Analysis on Accelerating Object Detection in Edge Device with Half-Precision Floating Point

유태관[0], 홍경환, 신동군
성균관대학교 전자전기컴퓨터공학과

Half-precision을 활용한 Edge Device에서의 Object Detection 가속 분석

Taekoan Yoo[0], Gyeonghwan Hong, Dongkun Shin
Department of Electronical and Computer Engineering, Sungkyunkwan University

요 약

딥러닝은 각종 IoT장치, 드론, 자율주행차 등에서 object detection의 정확도를 높이는 데 많이 사용되고 있다. 그러나 드론, IoT장치 같은 edge device는 전력소모가 적지만 프로세싱 성능이 낮기 때문에, deep learning 기반 object detection을 구동하려면 여러 가속 기법이 필요하다. Arm Compute Library와 같은 Edge Device용 딥 러닝 프레임워크에서는 SIMD(Single Instruction Multiple Data) 명령어나 GPU를 활용하여 딥 러닝 모델을 가속하는 기능을 제공한다. 기존 연구에서는 Classification 모델의 SIMD/GPU 가속에 대한 분석은 있었으나, Object Detection 모델에 대한 분석은 없었다. 본 논문에서는 경량 Object Detection 모델인 SqueezeDet을 Arm Compute Library를 기반으로 SIMD 또는 GPU로 가속하였을 때의 성능을 분석하였다. 특히, GPU로 가속 시에 inference time이 50% 감소하고, 이에 half precision(16bit)을 이용하면 12%만큼 추가로 감소해 real time에 한발 더 가깝게 만들 수 있음을 확인했다. 또한 이런 과정에서 GPU에서의 instruction의 변화를 분석해, 어떤 이유로 가속이 가능했는지 분석하였다.

1. 서 론

Deep learning을 활용한 object detection은 각종 영역에서 활용되고 있다. 드론, 자율주행 자동차 등에서 사용되는 기능인 장애물 감지, 차간 거리 유지 등은 deep learning을 활용하여 높은 정확도를 보여주고 있다. Deep learning 모델은 Nvidia Titan X 같은 고성능 GPU를 탑재한 클라우드에서는 이러한 높은 정확도를 보여줄 수 있다[1]. 그러나, 드론, 자율주행 자동차와 같은 edge device들은 응답 시간이 불안정한 무선 네트워크에 연결되기 때문에, 매번 서버에 연결되어 deep learning 모델의 결과를 전송 받을 수 없다. 이 때문에, 최근에는 edge device에서 연산을 수행하는 fog computing이 등장하였다[2].

그러나, edge device는 모바일 AP(Application Processor)를 탑재하기 때문에, 현재의 edge device에서 deep learning은 느리게 동작하여, real-time 성능인 60FPS(프레임 당 실행 시간 0.02초)[3]와는 거리가 매우 멀다. 예를 들어 Odroid-XU3(ARMv7)에서 최신 기법들이 포함된 딥러닝 object detection model들의 inference time을 같은 KITTI dataset에 대해 수행한 결과를 표1에 담았다. YOLOv2-tiny[4]는 4.01초, SSD[5]는 14.88초, SqueezeDet[6]은 1.658초의 프레임 당 실행 시간을 보여준다. 해당 시간은 SIMD를 활용했으며, 쓰레드를 최대한 활용한 결과이다.

Real time에 가까운 속도를 달성하기 위해 해당 model들에 가속 기법이 추가적으로 필요하므로, 본 논문은 이중에 가장 빠른 SqueezeDet을 선택해 ACL(Arm Compute Library)을 활용해 가속하고 real time에 한발 더 가까워질 수 있음을 제안한다. ACL은 ARM에서 제공하는 library로 GPU와 NEON을 활용해 딥러닝을 구현 하기 위한 API를 제공한다. 또한 half precision(16bit)을 이용한 수행을 쉽게 할 수 있도록 제공하므로, 이를 활용해 가속을 진행했다. 이에 앞서 half precision에 의한 mAP(mean Average Precision) 감소 여부를 확인하기 위해 실험한 결과, SqueezeDet의 기존 mAP는 78.8%인데 비해 16bit floating point로 바꾸어 확인해도 mAP가 유지되는 것을 확인했다.

mAP를 유지하며, ACL에 구현된 openCL을 이용한 GPU 가속기능과 half precision을 활용해 62%까지 가속이 가능한 것을 확인했다

Object Detection NN	Inference time
YOLOv2[4]	18.84s
YOLOv2-tiny[4]	4.01s
SSD[5]	14.88s
SqueezeDet [6]	1.658s
SqueezeDet+ [6]	4.67s`

표1. Object detection 기법 별 inference time

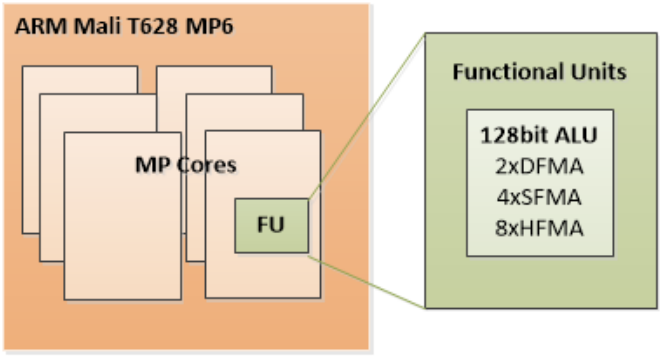

그림1. Mali T628의 Functional Units

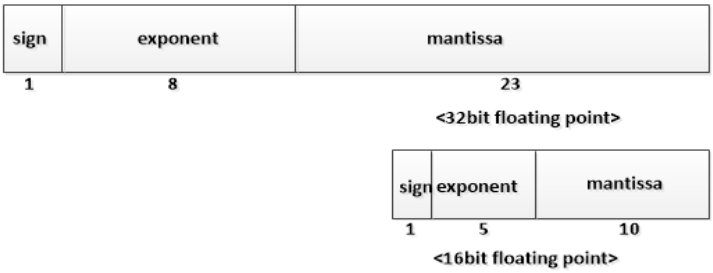

그림2. 32bit floating point vs 16bit floating point

2. 배 경

2.1. Half-precision floating point

CPU나 GPU의 SIMD(Single Instruction Multiple Data) 구조에서는 full precision floating point 대신 half precision floating point를 사용함으로써 2배까지 데이터 처리량을 늘린다. 예를 들어, 그림1과 같이 Mali-T628 GPU의 functional units은 2개의 64bit FMA(Fused Multiply ADD), 4개의 32bit FMA를 작동할 수 있다. 16bit FMA instruction의 경우에는 8개를 동시에 작동할 수 있다.

ARM CPU에서는 SIMD 명령어인 NEON을 제공한다. 그러나 NEON은 ARMv7에서 full precision floating point만을 지원하며, half precision floating point는 ARMv8.2부터 지원한다. Half precision구조는 그림2와 같으며 해당하는 data의 exponent와 mantissa를 잘라내 사용하는 방법을 선택한다. 반면 Mali-T628 같은 모바일 GPU는 half precision floating point를 지원한다. 따라서 본 논문에서는 full precision floating point를 활용하는 NEON 가속과 full precision 및 half precision floating point를 활용하는 GPU가속을 비교하였다.

2.2. ACL(Arm Compute Library)

ACL은 ARM에서 Computer Vision과 Machine Learning을 ARM에서 최적화 하기 위해 만든 library이다. NEON구조를 이용한 NE버전과 OpenCL을 이용해 GPU 가속을 하는 CL버전으로 나뉘어 있으며, 이미 구현된 Layer들을 이용해 Neural Network를 만들고 실행할 수 있다.

본 논문은 ACL을 활용해 SqueezeDet을 구현했으며 ACL에서 지원하는 여러가지 bit 모드를 활용해 실험을 진행했다.

3. 실 험

3.1. 실험환경

실험은 Odroid-XU3에서 진행했으며 ARMv7 기반의 Cortex-A7 4개와 Cortex-A15 4개의 core로 이루어 져 있으며 Mali-T628 MP6 GPU를 탑재하고 있다.

SqueezeDet을 실행하기 위한 weight는 기존 32bit floating point를 활용했다. Inference를 하기 위해서는 Odroid에서 ACL를 활용해 구현했으며, 6번의 inference를 통해 평균 시간을 찾고, 전체적으로 GPU 사용량을 측정했다.

3.2. OpenCL 가속

3.2.1. 수행시간 및 메모리

SqueezeDet을 ACL의 openCL을 활용해 가속한 결과는 그림3과 같다. ARM에서 각각 32bit와 16bit floating point, 8bit와 16bit fixed point를 활용해 inference time을 측정한 결과이다.

32bit floating point를 openCL을 활용하면 기존의 1.658초보다 약 50% 빠른 0.844초의 inference time을 가지며, 이에 half precision을 12% 더 빠른 0.64초의 inference time을 가지는 것을 볼 수 있다. 또한 fixed point로 openCL을 활용한 경우 기존의 시간보다 더 오래 걸리는 것을 알 수 있는데, 이는 GPU의 경우 오히려 fixed point가 구현되어 있지 않은 경우, FMA를 효율적으로 활용하기 위해 floating point로 바꾸는 추가 비용이 생기기 때문이라고 생각할 수 있다.

메모리의 경우, openCL을 활용해 32bit floating point로 가속을 하면 833MB GPU memory를 사용하는 반면 16bit floating point로 가속 시 423MB의 GPU memory만을 사용하는 것을 확인 할 수 있었다.

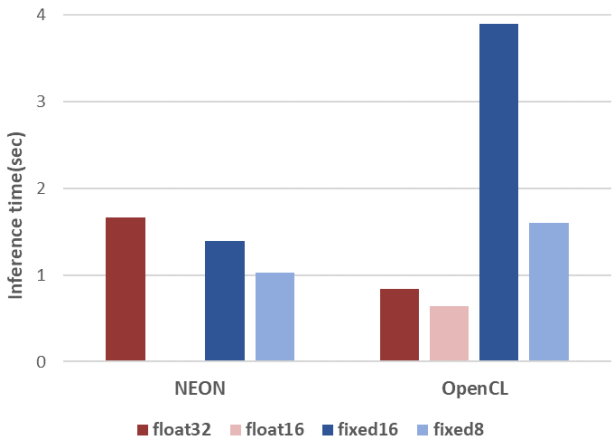

그림3. NEON과 OpenCL을 활용한 inference time(초)

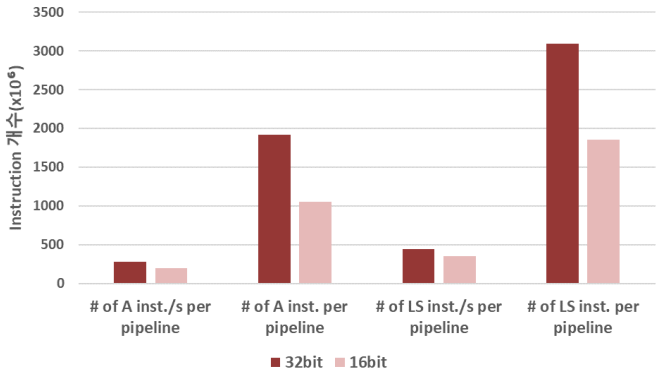

그림4. OpenCL 가속 시 32bit와 16bit 각각의 instruction 개수 비교

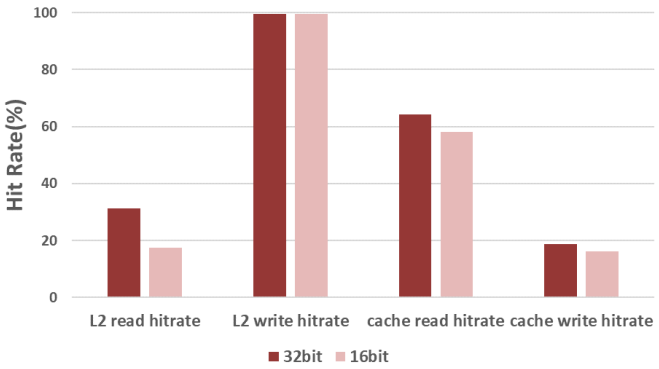

그림5. OpenCL 가속 시 32bit와 16bit 각각의 L1, L2 cache hit rate(%) 비교

3.2.2. GPU 사용량 변화

그림4는 속도에 따른 GPU 사용량의 변화를 분석한 것이다. 이는 ARM에서 제공하는 streamline profiling tool[7]을 이용해 분석한 것이다. 이 때 A instruction은 Arithmetic 연산에 대한 명령어를 말하며, LS instruction은 메모리를 Load/Store하기 위한 명령어를 말한다.

위 그림은 pipeline 하나 당 GPU 사용 data를 설명한 것인데, 시간당 명령어의 개수와 총 수행 시간 동안의 명령어 둘 다 감소 한 것을 볼 수 있다. 그러나 그림5를 통해 hit rate는 오히려 L2 캐시와 L1 캐시 모두 감소한 것을 볼 수 있으며, 특히 read 명령의 hit rate가 감소한 것을 볼 수 있다. 그럼에도 불구 하고 수행 시간이 감소할 수 있었던 것은, 명령어의 개수가 압도적으로 줄었기 때문이라고 볼 수 있으며, hit rate를 최적화 한다면 수행시간을 추가적으로 줄일 수 있음을 볼 수 있다.

3.3. NEON 가속

SqueezeDet을 ACL의 NEON을 활용해 가속한 결과 또한 그림3과 같다. OpenCL 가속과 비교하기 위해서 실험을 진행했는데, 위에서 보이는 바와 같이 16bit floating point에 대한 연산을 위해서는 FP16을 지원하는 NEON architecture를 사용해야 하는데 이는 ARMv8.2(Cortex-A55/A75) 이후에서만 지원이 되므로 해당 결과를 알 수 없었다.

NEON을 활용해 가속한 경우는 오히려 fixed point로 바꾸어 실행한 경우 오히려 수행시간이 감소했는데, 이는 GPU와 다르게 fixed point 연산을 할 수 있는 환경이 갖추어 졌기 때문이다.

4. 결 론

본 논문에서는 다양한 분야에서 활용되고 있는 object detection을 real time으로 활용하기 위해 ACL의 half-precision floating point를 활용해 가속하는 것을 제안한다.

논문에서는 제시하지 못한 NEON의 half-precision 연산이 ACL에 구현이 되어있지 않아 이를 OpenCL 가속과 비교해 어떤 지를 확인하는 것이 필요할 것으로 보인다. 또한 이번엔 Inference time에 32bit floating point로 미리 학습된 SqueezeDet의 data를 활용했지만 half-precision으로 직접 학습시키고 inference를 했을 시에 mAP가 어떻게 변할지를 다시 한번 확인할 필요가 있을 것으로 보인다.

마지막으로, 캐시의 hit rate가 낮음에도 불구하고 instruction 개수가 확연히 줄어 수행시간이 줄었으므로, 이후에는 hit rate를 개선하는 방법을 통해 수행시간을 줄여 더욱 real time(0.02초)에 가깝게 구현할 수 있음을 보았다.

참고문헌

[1] Lin, Shih-Chieh, et al. "The Architectural Implications of Autonomous Driving: Constraints and Acceleration." Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2018.

[2] Bonomi, Flavio, et al. "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, (2012).

[3] Zhu, Yuhao, Matthew Mattina, and Paul Whatmough. "Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective." arXiv preprint arXiv:1801.06274 (2018).

[4] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint (2017).

[5] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.

[6] Wu, Bichen, et al. "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving." arXiv preprint arXiv:1612.01051 (2016).

[7] ARM https://developer.arm.com/products/software-development-tools/ds-5-development-studio/streamline(2010)