JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 6, pp. 1-12, June 2024
https://doi.org/10.9708/jksci.2024.29.06.001

Resolving Memory Bottlenecks in Hardware Accelerators with Data Prefetch

Hyein Lee*, Jinoo Joung**

*Student, School of Intelligent Information Engineering, SANGMYUNG University, Seoul, Korea
**Professor, Dept. of Intelligent Information Engineering, SANGMYUNG University, Seoul, Korea

[Abstract]

Deep learning with faster and more accurate results requires large amounts of storage space and large computations. Accordingly, many studies are using hardware accelerators for quick and accurate calculations. However, the performance bottleneck is due to data movement between the hardware accelerators and the CPU. In this paper, we propose a data prefetch strategy that can efficiently reduce such operational bottlenecks. The core idea of the data prefetch strategy is to predict the data needed for the next task and upload it to local memory while the hardware accelerator (Matrix Multiplication Unit, MMU) performs a task. This strategy can be enhanced by using a dual buffer to perform read and write operations simultaneously. This reduces latency and execution time of data transfer. Through simulations, we demonstrate a 24% improvement in the performance of hardware accelerators by maximizing parallel processing with dual buffers and bottlenecks between memories with data prefetch.

▸ **Key words**: HW Accelerator, CNN, Bottleneck, Data Prefetch, Dual Buffer, RTL Simulation

[요 약]

최근 다양한 분야에서 딥러닝이 사용되면서, 더 빠르고 정확한 결과를 내는 딥러닝이 더욱 중요해졌다. 이를 위해서는 많은 양의 저장 공간이 필요하고, 대용량 연산을 진행해야 한다. 이에 따라 여러 연구는 빠르고 정확하게 연산 처리가 가능한 하드웨어 가속기를 이용한다. 하지만 하드웨어 가속기는 CPU와 하드웨어 사이를 이동하면서 병목현상이 발생하게 된다. 따라서 본 논문에서는 하드웨어 가속기의 병목현상을 효율적으로 줄일 수 있는 데이터 프리패치 전략을 제안한다. 데이터 프리패치 전략의 핵심 아이디어는 Matrix Multiplication Unit(MMU)가 연산을 진행하는 동안 다음 연산에 필요한 데이터를 예측하여 로컬 메모리로 올려 병목현상을 줄인다. 또한, 이 전략은 듀얼 버퍼를 이용하여 읽고 쓰는 두 가지 동작을 동시에 진행하여 처리율을 높인다. 이를 통해 데이터 전송의 지연시간 및 실행 시간을 감소시킨다. 시뮬레이션을 통해 듀얼 버퍼를 이용한 병렬 프로세싱과 데이터 프리패치를 이용한 메모리 간 병목현상을 최대한 감소시켜 하드웨어 가속기의 성능이 24% 향상함을 알 수 있다.

▸ **주제어**: 하드웨어 가속기, CNN, 병목현상, 데이터 프리패치, 듀얼 버퍼, RTL 시뮬레이션

• First Author: Hyein Lee, Corresponding Author: Jinoo Joung
*Hyein Lee (shyein106@gmail.com), School of Intelligent Information Engineering, SANGMYUNG University
**Jinoo Joung (jjoung@smu.ac.kr), Dept. of Intelligent Information Engineering, SANGMYUNG University
• Received: 2024. 04. 01, Revised: 2024. 05. 31, Accepted: 2024. 06. 03.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

1. Induction

Deep Neural Network(DNN)는[1] 현재 딥러닝 분야에서 많이 사용되는 모델 중 하나이다. DNN은 최근에 나온 개념은 아니지만, 이미지의 특징을 추출하여 분류하거나 객체 인식[2, 3, 4]하는 등의 이미지 처리 및 자율주행 자동차[4], 복잡한 게임[5] 등 다양한 분야에서 활용되고 있다. 이런 분야에서 인간보다 나은 수준의 성능을 보이기도 하며, 현재 산업 및 학문에 중요한 역할을 한다. DNN에서 응용된 알고리즘인 Convolution Neural Networks(CNN)[6]는 데이터 그중에서도 이미지나 영상의 특징을 추출하여 특징들의 패턴을 파악하는 알고리즘이다. CNN은 2차원의 이미지를 입력받아서 2차원 Convolution 연산을 거쳐, 최종 결론을 내는 구조로 이루어져 있다. 이는 사람이 수동으로 특징을 추출하는 것과 다르게 좋은 성능을 보인다. CNN은 여러 이미지에서 수백 개에서 수천 개의 객체에 대해 학습[6]하기 때문에, 이를 저장할 수 있는 메모리 공간과 많은 양의 연산이 필요하다. 또한, 정확도를 높이기 이미지의 픽셀 개수를 높일수록 연산량과 시간도 늘어나게 된다. 이에 따라 CPU로만 연산을 진행하는 데 큰 비용이 소모되는 등 한계가 생기게 되었고, 방대한 연산량을 효율적으로 처리하기 위해 적절한 컴퓨팅 플랫폼을 선정하는 것이 중요해졌다. 특히 시간이 흐르고 딥러닝이 발전될수록 점점 더 크고 많은 이미지를 처리해야 할 것이다. 이에 따라 훨씬 더 많은 계산이 필요하게 되고, 저전력 동작 조건에서 더 빠른 속도로 더 복잡한 연산을 수행하는 데 사용될 가능성이 높다[7].

이에 따라 순차적으로 처리하는 CPU 대신 병렬적으로 연산을 처리하는 하드웨어에서 연산을 진행하기 시작하였다. 특히 많은 양의 연산을 빠르고 정확하게 진행하기 위해 최근 많은 연구가 하드웨어 가속기를 이용한다. 이는 주로 CPU와 Graphic Processing Unit(GPU)를 사용하는 멀티코어 프로세서 기반의 하드웨어 가속기 또는 ASIC나 FPGA를 사용하는 하드웨어 가속기를 사용한다. 그 중, ASIC나 FPGA를 사용하는 전용 하드웨어 가속기는 몇 가지의 CNN 구조에 최적화되어 지원된다. 따라서 프로세서 기반의 하드웨어 가속기보다는 효율적이지만, 특정 구조의 CNN에만 적용되므로 유연성이 부족하다는 점이 있다. 물론 많은 연구에서 CPU와 Graphic Processing Unit(GPU) 기반의 하드웨어 가속기 역시 사용하고, 개발하고 있다. 하지만 우리 연구에서는 FPGA 기반 가속기를 사용하였고, 현재 많은 연구가 FPGA 기반으로 제공하는 하드웨어 가속기에 관심을 두고 있다. 또한, 이미 다양한 분야, 비디오 및 오디오 처리, 상시 음성 인식, 이미지 신호 처리 등과 같은 작업을 수행하기 위해 많은 하드웨어 가속기를 사용하고 있다[8].

하지만 하드웨어 가속기는 CPU에서 데이터를 전송받아 연산을 진행하는 것이기에 데이터 이동할 때, 지연시간이 발생하게 된다. 그리고 해당 지연시간은 하드웨어 가속기 성능을 저하하는 가장 큰 원인이다. 본 논문에서는 이를 해결하기 위해 최적화된 계산과 병렬 프로세싱으로 데이터 프리패치를 이용하여 길고 가변적인 메모리 지연시간을 감소시킬 수 있는 프레임워크를 설계하였다. 이때 데이터 프리패치는 하드웨어 가속기에 추가로 내부 캐시를 이용하여 데이터를 사전에 올리는 과정이 원활하게 이뤄지도록 하였으며, 이를 통해 메모리 접근하는 과정과 연산하는 과정을 분리하였다. 데이터 프리패치는 메모리 버스에 연결되며, 이를 통해 자체적으로 메모리에 접근하는 구조를 갖는 가속기를 설계할 수 있었다.

II. Preliminaries

최근 인공신경망 및 딥러닝이 빠르게 발전해 나가면서 많은 양의 연산이 필요해지고, 이를 빠르고 정확하게, 효율적으로 연산하기 위한 연구들이 계속되고 있다. 그중에 대표적인 방식이 바로 하드웨어 가속기이다[9]. 하드웨어 가속기는 멀티코어 프로세서 기반의 하드웨어 가속기인 CPU 또는 GPU와 ASIC 또는 FPGA를 사용한 맞춤형 하드웨어 가속기가 있다. CPU/GPU 기반의 하드웨어 가속기는 여러 CNN 구조에 사용되어 다양하게 활용이 가능하고, 다양한 해결 방법을 개발하는데 편리하다. 하지만, 사용할 수 있는 컴퓨팅 자원의 사용과 전력 효율성 측면에서는 비효율적이다. 그에 반해 맞춤형 하드웨어 가속기는 컴퓨팅 자원의 활용도가 훨씬 높지만, 여러 CNN 구조에 사용되기보다는 특정 CNN 구조에만 사용되어 활용도가 낮다[8]. 여러 하드웨어 가속기가 있지만 그중에서 대표적인 하드웨어 가속기의 예시들은 다음과 같다.

1. Tensor Processing Unit(TPU)

구글은 하드웨어 가속기로 자체 기계학습 전용 칩인 Tensor Processing Unit(TPU)를 개발하였다. TPU는 TensorFlow에 맞게 특별하게 ASIC를 이용하여 제작되었으며 기계학습을 위한 맞춤형 가속기로 개발되었다. CPU는 여러 동작을 할 수 있지만, 그에 반해 TPU는 DNN의

계산만을 위한 칩으로 개발되었다. TPU는 주요 연산이 일어나는 Matrix Multiply Unit과 연산에 필요한 값을 미리 제공해 주거나 연산의 결괏값을 저장하는 데이터 버퍼, 각 Unit이 제대로 수행되도록 하는 제어 Unit 등으로 구성되어 있다. 이는 기계학습에 최적화된 성능을 보이며, 이를 구글 데이터 센터 서버에서 Street View, RankBrain 등 서비스에서 사용되었다[10].

2. Graphic Processing Unit(GPU)

GPU는 범용 목적으로 설계된 프로세서로, 대부분의 연구 및 기업에서 많이 사용되고 있다. DNN[1]의 경우 십만 개 이상의 뉴런으로 구성되어 있어 연산량이 매우 많다. 하지만 간단한 연산의 반복이고, 병렬성이 높기 때문에 여러 코어를 가지고 있는 GPU에서 효율적으로 연산을 진행할 수 있다. GPU는 범용 연산기로서 각각 인공 신경망에 대해 최적화되지 않지만, 단순히 반복되는 인공신경망 연산을 빠르게 범용적으로 연산하는 데에는 최적화된 성능을 나타낸다. GPU는 병렬 연산 기능을 범용으로 하는 General Purpose on GPU(GPGPU)가 제안된 후에는 더 빠른 인공신경망 연산이 가능해졌다[9, 11].

대표적인 제조사 NVIDIA에서는 인공신경망 연산을 위한 전용 라이브러리를 제공하고 다양한 Tensorflow, Caffe 등과 같은 인공신경망 플랫폼들을 가속하기 위해 대표적인 소프트웨어 라이브러리로 cuDNN을 이용한 GPU 가속을 제공한다[11, 12].

3. High Bandwidth Memory(HBM)

현재 범용 프로세스 가속기의 성능이 정체됨에 따라 최적화된 하드웨어 가속기가 주목을 받고 있다. 이는 FPGA를 이용하여 설계하는데, FPGA의 유연성 덕에 기계학습 등 다양한 분야에서 데이터 처리가 원활하게 이루어지고 있다. 이때, 하드웨어 가속기에서 중요한 것은 메모리 간의 병목현상을 줄이는 것과 데이터를 처리하고 저장하는 것이다. 즉, 데이터에 대한 높은 대역폭 접근이 매우 중요하다. 이에 따라 최근 High Bandwidth Memory(HBM)와 같은 데이터 처리 속도를 올리기 위한 기술도 발전하고 있다. HBM은 DRAM을 적층하는 방식으로 구성되며, 기존 DRAM에 비해 향상된 용량, 대역폭 및 전력 효율을 제공한다[13].

4. Various dedicated Hardware Accelerator

이 외에도 하드웨어 가속기 병목현상을 해결하기 위한 다양한 방법들이 존재한다. 그중에서도 앞서 말했듯이 현재 성능 향상률이 뛰어나게 보이지 않는 범용 가속기보다 각각 알고리즘에 맞춰 설계된 맞춤형 가속기가 최근 활발하게 연구되고 있다. 각각 알고리즘에 맞게 설계되었기 때문에 다양한 구조를 가지고 있고, 학계에서도 90개 이상의 다양한 가속기가 제시되었다[7]. 하지만 맞춤형 가속기는 범용 가속기에 비해 지원되는 CNN을 심하게 제한한다는 단점이 존재한다.

맞춤형 가속기 예시로는 작은 용량의 캐시에 동일한 데이터를 여러 번 접근하는 문제를 방지하기 위해 데이터를 재사용하고, 이에 따라 접근 횟수를 감소시켜 병목현상을 최소화한다[14]. 그리고 데이터를 압축하여 저장공간 및 총 메모리 접근 횟수를 감소시키거나[15], 특정 알고리즘에 특화되게 개발하기도 한다[16].

4.1 Data Reuse

데이터 재사용하는 방식은 CNN에서 데이터가 여러 번 반복적으로 사용되는 특징을 이용한 방식이다. CNN은 2차원 Convolution으로 이루어져 있으며 커널과 convoltion 연산을 진행한다. 이 과정에서 입력되는 데이터가 반복적으로 사용되며, 이외에도 다른 커널과도 연산을 진행하면 더 많은 데이터가 반복적으로 사용되게 된다. 이때 GPU를 이용하면, L1 캐시는 용량이 작기 때문에 동일한 데이터에 여러 번 접근하여 지연 시간이 늘어나는 문제가 발생하게 된다. 이러한 점을 방지하기 위해 불러온 데이터를 재사용하는 Row Stationary 기술과 이를 이용한 가속기가 제안되었다[17]. 이는 해당 가속기 내에서 Row Stationary 기술을 이용하여 필요한 데이터를 인접한 곳에 저장해두고 재사용하므로 메모리 접근 횟수를 효율적으로 줄일 수 있다.

4.2 Data Compressed

다음으로는 데이터를 압축하여 저장 공간 및 총 메모리 접근 횟수를 감소시키는 방식이다. 이는 본 연구에서도 사용한 방식이다. GPU의 연산 정밀도는 인공신경망에서 필요로 하는 것 이상으로 정밀하게 진행된다. 하지만, ReLU 연산을 많이 사용하는 인공신경망의 경우 음수를 모두 0으로 바꾸기에 0의 값이 많아지게 된다. 이는 결과적으로 0을 곱하고 더하기에 모두 필요 없는 값이 되고, 쓸데없는 연산량만 늘어나는 결과를 갖게 된다. 이에 따라 필요 없는 연산은 생략하여 연산 성능을 향상시킬 수 있다.

그 외에도 PIM 메모리를 사용하여 데이터 이동을 최소화한 구조[18], 특정 알고리즘에 최적화된 구조[16] 등 많은 하드웨어 가속기 연구가 진행되고 있다[11].

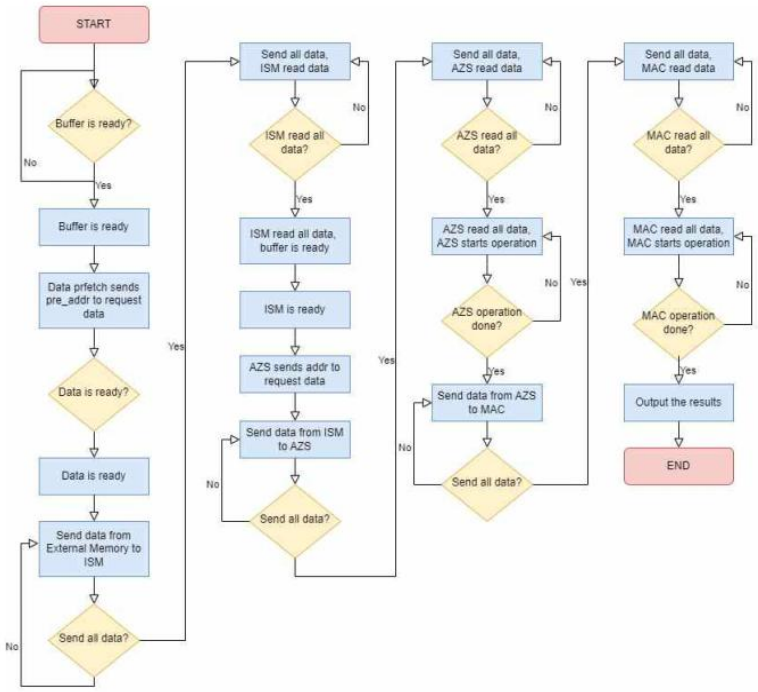

Fig. 1. System Flow Chart

5. Related Work

본 연구는 기존 연구에서 사용한 데이터를 압축하는 방식을 이용하여 저장 공간 및 메모리 접근 횟수를 감소시킨다. 해당 방식은 기존 연구에 따라 ReLU 연산과 pruning 기술로 인하여 많은 데이터 값이 0으로 변환되기 때문에 사용되는 방식이다. 결과에 영향을 미치지 않는 0 값이 많아짐에 따라 메모리 접근 횟수가 급격히 늘어나게 된다. 따라서 본 연구는 0 값을 제거하는 기술인 All-zero skipping 기술을 사용한다[7, 11].

하지만 Data Compressed 방식만으로는 하드웨어 가속기의 성능을 향상시키기 어려워 추가로 데이터 이동 역시 최소화한 Data Prfetch 기술을 사용한다. 이는 많은 양의 데이터가 필요한 딥러닝에서, 메모리 접근에서 발생하는 병목현상을 최소화하기 위한 기술이다. 이는 메모리 내부에 연산 장치를 융합시키는 PIM 구조와 유사하다고 볼 수 있다[19].

그림 1은 본 연구 진행의 전체적인 흐름을 나타낸 흐름도이다. 해당 흐름도는 듀얼 버퍼인지, 데이터가 CIFM인지 CKM인지 구분 없이 전체적인 연구 진행의 흐름을 나타내었다. 데이터를 외부 메모리에서 ISM으로 올리는 과정부터 시작해서 AZS에서 All-zero skipping 연산을 진행하고, 마지막으로 MAC 연산을 완료하여 출력하는 과정까지를 나타내었다.

III. The Proposed Scheme

1. Architecture

1.1 Basic Architecture

그림 2는 본 논문에서 핵심인 데이터 프리패치를 적용하기 전 전반적인 기본 구조를 보여준다. 해당 시스템은 소프트웨어의 CPU 단 즉, Vortex에서 데이터를 받는 부분부터 이를 연결해 주는 AXI Interface가 있다. 하드웨어에는 외부 데이터 메모리를 제어하는 Controller와 외부 데이터 메모리, 하드웨어 단에서 가속을 진행하는 Matrix Multiplication Unit(MMU) 모듈 및 내부에 데이터를 저장하고, 주고받는 Data Control 모듈로 구성되어 있다. 전체적인 흐름은 Vortex를 통해서 CPU에서 데이터를 전송받고, 이를 외부 메모리 Controller를 이용해 외부 메모리에 저장해둔다. 그리고 MMU가 원할 때마다 Data Control 모듈에 데이터를 올려준다. MMU는 받아온 데이터를 이용해서 각 프로세싱 블록에 분배하여 계산하고, 이를 다시 외부로 보내는 동작을 진행한다. 이때 통신은 외부에서 데이터를 주고받을 때는, 128bit AXI Interface를 사용하며, Input Stream Manager(ISM) 내부에서는 32bit AXI Interface를 사용한다. 기존의 구조에서는 하나의 버퍼로 구성되어 있고, 이때 ISM에 데이터가 들어오기 시작하면, MMU의 모든 연산이 종료될 때까지 ISM에 새로운 데이터가 추가로 들어오지 못한다. 이는 여분의 메

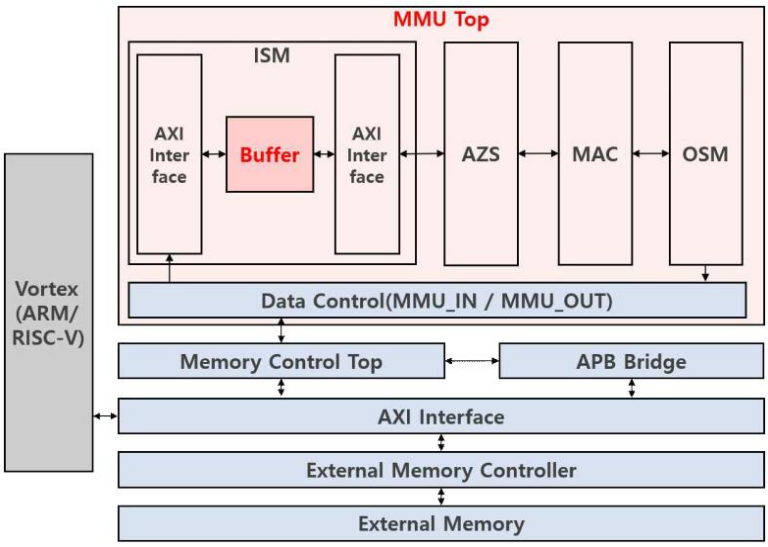

Fig. 2. System Architecture

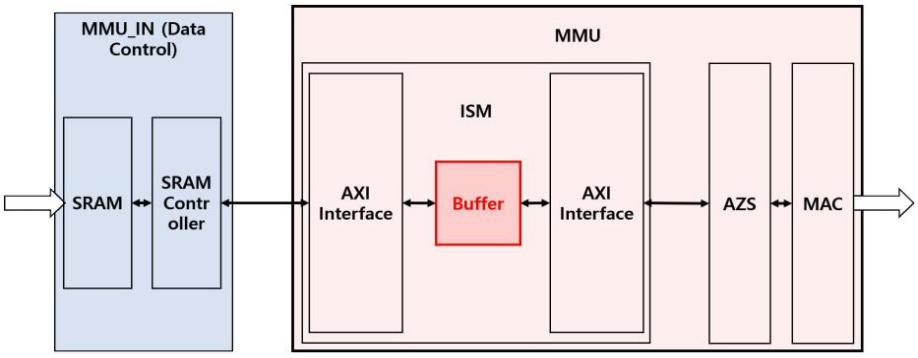

Fig. 3. System Architecture without Prefetch

모리 공간이 없어 새로운 데이터가 기존 데이터에 덮어씌워지는 경우가 생기기도 하며, 해당 경우에 메모리 손실이 발생할 수 있기에 이를 유지한다. 이에 따라 지연시간이 기하급수적으로 늘어나게 된다.

본 연구는 이를 해결하기 위해 해당 구조에서 데이터 프리패치를 추가하고, 듀얼 버퍼로 구성하여 빠르게 데이터를 이동시킴과 동시에 병목현상을 최소화한다. 또한, 가속기는 병렬 프로세싱을 이용한 파이프라인 연산이 가능하며, 온칩 캐시를 통해 메모리의 접근을 빠르게 구성한다. 그리고 Vortex 부분을 제외하고 외부 메모리에 데이터가 있다는 전제하에 연구를 진행한다. 이외의 모든 구조는 기존 연구와 동일하게 진행한다.

1.2 System Architecture without Data Prefetch

Non_Data Prefetch의 MMU_Top 구조는 그림 3과 같다. MMU_Top의 전체 구조는 크게 두 부분으로 나뉜다. 외부와의 통신과 데이터 입출력을 담당하는 Data Control 모듈과 실질적인 연산을 담당하는 MMU 모듈로 구성된다.

Data Control 모듈은 MMU_IN 모듈 안에 SRAM과 SRAM을 제어하는 Controller로 구성되어 있다. MMU_IN 모듈은 MMU의 입력을 제어하는 모듈로 주어진 데이터를 SRAM에 저장하고, Controller의 제어에 따라 이를 MMU 모듈에 공급한다. 이때 32bit AXI Interface를 사용하여 데이터를 주고, 받는다.

MMU 모듈은 ISM 모듈, All Zero-Skipping(AZS) 모듈, Multiply accumulate (MAC) 모듈로 구성된다. ISM 모듈은 입력 데이터를 분배하는 모듈로서 다음 모듈이 원하는 형태로 입력 데이터를 변형하여 공급한다. AZS 모듈은 All zero-skipping[7] 연산이 이루어지는 모듈로서, 필요 없는 데이터를 제거해서 연산량을 줄여준다. MAC 모듈은 MMU의 실제 연산이 이루어지는 모듈이다.

기존에 데이터 프리패치가 존재하지 않고, 버퍼도 1개만 존재하는 경우, 데이터 입출력을 순차적으로 진행한다는 특징이 있었다. 이를 순차적으로 진행하지 않는다면 연산이 제대로 이루어질 수 없기에 순차적으로 오는 데이터를 기다려야 한다. 그리고 이를 다시 기다리면 시간이 무

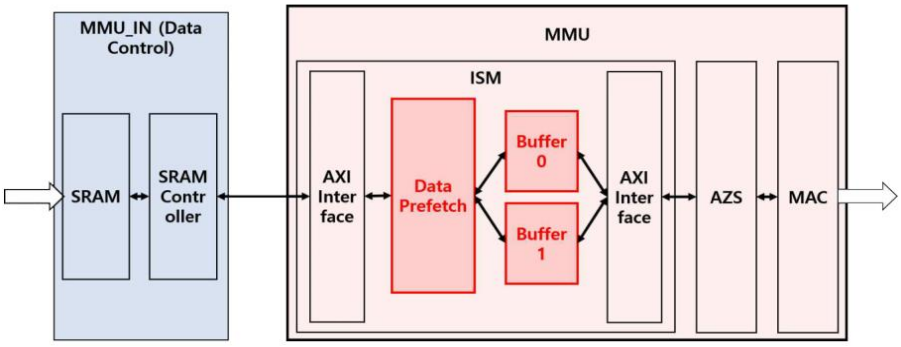

Fig. 4. System Architecture with Data Prefetch

한히 늘어나게 된다. 또한, MMU가 연산에 필요한 데이터를 요청하면 바로 데이터를 가져오고, 한번 데이터를 가져오면 연산이 끝날 때까지 해당 데이터를 유지한다. 이에 따라 지연시간과 연산 시간이 무한히 늘어날 수 있는 특징을 가지고 있다.

따라서 본 논문은 이를 해결하기 위해 데이터 프리패치와 듀얼 버퍼를 추가하였다. 다음은 데이터 프리패치와 듀얼 버퍼에 관한 설명이다.

1.3 System Architecture with Data Prefetch

데이터 프리패치 및 듀얼 버퍼를 추가로 구성한 MMU_Top의 전체적인 구조는 그림 4와 같다.

데이터 프리패치 구조는 싱글 버퍼와 전반적으로 동일한 구조를 가지고 있으며, 버퍼의 개수와 데이터 프리패치 유무만 차이를 가지고 있다. 기존에 데이터를 유지하는 단점을 해결하기 위해 버퍼의 개수를 늘렸으며, 그 외에도 지연시간이 생기는 것을 방지하기 위해 데이터 프리패치를 적용했다.

ISM 모듈을 기존과 다르게 듀얼 버퍼로 구성하여 데이터 입출력을 동시에 가능하게 하였다. 이는 데이터를 계속 유지하지 않아도 데이터 손실이 일어나지 않으며, 다른 동작을 함과 동시에 새로운 데이터를 받을 수 있어 지연시간을 감소시킨다. 또한, 데이터 프리패치를 이용하여 MMU가 연산을 진행하는 동안 다음 연산에 필요한 데이터를 미리 준비한다. 이를 통해 실제로 데이터가 필요하기 전에 외부 메모리에 있는 데이터를 이동 시간이 더 빠른 로컬 메모리로 이동하여 MMU가 쉬지 않고 계산할 수 있도록 한다. 이를 통해 MMU는 연산을 진행함과 동시에 다음 데이터를 받을 수 있고, 자연스럽게 지연시간을 감소한다. Data Prefetch 모듈은 모듈을 따로 동작시켜 연산을 진행함과 동시에 데이터를 미리 가져올 수 있게 하며, Address를 예측하기 쉽게 부여함으로 복잡한 메모리 접근 패턴을 줄이고 데이터가 손실되는 상황을 최소화한다.

본 연구에서는 고정된 단순한 스트라이드 액세스를 통해서 프리패치를 쉽게 예측하고, 미리 데이터를 준비할 수 있다. 이때 스트라이드 액세스는 간단하게 준비된 데이터를 순차적으로 가져오는 방식으로, 프리패치할 때 데이터가 손실되는 경우 없이 정확하게 다음 주소를 가져올 수 있도록 해준다. 이때 간단한 데이터는 3차원 형태의 데이터를 1차원 형태로 만들어, Address를 부여하고, 계산 순서는 Address를 순차적으로 가져오면 되도록 만들어 주었다. 또한, 데이터 프리패치는 MMU 가속기와 분리되어 동작을 독립적으로 진행하며, 이를 통해 지연시간을 최소화할 수 있다.

2. Data Control Module

Data Control 모듈은 소프트웨어와 하드웨어 간의 데이터 전송을 담당하는 부분으로, 여러 채널을 사용할 수 있는 AXI Interface를 사용한다. 본 논문에서는 MMU와 데이터 및 신호 등을 연동하는 Data Control 모듈을 설계 및 구현하였다. MMU_IN은 데이터가 저장되는 SRAM과 SRAM을 각 signal을 통해 제어하는 SRAM Controller로 구성되어 있다.

다음은 MMU_IN에서 MMU로 데이터를 전송하는 동작 프로세스이다. 먼저 MMU_IN에 데이터가 준비되면, MMU로 데이터를 내보낼 상태가 완료되었다고 알린다. MMU 내에 있는 버퍼에 데이터를 받을 준비가 되면, 준비되었다는 신호 MMU로 보낸다. 해당 신호를 받고, MMU_IN은 MMU에 데이터와 non-zero position, Address를 전송한다. 데이터 전송을 완료하면 MMU_IN은 데이터가 모두 전송됐음을 알리고, MMU는 데이터를 다 읽었음을 알린다. MMU가 데이터를 다 읽었다는 신호를 받으면, MMU_IN은 버퍼를 초기화하며, 다음 연산을 준비한다.

이때 해당 과정은 이상적인 환경에서 이루어졌으며, ISM 버퍼가 준비되지 않았거나 MMU_IN에 데이터가 미리 준비되어 있지 않는다면 충분히 더 많은 시간이 소요될 것이다.

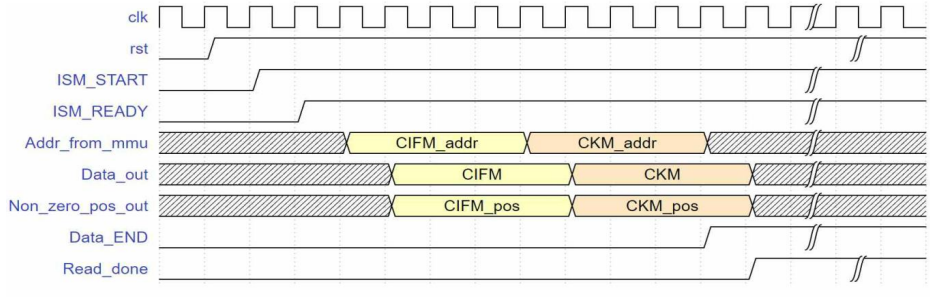

Fig. 5. Data Control Module Timing Diagram

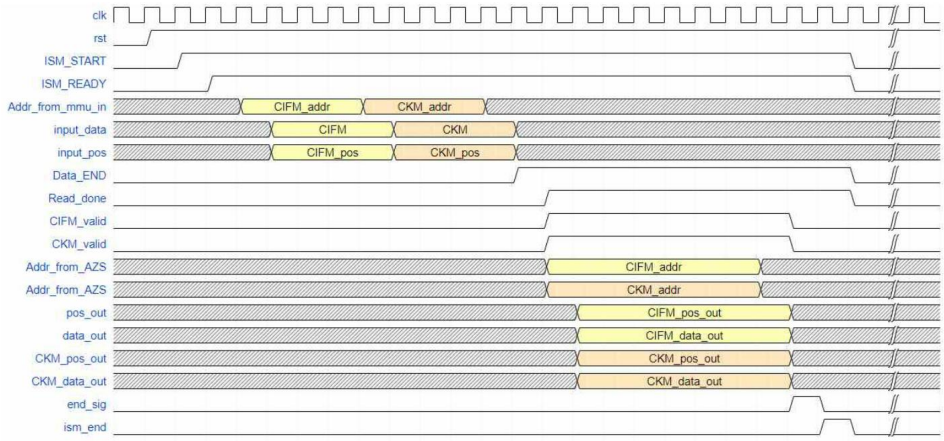

Fig. 6. ISM Module Timing Diagram

그림 5는 위에 설명한 프로세스를 구현한 Data Control 모듈 타이밍 다이어그램이다. 해당 프로세스 전에 MMU_IN에 데이터를 받아서 저장하는 과정을 거친다. 외부 메모리에 데이터가 모두 준비되고, 전송할 준비가 되면 MMU_IN에 이를 알리고, MMU에서 데이터를 받아서 완료한다. 그 후, MMU_IN에 저장된 데이터를 이용하여 위와 같은 동작을 진행한다.

3. MMU Module

MMU 모듈 하드웨어 가속기의 핵심 부분으로서 ISM, AZS, MAC 모듈로 구성되어 있다. 이때 ISM은 각각 데이터 프리패치를 적용한 경우와 아닌 경우로 나누어서 진행되었다. AZS, MAC 모듈은 데이터 프리패치를 적용 유무와 상관없이 모두 같은 알고리즘으로 동작하기 때문에, 추가로 구현하지 않고 동일한 모듈을 사용하였다.

3.1 Basic ISM Module

ISM 모듈은 MMU_IN에서 데이터를 받고, MMU가 연산하기 편하도록 데이터 구조를 변환해 주는 역할을 한다. 이를 통해 MMU가 연산을 보다 빠르고 지연 없이 진행할 수 있다. 기본 ISM의 구조는 그림 3과 같다. ISM 싱글 버퍼는 버퍼가 1개 존재하며, 읽기와 쓰기를 동시에 못 한다는 특징을 가지고 있다.

ISM의 동작 프로세스는 MMU_IN과 AZS와 연결되어 진행된다. 즉, ISM은 AZS가 연산하기 편하도록 데이터 구조를 변환해서 전송한다. 다음은 MMU_IN에서 받은 데이터를, AZS 모듈로 전달하는 동작 프로세스이다. 먼저 MMU_IN을 통해서 받은 데이터를 보낼 준비가 되면, AZS에 데이터를 보낼 상태가 되었다고 알린다. AZS는 ISM에서 보낸 신호를 통해 받을 준비를 완료하고, ISM은 각각 4개의 형태, CIFM_pos_out, CIFM_data_out, CKM_pos_out, CKM_data_out으로 데이터를 내보낸다. 데이터 전송을 완료하면 신호를 전송하여 데이터 전송을 완료했음을 알린다. 또한, AZS는 데이터를 다 읽었음을 ISM에 알린다.

그림 6은 해당 프로세스를 구현한 ISM의 타이밍 다이어그램이다.

3.2 Data Prefetch Module

Data Prefetch 모듈은 듀얼 버퍼를 이용한다. 듀얼 버퍼를 통해 기존에 버퍼가 한 개만 존재했을 때와 같이 연산이 종료될 때까지 데이터를 유지할 필요가 없다. 또한, 읽기와 쓰기를 동시에 진행할 수 있어 기존에 비해 병목

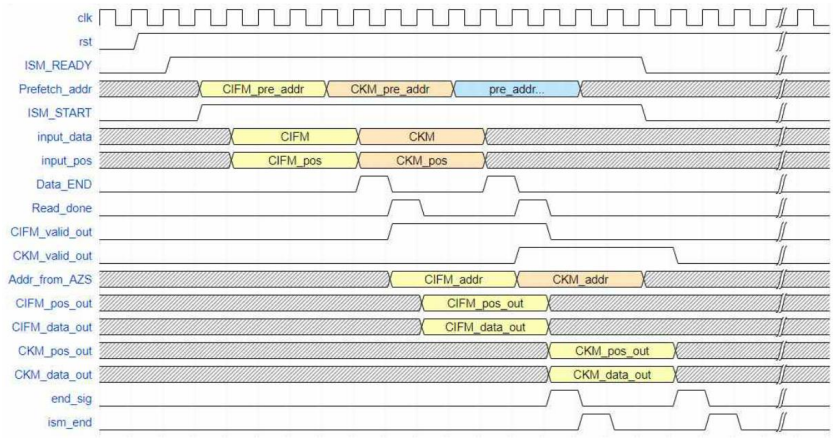

Fig. 7. Data Prefetch Module Timing Diagram

현상이 훨씬 해소된다.

데이터 프리패치는 MMU가 연산을 진행하는 동안 다음 연산에 필요한 데이터를 미리 준비하여 지연 시간을 감소시킨다. 이때 연산에 필요한 데이터는 이미 Vortex로부터 받아서 외부 메모리에 저장되어 있던 데이터를 가져온다. 데이터 프리패치는 정확도와 타이밍이 중요하다. 따라서 우리는 준비해 둔 Address를 스트라이드 액세스함으로 데이터 손실 없이 정확하게 MMU가 원하는 데이터만을 미리 준비한다.

먼저 데이터 프리패치는 MMU가 연산에 사용할 다음 데이터의 Address를 미리 파악하여 데이터를 올릴 준비 한다. 그리고 MMU_IN을 통해서 받은 데이터의 일부가 도착하면 이를 바로 AZS에 데이터를 보낼 상태가 되었음을 알린다. AZS는 받을 준비를 완료하고, ISM은 Data Prefetch를 통해 얻은 Address에 해당하는 데이터를 내보낸다. 동시에 ISM은 MMU_IN으로부터 새로운 데이터를 받고, Data Prefetch 모듈은 다음 연산에 필요한 데이터에 해당하는 Address를 미리 파악하여 데이터를 요청한다. 해당 과정을 모든 데이터 전송이 끝날 때까지 반복한다. 이때 CIFM과 CKM 데이터 중 먼저 준비되는 데이터를 보낼 수 있도록 하며, 데이터 프리패치를 위한 Address는 CIFM과 CKM 각각 따로 파악하여 미리 데이터를 준비한다.

그림 7은 위 프로세스를 진행한 데이터 프리패치의 Timing Diagram이다. 기존 싱글 버퍼를 가진 ISM과 비교했을 때, 데이터를 더 빠르게 AZS 모듈로 전송하는 것을 확인할 수 있다.

3.3 AZS Module

AZS 모듈에서는 ISM에서 보낸 CIFM_pos와 CKM_pos를 비교하여 같은 non-zero position에 해당하는 CIFM_data, CKM_data 값만 다음 모듈로 넘기는 역할을 한다. 이 외의 값은 0 값으로 연산을 진행해도 결괏값이 0이 되므로 연산 시간만 늘어나고, 결국에는 0을 더하는 것이기에 무의미하다. 따라서 All zero-skipping[7]은 입력 데이터를 연산한 결과가 0이 아닌 경우에만 데이터를 다음 모듈로 전달한다.

그림 8은 All zero-skipping의 동작 프로세스이다. 그림에서 알 수 있듯이 non-zero position을 통해 값을 비교하고, 동일한 position을 가진 값만 다음 모듈로 전달한다.

All-Zero Skipping은 그림 9에 있는 알고리즘과 같이 동작 한다. CIFM과 CKM의 non-zero position을 전부 하나하나 비교하여 non-zero posiotion이 같은 값이라면 CIFM, CKM 데이터를 내보낸다. 모든 non-zero position에 대해서 동작을 완료하면 해당 알고리즘을 종료하게 된다. 이 과정에서 약간의 시간이 소요되지만, 무의미한 값을 연산하는 시간보다 짧아 효율적이므로 해당 과정을 진행한다. 다만 0 값이 많이 없는 경우라면 0 값이 많은 데이터에 비해 시간 소요가 더 적다.

3.4 MAC Module

MAC 모듈은 MMU의 연산을 진행하는 모듈이다. AZS를 통해 공급된 한 쌍의 데이터 CIFM_val, CKM_val을 Convolution 연산을 해서 결과를 출력한다.

그림 8에서 MAC 모듈이 동작하는 방식을 알 수 있다. MAC 모듈은 AZS에서 공급된 데이터 단락 별로 Convoltion 연산을 진행하며 값을 더해서 최종 결과물을 출력한다. 연산은 다음 수식(1)과 같다.

$$MAC\_output+ = CIFM\_val * CKM\_val \quad (1)$$

Convolution 연산이지만 현재 데이터가 1차원 형태이므로 커널을 이동해서 연산할 필요 없이 주어진 데이터를 곱해서 더하는 방식을 가진다.

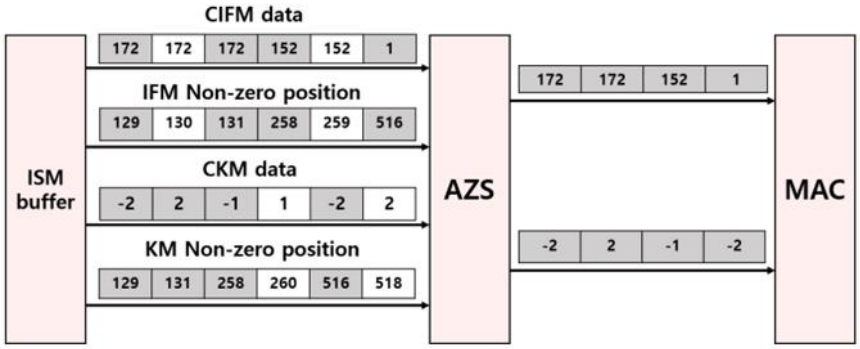

Fig. 8. All Zero-Skipping

Algorithm 1 AZS 모듈 Pseudo-code

```
L1 :  for(x = 0; x < CIFM_NZP ; x++)
L2 :      for(for y = 0; y < CKM_NZP; y++)
L3 :          if(CIFM_NZP == CKM NZF)
L4 :              CIFM_Data_out = CIFM[x]
L5 :              CKM_Data_out = CKM[y];
```

Fig. 9. AZS Module Pseudo-code

4. Data set

최종 개발 결과물은 Register-transfer level(RTL) 시뮬레이션으로 성능을 검증하였다. 이때 환경은 Intel의 QuartusII를 사용하여 개발을 진행하고, Questa를 이용하여 시뮬레이션을 진행하였다. 테스트 데이터는 CIFM과 CKM으로 각각 CIFM_data, CIFM_pos, CKM_data, CKM_pos를 이용하여 두 데이터 간 연산을 수행한 후, 이에 관한 결과를 분석하여 성능 평가를 진행하였다.

CIFM은 다양한 크기, 종류의 이미지에 그림 9와 같이 fine-grain pruning 처리를 진행한 후, 병렬 프로세싱을 위한 블록 샘플링을 진행하였다. 그 후, non-zero value만 추출해서 compressed sequence 생성했다. 이때 데이터의 형태는 스트라이드 액세스가 쉽게 가능하도록 그림 11과 같이 스트라이드 된 형태로 구조를 변경하였다. CKM은 다양한 커널에서 non-zero value만 추출해서 사용하였다. 이때 데이터는 실제로 Convolution 연산이 진행되는 것과 같이 준비하였다.

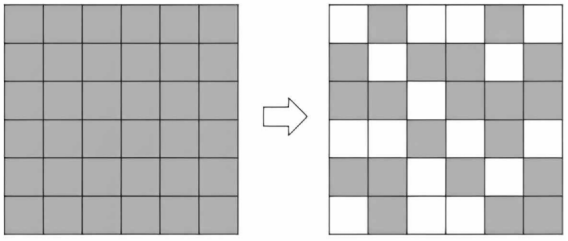

Fig. 10. Fine-grain pruning

그림 10은 fine-grain pruning 처리를 한 예시이다. 예를 들어 왼쪽과 같이 7*7의 이미지가 있다면, fine-grain pruning을 하여 일정한 양의 계수를 제거한다. 이를 이용해 CNN이 너무 필요 이상으로 세분화하여 계산하지 않도록 도와준다. 이는 CNN의 정확도를 낮출 수는 있지만, 너무 많은 시간을 연산하지 않고도 높은 정확도를 얻을 수 있어 많이 사용하는 방식이다. 이때 pruning 하는 방식은 커널 계수를 확인하여 가장 작은 값의 계수를 0으로 하는 방식을 사용했다. 이때 제거된 값은 기존에 있던 0 값과 더불어 평균적으로 52% 정도이다. 단, 이는 기존에 이미지에 있던 0 값도 포함된 값으로 정확하게 pruning 된 값만을 나타냈다고 볼 수는 없다.

그림 11의 데이터 크기는 예시이며, 모든 데이터에 대해 다음과 같이 준비하여 시뮬레이션을 진행하였다. 데이터는 기존의 3차원 형태에서 스트라이드 된 1차원 형태로 바꾸었다. 이때 블록 샘플링은 그림 11에서 색으로 구분된 형태와 같이 나눠서 진행되었다. 이는 각각 곱하는 영역을 같은 블록에 두고, 해당 영역만큼 데이터를 보내, 오차 없이 연산이 진행될 수 있도록 한다. 동시에 이를 통해 병렬적으로 데이터 처리를 진행하여 더 정확하고 빠른 연산을 진행할 수 있다.

이때 데이터의 총 크기를 구하는 방식은 CIFM 픽셀 수에 커널을 압축하여서 곱하는 과정을 진행하여 총 크기를 얻을 수 있었다. 이때 픽셀은 byte 즉, 8bit 단위이다.

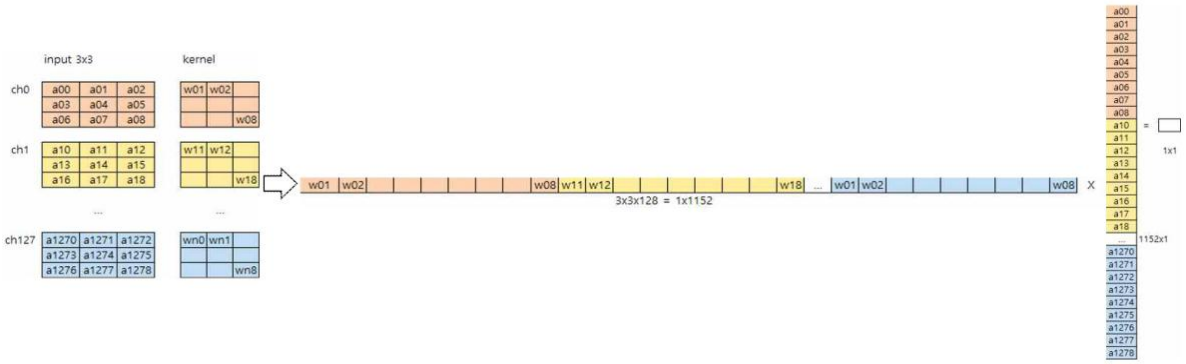

Fig. 11. Stride Data

Table 1. Comparison of Data Prefetch and Non-Data Prefetch results

	Average number of operations	Average processing time	Result	Average accuracy	Performance Improvement
Data Prefetch	14,974,554 byte	1,935,418.75 ns	7.7 byte/ns	100%	-
Non-Data Prefetch	14,974,554 byte	1,556,915.00 ns	9.6 byte/ns	100%	24%

CIFM과 CKM 데이터양은 다음과 수식 (2), (3) 과같이 나타낼 수 있다.

$$CIFM = CIFM\_Width * CIFM\_Height * CIFM\_Depth \quad (2)$$

$$CKM = CKM\ Width * CKM\ Height * CKM\_Depth \quad (3)$$

IV. Performance Analysis

성능은 주어진 데이터의 총 크기를 실제 연산한 시간으로 나누는 방식을 통해 측정하였다. 이는 실제 처리한 양을 시간으로 나누는 개념이다. 이에 따라 데이터의 크기 단위인 byte와 실제 RTL 연산 결과의 시간 단위인 ns를 사용하여 byte/ns로 측정 단위를 정하였다. 이때 시뮬레이션에 사용된 Clock cycle은 125MHz이다.

본 연구에서는 실제로 Data Control 모듈인 MMU_IN의 외부 메모리에 데이터가 있다고 가정하고, MMU로 데이터를 내보내는 것부터 시작하여, MMU에서 데이터를 처리하기까지의 시간을 측정 범위로 잡았다. 이를 통해 확실하게 데이터 프리패치와 듀얼 버퍼의 성능이 얼마나 되는지 측정할 수 있었다. 그리고 이를 실제 처리한 데이터양으로 나누어 결과를 내었다. 데이터양은 압축된 정도에 따라, 얼마나 많이 같은 non-zero pos를 갖느냐에 따라 압축되기 전 크기가 동일하더라도 성능이 다르게 나올 수 있다. 즉, 성능은 이미지 크기에 따라 달라지는 것이 아닌, 얼마나 알고리즘이 확실하게 적용됐는지에 따라 나뉜다. 이때, 시뮬레이션은 버퍼 개수와 데이터 프리패치 외의 모든 조건은 동일한 환경에서 진행하였다.

다음 표1은 Non-Data Prfetch의 RTL 시뮬레이션 결과와 Data Prefetch의 RTL 시뮬레이션 결과를 함께 나타내었다.

평균 연산 수는 Non-Data Prefetch와 Data Prefetch가 모두 동일한 이미지들을 가지고 측정했기에 동일한 값을 갖게 된다. Non-Data Prefetch를 진행한 결과 평균 7.7byte/ns @ 125MHz 성능을 얻을 수 있었고, 이는 평균 연산 수에서 평균 처리시간을 나눈 값이다. 이때 평균 연산 수는 byte 단위로 14,974,554byte이고, 평균 정확도는 100%로 손실 없이 데이터를 모두 정확하게 옮긴 것을 확인할 수 있었다. 또한, 평균 처리시간은 1,935,418.75ns로 이는 MMU_IN에서 MMU로 데이터를 보낸 시간부터 측정하여 MAC이 연산이 완료될 때까지의 시간을 의미한다.

Data Prefetch를 적용한 결과 약 9.6byte/ns @ 125MHz의 성능을 얻을 수 있었고, 이 역시 평균 연산 수

에서 평균 처리시간을 나눈 값이다. 이때 평균 연산 수는 byte 단위로 14,974,554byte로 Non-Data Prefetch와 동일하고, 평균 정확도는 100%로 손실 없이 데이터를 모두 정확하게 옮긴 것을 확인할 수 있었다. 또한, 평균 처리 시간은 1,556,915ns로서 기존의 Non-Data Prefetch보다 뛰어난 성능을 보이는 것을 확인 할 수 있다.

전체적으로 성능을 비교하면 결괏값이 기존에 비해 Data Prefetch를 적용한 결과가 378,503.75ns 정도 평균 처리 시간이 감소하였다. 따라서 해당 결과는 Data Prfetch를 적용한 경우, Non-Data Prefetch보다 약 24% 정도의 성능 향상을 보이는 것을 알 수 있다.

본 논문에서는 병목현상을 효율적으로 줄일 수 있는 하드웨어 가속기를 제안하였다. 하드웨어 가속기의 병목현상이 메모리 간의 이동에서 발생한다는 원인을 찾고, 데이터 프리패치와 듀얼 버퍼를 적용하여 구현하였다. 이는 데이터를 예측하여 필요한 데이터를 외부 메모리에 요청하는 신호를 효율적으로 줄여주며, 읽기와 쓰기 동작을 동시에 진행하며 효율적으로 처리하였다.

이는 데이터의 손실을 막고 정확하게 예측하기 위해 기존의 3차원 형태의 데이터를 스트라이드 처리하고, 데이터 접근 방식으로 스트라이드 액세스 방식을 사용하였으며, 병렬 프로세싱을 이용하여 효율적으로 연산을 진행하여 성능을 개선하였다. 그리고 RTL 시뮬레이션을 이용하여 기존에 비해 성능 향상 24% 이상 진행됨을 검증하였다.

V. Conclusion

본 논문에서 하드웨어 가속기의 병목현상을 효율적으로 줄일 수 있는 하드웨어 가속기의 데이터 프리패치 전략을 제안하였다. 최근 다양한 딥러닝 분야에서 하드웨어 가속기가 많이 사용되고 있으며, 연산 효율성을 높이기 위한 연구가 진행되고 있다. 본 논문에서 제시한 병목현상을 줄이는 것 외에도 데이터를 재사용하는 등 여러 가속기도 제안되고 있다.

게임, 자율주행 자동차와 같은 산업의 발전을 위해서도 저전력으로 연산을 효율적으로 처리할 수 있는 하드웨어 가속기의 발전은 계속될 것으로 예상된다.

본 연구도 추후에 Processing Block(PB)을 MAC 내부에 추가하여 병렬 프로세싱의 효율을 높일 것이다. 이때 PB는 실질적인 MAC 연산이 이루어지는 모듈로서, 이를 통해 MMU가 데이터 프리패치만 적용했을 때보다 더 빠르고 정확하게 데이터를 처리하여 전체적인 성능이 향상될 것으로 예상된다.

ACKNOWLEDGEMENT

This work was supported by “Development of a low-power Mobile GPU Platform(R-20210323-010906) linking deep learning-based image improvement technology” a project for “intensive fostering of innovative artificial intelligence semiconductor companies” funded by the Ministry of Science and ICT (Information and Communication Industry Promotion Agency).

REFERENCES

[1] Y. LeCun, Y. Bengio, G. Hinton, “Deep Learning”, Nature 521 (7553), pp. 436-444, 2015. DOI:10.1038/nature14539

[2] Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J.. “Object detection in 20 years: A survey.”, Proceedings of the IEEE, 111(3),pp.257-276, 2023. DOI: 10.1109/JPROC.2023.3238524

[3] Zhao, Zhong-Qiu, et al. "Object detection with deep learning: A review." IEEE transactions on neural networks and learning systems 30.11, pp.3212-3232, 2019. DOI:10.1109/TNNLS.2018.2876865

[4] C. Chen, A. Seff, A. Kornhauser and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, pp. 2722-2730, 2015. DOI: 10.1109/ICCV.2015.312. DOI:10.1109/ICCV.2015.312

[5] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587, pp.484-489, 2016. DOI:10.1038/nature16961

[6] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." Communications of the ACM 60.6, pp.84-90, 2017. DOI:10.1145/3065386

[7] Struharik, Rastislav JR, et al. "CoNNa–Hardware accelerator for compressed convolutional neural networks" Microprocessors and Microsystems 73, 2020. DOI:10.1016/j.micpro.2020.102991

[8] Chen, Tao, and G. Edward Suh. "Efficient data supply for hardware accelerators with prefetching and access/execute decoupling." 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2016. DOI:10.1109/MICRO.2016.7783749

[9] Chen, Yunji, et al. "DianNao family: energy-efficient hardware accelerators for machine learning." Communications of the ACM 59.11, pp.105-112, 2016. DOI:10.1145/2996864

[10] Jouppi, Norman P., et al. "In-datacenter performance analysis of

a tensor processing unit." Proceedings of the 44th annual international symposium on computer architecture. 2017. DOI:10.1145/3079856.3080246

[11] Park, Jong-Hyeon, et al. "Hardware Accelerator for Artificial Neural Network Operations Latest Research Trends." Communications of the Korean Institute of Information Scientists and Engineers 34.9, pp.21-26, 2016.

[12] Oyama, Yosuke, et al. "Accelerating deep learning frameworks with micro-batches." 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2018. DOI:10.1109/CLUSTER.2018.00058

[13] JUN, Hong shin, et al. Hbm (high bandwidth memory) dram technology and architecture. In: 2017 IEEE International Memory Workshop (IMW). IEEE, pp. 1-4, 2017. DOI: 10.1109/IMW.2017.7939084

[14] Marchisio, Alberto, Muhammad Abdullah Hanif, and Muhammad Shafique. "Capsacc: An efficient hardware accelerator for capsulenets with data reuse." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019. DOI:10.23919/DATE.2019.8714922

[15] Han, Song, et al. "EIE: Efficient inference engine on compressed deep neural network." ACM SIGARCH Computer Architecture News 44.3, pp.243-254, 2016. DOI:10.1145/3007787.3001163

[16] Hyo-chan Lee, Hyun-Hak Song, Jing-Lin Lwo, Fei Wang , Hak-Lim Ko, Tae-Ho Im. “The Design of Hardware Accelerator for Binary Segmentation Algorithm for Object Detection in Maritime.” Journal of the Korea Institute Of Information and Communication Engineering, 24(1), pp. 157-160, 2020. DOI:10.6109/jkiice.2020.24.10.1331

[17] Chen, Y. H., Yang, T. J., Emer, J., & Sze, V. "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9.2, pp.292-308, 2019. DOI:10.1109/JETCAS.2019.2910232

[18] Chi, Ping, et al. "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory." ACM SIGARCH Computer Architecture News 44.3, pp. 27-39, 2016. DOI:10.1145/3007787.3001140

Authors

Hyein Lee received the B.S. degree in human-centered artificial intelligence and the M.S. degree in artificial intelligence and informatics from Sangmyung University, Seoul, South Korea, in 2021 and 2024, respectively. Hyein Lee is interested in HW Accelerator, Artificial Intelligence, parallel computing, FPGA, and Processor Design.

Jinoo Joung received the B.S. degree in electronics engineering from the Korea Advanced Institute for Science and Technology, Daejeon, Korea, in 1992, and the M.S. and Ph.D. degrees in electrical and electronics engineering from New York University, New York City, NY, USA, in 1994 and 1997, respectively. From 1997 to 2005, he was with Samsung Electronics and the Samsung Advanced Institute for Technology, where he was involved in various network SOC research and developments, especially for general packet radio service for 3G mobile systems, network processors for high-speed IP routers, and mobile application processors for smart handheld devices. In 2005, he joined Sangmyung University, Seoul, South Korea. His research interests include network SOCs, high-speed network processing, switch architecture, network calculus, network QoS control, and autonomous wireless network scheduling/routing. He is active in international standardization activities. He is the main Editor of various ITU-T recommendations, including Y.2122, Y.3113, Y.3118, and Y.3120.