FPGA를 활용한 PIM-HBM 에뮬레이션

최성헌○, 김경모, 신동군
성균관대학교 전자전기컴퓨터공학과
csh9580@skku.edu, 7bvcxz@skku.edu, dongkun@skku.edu

PIM-HBM emulation with FPGA

Sunghern Choi○, Kyungmo Kim, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

최근 다양한 분야에서 머신 러닝 모델을 사용하고 있다. 머신 러닝 모델의 워크로드는 수십억 이상의 계산이 필요하다. 이러한 머신 러닝 모델을 위한 GPU, TPU와 같은 하드웨어들이 빠르게 발전하고 있고 그로 인해 컴퓨팅 속도가 빨라졌다. 컴퓨팅 속도가 빨라짐에 따라 컴퓨팅에 필요한 데이터를 메모리로부터 가져오는 메모리 대역폭이 하드웨어의 성능에 큰 영향을 미치게 되었다. DRAM 기술들도 발전하고 있지만 높은 메모리 대역폭을 요구하는 계산을 하기 위해서는 아직 부족하다. 이에 대한 해결책으로 PIM(Processing In Memory)의 연구가 활발히 진행되고 있다. 본 논문에서는 PIM-HBM 에뮬레이터를 FPGA를 사용해서 구현한다. 구현한 에뮬레이터를 사용해서 연산했을 때 CPU와 GPU보다 각각 6.7, 19.67배 빠른 성능을 보여준다.

1. 서 론

최근 머신 러닝은 음성 인식, 자율주행 등 여러 분야에서 광범위하게 사용되고 있다. 이러한 분야의 머신 러닝 워크로드는 수십억 이상의 계산을 수행한다. 수십억 이상의 계산을 수행하기 위해 GPU, TPU 등의 하드웨어가 머신 러닝 모델을 가속하기 위한 가속기로 사용되고 있다. 가속기는 CPU보다 높은 성능을 제공한다. 하지만 가속기의 성능을 최대화하기 위해서는 가속기의 컴퓨팅 성능만큼의 높은 메모리 대역폭이 필요하다. 메모리 대역폭이 낮다는 것은 컴퓨팅 처리량만큼 메모리에서 데이터를 가져올 수 없는 것이기 때문에 가속기의 성능을 최대화 할 수 없다. 현재 최신의 DRAM 기술들은 메모리 대역폭을 늘리기 위해 노력하고 있지만 높은 메모리 대역폭을 요구하는 계산을 하기에는 아직 부족하다. 따라서 메모리 내부에서 연산을 하는 PIM(Processing In Memory)기술이 메모리 대역폭 문제를 해결하는 해결책으로 많이 사용되고 있다. PIM 기술은 메모리 내부에 연산장치를 두고 메모리 내부에서 필요한 계산을 할 수 있도록 하는 기술이다. 메모리 내부에 연산장치가 있어서 계산할 데이터를 가속기까지 가져올 필요가 없다. 이러한 이점을 활용해서 메모리 대역폭 문제를 해결할 수 있다.

메모리 대역폭 문제를 해결하기 위한 기존 PIM 연구들이 많이 있다. 일반 DRAM command를 수정하지 않고 그대로 사용해서 PIM 연산을 하는 연구(PIM-HBM)[1], FPGA를 사용해서 PIM 연산장치를 구현한 연구(Silent-PIM)[2] 등이 있다. 또한 PIM 장치를 사용해서 계산을 하는 동안 호스트 CPU가 작업을 하지 않는 것을 지적한 연구[3]도 있다.

PIM-HBM[1]은 실제 HW가 공개되어 있지 않기 때문에 정확한 성능 측정이 불가능하다. 따라서 성능을 검증할 수 있는 환경을 구축할 필요가 있다.

Silent-PIM[2]은 FPGA를 사용해서 PIM 연산을 구현했다. 하지만 JUMP 명령어를 지원하지 않는다. JUMP 명령어는 하나의 명령어를 사용해서 여러 번의 명령어를 수행할 수 있기 때문에 같은 크기의 워크로드를 처리할 때 적은 명령어로 수행을 가능하게 하는 장점이 있다. 따라서 JUMP 명령어를 지원하는 PIM 구조설계가 필요하다.

본 논문에서는 PIM-HBM[1]에서 사용한 PIM 구조를 FPGA를 사용해서 구현한다. 그리고 FPGA와 호스트 CPU 간의 통신을 위해 PCIe를 사용하는데 이때 발생하는 PCIe overhead를 최소화하기 위해 DMA를 사용하고, DMA의 성능을 최대로 활용하기 위해 PIM 연산에 필요한 데이터를 모두 패킹(packing)해서 전송한다.

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)

2. PIM-HBM 구조

그림 1과 같이 PIM-HBM[1]은 register(crf, srf, grf_a, grf_b), compute unit, 그리고 control unit 으로 구성되어 있다. 일반 DRAM command를 사용해서 모든 register에 데이터를 저장하기 때문에 각 register는 bank의 일부 영역에 매핑되어 있다. 각 register에 할당된 특정 row/column 주소를 사용해서 접근할 수 있다.

2.1 crf register

사용될 명령어를 저장하는 register 이다. 저장된 명령어에는 처리할 연산 정보, 연산시 사용할 데이터 등의 정보가 들어있다.

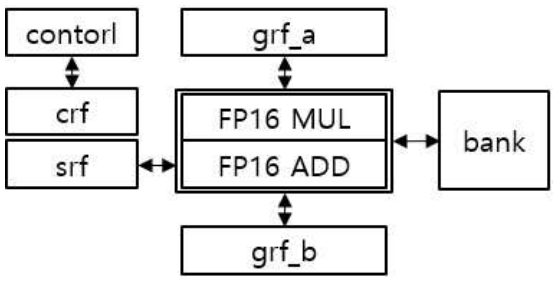

그림 1 PIM-HBM 구조. register(crf, srf, grf_a, grf_b)와 compute unit, control unit으로 구성되어있다.

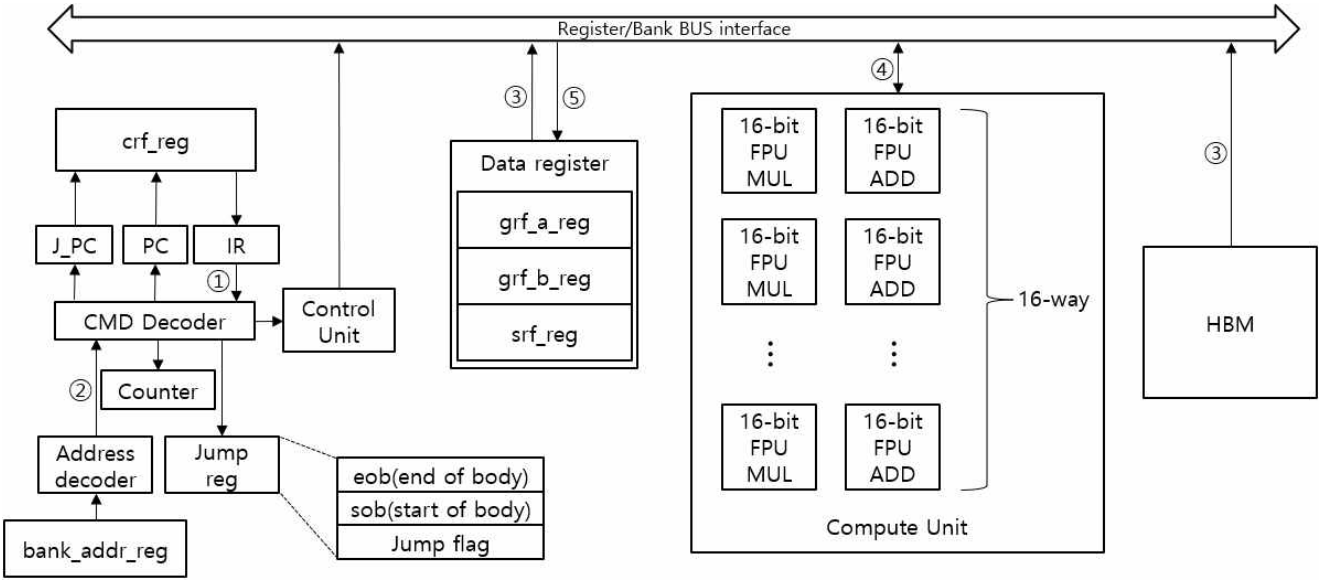

그림 2 PIM 연산을 위한 FPGA의 전체 구조를 나타낸다. 명령어 패치를 위한 PC, J_PC Address decoder 등이 있고 패치된 명령어를 처리하기 위한 Control Unit, Register, 그리고 Compute Unit이 있다.

2.2 srf, grf_a, grf_b register

계산에 사용될 데이터가 저장되어 있는 register 이다. grf_a와 grf_b는 사용될 데이터도 저장되어 있고 연산 결과도 저장한다.

3. FPGA를 활용한 PIM 연산장치 구현

본 논문에서는 PIM-HBM[1]에서 사용한 구조를 사용한다. 그림 2에서 볼 수 있듯이 PIM 연산에 사용할 crf, grf 등의 register 들이 있고 덧셈과 곱셈을 할 수 있는 16개의 FP-16 ADD, FP-16 MUL 연산장치가 있다. 그리고 JUMP 명령어를 처리하기 위한 J_PC(JUMP Program Counter)와 Jump register가 있다. 마지막으로 PIM 연산 시 필요한 주소 정보를 디코딩 하는 Address Decoder가 있다.

전체 동작은 명령어 패치(①)를 한 후 연산에 필요한 주소 정보를 디코딩 한다(②). 이후 연산에 필요한 데이터를 Data register 또는 HBM 메모리에서 읽는다(③). 읽은 데이터로 연산을 하고(④) 연산 결과를 grf register에 저장한다(⑤).

3.1 명령어 처리

crf에 저장되어있는 명령어를 처리하기 위해 PC(Program Counter)를 사용한다. JUMP 동작 시를 제외한 모든 상황에서 명령어는 PC를 사용해서 패치(Fetch) 한다. 명령어를 패치 하다가 JUMP 명령어가 패치되면 PC는 JUMP 다음 명령어를 포인팅(pointing)한 상태로 고정되고 JUMP 동작은 J_PC를 통해 수행된다. J_PC는 Jump Register에 저장된 SOB(Start Of Body), EOB(End Of Body) 정보를 바탕으로 동작한다. 그림 3에 나온 것처럼 SOB는 JUMP를 한 지점에 있는 명령어를 의미하고 EOB는 JUMP 명령어 바로 이전 지점에 있는 명령어를 의미한다.

JUMP 동작은 Counter에 저장된 JUMP 횟수만큼 수행되고 모든 JUMP 동작이 수행되면 다시 PC를 사용해 다음 명령어를 패치한다.

3.2 PIM 연산 수행

명령어가 패치되면 Control Unit을 통해 명령어가 수행된다. Control Unit은 연산에 필요한 데이터를 register와 bank에서 읽어오고 읽어온 데이터를 Compute Unit으로 보낸다. 이후 계산이 끝난 결과를 register(grf register)에 다시 저장한다.

3.3 PCIe overhead

PIM 연산을 수행하기 전에 연산에 사용할 데이터를 crf, srf 등의 register에 저장해야 한다. crf, srf 등의 register는 FPGA

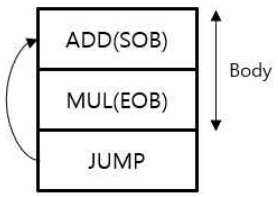

그림 3 JUMP Body 예시. JUMP 동작은 SOB와 EOB를 기준으로 수행된다.

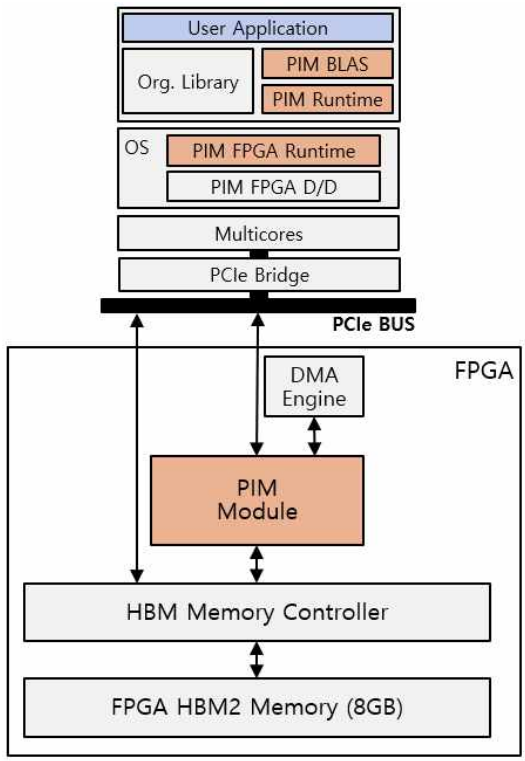

그림 4 PIM 연산을 위한 Software Stack. 색이 있는 부분이 추가/수정한 부분이다.

쪽에 있는 register이다. 따라서 데이터를 저장하기 위해 FPGA로 write command를 보내야 한다.

호스트 CPU와 FPGA 간의 command 전달을 할 때 PCIe를 사용하는데 이때 PCIe의 느린 전송속도로 인한 overhead가 발생한다. 이를 해결하기 위해 DMA를 사용해서 데이터를 전송한다.

3.4 DMA(Direct Memory Access)

PCIe overhead를 해결하기 위해 DMA를 사용한다. 하지만 적은 데이터를 전송할 때 DMA를 사용하면 성능이 좋지 않다. 예를 들어 512 bytes 이하의 데이터를 전송할 때의 속도는 4k 이상의 데이터를 전송할 때보다 7배 이상 느리다.

PIM-HBM[1]에서는 일반 DRAM command로 PIM 연산을 수행하기 위해 bank의 row/column 주소를 사용해 각 register에 데이터를 저장하고 하나의 command로 32 bytes의 데이터를 저장한다. 따라서 모든 register에 데이터를 저장하기 위해 상당히 많은 command가 필요하고 모든 command는 PCIe를 통해 전송된다. 이처럼 하나의 command로 32 bytes의 데이터를 저장하면 PCIe overhead가 더 커지고, 한 번의 command로 데이터를 전송할 때 적은 데이터(32 bytes)를 보내기 때문에 DMA의 장점도 활용할 수 없다.

그리고 software overhead도 발생한다. DMA를 사용할 때 FPGA로 데이터를 전송하기 위해서 시스템 콜 lseek()와 write()를 사용한다. 데이터를 나눠서 보낼 경우 32 bytes의 데이터를 보낼 때마다 lseek()와 write()를 호출한다. 따라서 매번 시스템 콜을 처리하기 위한 software들이 호출되고 이로 인한 software overhead가 발생한다. 따라서 시스템 콜을 한번 호출할 때 많은 데이터를 보내서 software에 의한 overhead를 줄일 필요가 있다.

이를 해결하기 위해 PIM 연산에 사용할 데이터를 패킹해서 전송한다. 패킹할 데이터는 crf, grf, srf, bank address register에 저장할 데이터, bank mode change, PIM 연산 실행 등 연산에 필요한 모든 데이터이다. 예를 들어 (1024 x 4096)의 GEMV 연산을 PIM을 사용해서 수행하려고 할 때 (1024 x 4096)의 GEMV 연산에 필요한 모든 데이터를 패킹해서 전송한다.

3.5 Bank Address Register

PIM-HBM[1]에서는 데이터를 저장할 때 만이 아니라 PIM 연산을 수행할 때도 주소 정보를 사용한다. 하지만 데이터를 패킹해서 한 번에 전송할 때는 하나의 command로 모든 데이터를 전송하기 때문에 PIM 연산에 필요한 주소 정보가 없다. 따라서 PIM 연산에 필요한 주소 정보를 저장할 register를 추가해서 사용한다. Bank Address Register에는 PIM을 사용한 연산 시 사용할 주소 정보를 저장하고 연산 시 저장된 주소 정보를 사용해서 연산을 수행한다.

4. PIM 연산을 위한 Software Stack

연산을 PIM으로 수행하기 위해서는 PIM 구조설계에 맞게 전달할 메모리 command를 세밀하게 설계할 필요가 있다. 그리고 PIM 구조에서 해당 연산을 수행하기 위해 PIM 메타데이터를 생성해야 한다.

이를 통합해서 수행할 수 있도록 PIM Software Stack을 구성하였으며, 범용성 있는 Library인 PyTorch에서 지원가능하도록 설계하였다. 지원 가능한 연산은 대표적인 연산인 ADD, MUL, GEMV 이다.

PIM Runtime은 PIM 연산에 필요한 동작들을 PIM Architecture에서 수행할 수 있도록 기본적인 함수를 제공한다. PIM BLAS는 PIM Runtime에서 제공하는 함수를 호출하고, PIM Runtime은 최종적으로 PIM FPGA Runtime의 함수를 호출하여 PIM 동작에 필요한 command를 FPGA로 보낼 수 있다. Software Stack의 전체 구조는 그림 4에 나와 있다.

5. 실험 환경 및 결과

5.1 실험환경

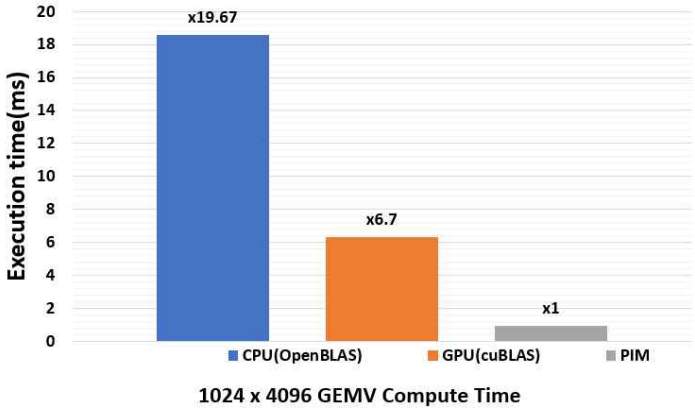

그림 5 실험결과 그래프. CPU(OpenBLAS), GPU(cuBLAS), 그리고 U280 보드에서 구현한 PIM(Packed data), PIM(No packed data)을 사용해서 (1x1024) x (1024x4096) GEMV 연산에 걸린 수행시간을 비교한다.

실험에 사용된 FPGA 보드는 U280 이고, CPU는 Intel(R) Core i5-6500, GPU는 NVIDIA TITAN V이다. CPU 실험은 openBLAS를 사용했고 GPU 실험은 cuBLAS를 사용했다.

5.2 실험결과

그림 5에 나온 것처럼 PIM(Packed data)을 사용해서 GEMV 연산을 했을 때 가장 성능이 좋았다. GPU와 CPU로 GEMV연산을 했을 때 각각 PIM을 사용했을 때 보다 6.7, 19.67배 느렸다. CPU와 GPU를 사용했을 때는 메모리로부터 데이터를 CPU, GPU까지 가져와서 연산을 한다. 하지만 PIM의 경우에는 메모리 근처에 있는 PIM Unit에서 연산을 하기 때문에 데이터를 가져오기 위한 overhead가 적다. 따라서 CPU, GPU에서 연산을 하는 것 보다 성능이 좋다.

6. 결론 및 향후 연구

본 논문에서는 최근 머신 러닝 모델의 워크로드를 계산할 때 발생하는 메모리 대역폭 문제를 해결하기 위해 PIM 구조를 설계했다. 사용한 PIM 구조는 PIM-HBM[1]을 사용했고 FPGA를 사용해서 구현했다.

구현된 PIM을 사용해서 (1x1024) x (1024x4096) GEMV 연산을 했을 때 CPU와 GPU를 사용해서 연산을 했을 때보다 각각 x6.7, x19.67배 빨랐다.

실제 머신 러닝 모델을 사용할 때 PIM 연산을 적용시키는 것이 향후 연구할 과제이다. 그리고 머신 러닝 모델을 사용할 때 호스트 CPU에서 PIM 장치를 사용하기 위한 워크로드와 자체적으로 계산할 워크로드를 분리하여 CPU를 최대한 활용할 수 있도록 스케줄링을 하는 것도 향후 연구할 과제이다.

7. 참고문헌

[1] S. Lee et al. 2021 . Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology. The International Symposium on Computer Architecture (ISCA).

[2] C. H, Kim, W. J. Lee, Y. Paik, K Kwon, S. Y. Kim, I. Park, and S. W. Kim. 2021. Silent-PIM: Realizing the Processing-in-Memory Computing with Standard Memory Requests. IEEE Transactions on Parallel and Distributed Systems (TPDS).

[3] A. Nag and R. Balasubramonian. 2021. OrderLight: Lightweight memory-ordering primitive for efficient fine-grained PIM computation. The IEEE/ACM International Symposium on Microarchitecture (MICRO).