The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 19, No. 3, pp.165-171, Jun. 30, 2019. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2019.19.3.165
JIIBC 2019-3-22

텐서 처리부의 분석 및 파이썬을 이용한 모의실행

Analysis of Tensor Processing Unit and Simulation Using Python

이종복*

Jongbok Lee*

요 약 컴퓨터 구조의 연구 결과, 특정 영역의 하드웨어를 개발하는 과정에서 가격 대 에너지 성능의 획기적인 개선이 이뤄진다고 알려져 있다. 본 논문은 인공신경망(NN)의 추론을 가속화시킬 수 있는 텐서 처리부(TPU) ASIC에 대한 분석을 수행하였다. 텐서 처리부의 핵심장치는 고속의 연산이 가능한 MAC 행렬곱셈기와 소프트웨어로 관리되는 온칩 메모리이다. 텐서 처리부의 실행모델은 기존의 CPU와 GPU의 실행모델보다 인공신경망의 반응시간 요구사항을 제대로 충족시킬 수 있으며, 수많은 MAC과 큰 메모리를 장착함에도 불구하고 면적이 작고 전력 소비가 낮다. 텐서플로우 벤치마크 프레임워크에 대하여 텐서 처리부를 활용함으로써, CPU 또는 GPU보다 높은 성능과 전력 효율을 나타낼 수가 있다. 본 논문에서는 텐서 처리부를 분석하고, 파이썬을 이용하여 모델링한 OpenTPU에 대하여 모의실행을 하였으며, 그 핵심장치인 행렬 곱셈부에 대한 합성을 시행하였다.

Abstract The study of the computer architecture has shown that major improvements in price-to-energy performance stems from domain-specific hardware development. This paper analyzes the tensor processing unit (TPU) ASIC which can accelerate the reasoning of the artificial neural network (NN). The core device of the TPU is a MAC matrix multiplier capable of high-speed operation and software-managed on-chip memory. The execution model of the TPU can meet the reaction time requirements of the artificial neural network better than the existing CPU and the GPU execution models, with the small area and the low power consumption even though it has many MAC and large memory. Utilizing the TPU for the tensor flow benchmark framework, it can achieve higher performance and better power efficiency than the CPU or CPU. In this paper, we analyze TPU, simulate the Python modeled OpenTPU, and synthesize the matrix multiplication unit, which is the key hardware.

Key Words : neural network, machine learning, matrix multiply unit

Ⅰ. 서 론

최근에 클라우드에 저장하는 빅데이터와 수많은 고성능 컴퓨터에 의하여 기계학습(machine learning)이 각광을 받고 있다. 특히, 딥뉴럴네트워크(DNN)에 의하여 음성인식과 영상인식에서 오차율을 각각 30 %와 23 %

*정회원, 한성대학교 전자정보공학과
접수일자 2019년 5월 1일, 수정완료 2019년 6월 1일
게재확정일자 2019년 6월 7일

Received: 1 May, 2019 / Revised: 1 June, 2019 /
Accepted: 7 June, 2019
*Corresponding Author: jblee@hansung.ac.kr
Dept. of EI Engineering, Hansung University, Korea

만큼 줄이는 획기적 돌파구를 마련하는 계기가 되었다. 딥뉴럴네트워크는 광범위한 문제에 적용이 가능하므로, DNN 전용 ASIC 하드웨어를 음성, 시각, 언어, 번역, 검색 등에 재활용할 수 있다[1].

신경망은 입력에 대한 가중치합의 비선형함수로 표현되는 간단한 신경세포를 기반으로 하며, 인간의 두뇌와 같은 기능을 겨냥한다. 이러한 신경세포를 모아서 신경망 층을 구성하고, 한 신경망 층의 출력을 순차적으로 다음 신경망 층의 입력으로 공급한다. 이 때 클라우드의 빅데이터 세트를 이용하여 추가의 대규모 신경망 층을 구성할 수 있으며, GPU에 의하여 충분한 계산이 가능하므로 DNN의 깊이가 가능해진다.

신경망은 학습(훈련)과 예측(추론)의 2 가지 단계로 구성되며, 각각 개발과 생산에 해당된다. 개발 단계에서 신경망 층의 개수와 신경망의 유형을 선택하며, 학습에 의하여 가중치를 결정한다. 오늘날 모든 학습은 실수형 데이터를 이용하기 때문에 GPU가 인기를 얻게 되었으나, 양자화의 단계를 거치면 실수형 데이터를 8 비트 정수형 데이터로 변환할 수 있으며, 정수형 데이터를 이용하더라도 신경망으로 추론하기에는 문제가 없다. 본 연산에 필요한 8 비트 정수형 곱셈기는 IEEE 754 16 비트 실수형 곱셈기에 비하여 전력과 면적이 1/6에 불과하며, 정수형 덧셈기는 실수형 덧셈기에 비하여 전력이 1/13이고 면적은 1/38에 그치기 때문에 매우 효율적이다.

최근에 가장 많이 쓰이는 신경망은 크게 3 가지 유형으로 나뉜다. 첫 번째인 다층퍼셉트론 신경망 (MLP)에서 각 신경망 층은 선행하는 신경망 층의 모든 출력의 가중치 합계의 비선형 함수 집합으로서, 가중치는 재사용된다. 두 번째, 콘볼루션 신경망 (CNN)에서 각 신경망 층은 선행하는 신경망 층의 출력과 공간적으로 인접한 부분집합의 가중치 합계의 비선형 함수 집합이며 가중치는 역시 재사용된다. 세 번째로 순환 신경망(RNN)에서 각 신경망 층은 출력과 이전 상태의 가중치 합계의 비선형 함수 집합이다.

신경망에서 비선형함수를 사용하는 이유는 신경망의 출력을 0과 1 또는 −1과 1 사이에서 결정하기 위해서이다. 출력을 결정할 때, 비선형 함수를 사용하여 다양한 데이터에 대하여 적용하고 일반화시킬 수 있으며 출력을 구분할 수 있다. 현재 콘볼루션 신경망과 딥러닝에 가장 많이 쓰이는 비선형함수는 ReLU 함수로서, z가 0보다 작을 때는 출력이 0이고, z가 0보다 크거나 같을 때는 출력이 z이다.

신경망은 주로 TensorFlow 프로그램을 이용하여 개발하는데, 코드의 길이는 100 줄에서 1500 줄 미만으로 그리 길지가 않으며, 각 모델은 500 만에서 1 억 개의 가중치를 이용하므로 시간과 전력소모가 많다. 따라서 학습 및 추론 과정에서 서로 다른 예제에 동일한 가중치를 재사용하여 성능을 높일 수 있다.

텐서 처리부 (TPU)는 신경망 학습 용도로 개발된 인공지능 가속 ASIC으로서, 구글에 의하여 2016 년도에 발표되었다[1]. 본 논문에서는 텐서 처리부에 대하여 기술하고 분석하며, 파이썬을 이용하여 모의실행하고, 핵심장치인 행렬 곱셈부를 합성하였다. 본 논문은 다음과 같이 구성된다. 2 장에서 기존의 신경망 하드웨어 연구에 대하여 되짚어본다. 3 장에서 텐서처리부의 특징과 명령어를 분석 및 고찰한다. 4 장에서 모의실험 환경 및 결과를 보이고, 5 장에서 결론을 맺는다.

Ⅱ. 기존의 신경망 하드웨어 연구에 대한 고찰

최초의 신경망 하드웨어는 최소한 25년 전으로 거슬러 올라가는데[2], CNAPS 칩은 16 비트 × 8 비트 곱셈기의 64 개 SIMD 배열로 구성되었으며, 시퀀서와 연결하였다[3]. 한편, Synapse−I 시스템은 MA−16이라는 시스톨릭 곱셈 및 누산 칩을 기반으로 하였는데, 한 번에 16 개의 16 비트 곱셈을 수행할 수 있다[4]. MA−16 칩을 여러 개 연결하여 기존의 하드웨어를 이용하여 활성기능을 수행했다.

1995년도에 개발된 T0 ASIC은 MIPS 명령어 세트에 벡터명령어를 추가한 것으로, 이것에 의하여 가속된 25 개의 SPERT−II 워크스테이션이 음성인식용 신경망의 훈련과 추론에 배치되었다[5]. 8 개의 레인으로 구성된 벡터는 8 비트와 16 비트 입력에 대하여 매 싸이클 당 16 개의 32 비트 결과를 연산하여 SPARC−20 워크스테이션에 비하여 학습을 20 배, 추론을 25 배 빨리할 수 있다. 이 때, 학습을 하기 위하여 16 비트가 부족하다고 판단하여 두 개의 16 비트 워드를 이용한 결과, 학습 시간이 두 배 소요되었다. 이러한 단점을 극복하기 위하여 32 개에서 1000 개의 데이터 세트에 대한 배치작업을 수행하여 가중치를 업데이트하는 시간을 단축하였다.

III. 텐서 처리부의 구조 및 구현

1. 배경 및 텐서 처리부의 블럭도

2013년도에 개인이 음성검색을 하루에 3분 이상 이용함에 따라, 이것을 충족시키기에 DNN을 활용했을 때 기존 CPU의 두 배의 성능이 필요하게 되었다. 따라서, 구글에서 가격대 성능이 일반 GPU의 10 배를 만족시킬 수 있는 ASIC인 텐서 처리부가 설계되었다. 최초의 텐서 처리부 프로젝트는 FPGA에서 출발하였지만, 그 당시 FPGA는 성능 면에서 GPU에 미치지 못하였다. 그러나, 점차 텐서 처리부가 GPU보다 낮은 전력소비와 빠른 속력으로 FPGA와 GPU를 능가하였기 때문에 결국 채택되었고 성공할 수 있었다. 텐서 처리부의 연산과정 중 활성 데이터는 DRAM에 저장되며, 통합버퍼 (Unified Buffer)는 적절한 크기로 설계되므로, 정상동작시 레지스터의 데이터를 DRAM으로 보내거나 다시 가져오는 일을 방지하였다.

텐서 처리부는 지연시간을 줄이기 위하여, CPU와 밀결합시키지 않고, GPU처럼 기존의 호스트 서버에 장착할 수 있도록 PCI 입출력버스용 보조프로세서로 설계되었다. 또한, 하드웨어 설계와 디버깅을 간단화하기 위하여 텐서 처리부가 직접 텐서 처리부 명령어를 인출하지 않고, 호스트 서버가 텐서 처리부로 명령어를 보내서 텐서 처리부가 실행하도록 하였다. 즉, 텐서 처리부의 인터페이스는 호스트 CPU와의 상호작용을 최소화하고 융통성이 있도록 설계하는 것을 목표로 하였으며, 그림 1에 텐서 처리부의 블럭도를 나타냈다[1].

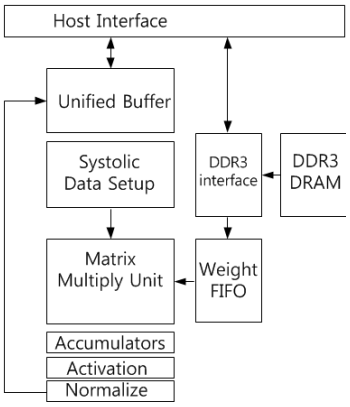

그림 1. 텐서 처리부의 블럭도
Fig. 1. The block diagram of TPU

텐서 처리부 명령어는 PCIe 버스를 통하여 호스트로부터 명령어 버퍼로 전송된다. 내부블럭은 256 바이트 넓이의 경로로 서로 연결되며, 블럭도의 우측 상단에 위치한 행렬 곱셈부 (Matrix Multiply Unit)가 텐서 처리부의 핵심장치이다.

2. 텐서처리부의 행렬 곱셈부

행렬 곱셈부는 256×256 MAC으로 구성되는데, 이것은 부호화 또는 부호화되지 않은 8 비트 정수에 대하여 8 비트 곱셈과 덧셈을 수행한다. 이 때 생성된 16 비트는 행렬 곱셈부의 아래에 위치한 4MB 32 비트 누산기로 출력되는데, 이것은 32 비트 누산기 256 개의 요소를 4096 개 모아서 설계되었다. 행렬 곱셈부는 매 클럭 싸이클마다 한 개의 256 요소의 부분합을 생성하는데, 4096 개로 설계한 이유는 최고성능에 도달하기 위한 바이트 당 연산수가 1350이므로 2048로 잡고, 컴파일러가 더블버퍼링을 이용할 수 있도록 그 두 배로 잡았기 때문이다.

8 비트의 가중치와 16 비트의 함수를 혼용할 때 행렬 곱셈부는 연산 소요 시간이 2 배로 늘어나며, 가중치와 함수가 둘 다 16 비트인 경우에는 그 값이 4 배가 된다. 행렬 곱셈부는 매 싸이클 당 256 개의 데이터를 읽고 쓰면서 행렬곱셈이나 콘볼루션을 수행할 수 있다. 행렬 곱셈부는 가중치 타일 64 KB와 한 개의 타일을 쉬프트 시킬 때 소요되는 256 싸이클을 숨기기 위한 더블버퍼링 동작을 위하여 64 KB를 갖춘다. 본 행렬 곱셈부는 밀도가 높은 행렬을 겨냥하여 설계됨에 따라 희박한 행렬은 시간상의 이유로 제외되었으므로, 차세대 설계에서 고려되어야 한다.

행렬 곱셈부에 대한 가중치는 온칩 FIFO로 설계되었는데, 추론을 할 때 가중치 메모리인 8 GB DRAM으로부터 가중치 데이터를 읽는다. 가중치 FIFO는 4 개의 타일로 구성되며 연산의 중간결과는 24 MB 온칩 통합버퍼에 저장되는데, 행렬 곱셈부의 입력에 공급된다. 프로그래밍이 가능한 DMA 제어기가 CPU의 호스트 메모리와 통합버퍼 사이의 데이터를 전송한다[1].

3. 텐서 처리부의 명령어

텐서 처리부는 마치 SATA 디스크를 서버에 삽입하듯이 PCB에 장착할 수 있으며, 텐서 처리부의 명령어는 PCIe 버스를 통하여 전송된다. 약 12 개로 구성되는 텐서 처리부의 명령어중에서 5 개의 명령어가 핵심이며, 명령어 당 소요 클럭 싸이클 수인 CPI는 10에서 20 범위의 값을 가진다.

Read_Host_Memory 명령어는 CPU 호스트 메모리로부터 데이터를 읽어서 통합버퍼로 전송한다. Read

Weights 명령어는 가중치 메모리로부터 가중치를 읽어서 가중치 FIFO로 전송하여 행렬 곱셈부의 입력을 구성한다. MatrixMultiply 및Convolve 명령어는 행렬 곱셈부가 행렬의 곱셈이나 콘볼루션 연산을 수행하여 통합버퍼에서 누산기로 전송하도록 한다. 이 때 행렬의 곱셈은 B 바이트 크기를 갖는 256 개 입력에 256×256 가중치 상수를 곱하여 B×256 출력을 생성하며, 완료하는데 B 싸이클이 소요된다. Activate 명령어는 인공신경세포의 비선형함수 연산인 ReLU, Sigmoid 등을 수행하는데, 이 때의 입력은 누산기이고 출력은 통합버퍼이다.이 명령어는 비선형함수 논리회로에 연결되어있으므로, 전용 하드웨어를 이용하여 콘볼루션 연산도 지원한다. 마지막으로, Write_Host_Memory 명령어는 통합버퍼의 데이터를 CPU 호스트 메모리로 전송한다.

그 외의 기타 명령어들은 다른 호스트 메모리에 대한 읽기/쓰기, 설정, 동기화, 인터럽트, 디버그, NOP, HALT이다. Matrix/Multiply 명령어는 12 바이트로 그 중에 3 바이트는 통합버퍼의 주소, 2 바이트는 누산기의 주소, 4 바이트는 길이이고 나머지 3 바이트는 Opcode와 플래그이다.

텐서 처리부 구조의 기본개념은 행렬 곱셈부를 중단하지 않고 가동하는 것으로서, 명령어에 대하여 4 단계 파이프라인을 적용하였다. 이 때 다른 명령어의 실행을 Matrix Multiply 명령어와 겹치게 하여 성능을 높였다. Read_Weights 명령어는 접근과 실행을 분리하는 개념으로 동작하여, 주소를 전송하고 가중치 메모리로부터 가중치를 인출하기 이전에 완료가 가능하다. 곱셈행렬부는 입력 데이터나 가중치 데이터가 준비가 되어있지 않으면 멈춘다. 텐서 처리부 명령어는 파이프라인으로 동작하는 RISC 명령어와 달리 CISC 명령어처럼 동작하여 수천 클럭 싸이클이 소요될 수가 있다.

4. 시스톨릭 처리 방식

대규모 SRAM을 읽어들임에 따르는 전력소모를 줄이기 위하여, 행렬 곱셈부는 통합버퍼로부터의 읽기와 쓰기를 줄여서 에너지를 절감하는 시스톨릭(systolic) 실행방식으로 운영된다. 시스톨릭 실행방식이란, 서로 다른 방향에서 세포배열에 도착하는 데이터들을 일정한 간격으로 결합시켜서 연산을 수행하는 방법이다. 그림 2에서 데이터가 왼쪽에서 흘러들어오고, 가중치는 위에서 적재되는 것을 나타내고 있다. 이 때, 256 요소에 대한 곱셈과 덧셈이 행렬을 대각선으로 이동하면서 수행된다. 미리 적재된 가중치는 데이터가 이동함에 따라 새로운 블럭의 첫 번째 데이터와 함께 작용한다. 제어와 데이터가 파이프라인 방식으로 동작하여 256 개의 입력을 동시에 읽고, 각 256 개의 누산기의 위치를 순간적으로 업데이트한다.

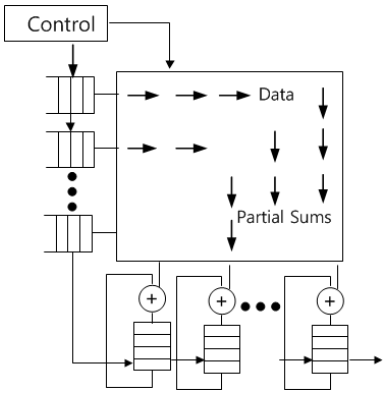

그림 2. 행렬 곱셈부의 시스톨릭 데이터의 흐름
Fig. 2. The Systolic Data Flow of Matrix Multiply Unit

5. 텐서 처리부와 소프트웨어

텐서 처리부 소프트웨어 스택은 CPU 및 GPU와 호환가능하도록 하여 응용프로그램을 텐서 처리부에 신속하게 이식할 수 있도록 하였으며, 텐서 처리부에서 실행되는 응용프로그램의 부분은 TensorFlow를 이용하여 프로그래밍되어 CPU 및 GPU에서 수행 가능한 API로 컴파일된다. GPU와 마찬가지로 텐서 처리부 스택은 사용자 공간 드라이버와 커널 드라이버로 구성되는데, 경량인 커널 드라이버는 장기간 동안 바뀌지 않으며, 메모리관리와 인터럽트만 처리한다. 반면에, 사용자 공간 드라이버는 자주 변경되며 텐서 처리부의 실행을 설정, 제어하고 데이터를 텐서 처리부의 순서대로 재형성하며, API 호출을 텐서 처리부 명령어로 번역하여 응용 프로그램의 기계어로 전환한다.

IV. 텐서 처리부 모의실험 환경 및 결과

1. 모의실험 환경

본 논문의 모의실험은 운영체제 Ubuntu 14.04 LTS에서 3.1 GHz로 동작하는 Intel Core i7-950 데스크탑 PC에서 시행하였다. 모의실험은 UCSB ArchLab 연구실에서 구글의 텐서 처리부를 재구현한 OpenTPU를 기반으로 하였으며[6], 모의실험 환경 구축을 위하여 Python

3.4.3, PyRTL 0.8.6, Numpy 1.8.2를 이용하였다.

OpenTPU 하드웨어의 모의실행은 PyRTL의 기능을 이용하였다. 이 때 runtpu.py를 실행시키는데, 입력으로 어셈블리 프로그램, 초기 호스트 메모리 및 가중치를 갖는 numpy 배열 화일을 이용한다. 이 때, 하드웨어 행렬 곱셈부는 파라미터를 입력함으로써 변경이 가능하다. OpenTPU의 기능적 모의실행은 sim.py 파이썬 코드로 구현되었다. 이 코드는 어셈블리 프로그램, 호스트 메모리 화일, 가중치 화일로 구성되는 세 개의 인자를 읽어들인다. TensorFlow로 작성된 고급 응용 프로그램과 OpenTPU 간의 상이한 양자화 원리를 고려하여 모의실험기는 32 비트 실수형 모드와 8 비트 정수형 모드 두 가지로 실행되며, 출력 역시 32 비트 실수형과 8 비트 정수형의 두 가지 세트로 내보낸다.

2. 명령어 집합

표 1에 OpenTPU 명령어 집합을 나타냈다[6]. RHM 명령어는 호스트 메모리의 src 주소에서 N 개의 벡터를 읽어서 통합버퍼의 dst 주소에 쓴다. 이와 반대로, WHM 명령어는 통합버퍼의 src 주소에서 N 개의 벡터를 읽어서 호스트 메모리의 dst 주소에 쓴다. RW 명령어는 DRAM의 주소 addr로부터 가중치 타일을 읽어서 온칩 FIFO에 적재한다.

MMC 명령어는 통합버퍼의 src 주소의 N 개의 벡터에 대하여 행렬곱셈을 수행하고, 누산기 버퍼의 dst 주소에 결과를 저장한다. 만일에 O 플래그가 생략되면 연산결과를 누산기의 결과에 더하여 저장하고, O 플래그가 포함되면 누산기 버퍼의 내용을 덮어 쓰기 한다. 만일에 S 플래그가 포함되면, 미리 적재되어 있을 새로운 가중치 화일로 전환한다. 따라서, 최초의 MMC 명령어는 반드시 S 플래그를 지정해야한다.

표 1. OpenTPU의 명령어 집합
Table 1. The instruction set of the OpenTPU

명령어	설명
RHM src, dst, N	호스트 메모리 읽기
WHM src, dst, N	호스트메모리 쓰기
RW addr	가중치 읽기
MMC src, dst, N	행렬 곱셈 및 콘볼루션
ACT src, dst, N	활성화
NOP	아무 연산을 하지 않음
HLT	정지

ACT 명령어는 누산기 버퍼 내 src 주소의 N 개의 벡터를 활성화하여 그 결과를 통합버퍼의 dst 주소에 저장한다. 활성함수는 플래그로 지정하는데, R은 ReLu이고 Q는 Sigmoid이다. 만일에 플래그가 생략되면 활성화를 하지 않고 값들을 그대로 통과시킨다. 합성할 때 정규화는 프로그래밍이 가능하지만, 실행할 때는 가능하지가 않다. 기본적으로 활성화가 된 이후에 상위 24 비트가 탈락하여 8 비트 정수형 데이터를 생성한다.

3. 마이크로 구조

다음은 OpenTPU의 마이크로구조에 대해서 고찰한다[6]. OpenTPU의 핵심은 파라미터에 따라 설정이 가능한 8 비트 곱셈누산기 (MAC) 배열로서, 각 배열은 8 비트 정수형 곱셈기와 16 비트에서 32 비트 사이의 정수형 덧셈기로 구성된다. 각 곱셈누산기는 8 비트의 가중치를 저장하는 두 개의 버퍼를 설치하여 가중치 프로그래밍을 병렬로 수행할 수 있다. 입력벡터는 배열의 왼쪽에서 진입하고 각 싸이클마다 오른쪽으로 한 곱셈누산기씩 전진한다. 각 곱셈누산기에서 입력값에 활성화된 가중치 값을 곱하고 윗 쪽의 곱셈누산기와 더하여 아래쪽 누산곱셈기에 전달한다. 이 때, 입력벡터는 대각선 방향으로 공급이 되어, 부분합이 배열을 따라 흘러내릴 때 값들이 올바르게 정렬되도록 한다. 곱셈기는 16 비트의 출력을 내며, 각 배열의 열을 따라서 값들이 아래로 이동할 때 폭을 32로 제한하여 연산과정에서 오버플로우가 발생하지 않도록 한다.

행렬곱셈 배열에서 출력된 결과벡터 값들은 소프트웨어에서 지정된 주소로 누산기 버퍼 내에 저장되며, 명령어에 따라 지정된 주소에 결과값을 더하거나, 덮어쓰기를 한다. 행렬곱셈 명령어는 통합버퍼로부터 데이터를 읽어서 누산기버퍼에 데이터를 쓰며, 활성화된 명령어는 누산기버퍼로부터 데이터를 읽어서 통합버퍼에 데이터를 쓴다.

곱셈누산기가 256×256 개일 때 가중치의 행렬을 타일이라 부르고 크기는 64 KB이다. 가중치들이 오프칩 가중치 DRAM에서 이동할 때 정지되는 것을 방지하기 위하여 버퍼 타일에 길이 4의 FIFO가 이용된다. 가중치 DRAM에 연결할 때, 데이터를 64 B 단위로 이동하는 표준 DDR 인터페이스를 가정한다.

만일에 행렬곱셈 명령어가 스위치 플래그를 포함하면, 각 덧셈 누산기들은 배열을 전파하는 명령어의 첫 번째 벡터로 활성 가중 버퍼를 전환시킨다. 행렬곱셈 명령어가 첫 번째 행의 끝에 도착하면 FIFO는 배열의 빈 버퍼에

새로운 가중치 값들을 공급하기 시작한다. 새로운 가중치 값들은 각 행의 목적지에 도달할 때까지 매 싸이클마다 배열을 통하여 전파된다.

벡터들은 통합버퍼로부터 한 번에 읽혀지나, 행렬 곱셈부에는 대각선 방향으로 공급이 되는데, 이것은 순차버퍼의 세트를 삼각형으로 구성함으로써 구현이 된다. 첫 번째 값은 행렬에 즉시 도착하고, 두 번째 값은 1 싸이클 후에, 세 번째 값은 2 싸이클 후에 도착함으로써, 같은 소오스 벡터의 대응하는 부분합과 동시에 도착하도록 한다.

메모리 제어기는 에뮬레이션으로 수행하며 지연이 없다. 호스트 메모리에 대한 연결 넓이는 한 개 벡터의 크기이며, 가중치 DRAM에 대한 연결은 64 B의 표준 크기를 이용한다. 에뮬레이션으로 동작하는 메모리 제어기는 매 싸이클마다 새로운 값을 반환해주기 때문에 OpenTPU 하드웨어 시뮬레이션은 고정적인 메모리 접근 지연시간을 가정한다.

4. 모의실험 결과 및 합성

표 2에 모의실험에 이용된 간단한 행렬의 곱셈을 수행하는 OpenTPU의 어셈블리어 프로그램을 나타냈다. 어셈블리어 프로그램 1 번 코드는 길이가 8인 호스트 메모리의 주소 0 번지 데이터를 통합버퍼의 주소 0번지로 읽어 들이는 명령을 실행한다. 2 번 코드는 DRAM 주소 0 번지에서 가중치를 FIFO로 읽어 들인다. 3 번 코드에서 통합버퍼 0번지의 길이 8인 데이터에 대하여 행렬곱셈을 수행하고 누산기 주소 0 번지에 저장하며, 4 번 코드에서 누산기 0 번지의 길이 8인 데이터에 대하여 ReLU를 적용시킨다. 5 번 코드에서 통합버퍼의 주소 0 번지에서 길이 8인 데이터를 호스트메모리의 주소 0 번지로 쓰고, 6 번 코드에서 실행을 멈춘다. 표 2에는 위 어셈블리어 프로그램을 실행하기 전의 호스트메모리, 가중치 및 실행이 끝난 후의 호스트메모리를 8×8 행렬의 형태로 함께 나타냈다. 위의 간단한 행렬의 곱셈 실행을 위하여 429 싸이클이 소요되었으며, 올바르게 결과가 나온다는 것을 알 수 있다.

마지막으로, 텐서처리부의 핵심장치인 행렬 곱셈부를 파이썬의 OutputToVerilog 기능과 PyRTL 0.8.6을 연동하여 Verilog 코드로 변환하였고, 이것을 다시 버클리 대학에서 개발한 Yosys 0.7에 입력하여 Xilinx FPGA를 목표로 합성하였다. Yosys 0.7로 합성한 결과를 표 3에 상세하게 나타냈는데, 행렬 곱셈부의 합성에 이용된 셀의 수는 10,016 개, 비트의 수는 16,113 개, 배선의 수는 7,770 개를 기록하였다.

표 2. OpenTPU 모의실행 결과
Table 2. The simulation result of OpenTPU

	어셈블리 프로그램	가중치
프로그램 및 가중치	1 RHM 0, 0, 8 2 RW 0 3 MMC.S 0, 0, 8 4 ACT.R 0, 0, 8 5 WHM 0, 0, 8 6 HLT	(126,135,135,135,135,135,135,1 135,126,135,135,135,135,135,1 135,135,126,135,135,135,135,1 135,135,135,126,135,135,135,1 135,135,135,135,126,135,135,1 135,135,135,135,135,126,135,1 135,135,135,135,135,135,126,1 135,135,135,135,135,135,135,1
	실행 전	실행 후
호스트 메모리	(3, 4, 0, 253, 254, 253, 253, 6 252, 1, 1, 255, 251, 253, 255, 6 252, 255, 253, 253, 0, 255, 2, 6 255, 4, 0, 254, 2, 4, 3, 6 2, 1, 2, 4, 254, 255, 4, 64 252, 3, 3, 253, 0, 4, 64	(100, 0, 0, 0, 0, 0, 0, 0 128, 0, 0, 0, 0, 0, 0, 0 44, 0, 0, 0, 0, 0, 0, 0 58, 0, 0, 0, 0, 0, 0, 0 198, 0, 0, 0, 0, 0, 0, 0 198, 0, 0, 0, 0, 0, 0, 0 72, 0, 0, 0, 0, 0, 0, 0 21, 0, 0, 0, 0, 0, 0, 0)

표 3 행렬 곱셈부를 Xilinx FPGA로 합성한 결과
Table 3. The XilinX FPGA synthesys report of Matrix Multipy Unit

합성에 이용된 FPGA 내부 요소	값
배선의 수	7,770
비트의 수	16,113
전역 배선의 수	649
전역 배선 비트의 수	8,114
셀의 수	10,016
FDRE	1,472
LUT1	21
LUT2	1,147
LUT3	387
LUT4	338
LUT5	982
LUT6	2,897
MUXCY	598
MUXF7	1,184
MUXF8	348
RAMB18E1	8
XORCY	634

FDRE는 클럭 인에이블과 동기 리셋을 갖는 D 플립플롭이다. LUT1은 일반 출력을 갖는 1 비트 테이블 참조(Look-Up Table) 회로이며, LUT2, LUT3, LUT4, LUT5, LUT6는 입력비트 수가 각각 첨부된 숫자인 것만 제외하고는 LUT1과 같다. MUXCY는 일반출력의 캐리 논리에 대한 2 대 1 멀티플렉서이고, MUXF7과 MUXF8은 일반출력을 갖는 2 대 1 테이블 참조 멀티플렉서이다. RAMB18E1은 18K 비트의 구성 가능한 동기 블럭 RAM이며, XORCY는 캐리 논리에 대한 XOR 논리기능을 수행하며 일반출력을 갖는다.

V. 결론

본 논문에서는 기존의 신경망 하드웨어를 되짚어보고, 구글의 텐서 처리부를 고찰 및 분석하였다. 또한, 텐서 처리부를 파이썬으로 모델링한 OpenTPU를 이용하여 모의실행을 하였다. 나아가서, 텐서 처리부의 핵심장치인 행렬곱셈부를 Xilinx FPGA를 목표로 합성하였다.

추후로, 행렬 곱셈부에 결여되어있는 콘볼루션 기능을 포함시키고, 행렬 곱셈부뿐만이 아니라 전체 텐서처리부에 대하여 Verilog 코드를 생성한 후에 합성하여 타이밍 시뮬레이션을 거친 후에, 정적 시간 분석 (STA)과 합성 후 모의실험(Post synthesis simulation)을 거쳐 최종적인 동작을 검증하고 FPGA로 프로그래밍하여 동작을 검증할 예정이다. FPGA로 검증을 완료한 후에는, Synopsis로 합성하여 국내 기관인 IDEC을 통하여 ASIC 칩으로 구현할 예정이다.

References

[1] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," 44th International Symposium on Computer Architecture (ISCA), Jun. 2017.

[2] P. Ienne, T. Cornu, and G. Kuhn, "Special Purpose digital hardware for neural networks: An architectural survery," Journal of VLSI signal processing systems for signal, image and video technology, Vol. 1, No. 13, 1996.

[3] D. Hammerstrom, "A VLSI Architecture for high-performnace, low-cost, on-chip learning," International Joint Conference on Neural Networks, Jun. 1990.

[4] U. Ramacher et. al., "Design of a 1st Generation Nerocomputer," VLSI design of Neural Networks. 1991.

[5] K. Asanovik et. al, "Training Neural Networks with Spert-II," Parallel Architectures for Artificial Networks : Paradigm and Implementations, Nov. 1998.

[6] https://github.com/UCSBarchlab/OpenTPU

저 자 소 개

이 종 복(정회원)

- 1964년 8월 20일생.
- 1988년 : 서울대 컴퓨터공학과 졸업.
- 1998년 : 동 대학 전기공학부 졸업(공박).
- 1998 ~ 2000 : LG반도체 선임연구원.
- 2000년 ~ 현재 : 한성대 전자정보공학과 교수
- Tel : 02-760-4497
- Fax : 02-760-4435
- E-Mail : jblee@hansung.ac.kr
- 관심분야 : 마이크로 프로세서, 멀티코어 프로세서, 텐서 프로세서 유닛.

※ 본 연구는 한성대학교 교내학술연구비 지원과제임.