메모리 자원이 한정된 IoT 시스템에서의 DNN 실행을 위한 GEMM 타일링

조근혜 O, 이하윤, 신동군

성균관대학교 전자전기컴퓨터공학과

{shurui, lhy920806, dongkun}@skku.edu

GEMM Tiling for the DNN Inference on Memory Constrained IoT Devices

Geunhye Jo O, Hayun Lee, Dongkun Shin

Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

DNN(Deep Neural Network)은 높은 정확도를 달성하기 위해서 레이어와 채널이 많은 구조로 발전하였다. 이에 따라 메모리 사용량이 증가하여 메모리 자원이 한정된 IoT 기기에서는 직접 실행하는 것이 어려워졌다. 이러한 문제를 해결하기 위해 DNN 실행 시 동적으로 메모리를 관리하는 연구들이 진행되었으나, 개별 레이어에 필요한 메모리 사용량 이하로는 최대 메모리 사용량을 줄이지 못하는 한계가 있다. 본 연구는 딥 러닝 네트워크의 Convolutional 레이어 내에서 사용되는 GEMM(General Matrix Multiplication) 연산의 최대 메모리 사용량을 줄이는 GEMM 타일링(Tiling) 기법을 Arm Compute Library에 적용하고 최대 메모리 사용량 감소에 따른 실행 시간 변화를 분석하였다. 그 결과, GEMM 연산의 최대 메모리 사용량이 최대 17.3배 감소하였고 이때 실행 시간은 4.1배 증가하였다

1. 서 론

최근 DNN(Deep Neural Network)은 텍스트, 이미지, 영상 그리고 각종 센서 데이터를 이용해 원하는 결과를 추론하는 작업에서 높은 성능을 보이며 서버부터 모바일 기기까지 다양한 환경에서 유용하게 사용되고 있다.

정확도가 높은 DNN 모델은 채널과 레이어가 많은 구조를 가지므로 연산량과 메모리 사용량이 크다. 그런데 IoT 기기 등 메모리 자원이 한정된 환경에서는 DNN 모델의 메모리 사용량보다 여유메모리가 부족하여 모델을 실행할 수 없는 경우가 생긴다.

이를 해결하기 위해 기존에는 모델 압축, 클라우드 컴퓨팅 등의 방법이 주로 이용되었다. 모델 압축은 Pruning[1], Quantization[2] 등으로 모델의 크기를 줄이는 기법이다. 클라우드 컴퓨팅은 IoT 기기로부터 추론이 필요한 데이터를 고성능 클라우드 서버로 전송하여 연산을 수행하고 그 결과값을 내려 받는 방법이다.

하지만 위의 방법들에는 문제점이 존재한다. 모델 압축은 DNN 모델의 메모리 사용량을 큰 폭으로 낮출 수 있지만, 압축률이 커지면 모델의 예측 정확도도 낮아진다. 클라우드 컴퓨팅에서는 IoT 기기에서 클라우드로 데이터를 전송하여야 하는데, 전송에 드는 시간이 네트워크 상황에 따라 예측 불가능하게 지연된다. 또한 개인 정보가 유출될 가능성이 있어 보안상 문제가 생긴다.

DNN 모델의 정확도를 감소시키지 않으면서도 IoT 기기 안에서 실행하기 위해서 메모리 자원을 관리하는 연구가 진행되었다[3]. 이러한 연구에서는 연산에 필요한 데이터를 스토리지로부터 미리 로드하고 필요 없는 데이터는 메모리로부터 해제하는데, Convolution 연산이나 Pooling 연산 등 개별 레이어에 필요한 만큼의 메모리를 최소 단위로 관리할 수 있다. 이러한 연구에서는 한 레이어를 수행하기 전에 필요한 모든 메모리 영역을 할당하기 때문에, 한 레이어를 연산하는 데에 필요한 메모리 사용량 이하로는 최대 메모리 사용량을 줄일 수 없다는 한계가 있다.

본 연구는 ARM 프로세서용 머신 러닝 라이브러리인 ArmCL(Arm Compute Library)[6]에 GEMM(General Matrix Multiplication) 연산을 나누는 기법[4]을 적용하여, GEMM 연산의 최대 메모리 사용량을 감소시켰다. 실험에서는 GEMM 연산의 최대 메모리 사용량 감소를 확인하고, 이에 따른 실행 시간 변화를 분석하였다. 그 결과 최대 메모리 사용량을 기존 GEMM 연산보다 최대 17.3배 감소하였으며, 이때 연산 시간은 4.1배 증가하였다.

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)

2. DNN 실행 시 메모리 관리

DNN 을 활용하여 정확도가 높은 추론을 하기 위해서는 레이어와 채널이 많은 구조를 사용하여야 한다. 큰 구조의 DNN 모델을 사용하면 연산량 및 메모리 사용량이 증가한다. 이에 따라 서버에서 IoT 기기에 이르기까지 DNN 모델 트레이닝 및 추론에 있어 컴퓨팅 자원의 제약이 문제가 된다. 특히 여유 메모리가 모델의 최대 메모리 사용량보다 부족한 환경에서는 수행 자체가 불가능하므로, 최대 메모리 사용량을 줄이기 위한 연구들이 진행되었다.

[5]는 DNN 트레이닝에 있어 세 가지 특징적인 기법을 사용하여 최대 메모리 사용량을 줄인다. 먼저, Liveness 분석을 통해 이후에 쓰이지 않을 tensor 를 알아내어 해제한다. 그리고 CPU 와 GPU 메모리를 통합적으로 관리하여, 추후에 활용될 tensor 를 CPU 메모리로 offloading 하고 곧바로 활용될 tensor 를 GPU 메모리로 prefetching 한다. 마지막으로 pooling 과 같이 비용이 적은 연산의 결과는 남겨두지 않고 버림으로써 GPU 메모리 사용량을 감소시킨다.

[3]은 정확도가 높은 DNN 모델을 모바일 기기 내에서 수행하기 위해 연산 직전에 필요한 tensor 를 code block 으로써 메모리로 올리고 쓸모가 없어진 code block 을 메모리에서 제거한다.

위의 연구들은 DNN 모델 전체에서 필요한 메모리를 나누어 관리하지만 그 최소 단위는 레이어이다. 그러므로 연산에 사용되는 tensor 가 큰 레이어가 최대 메모리 사용량을 결정짓게 되는 한계점이 있다.

3. GEMM 타일링

머신 러닝에 사용되는 GEMM 연산은 규칙적이고 높은 locality 를 가진다. GEMM 연산에 사용되는 matrix 들을 그림 1 과 같이 타일로 나누어 부분적인 GEMM 을 수행하고, 합하여 전체 GEMM 과 같은 결과를 내도록 하면 어떤 부분의 연산에 사용된 타일이 다른 부분의 연산에도 반복적으로 이용된다. 그러면 다른 부분을 연산할 때 다음 연산을 위해 새롭게 prefetch 할 메모리 크기가 줄어드므로, In-memory 연산에 가까운 속도를 낼 수 있다. 또한 연산이 끝난 부분의 타일은 메모리로부터 제거함으로써 사용량을 줄인다. 또한 타일들이 차지하는 메모리 사용량이 여유 메모리를 초과하지 않도록 조절하기 위해 메모리 budget 을 둔다. memory budget 이하의 크기로만 총 타일의 크기 합을 유지하도록 한다[4].

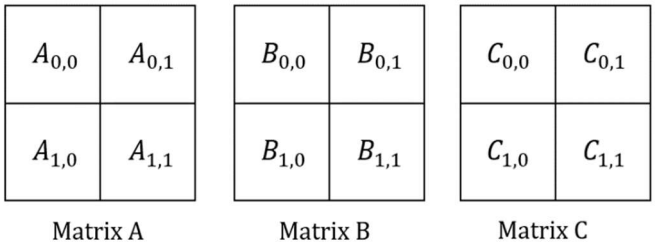

그림 1. 타일링된 GEMM 연산의 input matrix A, B 및 output matrix C

그림 1 은 GEMM 연산에 사용될 matrix 들이 타일링 된 모습이다. matrix A 와 B 를 곱하여 C 에 축적하는 GEMM 연산을 할 때, $C_{0,0} = (A_{0,0} * B_{0,0}) + (A_{0,1} * B_{1,0}) + C_{0,0}$ 와 같이 partial GEMM 을 축적해나가는 방식으로 진행된다. 이 연산을 시작하기 위해 $A_{0,0}$, $B_{0,0}$, $C_{0,0}$ 를 메모리로 올리면, $A_{0,0}$는 $C_{0,1}$를 연산할 때, $B_{0,0}$는 $C_{1,0}$ 를 연산할 때 다시 쓰일 수 있다.

본 연구는 이러한 GEMM 타일링 기법을 ArmCL 에 적용하여 ARM CPU 상에서 수행되는 GEMM 연산의 최대 메모리 사용량과 추론 시간의 변화를 관찰하였다.

4. 실험

GEMM 타일링의 성능을 관찰하기 위한 실험은 Odroid-XU4 를 사용하였다. ARM CPU 상에서 GEMM 연산을 실행하기 위한 프레임워크로는 ArmCL 을 사용하였다.

연산에 사용한 모든 수는 32bit 부동소수로 하였다. 두 개의 input matrix 과 하나의 output matrix 는 각 변에 4096 개의 숫자가 들어가는 정사각형 matrix 로 구성하였다. 타일 크기는 정사각형 Tile 의 한 변에 들어가는 숫자의 개수로, 타일 크기가 N 이면 원래의 matrix 를 N×N matrix 타일들로 나누게 된다.

4.1. 타일 크기 별 GEMM 연산의 최대 메모리 사용량 및 실행 시간 변화

메모리 budget 을 256MB 으로 고정하고, 타일 크기를 변화시키며 최대 메모리 사용량 및 실행 시간을 측정하여 결과를 [그림 2]에 나타냈다. 타일 크기가 4096 인 항목은 기존의 GEMM 연산과 동일하다.

전체적으로 타일 크기가 증가함에 따라 연산에 걸리는 시간은 감소하였고, 최대 메모리 사용량은 증가하였다. 연산에 사용된 matrix 전체의 크기의 합은 192MB 이나 타일 크기가 4096, 2048 인 경우에 400MB 에 근접하는 최대 메모리 사용량을 보이는 이유는 ArmCL 의 연산 도중 intermediate tensor 를 생성하는 것이 원인이다. 그 이하의 타일 크기를 가졌을 때에는 연산이 끝난 matrix 타일이 먼저 메모리에서 제거되고, 또한 생성되는 intermediate tensor 의 크기가 작으므로 최대 메모리 사용량이 급격히 줄어든다.

타일 크기가 2048 인 경우에 기존 GEMM 연산보다 실행 시간이 짧다. 이는 기존 GEMM 연산에서는 분리되어 있는 IO 시간과 연산 시간이 타일링을 하며 겹쳐지기 때문이다. 기존 GEMM 연산에서는 matrix 전체에 대한 읽기가 끝나고 나서 연산이 시작되고, 전체 연산이 끝난 후에 쓰기가 시작된다. 반면에 타일 크기를

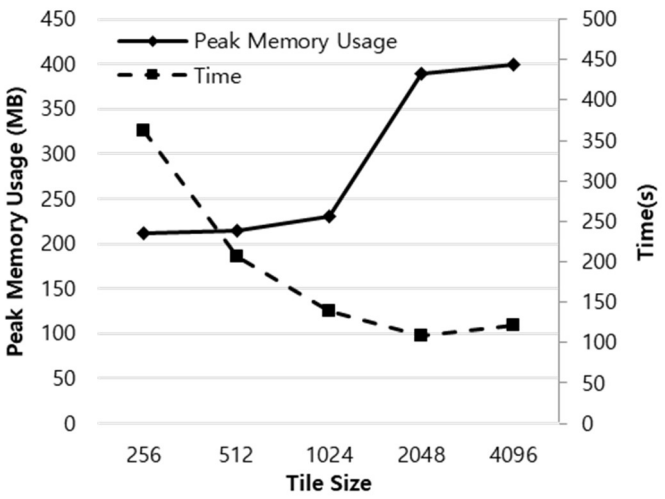

그림 2. 타일 크기 별 GEMM 연산의 최대 메모리 사용량 및 실행 시간 변화

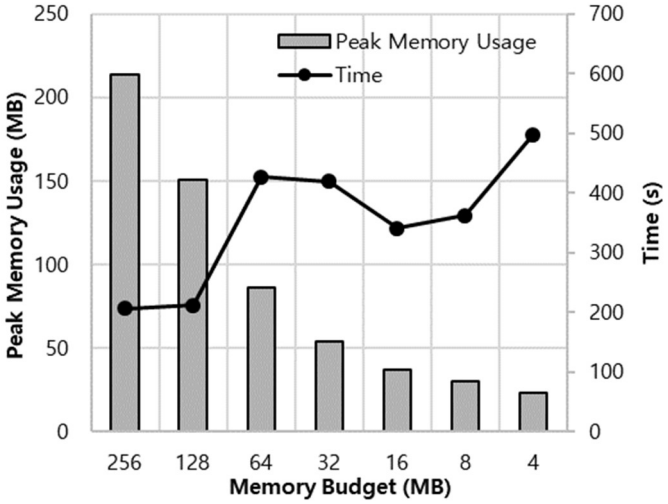

그림 3. 메모리 budget에 따른 최대 메모리 사용량 및 실행 시간. (타일 크기 = 512)

2048 으로 하면 일부분의 matrix 만을 읽어오고도 바로 연산을 시작할 수 있게 된다. 그러나 타일 크기가 작아질수록 한 번에 IO 를 요청하는 크기가 작아지고 횟수는 증가하므로 연산에 걸리는 시간이 길어진다.

4.2 메모리 budget 의 변화에 따른 GEMM 연산의 최대 메모리 사용량 및 Time 변화

메모리 budget 을 256MB 부터 4MB 까지 줄이면서 최대 메모리 사용량과 실행 시간을 측정하였다. 타일 크기는 512 로 고정하였다.

메모리 budget 의 감소에 따라 메모리 사용량은 일관적으로 감소하였고, 연산 시간은 전체적으로 증가하는 추세를 보였다.

메모리 budget 이 감소하면 메모리에 둘 수 있는 타일의 수가 줄어든다. 그러면 한 번 연산을 위해서 스토리지로부터 읽어온 타일이 메모리에서 제거되고 다른 연산을 위해 다시 읽어야 하는 경우가 생겨 IO 가 증가한다. 따라서 메모리 budget 이 줄어들면 연산 시간이 늘어나게 된다.

기존 GEMM 연산은 최대 메모리 사용량이 399MB, 실행 시간은 121 초이다. 실험 중 최대로 최대 메모리 사용량을 줄인 경우는 메모리 budget 을 4MB 으로 하고 타일 크기를 512 으로 사용하였을 때로 각각 23MB 와 498s 의 결과를 얻었다. 종합하여 볼 때, 최대 메모리 사용량을 17.3 배 줄였을 때 연산 시간은 4.1 배 증가하였으며 메모리 budget 과 타일 크기에 따라 최대 메모리 사용량 - 수행시간 간 tradeoff 를 보인다.

5. 결론 및 향후 연구

본 연구는 GEMM 연산에 이용되는 matrix 들을 타일 단위로 로드하고 연산하여 최대 메모리 사용량을 조절하는 방법을 IoT 장치의 CPU 에 적용하였다. 그 결과, 기존 GEMM 연산보다 최대 메모리 사용량을 약 17 배 작은 23MB 까지 줄일 수 있었으며, 실행 시간과 최대 메모리 사용량 간 다양한 tradeoff 를 획득하였다.

추후 이 방법을 IoT 환경의 GPU 에도 적용한다. 또한 CNN 네트워크를 구성하는 convolutional 레이어 및 fully connected 레이어로 확장하며, CNN 모델 전체의 메모리 사용량을 관리하는 시스템을 제안할 예정이다.

6. 참고문헌

[1] Han, Song, Mao, Huizi, and Dally,William J. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. arXiv preprint arXiv:1510.00149, 10, 2015.

[2] Gong, Yunchao, Liu, Liu, Yang, Ming, and Bourdev, Lubomir. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv: 1412.6115, 2014.

[3] WU, Chao, et al. Enabling Flexible Resource Allocation in Mobile Deep Learning Systems. IEEE Transactions on Parallel and Distributed Systems, 30.2: 346-360, 2019

[4] Subramanya, Suhas Jayaram, et al. BLAS-on-flash: an efficient alternative for large scale ML training and inference?. In: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, p. 469-483, 2019

[5] WANG, Linnan, et al. Superneurons: dynamic gpu memory management for training deep neural networks. In: ACM SIGPLAN Notices. ACM, p. 41-53. 2018

[6] Arm Compute Library - https://github.com/ARMsoftware/ComputeLibrary