논문 2022-17-43

FPGA기반 뉴럴네트워크 가속기에서 2차 타일링 기반 행렬 곱셈 최적화

(Optimizing 2-stage Tiling-based Matrix Multiplication in FPGA-based Neural Network Accelerator)

권 진 세, 이 제 민, 권 용 인, 박 제 만, 유 미 선, 김 태 호, 김 형 신*
(Jinse Kwon, Jemin Lee, Yongin Kwon, Jeman Park, Misun Yu, Taeho Kim, Hyungshin Kim)

Abstract : The acceleration of neural networks has become an important topic in the field of computer vision. An accelerator is absolutely necessary for accelerating the lightweight model. Most accelerator-supported operators focused on direct convolution operations. If the accelerator does not provide GEMM operation, it is mostly replaced by CPU operation. In this paper, we proposed an optimization technique for 2-stage tiling-based GEMM routines on VTA. We improved performance of the matrix multiplication routine by maximizing the reusability of the input matrix and optimizing the operation pipelining. In addition, we applied the proposed technique to the DarkNet framework to check the performance improvement of the matrix multiplication routine. The proposed GEMM method showed a performance improvement of more than 2.4 times compared to the non-optimized GEMM method. The inference performance of our DarkNet framework has also improved by at least 2.3 times.

Keywords : Versatile Tensor Accelerator (VTA), Neural Processing Unit (NPU), General Matrix Multiplication, GEMM Optimization

Ⅰ. 서 론

뉴럴네트워크의 경량화 및 가속화가 컴퓨터 비전 분야의 중요한 주제로 다루어지고 있다 [1-2]. 뉴럴네트워크 경량화를 위해 네트워크 구조 탐색 (Network Architecture Search), 지식증류 (Knowledge Distillation), 가지치기 (Pruning), 재매개변수화 (Reparameterization), 양자화 (Quantization) 등이 연구되고 있다 [3-5]. 하지만 단순히 뉴럴네트워크 경량화만으로는 성능 향상을 기대할 수 없으며, 뉴럴네트워크를 효율적으로 실행할 수 있는 하드웨어가 수반되어야 한다 [6]. 뉴럴네트워크 가속이 가능한 상용 하드웨어는 구글의 TPU (Tensor Processing Unit) [7], 인텔의 NCS (Neural Compute Stick) [8] 등 의 ASIC (Application Specific Integrated Circuit) 기반 가속기가 있다. FPGA (Field Programmable Gate Array) 기반의 연구용 가속 하드웨어로는 TVM 백엔드로 공개된 오픈소스 VTA (Versatile Tensor Accelerator) [9] 가속기가 대표적이다. CNN (Convolutional Neural Network:CNN)의 핵심 연산인 컨볼루션 연산은 Im2col (Image to Columnize)변환을 통해 행렬 곱셈으로 차원축소가 가능하다 [10]. 행렬 곱셈 연산으로 차원을 낮추게 되면 이미 잘 정의되어있는 BLAS (Basic Linear Algebra Subprogram) 라이브러리를 행렬 곱셈에 적용할 수 있다. 하지만, 기존 BLAS 라이브러리는 범용 CPU, GPU를 타겟으로만 최적화 되어, 새로운 하드웨어에 적용이 어렵다 [10-15]. 또한, 정확도 표현을 낮추어 데이터양을 줄이는 양자화 기법을 적용하고자 할 때 INT8 과 같은 저정확도 데이터 표현을 대부분의 BLAS에서 지원하지 않고 있다. 따라서, 기존의 BLAS를 새로운 가속 하드웨어 바로 적용하는 것은 현실적으로 불가능하다.

본 논문에서는 FPGA 기반 뉴럴네트워크 가속기 VTA와 같은 행렬 곱셈기반의 가속 회로에서 적용 가능한 2차 타일링 GEMM 기법을 제안하고자 한다. 본 논문의 기여를 정리하면 다음과 같다.

- 2차 타일링 기반 최적화 행렬 곱셈 : 행렬 곱셈에서 행렬들의 크기가 커질수록 메모리 접근에 비용이 커지게 된다. 이러한 메모리 접근의 비효율성을 개선하고자 타일링 기법을 사용하게 되는데, 본 논문에서는 하드웨어에 최적화된 타일 크기와 메모리에 최적화된 크기 2가지의 타일로 메모리를 구성함으로 연산 성능을 극대화하였다. 제안하는 기법은 비정방형행렬, 정방형행렬 케이스에서 평균 2.4배 이상의 성능 개선을 보였다.
- DarkNet [16] 딥러닝 프레임워크 개선 : 제안하는 행렬 곱셈 가속 기법의 성능을 확인하기 위해서 오픈소스 딥

*Corresponding Author (hyungshin@cnu.ac.kr)
Received: Oct. 21, 2022, Revised: Nov. 11, 2022, Accepted: Nov. 23, 2022.
Jinse Kwon: Chungnam National University (Ph.D. Student)
J. M. Lee, Y. I. Kwon, J. M. Park: ETRI (Senior Researcher)
M. S. Yu: ETRI (Principal Researcher)
T. H. Kim: ETRI (Assistant Vice President)
H. S. Kim: Chungnam National University (Prof.)
※ 이 논문은 2022년도 정부 (과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-00769,인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발).

ⓒ IEMEK J. Embed. Sys. Appl. 2022 Dec. 17(6) 367-374
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2022.17.6.367

러닝 프레임워크인 DarkNet의 코드를 수정하여 최적화된 VTA 백엔드를 구현하였다. DarkNet 프레임워크는 Im2Col 기반의 행렬 곱셈은 지원하지만, 특정 가속 하드웨어를 지원하으며 양자화 같은 가속 기능들이 적용되어있지 않다. 따라서 본 연구를 위해 DarkNet 프레임워크에 VTA 백엔드를 지원하도록 하였고, 양자화 일부 기능을 지원하도록 수정하였다. 제안하는 기법을 적용하였을 때 ResNet-18 모델 [17]의 추론 성능이 다크넷에서 기본 지원되는 CPU 대비 VTA 타겟에서 2.3배의 성능 향상을 달성하였다.

본 논문의 2장에서는 뉴럴네트워크 및 행렬곱셈과 관련된 연구들을 정리하여 연구의 목적을 설명하고자 한다. 3장에서는 제안하는 기법인 2차 타일링 행렬 곱셈 가속과 DarkNet 프레임워크 개선에 대해 자세히 소개한다. 4장에서는 제안하는 기법의 성능 개선을 실험 및 평가하여 연구의 성과를 제시하고, 5장에서 본 연구의 결론을 맺는다.

Ⅱ. 관련연구

1. 컨볼루션 연산의 차원 축소 기법

CNN의 핵심 연산은 그림 1 위와 같이 컨볼루션 연산으로써 입력이미지 (C x H x W)와 가중치값 (N x C x R x S)의 컨볼루션 곱셈으로 출력 이미지 (N x H x W)의 결과를 도출하게 된다. 직접 컨볼루션 (Direct Convolution) 연산의 경우 커널의 위치에 맞는 입력 이미지의 메모리를 접근하게 되는데, 이 경우 메모리 접근이 연속적이지 않으므로 캐쉬 미스가 빈번하게 발생하므로 성능 저하가 발생한다. 이러한 문제를 해결하기 위해 그림 1 중간과 같이 입력 이미지에서 커널 레이아웃이 접근할 위치의 값을 미리 복사하여 새로운 메모리에 저장하므로 메모리 사용량은 증가하지만, 고차원 컨볼루션 연산이 저차원 행렬 곱셈으로 차원 축소가 가능해진다. 이러한 변환 기법을 Im2Col (Image to Column) 라고 부른다. 차원 축소 기법은 입력 이미지의 메모리 사용량이 커널 크기를 곱한 만큼 증가하지만 연속적인 메모리 접근이 가능하므로 성능이 향상되고, 행렬 곱셈 가속 코드를 포함하는 선형대수 연산 라이브러리인 BLAS를 사용할 수 있게 된다. Im2Col과 비슷한 방식으로는 모델 파라미터를 분리하여 계산한 후 최종 출력 이미지에서 쉬프트 누적을 통한 컨볼루션 연산을 수행하는 Kn2Row (Kernel to Row) 기법이 존재한다 [18]. 이 방식은 그림1의 아래와 같이 결과를 미리 알 수 없는 입력 이미지 대신 상수값을 갖는 가중치 (weight) 파라미터의 메모리 레이아웃을 재배치하여 연산한다. 하지만, 출력 이미지의 쉬프트 덧셈 방식의 성능 저하와 출력 이미지의 메모리 사용량 증가로 성능 개선이 Im2Col 대비 크지 않아 범용적으로 사용되지 않는다 [19].

기존의 Im2Col의 최적화 기법으로는 MEC (Memory Efficient Convolution) [20], Indirect Convolution [21] 등이 있다. MEC 연산은 Im2Col 변환을 기반으로 하여, 메모리 접근 방향을 열 방향에서 행 방향으로 데이터 레이아웃을 변경함으로 커널의 재사용성을 높이므로 메모리 사용량을 줄

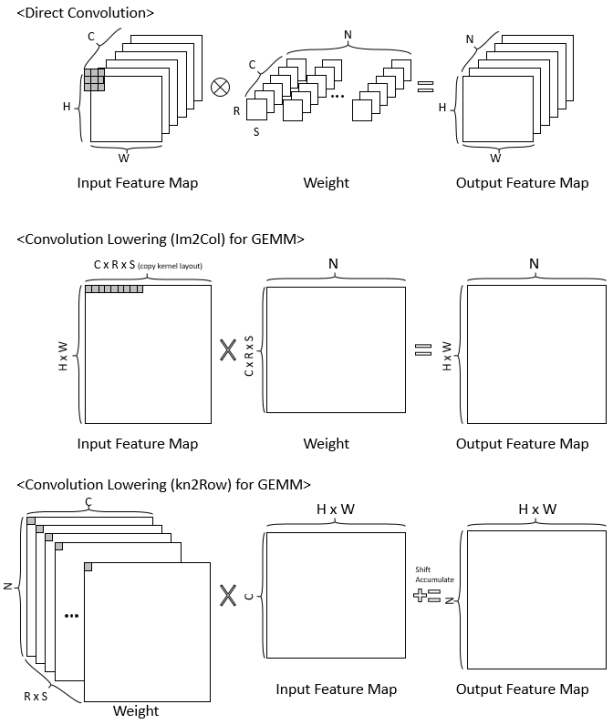

그림 1. (위) 직접 컨볼루션 연산 과정 도식,
(중간) Im2Col 차원 축소를 통한 행렬 곱셈 변환
(아래) kn2Row 차원 축소를 통한 행렬 곱셈 (누적) 변환
Fig. 1. (Above) Direct Convolution Method
(Middle) Im2Col Convolution Lowering Transformation
(Bottom) Kn2Row Convolution Lowering Transformation

일 수 있는 기법이다. Indirect Convolution은 Im2Col과 방식은 동일하지만 메모리를 복사하지 않고, 포인터 버퍼를 통해 메모리를 접근하도록 하여 메모리 사용량을 줄이는 기법을 제안하였다. MEC 의 경우 커널의 크기가 크고 stride가 1인 경우에는 성능 이득이 발생하지만, 커널의 크기가 작거나, stride가 2 이상인 경우에 오히려 성능이 저하되는 구조적 한계가 있다.

2. BLAS, BLIS 라이브러리

BLAS 라이브러리는 타겟 CPU, GPU에서 선형대수 연산 루틴들이 최적의 성능을 갖도록 연구 개발되고 있다. 범용 CPU에 대한 최적화 루틴을 갖는 BLAS 라이브러리로는 intel-MKL (Math Kernel Library)[15], LAPACK [12], OpenBLAS [11] 등이 있고, ARM 계열의 CPU BLAS는 직접 컨볼루션 루틴을 포함하는 BLIS (BLAS-like Library Initiation Software) 형태로 발전하고 있다. 대표적인 예로 ARM-CL (Compute Library) [13], NNPACK [14] 등이 있다. GPU BLAS는 NVIDIA 계열의 CuBLAS [22], 범용 GPU 프로그래밍을 위한 OpenCL를 이용한 CLBlast [23], clBLAS [24] 등이 있다. 오랜 기간 잘 정의된 BLAS, BLIS 라이브러리라고 할지라도, 새로운 딥러닝 가속기에는 바로 적용하는 것은 불가능하며, 새로운 가속기를 위한 연산 라이브러리를 새롭게 개발해야 하는 어려움이 있다.

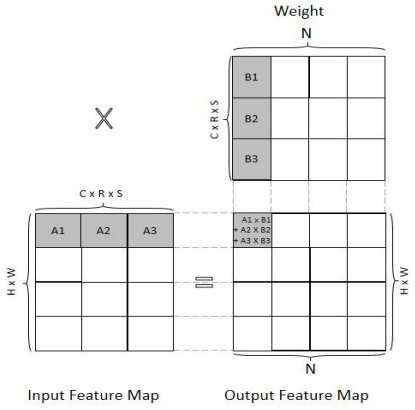

그림 2. 타일링 기반 행렬 곱셈
Fig. 2. Tiling-based matrix multiplication

3. 행렬 곱셈의 타일링 최적화

그림 2는 일반 행렬 곱셈에서 일반적으로 사용하는 타일링 기법을 도식하였다. 일반 행렬 곱셈은 좌측 행렬의 한 행과 우측 행렬의 한 열을 곱하여 출력 행렬의 하나의 값을 계산하게 되는데, 행렬의 크기가 커질수록 캐쉬라인을 넘어가는 메모리 접근이 빈번하게 발생하므로 성능 저하가 발생하게 된다. 이러한 캐쉬 미스를 방지할수 있는 방법으로 타일링 기법이 제안되어졌다 [25]. 캐쉬 미스를 최소화 하는 타일 크기의 서브 행렬로 나누고, 서브 행렬들의 곱셈을 수행한 후에 출력 서브 행렬에 누적하는 방식으로 연산하게 된다.

단순 타일링 기법이 모든 크기의 행렬에서 항상 성능 개선이 보이지는 않는다. A행렬 (MxK)과 B (KxN)행렬이 있다고 하자. 충분히 큰 정방형 행렬 (M=K=N)에서 타일 크기에 따른 성능 개선이 선형적으로 증가하지만, 일정 크기 이상으로 커지게 되면 오히려 성능이 저하된다. 비정방형 행렬에서 A행렬이 열 방향으로 길고, B 행렬이 행 방향으로 긴 경우 (M,N ≫ K) 오히려 타일의 빈번한 교체가 발생하므로 성능 저하가 발생할 수 있다.

4. VTA 하드웨어 구조

VTA 하드웨어는 오픈 소스로 공개되어 있어, 그림 3과 같이 VTA 하드웨어 여러 모듈과 세부요소들에 대해 상세히 살펴볼 수 있다 [9]. DRAM과 FPGA 메모리인 Buffer 사이의 데이터 흐름을 제어하기 위한 큐 (Queue)가 존재한다. LOAD 모듈은 DRAM으로부터 입력데이터와 가중치 데이터를 읽어오며, STORE 모듈은 결과 데이터를 DRAM으로 저장한다. 각 입력 버퍼와 가중치 버퍼는 16의 배수로 COMPUTE 모듈의 입력 텐서와 가중치 텐서에 로드된다. COMPUTE 모듈은 16x16 행렬과 16x1 벡터 연산을 한 번의 명령으로 수행할 수 있다. VTA 각 명령어는 Instruction Fectch 모듈을 통해 COMPUTE 모듈로 전송되며, 각 명령어는 OP-code와 Operand로 구성되어있다. Operand는 버퍼의 인덱스로 구성되는데 입력 버퍼는 16의 배수, 가중치 버퍼는 16x16을 한 개의 인덱스만으로 접근이 가능하다. 입력 버퍼, 가중치 버퍼, 출력 버퍼의 크기는 명령어의 Operand 길이에 의존하게

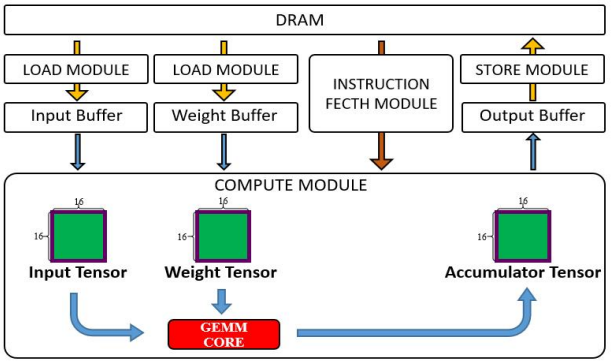

그림 3. VTA 하드웨어 구조 [9]
Fig. 3. VTA Hardware Architecture [9]

된다. 만약 인덱스를 초과 할 경우 기존의 버퍼를 DRAM으로 저장하고 비운 후에, 접근하고자 하는 위치의 데이터를 DRAM으로 새로 로드한 후에 사용해야 한다.

Ⅲ. 제안하는 기법

1. 2차 타일링 기법

그림 4는 단순 타일링 기법에 추가로 서브 행렬을 타일링하는 2차 타일링 방식을 도식하였다. 2차 타일링 기법은 1차 타일로 구획을 나누고, 2차 세부 타일로 실제 연산 부분을 구분하게 된다. 그림 5의 알고리즘과 같이 행렬 A,B를 block_size로 구획을 나누고 block_size에 대해 세부 타일 크기인 16x16의 크기로 잘게 쪼갤 수 있게 된다. 16x16 행렬의 곱셈에만 집중 하므로 최적화 문제를 쉽게 만들 수 있다. 본 논문에서 제안하는 2차 타일링 기법은 VTA 하드웨어 뿐만 아니라 텐서 기반의 뉴럴네트워크 가속회로에도 바로 적용이 가능할 것으로 기대한다.

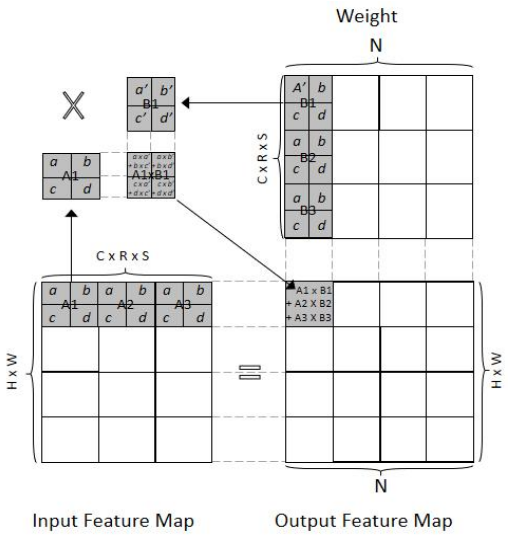

그림 4. 2차 타일링 기반 행렬 곱셈
Fig. 4. 2-stage tiling-based matrix multiplication

Algorithm 1 : 2-stage Tile-based baseline GEMM on CPU

```
1  for lm = 1:M/block_size do
2  for lk = 1:K/block_size do
3  for ln = 1:N/block_size do
4      for bm = 1:block_size/16 do
5      for bk = 1:block_size/16 do
6      for bn = 1:block_size/16 do
7          for h = 1:16 do
8          for w = 1:16 do
9          for x = 1:16 do
10             C[h,w](lm,ln,bm,bn) += A[h,x](lm,lk,bm,bk) * B[w,x](lk,lm,bn,bk)
11         end
12         end
13         end
14     end
15     end
16     end
17 end
18 end
19 end
```

그림 5. 2차 타일링 기반 CPU 기본 행렬곱셈 알고리즘
Fig. 5. 2-stage tiling-based GEMM algorithm on CPU

Algorithm 2 : 2-Stage Tile-based GEMM on VTA

```
1  for lm = 1:M/block_size do
2  for lk = 1:K/block_size do
3  for ln = 1:N/block_size do
4      LOAD_BLOCK( input_buffer, block_size * block_size)
5      LOAD_BLOCK( weight_buffer, block_size * block_size)
6      for bm = 1:block_size/16 do
7      for bk = 1:block_size/16 do
8      for bn = 1:block_size/16 do
9          VTA_GEMM(C[16x16](bm,bn) += A[16x16](bm,bk) * B[16x16](bn,bk))
10     end
11     end
12     end
13     STORE_BLOCK( output_buffer, block_size * block_size)
14 end
15 end
16 end
```

그림 6. 2차 타일링 기반 VTA 행렬곱셈 알고리즘
Fig. 6. 2-stage tiling-based GEMM algorithm on VTA

그림 6의 알고리즘은 VTA 하드웨어에 적용하기 위해 그림 5의 알고리즘을 개선한 버전이다. 그림 5의 7~13번 라인의 16x16 행렬 곱셈 연산을 위한 반복문 3개가 그림 6의 9번 라인의 VTA_GEMM 명령어 한 줄로 대치된다. 버퍼 로드와 연산 과정의 파이프라인이 형성되지 않을 경우 메모리 병목현상이 발생하게 된다. 따라서 그림 6에서 4~5번 라인의 입력 이미지와 가중치 파라미터의 로드 및 연산, 13번 라인의 저장을 연속적으로 수행하도록 작성하였다.

2. VTA 하드웨어에서 2차 타일링 최적화

VTA 하드웨어는 16x16 행렬과 16x1 벡터를 한 번에 연산 할 수 있다. 그림 7 에서와 같이 행렬 곱셈을 위해서는 DRAM에서 FPGA의 온 칩 메모리 (Buffer)로 데이터가 1차적으로 로드된다. 그리고 GEMM 코어 모듈 내의 레지스터 파일로 존재하는 텐서에 2차적으로 데이터가 로드되게 된다. 이러한 VTA 하드웨어 특성을 최대한 이용하려면 입력, 가중치 버퍼에 충분히 많은 데이터를 가지고 와야 한다. 그

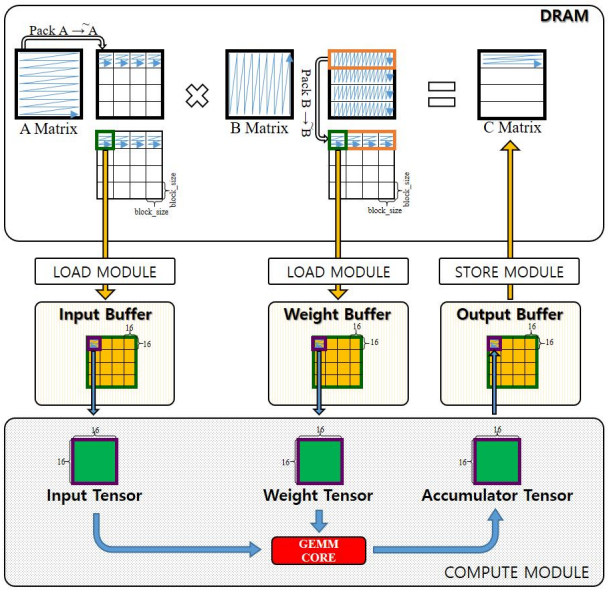

그림 7. VTA의 하드웨어 구조를 이용한 2차 타일링 기반 행렬 곱셈 최적화 기법
Fig. 7. 2-stage tiling-based matrix multiplication optimization technique on VTA

Algorithm 3 : Optimized 2-Stage Tile-based GEMM on VTA (Ours)

```
1  for lm = 1:M/ mblock_size do
2    for bm = 1: mblock_size/16 do
3      for lk = 1:K/ kblock_size do
4        LOAD_BLOCK( input_buffer, 16 * kBLOCK )
5        for n = 1: N/16 do
6          LOAD_BLOCK( weight_buffer, 16 * kBLOCK )
7          for bk = 1: kblock_size/16 do
8            VTA_GEMM(C[16x16](bm,bn) += A[16x16](bm,bk) * B[16x16](n,bk))
9          end
10       end
11     end
12   end
13   // Flushing after filling the accumulator buffer
14   STORE_BLOCK(output_buffer, mBLOCK * n/16 )
15 end
```

그림 8. VTA에 최적화된 2차 타일링 기반 행렬곱셈 알고리즘
Fig. 8. Optimized 2-stage tiling-based GEMM algorithm on VTA

리고 입력,가중치 텐서에서는 재사용성을 높일 수 있는 방식으로 GEMM 코어의 가동률을 높여야 한다. 이 두 가지를 모두 반영하기 위한 기법을 그림 8의 알고리즘으로 작성하였다. 그림 6 알고리즘과 차이점은 1차 타일을 기존에는 동일한 타일 크기로 구획하였지만, 그림8의 최적 알고리즘에서는 M블록과 K블록 두가지 타일 형태로 구분하여 구획하였다. 두가지 크기로 1차 타일을 구분한 이유는 A행렬과 B행렬에서 K의 크기는 동일한 점을 이용하여 A행렬의 재사용성을 높이기 위한 전략이다. 그림 6 알고리즘의 6번 라인이 최적 알고리즘의 2번 라인으로 옮겨진 것 또한 2차 세부 타일 단위로 A행렬을 재사용하도록 의도한 것이다. 그림 8의 4번 라인은 A 행렬을 한번 로드 할 때 5, 6번 라인의 코

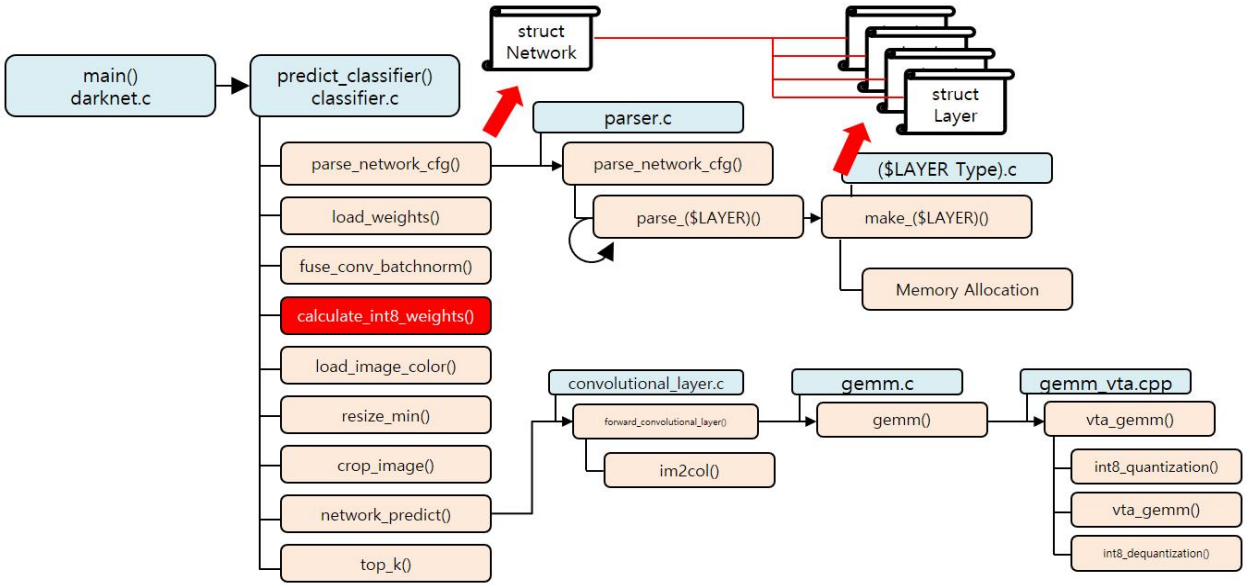

그림 9. DarkNet 프레임워크의 추론 코드 분석 및 VTA GEMM 루틴 추가
Fig. 9. Analysis of Inference code and Adding VTA GEMM routine in Darknet

드처럼 가중치 버퍼 (16x16)를 N블록 만큼 로드하여 입력 버퍼를 재사용한다. 뿐만아니라 8번 라인의 VTA_GEMM 연산과 6번 라인의 가중치 버퍼의 파이프라인이 생성되므로 연산과 로드가 슬랙 없이 지속적으로 공급 연산 된다. 이러한 과정이 무한정 반복될 수는 없으며, 누적 버퍼의 인덱스가 한계값에 도달하게 되면 누적 버퍼를 출력 버퍼로 저장하고 누적 버퍼를 초기화하게 된다. 이러한 과정을 입력 버퍼를 모두 연산할 때 까지 반복하므로 VTA 가동율을 최대로 올리게 되었다.

3. DarkNet 프레임워크 성능 최적화

DarkNet 프레임워크 [16]는 순수하게 C로만 작성된 오픈소스 딥러닝 프레임워크이다. 지원가능한 타겟 하드웨어는 범용 CPU와 NVIDIA GPU로 다른 하드웨어에 대한 백엔드는 구현되어 있지 않다. 따라서 본 연구의 VTA 하드웨어에 최적화된 행렬 곱셈 연산을 DarkNet 프레임워크에 구현하였다. DarkNet 프레임워크는 이미지 분류와 물체 인식 두가지 타스크를 제공하고 있다. 하지만 이미지 분류 모델은 개발 지원이 중단되어 실험을 위해 ResNet-18 모델을 ONNX (Open Neural Network eXchange) 변환을 통해 Darknet으로 변환하였다.

DarkNet 이미지 분류를 위한 코드 흐름을 분석여 그림 9에 작성하였다. 먼저 darknet.c 파일에 main() 함수가 작성되어 있어, predict_classifier() 함수를 호출하여 이미지 분류를 수행하게 된다. classifier.c 파일에는 DarkNet 구성 메타데이터를 담고 있는 cfg 파일을 파싱하는 parse_network_cfg()함수와 파라미터 파일을 불러오는 load_weights() 함수, 그리고 추론과정에서 BatchNorm 레이어를 생략하기 위한 fuse_conv_batchnorm() 함수등의 부수적인 함수를 통과하면 추론을 위한 준비과정이 완료된다. 입력 이미지는 load_image_color() 함수를 통해 OpenCV 또는 std_img 라이브러리를 통해 픽셀 단위의 조정을 통해 network_predict()함수의 입력으로 들어가게 된다. DarkNet은 Network, layer 두 가지 구조체를 기반으로 동작하게 되는데, Network 구조체에는 레이어 정보들을 배열형태로 가지고 있으며 뉴럴네트워크에 필요한 데이터들을 저장하게 된다. 그리고 Layer 구조체는 cfg 파일에 정의 되어진 레이어들을 실행하는데 필요한 정보들을 담고 있다.

VTA 하드웨어는 연산 가능한 데이터 형식이 INT8 이므로 제안하는 최적의 행렬 곱셈 알고리즘을 DarkNet에 붙이기 위해 calcuate_int8_weight() 함수를 작성하여 가중치 파라미터를 미리 양자화 하는 루틴을 추가하여 런타임에 가중치는 양자화를 스킵하게 하였다. VTA 하드웨어에서 GEMM 연산이 수행되도록 하기 위해 convolutional_layer.c 파일의 forward_convolutional_layer() 함수를 포함한 하위 함수들을 수정하였다.

Ⅳ. 실험 및 평가

1. 실험 환경

- 타겟 보드 : Xilinx ZCU102 MPSoC
- Processing System (PS)
  - CPU : ARM Cortex-A53 quad-core
  - RAM : 4GB
- Programmable Logic (PL)
  - FPGA-기반 VTA 하드웨어
  - 클럭 주파수 : 333MHz
  - 입/출력데이터 형식 : 8비트 (양자화)

표 1. VTA 성능 검증을 위한 정방, 비정방형 행렬의 테스트 케이스
Table. 1. Test Cases of Square and Non-square Matrix for VTA Performance Verification

							Matrix Dimension		
							M	K	N
Square Matrix							128	128	128
							256	256	256
							512	512	512
							1024	1024	1024
	CNN Layers Dimension						M	K	N
	H	W	C	R	S	N			
Non-square Matrix	64	64	64	3	3	64	4096	576	64
	32	32	128	3	3	128	1024	1152	128
	16	16	128	3	3	128	256	1152	128
	16	16	256	3	3	256	256	2304	256
	8	8	256	3	3	512	64	2304	512
	8	8	512	3	3	512	64	4608	512

표 2. 정방형/비정방형 행렬 곱셈의 성능 비교 표
Table 2. Performance comparison for square and non-square matrix multiplication

(단위 : millisecond)

	Method / M, K, N	Baseline on CPU	2-Stage Tiling on VTA	Optimized 2-stage Tiling on VTA
Square Matrix	128, 128, 128	6.7	1.7	**0.6**
	256, 256, 256	51.3	4.9	**2.6**
	512, 512, 512	403.0	28.9	**10.6**
	1024, 1024, 1024	3208.9	184.8	**58.4**
Non-square Matrix	4096, 576, 64	905.0	59.9	**19.1**
	1024, 1152, 128	448.8	34.5	**10.3**
	256, 1152, 128	112.2	10.8	**4.7**
	256, 2304, 256	446.7	39.5	**14.2**
	64, 2304, 512	225.1	18.9	**14.4**
	64, 4608, 512	452.3	42.5	**28.1**

2. 실험 방법

VTA에 최적화된 2차 타일링 기반 행렬 곱셈의 성능을 검증하기 위한 방법으로 정방형 행렬 곱셈과 비정방형 행렬 곱셈을 실험하였다. 정방형 행렬은 행렬 곱셈을 위한 행렬 A(M x K), B(K x N)이 다음과 같을 때, M,K,N 크기가 모두 동일한 경우를 말한다. 비정방형 행렬의 경우 CNN 레이어의 차원축소를 통해 결정된 행렬의 크기로 실험하였다. 실험 데이터는 표 1과 같다. 정방형 행렬은 128 크기에서 2배씩 키워가며 행렬의 크기를 1024행렬 까지 실험하였다. 비정방형 행렬의 경우 ResNet-18 모델의 컨볼루션 레이어의 파라미터 및 입력 크기를 Im2Col 변환을 통해 행렬 차원을 구하였고, 해당 차원을 시험 데이터로 사용하였다.

DarkNet 프레임워크에서 제대로 동작하는지를 확인하기 위해서 MxNet에서 학습된 ResNet-18 모델을 ONNX 변환을 통해 DakNet cfg 파일과 weights 파일로 생성하였고 해당 모델을 양자화를 통해 Int8 모델로 변환하여 수행하도록 하였다. 정확도는 기존 FP32와 비교하여 검증하였다.

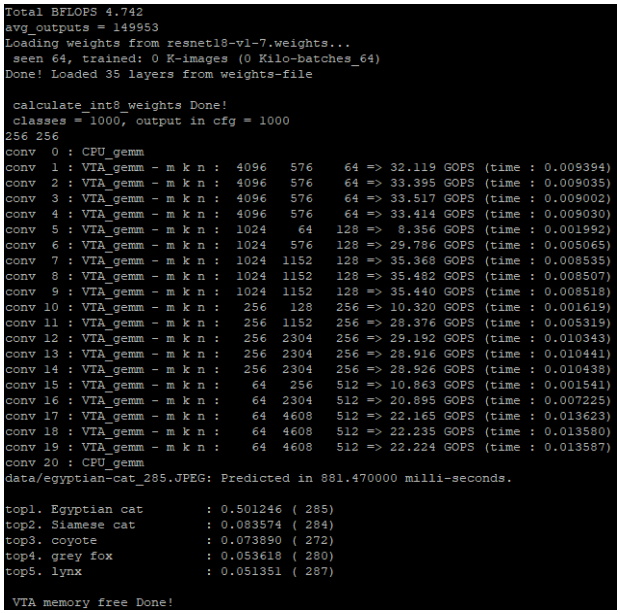

그림 10. Darknet 프레임워크에서 ResNet-18 모델의 수행 결과
Fig. 10. Results of ResNet-18 model in Darknet framework

3. 실험 결과 및 분석

표 2는 Xilinx ZCU102 보드의 CPU, VTA에서 행렬 곱셈 성능을 측정한 결과이다. CPU에서는 직접 컨볼루션 연산을 수행한 시간을 밀리 초 단위로 기록하였다. VTA 하드웨어의 성능 비교를 위해 2차 타일링 행렬 곱셈과 최적화된 2차 타일 행렬 곱셈의 성능을 비교하였다.

정방형 행렬의 크기가 2배씩 증가 됨에 따라 CPU의 연산 시간은 2^3 만큼 증가하였지만, VTA 하드웨어를 이용할 경우 기본 루틴에서 $2^{2.3}$배씩 증가, 최적화된 루틴에서 $2^{2.1}$배씩 증가하여 최적화된 루틴의 성능이 개선 되었음을 알 수 있다. 최적화 루틴은 비 최적화 루틴 대비 평균 2.6배 더 빨랐다.

비정방형 행렬은 CNN 레이어의 크기를 반영하므로 A, B 행렬의 크기가 일관되지 않으므로 행렬 곱셈 성능을 최대로 발휘되지 않는 경우들이 발생한다. 본 논문에서 제안한 기법은 비정방형 행렬의 경우에도 비최적화 루틴 대비 최적화 루틴의 성능이 평균 2.4배 더 개선되었음을 확인하였다.

그림 10은 DarkNet 프레임워크에서 ResNet-18 모델을 수행한 결과이다. 각 컨볼루션 레이어에서 평균 26 GOPS 이상의 성능을 보였다. 성능이 저하된 5, 10, 15번 레이어의 특징은 1x1 커널을 갖는 컨볼루션 레이어로써 연산량이 적은 부분으로써 기본적인 VTA 하드웨어를 구동하는 오버헤드가 오히려 크게 반영되어, 상대적으로 성능이 낮게 나오는 것 처럼 보인다. 하지만 실제 연산 수행시간은 CPU 대비 더 높은 성능을 보인다. DarkNet에서 제공하는 CPU 타

겟에서 OpenMP를 활성화 한 경우 ResNet-18은 평균 2.1초 이상의 추론 시간이 소요되었지만, VTA 타겟에서는 0.9초 미만의 추론시간을 보여 최소 2.3배 이상의 추론 성능 개선을 보였다.

V. 결 론

본 논문에서는 FPGA 기반 뉴럴네트워크 가속 회로인 VTA 에서 최적의 2차 타일링 행렬 곱셈 루틴을 제안하였다. 비 최적화 루틴의 빈번한 버퍼 메모리 교체로 인한 병목현상 해결을 위해 ACCUMULATOR 버퍼를 가득 채울 수 있도록 타일 크기를 수정하였다. 명령어의 버퍼 인덱스가 허용하는 범위 내에서 입력 버퍼와 가중치 버퍼가 최대한 재사용 되도록 루틴을 최적화하였다. 비 최적화 루틴 대비 제안하는 기법은 정방형 행렬에서 평균 2.6배, 비 정방형 행렬에서 2.4배 개선되었다. 또한 DarkNet 프레임워크에 제안하는 VTA 최적 행렬 곱셈 루틴을 구현하여 ResNet-18 모델에서 기존 CPU 대비 성능이 개선됨을 실험을 통해 보였다.

References

[1] J. W. Bae, B. G. Han, “Implementation of Deep Learning-based Label Inspection System Applicable to Edge Computing Environments,” IEMEK J. Embed. Sys. Appl, Vol. 17, No. 2, pp. 77-83, 2022 (in Korean).

[2] J. Y. Choi, H. J. Lee, C. W. Jeong, H. C. Jung, “Development of AI Service with Surgical Tools Segmentation and Action Recognition,” IEMEK J. Embed. Sys. Appl, Vol. 16, No. 2, pp. 51-57, 2021 (in Korean).

[3] H. D. Kim, “Design of Speech Enhancement U-Net for Embedded Computing,” IEMEK J. Embed. Sys. Appl, Vol. 15, No. 5, pp. 227-234, 2020 (in Korean).

[4] H. J. Kim, “Analysis of Reduced-Width Truncated Mitchell Multiplication for Inferences Using CNNs,” IEMEK J. Embed. Sys. Appl, Vol. 15, No. 5, pp. 235-242, 2020 (in Korean).

[5] J. M. Lee, M. S. Yu, Y. I. Kwon, T. H. Kim, "Quantune: Post-training Quantization of Convolutional Neural Networks using Extreme Gradient Boosting for fast Deployment," Future Generation Computer Systems 132, pp. 124-135. 2022.

[6] G. Y. Kwon, S. W. Park, T. W. Suh, “Cycle-accurate NPU Simulator and Performance Evaluation According to Data Access Strategies,” IEMEK J. Embed. Sys. Appl, Vol. 17, No. 4, pp. 217-228, 2022 (in Korean).

[7] https://coral.ai/products/dev-board

[8] https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html

[9] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, A. Krishnamurthy, "A Hardware - software Blueprint for Flexible Deep Learning Specialization," IEEE Micro, Vol. 39, No. 5, pp. 8-16, 2019.

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, "cudnn: Efficient Primitives for Deep Learning." arXiv preprint arXiv:1410.0759, 2014.

[11] https://www.openblas.net/

[12] https://github.com/Reference-LAPACK/lapack

[13] https://www.arm.com/technologies/compute-library

[14] https://github.com/Maratyszcza/NNPACK

[15] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, Y. Wang, "Intel Math Kernel Library," High-Performance Computing on the Intel® Xeon Phi™, Springer, Cham, pp 167-188, 2014.

[16] https://github.com/AlexeyAB/darknet

[17] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.

[18] A. Anderson, A. Vasudevan, C. Keane, D. Gregg, "Low-memory Gemm-based Convolution Algorithms for Deep Neural Networks," arXiv preprint arXiv:1709.03395, 2017.

[19] J. S. Park, K. M. Bin, K. H. Lee, "mGEMM: Low-latency Convolution with Minimal Memory Overhead Optimized for Mobile Devices," Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services, pp. 222-234, 2022.

[20] M .S. Cho, B. Daniel, "MEC: Memory-efficient Convolution for Deep Neural Network," International Conference on Machine Learning. PMLR, pp. 815-824, 2017.

[21] M. Dukhan, "The Indirect Convolution Algorithm." arXiv preprint arXiv:1907.02129, 2019.

[22] https://docs.nvidia.com/cuda/cublas/index.html

[23] C. Nugteren, "CLBlast: A Tuned OpenCL BLAS Library," Proceedings of the International Workshop on OpenCL, pp. 1-10, 2018.

[24] https://github.com/clMathLibraries/clBLAS

[25] K. Goto, V. D. G Robert, "High-performance Implementation of the Level-3 BLAS," ACM Transactions on Mathematical Software (TOMS) Vol. 35, No. 1, pp. 1-14, 2008.

Jinse Kwon (권 진 세)

2014 Mechanical Engineering from Chungnam National University (B.S.)

2017 Computer Engineering from Chungnam National University (M.S.)

2017~Computer Engineering from Chungnam National University (Ph.D. Student)

Field of Interests: Embedded AI Computing, Efficient AI, Embedded AI Acceleration

Email: kwonse@cnu.ac.kr

Jemin Lee (이 제 민)

2011 Computer Engineering from Chungnam National University (B.S.)
2017 Computer Engineering from Chungnam National University (Ph.D.)
2017~2018 Knowledge Service Engineering from KAIST(Post-Doc)

Career:
2018~Senior Researcher, AI SoC Research Division from ETRI
2019 Co-chair, ACM Mobisys
2022~Committee Member, IeMeK
Field of Interests: Mobile Computing, Efficient AI, AI Compiler
Email: leejaymin@etri.re.kr

Yongin Kwon (권 용 인)

2008 Electrical and Electronic Engineering from KAIST (B.S.)
2010 Electrical and Computer Engineering from Seoul National University (M.S.)
2015 Electrical and Computer Engineering from Seoul National University (Ph.D.)

Career:
2015~2019 Principal Researcher, System LSI, Samsung Electronics
2019~Senior Researcher, AI SoC Research Division from ETRI
Field of Interests: Mobile Cloud Computing, Compiler, Deep Learning, Embedded Systems
Email: yongin.kwon@etri.re.kr

Jeman Park (박 제 만)

2004 Electrical and Electronic Engineering from Hanyang University (B.S.)
2006 Electronics and Computer Engineering from Hanyang University (M.S.)
2014 Electronics and Computer Engineering from Hanyang University (Ph.D.)

Career:
2012~Senior Researcher, AI SoC Research Division from ETRI
Field of Interests: Computer Network, Mobil Edge Computing (MEC), AI Compiler
Email: jeman@etri.re.kr

Misun Yu (유 미 선)

1999 Computer Engineering from Chungnam National University (B.S.)
2002 Computer Engineering from Pohang University of Science and Technology (M.S.)

Career:
2002~Principal Researcher, AI SoC Research Division from ETRI
2018~Director of Neuromorphic Computing Software Platform for AI System
Field of Interests: Deep Learning Compiler, Neural Network Partitioning, Parallel Processing
Email: msyu@etri.re.kr

Taeho Kim (김 태 호)

1995 Information Engineering from Sungkyunkwan University (B.S.)
1997 Computer Science from KAIST (M.S.)
2005 Computer Science from KAIST (Ph.D.)

Career:
2001~2002 International Fellow, USA SRI CSL
2004~2005 Principal Researcher, Digital Printing Division, Samsung Electronics
2005~Assistant Vice President, AI SoC Research Division, ETRI
Field of Interests: AI SoC, Safety-critical, Intelligent CPS, Software Engineering, Artificial Intelligent, Code Generation
Email: taehokim@etri.re.kr

Hyungshin Kim (김 형 신)

1990 Computer Science from KAIST (B.S.)
1991 Satellite Communications Engineering from Univ. of Surrey (M.S.)
2003 Computer Science from KAIST (Ph.D.)
2004~Dept. of Computer Science and Engineering, Chungnam National Univ. (Professor)

Career:
1992 Researcher, SaTReC, KAIST
2003 Post Doc Researcher, CMU
Field of Interests: Real-time Embedded Software, Embedded AI Computing
Email: hyungshin@cnu.ac.kr