공학석사학위논문

오프 칩 메모리 접근이 NPU 시스템의 성능에 끼치는 영향에 대한 분석

Analysis of the Effect of Off-chip Memory Access on the Performance of an NPU System

2022년 2월

서울대학교 대학원
컴퓨터공학부
이 건 주

오프 칩 메모리 접근이 NPU 시스템의 성능에 끼치는 영향에 대한 분석

Analysis of the Effect of Off-chip Memory Access on the Performance of an NPU System

지도교수 하 순 회

이 논문을 공학석사 학위논문으로 제출함

2021년 11월

서울대학교 대학원

컴퓨터공학부

이 건 주

이건주의 공학석사 학위논문을 인준함

2022년 1월

위 원 장	이 창 건	(인)
부위원장	하 순 회	(인)
위 원	유 승 주	(인)

요 약

NPU는 계산집약적인 합성 곱 신경망 (Convolutional Neural Network, CNN) 연산을 효율적으로 가속하기 위해 최적화되어 왔고 다양한 연구와 함께 빠르게 발전하고 있다. NPU의 성능에 큰 영향을 주는 요인 중 하나는 오프 칩 DRAM 접근 지연 시간으로 이를 줄이기 위해 일반적으로 사용하는 방법은 온 칩 내부에 큰 사이즈의 SRAM을 두고 온 칩 안에서의 데이터 재사용을 최대화하는 것이다. 하지만 온 칩 데이터 최적화를 통해 성능을 높이는 많은 NPU 디자인 연구들이 진행되고 있는 반면에 오프 칩 DRAM 접근 지연 시간 자체에 대한 분석은 상대적으로 적다.

본 논문에서는 오프 칩 메모리 접근 지연 시간이 NPU 성능에 미치는 영향에 대해 시뮬레이션 결과을 통한 자세한 분석과 함께 살펴 본다. 실험을 위해 사이클 단위의 정확한 (Cycle-accurate) 시뮬레이션 환경을 구축하였고 NPU 시뮬레이터로는 MIDAP, DRAM 시뮬레이터로는 Ramulator, 버스 시뮬레이터로는 AMBA AXI4 TLM 2.0 시뮬레이터, 그리고 CACTI 툴 기반으로 구현한 커스텀 SRAM 시뮬레이터를 사용하였다.

다양한 CNN 모델로 실험을 진행하여 오프 칩 DRAM 접근 시간이 NPU 성능에서 매우 큰 비중을 가지고 있음을 확인하였고 NPU와 오프 칩 DRAM 사이에 위치한 큰 사이즈의 오프 칩 SRAM이 NPU 성능에 주는 영향에 대해 살펴보았다. 오프 칩 SRAM으로는 Cache와 스크래치패드 메모리 (Scratchpad Memory, SPM)을 사용하였고 SPM이 Cache보다 NPU 시스템에 더 적합한 SRAM임을 확인하였다. ResNet50은 SPM을 통해 오프 칩 메모리 접근 지연 시간이 약 70.7% 감소하였고 전체 시뮬레이션 사이클은 약 26.5% 감소하였다.

마지막으로 단일 NPU 시스템 대비 멀티 NPU 시스템의 오프 칩 메모리 접근에 대한 영향을 확인할 수 있도록 기존의 시뮬레이션 환경을 확장하였다. 여러 개의 NPU와 SPM 뱅크들을 연결할 수 있는 버스 구조를 통해 다양한 조건의 시스템 설정으로 설계 영역 탐색 (Design Space Exploration, DSE)이 가능하도록 하였다.

주요어 : NPU 성능, DRAM 접근 지연 시간, 스크래치패드 메모리, 멀티 NPU 시스템

학번 : 2020-23938

Contents

List of Figures

List of Tables

Chapter 1

서 론

1.1 연구의 배경

여러 응용에서의 합성 곱 신경망 (Convolutional Neural Network, CNN) 발전과 함께 합성 곱 연산을 가속하기 위한 장치인 NPU가 다양하게 소개되고 있다. NPU는 CNN에서 가장 많은 시간이 소요되는 합성 곱 연산을 병렬 수행하기 위한 다량의 MAC (Multiplier Accumulator) 유닛을 가지고 있다. NPU의 최대 성능은 MAC 사용률 (Utilization)이 100%가 되었을 때의 MAC 유닛 수와 클락 주파수 (Frequency) 값의 곱으로 나타낼 수 있다. 하지만 많은 연구에 의하면 실제 달성 가능한 성능은 하드웨어를 통해 얻을 수 있는 기대 성능보다 낮게 나타나는 것으로 알려져 있다 [2]. 합성 곱 연산을 사용하지 않는 레이어, 자원 경합, 그리고 메모리 접근 지연 등 NPU 성능에 영향을 주는 다양한 요인들이 존재하고 그 중 오프 칩 DRAM 접근 지연 시간은 성능에 가장 큰 영향을 주는 요인 중 하나로 나타났다.

오프 칩 DRAM 접근 지연 시간을 줄이기 위해 NPU는 온 칩에 큰 사이즈의 SRAM을 두고 DRAM으로부터 로드된 데이터의 재사용이 최대화될 수 있도록 최적화되어 설계된다. 데이터를 재사용 하는 방법에 따라 NPU의 데이터 흐름 구조 (Data Flow Architecture)가 결정되는데, 예를 들면, Eyeriss [3]는 특성 지도 (Feqture Map)와 가중치 행 (Weight Row) 의 재사용을 최대화하는 행 고정 (Row Stationary, RS) 방식의 데이터 흐름 구조를 사용한다. 그리고 합성 곱 신경망의 제로 데이터 통계를 활용하여 오프 칩 DRAM 접근을 위한 데이터 크기를 줄이는 압축 기술도 사용하였다. 그외에

데이터 정밀도를 낮추어 DRAM 접근 크기를 줄이는 것도 DRAM 접근 지연 시간을 줄이기 위한 방법이 될 수 있다.

NPU 성능에 큰 영향을 주는 DRAM 접근과 에너지 소모를 줄이기 위해 온 칩 SRAM 안에서의 데이터 재사용을 최대화하는 연구는 광범위하게 진행되고 있는 반면에 상대적으로 오프 칩 DRAM 접근이 NPU 성능에 미치는 영향에 대한 분석은 충분히 이루어지지 않고 있다. 시스톨릭 (Systolic) 합성 곱 신경망 가속기 시뮬레이터인 SCALESIM [4]은 시뮬레이션에 의해 계산된 각 레이어의 연산 시간 (Cycle)과 데이터 크기를 통해 합성 곱 신경망에서 요구하는 DRAM 대역폭을 구한다. 시스템에 필요한 DRAM 대역폭이 만족된다면 DRAM 지연 시간이 프리페치 (Prefetch)나 이중 완충법 (Double Buffering)에 의해 가려져서 NPU 성능에 영향을 주지 않는다고 가정하고 있다. 하지만 오프 칩 DRAM으로부터의 데이터를 기다리는 동안 NPU가 절대 멈추지 않는다는 이상적인 가정은 임베디드나 모바일과 같이 제약된 환경에서 충족되기는 어렵기 때문에 시뮬레이션 결과는 실제 오프 칩 DRAM 접근 지연 시간의 영향을 NPU 성능에 충분히 반영하지 못 하였다.

1.2 연구의 내용

본 논문에서는 오프 칩 DRAM 접근이 NPU 성능에 미치는 영향을 사이클 단위의 정확한 시뮬레이션 결과와 함께 살펴 본다. 또한, 다양한 실험을 통해 오프 칩 SRAM과 멀티 NPU 시스템이 NPU 성능에 주는 영향을 분석해본다. 논문의 내용은 크게 네 부분으로 나눌 수 있다.

- 오프 칩 DRAM 접근 지연 시간을 정확하게 반영하여 NPU 성능을 측정하기 위해 사이클 단위의 정확한 시뮬레이션 환경을 구축하였다. 신뢰성 있는 데이터를 위해 이미 검증된 오픈 소스 시뮬레이터를 이용하였다. NPU 시뮬레이터로 MIDAP [5], DRAM 시뮬레이터로는 Ramulator [6], 그리고 버스 시뮬레이터로는 AMBA AXI4 TLM2.0 시뮬레이터 [1]를 사용하였다. 또한, 오프 칩 SRAM 시뮬레이터는 여러 조합의 시뮬레이션을 가능하게 하기 위해 CACTI [7]로부터 생성된 SRAM

모델을 기반으로 직접 구현하여 추가 구성하였다.

- 오프 칩 DRAM 접근 지연 시간의 예측 방법을 3 가지 DRAM 접근 모델로 분류하고 각 모델의 특성을 분석하여 오프 칩 DRAM 접근 지연 시간이 NPU 성능에 미치는 영향을 자세히 살펴보았다. DRAM 접근 모델에는 DRAM 대역폭 기반으로 지연 시간을 예측하는 대역폭 모델 (Bandwidth Model), DRAM 접근에 영향을 주는 DRAM과 버스 등의 다양한 요소들을 수식화하여 지연 시간을 계산하는 분석 모델 (Analysis Model), 그리고 DRAM 접근에 영향을 주는 DRAM과 버스를 직접 시뮬레이션하여 지연 시간을 예측하는 시뮬레이션 모델 (Simulation Model)이 있다. 상대적으로 정확하지 않은 결과를 보여주는 대역폭 모델과 분석 모델보다 시뮬레이션 모델이 오프 DRAM 접근 지연 시간 예측 및 NPU 성능 측정에 필요함을 확인하였다. 그리고 온 칩 SRAM에서의 데이터 재사용을 통해 DRAM 접근 지연 시간을 줄이는 다양한 연구가 있음에도 불구하고 CNN 모델의 특성과 크기에 따라 불가피하게 발생하는 DRAM 접근으로 인해 NPU 성능이 여전히 크게 하락함을 알 수 있었다.

- NPU와 오프 칩 DRAM 사이에 위치한 오프 칩 SRAM이 NPU 성능에 미치는 영향에 대해 다양한 실험의 결과를 가지고 분석을 진행하였다. 일반적으로 오프 칩 DRAM 접근 지연 시간을 줄이기 위해 NPU는 온 칩에 큰 사이즈의 SRAM을 두고 로드된 데이터가 그 안에서 최대한 재사용될 수 있도록 최적화하여 설계된다. 하지만 임베디드 및 모바일 환경이 가지는 성능, 파워 소모, 칩 면적 등의 여러 가지 제약 사항은 각 CNN 모델이 사용하는 데이터 크기만큼 충분한 크기의 온 칩 SRAM을 사용하기 어렵게 만든다. 이러한 제약 안에서 오프 칩 SRAM은 오프 칩 DRAM 접근을 줄이고 오프 칩 SRAM 내에서의 데이터 재사용을 통해 NPU 성능을 향상시키는데 큰 도움을 준다. 본 논문에서는 오프 칩 SRAM으로써 Cache와 스크래치패드 메모리 (Scratchpad Memory, SPM)를 사용하여 다양한 실험을 진행하였고 SPM이 Cache보다 NPU 시스템에 훨씬 효율적임을 확인할 수 있었다.

- 다수의 NPU가 공존하며 오프 칩 메모리를 공유하는 시스템에서 NPU 성능을 측정하기 위해 단일 NPU로 구성하였던 기존의 시뮬레이터를 여러 NPU가 SPM을 공유하여 사용할 수 있는 멀티 NPU 시스템 시뮬레이터로 확장하였다. CNN의 발전과 함께 앞으로 다양한 응용 프로그램의 사용이 보편화되고 하나의 시스템에서 여러 CNN 응용 프로그램들을 에너지-시간 효율적으로 가속하기 위한 방법이 요구될 것으로 예상된다. 이를 위한 멀티 NPU 시스템 시뮬레이터를 구성하여 다양한 시나리오의 시뮬레이션이 가능하도록 하였다.

1.3 논문의 구성

본 논문은 다음과 같은 내용으로 구성되어 있다. 2장에서는 NPU 성능의 정확한 측정을 위해 만든 시스템 시뮬레이터의 구성에 대해 설명한다. 시스템 시뮬레이터는 NPU 시뮬레이터, DRAM 시뮬레이터, 버스 시뮬레이터, 그리고 오프 칩 SRAM 시뮬레이터로 구성되어 있으며, 각 시뮬레이터에 대해 순차적으로 다룬다. 3장에서는 오프 칩 DRAM 접근 지연 시간을 예측하는 3가지 DRAM 접근 모델인 대역폭 모델, 분석 모델, 그리고 시뮬레이션 모델을 설명하고 NPU 성능에 오프 칩 접근 지연 시간이 미치는 영향에 대해 분석한 결과를 보여준다. 4장에서는 NPU와 DRAM 사이에 위치한 큰 사이즈의 오프 칩 SRAM이 NPU 성능에 주는 영향에 대해 살펴 본다. 오픈 칩 SRAM으로써 Cache와 SPM이 사용되었으며, 다양한 CNN 모델을 통해 두 SRAM의 특성을 비교 분석하였다. 5장에서는 다수의 NPU가 오프 칩 메모리를 공유하여 사용하는 환경에서 NPU 성능을 측정할 수 있는 멀티 NPU 시스템에 대해 소개하고 추후 필요한 연구에 대해 설명한다. 6장에서의 결론을 마지막으로 본 논문은 마무리된다.

Chapter 2

시스템 시뮬레이터 환경

시스템 시뮬레이터는 그림 2.1과 같이 NPU 시뮬레이터, DRAM 시뮬레이터, 그리고 버스 시뮬레이터가 통합되어 하나의 시뮬레이터로써 동작하도록 구성되어 있다. 시스템을 구성하는 각 시뮬레이터는 오픈 소스로 공개되어 많은 연구에서 이미 검증된 시뮬레이터로 데이터의 신뢰성을 위해 선택되었다. NPU 시뮬레이터로는 MIDAP [5], DRAM 시뮬레이터로는 Ramulator [6], 버스 시뮬레이터로는 AMBA AXI4 TLM2.0 시뮬레이터 [1]가 사용되었다. SRAM 시뮬레이터는 다양한 매개 변수 변경을 통해 설계

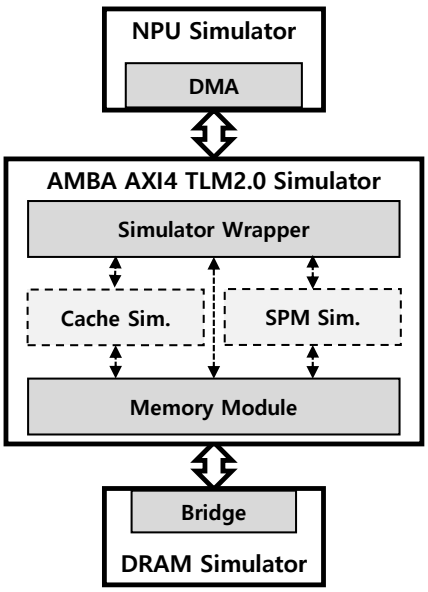

그림 2.1: 시스템 시뮬레이터 디자인

영역 탐색 (Design Space Exploration, DSE)을 할 수 있도록 CACTI [7] 툴로 생성한 SRAM 모델 기반으로 직접 구현하였다. Cache 시뮬레이터와 SPM 시뮬레이터가 있으며 시뮬레이터 옵션에 따라 선택적으로 채택되어 사용된다. NPU, 버스, 오프 칩 SRAM 모두 1 Ghz의 클럭 속도로 동작하고 DRAM은 LPDDR4의 스펙에 맞게 1.6 Ghz의 클럭 속도로 동작한다.

2.1 MIDAP

MIDAP [5]은 합성 곱 신경망 (CNN)을 구성하는 각 레이어의 파이프라이닝 (Pipelining) 실행을 통해 NPU가 가진 MAC 유닛의 사용률을 최대화할 수 있도록 디자인된 CNN 가속기 시뮬레이터다. 두 종류의 파이프라이닝을 지원하며, 합성 곱 (Convolution) 레이어, 활성화 (Activation) 레이어, 그리고 풀링 (Pooling) 레이어를 위한 모듈들로 구성된 매크로 파이프라이닝 (Macro Pipelining)과 합성 곱 레이어 안에서 이루어지는 마이크로 파이프라이닝 (Micro Pipelining)으로 나누어진다. MIDAP 내부에는 큰 사이즈의 SRAM이 있으며, 그 중 특성 지도 (Feature Map)를 위한 1MB SRAM과 필터 가중치 (Filter Weight)을 위한 512KB SRAM이 가장 큰 비중을 차지하고 있다. MIDAP은 온 칩 SRAM과 오프 칩 DRAM 사이의 데이터 이동을 위한 DMA (Direct Memory Access)을 가지고 있다. MIDAP에서 발생하는 모든 오프 칩 DRAM 접근은 DMA를 통해 트레이스 (Trace)의 형태로 공유 메모리에 저장되어 버스 시뮬레이터로 전달된다. DMA가 생성한 트레이스는 Read / Write 구분을 위한 타입과 요청 데이터의 크기뿐만 아니라 MIDAP 코어의 사이클 정보도 포함하고 있다. 이를 통해 서로 다른 프로세스에서 병렬로 동작하는 NPU 시뮬레이터와 버스 시뮬레이터가 시간 동기화를 맞출 수 있게 된다.

2.2 Ramulator

Ramulator [6]는 사이클 단위의 정확한 DRAM 시뮬레이터로 DRAM 성능 측정을 필요로 하는 많은 연구에서 널리 사용되고 있다. DDR3, DDR4, LPDDR4 등 다양한 종

류의 DRAM 표준을 지원하고 있는데, 본 논문에서는 NPU 시뮬레이터인 MIDAP [5]에서 레퍼런스로 사용하고 있는 2 채널의 LPDDR4를 선택하였다. Ramulator는 트레이스 기반으로 (Trace-driven) 동작하기 때문에 시스템의 다른 시뮬레이터들과의 시간 동기화를 위해 브릿지 (Bridge) 모듈이 Ramulator 내부에 추가 구현되었다. Ramulator는 외부에서 접근 가능한 API를 통해 트레이스 전달 받거나 시뮬레이션 결과를 전송하면서 사이클 단위의 시뮬레이션을 진행한다. 버스 시뮬레이터로부터 전달 받은 트레이스는 브릿지 모듈의 커맨드 큐에 저장되고 Ramulator의 Read 큐와 Write 큐로 나뉘어져 하나씩 순차적으로 처리된다. 버스 시뮬레이터는 매 사이클마다 Ramulator의 외부 API를 통해 완료된 트레이스를 확인하고 NPU 시뮬레이터에 응답을 하게 된다.

2.3 AMBA AXI4 TLM2.0 시뮬레이터

AMBA AXI4 TLM 2.0 시뮬레이터 [1]는 반도체 회사인 ARM에서 오픈 소스로 제공하는 버스 시뮬레이터로 오픈 소스 C++ 라이브러리인 TLM (Transaction Level Model) 2.0으로 작성되었다. 트랜잭션 (Transaction) 레벨의 시간 근사 (Approximately-timed, AT) 모델과 사이클 단위의 정확한 (Cycle-accurate, CA) 모델을 제공하며, 본 논문에서는 CA 모델이 사용되었다. AMBA AXI4는 양방향 핸드쉐이크 (2-way handshake) 메카니즘을 기반으로 독립적인 Read와 Write 채널을 가진다. 트랜잭션 당 최대 4,096 바이트의 데이터가 처리 가능하지만 실험에서 사용하는 NPU와 DRAM 스펙을 고려하여 1,024 바이트 크기의 트랜잭션이 가능하도록 설정하였다.

AMBA AXI4 TLM 2.0 시뮬레이터 (이하 AXI4 시뮬레이터)는 메모리 트래픽을 생성하는 마스터 (Master) 모듈과 메모리 역할의 슬레이브 (Slave) 모듈로 구성되어 있다. 별개의 시뮬레이터로써 동작하는 NPU 시뮬레이터 및 DRAM 시뮬레이터와의 인터페이스를 위해 각 모듈의 역할을 확장하였다. 마스터 모듈은 NPU 시뮬레이터와 연결되는 시뮬레이터 래퍼 (Simulator Wrapper)로써 동작한다. 시뮬레이터 래퍼는 파이썬 (Python)으로 작성된 MIDAP 시뮬레이터의 DRAM 접근 요청을 공유 메모리 (Shared Memory)를 통해 트레이스 (Trace)의 형태로 입력 받아 TLM 2.0으로 작성된 AXI4 시뮬레이터에서 사용하는 페이로드 (Payload)로 전환하는 역할을 한다. 서로 다

른 프로세스에서 동작하는 두 시뮬레이터의 시간 동기화는 트레이스에 포함된 NPU 시뮬레이터의 시간 정보를 통해 이루어진다. 슬레이브 모듈은 마스터 모듈로부터 전달 받은 페이로드를 패킷 (Packet) 단위인 64 바이트로 나누어 DRAM 시뮬레이터인 Ramulator에 전달한다.

2.4 커스텀 SRAM 시뮬레이터

본 논문에서 구축한 NPU 시스템에서는 오프 칩 SRAM으로 Cache와 SPM을 지원한다. 실험을 위한 시뮬레이터로는 시스템 설정을 통해 다양한 설계 영역 탐색 (Design Space Exploration, DSE)을 가능하게 하기 위해 직접 구현한 커스텀 SRAM 시뮬레이터를 이용하였다. CACTI [7] 툴에서 생성한 SRAM 모델을 기반으로 사이클 단위의 정확한 (Cycle Accurate) 시뮬레이션을 할 수 있다. CACTI는 메모리 분석 모델을 제공하는 오픈 소스 툴로 다양한 연구에서 사용되고 있으며, 접근 시간 (Cycle), 에너지, 면적 등 다양한 정보를 보여준다. 두 개의 커스텀 SRAM 시뮬레이터는 그림 2.1에서 보여지는 것처럼 AXI4 시뮬레이터의 인하우스 (In-house) 시뮬레이터로써 동작하게 되며, 시스템 설정에 따라 선택적으로 사용된다.

Cache 시뮬레이터는 매개 변수 (Parameter) 설정에 따라 크기와 구성을 달리 할 수 있도록 구현되었다. 블록 크기, 연관 (Associativity) 크기 등 다양한 설정을 통해 Cache 구조 변경이 가능하다. 블록 교체 정책으로는 랜덤 (Random)과 LRU (Least Recently Used)가 지원되며, 본 논문에서는 CNN 모델에서 더 좋은 성능을 보여준 LRU를 사용하였다. AXI4 시뮬레이터의 마스터 모듈로부터 전달 받은 페이로드는 Ramulator와 동일한 64 바이트 단위의 패킷으로 나뉘어져 Cache 시뮬레이터에 전달된다. 전달된 패킷은 Cache 정책에 따라 DRAM 데이터 접근 요청을 발생시키며 트레이스의 형태로 Ramulator에 전달된다.

SPM 시뮬레이터는 컨트롤 코어 (Control Core)와 여러 개의 뱅크 (Bank)로 구성되는 간단한 구조를 가지고 있다. 컨트롤 코어는 NPU 컴파일러로부터 생성된 SPM 트레이스 (Trace) 기반으로 동작하며, 트레이스에 저장된 커맨드를 순차적으로 처리하는 역할을 한다. 트레이스에서 사용하는 커맨드의 종류에는 총 네 가지가 있으며, DRAM

과 SPM 사이의 데이터 이동을 위한 PREFETCH와 WRITEBACK 커맨드, NPU와 SPM 사이의 데이터 동기화를 위한 SIGNAL과 WAIT 커맨드로 구성되어 있다. SPM의 각 뱅크는 사이클 당 64 바이트의 데이터 처리가 가능하며 뱅크의 수는 시스템 옵션을 통해 실험에 따라 다르게 설정할 수 있다.

2.5 오프 칩 메모리 접근 세부 동작

본 섹션에서는 NPU의 오프 칩 메모리 접근이 어떠한 순서로 진행되는지 각 구성 요소 별로 상세하게 살펴 본다. 오프 칩 메모리 접근은 크게 NPU 내부의 DMA, 버스, 그리고 DRAM로 구분되어 진행된다.

2.5.1 NPU

실험에 사용하고 있는 NPU 시뮬레이터인 MIDAP은 DMA를 시뮬레이션하기 위한 로직을 내부에 포함하고 있다. DMA는 커맨드 큐 (Command Queue)을 사용하여 MIDAP의 오프 칩 메모리 데이터 요청을 관리하는 역할을 한다. 커맨드 큐에는 3 종류가 있으며, 1개의 Write 큐와 서로 다른 우선 순위를 가진 2개의 Read 큐로 구성된다. DMA는 내부 스케줄링 방식에 따라 각각의 큐를 순차적으로 확인하고 오프 칩 메모리 데이터 요청이 있을 경우 트레이스 (Trace)를 생성하여 버스 시뮬레이터로 전달한다.

트레이스의 단위는 실험에 사용하는 AXI4의 스펙에 근거하여 정해진다. AXI4는 최대 128 바이트의 데이터 버스 폭 (Data Width)과 최대 256개의 버스트 전송을 4KB

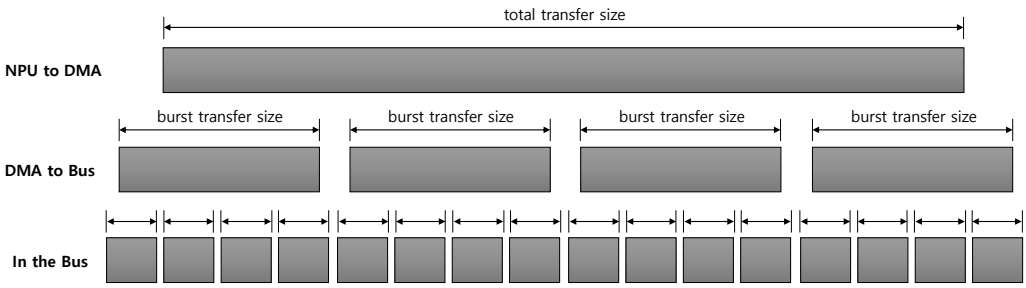

그림 2.2: 시뮬레이션 단계에 따른 트랜잭션 단위의 변화

주소 경계 (Address Boundary)의 트랜잭션 (Transaction) 단위로 지원하고 있으며, 본 논문의 실험에서는 1,024 바이트 단위의 트랜잭션을 사용하였다. NPU의 오프 칩 메모리 데이터 요청은 적게는 1 KB 이내, 크게는 1 MB를 넘어가는 단위로 발생할 수 있다. 따라서, 트랜잭션 단위를 초과하는 데이터 요청은 그림 2.2와 같이 트랜잭션 단위로 나뉘어져 DMA의 커맨드 큐에 저장이 되고 스케쥴링에 따라 트레이스로 변환되어 버스로 전달된다.

NPU 시뮬레이터와 버스 시뮬레이터의 트레이스 전달은 공유 메모리를 통해 이루어진다. 공유 메모리는 4096개의 트레이스를 저장할 수 있는 2개의 순환 큐 (Circular Queue)로 구성되어 있다. NPU 시뮬레이터가 버스 시뮬레이터에 전달하는 트레이스는 요청 큐 (Request Queue)에 저장이 되고, 버스 시뮬레이터는 오프 칩 메모리 데이터를 응답 큐 (Response Queue)를 통해 NPU 시뮬레이터에 전달한다. 데이터 응답을 필요로 하지 않는 Write 요청은 요청 큐를 통해 전달만 되고 응답은 받지 않는다.

두 시뮬레이터 사이의 시간 동기화는 트레이스의 시간 정보를 통해 이루어진다. 트레이스에 포함된 NPU 시뮬레이터의 사이클 정보는 버스 시뮬레이터가 오프 칩 메모리 트랜잭션을 수행하는 시작 사이클을 결정한다. 버스 시뮬레이터의 오프 칩 메모리 트랜잭션 수행이 완료되면 종료 시점의 사이클 정보가 데이터와 함께 NPU 시뮬레이터에 전달되고 이를 통한 시간 동기화가 이루어지게 된다. NPU 시스템 시뮬레이터 구성에 사용되는 시뮬레이션 테크닉에 따라 시간 동기화 방법은 달라질 수 있다.

2.5.2 Bus

버스 시뮬레이터로 사용하고 있는 AMBA AXI4 TLM2.0 시뮬레이터는 그림 2.3와 같이 5개의 독립적인 채널을 가지고 있으며, 각 채널은 Read 주소 채널, Read 데이터 채널, Write 주소 채널, Write 데이터 채널, 그리고 Write 응답 채널로 구성된다. Read와 Write 커맨드는 서로 다른 채널 기반의 핸드쉐이킹 메카니즘으로 처리되기 때문에 요청 순서와 완료 순서는 달라질 수 있다.

AXI4 시뮬레이터는 메모리 요청 패킷 (Packet)을 생성하는 마스터 모듈과 메모리 요청을 수행하는 슬레이브 모듈을 가지고 있다. 마스터 모듈은 그림 2.2에서 보여

지는 것처럼 NPU 시뮬레이터로부터 전달 받은 최대 1,024 바이트 단위의 트랜잭션 (Transaction)을 버스 폭 (Bus Width) 크기의 64 바이트 단위로 나누어 슬레이브 모듈에 전달하게 된다. 슬레이브 모듈은 마스터 모듈로부터 받은 패킷들을 DRAM 시뮬레이터로 전달하고 트랜잭션을 구성하는 모든 패킷에 대한 응답을 받게 되면 마스터 모듈에게 패킷을 보내 메모리 요청이 완료되었음을 알린다. Read 요청의 경우 응답 패킷에 데이터가 포함이 되지만, Write 요청의 경우 데이터가 포함되지 않은 패킷만 전달한다.

기존의 AXI4 시뮬레이터는 마스터 모듈과 슬레이브 모듈의 1:1 연결만 가능한 구조를 가지고 있었다. 본 논문에서는 NPU 코어들이 여러 개의 SPM 뱅크 또는 DRAM과 연결되는 구조를 가져야 하기 때문에 크로스바 (Crossbar) 역할을 하는 인터커넥트

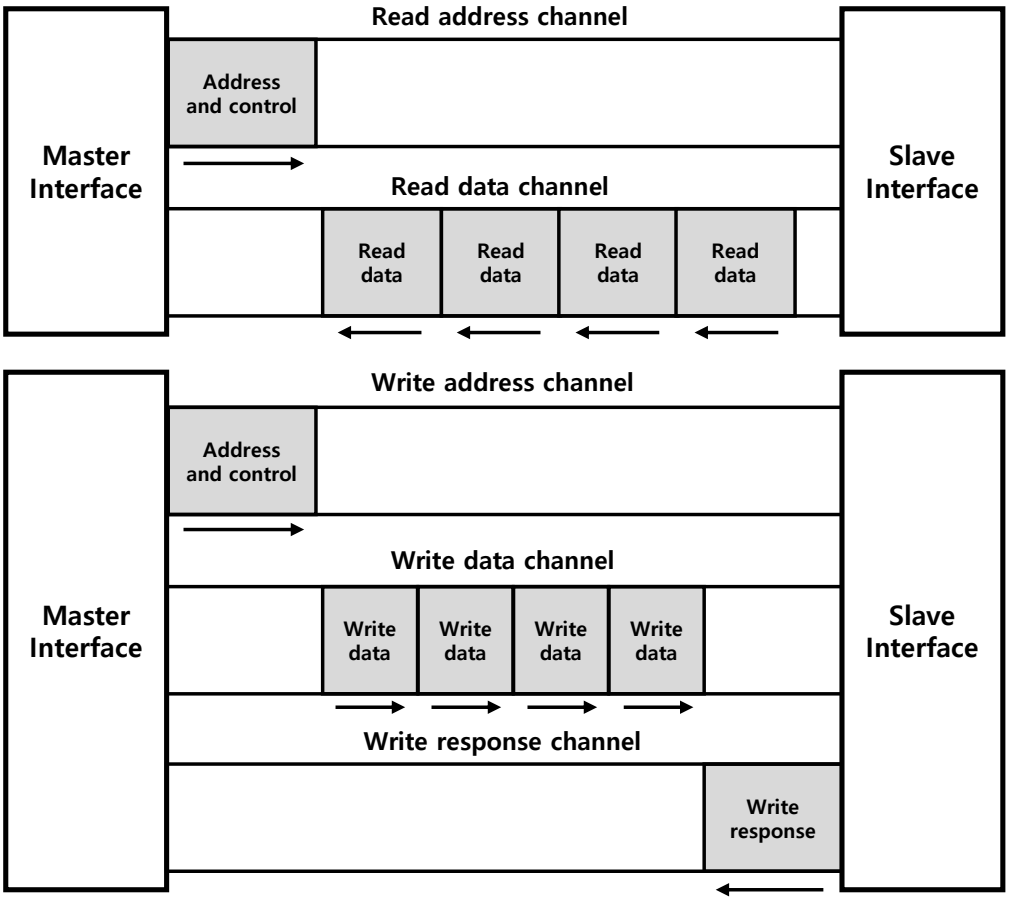

그림 2.3: AMBA AXI4 채널과 핸드쉐이킹 프로토콜 (ARM Holdings plc [1])

(Interconnect) 모듈을 추가하였다. 인터커넥트 모듈은 내부에 NPU 코어 수와 동일한 입력 큐 (Queue)를 가지고 있으며, 각 NPU로부터의 요청 패킷을 저장한다. 라운드 로빈 (Round Robin) 방식으로 하나의 큐를 선택하여 사이클마다 하나의 패킷을 처리할 수 있도록 하였다. SPM을 사용할 경우, 메모리 주소를 보고 타겟 메모리를 구분하여 SPM 또는 DRAM 모듈로 패킷을 전달한다.

2.5.3 DRAM

DRAM 시뮬레이터로 사용하는 Ramulator는 트레이스 (Trace) 기반으로 동작하는 사이클 단위의 정확한 시뮬레이션을 제공한다. TLM2.0으로 작성된 버스 시뮬레이터와의 인터페이스를 위해 Ramulator 내부에 브릿지 모듈을 추가하고 API를 통해 외부에서 호출 가능하도록 하였다. API를 통해 전달 받은 메모리 요청은 브릿지 모듈 내부의 큐 (Queue)에 저장되어 순차적으로 수행되며 버스 시뮬레이터는 매 사이클마다 외부 API를 통해 완료된 커맨드가 있는지 확인한다. 두 시뮬레이터 사이의 시간 동기화는 버스 시뮬레이터가 매 사이클마다 Ramulator의 외부 API를 호출하여 이루어진다.

Ramulator는 커맨드 종류에 따른 다양한 큐를 가지고 있으며 우선 순위에 따라 순차적으로 처리하고 있다. 커맨드의 종류에는 버스로부터 명시적으로 요청 받는 Read나 Write 커맨드 이외에 DRAM 동작에 따라 내부적으로 생성되는 Activate와 Refresh 커맨드가 존재한다. DRAM 컨트롤러의 스케줄링 방식에 따라 다음에 수행할 큐가 결정되는데 경우에 따라 요청 순서와 수행 시작 순서는 달라질 수 있다.

Chapter 3

DRAM 접근 모델 분석

NPU 성능에 영향을 주는 요인에는 여러 가지가 있지만 그 중 가장 많은 시간을 소모하며 성능을 저하시키는 요인 중 하나는 오프 칩 DRAM 접근이다. 그래서 NPU 설계에서 실제 하드웨어 구현 전에 오프 칩 DRAM 접근 지연 시간 및 NPU 성능을 예측하기 위한 방법으로 시뮬레이터를 이용하고 있다. 오프 칩 DRAM 접근으로 인한 지연 시간을 예측하는 방법은 시뮬레이터마다 다양하게 나타나고 있는데, 본 챕터에서는 오프 칩 DRAM 접근 지연 시간의 예측에 사용되는 DRAM 접근 모델을 크게 대역폭 모델 (Bandwidth Model), 분석 모델 (Analytical Model), 그리고 시뮬레이션 모델 (Simulation Model)로 분류하여 설명한다. Ramulator에서 제공하는 2채널의 LPDDR4 3200 Mbps 스펙을 기반으로 각 DRAM 접근 모델 환경을 구성하였고 다양한 CNN 모델로 실험을 진행하여 비교 분석하였다.

3.1 대역폭 모델 (Bandwidth Model)

대역폭 모델은 오프 칩 DRAM 접근의 지연 시간 예측에 DRAM의 최대 대역폭 (Max Bandwidth)만을 고려한 가장 간단한 모델이다. DRAM이 항상 최적의 조건에서 최대 대역폭으로 동작한다는 이상적인 가정 하에 계산된 DRAM 접근 지연 시간으로 NPU 성능을 예측한다. DRAM이 스펙에 명시된 최대 대역폭으로 동작하기 위해서는 동일한 Read 또는 Write 커맨드가 같은 행 (Row) 또는 페이지 (Page)에 속한 주소를 연속적으로 접근해야 한다. 그리고 데이터 전송 지연 시간을 제외한 다른 지연 시간은

거의 없거나 무시할 수 있다고 가정되어야 한다. 대역폭 모델에서 DRAM 접근 지연 시간 $D_{total}(B)$은 NPU가 DMA을 통해 DRAM으로 전달한 데이터 요청 크기를 사용하여 수식을 통해 계산되며, 식 3.1에서 보여지는 것처럼 요청 데이터 크기를 최대 대역폭으로 나누어 간단하게 얻을 수 있다. SCALESIM [4]은 시스템에서 필요로 하는 DRAM 대역폭 크기를 예측하기 위해 대역폭 모델과 유사한 방식을 사용하고 있다. 시뮬레이션을 통해 얻은 전체 DRAM 요청 데이터 크기를 시뮬레이션 수행 사이클로 나누어 DRAM 접근 지연 시간이 NPU 성능에 영향을 주지 않기 위해 필요한 최소 DRAM 대역폭을 계산한다.

$$D_{total}(B) = \frac{B}{BW_{dram}} \tag{3.1}$$

3.2 분석 모델 (Analytical Model)

대역폭 모델은 시스템이 일정한 DRAM 대역폭으로 동작하는 이상적인 환경을 가정하여 DRAM 접근 지연 시간을 계산하기 때문에 정확성을 신뢰하기 어렵다. 분석 모델은 대역폭 모델이 가진 한계를 극복하기 위해 오프 칩 DRAM 접근 시간에 영향을 미치는 다양한 요인들을 반영하여 수식화한 모델이다. 오프 칩 DRAM 접근에 영향을 주는 요인으로는 버스 전송 지연 시간, DRAM 페이지의 개행 지연 시간, DRAM 갱신 (Refresh) 지연 시간 등이 있다. 아래의 수식은 DRAM 시뮬레이터로써 사용되고 있는 Ramulator [6]의 LPDDR4 설정 값들과 AMBA AXI4 스펙을 참고하여 만들어진 것으로, 현재 MIDAP [5] 시뮬레이터에서 사용하고 있는 분석 모델의 수식들이다.

$$D_{total}(B) = D_{dram}(B) + D_{bus}(B) \tag{3.2}$$

$$D_{bus}(B) = C_{bus} * \left\lceil \frac{B}{B_{packet}} \right\rceil \tag{3.3}$$

$$D_{dram}(B) = D_{init}(B) + D_{tran}(B) + D_{ref}(B) \tag{3.4}$$

$$D_{init}(B) = C_{ready} * \left\lceil \frac{B}{B_{packet}} \right\rceil + C_{page} * \left\lceil \frac{B}{B_{page}} \right\rceil \tag{3.5}$$

$$D_{tran}(B) = \frac{B}{BW_{dram}} \tag{3.6}$$

$$D_{ref}(B) = C_{ref} * \left\lfloor \frac{D_{init}(B) + D_{tran}(B)}{Period_{ref}} \right\rfloor \tag{3.7}$$

DRAM 접근 지연 시간은 식 3.2와 같이 크게 DRAM 지연 시간과 버스 지연 시간으로 나눌 수 있다. DMA 로직은 AXI4 버스를 통해 1,024 바이트 단위로 패킷 (Packet)을 전달하는데, 버스 지연 시간 $D_{bus}(B)$은 식 3.3과 같이 패킷의 수와 패킷 당 소요되는 지연 시간 C_{bus}의 곱으로 표현될 수 있다. (B_{packet}은 실험에서 1,024 바이트로 설정되어 있다.) 패킷 당 발생하는 지연 시간 C_{bus}은 버스 데이터 전송 시간뿐만 아니라 DMA 로직과 DRAM 사이의 핸드쉐이크 프로토콜 (Handshake protocol)로 인한 지연 시간도 포함하고 있다. 패킷의 크기는 최대 4,096 바이트가 지원되지만 본 논문의 실험에서는 NPU와 DRAM의 사양을 고려하여 1,024 바이트를 사용하였다.

DRAM 지연 시간 $D_{dram}(B)$은 식 3.4에서 보여지는 것처럼 초기 지연 시간, 전송 지연 시간, 그리고 갱신 지연 시간으로 나누어 표현할 수 있다. 초기 지연 시간 $D_{init}(B)$은 식 3.5와 같이 준비 지연 시간과 페이지 지연 시간으로 나눈다. 1,024 바이트 크기의 패킷은 DRAM의 버스트 (Burst) 크기인 64 바이트의 작은 패킷들로 나누어지는데, 첫 번째 64 바이트 패킷은 이어지는 패킷들처럼 연속적인 접근으로 지연 시간이 숨겨지지 않기 때문에 전체 지연 시간에 직접적으로 영향을 주게 된다. 페이지 지연 시간은 DRAM의 각 페이지가 닫힌 (Close) 상태에서 열린 (Open) 상태로 바뀔 때 발생하는 지연 시간이다. 페이지가 열린 (Open) 상태로 바뀌는 시점을 정확하게 판단하는 것이 어렵기 때문에 페이지 크기 B_{page}인 8,092 바이트 단위로 페이지 열림 (Open)이 발생한다고 가정하였다.

전송 지연 시간 $D_{tran}(B)$은 어떠한 지연 시간도 포함되지 않는 순수한 전송 시간만을 나타내며, 식 3.6과 같이 요청된 데이터 크기를 DRAM의 최대 대역폭으로 나누어 계산한다. 대역폭 모델은 동일한 수식을 통해 전송 지연 시간만을 반영한 모델이었다. 갱신 지연 시간 $D_{ref}(B)$은 DRAM의 특성상 필요한 셀 (Cell)의 주기적인 갱신으로 인해 발생하는 지연 시간이다. 초기 지연 시간과 전송 지연 시간이 포함된 DRAM 수행 시간 동안 발생한 갱신 빈도 $\frac{1}{Period_{ref}}$와 갱신 지연 상수 값 C_{ref}의 곱으로 계산되며 식

3.7과 같이 표현할 수 있다.

3.3 시뮬레이션 모델 (Simulation Model)

분석 모델은 DRAM 접근 지연 시간에 영향을 주는 다양한 요인들을 반영하고 있음에도 불구하고 여전히 실시간으로 발생할 수 있는 외부 요인으로 인해 정확한 예측을 하지 못하였다. 시뮬레이션 모델은 DRAM 및 버스를 직접 시뮬레이션하여 얻은 오프 칩 DRAM 접근 지연 시간을 가지고 성능을 예측하는 모델이다. DRAM 및 버스 시뮬레이션이 추가됨으로 인해 전체 시스템 시뮬레이션 시간 (Time)이 다소 길어지긴 하지만 시뮬레이션 모델은 앞서 설명한 대역폭 모델과 분석 모델보다 훨씬 정확한 성능 예측을 가능하게 한다. 본 논문에서 사용한 시뮬레이션 모델은 DRAM 및 버스 시뮬레이션을 위해 챕터 2에서 설명한 Ramulator와 AMBA AXI4 TLM 2.0 시뮬레이터를 이용한다.

3.4 DRAM 접근 모델 비교

오프 칩 DRAM 접근 지연 시간이 NPU 성능에 미치는 영향을 측정하기 위해 앞서 설명한 세 가지 DRAM 접근 모델을 가지고 실험을 진행하였다. 다양한 데이터 패스를 사용하여 테스트하기 위해 많이 사용되는 8개의 CNN 모델들을 이용하였다. DRAM 및 버스 시뮬레이션이 필요하지 않은 대역폭 모델과 분석 모델은 NPU 시뮬레이터인 MIDAP 내부에 적용하여 독립적인 NPU 시뮬레이터로 실험을 진행하였고, 시뮬레이션 모델은 챕터 2에서 설명한 환경에서 DRAM 및 버스 시뮬레이션을 포함하여 실험을 진행하였다. 그림 3.1의 세로 축은 프리페치 (Prefetch)나 이중 버퍼링 (Double Buffering)으로 숨겨지지 않는 오프 칩 DRAM 접근 지연 시간을 보여주며, 각 CNN 모델의 세 막대 그래프는 왼쪽부터 대역폭 모델, 분석 모델, 그리고 시뮬레이션 모델의 결과를 나타낸다.

대역폭 모델은 다른 DRAM 접근 모델과 비교하여 현저하게 낮은 수치를 가지며 낙관적인 결과를 보여주었다. 실험에서 사용된 대역폭 값이 이상적인 환경에서만 얻

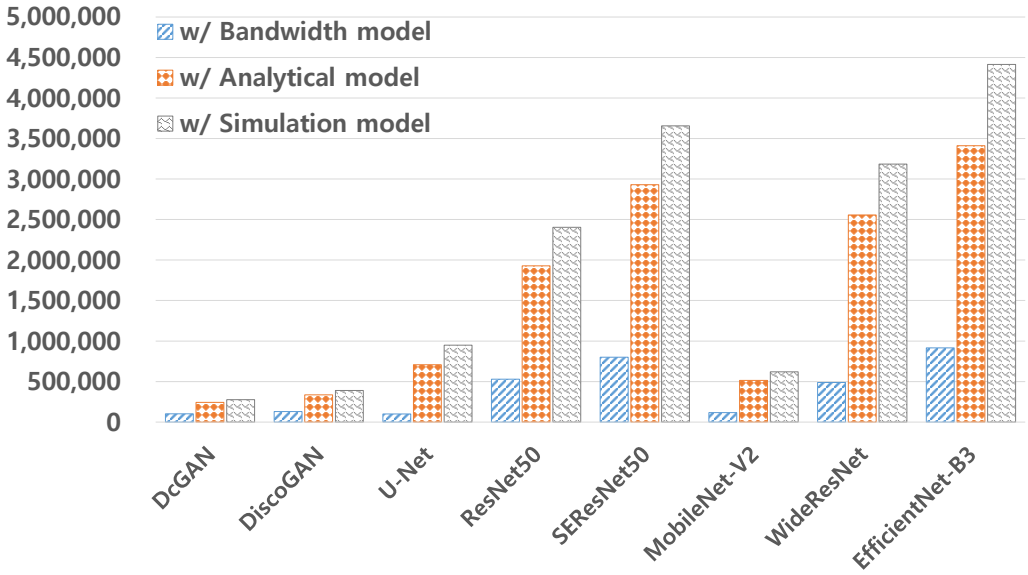

그림 3.1: DRAM 접근 모델의 지연 예측 시간 비교

을 수 있는 DRAM의 최대 대역폭이기 때문에 차이가 더욱 크게 나타났다. 하지만 최대 대역폭이 아닌 다른 임의의 대역폭 값을 선택하더라도 일정한 DRAM 대역폭이 유지된다는 가정이 있어야 하기 때문에 정확한 결과를 얻기 쉽지 않고 이를 위한 적절한 대역폭 값을 찾는 방법 또한 명확하지 않다. 따라서, 대역폭 모델에서 얻은 오프 칩 DRAM 접근 지연 시간으로 예측한 성능은 신뢰하기 어렵다.

분석 모델은 대역폭 모델과 비교하여 오프 칩 DRAM 접근 지연 시간을 보다 정확하게 예측하였다. 그러나 시뮬레이션 모델과 비교하면 약 12.5 %에서 25.3 % 정도 저예측한 결과를 보여주었다. 분석 모델이 버스 지연 시간과 DRAM 지연 시간을 고려하여 오프 칩 DRAM 접근 지연 시간을 계산했지만 공식화하여 표현하기 힘든 외부 요인들이 남아 있기 때문에 여전히 오차가 존재한다. 분석 모델은 연속된 주소의 데이터를 순차적으로 접근한다는 가정을 가지고 요청된 데이터 크기 기반의 수식으로 오프 칩 DRAM 접근 지연 시간을 계산하고 있다. 하지만 실제 환경에서는 이러한 데이터 접근의 연속성이 보장되지 않는다. AMBA AXI4 버스는 독립적인 Read / Write 채널 기반으로 동작하기 때문에 버스 혼잡도 및 DRAM 컨트롤러의 커맨드 선택 방법에 따라서 커맨드의 순서가 바뀔 수 있다. Ramulator [6]의 DRAM 컨트롤러는 Read

큐 (Queue)와 Write 큐 (Queue)의 상태에 따라 요청 시간에 상관 없이 Read 또는 Write 커맨드를 선택한다. 변경된 순서로 인해 Read / Write 커맨드가 교차하여 처리되면 한 종류의 커맨드가 연달아 처리될 때와 비교하여 DRAM의 준비 지연 시간이 훨씬 길어지게 된다. NPU의 데이터 요청 순서가 주변 환경으로 인해 달라지는 것은 데이터 흐름에 따라 동적으로 결정되기 때문에 분석 모델에서 이를 고려하여 정확하게 수식화하는 것은 쉽지 않다. EfficientNet-B3 [8]을 보면, 시뮬레이션 모델 대비 분석 모델의 저예측 정도가 약 22.7 %로 다른 CNN 모델들과 비교하여 크게 나타난다. 그 이유는 EfficientNet-B3의 Write 요청 비율이 상대적으로 높아 Read / Write 커맨드 사이의 교차 현상이 더 자주 발생하기 때문이다.

이 밖에 DRAM 접근 지연 시간에 영향을 주는 다른 요인들도 존재한다. 한 가지는 DRAM으로의 데이터 저장 방식이다. 만약 데이터가 DRAM의 페이지 (Page) 범위에 맞게 정렬되어 저장되지 않는다면 식 3.5와는 다르게 페이지를 오픈하기 위한 지연 시간이 추가로 발생하게 된다. 또 다른 요인은 DRAM 접근 중에 발생하는 DRAM 갱신 (Refresh) 빈도를 정확하게 체크하기가 어렵다는 것이다. DRAM이 항상 액티브 (Active) 상태로 동작하는 것을 가정하고 수행 시간 기반으로 식 3.7과 같이 계산을 하는데 실제 갱신 주기는 DRAM이 동작하지 않는 (Idle) 상태에서의 사이클도 포함하기 때문에 추가적인 갱신 지연 시간이 생길 수 있다. 이 밖에 실시간으로 발생하는 DRAM 뱅크 스위치 (Bank Switch)도 예측 오차에 영향을 줄 수 있다. 이와 같이 수식화하여 표현하기 어려운 다양한 요인들이 존재하기 때문에 분석 모델의 오프 칩 DRAM 접근 지연 시간 예측은 어느 정도 오차를 가질 수 밖에 없다. 따라서 NPU 성능에 영향을 미치는 오프 칩 DRAM 접근 지연 시간을 정확하게 측정하고 분석하기 위해서는 시뮬레이션 모델이 필요하다.

테이블 3.1은 시뮬레이션 모델을 통해 얻은 각 CNN 모델의 전체 시뮬레이션 사이클 (Cycle)과 그 중 오프 칩 DRAM 접근 지연 시간 (Cycle)이 차지하는 비율을 보여준다. U-Net [9]의 경우 대부분의 DRAM 접근 지연 시간이 NPU의 동작 시간 동안 프리페치 (Prefetch)나 이중 버퍼링 (Double Buffering)에 의해 성공적으로 숨겨지며 전체 시뮬레이션 사이클에서 차지하는 비율이 약 7.6 %로 매우 낮게 나타나는 것을 알

테이블 3.1: 8개 CNN 모델의 시뮬레이션 결과

CNN Models	Time (cycles)	DRAM Latency (%)
DcGAN	379,293	73.1%
DiscoGAN	579,038	67.3%
U-Net	12,422,621	7.6%
ResNet50	6,412,221	37.5%
SEResNet50	7,753,888	47.2%
MobileNet-V2	1,517,676	40.9%
WideResNet50	13,922,360	22.9%
EfficientNet-B3	8,991,889	49.1%

수 있다. 하지만 그 외의 다른 CNN 모델에서는 DRAM 접근 지연 시간이 높은 비율을 가지며 NPU 성능에 매우 큰 영향을 주고 있는 것을 볼 수 있다. 특히 DcGAN [10]은 전체 시뮬레이션 수행 사이클의 약 73.1 %가 오프 칩 DRAM 접근으로 인해 발생하고 있다. 어떤 NPU를 사용하는가에 따라 오프 칩 DRAM 접근 지연 시간이 성능에 주는 영향은 다르게 나타날 수 있다. 하지만 오프 칩 DRAM 접근 지연 시간이 NPU 성능에 큰 영향을 준다는 점은 분명하고 테이블 3.1에서 보여지는 것처럼 CNN 모델에 따라 다양한 결과가 나타날 것으로 예상된다. 본 챕터의 실험 결과는 NPU 성능에 영향을 주는 오프 칩 DRAM 접근 지연 시간을 줄이기 위해 오프 칩 SRAM을 이용한 메모리 계층 구조 탐색의 동기가 되었다.

Chapter 4

메모리 계층 구조

본 챕터에서는 오프 칩 DRAM과 NPU 사이에 큰 사이즈의 오프 칩 SRAM이 있는 메모리 계층 구조가 NPU 성능에 어떠한 영향을 주는지 살펴본다. 기존의 많은 연구에서 CNN 가속기는 일반적으로 칩 내부에 큰 사이즈의 SRAM을 두고 이를 활용하여 데이터 재사용을 최대화할 수 있도록 설계되었다. 그러나 임베디드나 모바일 환경에서는 성능, 파워 소모, 면적 등의 다양한 제약으로 충분히 큰 사이즈의 SRAM을 온 칩에 두고 사용하기에는 한계가 있다. 그래서 데이터의 재사용은 단일 레이어 또는 융합(Fused) 레이어 범위 안에서 제한적으로 이루어졌고 여러 레이어 사이에서 특성 지도(Feature Map) 데이터를 공유하는 것은 쉽지 않았다. 챕터 3의 시뮬레이션 모델 결과는 이러한 한계로 인해 NPU의 다양한 최적화에도 불구하고 오프 칩 DRAM 접근이 NPU 성능에 큰 영향을 준 것을 보여주었다. 이를 극복하기 위한 방안으로 오프 칩 SRAM이 여러 레이어 사이에서 오프 칩 DRAM 접근 없이 데이터 재사용을 가능하게 만들 것이라 예상하였다.

비슷한 아이디어를 가진 기존의 연구에는 NVDLA (NVIDIA Deep Learning Accelerator)를 통합한 FPGA (Field Programmable Gate Array) 가속 시스템 시뮬레이터인 FIRESIM [11]가 있다. 저자는 마지막 레벨 Cache (Last Level Cache, LLC)를 이용하여 특정 CNN 모델인 YOLOv3 [12]에서 성능 향상을 보일 수 있다고 말하였다. 하지만 특정 NPU에서 하나의 CNN 모델만 가지고 얻은 결과로는 오프 칩 SRAM으로의 Cache 성능을 일반화하기는 어렵다. 본 논문에서는 다양한 CNN 모델들을 가지고 오프 칩

SRAM으로의 Cache 성능을 측정하였고 CNN 모델의 특성에 따라 다르게 나타나는 큰 성능 편차는 Cache가 범용적인 목적의 NPU 시스템에 적합하지 않음을 보여주었다.

본 챕터에서는 다양한 CNN 모델을 사용하여 오프 칩 SRAM으로써 Cache와 스크래치패드 메모리 (Scratchpad Memory, 이하 SPM)를 비교 분석하였고 SPM이 Cache보다 범용적인 목적의 NPU 시스템에 더 적합함을 살펴본다. 또한, 오프 칩 SRAM으로 SPM을 사용하였을 때의 예상되는 파워 소모, 오프 칩 SPM에서의 데이터 재사용을 이용한 온 칩 SRAM 크기의 최적화, 그리고 오프 칩 SPM과 DRAM 사이에서의 데이터 압축에 대해 이어서 설명한다.

4.1 Cache을 사용한 메모리 계층 구조

Cache는 메모리 접근의 시간적 지역성 (Temporal Locality)과 공간적 지역성 (Spatial Locality)을 이용하여 오프 칩 DRAM 접근을 획기적으로 줄인 하드웨어로 범용 컴퓨터 시스템의 성능 개선에 크게 기여하였다. 본 섹션에서는 NPU 시스템에서 오프 칩 SRAM으로 Cache를 사용한 메모리 계층 구조가 NPU 성능에 미치는 영향에 대해서 다양한 종류의 CNN 모델들을 통해 살펴본다. 지금부터는 오프 칩 DRAM 접근 지연 시간이라는 용어를 오프 칩 메모리 접근 지연 시간으로 바꾸어 사용한다. NPU 성능에 영향을 주는 오프 칩 메모리 접근은 오프 칩 DRAM 접근뿐만 아니라 오프 칩 SRAM 접근도 포함하기 때문이다.

Cache는 하드웨어를 구성하는 방식에 따라 성능이 다르게 나타난다. NPU 시스템에서 Cache 성능 측정의 기준을 잡기 위해 다양한 Cache 파라미터 설정을 통한 설계 영역 탐색 (Design Space Exploration, DSE)을 진행하였다. 타겟 CNN 모델은 지난 챕터의 테이블 3.1에서 전체 시뮬레이션 수행 사이클 (Cycle) 대비 오프 칩 DRAM 접근 지연 시간 (Cycle)이 상대적으로 크게 나타났던 SEResNet50 [13]로 정하였다. 그림 4.1은 SEResNet50에서의 오프 칩 메모리 접근 지연 시간이 8MB Cache에서 블록 크기와 연관성 (Associativity) 크기에 따라 어떻게 변화하는지 보여주었고 세 가지 사실을 확인할 수 있었다. 첫 번째는 모든 종류의 Cache 조합 중에 2-way 집합 연관 (Set Associative) Cache가 가장 좋은 성능을 보였다는 점이다. 두 번째는 Cache 블록 크기가

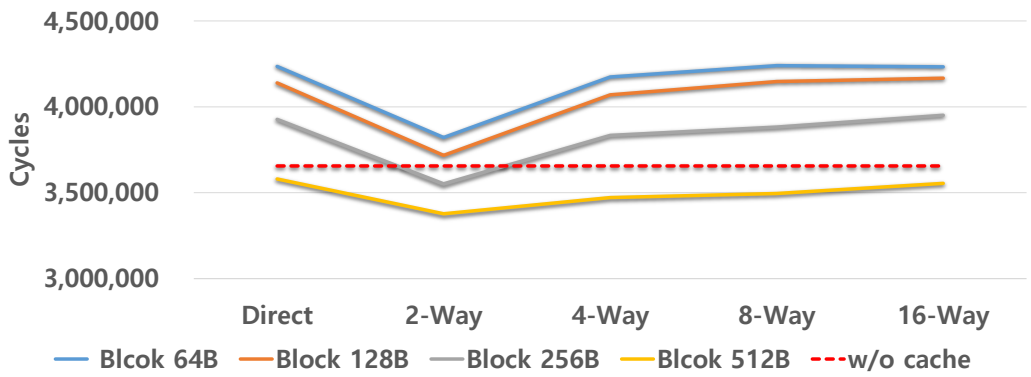

그림 4.1: Cache 옵션에 따른 오프 칩 메모리 접근 지연 시간 (SEResNet50)

Cache 성능에 큰 영향을 준다는 점이다. 대부분의 오프 칩 메모리 접근은 대용량의 특성 지도 (Feature Map)나 필터 가중치 (Filter Weight) 데이터를 읽기 위해 발생하는데, 큰 사이즈의 블록은 Cache의 공간적 지역성을 활용하여 NPU 성능을 높이는데 도움이 되었다. 마지막으로는 Cache을 사용함으로 인해 얻은 오프 칩 메모리 접근 지연 시간의 감소가 기대보다 크지 않았다는 점이다. 가장 좋은 성능을 보였던 8MB 2-Way 집합 연관 Cache에서 오프 칩 메모리 접근 지연 시간은 Cache를 사용하지 않았을 때와 비교하여 오직 7.6 % 정도 감소했을 뿐이다.

본격적으로 NPU 시스템에서의 Cache 성능을 확인하기 위해 설계 영역 탐색에서 가장 좋은 성능을 보였던 8MB 2-Way 집합 연관 Cache을 가지고 다양한 종류의 CNN 모델들에 대한 실험을 진행하였다. 그림 4.2의 그래프는 세 가지 Cache 설정에 따른 오프 칩 메모리 접근 지연 시간을 보여준다. 각 CNN 모델에서 가장 왼쪽의 막대 그래프는 Cache를 사용하지 않았을 때의 오프 칩 메모리 접근 지연 시간을 나타내고, 다음의 두 막대 그래프는 64 바이트와 512 바이트 블록 크기의 Cache를 사용하였을 때의 결과를 나타낸다. 이전 실험에서 Cache 블록 크기가 NPU 성능에 영향을 준 것을

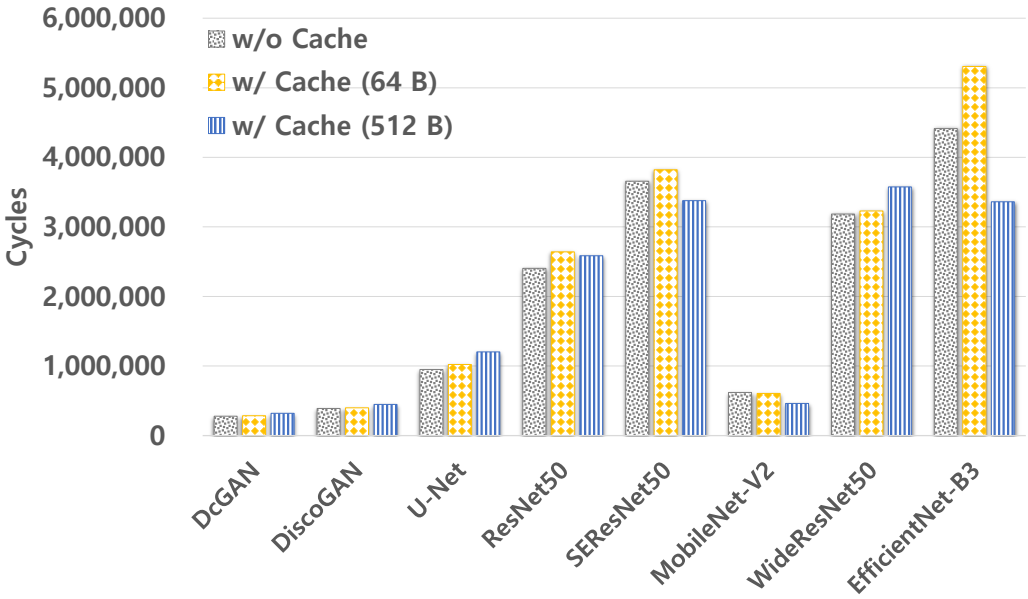

그림 4.2: Cache 유무에 따른 오프 칩 메모리 접근 지연 시간

참고하여 데이터 버스 폭 (Data Bus Width) 크기인 작은 사이즈의 64 바이트 Cache 블록과 비교적 큰 크기인 512 바이트의 Cache 블록을 가지고 실험을 진행하였다. 실험 결과는 동일한 Cache 조건일지라도 CNN 모델의 종류에 따라 오프 칩 메모리 접근 지연 시간의 차이가 다르게 나타남을 보여주었다.

Cache가 NPU 성능 향상에 도움을 줄 것이라는 기대와는 다르게 SEResNet50 [13], MobileNet-V2 [14], 그리고 EfficientNet-B3 [8]을 제외한 나머지 모델들은 Cache을 사용하였을 때의 성능이 오히려 Cache를 사용하지 않았을 때보다 하락된 결과를 보였다. 그리고 Cache을 사용하여 성능이 향상된 모델들도 범용 컴퓨터 시스템에서의 Cache 성능 기대치에는 크게 미치지 못하는 결과로 나타냈다. 또한, Cache 블록 크기에 따른 결과도 CNN에 모델에 따라 성능 향상 정도가 다르게 나왔고 일부 CNN 모델들은 큰 크기의 Cache 블록을 사용하였을 때 오히려 성능이 하락되었다.

NPU 시스템에서 Cache로 인한 성능 향상이 크지 않은 주요 요인으로는 NPU 구조의 특성이 있다. 일반적으로 NPU는 온 칩 SRAM에서의 데이터 재사용을 최대화할 수 있도록 설계된다. 대부분의 데이터는 온 칩 SRAM의 구조에 맞게 저장되고 온 칩 SRAM의 크기를 초과하는 큰 사이즈의 데이터는 오프 칩 메모리로 저장이 된다. 하지만 대부분의 NPU는 오프 칩 메모리에서의 데이터 저장 방식을 크게 고려하지 않기 때문에 Cache의 지역성을 충분히 활용하지 못한 결과를 보여주었다. 테이블 4.1은 오프 칩 SRAM으로 Cache을 사용하는 NPU 시스템에서 각 CNN 모델의 Cache 접근 통계

테이블 4.1: 8개 CNN 모델에 대한 Cache 접근 통계

CNN Models	Size	Write	R Hit (64 / 512)
DcGAN	3.6MB	1.69%	0.0% / 7.6%
DiscoGAN	5.4MB	1.12%	0.0% / 12.1%
U-Net	36.1MB	7.70%	2.4% / 22.4%
ResNet50	37.3MB	14.19%	16.5% / 28.6%
SEResNet50	50.6MB	20.90%	26.9% / 47.8%
MobileNet-V2	6.5MB	16.83%	20.2% / 51.8%
WideResNet50	78.9MB	6.90%	7.0% / 26.7%
EfficientNet-B3	56.0MB	36.10%	60.1% / 78.0%

를 보여준다. 두 번째 열인 Size는 Cache로 요청되는 모든 데이터의 총 합을 나타내며 다음 열은 Write 타입의 데이터 요청이 차지하는 비율을 보여준다. 마지막 열은 Read 요청에 대한 Cache Hit 비율을 나타내는데 범용 컴퓨터 시스템에서의 Cache Hit 비율이 일반적으로 90 % 이상인 것에 비해서 대부분의 CNN 모델에서 Cache Hit 비율이 매우 낮게 나타난 것을 확인할 수 있다.

ResNet50 [15]나 EfficientNet-B3 [8]와 같은 일부 CNN 모델에서는 Cache를 사용하였을 때 오히려 성능이 하락된 결과를 보여주었는데, 그 이유는 Cache의 동작 방식이 DRAM으로부터의 연속적인 데이터 전송을 방해할 수 있기 때문이다. DRAM 컨트롤러는 내부적으로 커맨드 큐 (Command Queue)를 가지고 있으며, 버스로부터 전달 받은 데이터 요청을 커맨드 큐에 저장하여 각 커맨드를 연속적으로 DRAM에 요청할 수 있게 한다. 하지만 Cache를 사용하게 되면 Cache Miss가 발생하였을 때 블록 사이즈 단위로 DRAM 요청이 발생하게 되어 Cache 블록 업데이트가 완료되기 전까지는 DRAM 컨트롤러의 커맨드 큐에 추가적인 데이터 요청이 들어오지 않게 된다. 실험에서 사용하는 DMA는 데이터 요청을 1KB 크기로 나누어 전송하는데 1KB보다 작은 사이즈의 블록을 사용할 경우 단일 Cache 데이터 요청이 복수의 Cache Miss을 발생시켜 오프 칩 DRAM 접근 지연 시간을 길어지게 하는 것이다. 반면에 큰 사이즈의 블록을 사용할 경우 Cache Miss에 대한 페널티가 작은 사이즈의 블록을 사용할 때보다 크게 나타나기 때문에 WideResNet50 [16]처럼 Cache Hit 비율이 높아졌음에도 불구하고 오히려 오프 칩 메모리 접근 지연 시간은 길게 나타날 수 있다. 따라서 적절한 크기의 Cache 블록을 사용하는 것은 NPU 성능 향상에 필수적인 요소이다. 하지만 CNN 모델마다 좋은 성능을 보이는 Cache 블록의 크기는 다르게 나타나기 때문에 모든 CNN 모델에서 좋은 성능을 얻을 수 있는 옵션을 찾기는 쉽지 않다.

EfficientNet-B3, MobileNet-V2, 그리고 SEResNet50은 큰 크기의 Cache 블록을 사용하였을 때 성능이 향상되었는데 테이블 4.1을 보면 다른 모델에 비해 상대적으로 Write 요청의 비율이 높게 나타나는 것을 볼 수 있다. Write 동작은 레이어 간 데이터 재사용을 위해 현재 레이어의 출력 특성 지도 (Feature Map) 데이터를 저장할 때 발생하는데, Write 동작 시 업데이트된 Cache 블록이 다음 데이터 접근에서 Cache Hit

로 이어지게 된다. Cache를 사용하지 않을 때 있었던 출력 특성 지도 데이터의 DRAM 접근이 발생하지 않기 때문에 전체적으로 오프 칩 메모리 접근 지연 시간을 줄일 수 있다.

일부 CNN 모델에서는 Cache로 인해 성능이 향상되었지만 그 비율이 크지 않았고 오히려 성능이 하락된 CNN 모델들도 많았다. NPU 시스템에서 Cache는 데이터 패스에 의한 성능 편차를 크게 갖는 것으로 판단된다. 따라서, 특정 CNN 모델을 대상으로 하는 NPU 시스템이 필요한 것이 아니라면 범용적인 목적의 NPU 시스템에서는 오프 칩 SRAM으로 Cache를 사용하는 것은 적합하지 않다고 보여진다.

4.2 SPM을 사용한 메모리 계층 구조

SPM (Scratchpad Memory)은 Cache와는 달리 태그 및 일관성 기능을 위한 하드웨어 로직을 제거한 SRAM으로 DRAM과는 분리된 별도의 메모리 주소를 사용함으로써 프로그래머가 SRAM을 자유롭게 컨트롤 할 수 있도록 한다. 본 섹션에서는 NPU 시스템에서 오프 칩 SRAM으로써 SPM을 사용한 메모리 계층 구조가 NPU 성능에 미치는 영향에 대해 다양한 종류의 CNN 모델들을 통해 살펴본다. 실험에 사용된 SPM은 하나의 뱅크 (Bank)와 컨트롤 코어로 구성된다. SPM 뱅크는 DRAM과 독립된 주소 영역을 가지고 있으며 매 cycle 당 64 바이트의 데이터를 처리한다. SPM 뱅크에는 NPU와 DRAM과의 Read / Write 데이터 요청을 처리하기 위한 2개의 큐가 있으며 NPU와 연결되는 큐가 DRAM과 연결되는 큐보다 우선 순위를 가진다. 컨트롤 코어는 SPM 트레이스 (Trace)를 읽고 저장된 커맨드를 순차적으로 처리한다. 커맨드의 종류에는 데이터 전송을 위한 READ / WRITE와 데이터 동기화를 위한 SIGNAL / WAIT가 있다.

그림 4.3는 세 가지 조건의 오프 칩 SRAM 설정을 가지고 실험한 결과를 보여주며

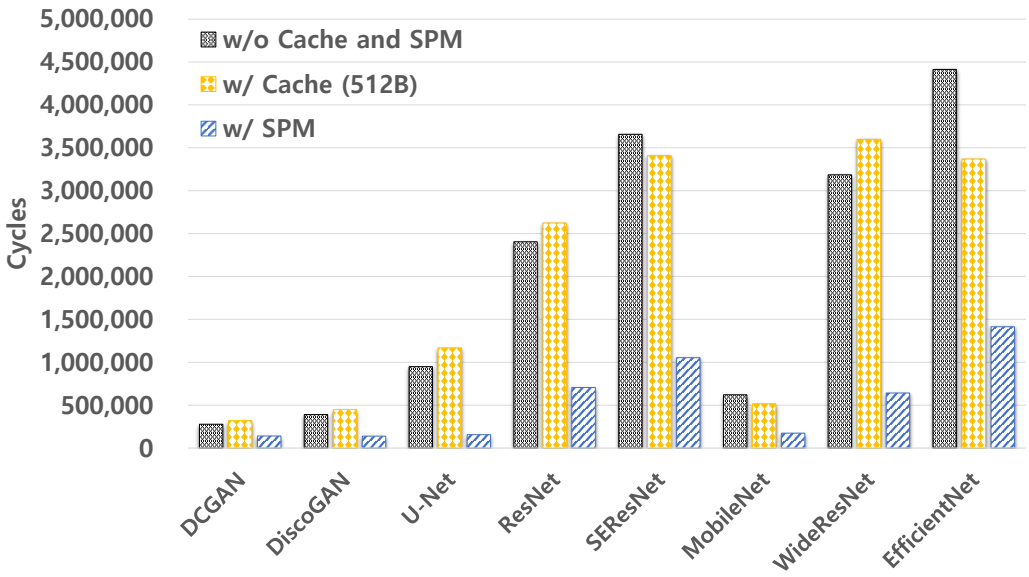

그림 4.3: SPM 유무에 따른 오프 칩 메모리 접근 지연 시간

세로 축은 CNN 모델의 오프 칩 메모리 접근 지연 시간을 나타낸다. 각 CNN 모델의 첫 번째 막대 그래프는 오프 칩 SRAM을 사용하지 않았을 경우의 오프 칩 메모리 접근 지연 시간, 두 번째는 512B 블록 크기의 Cache를 사용하였을 때의 결과, 마지막은 SPM을 사용하였을 때의 결과를 보여 준다. 실험 결과는 SPM을 사용하였을 때 오프 칩 메모리 접근 지연 시간의 감소 폭이 Cache를 사용하였을 때보다 훨씬 크게 나타났다. SPM을 사용하였을 때 오프 칩 메모리 접근 지연 시간은 오프 칩 SRAM을 사용하지 않았을 때와 비교하여 최소 49.2 %, 최대 83.6 % 개선되었다. 특히, SEResNet50 [13]의 경우 오프 칩 메모리 접근 지연 시간은 71.2 % 줄었고 전체 시뮬레이션 사이클 (Cycle)은 33.6 % 감소하였다. 오프 칩 메모리 접근 시간이 전체 시뮬레이션 사이클에서 차지하는 비율에 따라서 NPU의 성능 개선 정도는 차이가 있지만 실험에 사용된 모든 CNN 모델에서 큰 성능 향상을 보인 것을 확인할 수 있다.

SPM을 오프 칩 SRAM으로 채택할 때 시스템 성능이 크게 개선될 수 있었던 가장 큰 이유는 NPU가 사용할 데이터를 SPM에 미리 프리페치 (Prefetch) 하는 것이 가능하기 때문이다. NPU의 데이터 요청을 받아야만 Cache Miss 및 블록 교체 정책에 따라 DRAM으로부터의 데이터 프리페치가 가능했던 Cache와는 다르게 SPM은 컨트롤러가 NPU 컴파일러에 의해 생성된 SPM 트레이스를 읽고 DRAM으로부터의 데이터 프리페치를 미리 수행할 수 있다. 이를 통해 NPU 성능 하락의 주요 원인인 DRAM 접근 지연 시간이 NPU에 직접적으로 영향을 주지 않아서 오프 칩 메모리 접근 지연 시간이 크게 감소하였다.

SPM을 사용함으로 인해 감소된 DRAM 접근 빈도 또한 NPU 성능 향상에 크게 기여하였다. 테이블 4.2는 8개의 CNN 모델에서의 DRAM과 SPM 접근 통계 값을 보여주는데 SPM 사용시 DRAM 접근 빈도가 크게 감소한 것을 확인할 수 있다. 추론 과정에서 레이어 간 데이터 재사용을 위한 특성 지도 (Feature Map) 출력이 발생할 경우, SPM을 사용하지 않는다면 이를 쓰고 읽기 위한 DRAM 접근이 반복적으로 필요하지만 SPM을 사용할 경우에는 추가 DRAM 접근 없이 SPM에서 임시 데이터를 관리할 수 있기 때문에 DRAM Write가 크게 감소한다. 또한, DRAM에서 한 번 읽은 데이터는 소프트웨어 관리 하에 SPM에서 재사용될 수 있기 때문에 DRAM Read 횟수

테이블 4.2: 8개의 CNN 모델에 대한 SPM 및 DRAM 접근 통계

CNN Models	w/o SPM	w/ SPM	
	D (R/W)	D (R/W)	S (R/W)
DcGAN	58K / 1K	58K / 0	58K / 59K
DiscoGAN	87K / 1K	87K / 0	87K / 88K
U-Net	546K / 46K	501K / 0	546K / 547K
ResNet50	524K / 87K	437K / 0	524K / 524K
SEResNet50	655K / 173K	478K / 0	655K / 651K
MobileNet-V2	88K / 18K	70K / 0	88K / 88K
WideResNet50	1.2M / 89K	1.1M / 0	1.2M / 1.2M
EfficientNet-B3	587K / 332K	234K / 0	587K / 565K

또한 줄어든 것을 확인할 수 있다.

컴파일 단계에서 모든 데이터 패스가 결정되는 NPU의 특성은 데이터 접근을 소프트웨어로 관리할 수 있는 SPM을 사용하기에 매우 적합하다. 메모리 관리를 위한 소프트웨어의 최적화가 추가적으로 필요하지만 SPM을 통해 얻을 수 있는 성능 이득은 소프트웨어 개발 비용을 감수하기에 충분할 것으로 사료된다. 하드웨어에 의해 컨트롤되어 데이터의 유효성을 보장할 수 없었던 Cache와 비교하여 SPM은 보다 효율적으로 데이터를 관리할 수 있기 때문에 오프 칩 메모리 접근 지연 시간을 줄이기 위한 좋은 선택이 될 수 있을 것이라 생각된다.

4.3 SPM 에너지 소모

본 섹션에서는 오프 칩 SRAM으로 SPM을 사용할 때 발생하는 에너지 소모의 변화를 확인한다. 메모리 분석에 많이 쓰이는 CACTI [7] 툴을 사용하여 32nm 공정 기반의 메모리 모델을 만들었으며 동적 에너지 소모만을 고려하여 실험을 진행하였다. 테이블 4.3은 8MB SRAM과 1GB DRAM에 접근시 소모되는 Read / Write 에너지 값을 나타내고 그림 4.4은 SPM을 사용하지 않았을 때 DRAM에서 발생하는 에너지 소모와 SPM을 사용하였을 때 DRAM 및 SPM에서 발생하는 에너지 소모를 비교하여 보여준다. 그래프의 세로 축은 SPM과 DRAM의 총 에너지를 나타내는데 테이블 4.2의 각 메모리 접근 빈도 수와 테이블 4.3의 접근 당 에너지 소모 값을 곱하여 계산되었다.

SPM을 추가하였을 때의 에너지 소모 변화는 CNN 모델에 따라 다르게 나타났다. 오프 칩 DRAM 접근 빈도가 SPM으로 인해 감소되어 DRAM 에너지 소모는 모든 CNN

테이블 4.3: CACTI에 의해 모델링된 메모리 접근시 에너지 소모

Memory	Size	Read Energy (pJ)	Write Energy (pJ)
SPM	8 MB	919.84	805.57
DRAM	1 GB	4429.05	10720.30

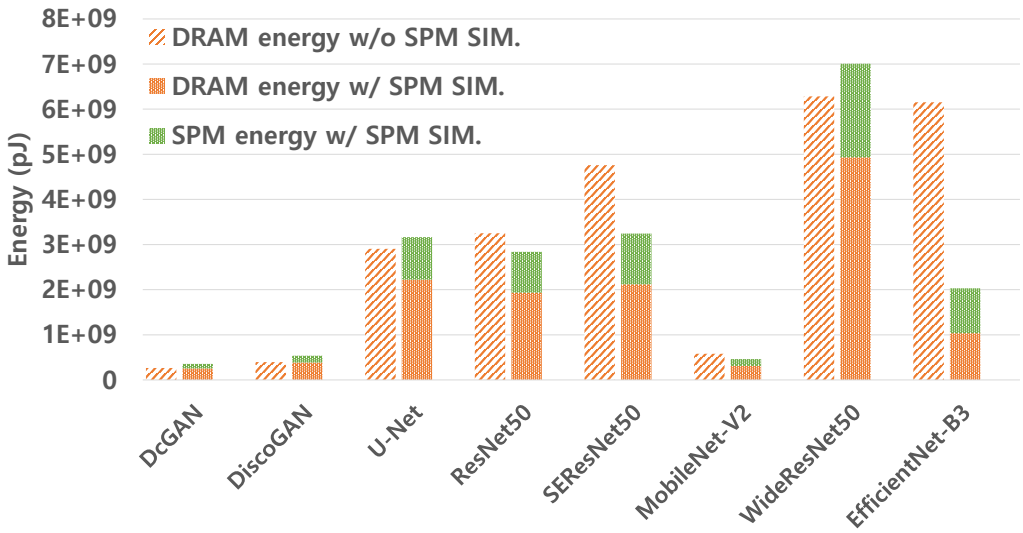

그림 4.4: 8 가지 CNN 모델의 SPM 에너지 소모

모델에서 감소된 결과를 보였지만 그만큼 SPM 접근으로 인한 에너지 소모가 증가하였다. 전체 에너지 소모에 가장 큰 영향을 준 것은 SPM에 의해 감소된 오프 칩 DRAM의 접근 빈도였다. 테이블 4.2의 EfficientNet-B3 [8]를 보면 SPM을 사용하였을 때 DRAM 데이터의 중복 접근이 사라지면서 DRAM Read 접근의 빈도 수가 크게 감소한 것을 알 수 있다. SPM에 비해 상대적으로 큰 에너지를 소모하는 DRAM 접근 빈도의 감소는 전체 에너지 소모가 약 67 % 감소한 결과로 나타났다. 반면에 WideResNet50 [16]는 오프 칩 메모리 접근 지연 시간이 79.8 %로 크게 감소하였지만 전체 에너지 소모는 오히려 11.5 % 증가하였는데 그 이유는 상대적으로 큰 필터 (Filter)의 사용으로 가중치 (Weight) 데이터의 크기가 커지기 때문에 DRAM Read 빈도수 감소가 크지 않았다.

실험은 DRAM과 SPM의 접근 빈도 기반으로 동적 에너지 소모만 측정했기 때문에 실제 하드웨어에서의 측정 결과와는 차이가 있을 수 있다. 하지만 SPM이라는 별도의 하드웨어가 추가되어 에너지 소모가 다소 증가하더라도 오프 칩 DRAM 접근이 감소하면서 에너지 소모의 증가가 생각보다 크게 나타나진 않을 것이라 예측할 수 있었고 SPM을 사용하였을 때의 NPU 성능 개선을 생각해본다면 약간의 에너지 소모 증가는 어느 정도 감수할 수 있는 부분이라 생각된다.

4.4 온 칩 SRAM 크기 최적화

칩 면적은 비용과 파워 소모 감소 측면에서 SoC (System on Chip) 디자인에 매우 중요한 요인으로 다루어져 왔고 칩 면적을 줄이기 위한 많은 노력이 계속되고 있다. NPU는 전체 면적의 큰 부분을 온 칩 전역 버퍼 (Global Buffer)를 위해 사용하고 있다. 대부분의 NPU는 온 칩 전역 버퍼를 최대한 활용하여 데이터 재사용을 하도록 디자인 되기 때문에 가능한 큰 크기의 온 칩 메모리를 사용하려고 한다. 하지만 임베디드나 모바일과 같이 환경에서는 칩 면적에 대한 제약이 크기 때문에 온 칩 전역 버퍼의 최대 크기는 제한될 수 밖에 없다. 따라서, NPU를 설계할 때 성능과 면적 사이에 적절한 균형을 유지하는 것이 무엇보다 중요하다.

앞서 설명한 섹션 4.2에서는 NPU와 오프 칩 DRAM 사이에 위치한 오프 칩 SPM이 모든 CNN 모델을 대상으로 오프 칩 메모리 접근 지연 시간 개선에 크게 기여한 것을 보여주었다. 본 섹션에서는 온 칩 SRAM의 크기를 줄이고 오프 칩 SPM에서의 데이터 재사용을 늘렸을 때 어느 정도 성능을 보일 수 있을지에 대한 실험을 진행하였다. 오프 칩 SPM은 온 칩 SRAM보다 데이터 접근 시간이 느리지만 상대적으로 큰 사이즈를 가지고 있기 때문에 DRAM 접근 없이 SPM에서의 데이터 재사용이 가능하다. 온 칩 SRAM 크기 감소로 인한 NPU의 성능 저하가 다소 발생하겠지만 오프 칩 SPM에서의 높은 데이터 재사용을 통해 충분한 성능 이득을 볼 수 있을거라 예상하였다.

테이블 4.4: 온 칩 메모리 크기에 따른 수행 시간 (Cycle)과 오프 칩 메모리 접근 지연 시간 비율 (%)

CNN Models	MEM 1,280KB	MEM 1,024KB	MEM 768KB
DcGAN	242,793(58.0%)	242,793(58.0%)	242,793(58.0%)
DiscoGAN	329,012(42.4%)	329,012(42.4%)	329,012(42.4%)
U-Net	11,629,916(1.3%)	11,682,869(1.8%)	11,779,946(2.6%)
ResNet50	4,713,398(15.0%)	4,755,842(15.7%)	4,851,908(17.4%)
SEResNet50	5,150,330(20.4%)	5,217,425(21.5%)	5,354,882(23.5%)
MobileNet-V2	1,070,456(16.2%)	1,081,067(17.0%)	1,138,830(21.2%)
WideResNet50	11,379,810(5.6%)	11,440,317(6.1%)	11,616,816(7.6%)
EfficientNet-B3	5,991,630(23.6%)	6,199,476(26.1%)	6,480,897(29.3%)

실험에서 사용된 NPU인 MIDAP [5]은 크게 두 종류의 온 칩 전역 버퍼를 가지고 있다. 하나는 특성 지도 (Feature Map) 데이터를 위한 1MB SRAM인 FMEM (Feature Memory)이고 또 다른 하나는 필터 가중치 (Filter Weight) 데이터를 위한 256KB SRAM인 WMEM (Weight Memory)이다. 본 실험은 WMEM의 크기는 유지한 상황에서 상대적으로 큰 사이즈를 가진 FMEM의 크기를 728 KB, 512 KB로 줄여가며 진행되었다. 테이블 4.4은 온 칩 SRAM 전체 크기에 따른 전체 시뮬레이션 수행 사이클 (Cycle)을 보여준다. 그 중 오프 칩 메모리 접근 지연 시간이 차지하는 비중은 괄호 안에 표기되어 있다.

DcGAN [10]과 DiscoGAN [17] 모델의 경우, 필요한 특성 지도 데이터의 크기가 온 칩 전역 버퍼인 FMEM의 크기보다 크지 않아서 성능 차이는 발생하지 않았다. 반면에 다른 CNN 모델에서는 온 칩 SRAM의 크기 감소에 따라 최소 1.3 % (U-Net [9])에서 최대 8.2 % (EfficientNet-B3 [8])까지 전체 수행 사이클 (Cycle)이 다소 하락하는 것을 볼 수 있다. 하지만 온 칩 SRAM 크기 감소로 인한 NPU 성능 하락의 정도는 생각보다 크게 나타나지 않았고 SPM을 사용하지 않았을 때의 성능과 비교하면 여전히 큰 성능 향상을 보이고 있다.

온 칩 SRAM은 NPU의 전체 면적에서 약 70 % 이상의 매우 큰 부분을 차지하고 있기 때문에 온 칩 SRAM 크기의 40 %을 줄인다는 것은 칩 면적의 감소에 큰 도움이 될 것이다. CACTI [7] 툴에서 32nm 공정 기반으로 모델링한 결과는 온 칩 SRAM 면적을 $2.44mm^2$에서 $1.68mm^2$로 줄일 수 있음을 보여주었다. 온 칩 SRAM 크기가 감소하여 챕터 4의 그림 4.3이 보여준 기대 성능보다는 낮은 성능 향상을 보이겠지만 그만큼 칩 크기가 최적화될 수 있기 때문에 적절한 온 칩 SRAM 크기의 선택으로 성능 향상과 칩 면적 감소 모두 만족시킬 수 있는 결과를 얻을 수 있을 것이라 생각된다.

4.5 SPM 에서의 데이터 압축

오프 칩 메모리 접근의 크기를 줄이기 위한 다양한 방법 중 많이 쓰이는 것으로 데이터 압축이 있다. 여러 가지 알고리즘을 적용하여 DRAM에 저장될 데이터의 크기를 최소화함으로써 오프 칩 DRAM 접근 지연 시간이 NPU 성능에 주는 영향을 줄인다. 데이터 압축은 온 칩 내부, NPU와 오프 칩 SRAM 사이, 또는 오프 칩 SRAM과 오프 칩 DRAM 사이에서 이루어질 수 있고 위치에 따라 요구되는 하드웨어 복잡도나 NPU의 성능 향상 정도가 달라질 수 있다. 본 논문에서는 오프 칩 SRAM으로써 SPM을 사용하였을 때 SPM과 DRAM 사이에서의 데이터 압축이 NPU 성능에 어떠한 영향을 주는지 살펴 본다.

실험은 두 가지 가정을 가지고 진행이 되었다. 한 가지는 데이터 압축은 일정한 압축률을 항상 유지할 수 있다는 가정이다. 다양한 데이터 압축 알고리즘이 존재하지만 본 논문의 연구 범위에는 포함되지 않기 때문에 특정 비율의 데이터 압축이 항상 유지된다고 가정하고 실험을 진행하였다. 또 다른 한 가지는 데이터 압축 및 해제가 하드웨어를 통해 이루어지며 이에 대한 지연 시간이 거의 없거나 성능에 영향을 주기에 미미하다고 가정하였다. 압축 알고리즘을 실제로 적용한 것은 아니라서 하드웨어 지

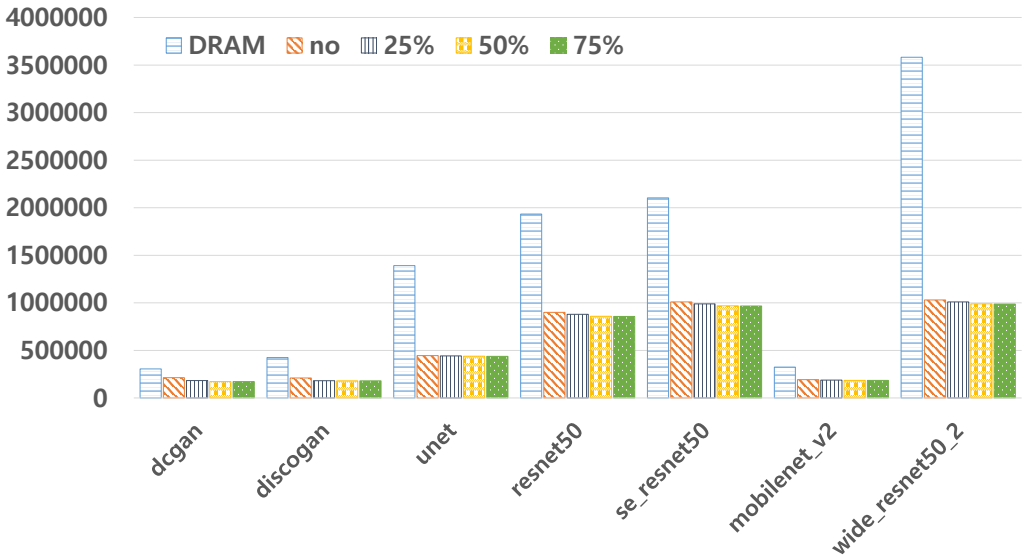

그림 4.5: 데이터 압축률에 따른 오프 칩 메모리 접근 지연 시간 (SPM 환경)

연 시간을 명시적으로 정의하기가 어렵다는 점이 있었고 SPM을 사용하면 데이터 압축 및 해제가 직접적으로 NPU 성능에 영향을 주지 않을 것이라고 예상했기 때문이다.

그림 4.5은 데이터 압출률에 따라 변화하는 오프 칩 메모리 접근 지연 시간을 보여준다. 가장 왼쪽의 막대 그래프는 SPM을 사용하지 않고 데이터 압축도 하지 않는 환경에서의 결과를 나타낸다. 다음 4 개의 막대 그래프는 SPM을 사용하는 환경에서 데이터 압축률이 0%, 25%, 50%, 그리고 75% 에 해당할 때의 결과를 보여준다. 실험 결과는 데이터 압축이 SPM을 사용하는 환경에서 성능 향상을 거의 보이지 않는 것으로 나타났다.

데이터 압축이 NPU 성능 개선에 큰 도움이 되지 않은 이유는 DRAM 데이터 크기의 감소가 NPU 성능에 직접적으로 영향을 주지 않기 때문이다. NPU가 사용하는 데이터는 항상 SPM을 통해 가져오게 되는데, NPU가 SPM에 데이터 요청을 할 때 대부분의 데이터는 이미 SPM에 프리페치되어 있어서 데이터 압축으로 인한 NPU 성능 향상의 여지가 크지 않다. 하지만 오프 칩 DRAM 접근의 크기가 데이터 압축률 만큼 감소되기 때문에 오프 칩 DRAM 접근으로 인한 동적 에너지 소모를 크게 줄일 수 있을 것이라고 예상한다.

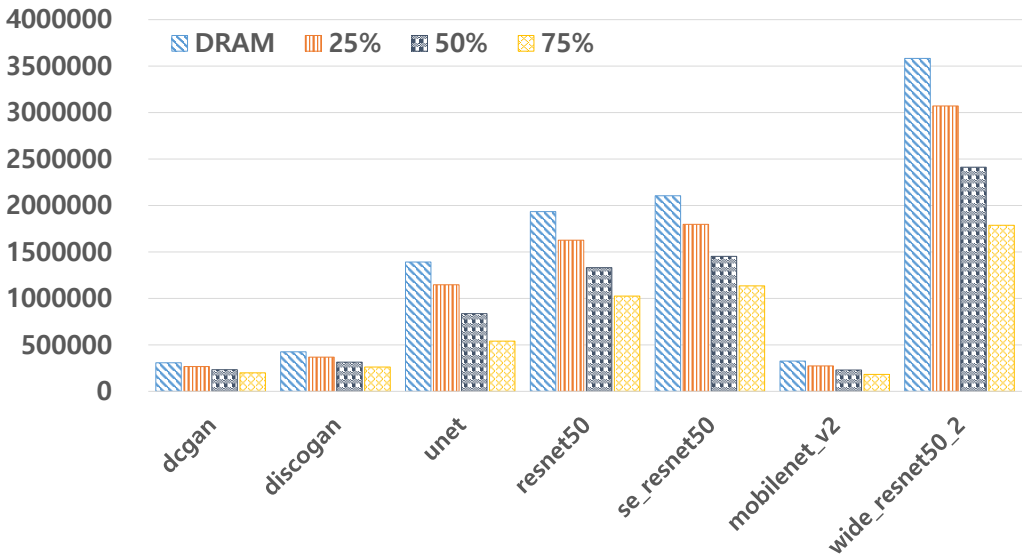

그림 4.6: 데이터 압축률에 따른 오프 칩 메모리 접근 지연 시간 (DRAM 환경)

데이터 압축이 항상 도움이 되지 않는 것은 아니다. 그림 4.6은 SPM을 사용하지 않는 환경에서 데이터 압축률에 따른 오프 칩 메모리 접근 지연 시간을 나타낸다. 실험 결과는 데이터 압축률이 높아질수록 오프 칩 메모리 접근 지연 시간이 크게 감소하는 것을 보여주었고 SPM을 사용하지 않는 환경이라면 데이터 압축이 매우 효율적임을 알 수 있다.

Chapter 5

멀티 NPU 시스템

다양한 CNN 응용의 발전에 따라 엣지 디바이스에서 사용 가능한 응용의 개수 또한 증가하고 있다. 하지만 NPU 특성상 여러 개의 응용을 하나의 코어에서 가속하기는 쉽지 않다. 이를 교차 실행 방식으로 지원할 경우 비효율적인 메모리 접근 횟수의 증가로 인해 시스템 성능이 감소하게 된다. 따라서, 여러 응용을 동시에 수행할 수 있는 멀티 NPU 시스템이 앞으로 필요해지고 멀티 NPU에서의 효율적인 자원 관리 또한 중요해질 것이라 예상된다.

본 챕터에서 기존의 단일 NPU 코어로 동작하던 NPU 시스템을 여러 개의 NPU 코어로 구성된 멀티 NPU 시스템으로 확장하고 멀티 NPU 시스템에서 오프 칩 메모리 접근이 성능에 미치는 영향을 살펴본다. 챕터 4와 동일한 메모리 계층 구조를 가지며 오프 칩 SRAM으로써 스크래치패드 메모리 (Scratchpad Memory, SPM)가 사용된다. 확장된 멀티 NPU 시스템은 최대 16개의 NPU 코어를 지원하며 최대 16개의 SPM 뱅크 (Bank)와 연결된 구조를 갖는다.

5.1 버스 토폴로지 (Bus Topology)

NPU 시스템의 버스로 사용되고 있는 AMBA AXI4 버스는 기본적으로 세 가지 토폴로지를 지원하고 있다. 첫 번째는 Shared Address Shared Data (SASD)로 여러 AXI4 마스터가 Read / Write 주소 채널과 데이터 채널을 모두 공유하여 사용한다. 하나의 마스터가 채널을 사용하면 다른 마스터들은 모두 대기 상태가 되기 때문에 세 가지 토

폴로지 중 가장 성능이 떨어진다. 두 번째는 Multiple Address Multiple Data (MAMD)로 여러 AXI4 마스터가 여러 개의 Read / Write 주소 채널과 데이터 채널을 사용한다. 모든 마스터는 사용하는 채널의 수만큼 서로 다른 슬레이브와 동시에 커뮤니케이션이 가능하기 때문에 가장 좋은 성능을 보이지만 필요로 하는 하드웨어 유닛의 수가 SASD보다 훨씬 많아지게 된다. 마지막은 Shared Address Multiple Data (SAMD)로 여러 AXI4 마스터가 Read / Write 주소 채널은 공유하지만 데이터 채널은 여러 개를 사용한다. SASD와 MAMD의 중간 버전으로 볼 수 있다.

MAMD 기반으로 만들어진 멀티 NPU 시스템은 마스터 모듈과 슬레이브 모듈의 연결 방식에 따라 시스템의 성능과 필요로 하는 하드웨어 유닛의 크기가 달라지게 된다. 예를 들면, 최대 지원 가능한 수인 16개의 NPU와 16개의 SPM 뱅크가 모두 완전한 연결 (Fully Connected)을 이루는 16X16 크로스바 구조는 각 NPU가 서로 다른 16개의 SPM 뱅크에 간섭 없이 동시에 접근 가능하기 때문에 다른 어떠한 구성보다도 좋은 성능을 보일 수 있다. 하지만 그만큼 복잡한 하드웨어 구성이 필요하기 때문에 시스템에 요구되는 LUT (Look Up Table), Register, Mux 등의 하드웨어 유닛 수 또한 많아지게 된다. 반면에 동시 접근 가능한 SPM 뱅크의 수에 제약을 두고 적은 수를 사용하게 된다면 필요한 하드웨어 유닛의 수가 감소하게 된다. 하지만 SPM 뱅크에서의 자원 경합이 빈번하게 발생할 수 있기 때문에 성능 하락은 피할 수 없다. 테이블 5.1은 FPGA (Field Programmable Gate Array) 설계 툴인 Vivado [18]를 이용하여 AXI4 크로스바를 합성하였을 때 필요로 하는 하드웨어 유닛의 수를 보여준다. 16개의 NPU가 공유하여 사용하는 SPM 뱅크의 수가 감소할수록 필요한 하드웨어 유닛의 수가 크게 감소하는 것을 볼 수 있다.

테이블 5.1: 슬레이브 모듈 수에 따른 AXI4 크로스바의 VIVADO 툴 합성 결과

Crossbar	LUTs	Registers	Muxes
16X16 Crossbar	10,737	3,841	2,574
16X12 Crossbar	7,470	3,182	2,583
16X8 Crossbar	5,730	2,510	1,849
16X4 Crossbar	3,486	1,838	1,102

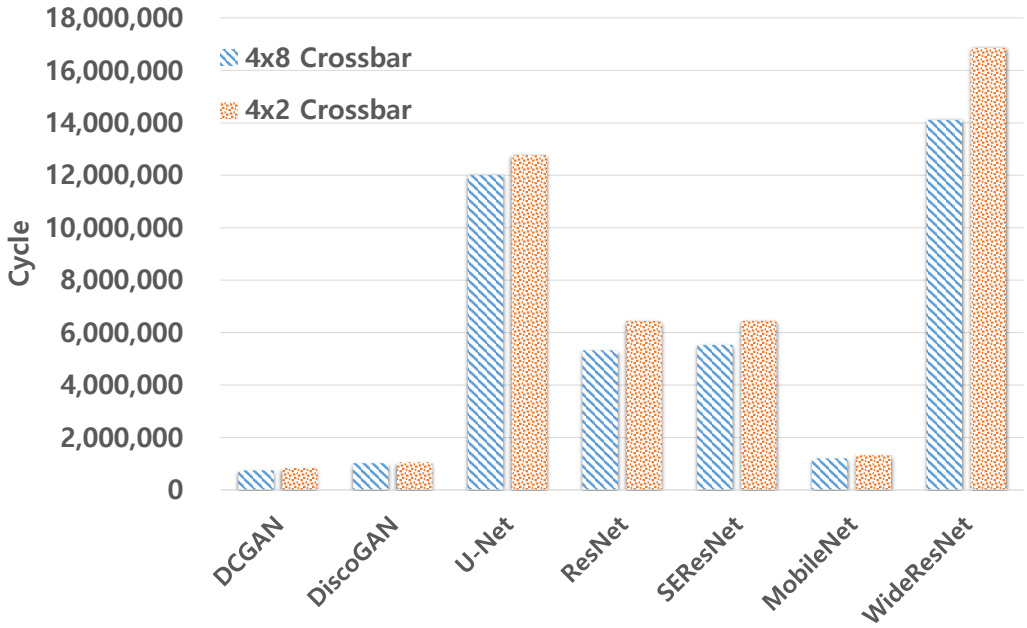

그림 5.1: 버스 토폴로지에 따른 오프 칩 메모리 접근 지연 시간 변화

그림 5.1은 SPM 뱅크를 공유하여 사용하는 것이 전체 시뮬레이션 사이클에 어느 정도 영향을 주는지 보여준다. 실험의 용이성을 위해 NPU와 SPM 뱅크의 수를 제한하여 실험을 진행하였다. 왼쪽의 막대 그래프는 4개의 NPU 코어가 8개의 SPM 뱅크와 연결되어 있고 각 코어는 전용 SPM 뱅크를 사용하기 때문에 SPM 뱅크 접근시 서로 다른 NPU 코어 간의 간섭이 발생하지 않는다. 오른쪽의 막대 그래프는 4개의 NPU 코어가 2개의 SPM 뱅크를 공유하고 있으며 SPM 뱅크 접근시 서로 NPU 코어 간의 간섭이 발생하는 환경이다. CNN에 모델에 따라서 차이는 있지만 약 4 %에서 21 %의 시뮬레이션 사이클이 증가하였다. 버스 토폴로지의 구성은 시스템의 성능 및 하드웨어 유닛 수에 영향을 주기 때문에 요구되는 하드웨어의 스펙에 맞게 적절한 구조의 버스 토폴로지는 정하는 것이 필요하다.

멀티 NPU 시스템 시뮬레이터는 다양한 환경을 고려하여 사용자가 시스템 설정을 통해 직접 버스 토폴로지를 구성할 수 있도록 만들어졌다. 시뮬레이터의 버스 구조는 그룹 단위의 완전 연결된 구조를 기본으로 하여 설정 가능하다. 사용자는 시스템 설정에서 사용할 그룹의 수와 각 그룹에 속한 NPU의 수 및 SPM 뱅크의 수를 결정한다.

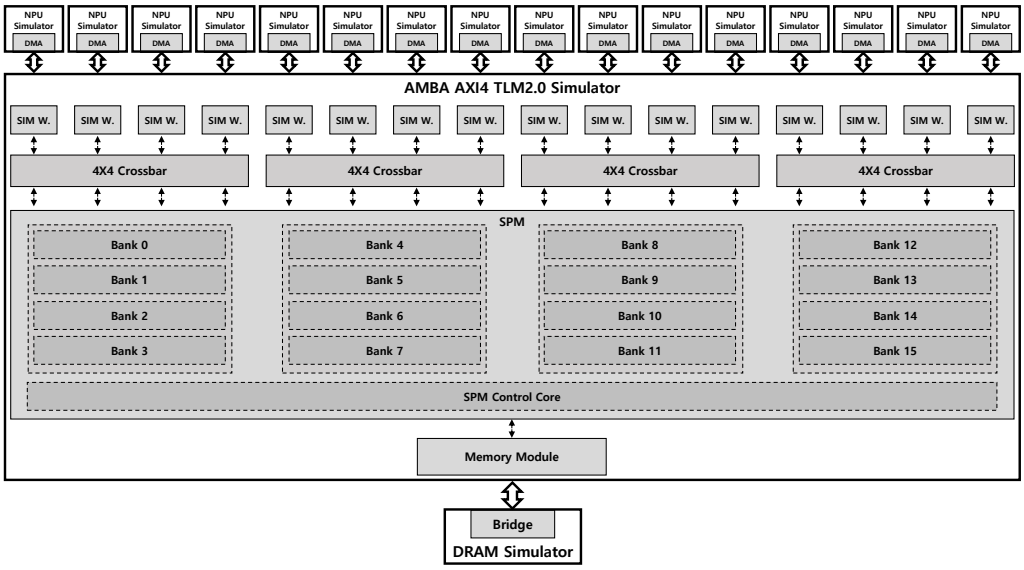

그림 5.2: 4X4 크로스바로 구성된 멀티 NPU 시스템 시뮬레이터

NPU는 동일 그룹의 모든 SPM 뱅크에 접근 가능하며 다른 그룹의 SPM 뱅크는 접근할 수 없다. 16X16 크로스바를 예로 들면, 1개의 그룹에 16개의 NPU와 16개의 SPM 뱅크를 설정하여 구성할 수 있다. 만약 하드웨어 유닛 수의 제약 또는 모든 NPU가 모든 SPM 뱅크에 접근할 필요가 없다면, 그림 5.2과 같이 구성하는 것도 가능하다. 4개의 그룹을 가지며 각 그룹은 4X4 크로스바를 통해 4개의 NPU와 4개의 SPM 뱅크가 완전 연결된다. 서로 다른 그룹에 속한 NPU 코어들은 SPM 뱅크 접근시 간섭이 발생하지 않으며 성능을 최대화할 수 있다. 또한, 상대적으로 적은 수의 모듈들로 연결되는 4X4 크로스바를 사용하기 때문에 전체 하드웨어 유닛의 수도 줄일 수 있다. 테이블 5.2는 16X16 크로스바 1개를 사용하여 모든 NPU 코어와 SPM 뱅크가 연결되도록 구성하였을 때와 4X4 크로스바 4개를 사용하여 각각 4개의 NPU 코어와 4개의 SPM 뱅크를

테이블 5.2: 16X16 크로스바 1개와 4X4 크로스바 4개의 VIVADO 툴 합성 결과 비교

Crossbar	LUTs	Registers	Muxes
16X16 Crossbar (1 ea)	10,737	3,841	2,574
4X4 Crossbar (4 ea)	3,747	3,240	672

연결되도록 구성하였을 때의 VIVADO [18] 툴 합성 결과를 보여주며 필요한 하드웨어 유닛의 수가 크게 감소하는 것을 확인할 수 있다.

멀티 NPU 시스템 시뮬레이터는 유연한 버스 구조의 설정이 가능하기 때문에 사용자는 목표로 하는 시스템에 적합한 버스 구조를 선택하여 타겟 CNN 모델을 시뮬레이션할 수 있다. 선택된 버스 구조에 맞게 CNN 모델을 NPU 코어에 맞게 최적화하여 매칭하는 것은 NPU 컴파일러의 몫으로 본 논문에서는 다루지 않는다.

5.2 실험 결과

본 섹션에서는 멀티 NPU 시스템에서 오프 칩 메모리 접근이 NPU 성능에 주는 영향을 두 가지 시나리오의 실험을 통해 확인하였다. 먼저 각 NPU 코어가 개별적으로

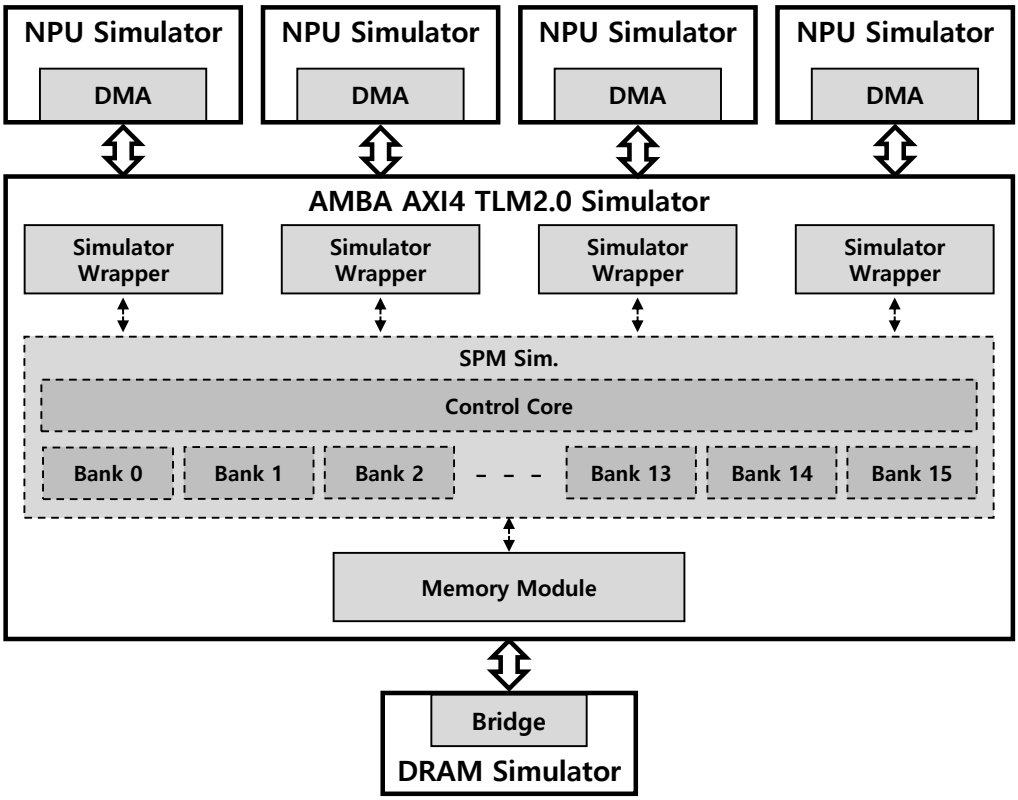

그림 5.3: 멀티 NPU 시스템 시뮬레이터 디자인

CNN 모델을 실행하는 것이다. 각 NPU 코어가 처리하는 CNN 모델의 데이터는 서로 다른 SPM 뱅크에 저장되기 때문에 동일한 SPM 뱅크에 접근하여 간섭이 발생할 일은 없다. CNN 모델은 NPU가 지원하는 어떠한 종류의 모델을 사용해도 상관 없으나 수행 종료 시간을 동일하게 하여 단일 NPU 코어 시스템과의 비교를 쉽게 하기 위해 모든 NPU 코어는 각각 동일한 CNN 모델을 수행하도록 하였다. 다음으로는 하나의 CNN 모델을 여러 NPU 코어가 나누어 수행하는 실험을 진행하였다. 효율적인 업무 분담을 위해 각 NPU 코어는 시스템 레벨의 파이프라이닝을 구성하는 하나의 모듈로써 동작하게 된다. 또한, NPU 코어 사이에 데이터를 공유하여 사용하기 때문에 SPM 뱅크 접근시 서로 간의 간섭이 발생할 수 있다. 두 실험 모두 NPU 시뮬레이터인 MIDAP [5]의 개발 진행 상황과 성능 비교의 원활함을 고려하여 그림 5.3와 같이 NPU의 수를 4개로 제한한 환경에서 진행되었다.

그림 5.4는 4개의 NPU 코어가 독립적으로 CNN 모델을 수행한 결과와 1개의 NPU 코어만을 사용했을 때와 비교한 결과이며, 모든 NPU 코어가 시뮬레이션을 끝냈을 때

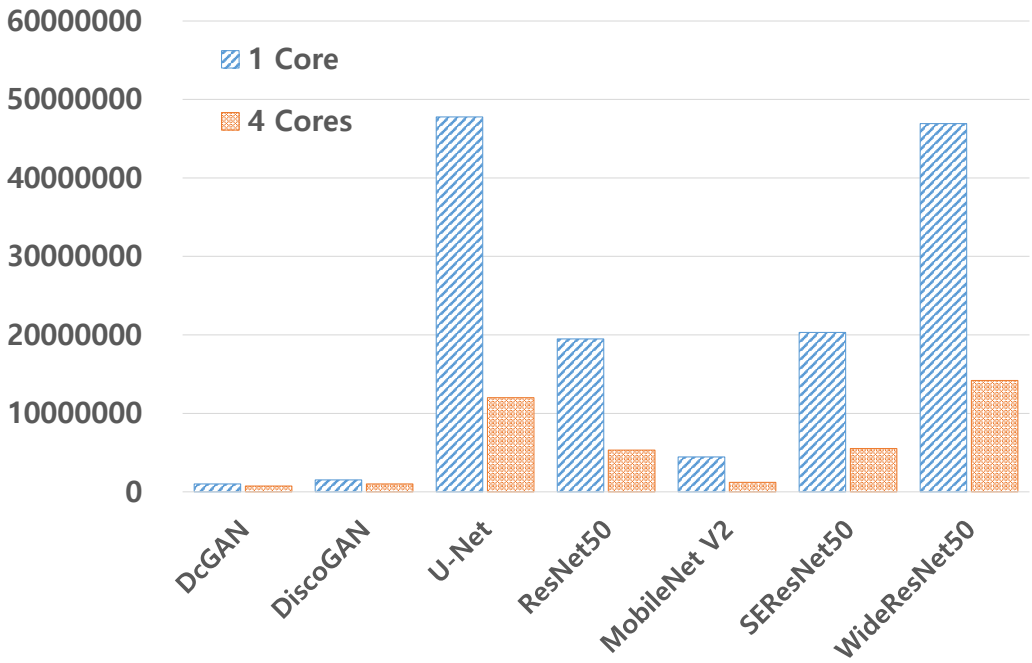

그림 5.4: 독립적인 CNN 모델을 수행시 시뮬레이션 시간 (Cycle)

의 시간 (Cycle)을 보여준다. 왼쪽의 막대 그래프는 1개의 CNN 응용을 1개의 NPU 코어에서 4회 반복수행하였을 때의 시뮬레이션 시간이다. 오른쪽의 막대 그래프는 4개의 NPU 코어가 동일한 CNN 응용을 각각 1회 수행한 결과이다. 당연한 결과이지만 모든 CNN 모델에서 시뮬레이션 사이클은 크게 감소하였고 특히 U-Net [9]의 경우 시뮬레이션 시간이 약 75 % 줄었다. 하지만 DcGAN [10]과 DiscoGAN [17]의 경우, 각각 약 26 %, 33 % 정도로 상대적으로 낮은 시뮬레이션 시간 감소를 보였다. NPU 코어의 수가 1개에서 4개로 증가하였음에도 불구하고 낮은 성능 향상을 보인 이유는 SPM 뱅크 접근에서의 간섭이 발생하지 않지만 오프 칩 DRAM의 데이터를 각 SPM 뱅크에 프리페치 (Pre-Fetch) 하는 과정에서 병목 현상 (Bottleneck)이 발생할 수 있기 때문이다. 특히, 두 CNN 모델은 오프 칩 메모리 접근 지연 시간이 전체 시뮬레이션 사이클 (Cycle)에서 차지하는 비중이 각각 73.1 %와 67.3 %로 매우 큰 모델들이기 때문에 오프 칩 DRAM에서의 병목 현상도 상대적으로 크게 나타나서 낮은 성능 향상을 보여주었다.

그림 5.5는 4개의 NPU 코어가 1개의 CNN 모델을 시스템 레벨의 파이프라이닝을 통해 수행한 결과이며, 세로 축은 모든 NPU 코어가 시뮬레이션을 끝냈을 때의 수행 사

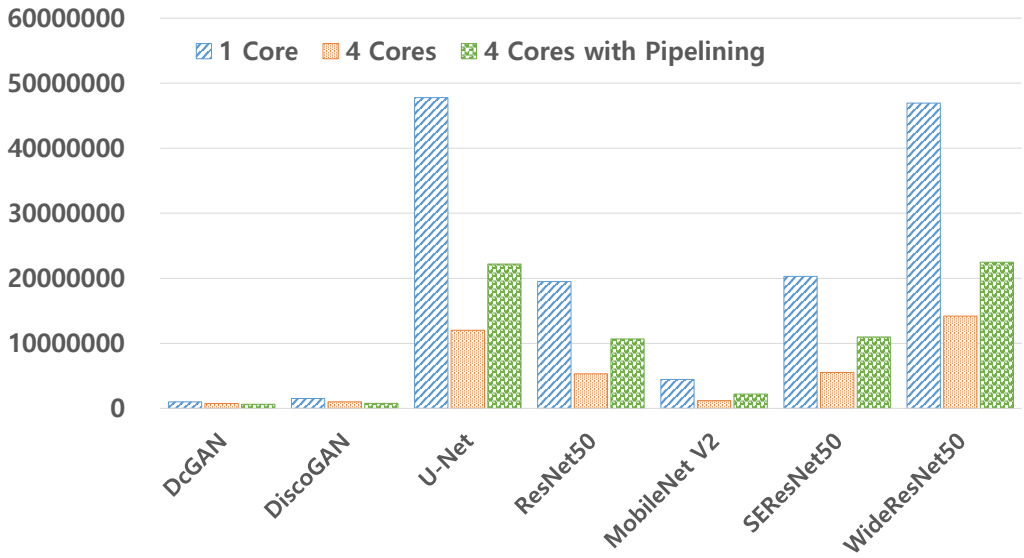

그림 5.5: 파이프라이닝으로 CNN 모델을 수행시 시뮬레이션 시간 (Cycle)

이클 (Cycle)을 보여준다. 왼쪽의 두 막대 그래프는 첫 번째 시나리오의 실험 결과이며, 가장 마지막의 막대 그래프는 4개의 NPU 코어가 시스템 레벨 파이프라인의 모듈로써 동작하며 각 CNN 응용을 4회 수행한 결과이다. 이전 실험의 결과와 비교하여 4개의 NPU 코어가 파이프라이닝하여 동작할 때 상대적으로 긴 시뮬레이션 사이클 (Cycle)을 보여주였다. 가장 높은 성능 향상을 보인 CNN 모델은 U-Net [9]으로 단일 NPU 시스템 대비 약 54 %의 시뮬레이션 사이클이 감소하였다. 하지만 이전 실험 결과에서 약 75 %의 시뮬레이션 사이클 감소를 보이는 것과 비교하면 낮은 성능을 보였다.

동일한 수의 NPU의 코어를 사용하였지만 시스템 레벨 파이프라이닝에서 낮은 성능 향상 폭을 보인 이유에는 크게 세 가지가 있다. 첫 번째 이유는 SPM 뱅크 접근 시 NPU 코어 사이에 간섭이 발생하기 때문이다. 실험에서는 4개의 코어가 같은 SPM 뱅크를 공유하여 사용하는데 각 코어가 동일 SPM 뱅크에 위치한 데이터에 접근하게 되면 불가피하게 접근 지연 시간이 발생하게 된다. 두 번째 이유는 NPU 코어 사이에 의존 (Dependency) 관계가 생기기 때문이다. 그림 5.6는 시스템 레벨의 파이프라이닝이 어떻게 동작하는지를 보여주며 동일한 색의 블록은 같은 이미지를 의미한다. 연속적인 여러 레이어들로 구성되는 CNN 모델을 4개의 NPU가 나누어서 처리하기 때문에 모든 NPU 코어는 의존 관계를 가지게 된다. 즉, 이전의 NPU 코어가 프로세싱을 끝내야만 다음 NPU 코어가 이어지는 레이어에 대한 동작을 수행할 수 있다. NPU 컴파일러의 최적화에 따라 차이는 있겠지만 NPU 코어 간 의존 관계로 인한 지연 시간은 불가피하게 발생할 수 밖에 없다. 마지막 이유는 파이프라이닝의 특성상 파이프라이닝의 시작 부분과 끝 부분은 모든 NPU 코어가 아닌 일부만 동작하기 때문이다. 실험에서는 CNN 모델을 4회 수행하였는데 실질적으로 4개의 NPU 코어가 모두 동작하는 시간은 길지

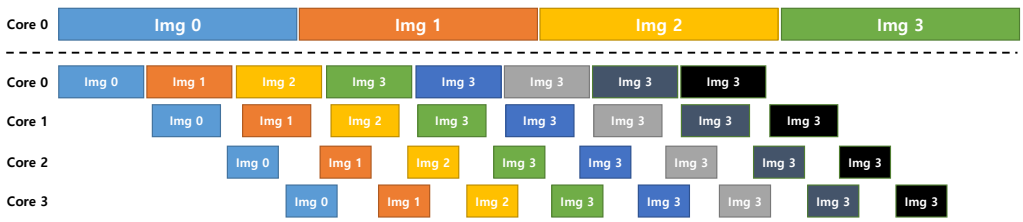

그림 5.6: 4개 NPU 코어에서의 파이프라이닝

않고 대부분의 시간은 1개에서 3개 사이의 NPU 코어만 동작하게 된다. 실험의 반복 수행 횟수를 증가시키면 각 NPU 코어의 사용률 (Utilization)이 증가하면서 더 높은 성능 향상을 보일 수 있을 것이라 예상되지만 본 논문에서 해당 실험을 포함하지 않았다.

두 시나리오의 실험 결과를 통해 멀티 NPU 시스템에서도 오프 칩 메모리 지연 시간이 성능에 큰 영향을 주고 있음을 확인할 수 있었다. 단일 NPU 시스템에서는 NPU의 데이터 요청 전에 오프 칩 DRAM으로부터 데이터를 미리 프리페치하여 오프 칩 DRAM 접근에 대한 지연 시간이 주는 영향이 크지 않았다. 하지만 멀티 NPU 시스템은 여러 NPU에서 필요한 DRAM 데이터가 동시에 프리페치될 수 있기 때문에 오프 칩 DRAM 접근으로 인한 병목 현상의 영향이 있었다. 또한, NPU 코어 사이에서의 의존 관계가 있을 경우 멀티 NPU가 비효율적으로 동작할 수 있었다. 일부 시나리오에 대한 결과만 확인하였지만 본 논문에서 구축한 멀티 NPU 시스템 시뮬레이터는 다양한 시나리오에서의 실험을 가능하게 하여 NPU 컴파일러의 개발 및 최적화를 쉽게 진행할 수 있도록 도움이 될 것이라 생각한다.

5.3 추후 연구

멀티 NPU로 확장된 시스템 시뮬레이터는 앞으로의 연구에 다양한 기회를 줄 것이라 생각한다. 1개의 NPU에서 1개의 CNN 모델을 수행하는 제한된 실험만이 가능했던 기존의 환경을 개선하여 더 다양한 조건의 시스템을 복합적으로 수행할 수 있는 환경을 제공하였다.

추후에는 멀티 NPU 시스템의 버스 구조 개선과 NPU 컴파일러의 최적화 두 부분으로 나누어 연구가 진행될 것이다. 먼저 멀티 NPU 시스템의 버스 구조 개선이 필요하다. 현재는 성능과 비용 모두 만족시키기 위해 그룹 단위의 완전 연결 가능한 크로스바 구조를 사용하였는데 그룹 단위의 제한된 접근만 가능하다는 단점을 가지고 있다. 멀티 NPU 시스템에서 수행 가능한 시나리오의 제한이 생기는 것이기 때문에 적은 비용으로 모든 NPU가 데이터를 공유할 수 있는 버스 구조에 대한 연구가 필요하다. 다음으로는 컴파일러 최적화가 필요하다. 앞서 진행한 실험은 멀티 NPU 시스템을 구성하는 것에 목적을 두었기 때문에 SPM 자원 관리에 대한 최적화가 거의 없었다. NPU

컴파일러 최적화를 통해 사용 가능한 자원을 효율적으로 분배함으로써 NPU 시스템의 성능을 향상시킬 수 있을거라 생각한다.

Chapter 6

결 론

본 논문에서는 오프 칩 DRAM 접근 지연 시간이 NPU 성능에 미치는 영향을 3 가지 DRAM 접근 모델 기반으로 다양한 실험과 함께 살펴보았다. 기존의 많은 연구에서 사용되었던 대역폭 모델 (Bandwidth Model)과 분석 모델 (Analytical Model)은 실시간으로 발생하는 다양한 외부 요인으로 인해 오프 칩 DRAM 접근 지연 시간을 정확하게 예측하지 못하였다. 시뮬레이션 모델 (Simulation Model)은 앞의 두 DRAM 접근 모델의 단점을 극복하기 위해 오프 칩 DRAM 접근 지연 시간에 영향을 주는 버스와 DRAM을 직접 시뮬레이션하는 모델이다. 시뮬레이션 모델을 위해 사이클 단위의 정확한 (Cycle Accurate) 시스템 시뮬레이션 환경을 구축하였고 시스템 시뮬레이터의 각 구성 요소는 오픈 소스 코드로 공개되어 많이 검증된 시뮬레이터들을 이용하였다. NPU 시뮬레이터로는 MIDAP [5], 버스 시뮬레이터로는 AMBA AXI4 TLM 2.0 시뮬레이터 [1], 그리고 DRAM 시뮬레이터로는 Ramulator [6]를 이용하였고 커스텀 SRAM 시뮬레이터로 Cache 시뮬레이터와 SPM 시뮬레이터를 직접 구현하여 다양한 옵션의 성능 측정이 가능하도록 하였다. 시뮬레이션 모델의 결과는 오프 칩 DRAM 접근 지연 시간이 NPU 성능에 매우 큰 영향을 주고 CNN 모델의 특성에 따라 결과가 다양하게 나타나는 것을 보여주었다.

NPU 성능에 영향을 주는 오프 칩 DRAM 접근 지연 시간을 줄이기 위해 본 논문에서는 NPU와 DRAM 사이에 큰 사이즈를 가지는 오프 칩 SRAM을 두는 메모리 계층 구조를 만들었고 오프 칩 SRAM이 NPU 성능에 주는 영향을 확인하였다. 오프

칩 SRAM으로써 Cache와 SPM을 사용하였고 다양한 실험으로 비교 분석한 결과 SPM이 Cache보다 NPU 시스템에서 훨씬 효과적임을 확인하였다. CNN 모델의 종류에 따라 성능 편차를 크게 나타났던 Cache와는 다르게 SPM은 실험에서 사용한 모든 CNN 모델에서 큰 성능 향상을 보여주었다.

마지막으로 멀티 NPU 시스템에서 오프 칩 메모리 접근이 NPU 성능에 주는 영향을 살펴보았다. 다양한 구성을 가진 AXI4 크로스바를 지원할 수 있도록 기존의 시스템 시뮬레이터를 확장하였다. 멀티 NPU 시스템의 성능을 최대화하기 위해서는 효율적인 자원 관리뿐만 아니라 각 NPU 코어 사이의 관계도 중요하기 때문에 무엇보다 NPU 컴파일러의 최적화가 필요함을 확인하였다.

Bibliography

[1] ARM Holdings plc. Amba tlm library developer guide, 2019.

[2] S. Yao et al. The evolution of deep learning accelerators upon the evolution of deep learning algorithms. In *HotChips*, 2018.

[3] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[4] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[5] Donghyun Kang, Jintaek Kang, Hyungdal Kwon, Hyunsik Park, and Soonhoi Ha. A novel convolutional neural network accelerator that enables fully-pipelined execution of layers. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 698–701, 2019.

[6] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.

[7] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 147, 2009.

[8] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[10] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[11] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim. In *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 21–25, 2019.

[12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

[17] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[18] Xilinx Company. Vivado design suite user guide, 2019.

Abstract

Analysis of the Effect of Off-chip Memory Access on the Performance of an NPU System

Lee Keonjoo
Department of Computer Science and Engineering
College of Engineering
The Graduate School
Seoul National University

Numerous CNN accelerators, called neural processing units (NPUs), have been proposed and developed recently to accelerate CNN computation with a customized chip. To minimize the DRAM access volume, NPUs commonly have a large on-chip memory and try to reuse the fetched data from the off-chip DRAM maximally. While extensive researches have been conducted to minimize the effect of off-chip DRAM access on the performance in the NPU design, little attention is paid to the detailed analysis of the DRAM access overhead and the use of memory hierarchy to minimize the off-chip DRAM access overhead. In this paper, I analyze the effects of DRAM access latency and the use of a large SRAM on the NPU performance based on a cycle-accurate system simulation environment. A cycle-accurate system simulation environment is built that consists of an NPU simulator, a Ramulator for DRAM simulation, a custom SRAM simulator based on CACTI, and the AMBA AXI4 TLM 2.0 simulator. Through extensive simulations with various CNN models, it is shown that using the SRAM as an SPM is more effective than using it as a cache. For the NPU simulator used for experiments, as an example, adding an SPM could reduce the memory access overhead by about 70.7%

in ResNet50 and improve the end-to-end performance by 26.5%. Finally, I extended the existing NPU system to a multi-NPU system that makes multiple NPUs shares the SPM and enables Design Space Exploration (DSE) through system setting under various conditions.

Keywords : NPU performance, DRAM access latency, Scratchpad memory, Multi NPU system

Student Number : 2020-23938