림 1. DLRM 구조.

이는 임베딩 레이어가 DLRM 의 추론 시간의 대부분을 차지하게 되는 문제를 일으킨다.

본 연구에서는 현존하는 DLRM 가속화 방법을 분석하고 Processing-in-Memory 를 이용해 DLRM 의 추론을 가속한 뒤, CPU 만을 이용한 DLRM 의 추론과 비교하여 성능이 개선되었음을 보인다. PIM 을 이용한 DLRM 가속을 위해 임베딩 테이블을 나누는 다양한 Partitioning 을 적용하고 성능을 분석한다. 마지막으로 PIM 을 이용한 DLRM 가속에서의 문제점과 향후 개선 방향에 대해 논의한다.

2. 배경

2.1. Processing-in-Memory

Processing-in-Memory(PIM)은 데이터가 저장되어 있는 메모리 근처에 연산기를 위치시킴으로써 데이터에서 더 가까운 곳에서 연산을 할 수 있는 가속기이다. CPU 나 GPU 에 비해 PIM 은 어플리케이션 시간의 병목 현상이 데이터 이동일때 특히 더 효과적으로 작동한다.[3] DLRM 의 임베딩 레이어 또한 연산량에 비해 데이터 이동이 많은 패턴을 띄기 때문에 PIM 을 적용하면 DLRM 을 효과적으로 가속할 수 있다.

2.1. DLRM Acceleration

DLRM 의 임베딩 레이어를 가속하는 방법은 크게 두가지가 있다. 첫번째는 자주 접근하는 임베딩 벡터를 캐싱(caching)하는 방법이다.[4] DLRM 의 임베딩 테이블에서 사용자가 선호하는 아이템들을 입력으로 받는데 데이터셋 특성상 선호하는 아이템들이 존재한다. 이 아이템들을 의미하는 임베딩 벡터들 각각과 이들의 부분합을 CPU 나 GPU 메모리에 미리 위치시킴으로써 데이터 이동을 효과적으로 줄일 수 있다.

두번째는 PIM 을 이용한 방법이다. 임베딩 벡터들이 위치한 메모리 근처에 있는 연산기에서 임베딩 벡터들의 부분합을 계산해서 계산된 결과값을 CPU 로 가져온다.[5] 메모리 근처에서 이루어진 연산 결과값을 가져오기 때문에 PIM 을 이용하면 데이터 이동을 효과적으로 줄임으로써 어플리케이션을 가속할 수 있다.

3. PIM 을 이용한 DLRM 가속화

PIM 을 이용한 기존 DLRM 가속화 연구들은 대부분 임베딩 테이블을 PIM 에 저장하고 임베딩 레이어만을 가속하였다. 이 논문에서는 다양한 partitioning 기법을 이용하여 DLRM 의 Bottom MLP, Top MLP, Feature Interation 을 전부 PIM 에서 가속화한다.

Partitioning 에 따른 가속화를 분석하기 위해 테이블별, 가로별, 세로별로 임베딩 테이블을 나누는 기법을 적용하였다.[6] 나누어진 임베딩 테이블은 PIM 의 각 메모리 뱅크내에 저장되어 근처에 위치한 연산기에 의해 계산이 된다.

Bottom MLP, Top MLP, Feature Interaction 은 배치(batch) 단위로 진행되지만 임베딩 테이블은 PIM 의 각 메모리가 고유한 영역을 가지고 있기 때문에 모든 배치로부터 해당하는 데이터를 가져와야 한다. 이를 위해 PIM 의 메모리 간에 AlltoAll, ReduceScatter 이라는 통신 기법이 사용되었다.[7] 임베딩 테이블을 테이블별 또는 세로별로 나누면 데이터의 위치만 이동시켜주면 되므로 AlltoAll 이 사용되지만, 가로로 나누면 각 PIM 의 메모리에서는 연산을 통해 임베딩 벡터의 부분합만 얻을 수 있으므로 ReduceScatter 이 사용된다.

4. PIM 을 이용한 DLRM 추론 시간 분석

4.1. 실험환경

데이터셋은 추천시스템에서 가장 대표적인 Criteo dataset[8]을 사용하였으며, 시뮬레이션이 아닌 16 개의 랭크로 구성된 실제 UPMEM[9] PIM 시스템을 사용하였다. 호스트 프로세서로는 Intel Xeon Gold 5215 CPU 가 사용되었다.

4.2. PIM 을 이용한 DLRM 성능 분석

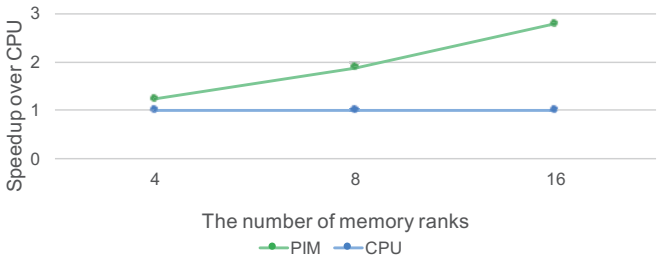

그림 2. PIM 의 랭크 수에 따른 PIM 으로 가속화한 DLRM 과 CPU 를 이용한 DLRM 의 추론 시간 비교.

PIM 의 메모리 뱅크의 용량 제한을 위해 임베딩 테이블을 테이블별, 가로별, 세로별로 나누는 기법을 적용하였다. 그림 2 는 3 가지 partitioning 기법을 모두 적용한 PIM 시스템과 CPU 시스템 간의 추론 시간 비교를 나타낸다. PIM 의 메모리 랭크 수를 늘림에 따라 CPU 에 비해 최대 2.8 배로 성능이 거의 선형적으로 증가하였다. 이는 PIM 이 기대했던 가속기의 역할을 제대로 수행하고 있음을 의미한다.

CPU 에서는 불규칙적이고 잦은 데이터 전송에 비해 연산량이 부족하여 CPU 가 비효율적으로 사용된다. 반면 PIM 에서는 메모리 근처에서 임베딩 벡터를 더함으로써 메모리 전송의 비효율성을 효과적으로 해결하였다. PIM 의 메모리 뱅크 수가 늘어남에 따라 성능이 거의 선형적으로 증가하지만 완전히 선형이 아닌 이유는 병렬처리를 위해 partitioning 을 했기 때문이다. PIM 을 이용하기 위해 임베딩 테이블들을 나눠서 저장하였으므로 임베딩 벡터들을 더하기 위해 입력 데이터가 알맞은 위치의 임베딩 테이블들을 찾아가야 하기때문에 메모리 뱅크들 간의 통신이 필요하다. 또한, 성공적으로 임베딩 벡터들을 구한 뒤에도 Top MLP 계산을 위해 추가적인 메모리 뱅크들 간의 통신이 필요하다. 결국 메모리 랭크 수를 늘릴수록 통신량이 늘어가게 되고 이는 오버헤드를 의미한다. 그럼에도 불구하고 통신에 의한 오버헤드에 비해 연산 유닛과 메모리 용량이 늘어남에 따라 성능이 거의 선형적으로 증가하였고 이는 PIM 이 효과적으로 임베딩 벡터를 처리한다는 것을 의미한다.

4.2. Partitioning 에 따른 Embedding Layer

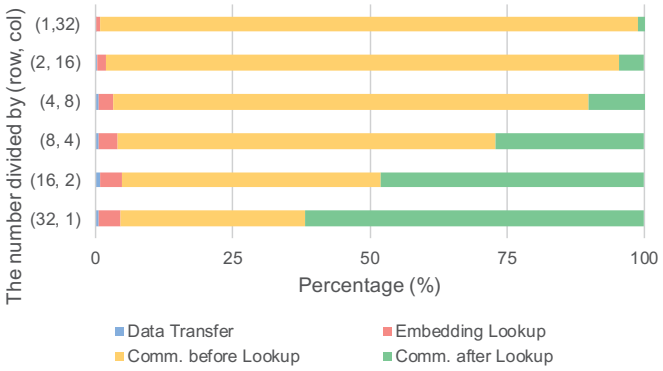

그림 3. 임베딩 테이블을 다른 Partitioning 기법을 사용하여 PIM 에 나눴을때의 실행 시간 비중 비교.

본 논문의 실험에서는 1,024 개의 메모리 뱅크를 사용하였다. 테이블별로 임베딩 테이블들을 나눌 때 전체 메모리 뱅크를 32 개로 나누어서 사용해야 하므로 각 임베딩 테이블을 가로와 세로별로 나눌 때 32 개의 메모리 뱅크를 사용할 수 있다.

그림 3 에서는 가로별과 세로별로 나누는 메모리 뱅크의 개수를 조절해서 얻은 실행 시간의 비중을 알 수 있다. 먼저 입력 데이터 이동 시간은 변화가 없었지만 임베딩 벡터들을 메모리 근처에서 연산하는 작업인 Embedding Lookup 부분의 비중이 소폭 증가하였다. 이는 임베딩 테이블을 세로로 자를수록 메모리 뱅크 간의 불균형이 줄어들기 때문이다. 데이터셋 특성상 임베딩 테이블을 균일하게 접근하는 것이 아닌 불규칙하게 특정 임베딩 벡터(특정 아이템)을 접근하는 현상이 존재한다. 임베딩 테이블을 가로로 자르면 이 현상으로 인해 연산기 간의 연산 불균형이 심해진다. 반면, 임베딩 테이블을 세로로 자르면 같은 임베딩 벡터를 여러 연산기가 나눠서 연산하기 때문에 연산 불균형이 사라진다.

PIM 의 메모리 뱅크에 저장되어 있는 임베딩 테이블 일부분을 알맞게 이용하기 위해서는 연산 전에 메모리 뱅크 간의 통신이 필요하다.[10] 또한, 연산 결과를 Top MLP 에서 이용하기 위해서는 연산 후에 메모리 뱅크 간의 통신이 추가적으로 필요하다. 연산 전 통신은 임베딩 테이블을 가로로 자를수록 줄어드는데 이는 임베딩 테이블을 세로로 자르면 같은 입력 데이터를 다수의 메모리 뱅크로 보내주어야 해서 통신량이 늘어나기 때문이다. 한편, 연산 후 통신은 임베딩 테이블을 가로로 자를때 더 커지게 되는데 이는 임베딩 테이블을 가로로 자르면 연산 후 통신이 빠른 AlltoAll 통신이 아닌 느린 ReduceScatter 통신을 사용해야 되기 때문이다.

5. 결론 및 향후 연구

본 논문에서는 DLRM 을 가속하는 다른 연구들을 살펴보고 PIM 을 이용하여 AI 기반 추천 시스템인 DLRM 을 가속하였다. 다양한 partitioning 기법을 이용하여 PIM 의 메모리 뱅크에 임베딩 테이블을 나눠서 배치하였으며 Bottom MLP, Top MLP, Feature Interaction 과 같은 DLRM 에서 임베딩 레이어를 제외한 부분도 PIM 을 이용해 가속하였다. 1024 개의 메모리 뱅크를 사용한 실험 환경에서 PIM 이 CPU 보다 최대 2.8 배 가속화되었음을 보였으며, PIM 의 크기가 커질수록 성능도 거의 선형적으로 증가하는 것을 통해 PIM 이 DLRM 에 효과적으로 작동하는 것을 알 수 있었다. Partiitoning 기법에 따른 PIM 에서의 DLRM 추론 시간을 비교하였고, PIM 시스템 상에서 추천 시스템을 사용할 때 어떤 점을 고려해야 하는지를 분석하였다.

이 분석을 토대로 향후 연구에서는 PIM 을 이용해 DLRM 을 더 가속화한다. 첫 번째로 단순히 테이블별, 가로별, 세로별로 임베딩 테이블을 나눠서 PIM 의 메모리 뱅크에 저장하는 것이 아닌 데이터셋 특성을 고려한 partitioning 기법을 통해 메모리 뱅크 근처에 위치한 연산기 간의 연산 불균형을 해결한다. 임베딩 테이블을 세로로 자르면 연산 불균형은 해결되지만 PIM 의 특성상 더 작은 데이터를 읽게 되어 비효율적이므로 이를 해결할 수 있는 방법이 필요하다. 두번째로 PIM 내부의 연산기가 버거워하는 연산인 Bottom MLP, Top MLP, Feature Interaction 을 GPU 와 같이 더 특화된 가속기에 오프로딩함으로써 DLRM 을 더 가속화하는 방안이 필요하다.

참고문헌

[1] M. Naumov et al., “Deep learning recommendation model for personalization and recommendation systems”, arXiv preprint arXiv:1906.00091, 2019.

[2] U. Gupta et al., “The Architectural Implications of Facebook’s DNN-Based Personalized Recommendation”, in IEEE International Symposium on High Performance Computer Architecture, 2 2020, pp 488-501.

[3] J. Gomez-Luna et al., “Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system”, IEEE Access, vol. 10, pp. 52 565-52 608, 2022.

[4] H. Ye et al., “GRACE: A Scalable Graph-Based Approach to Accelerating Recommendation Model Inference”, in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Vancouver BC Canada: ACM, 3 2023, Volume 3, pp 282-301.

[5] 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, 5 2020, pp 790-803.
[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, 2022, pp 993-1011.
[7] E. Chan et al., “Collective communication: theory, practice, and experience.”, Concurrency and computation: Practice and Experience, vol. 10, no. 13, pp. 1749-1783, 2007.
[8] “Criteolabs Kaggle display advertising challenge dataset.” https://labs.criteo.com/2014/02/download-kaggle-display-advertising-challenge-dataset/
[9] “UPMEM SDK.” https://sdk.upmem.com/
[10] S. U. 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, 2024.