ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 21-04-11

j.inst.Korean.electr.electron.eng.Vol.25,No.4,664~671,December 2021
https://doi.org/10.7471/ikeee.2021.25.4.664
81

딥러닝 합성곱에서 데이터 재사용에 최적화된 GPGPU 설계

Design of an Optimized GPGPU for Data Reuse in DeepLearning Convolution

남 기 훈*, 이 광 엽*, 정 준 모**★

Ki-Hun Nam*, Kwang-Yeob Lee*, Jun-Mo Jung**★

Abstract

This paper proposes a GPGPU structure that can reduce the number of operations and memory access by effectively applying a data reuse method to a convolutional neural network(CNN). Convolution is a two-dimensional operation using kernel and input data, and the operation is performed by sliding the kernel. In this case, a reuse method using an internal register is proposed instead of loading kernel from a cache memory until the convolution operation is completed. The serial operation method was applied to the convolution to increase the effect of data reuse by using the principle of GPGPU in which instructions are executed by the SIMT method. In this paper, for register-based data reuse, the kernel was fixed at 4x4 and GPGPU was designed considering the warp size and register bank to effectively support it. To verify the performance of the designed GPGPU on the CNN, we implemented it as an FPGA and then ran LeNet and measured the performance on AlexNet by comparison using TensorFlow. As a result of the measurement, 1-iteration learning speed based on AlexNet is 0.468sec and the inference speed is 0.135sec.

요 약

본 논문은 합성곱 신경망에 데이터 재사용 방법을 효과적으로 적용하여 연산 횟수와 메모리 접근 횟수를 줄일 수 있는 GPGPU구조를 제안한다. 합성곱은 kernel과 입력 데이터를 이용한 2차원 연산으로 kernel이 slide하는 방법으로 연산이 이루어 진다. 이때, 합성곱 연산이 완료될 때 까지 kernel을 캐시메모리로 부터 전달 받는 것이 아니고 내부 레지스터를 이용하는 재사용 방법을 제안한다. SIMT방법으로 명령어가 실행되는 GPGPU의 원리 이용하여 데이터 재사용의 효과를 높이기 위해 합성곱에 직렬 연산 방식을 적용하였다. 본 논문에서는 레지스터기반 데이터 재사용을 위하여 kernel을 4x4로 고정하고 이를 효과적으로 지원하기 위한 warp 크기와 레지스터 뱅크를 갖는 GPGPU를 설계하였다. 설계된 GPGPU의 합성곱 신경망에 대한 성능을 검증하기 위해 FPGA로 구현한 뒤 LeNet을 실행시키고 TensorFlow를 이용한 비교 방법으로 AlexNet에 대한 성능을 측정하였다. 측정결과 AlexNet기준 1회 학습 속도는 0.468초이며 추론 속도는 0.135초이다.

Key words : Data Reuse, CNN, GPGPU, Row stationary, SIMT, Warp, Register bank

---

* Dept. of Computer Eng.
** Dept. of Electronics Eng., Seokyeong University
★Corresponding author
E-mail : jjmo@skuniv.ac.kr, Tel : +82-2-940-7732
Manuscript received Dec. 13, 2021; revised Dec. 16, 2021; accepted Dec. 16, 2021.
※Acknowledgment: This work was supported by Seokyeong University in 2021 and by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP). (No. 2016-0-00204, Development of mobile GPU hardware for photo-realistic realtime virtual reality)

Ⅰ. 서론

딥러닝 신경망을 이용하여 입력 이미지(image)를 분류(classification)하는 것은 기존의 영상 알고리즘 수준으로 할 수 있으며 사람의 판단을 앞서는 성능을 보여주고 있다. 이미지 분류는 영상 인식을 활용하는 분야에 널리 사용되고 이 분야에서 딥러닝 신경망이 좋은 결과를 나타냄으로써 딥러닝 신경망은 인공지능이 산업에 뿌리를 내릴 수 있도록 기여를 하였다. 특히, ILSVRC[1]를 통하여 발표된 AlexNet[2], VGGNet, GoogleNet, ResNet 등 합성곱 신경망(CNN : convolutional neural network)은 딥러닝을 이용한 영상인식 기술의 획기적인 발전에 기여 하였다. 합성곱이라고 하는 convolution 연산은 이미지에서 특징을 추출하는데 널리 사용되는 kernel 연산 방법이다. 입력 이미지에 kernel를 적용하여 픽셀별로 곱을 한 후 모두 합을 하면 한 개의 출력을 만드는 elementwise-sum 연산이 반복된다. 한 번의 convolution 연산이 완료되면 kernel를 stride수 만큼 우측으로 이동하여 다음 convolution 연산을 수행하고 또 하나의 출력값을 만들게 된다.

내부 메모리나 레지스터 파일을 이용한 딥러닝 데이터 재사용 연구는 대부분 2차원 어레이 구조의 가속기를 대상으로 하기 때문에 성능이 어레이 크기나 2차원 파이프라인에 제한을 받기 때문에 병렬처리 능력에서 한계가 있다.

본 논문에서는 core의 개수를 늘려 병렬처리할 수 있는 방법으로 SIMT(Single Instruction Multiple Thread)기반의 GPU[3]를 대상으로 하며 thread수 만큼 병렬실행 능력을 늘릴 수 있다. 특히, 모바일 응용 시스템을 목적으로 적은 크기의 GPU를 사용하기 위해 warp의 크기를 적게 하고 딥러닝 kernel의 특징에 맞도록 하였다[4]. 작은 크기의 kernel이 딥러닝 convolution에서 대부분을 차지 하고 있기 때문에 이러한 특징을 최대한 반영하여 병렬 thread 개수와 레지스터 뱅크를 결정하고 kernel과 입력 데이터를 thread 단위로 병렬화 및 데이터 재사용을 적용하였다. 연구 결과는 AlexNet과 비교 평가 하였다.

Ⅱ. 내부 메모리 기반 합성곱 데이터 재사용 구조

일반적인 CNN(Convolutional Neural Network) 구조에서 convolution연산[5]의 병렬화는 다음 네 가지 측면에서 가능하다. 병렬화는 연산 속도의 개선 뿐만 아니라 convolution에 사용되는 입력 채널 데이터, kernel 데이터, 출력 채널 데이터를 DPU 내부 메모리에 상주 시켜 최종 결과가 이루어질 때까지 외부 DDR 메인메모리에 Load/Store하는 횟수가 줄어들어 처리시간 단축과 전력소모를 최소화 할 수 있다.

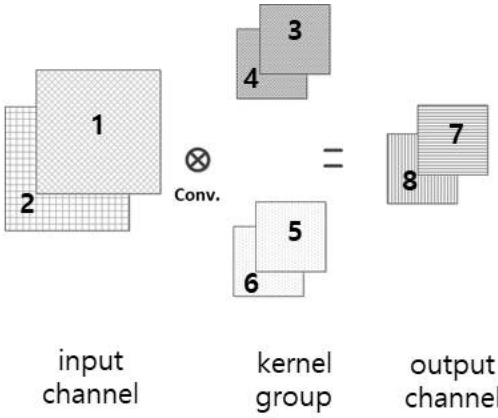

Fig. 1. Convolution and Data Reuse.
그림 1. 합성곱 연산과 데이터 재사용

ⓐ입력채널과 kernel 병렬연산 : 입력 데이터는 입력 채널수 만큼 동시에 병렬로 입력되며 각 채널별로 kernel과 convolution 연산을 실행한다. 입력 채널 마다 각각 다른 kernel을 사용하거나 한 개의 kernel을 공유할 수 있다. 그림 1에서 입력 채널 ①과 kernel ③, 입력 채널 ②와 kernel ④가 동시에 convolution 연산을 병렬 실행하며 각각 연산에서 만들어진 결과는 합(sum)을 하여 출력 채널 ⑦을 만들게 된다.

이러한 병렬연산 과정에서 kernel은 convolution 연산이 끝날 때 까지 내부 메모리에 상주하며 재사용(reuse)되는 장점이 있다.

ⓑ입력과 kernel group 병렬연산 : 입력 데이터 한 개의 채널에 여러개의 kernel을 적용할 수 있는데 이와같이 적용되는 kernel set을 kernel group이라고 한다. 그림에서 입력 채널 ①은 kernel ③, ⑤와 각각 convolution연산을 하게 되며 kernel ③, ⑤는 kernel group이 된다. 이때, kernel ③의 연산 결과는 임시 출력 채널 ⑦을 만들고 kernel ⑤의 연산 결과는 임시 출력 채널 ⑧을 만들게 되며 이 두개의 연산 과정이 병렬로 처리가 가능하다. 같은

방법으로 입력 채널 ②는 kernel ④, ⑥과 각각 convolution연산을 하게 되며 kernel ④, ⑥은 kernel group이 된다. 이때, kernel ④의 연산 결과는 앞서 임시 출력 채널 ⑦과 합(sum)이 되어 최종 출력 채널 ⑦을 만들고 kernel⑥의 연산 결과는 임시 출력 채널⑧과 합(sum)이 되어 최종 출력 채널 ⑧을 만들게 된다.

입력 데이터는 해당 kernel group의 kernel들과 convolution 연산과정에서 내부 메모리에 상주하며 kernel의 수 만큼 재사용이 된다.

ⓒ출력채널 병렬연산 : 출력채널의 수는 kernel group의 수와 같다. 그림에서 kernel ③, ④가 하나의 group이 되어 출력채널 ⑦을 만들고 kernel ⑤, ⑥이 또 다른 group이 되어 출력채널 ⑧을 만들게 된다. 그림과 같이 kernel group이 두 개인 경우 출력채널도 두 개가 된다. 이 과정을 병렬연산 하는 것이 출력채널 병렬화이다.

출력채널을 만들때 같은 group에 있는 kernel과 입력 채널과의 convolution연산 결과가 누적합(accumulation-sum)이 되는데 이 과정에서 출력채널이 내부 메모리에 상주하기 때문에 출력채널 데이터 재사용이 이루어 진다.

III. 직렬연산 기반 데이터 재사용 구조

1. Weight Stationary(WS) 구조

Local memory에 가중치값을 저장해 놓고 global buffer로부터 입력 데이터와 convolution의 부분합을 읽어서 가중치와 계산된 부분합을 global buffer에 저장하는 구조이다. 레지스터 파일로 부터 가중치 읽는 횟수를 최소화 하여 전력소모를 줄이면서 kernel reuse를 최대화 한다. 구글의 DNN 가속기인 TPU[]에 적용하고 있다.

2. Output Stationary(OS) 구조

입력이 여러 채널로 구성될 때 kernel은 입력의 각 채널과 convolution을 실행하면서 부분 합을 출력한다. 각 채널의 부분 합은 최종 합을 통하여 convolution이 완성된다. 이때, 채널 별 부분 합을 레지스터 파일에 저장하기 때문에 최종 합을 할 때 레지스터에서 바로 읽어 오기 때문에 global buffer나 local memory 접근이 없어 소모 전력을 줄일 수 있다.

3. Row Stationary(RS) 구조

RS에서 입력 데이터(input activation)은 PE(Processor Element)의 row단위로 재사용이 되고 weight는 대각선 단위로 재사용이 된다. 그림 2에서 점선 화살표는 가중치 재사용을 의미하며 실선 화살표는 부분합의 전달을 의미한다.

PE3, PE6의 연산이 종료된 후 Row3와 Row4는 데이터 재사용을 위해 각각 PE5, PE8로 weight를 전달한다. Eyeriss[6]는 그림 2와 같이 사용한 데이터를 다음 PE로 전달하면서 데이터를 재사용한다.

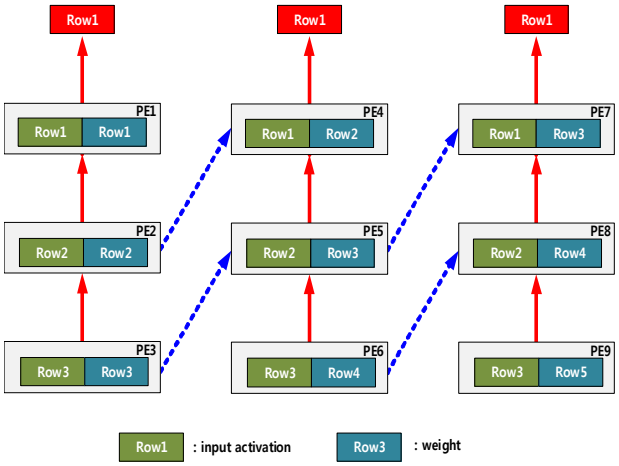

Fig. 2. RS convolution method in Eyeriss.
그림 2. Eyeriss에서 RS 합성곱 방법

RS 방법으로 데이터를 재사용하면 내부 레지스터 파일에 데이터를 저장한 후 여러 번 사용이 가능하여 local memory나 global buffer에 접근하는 횟수를 줄일 수 있다.

데이터 재사용 방법은 앞에서 제시한 세가지 방법에 따라 가중치(WS), 부분 합(OS), 입력 데이터(pixels)를 각각 재사용하거나 RS와 같이 동시에 재사용함으로써 그림 3과 같이 에너지에 대한 효율을 비교하면 RS가 1.4~2.2배 가량 앞서 있다.

위와같이 WS, OS 재사용 방법에 비하여 RS가 높은 에너지 효율을 갖지만 세가지 방법 모두 PE로 구성된 2차원의 어레이 기반 가속기이며 내부에 global buffer가 지원되어야 하며 2차원의 어레이 크기 제한에 따라 데이터를 분할해야 하는 복잡도가 높은 단점이 있다.

본 논문에서는 2차원 어레이 크기에 제한되는 특정 가속기가 아닌 SIMT기반의 GPU 병렬처리 능력을 활용한 병렬 쓰레드 데이터 재사용을 제안한다.

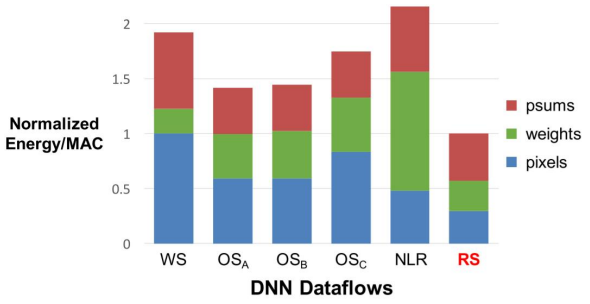

Fig. 3. Comparison of power consumption efficiency by data reuse structure.
그림 3. 데이터 재사용 구조별 소모전력 효율비교[6]

Ⅳ. SIMT 기반 병렬처리 데이터 재사용 구조

1. 제안하는 데이터 재사용 방법

딥러닝의 convolution 연산을 구성하는 kernel 값과 입력 데이터 값, 그리고 연산의 결과 값을 GPU 내부 레지스터 저장함으로써 convolution 연산이 종료될 때 까지 GPU 외부 메모리 접근을 하지 않도록 warp 크기, 레지스터 구조, 직렬 convolution 연산 방법을 개선한다.

첫 번째는 warp크기를 kernel크기에 맞추어 설계하였다. GPU는 SIMT구조로 되어 있어 thread단위의 병렬처리를 하게 되고 thread를 관리하기 위한 묶음 단위를 warp으로 하며 warp당 관리하고 동시에 처리할 수 있는 tread의 갯수를 warp 크기로한다. 이때, warp크기를 결정할 때 딥러닝 convolution을 실행할 때 데이터 재사용 효율을 최대화 할 수 있도록 kernel의 크기를 최대 4x4(4행, 4열)로 정한다. 4x4로 정하여도 1x1, 2x2, 3x3, 4x4의 크기를 지원하여 유연성을 높인다.

둘째는 warp크기에 맞춘 레지스터 뱅크 설계이다. 딥러닝 convolution은 GPU 명령어로 실행되며 명령어들은 thread 단위로 병렬 실행되는데 명령어는 데이터를 캐시메모리로 부터 레지스터로 옮겨와서 사용한다. 캐시메모리로 부터 옮겨진 데이터를 레지스터에서 convolution연산이 종료될 때까지 재사용한다. 이때, 병렬 실행하는 thread의 한 묶음인 4이기 때문에 레지스터 구조도 이와 동일하게 4개의 뱅크로 구성하면 4x4 kernel의 데이터 재사용을 효과적으로 지원한다.

세 번째는 메모리 접근 구조이다. 딥러닝에서는 많은 양의 kernel과 연속되는 입력 데이터를 사용하기 때문에 kernel과 입력 데이터는 자주 교체가 된다. 따라서, 레지스터내의 데이터 재사용에도 한계가 있기 때문에 kernel과 입력 데이터 교체시 메모리 접근이 불가피하다. 딥러닝의 convolution 연산을 고성능으로 유지하기 위해서는 레지스터 기반의 데이터 재사용이외에 메모리 접근 지연시간을 최소화 하는 메모리 latency hiding기술이 적용되어야 한다. 메모리 접근시 지연시간이 발생하는 thread는 대기 시키고 지연 발생이 없는 thread로 순서를 대체하는 방법이 필요하다.

2. 제안하는 SIMT기반 직렬 데이터 재사용

본 연구에서는 데이터 재사용에 적합한 SIMT구조 GPU[3]를 설계하였으며 warp 크기와 레지스터 뱅크 크기에 맞추어 다음과 같이 convolution 데이터 재사용을 제안한다.

딥러닝 convolution 연산을 GPU 내부 레지스터를 이용하여 최대한 데이터 재사용을 하기 위해 직

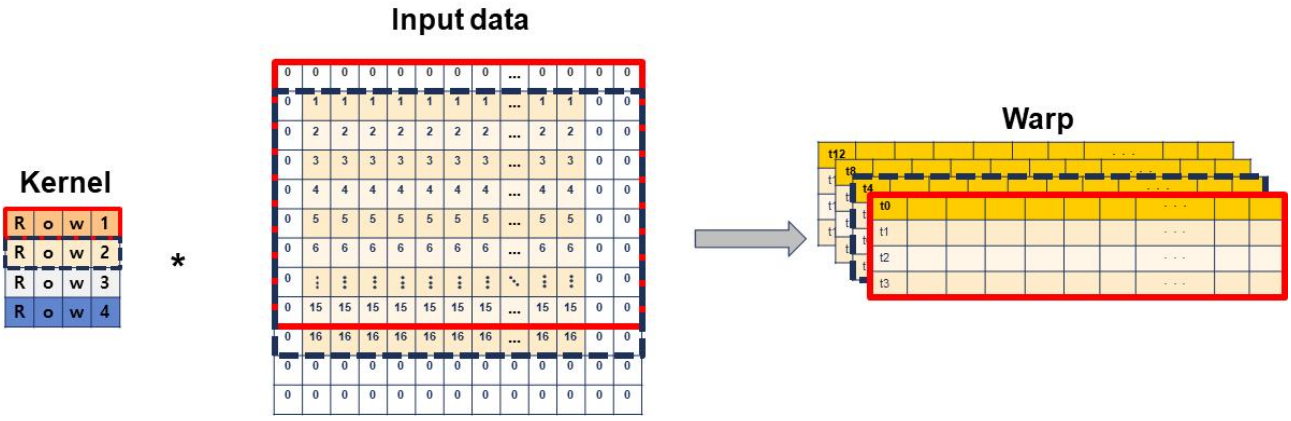

Fig. 4. Serial calculation method of convolution using register file.
그림 4. 레지스터 파일을 이용한 직렬 합성곱 계산 방법

렬연산(serial operation)을 적용한다. 직렬연산은 그림 4와 같이 입력 데이터와 kernel을 한 행(row)씩 convolution 연산을 수행하며 한 행의 연산은 한 warp의 한줄의 thread들에 할당한다. Kernel의 굵은 실선으로 표시된 row는 입력 데이터의 굵은 실선 부분과 한 행씩 연산을 하게되며 이 부분은 굵은 실선으로 표시된 warp에 할당된다. Kernel의 점선 row는 입력 데이터의 점선 부분과 한 행씩 연산을 하며 점선 warp에 할당한다. 이 방법으로 4줄의 kernel을 차례대로 warp에 할당함으로써 warp 마다 갖고 있는 레지스터를 이용하여 행단위로 convolution을 하게 되면 kernel 데이터와 입력 데이터를 메모리 접근 없이 레지스터 내부에서 재사용이 된다.

위와 같은 kernel과 입력 데이터의 행단위 직렬 convolution 연산을 위해서는 kernel의 크기를 4x4 (row 4개, column 4개) 크기로 고정한다. 딥러닝 convolution에서 kernel 크기는 다양할 수 있지만 주로 3x3과 4x4를 사용하기 때문에 4x4로 고정해도 3x3과 4x4에 모두 위와 같은 방법을 적용할 수 있어 대부분의 딥러닝 convolution 연산의 데이터 재사용 방법에 적용이 가능하다.

Ⅴ. SIMT기반 데이터 재사용의 성능측정

1. Convolution 연산 능력 측정

딥러닝에 직렬구조의 convolution 방법을 적용하였을 때 기존의 직렬방식 convolution연산에 필요한 연산 횟수와 본 발명의 GPU구조에서 convolution연산에 필요한 연산 횟수를 비교하였다. 기존 방식에서는 그림 5와 같이 kernel의 행(row)으로 입력 데이터에 convolution을 진행한 후 kernel의 열(column)으로 중간 출력 특징맵(output feature map)에 convolution 연산을 하는 방법으로 28x28 크기의 입력 데이터에 4x4 크기 kernel을 적용하였을 때 100,352번 연산횟수가 소요된다. 본 발명의 방식에서는 그림 6과 같이 4x4 kernel과 입력 데이터를 한 개의 warp에서 병렬 convolution 연산을 수행한 후 한 장의 중간 출력 특징맵에 1x1 convolution을 수행하여 기존방식과 같은 출력을 나타내는데 25,088번 연산횟수를 사용하여 기존 방식에 비하여 약 4배의 성능 개선을 보였다.

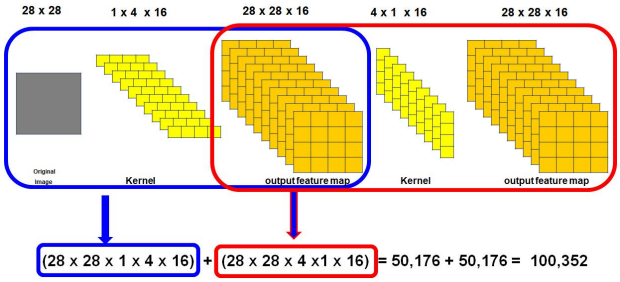

Fig. 5. Number of MNIST data set execution cycles in a typical serial convolution structure.

그림 5. 일반적인 직렬 convolution구조에서 MNIST 데이터 세트 실행 사이클 수

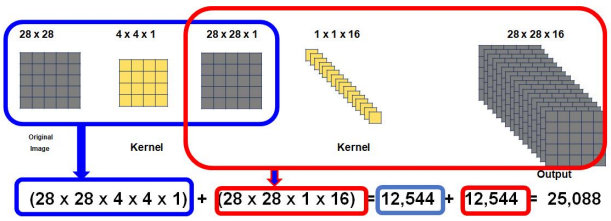

Fig. 6. Number of MNIST data set execution cycles in the proposed GPU serial convolution structure.

그림 6. 제안하는 GPU의 직렬 convolution구조에서 MNIST 데이터 세트 실행 사이클

2. AlexNet 신경망 추론 능력 측정

본 논문의 GPU는 RTL설계와 FPGA 구현으로 1개 core에서 실제 측정된 MNIST 데이터 세트의 딥러닝 처리 능력을 기준으로 CNN의 기준이 되는 신경망 중 하나인 AlexNet에 대한 비교 측정을 다음과 같이 진행하였다.

설계된 GPU core 32개를 사용하여 Alexnet 연산을 기준으로 한다. 딥러닝 추론 능력을 입증하기 위해 MNIST를 그림 7같은 규격의 인공신경망을 본 과제의 GPGPU core 1개에 포팅하고 FPGA상에서 실행을 시켰다. Alexnet은 대규모 인공신경망으로 본 논문의 GPGPU core가 포팅된 FGPA 메모리 용량을 초과하기 때문에 MNIST 데이터세트를 사용하는 인공신경망(ConvA로 명칭)에서 측정된 결과를 기준으로 Alexnet과 비교 하였다.

표 1은 TensorFlow 시뮬레이션상에서 MNIST 신경망과 AlexNet 처리시간 비교 결과로 인공신경망 처리량은 1초에 신경망(Net)에서 테스트(1 iteration) 처리량을 의미한다. 즉, AlexNet은 MNIST신경망에 비하여 341.46배의 처리 시간을 요구한다.

GPGPU core 1개를 포팅한 FPGA상에서 실행시

켜 측정된 MNIST Net 실행시간과 앞에서 계산된 Alexnet의 처리량의 상대 비율을 적용 할 때 본 GPGPU core 1개에서 Alexnet의 실행을 12.711ms x 341.46 = 4,340.29ms로 추정할 수 있다.

Input image	(1@28 x 28)
Conv0	(8@24 x 24) (k : 5 x 5) (ReLU) (VALID)
Pooling	(8@12 x 12) (max)
Conv1	(16@8 x 8) (k : 5 x 5) (ReLU) (VALID)
Pooling	(16@4 x 4) (max)
FC0	(128)(ReLU)
FC1	(64)(ReLU)
FC2	(10)

Fig. 7. Neural network structure implemented by the proposed GPU.
그림 7. 제안하는 GPU에서 구현한 신경망 구조

Table 1. ConvA and AlexNet's one-iteration execution time in the train and inferece.
표 1. ConvA와 AlexNet의 1 iteration 학습 및 추론 실행시간

	Training(1-iteration)	Testing(1-iteration)
ConvA(MNIST)	0.0486msec	0.0116msec
AlexNet(Image Net)	62.43msec	3.961msec

Table 2. Comparison of simulation execution time between ConvA and AlexNet.
표 2. ConvA와 AlexNet의 시뮬레이션 실행시간 비교

	Simulation Time (Number of cycles)	ratio
ConvA(MNIST)	12.711msec (1,271,262 cycles)	1
AlexNet(Image Net)	4,340.29msec	341.46

MNIST 인공신경망과 Alexnet에 이미지 50,000장으로 학습하였을 때 학습 1 iteration 시간과 이미지 1장으로 테스트하였을 때 실행시간(Nvidia 환경에서 실행) 표 2와 같다. MNIST신경망은 100MHz FPGA에서 12.711ms 실행시간(1,271,262 클럭사이클 소요) FPGA 100MHz에서 검증된 GPGPU core를 32개 사용하게 되면 Alexnet 연산기준 인공신경망 처리 성능은 다음과 같다.

$$\frac{4{,}340.29ms}{32} = 135.63ms = 7.37/s$$

즉, Alexnet 인공신경망을 1초당 7.37회 처리

3. AlexNet 신경망 학습 능력 측정

설계된 GPU의 신경망 학습 능력도 추론 측정과 같은 방법으로 AlexNet과 비교 하였다. 32개 core를 사용하여 학습 1 iteration 실행시간을 Alexnet 연산을 기준으로 측정하였다. 설계된 GPGPU core 1개에 MNIST 인공신경망을 포팅하고 이미지 1장으로 테스트를 하였을 때 실행시간은 12.711ms 이다. Alexnet의 학습연산 기준으로 비교 하면 다음과 같다.

표 1에서 MNIST 테스트와 학습(1 iteration)과의 비율은 4.189배 이다. 즉, MNIST 인공신경망에 이미지 1장으로 테스트를 하였을 때 실행시간은 12.711ms 이다. 따라서, 학습(1 iteration)의 실행시간은 12.711ms x 4.189= 53.246 ms로 환산된다. 표 1에서 AlexNet의 학습(1 iteration)시간은 MNIST학습 시간에 비하여 1,284.5배 이기 때문에 AlexNet 연산 기준 학습의 한번 Iteration 시간은 68,394.4ms이다.

설계된 FPGA 100MHz GPGPU core를 32개 사용하게 되면 AlexNet 연산기준 인공신경망 학습처리 성능은 다음과 같습니다.

$$\frac{68{,}394.4ms}{32} = 2{,}137.3ms = 0.468/s$$

즉, 1초당 0.468 Iteration을 수행

4. 타 연구와 신경망 추론 능력 비교

AlexNet 신경망은 여러 연구[7][8] 결과를 통하여 비교 대상이 되는데 이 가운데 CPU, GPU, FPGA에서 구현하고 이를 비교한 선행 논문[7] 결과와 설계된 GPU의 AlexNet 신경망 추론 능력을 비교하였으며 그 결과는 다음 표 4와 같다. 선행 논문에서 GPU의 실행 시간은 0.358초 이며 본 논문에서 설계한 GPU의 실행 시간은 0.135초 이다.

Table 3. AlexNet inference time comparison.
표 3. AlexNet 추론 시간 비교

Reference	Target Device	Time
[7]	CPU i5-6400	0.539sec
	GPU GTX960	0.358sec
	FPGA	20.67sec
Proposed	GPU	0.135sec

Ⅴ. Conclusion

딥러닝에 가장 널리 사용되는 합성곱 신경망은 여러 종류의 신경망으로 구성 되지만 convolution 계층이 가장 많은 연산 시간 및 메모리 사용량이 가장 크기 때문에 전체 성능의 대부분을 차지 한다. 또한, 모바일기기에서 딥러닝을 활용하기 위해서는 소모 전력대비 성능을 높이기 위한 구조가 필수적이다. 본 논문에서는 소모전력을 줄이기 위해서 core수가 줄어든 대신 성능을 높이기 위해 thread를 관리하기 위한 warp scheduler의 size를 4로 하여 일반적인 GPGPU의 32 보다 크게 줄였다. 메모리 접근시간을 최소화 하기 위해 내부 레지스터를 이용한 데이터 재사용 방법을 제안하는데 특히 kernel의 크기를 4x4 이하로 제한하고 레지스터 뱅크를 4개로 하여 데이터 재사용의 효율을 최대화 하였다. 본 논문에서 제안하는 데이터 재사용과 메모리 접근 지연시간을 줄이는 SIMT GPU는 convolution 연산에 효율적인 구조로 설계되어 있어 일반적인 GPU보다 AlexNet 신경망 시뮬레이션 비교 기준으로 추론 성능에서 2.5배 개선 하였다. 향후, 내부 캐시 메모리의 prefetch와 warp scheduling 개선으로 딥러닝 데이터 재사용 효율을 높일 것으로 기대된다.

References

[1] https://www.image-net.org/challenges/LSVRC/

[2] Alex Krizhevsky, Ilya Sutshever, Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Communications of the ACM,* Vol.60, No.6, pp.84-90, 2017.
DOI: 10.1145/3065386.

[3] Kwang Yeob Lee, “Design of a High-Performance Mobile GPGPU with SIMT Architecture based on a Small-size Warp Scheduler,” *j.inst.Korean. electr.electron.eng,* Vol.25, No.3, pp.479-484, 2021.
DOI: 10.7471/ikeee.2021.25.3.479

[4] Ahmad Lashgar, A. Baniasadi, & A. Khonsari. “Investigating Warp Size Impact in GPUs. Computer Sciencear,” ArXiv:1205.4967, 2012.

[5] Cheol-Won Jo, Kwang-Yeob Lee, Chi-Yong Kim, “Low-area DNN Core using data reuse technique,” *j.inst.Korean.electr.electron.eng,* Vol.25, No.1, pp.229-233, 2021.
DOI: 10.7471/ikeee.2021.25.1.229

[6] Chen, Yu-Hsin, Joel Emer, and Vivienne Sze. “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,” *43rd ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2016.
DOI: 10.1109/ISCA.2016.40

[7] Firas Al-Ali, Thilina Doremure Gamage, Hewa WTX Nanayakkara, Farhad Methdipour, Sayan Kumar Ray, “Novel Casestudy and Benchmarking of AlexNet for Edge AI: From CPU and GPU to FPGA,” *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2020.
DOI: 10.1109/ CCECE477 87.2020.9255739,

[8] Sunayana Arya, Rajeev Singh, “A Comparative Study of CNN and AlexNet for Detection of Disease in Potato and Mango leaf,” 2019 2nd International Conference on Issues and Challenges in Intelligent Computing Techniques(ICICT), Vol.1, pp.1-6, 2019.
DOI: 10.1109/ICICT46931.2019.8977648,

BIOGRAPHY

Ki-Hun Nam (Member)

1999 : BS degree in Computer Engineering, Seokyeong University.
2001 : MS degree in Computer Engineering, Seokyeong University.
2006 : PhD degree in Computer Engineering, Seokyeong University.
2006~2009 : Associate Research Engineer, Information Display Research Institute in Hanyang University.
2009~2010 : Associate Research Enginee, BK21 University Specialization Projects in Chungbuk University.
2011~ : Professor, Seokyeong University, Dept. of Computer Engineering

Kwang-yeob Lee (Life Member)

1985 : BS degree in Electronics Engineering, Sogang University
1987 : MS degree in Electronics Engineering, Yonsei University.
1994 : PhD degree in Electronics Engineering, Yonsei University.
1989~1995.2 : Senior Researcher, Hyundai Electronics Inc.
1995.3~present : Professor, Dept. of Computer Engineering, Seokyeong University

Jun-Mo Jung (Member)

1985 : BS degree in Electronics Engineering, Hanyang University.
1987 : MS degree in Electronics Engineering, Hanyang University.
1992 : PhD degree in Electronics Engineering, Hanyang University.
1991~1995 : Assistant Professor, Bucheon University
1995~ Seokyeong Univeristy, Dept. of Electornics Engineering, Professor
<Research interests> VLSI Circuits Design and Test, Embedded System and Processor