딥뉴럴네트워크를 위한 기능성 기반의 핌 가속기

김민재*, 김신덕**
*연세대학교 컴퓨터과학과
**연세대학교 컴퓨터과학과
mj0098@yonsei.ac.kr, sdkim@yonsei.ac.kr

Functionality-based Processing-In-Memory Accelerator for Deep Neural Networks

Min-Jae Kim*, Shin-Dug Kim**
*Dept. of Computer Science, Yonsei University
**Dept. of Computer Science, Yonsei University

요 약

4 차 산업혁명 시대의 도래와 함께 AI, ICT 기술의 융합이 진행됨에 따라, 유저 레벨의 디바이스에서도 AI 서비스의 요청이 실현되었다. 이미지 처리와 관련된 AI 서비스는 피사체 판별, 불량품 검사, 자율주행 등에 이용되고 있으며, 특히 Deep Convolutional Neural Network (DCNN)은 이미지의 특색을 파악하는 데 뛰어난 성능을 보여준다. 하지만, 이미지의 크기가 커지고, 신경망이 깊어짐에 따라 연산 처리에 있어 낮은 데이터 지역성과 빈번한 메모리 참조를 야기했다. 이에 따라, 기존의 계층적 시스템 구조는 DCNN 을 scalable 하고 빠르게 처리하는 데 한계를 보인다.

본 연구에서는 DCNN 의 scalable 하고 빠른 처리를 위해 3 차원 메모리 구조의 Processing-In-Memory (PIM) 가속기를 제안한다. 이를 위해 기존 3 차원 메모리인 Hybrid Memory Cube (HMC)에 하드웨어 및 소프트웨어 모듈을 추가로 구성하였다. 구체적으로, Processing Element (PE)간 데이터를 공유할 수 있는 공유 캐시 및 소프트웨어 스택, 파이프라인화된 곱셈기 및 듀얼 프리페치 버퍼를 구성하였다. 이를 유명 DCNN 알고리즘 LeNet, AlexNet, ZFNet, VGGNet, GoogleNet, RestNet 에 대해 성능 평가를 진행한 결과 기존 HMC 대비 40.3%의 속도 향상을 29.4%의 대역폭 향상을 보였다.

1. 서론

Deep Convolution Neural Network (DCNN)은 이미지 처리 AI 기술로서 이미지의 특색 파악에 큰 성과를 보여왔다. 최근 DCNN 에 이용되는 필터의 수, 신경망의 깊이가 증가함에 따라 많은 양의 합성곱 연산이 발생하였다. 또한, 이미지의 크기 증가와 Sparse 한 행렬의 처리로 인해 DCNN 의 연산은 낮은 데이터 지역성 및 빈번한 메모리 참조의 특징이 있다. 이에 따라, 기존의 계층적 시스템 구조는 '폰 노이만 보틀넥'으로 인하여 DCNN 을 scalable 하고 빠르게 처리하는데 한계를 보이고 있다.

합성곱 연산은 이미지를 행렬화 하여 각 행렬의 대응 원소를 곱하고(Multiply), 그 값들을 더하여(Accumulate) 합하는 MAC (Multiply-Accumulate) 연산을 이용한다. DCNN 에는 이러한 MAC 연산 외에 ImagePatch, Resize, Contraction 과 같은 독립적인 기능성(functionality) 함수 블록이 존재한다. 따라서, MAC 연산을 가속화하고 DCNN 의 기능성 함수 블록을 빠르게 처리할 수 있다면, DCNN 의 수행을 가속화할 수 있다.

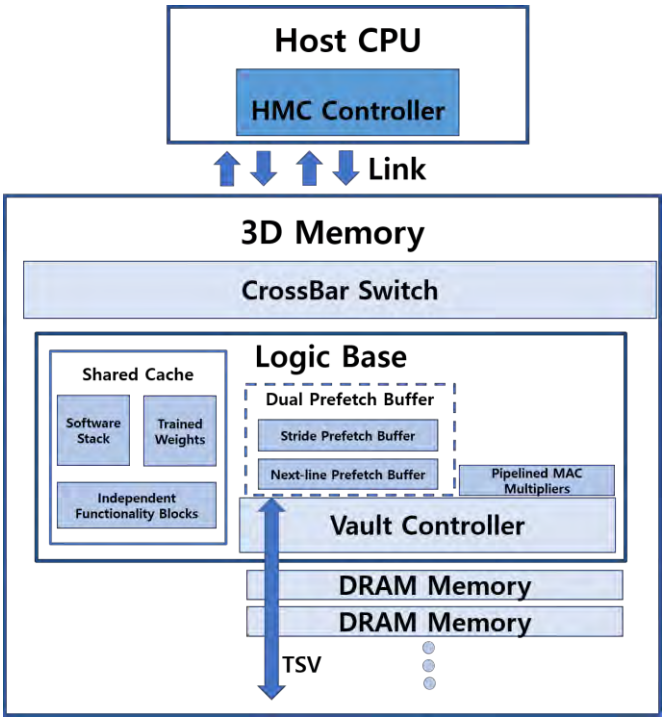
최근 TSV (Through-Silicon-Via)[1] 기술을 바탕으로 메모리 다이를 수직으로 적층하여 만든 3D 메모리는 Processing Element (PE)를 메모리 근처에 위치시켜 Processing-In-Memory (PIM)를 실현하였다. 이러한 PIM 구조는 대량의 데이터를 메모리에서 PE 로 빠르게 이동시켜 처리할 수 있어, 낮은 데이터 지역성 및 빈번한 메모리 액세스를 보이는 워크로드를 scalable 하고 빠르게 처리할 수 있는 차세대 아키텍쳐로 평가받고 있다. 하지만, 기존의 대표적인 3D 메모리인 Hybrid Memory Cube[2]는 다음과 같은 한계점이 있다.

(1) 호스트에서 PE 로의 연산 offload 과정
(2) Vault Controller 의 제한된 메모리 접근

이로 인해 HMC 는 비교적 큰 용량의 데이터 저장을 요구하는 DCNN 의 가속기로 이용되기에 한계가 있다. 본 연구에서는 HMC 를 DCNN 의 가속기로 이용하기 위해 몇 가지 하드웨어적/소프트웨어적 모듈을 추가로 제안하였다.

(1) HMC 의 PE 들이 데이터 및 함수를 공유할 수 있는 공유 캐시를 구성하였다.
(2) 공유 캐시를 이용하여 DCNN 의 여러 기능성 (functionality) 블록을 PE 에서 분산 처리하였다.
(3) 소프트웨어 스택을 구성하여 PE 가 자율적으로 명령어를 수행하고 다른 PE 를 INVOKE 할 수 있게 구현하였다.
(4) MAC 연산을 가속할 수 있도록 PE 에 파이프라인화한 곱셈기와 듀얼 프리페치 버퍼를 추가로 구성하였다.

이를 통해 낮은 데이터 지역성 및 빈번한 메모리 접근을 보이는 DCNN 을 가속하기 위한 기능성 기반의 PIM 가속기를 제안한다.



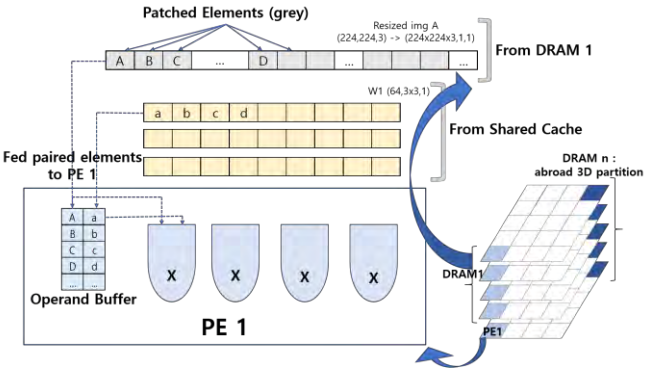
(그림 1) Overall Architecture

2. 기능성 기반의 핌 모델

본 연구에서는 기존의 HMC 구조에 하드웨어적/소프트웨어적 모듈을 추가 구성하여 DCNN 을 가속하는 아키텍처를 제안한다. 전체 구성은 [그림 1]과 같다.

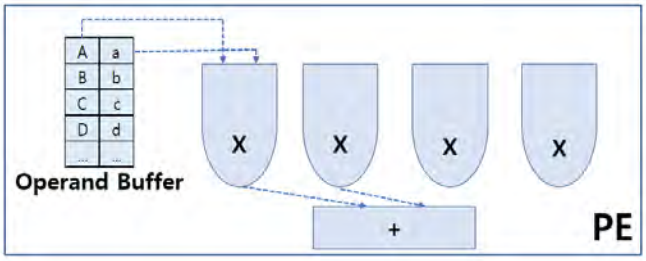
2.1 공유 캐시의 구성

3D 메모리의 로직 베이스에 공유 캐시를 구성하여 PE 간 데이터 및 함수 블록을 공유하였다. 먼저, 공유 캐시에는 DCNN 의 여러 독립적인 기능성 함수 블록이 구현되어 있어 이를 여러 PE 에서 분산 처리할 수 있도록 구현하였다. 예를 들어, PE1 에서는 ImagePatch 를 공유 캐시로부터 로드하여 수행하고, PE2 에서는 ImageContraction 을 로드하여 수행하여 DCNN 을 빠르게 처리한다.



(그림 2) 공유 캐시를 이용한 합성곱 연산 수행 과정

또한 공유 캐시에는 딥뉴럴네트워크 (DNN)의 훈련 과정에서 고정된 웨이트를 저장하여, 기존 HMC 의 PE 가 제한된 3D 메모리 파티션에 대해서만 접근 가능한 한계를 해결하였다. 예를 들어, [그림 2]와 같이 DRAM1 에 저장된 이미지 A (img A)가 DCNN 의 Layer1 의 합성곱 연산을 PE1 에서 수행할 때, PE1 은 DRAM1 로부터 img A 벡터의 원소를, 공유 캐시로부터 Layer1 의 웨이트에 해당하는 W1 을 로드 하여 내부의 곱셈기로 feed 하는 방식으로 동작한다.



(그림 3) Pipelined Multipliers

2.2 Pipelined Multipliers

본 연구는 HMC-MAC[3] 아키텍처를 베이스라인으로 지정하여 모델을 구성하였다. HMC-MAC 은 기존의 HMC 의 Vault Controller 에 MAC 연산을 가속하기 위한 MAC Operator 를 추가한 구조로 HMC 를 cycle accurate 하게 구현하였다. HMC-MAC 은 부동소수점 곱셈을 4cycle 에 덧셈을 1cycle 에 처리한다. 본 연구에서는 이를 개선하여 [그림 2]와 같이 PE 에 4 개의 파이프라인화된 곱셈기를 구성하였다. 이를 통해 MAC 연산을 1cycle 에 처리할 수 있다.

2.3 Dual Prefetch Buffer

3D integrated Memory 구조 상, 복잡한 모듈 구성은 thermal problem 을 야기할 수 있다. 본 연구에서는 이 점을 고려하여 메모리 레이턴시는 낮추되 최대한 간단한 프리페처를 구성하였다. 이에따라 각각의 PE 에 총 4KB 의 Next-line 프리페처와 Stride 프리페처를 듀얼로 구성하였다. Next-line 프리페처는 현재 페치된 주소의 다음 8 개 라인을 페치하며, Stride 프리페처는 이전 메모리 접근 주소와 현재 주소의 차이, 즉 Stride 를 관찰한 뒤 Stride 만큼 프리페치 한다.

2.4 Software Stack

공유 캐시에 소프트웨어 스택을 구현하여 기존 HMC 의 호스트 프로세서로부터 명령어를 offload 받아 PE 가 동작하는 방식이 아닌, PE 유닛에서 자율적으로 리퀘스트의 처리가 가능하도록 구성하였다. 소프트웨어 스택에 구현된 Function Call 은 [표 1]과 같다. 이를 바탕으로 PE 는 호스트 프로세서의 동작과 상관없이 자체적으로 리퀘스트를 처리하고, 다른 PE 를 INVOKE 하여 명령어의 할당, 수행 및 offload 를 지시할 수 있다.

<표 1> 소프트웨어 스택에 구현된 함수

AllocReq()	볼트로 리퀘스트를 할당
ReceiveReq()	리퀘스트를 할당받아 해석하여 기능성 연산 블록 수행 시작
ProcessReq()	별도의 Block 이 없을 시까지, 기능성 연산 블록 처리
OffloadBack()	리퀘스트를 다른 볼트에서 분산 처리하기 위해, 연산 블록을 크로스바 스위치로 전달
WriteToCache()	캐시에 데이터 쓰기

3. 실험 및 결과

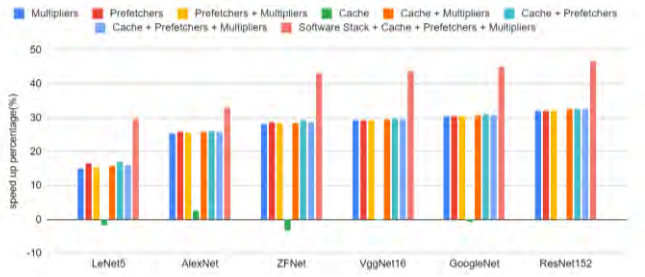
3.1 실험 설계

본 연구에서 제안하는 아키텍처의 성능을 평가하기 위해 HMC-MAC 시뮬레이터를 베이스라인으로 시뮬레이터를 구성하였다. 트레이스를 추출하기 위해 Intel pin tool[4]을 이용해 메모리 접근 시 참조 주소 및 MAC operator count 에 관한 정보를 추출했다. 또한, 머신러닝 라이브러리 Tensorflow[5]에서 합성곱 연산의 기능성 블록들을 Region Of Interest (ROI)로 지정하여 해당 영역을 PE 에서 분산처리할 수 있도록 맵핑하였다. 사용된 DCNN 모델은 유명 DCNN 알고리즘으로, LeNet[6]의 경우 28x28x1 사이즈의 손글씨 데이터를 zero padding 하여 32x32x1 로 만들어 수행하였고, AlexNet[7], ZFNet[8], VGGNet[9], GoogleNet[10], ResNet[11]의 경우 224x224x3 사이즈의 Imagenet[12] 벤치마크를 이용하였다. DCNN 의 특징은 [표 2] 와 같다.

<표 2> 사용된 DCNN 알고리즘

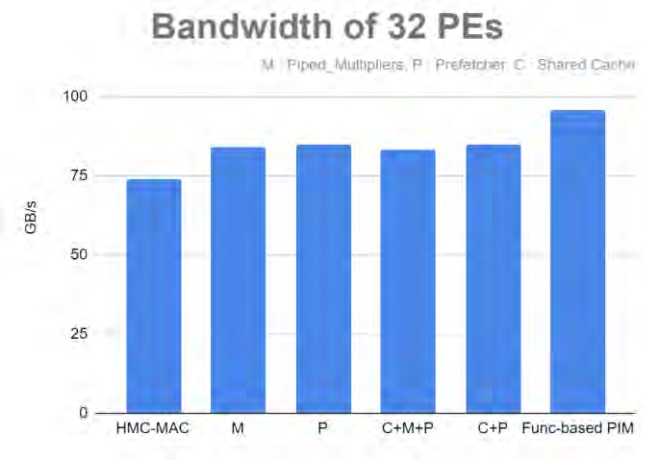
DCNN	# of Layers	input sizes
LeNet	5	32x32x1
AlexNet	8	224x224x3
ZFNet	8	224x224x3
VGGNet	16	224x224x3
GoogleNet	22	224x224x3
ResNet	152	224x224x3

3.2 Speed up



(그림 4) DCNN 워크로드별 각 모델의 속도 향상

HMC-MAC 을 베이스라인으로 각 제안하는 4 가지 모듈을 추가하였을 때, DCNN 워크로드별 모델의 속도 향상 정도는 [그림 4]와 같다. 마지막 막대인 분홍색 막대는 HMC-MAC 에 본 연구에서 제안하는 4 가지 하드웨어적/소프트웨어적 모듈을 추가로 구성한 것으로, 6 가지 DCNN 워크로드에서 평균 40.3%의 속도 향상을 보였다. 또한, 총 5 개 레이어로 구성된 LeNet 에서부터 152 개의 레이어로 구성된 ResNet 까지 레이어의 깊이가 깊어질수록 속도 향상이 더 높아지는 것을 확인할 수 있다. 이는 4 가지 모듈을 추가했을 때, HMC-MAC 에 비해 더욱 scalabe 한 처리를 하는 것에 기인한다.



(그림 5) GoogleNet 수행 시, 32 개 PE 에서의 대역폭

3.3 Bandwidth

GoogleNet 수행 시, 본연구에서 제안된 모델들의 32 개 PE 의 대역폭은 (그림 5)와 같다. HMC-MAC 에 4 개 모듈을 추가한 가장 마지막 막대 그래프에서 95.6GB/s 로 기존 HMC 대비 약 29.4%의 대역폭 향상을 보였다.

4. 결론

본 연구에서는 낮은 데이터 지역성 및 빈번한 메모리 참조의 특징을 보이는 Deep Convolutional Neural Network 를 가속하기 위한 구조로 기능성 기반의 Processing-In-Memory 아키텍처를 제안하였다. 기존 PIM 구조의 한계를 극복하기 위해, PE 들의 데이터 및 함수를 공유할 수 있는 공유 캐시를 구성하였다. 공유 캐시에는 Trained-Weight 및 DCNN 의 독립적인 기능성 함수 블록이 저장되어 있어, 3D 메모리의 데이터 접근 제한 해결 및 PE 단위에서 기능성 함수 블록의 분산 처리를 구현하였다. 또한 MAC 연산을 가속하기 위해 PE 에 Pipelined Multiplier 및 Stride Prefetcher, Next-line Prefetcher 로 구성된 Dual Prefetch Buffer 를 구성하였다. 이를 통해 기존의 HMC 에 비하여 최종 제안된 모델에서 40.3%의 속도 향상, 29.4%의 대역폭 향상을 달성하였다. 또한, LeNet, AlexNet, ZFNet, VGGNet, GoogleNet, ResNet 으로 점점 깊어지는 DCNN 레이어에 대해 점점 더 빠른 속도 향상을 보였다. 이를 통해, 본 연구에서 제안하는 기능성 기반의 핌 가속기가 기존 HMC 에 비하여 Scalable 한 DCNN 연산 수행을 하는 것으로 평가할 수 있다.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2019R1A2C1008716).

참고문헌

[1] M. Motoyoshi "Throug-Silicon Via (TSV)" Proceedings of the IEEE, vol. 97, no. 1, pp. 43-48, Jan 2009.

[2] Hybrid Memory Cube Consortium (HMCC), Hybrid Memory Cube Specification 2.1, Online, 2016.

[3] D. Jeon, K. Park and K. Chung, "HMC-MAC: Processing-in Memory Architecture for Multiply-Accumulate Operations with Hybrid Memory Cube," in IEEE Computer Architecture Letters, vol. 17, no. 1, pp. 5-8, 1 Jan.-June 2018, doi: 10.1109/LCA.2017.2700298.

[4] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In PLDI '05: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 190–200, 2005

[5] Abadi, M. et al. TensorFlow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016; pp 265−283..

[6] Y. LeCun, LeNet-5 convolutional neural networks [J], Online, 2015.

[7] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.

[8] M. D. Zeiler and R. Fergus. Visualizing and understanding
convolutional networks. In Computer Vision–ECCV 2014, pages 818–833. Springer, 2014.

[9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern
Recognition, pages 770–778, 2016.