Processing-in-Memory 연산을 위한 메모리 커맨드 생성 Hardware

장재혁○, 최성헌, 신동군
성균관대학교 전자전기컴퓨터공학과
jjh4777@g.skku.edu, csh9580@skku.edu, dongkun@skku.edu

Memory Command Generation Hardware for Processing-in-Memory Operations

Jaehyuk Jang○, Sunghern Choi, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

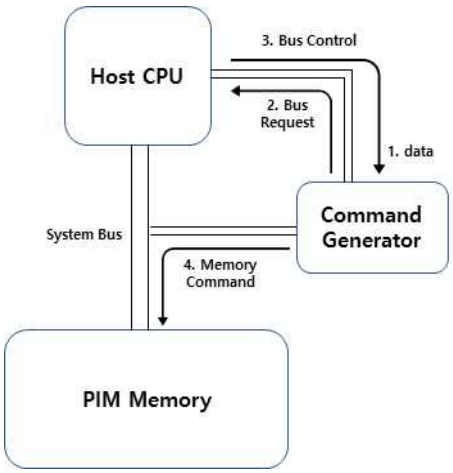
최근 다양한 분야에서 사용되는 머신 러닝 모델은 많은 양의 데이터와 연산량이 필요함에 따라, 메모리 내부에서 데이터 연산을 수행하도록 하여 CPU로의 데이터 이동을 최소화하는 PIM(Processing In-Memory)기법이 활발히 연구되고 있다. 그중 상용 DRAM을 기반으로 한 PIM-HBM은 PIM용 소프트웨어를 통해 특정 연산을 PIM에서 수행할 수 있도록 메모리 명령어들을 생성하여 PIM 메모리에게 전달해야 한다. 하지만 각 PIM 동작마다 필요한 메모리 명령어를 생성하여 전달하는 것은 Overhead가 발생하게 되며 CPU의 효율성을 감소시킨다. 본 논문에서는 이러한 문제를 해결하기 위해 메모리 명령어들을 자동적으로 생성하고 PIM 메모리에 전달하는 Hardware인 PIM Command Generator를 제안한다. 실험 결과 Command Generator를 사용했을 때 기존 보다 약 30%의 수행 시간 감소를 보여준다.

1. 서 론

최근 머신 러닝 모델들은 컴퓨터 비전, 자연어 처리 등 다양한 분야에서 사용되며 더 나은 성능을 위해 대량의 데이터와 계산량이 필요하다. 이에 따라 프로세서의 컴퓨팅 성능이 대폭 증가하면서, 많은 양의 데이터를 메모리로부터 읽어와야 하는데 이로 인해 메모리 병목현상이 생기는 문제가 발생하게 되었다. 최근 이러한 병목현상을 해소하기 위해 메모리 내부에서 데이터 연산을 수행하여 프로세서와 메모리 간의 데이터 이동을 최소화하는 PIM (Processing In-Memory) 기법이 활발히 연구되고 있다.

기존에 PIM 연구 중에는 대표적으로 DRAM 명령어를 수정하지 않고 그대로 사용하여 기존 상용 DRAM 기반으로 PIM 연산이 가능하도록 제안된 PIM-HBM[1]이 있다. PIM-HBM은 PIM용 소프트웨어를 통해 특정 연산을 PIM에서 수행하기 위해 필요한 정보들을 생성한다. 또한 생성된 정보를 바탕으로 메모리 명령어를 생성하고 정의된 연산을 수행할 수 있도록 알맞은 순서로 명령어를 PIM 메모리에게 전달하여 PIM을 동작시킨다. 그러나 PIM을 통해 메모리 내부에서 연산을 동작시키는 동안 호스트 CPU에서 작업이 이루어지지 않는 것을 지적한 연구[2]가 있으며 각 PIM 동작마다 필요한 메모리 명령어를 CPU에서 생성하고 전달하는 작업이 오버헤드를 발생시키고 효율성을 감소시키는 문제가 발생하고 있다. 이를 해결하기 위해 본 논문에서는 메모리 명령어들을 자동으로 생성하고 PIM 메모리에게 전달하는 하드웨어인 PIM Command Generator (PIM-CG)를 제안한다. PIM-Command Generator는 FPGA를 기반으로 설계한 가속기를 기존의 PIM-HBM과 통합하여 머신 러닝 모델의 워크로드를 가속화할 수 있도록 진행된 연구[3]와 유사하게 메모리 Command를 생성하고 전달하는 작업을 담당하는 하드웨어를 FPGA기반으로 설계하여 PIM-HBM과 통합한다.

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)



[그림 1] PIM Command Generator 다이어그램

PIM-CG는 현재 많이 쓰이고 있는 DMA (Direct Memory Access) 시스템이 CPU를 대신하여 지정된 메모리 공간과 I/O 장치 간의 데이터 이동을 담당하는 것과 유사하다. DMA와 같이 동작하도록 함으로써 CPU가 처리해야 할 작업들을 Command Generator가 수행하고 CPU 효율성을 높일 수 있다.

2. PIM-HBM 동작 방식

PIM-HBM[1]은 HBM (High Bandwidth Memory) 기반으로

[표 1] CMD_INFO 정보

Command INFO	저장된 값
OP Code Target(TG)	PIM OP Register의 Index
Addr TG	Addr Register의 Index
Data TG	Data Register의 Index
N_cmd	현재 CMD_INFO의 반복 횟수
Addr TG Step	Addr Register의 Index 증가 값
Data TG Step	Data Register의 Index 증가 값

제안된 PIM 구조로 표준 DRAM의 수행을 위한 Single-Bank 모드 외에 PIM 수행을 위한 All-Bank, All-Bank-PIM 모드를 추가로 지원하며 DRAM controller의 변경 없이 기존 DRAM 명령어 만으로 동작이 가능하다. 또한 PIM용 Software Stack은 특정 연산을 PIM에서 동작시키기 위한 연산 커널(Micro-Kernel)을 생성하고 데이터 접근 순서에 맞게 메모리의 알맞은 위치에 데이터를 저장한다. 최종적으로 연산 정보를 통해 메모리 명령어를 생성하여 PIM 연산을 수행하도록 한다.

PIM-HBM에서 특정 연산을 수행하는 자세한 과정은 아래와 같다. ① Single-Bank mode에서 연산 데이터들을 각 Bank 및 PIM 연산 장치의 Register에 저장한다. ② All-Bank mode로 전환하여 특정 연산에 대한 Micro-Kernel을 각 PIM 연산 장치의 CRF (command register files) Register에 저장한다. ③ All-Bank-PIM mode로 전환하여 Micro-Kernel의 각 PIM Instruction에 대응하는 메모리 접근 명령을 호스트에서 생성하고 메모리에 전달하여 PIM Instruction을 하나씩 수행시킨다.

즉 특정 연산을 PIM으로 수행하기 위해서는 ⓐ 뱅크 간 모드 전환을 위한 명령어, ⓑ 연산 데이터를 write 하는 명령어, ⓒ Micro-Kernel을 저장하는 명령어, ⓓ 연산을 trigger 하는 명령어 등 특정 연산을 실행할 때 마다 다수의 명령어를 전달해야 한다. 이것은 PIM 연산 시간에 오버헤드가 되고 있으며 오버헤드를 최소화하고 CPU 효율성을 증가시키기 위해 CPU 외부에서 동작하는 Command Generator를 제안한다.

3. PIM Command Generator

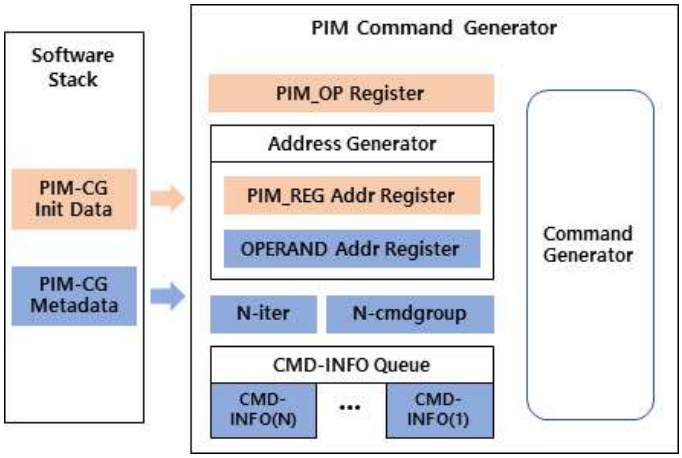
PIM-CG는 호스트에서 메모리 명령을 생성하는 역할을 대신한다. PIM-CG가 동작하는 방식은 그림 1과 같다. ① 필요한 데이터를 pre-load시킨다. ② System bus를 공유하는 PIM-CG가 CPU에게 Bus Request를 보낸다. ③ PIM-CG는 CPU로부터 Control을 넘겨받는다. ④ 사전에 전달받은 메타데이터를 통해 명령어를 생성하여 메모리로 전달하는 작업을 수행한다. 우리는 이 절에서 PIM command generator의 메타데이터와 구조 및 동작 과정 자세하게 설명한다.

3-1. PIM Command Metadata

PIM Command Metadata는 PIM용 Software Stack을 통해 생성되는 특정 연산에 대한 정보들이다. 예를 들어 수행할 연산의 종류, 연산의 크기, Input이 메모리에 저장되어 있는 위치 등이 해당된다. 우리가 제안한 PIM-CG에 전달되는 메타데이터는 데이터가 저장된 메모리 주소 및 연산 커널을 수행하기 위한 정보로 이루어져 있다. 연산 커널 정보는 특정 연산 커널의 반복수(N_iter), 메모리 명령어에 대한 정보(CMD_INFO), 메모리 CMD_INFO의 개수(N_cmdgroup)를 포함한다. 메모리 명령어 정보인 CMD_INFO는 표 1과 같은 6가지 정보를 포함한다.

PIM-CG가 메모리 명령어를 생성하기 위해서는 PIM Command Metadata와 함께 PIM Initialize Data가 필요하다. PIM Initialize Data는 PIM 장치의 H/W 정보로 PIM 연산 장치의 레지스터 주소와 Op Code와 같은 PIM 명령어 정보를 가진다.

CPU가 메타데이터와 initilize data를 Command Generator



[그림 2] PIM Command Generator 레지스터 구조

에게 전달하면 전달된 정보들은 그림 2와 같이 각 레지스터에 저장된다. Command Generator는 저장된 정보들을 바탕으로 메모리 명령어를 생성하며 System Bus를 통해 메모리 컨트롤러로 전달한다.

3-2. PIM Command Generator 구조

PIM Command Generator의 하드웨어 구조는 그림 2와 같이 메타데이터를 저장하는 레지스터, 필요한 주소를 제공해주는 Address Generator, 주어진 정보를 통해 메모리 명령어를 생성 및 전달하는 Command Generator로 구성되어 있다.

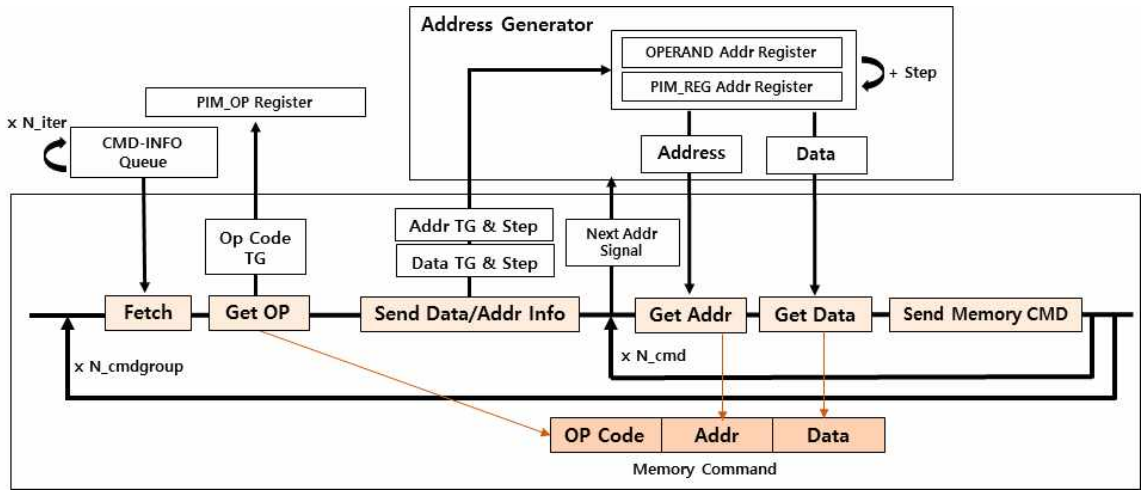
메타데이터를 저장하는 레지스터들은 메타데이터의 종류에 따라 구분되어 있으며 OPERAND_Addr 레지스터와 N_iter, N_cmdgroup 레지스터로 구분된다. 생성할 메모리 명령어에 대한 정보를 가진 CMD_INFO는 CMD_INFO 큐에 저장되어 있다. CMD_INFO는 입력받은 순서에 맞게 First-In-First-Out(FIFO) 방식으로 명령어를 생성하기 위해 큐에 저장된다. 또한 Initialize data를 저장하기 위한 PIM_OP 레지스터, PIM_REG Addr 레지스터가 추가로 존재한다.

Address Generator는 피연산자 데이터의 위치를 저장하고 있는 OPERAND Addr 레지스터, PIM_REG Addr 레지스터가 존재한다. Address Generator는 Command Generator가 명령어를 생성할 때 필요한 데이터와 주소를 전달한다. 이를 위해, 메모리에 WRITE할 데이터를 저장하는 Data 레지스터를 두고 Command Generator가 필요한 주소와 데이터를 요청하면 반환하는 작업을 수행한다.

3-3. PIM Command Generator 동작

PIM-CG는 레지스터에 저장되어 있는 PIM 동작을 위해서 생성된 메타데이터를 이용하여 메모리 명령어를 생성하기 위해 일련의 과정을 수행한다.

① Fetch 단계는 CMD_INFO 큐로부터 수행할 CMD_INFO를 FIFO 방식으로 가져온다. ② Get OP 단계에서는 CMD_INFO의 OP Code TG 값을 통해 PIM OP 레지스터로부터 OP Code를 가져온다. ③ Send Data/Addr Info 단계에서 Command Generator가 Address Generator에게 Data/Addr TG 값과 Step 값을 통해 요청할 주소와 데이터의 메타데이터를 전송한다. ④ Send Next Addr Signal 단계는 Address Generator에게 메모리 커맨드 생성에 사용될 Addr, Data 값을 요청한다. ⑤ Get Addr 단계는 Address Generator가 전달받은 Addr TG 값을 기반으로 OPERAND 또는 PIM_REG Addr Register로부터 알맞은 Address를 반환한다. 예를 들어, Addr TG 값이 PIM_REG[2]로 설정되면, 2번째 PIM_REG Addr Register에 저장되어 있는 주소를 반환한다. ⑥ PIM Get Data는 Data TG 값을 기반으로 Data Register를 통해 데이터를 전



[그림 3] PIM Command Generator 수행 과정

달받는다. ⑦ 마지막으로 생성 완료된 메모리 명령어를 PIM 메모리에게 전달하는 Send Memory CMD를 수행한다. 일련의 과정은 CMD_INFO 큐의 모든 CMD_INFO를 순차적으로 수행하기 위해 N_cmdgroup 만큼 반복된다.

모든 CMD_INFO들을 처리한 이후 피연산자의 크기에 따라 제한된 PIM 연산 장치로 인해 연산 작업을 반복해야 하는 경우가 있다. 이러한 경우 CMD_INFO Queue의 처음부터 N_iter 횟수만큼 반복 수행할 수 있다. 또한 같은 명령어를 반복 수행해야 하는 경우에 PIM-CG는 N_cmd 정보를 통해 Send Next Addr Signal 단계부터 메모리 생성까지 반복할 수 있도록 설계되었다. 그림 3에서는 해당 과정에 동작 예시를 보여준다.

3. 실험 환경 및 결과

PIM-CG의 성능을 측정하기 위해 PIM-CG를 사용하지 않았을 때와 실험 결과를 비교한다. 실험은 기존에 FPGA로 PIM 동작을 구현한 연구[4]와 유사하게 Xilinx Alveo U50 FPGA를 활용하여 PIM-HBM구조를 구현한 emulator를 사용하고 Command Generator 또한 FPGA를 사용하여 emulator와 통합하여 실험환경을 구축하였다. 특정 연산을 PIM으로 수행하기 위해 PIM Software Stack을 통해 지원되는 연산 중 GEMV 연산을 기준으로 실험을 진행한다.

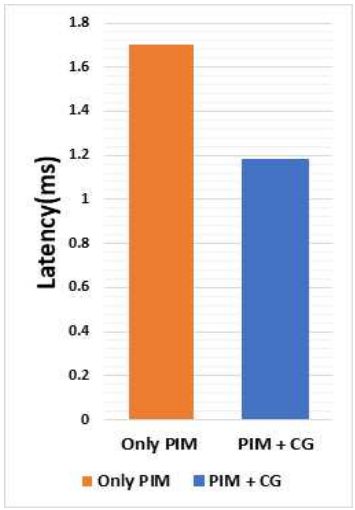
그림 4는 1024 X 4096 크기의 GEMV 연산을 기존 PIM에서 사용했을 때와 Command Generator를 적용하였을 때를 비교한다. 결과적으로 PIM-CG를 사용했을 때, 기존보다 약 30% 연산 시간이 감소하였다.

이는 기존 CPU에서 메타데이터를 명령어에 대해 한 번씩 읽어와 명령어를 생성/전달하였지만 구현한 H/W에서는 연속적으로 메타데이터에 대해 명령어를 생성하고 전달함에 의한 성능 향상이라고 볼 수 있다.

4. 결론 및 향후 연구

본 논문에서는 최근 머신 러닝 모델의 메모리 집약적인 연산을 PIM으로 수행할 때 CPU에서 생기는 오버헤드를 문제점으로 인식하였다. 오버헤드는 메모리 Command를 생성하고 전달하는 과정에서 발생하며 우리가 제안한 PIM-CG는 CPU로부터 Command 생성에 필수적인 metadata만을 전달받아 Command 생성 및 전달을 대신할 수 있는 하드웨어인 PIM-CG를 제안하였다. PIM-CG는 FPGA를 기반으로 설계되었으며 PIM과 통합했을 때 DMA 시스템과 유사한 방식으로 동작하도록 구현하였다. 그에 따라 CPU 효율성을 증가시켰고 연산을 수행할 때 걸리는 Latency를 효과적으로 줄일 수 있었다.

메모리 집약과 연산 집약이 동시에 요구되는 최신 머신 러닝 모델 관점에서 이와 같은 결과는 연산 시간을 개선하는 데에



[그림 4] 1024 X 4096 GEMV Test

있어서 굉장히 긍정적인 결과이며 앞으로 Command 생성뿐만 아니라 PIM 내부에서 처리하기 어려운 추가적인 기능을 PIM-CG에 구현하여 성능 개선을 할 수 있도록 하는 것도 향후 연구할 과제이다.

5. 참고문헌

[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] Nag, Anirban, and Rajeev Balasubramonian. "OrderLight: Lightweight memory-ordering primitive for efficient fine-grained PIM computations." MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. 2021.

[3] Kang, Shinhaeng, et al. "An FPGA-based RNN-T Inference Accelerator with PIM-HBM." Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2022.

[4] Kim, Chang Hyun, et al. "Silent-PIM: Realizing the processing-in-memory computing with standard memory requests." IEEE Transactions on Parallel and Distributed Systems 33.2 (2021): 251-262.