ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 20-02-27
188

j.inst.Korean.electr.electron.eng.Vol.24,No.2,559~569,june 2020
http://dx.doi.org/10.7471/ikeee.2020.24.2.559

CNN 가속기의 효율적인 데이터 전송을 위한 메모리 데이터 레이아웃 및 DMA 전송기법 연구

Memory data layout and DMA transfer technique research For efficient data transfer of CNN accelerator

조 석 재 *, 박 성 경* , 박 성 정**★

Seok-Jae Cho*, Sungkyung Park*, Chester Sungchung Park**★

Abstract

One of the deep-running algorithms, CNN's artificial intelligence application uses off-chip memory to store data on the Convolution Layer. DMA can reduce processor load at every data transfer. It can also reduce application performance degradation by varying the order in which data from the Convolution layer is transmitted to the global buffer of the accelerator. For basic layouts with continuous memory addresses, SG-DMA showed about 3.4 times performance improvement in pre-setting DMA compared to using ordinaly DMA, and for Ideal layouts with discontinuous memory addresses, the ordinal DMA was about 1396 cycles faster than SG-DMA. Experiments have shown that a combination of memory data layout and DMA can reduce the DMA preset load by about 86 percent.

요 약

딥 러닝 알고리즘 중 하나인 CNN 인공지능 어플리케이션은 하드웨어 측면에서 컨벌루션 레이어의 많은 데이터들을 저장하기 위해 오프 칩 메모리를 사용 하고, DMA를 사용하여 매 데이터 전송 시 프로세서의 부하를 줄여 성능을 향상 시킬 수 있다. 또한 컨벌루션 레이어의 데이터를 가속기의 글로벌 버퍼에 전송되는 순서를 다르게 하여 어플리케이션의 성능의 저하를 줄일 수 있다. 불 연속된 메모리 주소를 가지고 있는 베이직 레이아웃의 경우 SG-DMA를 사용 할 때 ordinary DMA를 사용할 때보다 DMA를 사전 설정하는 부분에서 약 3.4배의 성능향상을 보였고 연속적인 메모리 주소를 가지고 있는 아이디얼 레이아웃의 경우 ordinary DMA 와 SG-DMA를 사용하는 두가지 경우 모두 1396 사이클 정도의 오버헤드를 가졌다. 가장 효율적인 메모리 데이터 레이아웃과 DMA의 조합은 프로세서의 DMA 사전 설정 부하를 약 86 퍼센트까지 감소할 수 있음을 실험을 통해 확인했다.

Key words : CNN, memory data layout, Loop-tiling, Accelerator, Scatter-Gather DMA

* Dept. of Electronics Engineering, Pusan National University
** Dept. of Electronics Engineering, Konkuk University
★ Corresponding author
E-mail : chester@konkuk.ac.kr Tel : +82-2-450-0539

※ Acknowledgment
This work was supported by a 2-Year Research Grant of Pusan National University
Manuscript received Jun. 1, 2020; revised Jun. 14, 2020; accepted Jun. 22, 2020.

Ⅰ. 서론

1. 연구목적

최근 딥 러닝(Deep learning)알고리즘을 중점으로 둔 빠르게 발전하고 있는 기계학습 분류 알고리즘은 기존의 방법들보다 뛰어난 성능으로 인하여 주목받고 있다.

딥 러닝은 인간의 정보 처리 방식을 기계에 적용한 인공 신경망을 기반으로 대량의 데이터를 컴퓨터가 스스로 학습하는 기계학습 알고리즘의 분야이다. 수많은 딥 러닝 알고리즘 중 하나인 CNN

(Convolution Neural Network)은 현재 컴퓨터 비전, 음성 인식, 로봇 공학을 포함하는 인공지능 어플리케이션에 널리 쓰이고 있다[1], [2], [3], [4], [5].

CNN은 MAC(Multiply and Accumulation) 연산으로 구성된 연산 집약적인 컨벌루션 레이어 와 모든 레이어들이 연결된 풀리 커넥티드 레이어로 분류 할 수 있고, CNN을 사용한 인공지능 어플리케이션의 성능은 데이터의 흐름과 하드웨어의 구성에 크게 좌우된다.

하드웨어 구성에서 고차원의 컨벌루션 레이어를 처리할 때 생기는 수 많은 데이터 들을 처리하기 위해 저장 용량이 적은 온 칩 메모리를 사용하기보다, 저장 용량이 많은 오프 칩 메모리를 사용하는 것을 선호한다.

하지만 범용의 프로세서를 사용할 경우 오프 칩 메모리로의 잦은 접근으로 인하여, 계산 속도나 에너지 소모 측면에서 어플리케이션의 성능 저하로 이어지게 된다[6], [7], [8].

이를 해결하기 위해 매 데이터 전송 시 프로세서가 관여하는 부하를 DMA(Direct Memory Access)를 사용하여 계산 속도나 에너지 소모 측면에서 어플리케이션의 성능의 저하를 줄인다.

또한, 고차원의 컨벌루션 레이어를 처리하기 위해 발생하는 수 많은 데이터들의 이동으로 인한 높은 에너지 소비량을 개선하기 위해, 데이터를 다양한 방식으로 재사용하는 대표적인 데이터플로우 기법들이 있다. 대표적인 데이터 플로우 기법들은 WS(Weight Stantionary) [9], [10], OS(Output Stantionary) [11], [12], RS(Row Stantionary) [13], [14], NLR(No Local Reuse) [15], [16], [17] 기법이 있다.

본 논문에서는 NLR 데이터플로우를 사용하고 두 종류의 DMA의 성능을 비교/분석하여, 오프 칩 메모리의 접근을 최소화하는 적합한 메모리 데이터 레이아웃과 두 종류의 DMA의 조합을 기술한다.

본 논문의 구성은 다음과 같다. 2장은 기본적인 DMA 동작 방식, CNN 구조 및 CNN의 매개변수, 메모리 데이터 레이아웃에 대하여 살펴본다. 3장에서는 Xilinx Zybo board에서 AlexNet과 DMA 그리고 하드웨어 가속기를 구현하여 시뮬레이션 실험 및 결과를 제시한다. 마지막으로 4장에서는 결론, 연구가 갖는 한계점 및 향후 연구에 대해 기술한다.

2. 관련연구

컨볼루션 뉴럴 네트워크의 루프가 중첩되는 부분을 루프 언롤링 및 타일링과 같은 다양한 최적화 기법을 사용하여 컴퓨팅 처리량과 필요한 메모리 대역폭을 정량적으로 분석한 논문들이 있다[17], [20], [23], [24]. [23]. 참조된 문헌들은 데이터플로우 기법에 의해 제약되는 PE(Processing Elements)들의 개 수 및 PE의 효율에 초점은 둔다.

앞서 제시한 논문들은 다양한 데이터플로우 기법에 대해, 가속기의 성능 지표를 추정하지만, DMA와 오프 칩 디램에서 가속기로 전달되는 데이터들의 형태, 즉 가속기 내부의 글로벌 버퍼로 데이터가 전송되는 형태인 메모리 데이터 레이아웃과 어플리케이션 성능의 관계를 고려하지 않는다.

또한 MAC 연산을 위해 필요한 데이터들이 오프 칩 디램에 연속된 메모리 주소를 가지고 있다고 가정하므로, ordinary DMA를 사용하여 데이터를 가속기의 글로벌 버퍼에 공급하는 것만 고려하고 있다.

이 논문은 기존의 논문들과 다르게 오프 칩 디램에 저장되어 있는 연속된 메모리 주소에 데이터를 가속기의 글로벌 버퍼에 공급하는 것 뿐만 아니라 연속되지 않은 메모리 주소에 데이터를 가속기에 공급하는 방식과 데이터를 가속기에 전달하는 방법에 따른 적절한 DMA 선택과 메모리 데이터 레이아웃의 조합을 제안한다.

II. 본론

1. DMA

프로세서가 매번 오프 칩 디램의 데이터들을 가속기의 글로벌 버퍼에 전달하기 위해 컨트롤러에 명령어를 보내는 프로세서의 빈번한 인터럽트의 발생을 줄여 전체 시스템의 성능을 높이기 위해 가속기가 직접 메모리의 데이터를 읽고 쓸 수 있도록 DMA가 사용된다.

DMA는 크게 3가지의 동작 방식으로 나뉜다. 전송할 데이터의 메모리 주소와 데이터가 전송될 메모리 주소를 설정하는 DMA set 동작과, 데이터가 전송될 메모리 주소로 데이터를 전송하는 DMA run과 DMA의 작동 여부를 확인하는 busy check 동작 방식으로 나뉜다.

가. ordinary DMA

ordinary DMA는 오프 칩 디램에 있는 컨피규레이션 정보(연속적인 데이터의 시작주소와 연속적인 데이터 길이 정보)를 프로세서로부터 받아 가속기의 글로벌 버퍼와 오프 칩 디램 사이의 데이터 전송을 시작한다.

ordinary DMA는 오프 칩 디램의 데이터가 만약 N번의 불연속적인 주소를 가지고 있다면 N개의 컨피규레이션 정보가 필요하다.

나. Scatter-Gather DMA

ordinary DMA가 연속된 메모리 공간의 데이터 전송만을 지원한다면 SG-DMA는 좀 더 발전해 연속적인 메모리 공간에서 불연속적인 메모리 공간으로 데이터를 이동시키는 Scatter 기능과 불연속적인 메모리 공간에서 연속적인 메모리 공간으로 데이터를 이동시키는 Gather 기능이 있다.

그림 1은 SG-DMA의 Scatter Gather 기능을 나타낸 그림이다. 연속된 주소의 메모리의 데이터를 불연속적인 주소의 메모리에 데이터를 흩뿌리는 과정을 Scatter 과정이라고 하고 불연속된 메모리의 데이터를 연속된 메모리 주소에 데이터를 전송하는 과정을 Gather라고 한다.

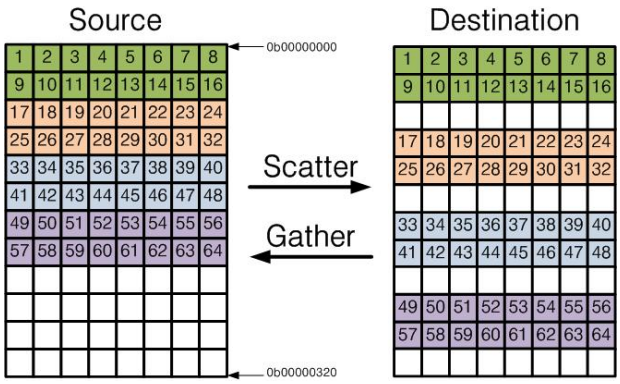


Fig. 1. Scatter Gather Function description of SG-DMA.
그림 1. SG-DMA의 Scatter Gather 기능

SG-DMA는 불연속적인 데이터를 처리하기 위해 미리 버퍼 디스크립터에 불연속적인 데이터의 컨피규레이션 정보를 기록하여 사용한다[18].

만약 N개의 불연속적인 데이터 주소를 가지는 데이터가 있다면 N개의 버퍼 디스크립터를 필요로 하고 각 N개의 버퍼 디스크립터에는 대응되는 불연속적인 데이터의 시작주소와 데이터의 길이 정보가 기록된다.

SG-DMA는 ordinary DMA와 다르게 SG port 포트를 통해 버퍼 디스크립터를 읽어 접근할 데이터 정보를 확인한다. 그 후 확인한 컨피규레이션 정보를 이용해 데이터 전송을 시작한다.

SG-DMA는 ordinary DMA와 비교해서 버퍼 디스크립터 테이블을 이용하여 한 번의 컨피규레이션 정보로 불연속적인 데이터를 처리할 수 있지만 버퍼 디스크립터 테이블을 작성하고 버퍼 디스크립터를 읽어오는 오버헤드가 발생한다.

2. AlexNet

CNN의 대표적인 하드웨어 아키텍처인 AlexNet은 8개의 레이어들로 구성되어 있으며, 5개의 컨벌루션 레이어와 3개의 풀리 커넥티드 레이어로 구성되어 있다.

두 번째, 네 번째, 다섯 번째 컨벌루션 레이어 들은 전 단계의 같은 채널의 특성 맵들만 연결되어 있는 반면, 세 번째 컨볼루션 레이어는 전 단계의 두 채널의 특성 맵들과 모두 연결되어 있다.

이중 세 번째 컨벌루션 레이어가 384개의 3×3×256 커널을 사용하여 전 단계인 두 번째 특성 맵을 컨볼루션 해준다.

8개의 컨벌루션 레이어중 세 번째 컨벌루션 레이어가 네 번째 컨벌루션 레이어로 전송하는 데이터가 제일 많기 때문에 세 번째 레이어의 메모리 데이터 레이아웃을 최적화 해줌 으로서 큰 성능향상을 기대 할 수 있다.

그림 2와 그림 3은 컨벌루션 레이어의 기본적인 형태를 표현하고 매개변수들을 슈도코드로 기술 한 것이다[17].

인풋 특성 맵(Input Feature map)데이터는 3차원으로 구성되며, 높이(H), 너비(W) 및 입력 채널(C)을 가진다. C는 인풋 채널의 수를 나타내고 C값은 특성 맵(Weights) 데이터의 인풋 채널 수와 같다.

특성 맵 데이터는 4차원으로 구성되며, 높이(R), 너비(S), 입력 채널(C) 및 출력 채널(M)을 가진다. 3D 특성 맵의 개수(M)는 아웃풋 특성 맵(Output feature map) 데이터의 아웃풋 채널의 수와 같다.

아웃풋 특성 맵 데이터는 3차원으로 구성되며, 높이(E), 너비(F) 및 출력 채널을 가진다. 루프의 순서를 바꾸더라도 동일한 값을 가지는 아웃풋 특성 맵 데이터가 생성된다.

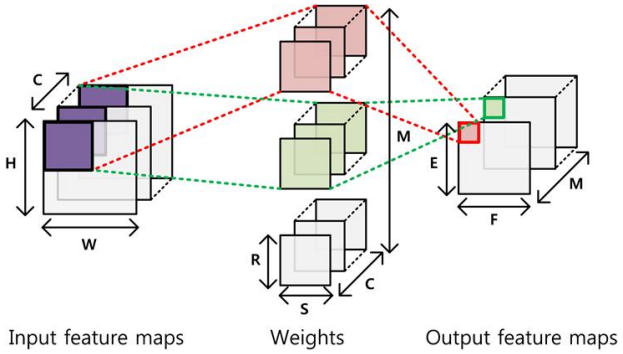


Fig. 2. Structure of a Convolution layers.
그림 2. 컨벌루션 레이어의 기본 구조

```
short IAs[C][H*W];
short OAs[M][E*F];
short Ws[M][C][R*S];
short wx, ix;
for (m = 0; m < M; m++) {
  for (c = 0; c < C; c++) {
    for (e = 0; e < E; e++) {
      for (f = 0; f < F; f++) {
        for (r = 0; r < R; r++) {
          for (s = 0; s < S; s++) {
            wx = Ws[m][c][r*S+s];
            ix = IAs[c][e*H+s+f];
            OAs[m][e*F+f] += wx*ix;
} } } } } }
```

Fig. 3. Basic Pseudo Code of a Convolution layers.
그림 3 컨벌루션 레이어의 기본 슈도코드

3. 루프 타일링

루프 타일링은 루프가 중첩되는 부분에서 데이터가 접근 할 수 있는 spatical locality 활용하고 temporal locality를 효율적으로 활용 할 수 있는 루프 변환 기법의 하나이다. 이 개념을 가속기에 적용해본다면 루프 타일링 요소는 가속기 내부 글로벌 버퍼의

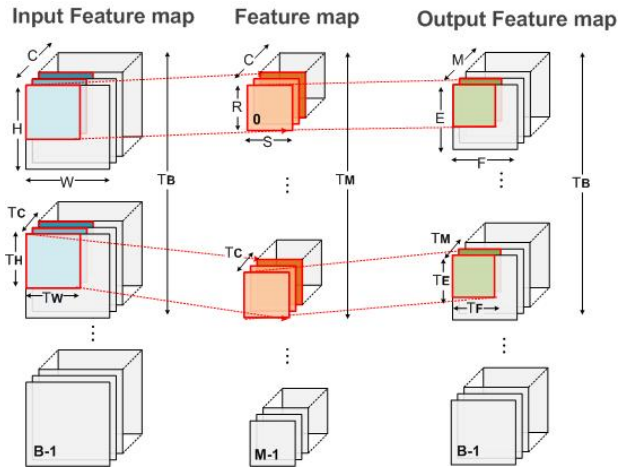


Fig. 4. Structure of a Tiled Convolution layers.
그림 4. 타일링 된 컨벌루션 레이어의 기본 구조

```
I[N][(R-1)*S+K][(C-1)*S+K]//input map
O[M][R][C] //output map
W[M][N][K][K]
Ibuf[Tn][(Tr-1)*S+K][(Tc-1)*S+K]
Obuf[Tm][Tr][Tc]
Wbuf[Tm][Tn][K][K]
for (r = 0; r < R; r+=Tr) {
  for (c = 0; c < C; c+=Tc) {
    for (m = 0; m < M; m+=Tm) {
      for (n = 0; n < N; n+=Tn) {
      irx=r*S:(r+Tr-1)*S+K
      icx=c*S:(c+Tc-1)*S+K
      Ibuf=I[n:n+Tn][irx][icx]
      Wbuf=W[m:m+Tm][n:n+Tn]
      for (i = 0; i < K; i++) {
        for (j = 0; j < K; j++) {
          for (tr = 0; tr+r<min(R,r+Tr); tr++) {
            for (tc = 0; tc+c<min(C,c+Tc); tc++) {
              for (tm = 0; tm<Tm; tm++) #UNROLL{
                for (tn = 0; tn<Tn; tn++) #UNROLL{
                  wx = Wbuf[tm][tn][i][j];
                  ix = Ibuf[tn][S*tr+i][S*tc+j];
                  Obuf[tm][tr][tc] += wx*ix;
} } } } } }
```

Fig. 5. Pseudo Code of a Convolution layer.
그림 5. 타일링 된 컨벌루션 레이어의 슈도코드

읽기 쓰기 당 전송되는 데이터양과 데이터가 전송되는 순서를 제어한다.

그림 4와 그림 5에서 볼 수 있듯 루프 파라미터 R, C, M 및 N은 각각 매개변수 Tr, Tc, Tm 및 Tn으로 타일링 된다. Tm과 Tn을 기반으로 하는 가장 안쪽 루프는 연속되기 때문에 루프 타일링은 계산 모듈의 구성 방식도 제어하게 된다[17].

3. 메모리 데이터 레이아웃

메모리 데이터 레이아웃은 오프 칩 디램에 저장되어 있는 컨벌루션 특성 맵 데이터들의 픽셀들의 순서를 임의의 순서대로 가속기의 글로벌 버퍼에 전송하는 방법을 나타낸다.

가. 베이직 레이아웃

베이직 레이아웃은 특성 맵 데이터 들의 픽셀 순서로 가속기의 글로벌 버퍼 영역에 특성 맵 데이터가 저장되는 방식이다.

예를 들어 그림 6의 인풋 특성 맵 데이터의 타일 매개변수들은 TileH = TileW = 4이다. 즉 오프 칩 디램으로부터 가속기의 글로벌 버퍼에 전송되는 픽셀들은 사각형으로 표현된 영역이다.

첫 번째 타일의 인풋 특성 맵 데이터들은 메모리에 4개의 픽셀씩 메모리 주소가 바뀌는 형태로 저장되어 있기 때문에 ordinary DMA는 한 번의 컨

피규레이션 정보로 픽셀들을 가속기의 글로벌 버퍼로 전송 할 수 없다.

즉 ordinary DMA가 베이직 레이아웃에서 필요한 컨피규레이션 정보는 4개로 TileH와 같다.

하지만 SG-DMA는 불연속적인 메모리 주소의 데이터들을 연속적인 메모리 주소의 데이터를 gather 기능을 사용하여 전송 할 수 있다.

즉 4개의 버퍼 디스크립터를 사용하여 한 번의 컨피규레이션 정보로 데이터 전송을 처리할 수 있다. SG-DMA가 베이직 레이아웃에서 필요한 버퍼 디스크립터의 개 수는 TileH와 같다.

그림 6은 인풋 특성 맵 데이터가 오프칩 디램에 픽셀 순서대로 저장 되어 있는 모습을 모략화 한 그림이다.

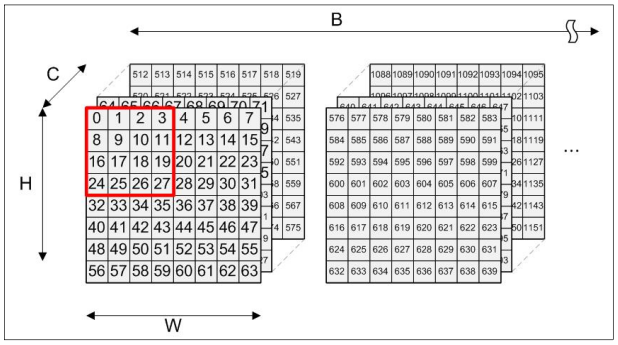


Fig. 6. Basic layout of Input Feature map.
그림 6. 인풋 특성 맵의 베이직 레이아웃

그림 7은 가속기의 글로벌 버퍼에 데이터들이 픽셀 순서대로 저장되는 모습을 나타내었다.

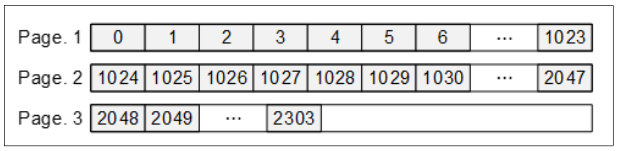


Fig. 7. Basic layout of Input Feature map data in global buffer.
그림 7. 글로벌 버퍼에 저장 될 인풋 특성 맵 데이터의 베이직 레이아웃

```
for(i=0;i<No.if_pixel;i++){
  Mem[if_base_addr++] = Ifmap[i++];
}
```

Fig. 8. Pseudo Code of Basic layout.
그림 8. 베이직 레이아웃의 슈도코드

그림 8은 베이직 레이아웃의 슈도코드를 나타내었다. i가 증가함에 따라 픽셀의 순서대로 오프 칩 디램의 데이터가 가속기의 글로벌 버퍼에 저장되는 모습을 볼 수 있다.

나. 아이디얼 레이아웃

아이디얼 레이아웃은 오프 칩 디램에 저장된 픽셀들을 타일 단위로 가속기의 글로벌 버퍼에 연속적이게 저장하는 방식이다.

한 타일 안의 픽셀들의 연속적인 메모리 주소에 저장하기 위해 픽셀의 데이터 순서를 재정렬하는 과정이 필요하다.

한개의 컨피규레이션 정보로 ordinary DMA가 한 타일 안의 모든 픽셀 데이터들을 오프 칩 디램에서 가속기의 글로벌 버퍼로 전송 할 수 있다.

만약 타일링 된 컨벌루션 레이어의 슈도코드에서 타일 매개변수가 Tc = 2, Th = 5, Tw = 5 라면 그림 9의 사각형만큼의 인풋 특성 맵 데이터가 타일링 된다.

그림 10은 타일링 된 만큼의 인풋 특성 맵 데이터가 픽셀들의 순서에 관계 없이 메모리에 저장되는 모습을 보여준다. 즉 가속기의 글로벌 버퍼에 저장되기 전 데이터를 재정렬하는 과정이 필요하다.

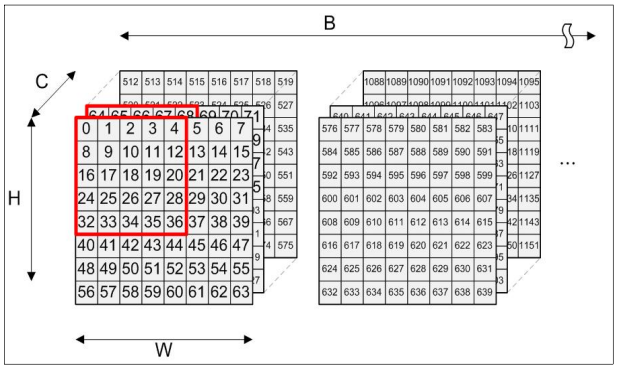


Fig. 9. Ideal layout of Input Feature map data.
그림 9. 인풋 특성 맵 데이터의 아이디얼 레이아웃

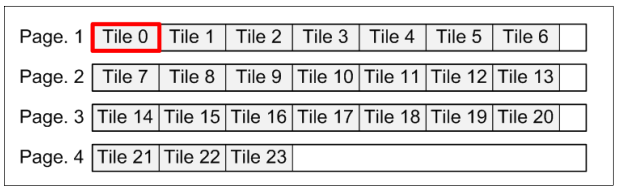


Fig. 10. Ideal layout of Input Feature map data in global buffer.
그림 10. 글로벌 버퍼에 저장될 인풋 특성 맵 데이터의 아이디얼 레이아웃

그림 11은 아이디얼 레이아웃의 슈도코드를 나타내었다. 타일링 된 만큼의 인풋 특성 맵 데이터가 픽셀 순서에 관계 없이 저장되는 것을 볼 수 있다.

```
for(th=0;th<TileH;th++){
  for(tw=0;tw<TileW;tw++){
    Mem[if_base_addr++] = Ifmap[th*W+ tw];
} }
```

Fig. 11. Pseudo Code of Ideal layout.
그림 11. 아이디얼 레이아웃의 슈도코드

4. 실험 환경

그림 12은 가장 데이터 전송량이 많은 AlexNet 레이어 3에 NLR 데이터플로우를 적용한 가속기와 인풋 특성 맵, 특성 맵 데이터를 읽어 오기 위한 DMA 2개, 아웃풋 특성 맵 데이터를 다시 메모리에 쓰기 위한 DMA 1개를 사용하는 실험 환경의 하드웨어 구조를 나타내었다.

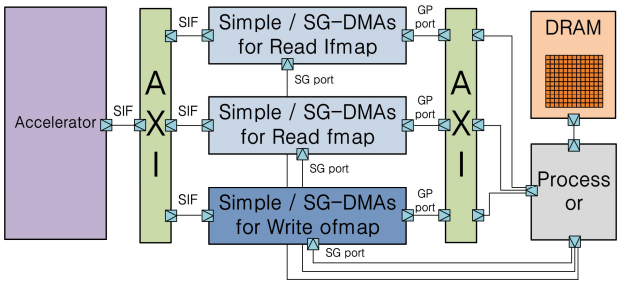


Fig. 12. hardware block diagram.
그림 12. 하드웨어 블록 다이아 그램

Table 1은 실험 환경의 컨벌루션 레이어의 매계변수 설정 값을 나타 내었다.

Table 1. Experiment Environment of CNN parameter.
표 1. 실험 환경의 CNN 매계변수 설정 값

Experiment Environment of CNN parameter	
Output feature map channel (M)	384
Input feature map Channel (C)	256
Input Feature map size (H x W)	15 x 15
Feature map size (R x S)	3 x 3
Output feature map size (E x F)	13 x 13
Tiled Output feature map channel (Tile M)	64
Tiled Input feature map channel (Tile C)	2
Tiled Output feature map (Tile E x Tile F)	13 x 13
Tiled Input feature map (Tile H x Tile W)	15 x 15

멀티 DMA를 사용하지 않고 싱글 DMA를 사용하는 이유는 AXI 버스에서의 데이터 전송 효율을 최대화하기 위해 싱글 채널 DMA를 사용하였다.

프로세서는 오프 칩 디램 속의 픽셀들을 가속기의 글로벌 버퍼로 전달하기 위해 AXI 프로토콜을 사용한다.

프로세서는 GP(General Perpose) 포트를 통해 DMA 컨트롤러를 제어하고 SG-DMA 사용 시 버퍼 디스크립터가 필요한 경우 SG-port를 통해 불연속적인 메모리 주소 정보들을 DMA에 전송한다. 가속기와 DMA들은 AXIS(AXI stream Interface) 포트를 통해 데이터를 오프 칩 디램에 읽고 쓰게 된다.

이 논문에서는 2가지의 메모리 데이터 레이아웃과 2가지의 DMA를 조합한 총 4가지 환경을 적용하여 실험을 진행하였다.

메모리 데이터 레이아웃에 따라 적합한 DMA의 종류를 선택하기 위해, DMA의 burst length는 4로 동일하게 설정하고, PL 영역의 동작 주파수는 100 MHz로 설정한다. 가속기를 위한 타일 매개변수의 경우, Tm은 64, Tc는 2, Th와 Tw는 15, Te와 Tf는 13으로 설정한다.

5. 실험 결과

베이직 레이아웃과 ordinary DMA의 조합의 경우 오프 칩 디램에는 인풋 특성 맵 데이터, 특성 맵 데이터가, 아웃풋 특성 맵 데이터가 저장되어 있다. 가속기의 글로벌 버퍼는 오프 칩 디램에 저장 되어 있는 특성맵 데이터들에 접근하기 위해 ordinary DMA를 사용한다. 각 ordinary DMA는 불연속적인 데이터로의 접근을 위해 다수의 컨피규레이션 정보가 요구된다.

AlexNet의 컨벌루션 레이어 3을 처리할 때, ordinary DMA의 경우, 인풋 특성 맵 데이터는 픽셀의 열(H)이 바뀔 때마다 오프 칩 디램에 저장된 데이터의 메모리 주소가 불연속적이므로 총 Th × Tc 번, 즉 30번의 DMA set과 busy check를 반복한다.

특성 맵 데이터의 경우, 서로 다른 아웃풋 채널(M)을 처리하게 될 때마다 인풋 특성 맵 데이터처럼 데이터의 메모리 주소가 불연속적이므로 총 Tm 번, 즉 64번의 DMA set과 busy check를 반복한다.

아웃풋 특성 맵 데이터의 경우, 픽셀의 열(E)의 순서가 바뀔 때마다 오프 칩 디램에 저장될 메모리 주소가 불연속적이므로 총 Te 번, 즉 13번의 DMA set과 busy check를 반복한다.

ordinary DMA를 set 하기 위해서, 프로세서는 컨피규레이션 정보를 DMA 내부 컨트롤 레지스터와 상태 레지스터에 쓰는데 약 78 사이클이 소요된다.

그림 13은 프로세서가 DMA들에게 오프 칩 디램의 특성 맵 데이터를 베이직 레이아웃의 형태로 가속기의 글로벌 버퍼에 전송하기 위해 DMA를 set하고 DMA가 동작하는 모습을 타이밍 다이어그램으로 나타내었다. 특성 맵 데이터의 Tm 번의 픽셀 전송 후 프로세서는 busy check를 시작하게 된다.

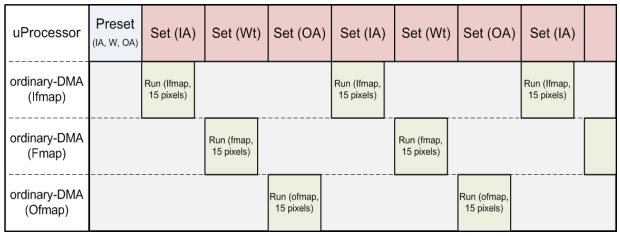


Fig. 13. basic layout timing diagram with ordinary DMA.
그림 13. ordinary DMA를 사용한 베이직 레이아웃의 타이밍 다이어 그램

DMA가 오프 칩 디램으로부터 가속기의 글로벌 버퍼로 정해진 데이터의 길이만큼을 전송한 다음, 프로세서는 DMA가 동작을 제대로 마쳤는지 확인하기 위해, DMA 내부의 상태 레지스터를 통해 DMA가 작동되고 있는지 확인을 하는데, 이를 위해 약 18 사이클이 소요된다.

따라서 베이직 레이아웃의 경우, ordinary DMA는 오프 칩 디램에 불연속적인 주소로 저장 되어있는 인풋 특성 맵 데이터, 특성 맵 데이터를 전송하기 위해 107번의 DMA set과 busy check를 반복하므로, 약 10,272 사이클의 오버헤드를 가진다.

베이직 레이아웃과 SG-DMA의 경우 가속기의 글로벌 버퍼는 오프 칩 디램의 데이터에 접근하기 위해 SG-DMA를 사용한다. 각 SG-DMA는 불연속적인 주소를 가지고 있는 특성 맵 데이터로의 접근을 위해 버퍼 디스크립터 테이블을 사용하고 한 번의 컨피규레이션 정보가 요구된다.

이 때, 데이터를 어디에 써야하는 정보를 담당하고 있는 T×BD(Transmit Data BD)를 생성하는데 소요되는 오버헤드는 약 780 사이클이다. SG-DMA가 데이터를 어디서 얼마만큼의 읽어야 하는지에 대한 정보를 T×BD에 기술하는데 소요되는 오버헤드는 약 670 사이클이다.

마찬가지로 R×BD(Receive Data BD)를 생성하는데 소요되는 오버헤드는 약 610 사이클이다. SG-DMA가 어디에서 읽어 올지에 대한 정보를 R×BD에 기술하는데 소요되는 overhead는 약 850 사이클이다.

SG-DMA가 데이터를 모두 전송한 후, busy check는 약 80 사이클 소요된다.

따라서 베이직 레이아웃의 경우, SG-DMA가 인풋 특성 맵 데이터, 특성 맵 데이터, 아웃풋 특성 맵 데이터 전송하기 위해 버퍼 디스크립터를 set하고 DMA의 busy 상태를 체크 하는데 약 2,990 사이클의 overhead를 가진다.

그림 14는 프로세서가 DMA들에게 오프 칩 디램의 특성 맵 데이터 들을 베이직 레이아웃 형태로 가속기의 글로벌 버퍼에 전송하기 위해 DMA를 set하고 DMA가 동작하는 모습을 타이밍 다이어그램에 나타내었다. 데이터의 전송이 끝날 때 부터 각 특성 맵 별로 프로세서가 각 DMA에 busy check를 시작한다.

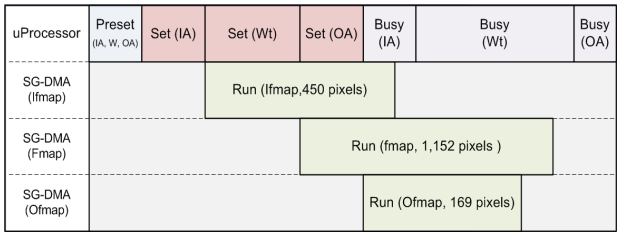


Fig. 14. basic layout timing diagram with SG-DMA.
그림 14. SG-DMA를 사용한 베이직 레이아웃의 타이밍 다이어그램.

아이디얼 레이아웃과 ordinary DMA의 경우 가속기의 글로벌 버퍼는 오프 칩 디램의 데이터에 접근하기 위해 ordinary DMA를 사용한다.

따라서 가속기가 컨벌루션 레이어를 처리하기 전 인풋 특성 맵의 데이터를 순서대로 재정렬하는 과정과 컨벌루션 레이어의 특성 맵들을 연산한 후에 아웃풋 특성 맵 데이터를 재정렬하는 과정이 필요하다. 즉 ordinary DMA는 연속적인 데이터로 접근하므로 한 개의 컨피규레이션 정보가 요구된다.

아이디얼 레이아웃과 SG-DMA의 경우 각 SG-DMA는 ordinary DMA와 마찬가지로 연속적인 주소를 가진 데이터로의 접근을 위해 버퍼 디스크립터 테이블을 사용하고 한 번의 컨피규레이션 정보가 요구된다.

두 조합 모두 오프 칩 디램에서 데이터가 이미 재정렬 되어있기 때문에 DMA를 set 하는 과정에

서 약 1300 사이클이 소모되며, 데이터 전송이 끝난 후 busy check로 80 사이클이 소모 된다.

그림 15와 그림 16은 프로세서가 DMA 들에게 오프 칩 디램의 특성 맵 데이터 들을 아이디얼 레이아웃 형태로 가속기의 글로벌 버퍼에 전송하기 전, 오프칩 디램 내부의 특성 맵 데이터들의 형태를 재정렬하고, DMA를 set과 DMA가 동작하는 모습을 타이밍 다이어그램에 나타내었다. 데이터의 전송이 끝날 때부터 각 특성 맵 별로 프로세서가 각 DMA에 busy check를 시작한다.

연속된 메모리 주소(오프 칩 디램)에서 연속된 메모리(가속기 내부의 글로벌 버퍼)로 데이터를 전송하는 것이므로 두 DMA는 같은 데이터 전송 방식을 보여준다.

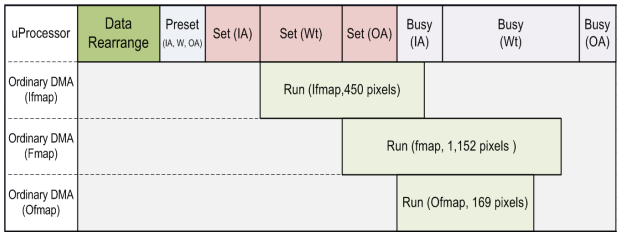


Fig. 15. Ideal layout timing diagram with ordinary DMA.
그림 15. ordinary DMA를 사용한 아이디얼 레이아웃의 타이밍 다이어그램.

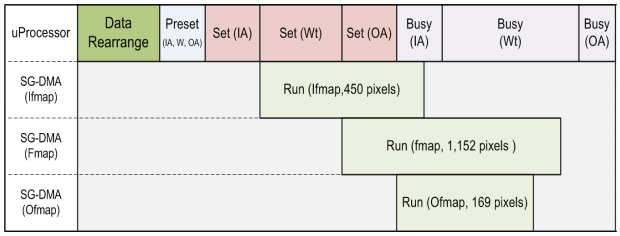


Fig. 16. Ideal layout timing diagram with SG-DMA.
그림 16. SG-DMA를 사용한 아이디얼 레이아웃의 타이밍 다이어그램

컨벌루션 레이어의 구조나 다른 CNN 하드웨어 아키텍처에 따라 적합한 메모리 데이터 레이아웃과 DMA의 조합은 달라질 수 있다.

최근 CNN의 하드웨어 아키텍처는 기존 모델을 바탕으로 정확성을 증가시키기 위해 트레이드 오프 관계인 CNN 하드웨어 아키텍처의 구성의 일부를 복잡하게 바꾸는 방법을 사용하는 경향이 있다.

그림 17은 CNN 하드웨어 아키텍처의 대표적인 모델 스케일링 방법을 보여준다. (a)의 baseline model에 (b)-(d) 모두 컨벌루션 레이어에 오직 하나의 차원을 증가시키는 방법으로 CNN 하드웨어 아키텍처 모델을 스케일링 한다. 즉 CNN 컨버루션 레이어의 기본 모델은 크게 차이가 없다.

예를 들어 대표적인 CNN 하드웨어 아키텍처인 ResNet [24]은 망의 깊이 즉 컨벌루션 레이어의 개수를 늘리는 depth scaling을 통해 CNN 하드웨어 아키텍처의 정확성을 높인다.

MobileNet [25] 및 ShuffleNet [26]은 특성 맵 데이터의 개수를 늘리는 Width scaling을 통해 채널의 크기를 조절하여 CNN 하드웨어 아키텍처의 정확성을 높인다. with를 제어하는 모델은 대게 작은 크기의 모델들이며 width를 넓게 늘릴수록 미세한 정보 (fine-grained feature)를 더 많이 담을 수 있다.

EfficientNet [27] 같은 경우 width scaling, depth scaling과 인풋 이미지의 해상도를 높이는 resolution scaling 3가지의 방법을 동시에 사용하여 CNN 하드웨어 아키텍처의 정확성을 높인다.

즉 CNN 하드웨어 아키텍처의 구성은 CNN 하드웨어 아키텍처 모델을 변경하더라도 크게 바뀌지 않으며, 아이디얼 레이아웃의 메모리맵 데이터 레이아웃의 재정렬 시간은 인풋 특성 맵 데이터의 크기에 따라 베이직 레이아웃 보다 큰 오버헤드를 가질 것이며, 아이디얼 레이아웃에서 각 ordinary DMA와 SG-DMA의 전송 오버헤드는 비슷 할 것이다.

아이디얼 레이아웃의 경우 ordinary DMA와 SG-DMA의 성능은 거의 동일 하다.

하지만 베이직 레이아웃의 경우, 컨벌루션 레이어의 픽셀 데이터를 재정렬하는 과정을 따로 수행하지는 않기 때문에 ordinary DMA가 오프 칩 디램에서 불 연속적인 메모리 주소를 가진 데이터를 읽기 위한 오버헤드가 발생하게 된다.

ordinary DMA는 베이직 레이아웃으로 저장된 데이터를 읽어 오기 위해, 불 연속적인 메모리 주소가 변경 될 때 마다 DMA 전송을 수행해야 한다. 이로 인해, 하드웨어 내부의 버스 효율이 감소한다.

하지만 SG-DMA가 오프 칩 디램에 베이직 레이아웃으로 저장된 데이터를 읽기 위해서는 ordinary DMA와는 달리 오프 칩 디램에 저장 된 데이터가 불 연속적이기 이라도 한 번 이상의 DMA의 데이터 전송을 추가로 수행할 필요는 없다. 하지만, 데이터 전송을 위한 버퍼 디스크립터를 사전 설정하는 오버헤드가 발생한다.

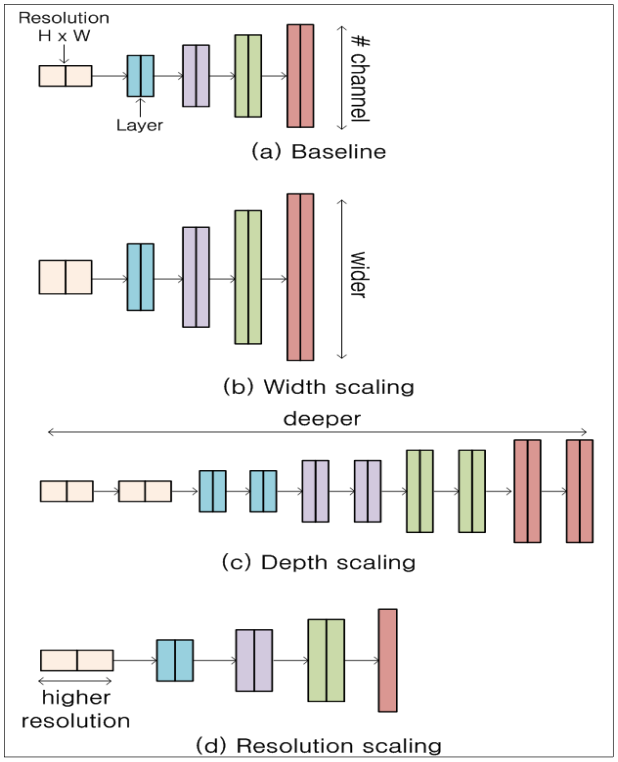


Fig. 17. Model scailing method of CNN hardware architecture.
그림 17. CNN 하드웨어 아키텍쳐의 모델 스케일링 방법

즉 ordinary DMA는 불 연속적인 메모리 주소의 변경에 따른 DMA 전송 횟수가 추가되는 오버헤드(DMA set & DMA busy check)가 더 큰지, SG-DMA의 BD 사전설정으로 인한 오버헤드가 큰지에 따라 베이직 레이아웃에 적합한 DMA가 결정된다.

Table 2. Comparison of DMA transfer overhead.
표 2. DMA 전송 오버헤드 비교

	Basic layout with ordinary DMA	Basic layout with SG-DMA	Ideal layout with ordinary DMA	Ideal layout with SG-DMA
Data rearrange	Not rearranged	Not rearranged	Rearranged	Rearranged
Data address in off-chip DRAM	Not continuous	Not continuous	Continuous	Continuous
DMA set (Cycle)	8,346 cycles	2,910 cycles	1,316 cycles	1,316 cycles
DMA busy check (Cycle)	1,926 cycles	80 cycles	80 cycles	80 cycles
Total DMA transfer overhead (Cycle)	10,272 cycles	2,990 cycles	1,396 cycles	1,396 cycles

III. 결론

AlexNet의 컨벌루션 레이어 3에 대해 실험한 결과, 베이직 레이아웃을 ordinary DMA가 처리하는 것이 가장 부적합하며, 빈번한 메모리 주소 변경으로 인한 오버헤드가 크다. 불 연속적인 데이터에 접근할 때마다 DMA set과 busy check를 위해 10,272 사이클의 오버헤드가 발생한다.

AlexNet 컨벌루션 레이어 3에 대해, 아이디얼 레이아웃을 ordinary DMA 혹은 SG-DMA로 처리하는 것이 가장 적합하며 ordinary DMA, SG-DMA 모두 DMA set 및 busy check를 하는데 1,396 사이클의 오버헤드를 가진다.

가장 비효율적인 메모리 데이터 레이아웃과 DMA의 조합에 비해, 가장 효율적인 메모리 데이터 레이아웃과 DMA 조합의 오버헤드는 약 86 퍼센트까지 감소할 수 있음을 실험을 통해 확인한다.

이 연구를 통해, DMA를 사용할 때 데이터 재정렬을 위한 오버헤드와 DMA set을 위한 오버헤드 간의 트레이드 오프 관계를 따져볼 필요성이 있음을 확인했다.

References

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556v6, 2015.

[2] LeCun, Yann, Leon Bottou, Yoshua Bengio, and Patrick Haffner, “Gradient-based learning applied to document recognition,” *in IEEE,* 1998. DOI: 10.1109/5.726791

[3] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. Cnp: “An fpga-based processor for convolutional networks. In Field Programmable Logic and Applications,” *2009. FPL 2009. International Conference on* IEEE, pp.32-37, 2009.
DOI: 10.1109/FPL.2009.5272559

[4] Google. Improving photo search: A step across the semantic gap. http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html.

[5] S. Ji, W. Xu, M. Yang, and K. Yu. “3d convolutional neural networks for human action recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.,* Vol.35, No.1, pp.221-231, 2013.
DOI: 10.1109/TPAMI.2012.59

[6] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf. “A programmable parallel Accelerator for learning and classication,” *In Proceedings of the 19th international conference*

on Parallel architectures and compilation techniques, pp.273-284. ACM, 2010.

[7] R. Hadsell, A. Erkan, P. Sermanet, J. Ben, K. Kavukcuoglu, U. Muller, and Y. LeCun, "A multi-range vision strategy for autonomous offroad navigation," *in Proc. Robotics and Applications (RA'07),* 2007.

[8] Y. Ma, Y. Cao, S. Vrudhula and J. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," *in IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.26, no.7, pp.1354-1367, 2018. DOI: 10.1109/TVLSI.2018.2815603

[9] Lukas Cavigelli, Luca Benini "Origami: A 803 GOp/s/W Convolutional Network Accelerator in Origami: A 803 GOp/s/W Convolutional Network Accelerator," 2017.

[10] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," *in IEEE CVPRW,* 2014. DOI: 10.1109/CVPRW.2014.106

[11] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam, "Shidiannao: shifting vision processing closer to the sensor," *in Proceedings of the 42nd. Annual International Symposium on Computer Architecture,* pp.92-104, 2015.

[12] Dao-Fu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Temam, XiaobingFeng, Xuehai Zhou, and Yunji Chen "PuDianNao: A Polyvalent Machine Learning Accelerator," *in ASPLOS '15 Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015. DOI: 10.1145/2786763.2694358

[13] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable Accelerator for deep convolutional neural networks," *in IEEE Journal of Solid-State Circuits (JSSC),* Vol.52, No.1, pp.127-138, 2017.
DOI: 10.1109/JSSC.2016.2616357

[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *in 43rd Annual International Symposium on Computer Architecture (ISCA),* 2016. DOI: 10.1145/3007787.3001177

[15] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam, "DaDianNao: A Machine-Learning Supercomputer," *in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014. DOI: 10.1109/MICRO.2014.58

[16] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," *in ASPLOS '14 Proceedings of the 19th international conference on Architectural support for programming languages and operating systems,* 2014.
DOI: 10.1145/2644865.2541967

[17] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," *in FPGA,* 2015. DOI: 10.1145/2684746.2689060

[18] Youngjin Jo, Youngnam Kim, Sanghyuk Jung, Yong Ho Song "Implementation of Low Cost and High Performance DMA for PCI Express based SSD," *in Korea Institute Of Communication Sciences,* 2012.

[19] GUO, Kaiyuan, et al. "A survey of fpga-based neural network accelerator," arXiv preprint arXiv: 1712.08934, 2017.

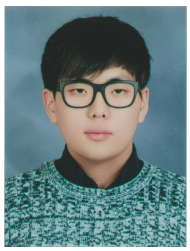
[20] Ma, Yufei, et al. "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017.
DOI: 10.1145/3020078.3021736

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks," In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, "Advances in Neural Information Processing Systems 25," *Curran Associates, Inc.*, pp.1097-1105, 2012.

[22] K. Simonyan and A. “Zisserman. Very deep convolutional networks for largescale image recognition,” CoRR, abs/1409.1556, 2014.
[23] Chen, Y. H., Emer, J., & Sze, V. (2018). “Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks,” arXiv preprint arXiv:1807.07928
[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Microsoft Research, “Deep Residual Learning for Image Recognition,” arXiv:1512.03385v1, 2015.
[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto. Hartwig Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” arXiv:1704.04861v1, 2017.
[26] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” arXiv:1707.01083, Dec 2017.
[27] Mingxing Tan, Quoc V. L, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” arXiv:1905.11946v3, 2019.

BIOGRAPHY

Seok-Jae Cho (Member)



2018 : BS degree in Electrical Engineering, dongseo University.
2021 : MS degree in Electrical Engineering, Pusan University.

SungkyungPark (Member)

2002 : Ph.D in Electrical Engineering, Seoul National University
2004 : Senior Engineer, Samsung Electronics
2006 : a Senior Member of Research Staff, Electronics and Telecommunications Research Institute
2009 : a Senior Staff Hardware Designe, Ericsson, Inc
2009~ faculty of the Department of Electronics Engineering, Pusan National University

Chester Sungchung Park (Member)

2006 : Ph.D. degree in electrical engineering, KAIST.
2007 : Samsung Electronics Inc
2013 : Senior Engineer, Ericsson Research, USA
2013~ Department of Electronics Engineering, Konkuk University, Korea, as an Associate Professor