공학석사학위논문

매니코어 기반 뉴럴 프로세서의 빠른 성능 예측을 위한 하이브리드 시뮬레이션 기술

2019 년 2 월

서울대학교 대학원

공과대학 컴퓨터공학부

정 광 현

요약

뉴럴 프로세서의 성능을 예측하기 위하여 사이클 정확(Cycle-accurate)한 시뮬레이터를 이용하여 설계할 프로세서의 성능을 예측하면 시뮬레이션 시간이 오래 걸려 시간적 설계 비용이 매우 크다. 또한, 분석 모델을 통한 예측 또는 통계적인 방법을 사용한 예측은 정확도가 낮다. 본 논문에서는 샘플드(Sampled) 시뮬레이션과 분석 모델, 그리고 통계적 접근 방법을 결합한 하이브리드 시뮬레이션 기법을 매니코어 뉴럴 네트워크 가속기의 한 종류인 ZeNA 아키텍처에 적용하여 AlexNet 의 합성곱 레이어들에 대하여 기본 시뮬레이터 대비 1.15%의 오차로 최대 91 배 가속을 하였다. 또한, 제안한 성능 예측 방법이 설계 공간 탐색의 도구로써 유효성을 검증하기 위해, 하드웨어 구성을 변경하며 실험을 수행했으며 기존 시뮬레이터와의 경향성이 일치하는 것을 확인하였다.

주요어: 프로세서 성능 예측, 샘플드 시뮬레이션, 분석 모델, 통계 모델

학 번: 2016-24657

목차

그림 목차

표 목차

1. 서론

최근 영상처리, 음성인식 등 다양한 영역에서 뉴럴 네트워크 사용이 매우 증가하고 있다. 이러한 뉴럴 네트워크 응용은 높은 계산 복잡도를 갖기 때문에 임베디드향 기기에서 수행하기 위해 연산을 빠르게 처리할 수 있는 가속기가 필수적이다.

매니코어 가속기를 설계할 때, 최적의 설계 공간(메모리 크기, 버스 대역폭 등)을 탐색하는 과정이 필요하다. 실제 하드웨어가 나오기 전에는 이러한 검증 과정을 거칠 수 없기 때문에 일반적으로 시뮬레이션을 통해 성능을 검증한다. 하지만 매니코어 프로세서의 모든 동작을 반영하는 사이클 정확(Cycle-accurate)한 시뮬레이터를 사용하면 1분 정도의 동작을 시뮬레이션하는데 수십 일이 걸리기 때문에 시뮬레이션을 통한 설계 공간 탐색이 비효율적이다 [1]. 예를 들어, 대표적인 사이클 정확한 GPGPU 시뮬레이터인 GPGPU-sim[2]을 이용하여 뉴럴 네트워크 응용을 수행할 경우, 네트워크의 크기에 따라 길게는 2일 12시간, 짧게는 8시간이 걸린다. [표 1]은 GPGPU-sim에서 각기 다른 뉴럴 네트워크 응용을 수행할 때 걸리는 수행 시간을 나타낸다[3].

또한, 빠른 성능 예측을 위해 가속기 구조와 응용에 대한 정적인 정보만을 이용하여 성능을 예측하면 가속기의 동적인 동작이 반영되지 않아 성능 예측의 정확도가 낮아진다.

본 논문에서는 뉴럴 네트워크 가속기의 동적인 동작을 반영한 빠른 성능 예측을 위해 샘플드(Sampled) 시뮬레이션과 분석 모델(Analytical Model), 그리고 통계적(Statistical) 모델을 결합한 형태의 하이브리드(Hybrid)

시뮬레이션 기법 이용한다. 타겟하는 가속기는 액티베이션(Activation) 또는 가중치(Weight)가 0인 곱셈은 건너뛰며 연산하는 ZeNA[4]로 하였고, 네트워크 응용은 Alexnet[5]의 합성곱 레이어(Convolutional layer)을 수행하여 검증한다. 또한, 기존 시뮬레이터와 하이브리드 시뮬레이터를 하드웨어 설정을 변경하며 실험하여 경향성 비교를 통해 설계 공간 탐색의 도구로써 본 논문에서 제안한 성능 예측 기법의 유효성을 확인한다.

네트워크	TinyYolo	Resnet50	Resnet152	Densenet
수행시간	8Hour 5Min	2Day 12Hour16Min	2Day 32Min	23Hour 23Min

[표 1] **GPGPU-sim 에서 뉴럴 네트워크 응용을 수행할 때의 수행시간[3]**

2. 관련 연구

프로세서의 성능 예측을 위한 기존의 방법은 분석모델, 통계모델, 시뮬레이션 기반으로 크게 3가지로 구분될 수 있다.

분석 모델을 통한 예측은 하드웨어 아키텍처에 대한 정보 및 수행시킬 응용에 대한 분석 정보를 바탕으로 수학적 모델을 설계하고 이를 이용하여 성능을 예측한다. 타겟 프로세서에 동적인 동작이 없을 경우, 빠르고 정확하게 성능을 예측할 수 있지만, 동적인 동작이 존재할 경우, 이러한 동작이 성능에 미치는 영향력을 모델하기 어려운 점이 있다.

통계적 모델은 응용을 프로세서에서 수행시킬 때 얻을 수 있는 다양한 프로파일 정보를 바탕으로 통계적인 모델을 세워 예측한다. 대표적인 방법은 회귀분석이 있다[6].

시뮬레이션 기반 성능 예측은 프로세서를 이루는 모든 구성요소를 소프트웨어로 모델링하는 방법이다. 응용이 타겟 프로세서에서 수행될 때의 하드웨어 동작을 사이클 단위로 모사하기 때문에 정확하지만 성능 예측을 위한 시간적 비용이 크다.

하이브리드 시뮬레이션 기법[3]은 타겟 아키텍처의 성능에 영향을 미치는 변수들의 종류와 성능과의 관계를 분석 모델을 통해 모델링한다. 이때 분석 모델의 입력 변수로는 타겟 아키텍처 및 응용의 정적인 정보와 응용을 수행하며 변화하는 동적인 정보가 모두 포함된다. 정적인 계산을 통해 얻을 수 없는 동적인 정보는 샘플드 시뮬레이션을 통해 얻어질 수 있다. 샘플드 시뮬레이션은 타겟 아키텍처의 구성요소 중 성능에 가장 큰

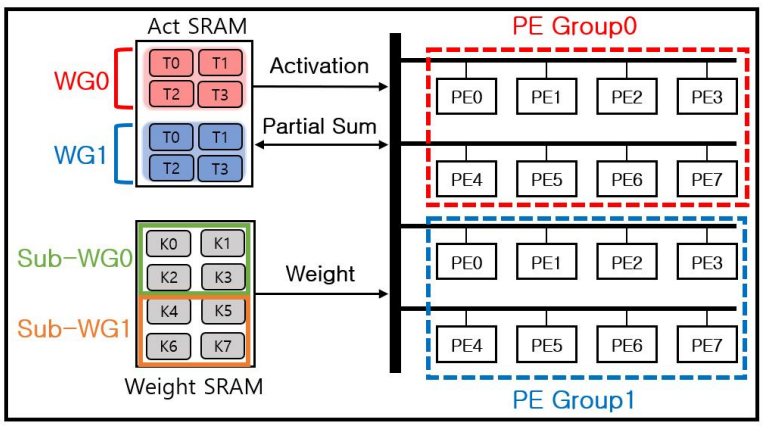
영향을 미치는 구성 요소만을 시뮬레이션함으로써 시뮬레이션의 시간을 단축한다.

본 논문에서는 빠른 예측을 위한 분석 모델을 기반으로 예측의 정확도를 올리기 위하여 샘플드 시뮬레이션을 적용하였고, 이 과정에서 유실되는 동적인 정보는 통계적 모델을 이용하여 예측하는 하이브리드 시뮬레이션 기법을 매니코어 기반 뉴럴 프로세서인 ZeNA에 적용하여 빠른 성능 예측 방법의 효과와 유효성을 검증하였다.

3. 타겟 가속기 구조

[그림 1]은 본 논문에서 하이브리드 시뮬레이션 기법을 적용할 ZeNA 아키텍처의 구조를 나타낸다. ZeNA 는 액티베이션과 부분합을 저장하기 위한 Act SRAM, 가중치를 저장하기 위한 Weight SRAM, SRAM 과 PE 들 사이의 데이터를 주고받기 위한 버스, 그리고 연산이 이루어지는 PE(Processing Element)들로 구성된다.

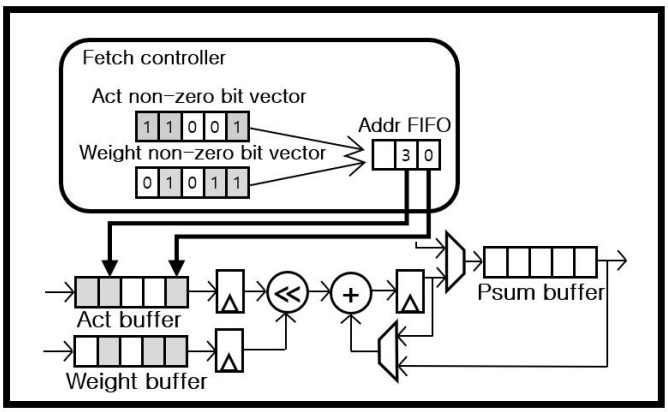
ZeNA 는 다수의 PE 가 하나의 PE 행(Row)를 구성하며, 다수의 PE 행이 하나의 PG(PE Group)를 이룬다. [그림 1]을 예로 들면, 한 행이 4 개의 PE 로 구성되고 2 개의 행이 하나의 PG 를 이룬다. 해당 시스템은 총 16 개의 PE 가 두 개의 PG 를 이룬다.



[그림 1] **ZeNA** 가속기 구조

[그림 2]는 ZeNA 의 PE 내부 구조를 나타낸다. PE 내부에는 액티베이션 버퍼(Act buffer), 가중치 버퍼(Weight buffer), 부분합 버퍼(Psum buffer)를

가지고 있어서 액티베이션 버퍼에는 입력 특징 지도(Feature map), 가중치 버퍼에는 해당 PE 가 담당하는 가중치(Filter Weight), 그리고 부분합 버퍼에는 합성곱 연산의 결과가 저장된다. 또한, 연산할 값이 0 이면 연산을 건너뛰기 위해 액티베이션 및 가중치 버퍼의 데이터가 0 인지 아닌지를 나타내는 비트 벡터(Bit vector)가 존재한다. 이를 이용해 액티베이션과 가중치가 둘 다 0 이 아닌 값의 인덱스(Index)를 판단하고 해당 인덱스의 데이터만을 연산한다.



[그림 2] **PE** 내부 구조

ZeNA 는 각각의 PG 에 액티베이션을 WG(Work Group)라는 단위로 할당하여 연산을 진행한다. [그림 1]을 예로 들면, PG0 와 PG1 에 각각 WG0 와 WG1 이 할당된 시스템을 나타낸다. 액티베이션은 다시 타일(Tile) 단위로 나누어져 버스를 통해 순차적으로 PG 에 전송되고 PG 내에서 타일은 PE 에 브로드케스트(Broadcast)된다. 액티베이션이 SRAM 에서 PG 로 전송될 때, ZeNA 는 버스 대역폭(Bandwidth)을 높이기 위해 합성곱 연산의 필터가 이동하는 간격(Stride)을 고려하여 다음 합성곱 연산에

사용할 액티베이션 중 이전 액티베이션과 중복되지 않는 크기의 데이터만을 전송한다.

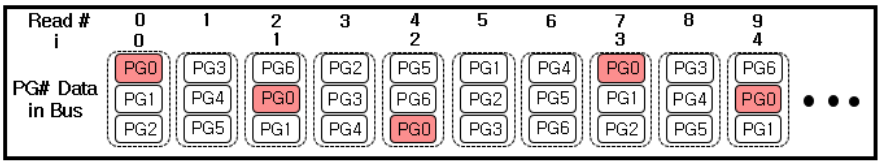
가중치의 경우, Sub-WG 로 나누어져 순차적으로 PE 에 할당된다. 하나의 Sub-WG 를 이루는 가중치 타일의 개수는 하나의 PG 를 이루는 PE 개수와 같다. Sub-WG 내의 각각의 가중치 타일은 PG 내의 PE 인덱스에 맞게 할당된다. 예를 들어, [그림 1]에서 가중치 K0 는 PG0 와 PG1 의 PE0 에 할당되고, K1 은 PE1 에 할당된다. Sub-WG0 에 대한 연산을 완료한 후에 Sub-WG1 에 대한 연산을 수행하며 합성곱 레이어에 대한 부분합을 누적한다.

액티베이션 읽기의 경우, 버스의 대역폭이 허용하는 만큼 복수의 타일을 라운드로빈 순서로 여러 PG 로 동시에 전송하며, 부분합 쓰기의 경우, 한 번에 한 PG 만 버스를 점유하여 SRAM 에 쓰인다.

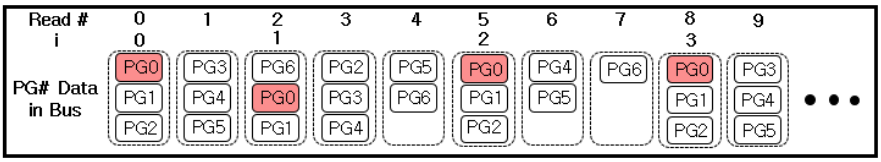
ZeNA 는 연산할 액티베이션 타일을 SRAM 으로부터 읽어와 PE 의 액티베이션 버퍼에 저장한 뒤 연산을 진행한다. 연산의 결과는 PE 의 부분합 버퍼에 저장된다. 이때 연산 과정에서 ZeNA 는 계산할 액티베이션 또는 가중치가 0 일 경우 계산하지 않는다. 따라서 PG 에 할당된 액티베이션과 가중치의 0 비율에 따라 연산이 끝나는 순서가 달라진다. 이로 인해 할당받은 액티베이션과 가중치의 0 의 비율에 따라 PG 사이의 동적인 동작이 발생한다. 연산할 데이터에 0 의 비율이 높아 액티베이션 버퍼에 타일이 없을 경우, 해당 PG 의 읽기 우선순위가 높아진다. 또한, 연산량이 많은 PG 는 액티베이션 버퍼가 가득 차 더 이상 액티베이션 타일을 받지 못하는 경우가 생긴다. 이러한 동적인 동작으로 ZeNA 에서 버스 이용률은 매 버스 사이클 마다 달라진다. 이상적인 경우 모든 PG 에 0 의 비율이 높아 매 버스 사이클에 모든 PG 가 액티베이션 타일을 읽을

수 있으면 버스 이용률은 100%가 되고, 그렇지 않은 경우, 액티베이션 타일을 받을 수 있는 PG 로만 전송되어 버스의 비트 폭(Bit-width)을 다 채우지 못하게 된다.

[그림 3]과 [그림 4]는 PG 가 7 개이고 한 번에 3 개의 타일을 전송할 수 있는 비트 폭을 가진 ZeNA 에서 타일 읽기 양상을 나타낸다. [그림 3]은 모든 PG 가 라운드 로빈 순서에 의해 타일을 받아 버스 이용률이 100%인 경우이다. [그림 4]는 연산할 데이터가 많아 타일을 전송하는 속도보다 연산 속도가 느려서 액티베이션 버퍼가 꽉 차게 되어 버스를 100% 활용하지 못하는 경우이다.



[그림 3] 이상적인 경우의 타일 읽기 양상



[그림 4] 타일 내 연산량으로 버퍼가 가득 차 타이밍이 밀리는 경우

4. 제안하는 성능 예측 방법

본 논문에서는 매니코어 기반 뉴럴 프로세서인 ZeNA 의 빠른 성능 예측을 위해 분석 모델, 통계적 모델, 샘플드 시뮬레이션을 결합한 하이브리드 시뮬레이션 기법을 이용한다. 최종적인 하이브리드 시뮬레이션 기법을 적용하기 위해 기본 분석 모델을 시작으로 샘플드 시뮬레이션을 적용, 그리고 통계적 모델을 포함하여 분석 모델을 확장한다.

4.1. 기본 분석 모델

ZeNA 아키텍처의 성능은 [그림 3]과 같이 버스의 활용률이 100%인 상황에서 식 (1) 과 같이 모델링 될 수 있다. 모든 연산이 통신과 병렬적으로 이루어져 통신이 성능의 병목이 된 경우의 모델이다. 즉, 모든 PG 에서 라운드 로빈 순서에 의해 타일을 읽을 수 있는 상황에 액티베이션 버퍼에 타일을 받을 만큼의 공간이 남아있어 매 버스 사이클마다 버스 비트 폭을 가득 채워 보내는 상황이다.

$$Cycle = (\#Read + \#Write) \times Rate_{Freq} \quad - \quad (1)$$

이때 *#Read* 는 버스 활용률이 100%인 상황에서 버스의 읽기 사이클 수이다. ZeNA 가 동작하기 위해 기본으로 필요한 읽기 횟수를 나타낸다. *#Write* 는 연산의 결과인 부분합을 SRAM 에 쓰는 버스 사이클 수를 나타낸다. ZeNA 는 쓰기 연산에서 하나의 PG 만 버스를 독점하고 부분합을 SRAM 에 보내기 때문에 *#Write* 는 전체 타일의 개수와 같다. *#Read* 와 *#Write* 는 데이터를 PG 에 전송하는 데 필요한 기본적인 버스 동작 횟수이므로 정적으로 분석할 수 있다.

$Rate_{Freq}$ 는 PE 동작주파수와 버스 동작주파수의 비율을 나타낸다. 읽기/쓰기는 버스 사이클마다 일어나지만, 성능은 PE 사이클 기준으로 측정되기 때문에 PE 와 버스의 동작주파수의 비율을 곱해주어야 한다.

$$Rate_{Freq} = \frac{Freq_{PE}}{Freq_{Bus}} \quad - \quad (2)$$

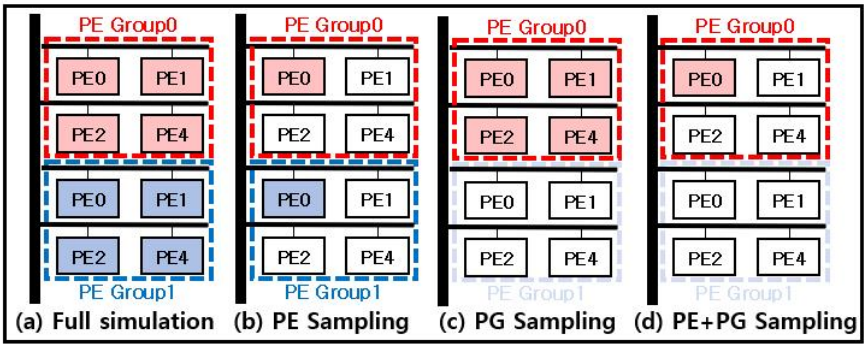
식 (1)은 통신의 횟수로만 성능을 측정하였기 때문에 [그림 4]처럼 유효 연산이 많아지는 경우, 연산에 의한 추가적인 지연(Delay)이 성능에 미치는 영향을 고려하지 않는다. 이러한 지연은 응용을 수행할 때 시간에 따라 달라지는 동적인 정보이기 때문에 시뮬레이션을 통해 얻어야 한다. 4.2 절에서는 샘플드 시뮬레이션을 통해 성능에 영향을 미치는 연산 지연을 측정하고, 이를 분석 모델에 적용하여 성능을 예측한다.

4.2. 샘플드 시뮬레이션을 적용한 분석 모델 확장

4.1 에서 사용한 기본 분석 모델은 0 의 비율이 높아 버스 사이클마다 버스 활용률이 100%인 경우에 동적인 동작이 발생하지 않기 때문에 0 의 비율이 높은 경우에 대하여 읽기 및 쓰기 횟수만 분석 모델로 모델링하여도 오차가 매우 적다. 하지만 0 의 비율이 낮아 연산 시간이 통신 시간보다 커지게 된다면 연산에 의해 통신이 지연되고 식 (1)은 이러한 지연을 결과에 반영할 수 없다. 따라서, 샘플드 시뮬레이션을 통해 연산에 의한 지연 정보를 추출한다.

샘플드 시뮬레이션은 분석 모델에 필요한 변수를 추출하기 위하여 성능에 영향을 미치는 구성 요소만 샘플링하여 시뮬레이션하는 기법이다. 본

논문은 PE 및 PG 단위로 샘플링을 적용한다. [그림 5]는 샘플드 시뮬레이션이 적용된 ZeNA 를 나타낸다.



[그림 5] **ZeNA** 의 샘플드 시뮬레이션

4.2.1 PE 샘플드 시뮬레이션

PE 샘플드 시뮬레이션은 PG 를 구성하는 다수의 PE 중 몇 개의 PE 만을 시뮬레이션하는 방법이다. [그림 5-(b)] PE 샘플링은 4 개의 PE 로 구성된 PG 에서 하나의 PE 만을 시뮬레이션하는 예를 나타낸다. PG 의 읽기와 쓰기는 PG 를 구성하는 PE 들에 동기화되어 일어난다. 따라서 PG 내에서 가장 연산량이 많은 PE 의 상태에 따라 읽기와 쓰기 시점이 결정된다. 그러므로 PE 샘플드 시뮬레이션을 할 때, 가장 연산량이 많은 PE 를 선택하는 것이 중요하다. 본 논문에서는 할당받은 가중치에 0 이 아닌 값의 개수가 가장 많은 PE 를 선택하였다. 하지만 액티베이션과 연산이 될 때, 액티베이션의 0 비율에 따라 실제로 연산량이 가장 많은 PE 는 가중치에서 0 이 아닌 값의 개수가 가장 많은 PE 가 아닐 수 있다. 따라서 가중치에 0 이 아닌 값의 개수가 가장 많은 PE 순으로 샘플링 되는 PE 의 개수를 증가시켜 시뮬레이션하면 성능 예측의 정확도는 올라간다. 본 논문의 5 장

실험에서는 가중치에 0 이 아닌 값이 많은 순으로 5 개, 3 개, 1 개의 PE 를 샘플링하여 PE 샘플드 시뮬레이션의 정확도를 실험하였다. 이 경우 정확도와 속도 간의 트레이드오프(Trade off)가 발생한다.

4.2.2 PG 샘플드 시뮬레이션

PG 샘플드 시뮬레이션은 ZeNA 를 구성하는 다수의 PG 중 하나의 PG 만을 시뮬레이션하는 방법이다. [그림 5-(c)]의 PG 샘플링은 2 개의 PG 중 하나의 PG 만을 시뮬레이션하는 예를 나타낸다.

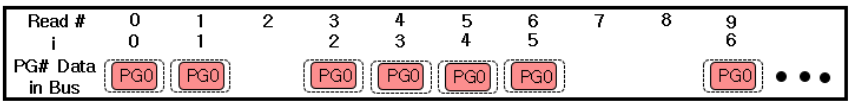
PG 샘플드 시뮬레이션을 적용할 때 PG 를 선택하는 기준은 할당받은 WG(액티베이션)에서 0 이 아닌 값이 가장 많은 PG 로 한다. 연산이 병목이 되는 경우에서 시스템의 최종 성능은 가장 연산량이 많은 PG 에 의해 결정되기 때문이다.

PG 샘플링을 하면 [그림 6]와 같이 시뮬레이션 되지 않는 PG 들의 읽기/쓰기가 반영되지 않기 때문에 시뮬레이션 시 이를 고려해야 한다. 버스의 활용률이 100%인 이상적인 경우에서 다른 PG 가 버스를 점유하여 기다려야 하는 라운드로빈 간격을 *RRWait(Round Robin Wait)*으로 나타내었다([그림 7]). 예를 들어, [그림 7]에서 PG0 는 0 번째 버스 사이클(*Read 0*)에서 타일을 읽은 후, 라운드 로빈 방식으로 순서를 기다린 후 2 번째 버스 사이클(*Read 2*)에서 타일을 읽을 수 있다. *RRWait* 은 식 (4)를 통해 버스가 한 번에 전송할 수 있는 타일의 개수와 PG 의 개수를 통해 계산할 수 있으므로 이를 반영하여 시뮬레이션한다. *RRWait* 은 PG 가 타일을 읽은 후 몇 번의 버스 사이클 후에 타일을 읽을 수 있는지를

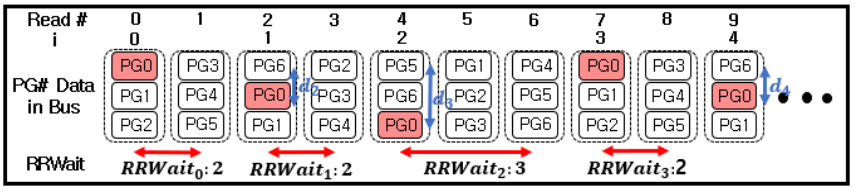
나타내므로 타일을 읽을 때마다(*i*) 계산한다. *RRWait* 의 합은 식 (1)의 이상적인 경우에서의 버스의 읽기 사이클 수와 같다.

$$\#Read = \sum_{i} RRWait_i \qquad - \quad (3)$$

#Read 는 전체 액티베이션 타일의 개수(*#TotalTile*)와 버스가 한 번에 전달할 수 있는 타일의 개수(*#TileInBus*) 등을 이용하여 정적으로 구할 수 있지만 식(3)을 통해서도 계산할 수 있다. 본 논문에서는 시뮬레이션을 진행하며 *RRWait* 이 계산될 때 *RRWait* 의 값을 누적하며 *#Read* 를 계산하였다.



[그림 6] PG 샘플링을 할 경우 타일 읽기 양상



[그림 7] RRwait (Round Robin Wait)

식 (4)는 *RRWait* 을 나타낸다.

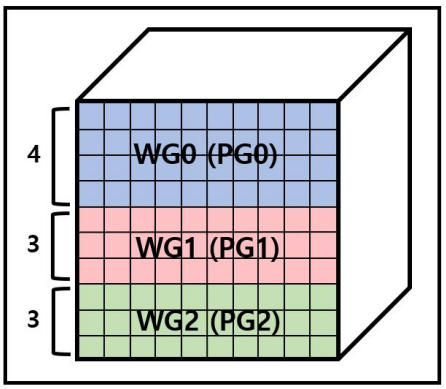
$$RRWait = \left\lfloor \frac{d_i + \#PG_i}{\#TileInBus} \right\rfloor \qquad - \quad (4)$$

이때 버스가 한 번에 전송할 수 있는 타일의 개수(*#TileInBus*)와 PG 의 개수가 정수배가 되지 않는 경우, *#RRWait* 은 일정하지 않고, [그림 7]과

같이 (2, 2, 3)의 패턴으로 나타나게 된다. 이를 고려해주기 위한 변수는 d_i이며 식 (5) 를 통해 계산할 수 있다.

$$d_i = \{(\#PG\%\#TileInBus) \times i\}\%\#TileInBus \quad - \quad (5)$$

또한, WG 내의 타일 개수가 다르기 때문에 시간이 흐름에 따라 할당받은 WG 내의 타일을 모두 처리한 PG 가 생길 수 있다. [그림 8]을 예로 들면, 특징 지도의 높이가 10 이고 3 개의 WG 으로 나누질 경우, 각각의 WG 의 행에 할당된 타일의 개수(*#TilePerRow*)는 WG0 는 4 개이고 WG1 과 WG2 는 3 개이다.



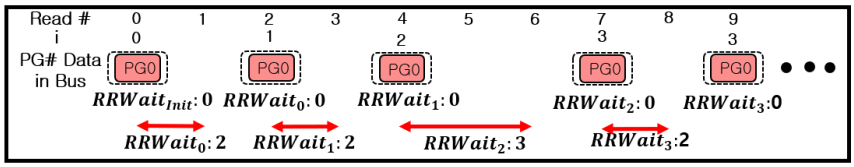
[그림 8] 높이 10 의 특징 지도가 3 개의 WG 으로 나뉘어 PG 에 할당된 모습

즉, 특징 지도의 높이가 PG 의 개수의 배수가 아닌 경우, PG 별 연산 총량이 달라진다. 이런 경우 적은 타일을 할당 받은 PG 들은 먼저 연산을 끝마칠 것이고, 더 많은 타일 할당 받은 PG 만이 남아 연산을 하게 될 것이다. 시뮬레이션 중 이러한 PG 의 개수 변화를 반영한 것이 $\#PG_i$ 이며 식 (6)으로 나타낼 수 있다.

$$\#PG_i = \begin{cases} \#\ TotalPG\ (\left\lfloor \frac{FMHeight}{\#TotalPG} \right\rfloor \times TilePerRow < i) \\ FMHeight\%TotalPG(\left\lfloor \frac{FMHeight}{\#TotalPG} \right\rfloor \times TilePerRow \geq i) \end{cases} \quad - \quad (6)$$

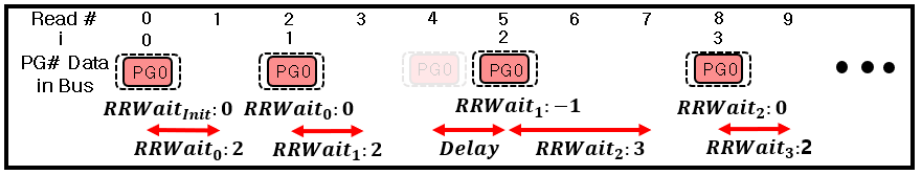
샘플링 된 PG 가 타일을 읽은 시점에서 *RRWait* 이 계산되면 *RRWait* 은 버스 사이클마다 1 씩 감소한다. *RRWait* 값이 0 이 되면 다른 PG 들에 읽기가 반영된 것으로 판단하고 다음 타일을 읽는다.

[그림 9]는 PG 샘플드 시뮬레이션에서 *RRWait* 을 적용하여 시뮬레이션했을 때의 타일 읽기 양상을 나타낸다. [그림 9]를 통해 알 수 있듯이 *RRWait* 을 적용한 PG0 의 타일 읽기 양상은 [그림 3]의 이상적인 경우에서 PG0 의 타일 읽기 양상과 동일한 것을 확인할 수 있다.



[그림 9] PG 샘플링에서 *RRWait* 을 반영한 타일 읽기 양상(이상적인 경우)

만약 샘플링되어 시뮬레이션 되는 PG 내 PE 에서 연산량이 많아 액티베이션 버퍼가 가득 차게 되어 타일을 읽을 수 없게 되면 *RRWait* 은 음수가 된다. *RRWait* 이 음수가 되는 것은 연산에 의해 통신이 지연되는 것을 나타낸다. [그림 10]은 연산에 의해 통신이 지연된 상황을 나타낸다. [그림 10]은 액티베이션 버퍼에 들어갈 수 있는 최대 타일의 수가 2 개인 시스템을 가정한다. 타일 0(i=0)과 타일 1(i=1)에 0 이 아닌 값이 많아 액티베이션 버퍼에 더 이상 타일을 읽을 수 없게 되어 4 번째 타일 읽기(*Read* 4)에서 읽어야 할 타일 2(i=2)가 지연되어 5 번째 타일 읽기(*Read* 5)에서 읽은 상황이다. 이때 $RRWait_1$ 의 값이 음수가 된다.

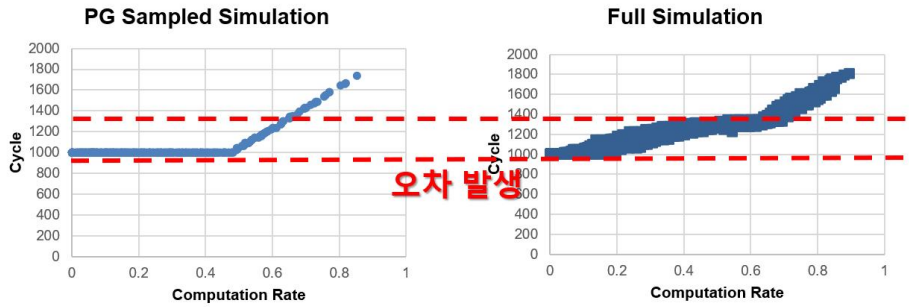


[그림 10] **PG 샘플링에서 연산에 의해 통신 지연이 발생된 타일 읽기 양상**

[그림 5-(d)]는 PE 와 PG 샘플링을 함께 적용하여 하나의 PE 만 시뮬레이션하는 상황이다. 이때 시뮬레이션을 위해 선택된 PE 는 실제 유효한 연산이 가장 많을 것으로 예측되는 PE 이다. 따라서 해당 PE 에서 생기는 연산 지연이 가장 크며 최종 성능에 가장 큰 영향을 끼칠 것으로 판단되는 PE 이다. 시뮬레이션을 통해 해당 PE 에서 생기는 연산 지연(*#ReadDelay*)이 측정된다. *#ReadDelay* 는 *RRWait* 을 통해 버스 사이클 기준으로 측정된다. 버스 사이클마다 *#ReadDelay* 는 *RRWait* 이 음수인 경우 1 씩 증가한다. *#ReadDelay* 는 버스 사이클에서 읽기가 지연되는 것을 나타내므로 해당 버스 사이클의 부분합 버퍼에 데이터가 존재할 경우 쓰기가 일어날 수 있다. 이를 반영하여, 식 (1)은 식 (7)와 같이 확장될 수 있다.

$$Cycle = (\#Read + Max(\#ReadDelay, \#Write)) \times Freq_{rate} \quad - \quad (7)$$

식(7)을 이용하여 PG 샘플드 시뮬레이션을 하면 [그림 11]과 같이 오차가 발생한다. 오차의 원인은 시뮬레이션 되지 않은 PG 에서 발생하는 추가적인 버스의 읽기 사이클 수이다. 이는 PG 샘플드 시뮬레이션으로 알 수 없으므로 통계적 모델을 통해 예측한다.



[그림 11] 식(7)을 이용한 PG 샘플드 시뮬레이션 결과와 전체 시뮬레이션 결과의 비교

4.3. 통계적 모델을 이용한 분석 모델 확장

식 (7)은 하나의 PG 만을 시뮬레이션 할 때의 모델이다. 하지만 실제 시스템은 여러 PG 에서 연산이 일어나고 다른 PG 에서도 연산에 의한 지연이 발생하면 읽기가 지연된다. 여러 PG 에서 동시에 지연이 일어나면 데이터를 전송 받을 PG 가 부족해져 버스 활용률이 떨어지며([그림 3]), *#ReadDelay* 에 다른 PG 의 읽기가 발생할 수 있다. 때문에 다른 PG 에 의한 추가적인 읽기가 필요하다. 이러한 추가적인 읽기 횟수를 *#Addit.Read* 라고 할 때 이를 적용한 수식은 식 (8)과 같다.

$$Cycle = (\#Read + Max(\#ReadDelay, \#Addit.Read + \#Write)) \times Freq_{rate} \quad - \quad (8)$$

#Addit.Read 는 *#Write* 과 마찬가지로 시뮬레이션 되는 PG 의 읽기 지연 시간에 일어날 수 있으므로 *Max(#ReadDelay, #Addit.Read+#Write)*로 나타낼 수 있다.

#Addit.Read 는 복수의 PG 에 의한 영향이므로 PG 샘플드 시뮬레이션을 통해서 얻을 수 없다. 따라서 통계적 모델을 이용하여 *#Addit.Read* 를

예측하였다. *#Addit.Read* 는 전체 타일 개수(*#TotalTile*)에 대한 비율로 발생하므로 식(9)와 같이 통계적 모델을 세우고 비율($Rate_{Addit.Read}$)을 회귀분석을 통해 예측하였다.

$$\#Addit.Read = \#TotalTile \times Rate_{Addit.Read} \quad - \quad (9)$$

회기 분석 모델을 세우기 위해 시스템에서 *#Addit.Read* 에 영향을 미치는 요소를 파악하고 파악한 요소에 대하여 경향성을 확인해야 한다. 본 논문에서는 추가적인 읽기 횟수에 미치는 요소를 액티베이션과 가중치가 둘다 0 이 아닌 값이어서 실제 연산이 일어난 비율, 버스 비트 폭에 실을 수 있는 타일의 개수(*#TileInBus*) 그리고 액티베이션 버퍼에 들어갈 수 있는 타일의 개수(*#TileInBuff*)로 파악하였다. 다음은 3 가지 요소가 추가적인 읽기 횟수에 미치는 영향을 나타낸다.

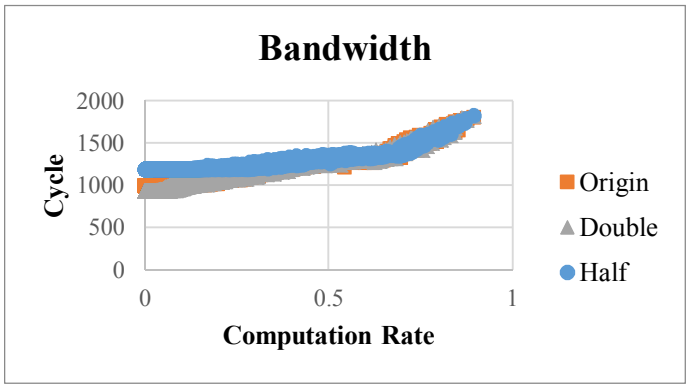
⑴ 실제 연산이 일어난 비율 ($Rate_{Comp}$)

실제 연산이 일어난 비율은 PG 샘플드 시뮬레이션을 통해 추출한다. 시뮬레이션 되는 PG 에서 입력받은 데이터 중 액티베이션과 가중치가 둘다 0 이 아니어서 실제로 연산이 일어난 비율이다. 추가적인 읽기 횟수는 연산이 많을수록 증가하기 때문에 $Rate_{comp}$ 과 비례한다.

⑵ 버스 비트 폭에 실을 수 있는 타일의 개수(*#TileInBus*)

추가적인 읽기 횟수는 버스에 실을 수 있는 타일의 개수에 영향을 받을 것으로 예측하였다. 만약 동시에 다수의 PG 에서 추가적인 읽기가 발생할 경우 *#TileInBus* 에 따라 추가 읽기의 횟수가 달라질 것이기 때문이다. *#TileInBus* 가 추가적인 타일 읽기에 영향을 미치는 경향성을 파악하기 위해 버스의 비트 폭을 변경해가며 전체 시뮬레이션을 통해

확인하였다. [그림 12]는 버스 비트 폭을 참고 문헌[2]의 설정(Origin)과 2 배(Double), 1/2 배(Half) 하였을 때 나타난 전체 시뮬레이션의 성능이다. [그림 12]를 통해 버스 비트 폭을 2 배하였을 때, 추가적인 통신이 증가하고, 1/2 배하였을 때, 추가적인 통신이 감소하는 것을 확인할 수 있다. 따라서 추가적인 읽기 횟수는 $Rate_{\mathrm{Comp}}$ 가 증가함에 따라 *#TileInBus* 와 반비례한다.

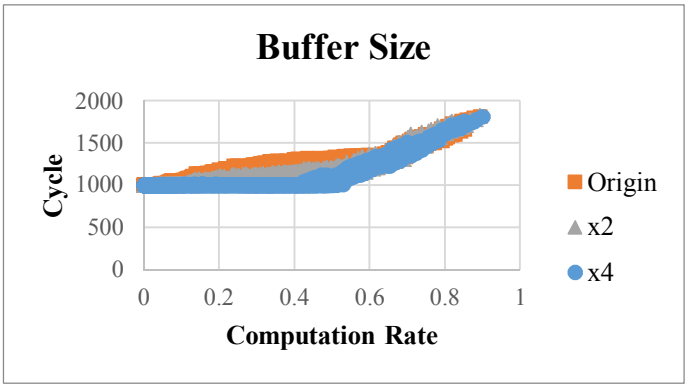


[그림 12] 버스 비트 폭에 따른 성능 비교

⑶ 액티베이션 버퍼에 들어갈 수 있는 타일의 개수(*#TileInBuff*)

마지막으로 추가적인 읽기 횟수는 액티베이션 버퍼에 들어갈 수 있는 타일의 개수에 영향을 받을 것으로 예측하였다. 액티베이션 버퍼의 크기가 크면 더 많은 타일을 저장할 수 있고, 이에 따라 액티베이션 버퍼의 공간이 부족하여 읽기가 지연되는 경우가 줄어들기 때문이다. 이러한 경향성을 파악하기 위해 전체 시뮬레이션을 통해 액티베이션 버퍼의 크기를 2 배, 4 배하였을 때의 성능을 비교하였다. [그림 13]을 보면 본래의 설정의 경우, 액티베이션 버퍼의 크기가 작아 통신 지연이 많이 발생하며 버퍼의 크기를 2 배와 4 배를 하였을 때 추가적인 읽기

횟수가 줄어드는 것을 확인할 수 있다. 이를 통해 추가적인 읽기 횟수는 $Rate_{\mathrm{Comp}}$가 증가함에 따라 액티베이션 버퍼의 크기에 반비례함을 알 수 있다.



[그림 13] 액티베이션 버퍼 크기에 따른 성능 비교

[그림 12]과 [그림 13]를 통해 알 수 있듯이 추가적인 타일 읽기 회수에 영향을 미치는 요소인 *#TileInBus* 와 *#TileInBus* 는 그림의 X 축 값인 $Rate_{\mathrm{Comp}}$ 의 기울기에 영향을 준다. 따라서 본 논문에서는 식(10)과 같이 $Rate_{Addit.Read}$ 에 대한 회귀 분석 모델을 구성하였다. 회귀 분석 모델에 사용될 $Rate_{Addit.Read}$는 식 (11)과 같이 전체 시뮬레이션을 통해 얻은 읽기 횟수와 샘플드 시뮬레이션을 통해 얻은 읽기 횟수의 차를 이용하여 추가적인 읽기 횟수를 구하고 전체 타일 수와의 비율을 통해 구하였다.

$$Rate_{Addit.Read} = \alpha \times Rate_{\mathrm{Comp}} \times \frac{\#TileInBus}{\#TileInBuff} + \beta \quad - \quad (10)$$

$$Rate_{Addit.Read} = \frac{\#Read_{Full} - \#Read_{Sampled}}{\#TotalTile} \quad - \quad (11)$$

5. 실험

본 논문에서는 제안한 시뮬레이션 기술을 검증하기 위해 [표 2]에 나타낸 5-bit Logquant 버전의 ZeNA[2] 설정으로 전체 시뮬레이션의 결과와 하이브리드 시뮬레이션의 결과를 AlexNet 합성곱 레이어에 대해 비교하였고 연산 비율에 따라 시뮬레이터의 하드웨어 설정을 변경하며 비교하였다.

PE 의 개수	액티베이션 버퍼	가중치 버퍼	부분합 버퍼
360 (8x45)	121x5b	121x5b	16x11b

[표 2] **5-bit Logquant 버전의 ZeNA 하드웨어 설정**

실험을 진행한 환경의 운영체제는 Ubuntu 14.04, CPU 는 Intel Core i7-4790 @3.60Hz, 그리고 시뮬레이터는 g++ 4.8.4 를 이용하여 컴파일되었다.

회귀 분석 모델에 대한 α, β 값을 구하기 위해 액티베이션과 가중치를 Alexnet 합성곱 레이어 3 의 크기에 대하여 각각 1 의 비율을 100%부터 0%까지 5%씩 줄여가며 총 400 개의 시뮬레이터 입력을 생성하였다. 그 후 생성한 입력을 이용하여 [표 2]의 설정, 액티베이션 버퍼의 크기를 2 배, 4 배 한 설정, 그리고 버스의 비트 폭을 2 배, 1/2 배 한 설정, 총 5 가지 설정에 대해 전체 시뮬레이션과 하이브리드 시뮬레이션을 수행하여 회귀 분석할 정보를 얻고 R 을 통해 α, β 값을 구하였다. (α=0.370, β=0.027)

5.1 AlexNet 합성곱 레이어를 통한 검증

AlexNet 합성곱 레이어를 통한 검증은 PE 샘플드 시뮬레이션 만을 적용했을 경우와 PE 와 PG 샘플드 시뮬레이션을 모두 적용했을 경우에 대하여 실험하였다.

5.1.1 PE 샘플드 시뮬레이션 실험

(Cycle, Err(%))

Layer	Full Sim.	PE Sampled Sim.		
		Top5	Top3	Top1
Conv1	368,320	368,320 (0%)	368,224 (−0.03%)	368,128 (−0.05%)
Conv2	1,034,160	1,024,536 (−0.93%)	1,007,648 (−2.56%)	928,488 (−10.22%)
Conv3	370,104	361,008 (−2.46%)	355,088 (−4.06%)	344,456 (−6.93%)
Conv4	556,580	543,640 (−2.32%)	534,576 (−3.95%)	515,968 (−7.30%)
Conv5	384,480	377,336 (−1.86%)	371,320 (−3.42%)	357,332 (−7.06%)
Total	2,713,644	2,674,840 (−1.43%)	2,636,856 (−2.85%)	2,514,372 (−7.34%)
Sim time(sec)	289.3	47.3 (6x)	29.4 (10x)	11.4 (25x)

[표 3] **AlexNet 합성곱 레이어에 대한 PE 샘플드 시뮬레이션 결과**

[표 3]은 본 논문에서 제안한 PE 샘플드 시뮬레이션의 결과 나타낸다. PE 샘플링을 가중치에 0 이 아닌 값이 많은 순으로 5 개, 3 개, 1 개의 PE 를 샘플링하여 실험하였다. PE 의 가중치에 0 이 아닌 값이 가장 많더라도 액티베이션 값에 따라 가장 오래 걸리는 PE 가 아닐 수 있기 때문에 시뮬레이션 되는 많은 PE 시뮬레이션 할 때, 오차가 줄어드는 것을 확인할 수 있다.

5.1.2 PG 샘플드 시뮬레이션 실험

PE 와 PG 샘플드 시뮬레이션을 모두 적용한 실험은 회귀 분석 모델을 만들 때, PE 의 수를 5 개로 하여 회귀 분석 모델을 만들었다. 추가적인 읽기 횟수를 구하기 위한 식(11)의 $Read_{Fulld}$ 은 전체 시뮬레이션을 통해 추출하였다.

[표 4]은 Alexnet 합성곱 레이어에 대하여 전체 시뮬레이션과 PE 와 PG 샘플드 시뮬레이션 적용한 분석모델의 실험 결과이다. 실험 결과, 대체적으로 PE 개수를 늘려 시뮬레이션 할 때 정확도가 좋아진다. 모든 경우에서 적은 수의 PE 를 시뮬레이션 할수록 결과가 낮게 나오는 것을 확인할 수 있는데 이는 적은 수의 PE 를 시뮬레이션 할수록 $Rate_{Comp}$ 가 감소하기 때문이다. Conv2 의 경우 Top1 이 가장 정확도가 높은데 이는 회귀 분석 오류로 인해 Top5 의 성능이 과예측 되어 시뮬레이션 결과가 낮을수록 정확도가 높아지기 때문이다.

Layer	Full Sim.	PE+PG Sampled Sim.		
		Top5	Top3	Top1
Conv1	368,320	368,388 (0.02%)	368,296 (−0.01%)	368,208 (−0.03%)
Conv2	1,034,160	1,065,168 (3.00%)	1,064,568 (2.94%)	1,062,800 (2.77%)
Conv3	370,104	355,328 (−4.42%)	354,996 (−4.46%)	353,600 (−4.80%)
Conv4	556,580	531,992 (−4.42%)	531,768 (−4.46%)	529,856 (−4.80%)
Conv5	384,480	369,616 (−3.87%)	369,344 (−3.94%)	367,944 (−4.30%)
Total	2,713,644	2,690,492 (−0.85%)	2,688,972 (−0.91%)	2,682,408 (−1.15%)
Sim time(sec)	289.3	7.314 (39x)	5.241 (55x)	3.166 (91x)

[표 4] **AlexNet 합성곱 레이어에 대한 PE+PG 샘플드 시뮬레이션 결과**

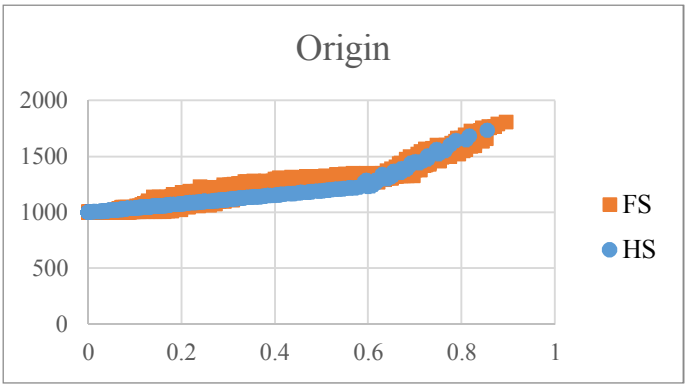
본 논문에서 제안한 하이브리드 시뮬레이션 기법을 모두 적용하여 AlexNet 의 전체 시뮬레이션과 성능을 비교하였을 때, PE 를 5 개 이용한 하이브리드 시뮬레이션(Top5)의 경우, −0.85%의 오차를 보이며 속도를 39 배 가속하였고, PE 를 1 개 이용한 경우(Top1), −1.15%의 오차를 보이며 91 배 가속하였다.

5.2 하드웨어 설정 변경에 따른 비교 실험

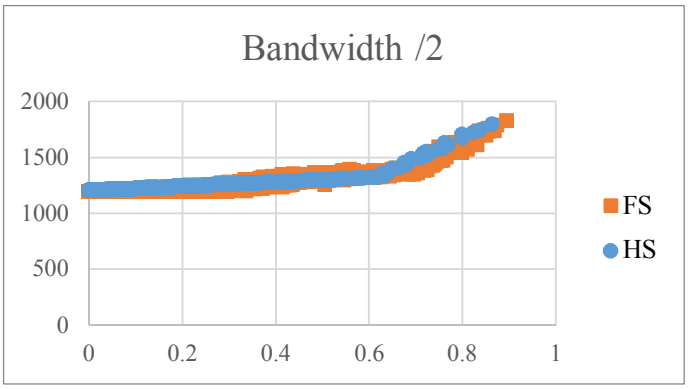
본 논문에서는 ZeNA 의 설계공간탐색을 위한 도구로써 제안한 성능 예측 방법을 검증하기 위해 하드웨어 설정을 변경하며 실험하였다. 시뮬레이터의 입력 데이터는 액티베이션과 가중치 각각에 대하여 0 의 비율을 0%에서 100%까지 5% 간격으로 생성하였다. [그림 14]부터 [그림 18]까지 나타난 실험 결과를 통해 전체 시뮬레이션과 하이브리드 시뮬레이션의 경향성이 일치하는 것을 확인할 수 있다.

(1) 버스 비트 폭 변경에 따른 실험

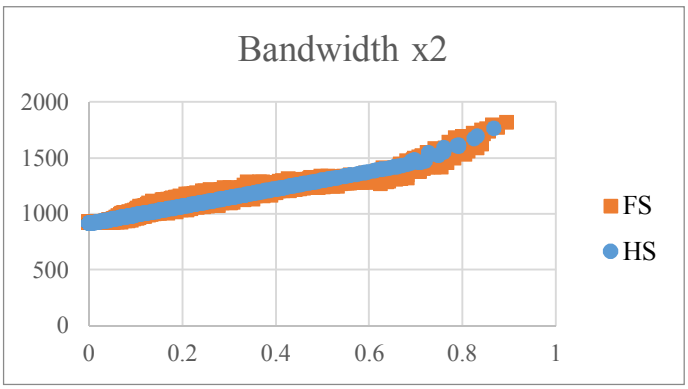
버스 비트 폭은 [표 2]에 나타낸 설정과 해당 설정에서 버스 비트 폭을 절반으로 줄였을 때와 두배로 늘렸을 때에 대하여 실험하였다. 그래프의 X 축은 $Rate_{Comp}$ 이고 Y 축은 성능을 나타낸다. 실험 결과 그림을 통해 알 수 있듯이 전체 시뮬레이션 (FS, Full Simulation)과 하이브리드 시뮬레이션 (HS, Hybrid Simulation)의 경향성이 유사한 것을 확인할 수 있다.



[그림 14] [표 2]의 설정에 대한 경향성 비교



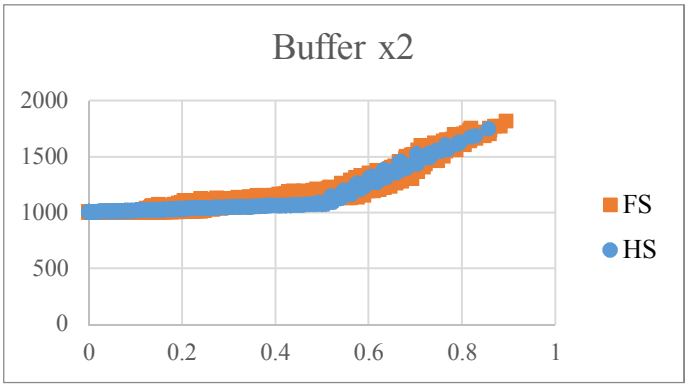
[그림 15] 버스 비트 폭을 절반으로 줄였을 때 경향성 비교



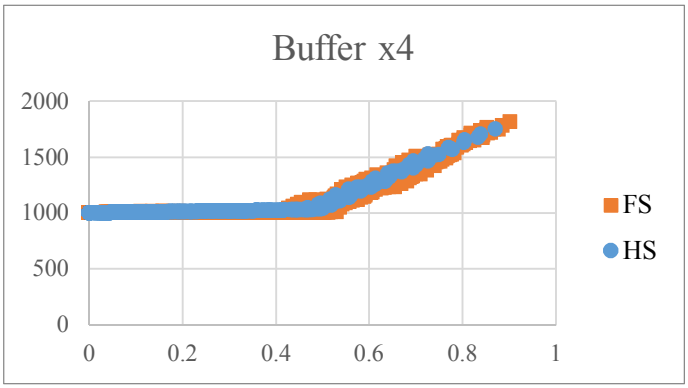
[그림 16] 버스 비트 폭을 두배로 늘렸을 때 경향성 비교

(2) 액티베이션 버퍼 크기 변경에 따른 실험

액티베이션 버퍼 크기 변경은 [표 2]에서 액티베이션 버퍼의 크기만 2 배, 4 배 한 설정에 대하여 실험하였다. 실험 결과 액티베이션 버퍼의 크기를 4 배로 늘렸을 때, ZeNA 는 $Rate_{Comp}$ 가 대략 0.5 일 때까지 일정한 성능을 유지하는 것을 확인할 수 있다. 이는 액티베이션 버퍼에 많은 타일을 저장할 수 있어 연산에 의한 통신지연이 발생하지 않는 것을 의미한다. 액티베이션 버퍼 크기에 변경에 따른 실험에서도 전체 시뮬레이션의 경향성과 유사한 것을 확인할 수 있다.



[그림 17] 액티베이션 버퍼의 크기를 2 배 늘렸을 때 경향성 비교



[그림 18] 액티베이션 버퍼의 크기를 4 배 늘렸을 때 경향성 비교

6. 결론

본 논문에서는 타겟 아키텍처의 빠른 성능 예측을 위하여 분석 모델을 세웠다. 정적인 정보만을 사용하는 기본 분석 모델과는 달리, 수행 시간에 변화하는 동적인 정보도 고려하여 예측의 정확도를 올렸다. 동적인 정보를 추출해내는 시뮬레이션의 시간을 단축하기 위하여 프로세서의 일부 구성 요소만을 시뮬레이션하는 샘플드 시뮬레이션을 이용하였고 샘플링 과정에서 유실되는 정보는 통계적 모델을 통해 유추하였다. 이러한 하이브리드 시뮬레이션을 통하여 AlexNet 의 실험결과 -1.15% 의 정확도 손실로 최대 91 배의 성능 향상을 얻었고, 하드웨어 설경 변경에 따른 비교를 통해 제안하는 성능 예측 기법의 효과와 유효성을 검증하였다.

참고 문헌

[1] Huang et al. 2014. “TBPoint: Reducing simulation time for large-scale GPGPU kernels.” In IPDPS. 437–446.

[2] Bakhoda et al. 2009. “Analyzing CUDA Workloads Using a Detailed GPU Simulator.” In ISPASS, 163-174.

[3] Kang et al. 2018. “NNSim: Fast Performance Estimation based on Sampled Simulation of GPGPU Kernels for Neural Networks.” In Proceedings of the 55th Annual DAC.

[4] Kim et al. 2018. “ZeNA: Zero-Aware Neural Network Accelerator.” IEEE Design & Test 35, 1(2018). 39-46.

[5] Krizhevsky et al. 2012. “ImageNet classification with deep convolutional neural networks.” in Proceedings of the NIPS.

[6] O’Neal et al. 2017. “HALWPE: Hardware-Assisted Light Weight Performance Estimation for GPUs.” In Proceedings of the 54th Annual DAC