디지털산업정보학회 논문지 제17권 제4호-2021년 12월
http://dx.doi.org/10.17662/ksdim.2021.17.4.053

경량 딥러닝 가속기를 위한 희소 행렬 압축 기법 및 하드웨어 설계

김 선 희* · 신 동 엽** · 임 용 석***

Sparse Matrix Compression Technique and Hardware Design for Lightweight Deep Learning Accelerators

Sunhee Kim · Dongyeob Shin · Yong-Seok Lim

〈Abstract〉

Deep learning models such as convolutional neural networks and recurrent neual networks process a huge amounts of data, so they require a lot of storage and consume a lot of time and power due to memory access. Recently, research is being conducted to reduce memory usage and access by compressing data using the feature that many of deep learning data are highly sparse and localized. In this paper, we propose a compression-decompression method of storing only the non-zero data and the location information of the non-zero data excluding zero data. In order to make the location information of non-zero data, the matrix data is divided into sections uniformly. And whether there is non-zero data in the corresponding section is indicated. In this case, section division is not executed only once, but repeatedly executed, and location information is stored in each step. Therefore, it can be properly compressed according to the ratio and distribution of zero data. In addition, we propose a hardware structure that enables compression and decompression without complex operations. It was designed and verified with Verilog, and it was confirmed that it can be used in hardware deep learning accelerators.

Key Words : Accelerator, Bitmap, Compression, Decompression, Sparse Matrix

Ⅰ. 서론

합성곱 신경망(Convolutional Neural Network, CNN)을 비롯한 다양한 딥러닝 기술이 이미지 분류 및 객체 인식, 음성 인식, 데이터 추론, 자연어 처리, 로봇 제어 등의 분야에서 활발하게 사용되고 있다[1-4]. 일반적으로 딥러닝 모델은 대용량 데이터를 기반으로 학습 과정을 거쳐 생성한다. 그리고, 모델의 깊이가 깊어질수록 모델의 정확도가 향상되는 경향이 있다[5]. 따라서 정확도 높은 모델을 구축하기 위해서는 학습을 위한 시간 및 연산량, 저장 용량 등이 급증하게 되며, 추론을 위해서도 고성능 컴퓨팅 및

* 상명대학교, 시스템반도체공학과, 조교수(제1저자)
** 한국전자기술연구원, 스마트네트워크연구센터, 선임연구원(참여저자)
*** 한국전자기술연구원, 스마트네트워크연구센터, 센터장(교신저자)

대용량 메모리를 요구한다. 그래서, GPU 및 딥러닝 전용 가속기를 사용하여 대규모 벡터/행렬 데이터를 고속 병렬 처리하여 성능을 향상시킨다[6, 7].

최근에는 IoT, 모바일 기기 등 소형 컴퓨팅 장치에서도 딥러닝 기술을 적용한 응용 프로그램 사용 요구가 증가하고 있다[8, 9]. 따라서, 저성능 컴퓨터에서 실행이 가능하도록, 학습된 고성능 모델에 대하여 정확도 손실을 최소화하며 모델의 파라미터 수와 복잡도를 낮추려는 연구가 진행되고 있다[10]. 대표적인 모델 경량화 기술로는 가지치기(pruning)와 양자화(quantization)가 있으며, 이 기술들을 적용하면 모델 파라미터의 희소성이 증가한다[11, 12].

희소성 높은 모델은 시스템 구조적 측면에서 경량화가 가능하다. 0의 값을 갖는 데이터는 저장하지 않음으로써 필요한 메모리 용량을 줄이고 동시에 메모리 접근에 따른 전력 소모를 낮출 수 있다[13, 14]. 이를 위해서는 0이 아닌 데이터만 저장하고, 0 데이터의 위치와 0이 아닌 데이터의 위치를 구분할 수 있도록 압축하는 방법이 필요하다.

본 논문에서는 딥러닝 가속기에 적합한 0 제거 압축 알고리즘 및 하드웨어 구조를 제안한다. 다음 장에서 기존 압축 알고리즘을 살펴본 뒤 하드웨어 구현에 적합한 압축 알고리즘 및 구조를 제안한다. 그리고 하드웨어 설계 결과를 보인 후 결론을 내린다.

II. 압축 알고리즘

2.1 관련 연구

딥러닝 가속기에서 희소 행렬 데이터에 대하여 일반적으로 사용하는 압축 방법으로 Run-length Compression(RLC), Compressed Sparse Row/Column (CSR, CSC), Zero Value Compression(ZVC) 등이 있다. RLC는 연속된 0의 개수와 그 뒤에 0이 아닌 값들을 연속으로 나열하는 방법이다[14]. 0이 연속으로 배치되어 있을수록 압축률이 크다. 비교적 하드웨어 구현에 적합하지만, 임의 위치의 데이터 접근에는 한계가 있다.

CSR/CSC[15]은 0이 아닌 데이터와 그 데이터의 위치(행, 열)를 저장한다. 필요한 값들만 저장하기 때문에 압축 효율이 높다는 장점이 있다. 하지만 행렬의 크기가 큰 경우에 위치 저장을 위하여 많은 비트가 할당되어야 한다는 단점이 있다.

ZVC은 위치 저장의 비효율을 개선하기 위하여 0이 아닌 데이터의 위치를 좌표 값 대신 비트 마스크로 저장한다. 즉, 행렬 크기의 비트 마스크가 있어서, 행렬에서 0 데이터의 위치에는 0으로, 0이 아닌 데이터의 위치에는 1로 설정하여 마스크를 저장하는 방법이다[16]. 행렬의 모든 위치에 대하여 1 bit를 저장해야 하므로, 0이 많은 행렬에 대해서는 효율이 상대적으로 낮다.

본 논문에서는 ZVC 방법을 개선하여 간략화된 비트 마스킹 방법을 제안하고, 이를 구현하기 위한 복잡도 낮은 하드웨어 구조를 제안한다.

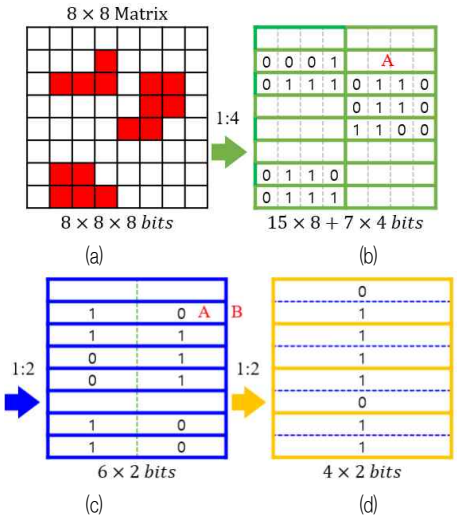
2.2 제안하는 압축 알고리즘

제안하는 0 제거 압축 알고리즘에서는 0이 아닌 데이터와 0이 아닌 데이터의 위치를 나타내는 비트 마스크 배열을 저장하여 압축 효율을 높이며 데이터의 메모리 엑세스를 줄인다. 우선, 위치 비트 마스크는 ZVC처럼 행렬의 각 데이터에 대하여 0이면 0으로 표시하고, 0이 아니면 1로 표시한다. 생성된 위치 비트 마스크에 대하여 일정 비트 수만큼씩 묶어서 새로운 행렬을 구성한다. 그리고, 새로운 행렬에 대하여 다시 위치 비트 마스크를 생성한다. 앞서와 마찬가지로, 하나로 묶인 비트 데이터가 모두 0일 때는 0으로 표시하고, 그렇지 않을 때는 1로 표시한다. 이때, 0으로 표시되었다면, 이전에 만들어진 위치 비트 마스크에서 해당 비트의

값들이 모두 0이었다고 해석할 수 있으므로, 이전 위치 비트 마스크의 해당 비트 묶음은 저장할 필요가 없다. 새로 생성된 위치 비트 마스크의 값들이 모두 1이 되면 더이상 압축을 진행할 필요가 없다.

비트 묶음 수는 행렬의 특성에 따라 선택할 수 있으며, 비트 묶음 수가 압축 비율이 된다. 또한 압축 비율이 단계적으로 선택되므로, 0이 적을 때는 최하위 단계의 묶음 수 만큼만 압축이 될 수도 있고, 0이 많을 때는 각 단계의 묶음 수를 곱한 값만큼 압축이 될 수도 있다.

<그림 1>은 제안하는 알고리즘에 대한 하나의 예이다. 그림 1(a)와 같이, 데이터가 8 bits인 8x8 행렬이 있다. 전체 데이터의 크기는 8x8x8=512 bits이다. 이 중 색이 칠해져 있는 15개의 데이터가 0이 아닌 값을 갖는다고 가정하자. 첫 번째로 주어진 행렬 데이터 중에서 0이 아닌 데이터만 저장하기 때문에, 15x8=120 bits의 저장 공간이 필요하다.



<그림 1> 0 제거 압축 알고리즘 적용 예. (a) 압축 전 행렬, (b) 압축 1단계, (c) 압축 2단계, (d) 압축 3단계

두 번째로 위치 비트 마스크를 생성한다. 총 64 bits 중 15 bits가 1의 값을 갖고 나머지 49 bits는 0의 값을 갖는다. 그리고 위치 비트 마스크를 그림 1(b)와 같이 4개씩 묶어서 표현한다. 기존 64개의 영역이 64/4=16개의 영역으로 구분된다. 이 중 4개의 비트가 모두 0인 묶음은 9개이고, 하나라도 1이 포함된 묶음은 7개이다. 저장되는 위치 비트 마스크는 <그림 1(b)>에 숫자 1이 포함되어 있는 총 7개의 묶음이다. 즉, 7x4=28 bits의 저장 공간이 필요하다.

모두 0인 묶음이 있으므로, 다시 영역을 나누어 위치 비트 마스크를 만든다. 이 예에서는 <그림 1(c)>와 같이 2개씩 묶어서 총 8개의 영역으로 구분하였다. <그림 1(c)>에서 위에서부터 차례대로 두 번째 영역 B를 살펴보면, 첫 번째 칸은 이 전 위치 비트 마스크 결과에 1이 포함되어 있으므로 1의 값을 갖는다. 두 번째 칸은 이전 위치 비트 마스크 값이 모두 0이었으므로 0의 값을 갖는다. <그림 1(b)>에서는 영역 A의 값이 모두 0이었기 때문에 저장하지 않았지만, <그림 1(c)>에서는 A 영역이 1을 포함하고 있는 B 영역의 일부이기 때문에 0을 기록하게 된다. 2번째 단계에서는 6x2=12 bits의 저장 공간이 필요하다.

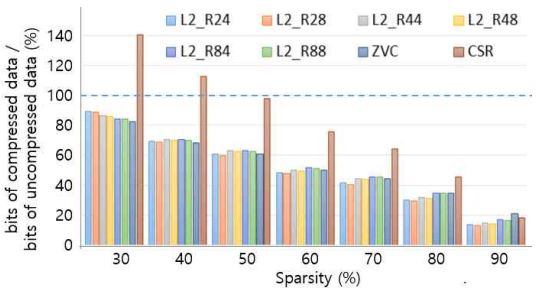
2번의 위치 비트 마스크 생성 후에도 모두 0인 곳이 있으므로, <그림 1(d)>와 같이 한 번 더 영역을 나누어 위치 비트 마스크를 생성한다. 이번에도 2개씩 묶어서 영역을 나누었다. <그림 1(d)>에서 보는 바와 같이 모두 0인 영역이 없으므로 여기에서 위치 비트 마스크 생성을 멈춘다. 이번 단계에서는 4x2=8 bits의 저장 공간이 필요하다.

결과적으로, 데이터를 압축하지 않았다면 총 512 bits의 저장 공간이 필요한데, 제안하는 방법으로 압축을 하면 총 168 bits의 저장 공간이 필요하다. 168 bits 중 120 bits가 순수한 데이터 저장 공간이고 48 bits가 위치 비트 마스크 배열을 위한 저장 공간이다. 64 bits의 위치 마스크 크기가 약 25% 감소되었다. 정리하면, 제안하는 방법은 0이 아닌 데이터의 위치를 표시하기 위하여 단계적으로 비트 마스크를 작성함으로써, 0이 연속으로 있을 때 마스크 비트 수를 감소

시킬 수 있다.

2.3 압축 성능

<그림 2>는 ZVC, CSR 그리고 제안하는 알고리즘에 대하여 압축 결과 비트 수를 비교하여 보여준다. 비교를 위하여 사용된 데이터는 8-bit 256x256 행렬로, 압축하지 않았을 때 필요한 저장 공간은 524,288 bits이다. 행렬 데이터는 random() 함수를 이용하여 0을 임의로 배치하였으며, 0의 확률을 30% - 90%로 설정하였다. 제안하는 알고리즘에 대해서는 2단계로 압축 과정을 진행하였다. <그림 2>에서 L2_Rmn은 첫 번째 단계의 묶음 수가 m이고 두 번째 단계의 묶음 수가 n임을 의미한다.

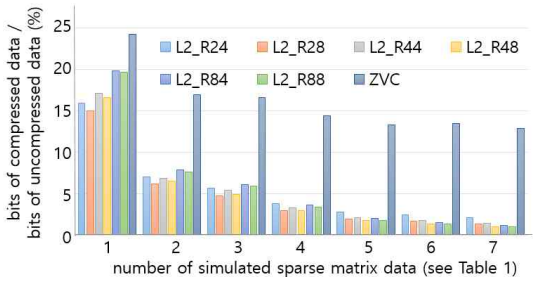


<그림 2> 0을 임의로 배치한 행렬에 대하여 ZVC, CSR, 그리고 제안하는 압축 알고리즘에 대한 압축률 비교

<그림 2>에서 보는 바와 같이 CSR은 0의 비율이 90%보다 작을 때 비트 마스크 방식에 비하여 압축률이 좋지 않다. 0이 아닌 데이터만 저장하는 것은 세 가지 방법 모두 동일하다. 하지만, CSR은 위치를 나타내기 위하여 0이 아닌 데이터의 열 번호와 행 누적 수를 저장하므로 0이 아닌 데이터의 수에 따라 변화량이 크기 때문이다.

비트 마스크를 기본으로 하는 ZVC 및 제안하는 알고리즘은 0의 비율이 60% 이상일 때 압축률이 약 50% 이상이다. 0의 비율이 50%보다 작을 때는 ZVC의 압축률이 가장 좋다. 반면에 0의 비율이 50% 이상이 되면 제안하는 알고리즘의 압축률이 좋다. 0이 많을수록 정해진 묶음 안의 데이터가 모두 0일 확률이 높아져서 저장하지 않는 위치 비트가 증가하기 때문이다.

<그림 3>은 Sparse Suite Collection[17]에서 제공하는 희소 행렬 데이터 7개를 사용하여 ZVC와 제안하는 압축 알고리즘에 대한 압축 정도를 비교한 결과이다. <표 1>은 7개의 데이터에 대한 정보로, 전체 행렬 데이터에서 0 데이터는 약 90% 이상이다. <그림 3>에서 보는 바와 같이, 0 데이터의 비율이 증가할수록 제안하는 알고리즘에서 요구하는 저장 공간이 ZVC에 비하여 크게 감소한다. 7번 데이터의 경우 L2_R44로 압축하였을 때, 압축 전 데이터에 비하여 약 1.4%, ZVC(12.87%)에 비하여 약 10%의 저장이 요구된다.



<그림 3> 테스트 희소 행렬에 대하여 ZVC, 그리고 제안하는 압축 알고리즘에 대한 압축률 비교 (데이터 번호는 표1에 표시된 번호임)

<표 1> 테스트 희소 행렬의 특성

number	name of sparse matrix data	row	column	non-zero	sparsity (%)
1	cis-n4c6-b1	210	21	420	90.47
2	GD06_theory	101	101	308	96.98
3	wheel_7_1	114	113	338	97.37
4	cat_ears_3_1	204	181	542	98.53
5	ch5-5-b4	120	600	600	99.16
6	08blocks	300	300	592	99.34

7	GD00_a	352	352	458	99.63

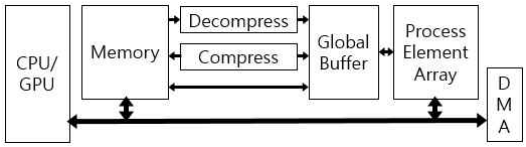
ZVC와 제안하는 알고리즘의 압축률 차이가 <그림 2>에서 0의 비율이 90%일 때에 비하여 <그림 3>에서 더 크게 나타나는 이유는 딥러닝 데이터의 근접성(locality) 때문이다. <그림 2>에서 사용된 데이터에서 0은 임의로 배치되어 있지만, <그림 3>에서 사용된 데이터에서 0은 근접성에 의하여 연속적으로 배치되어 있다. 데이터에서 0이 아닌 데이터들이 연속적으로 있으면, 불연속적으로 있을 때보다 같은 묶음으로 묶일 확률이 증가하면서 위치 정보 비트맵의 수가 감소하게 된다. 따라서 근접성을 갖는 딥러닝 데이터에 대하여 데이터의 희소성이 증가할수록 제안하는 알고리즘이 ZVC에 비하여 압축률이 좋아진다.

<그림 2>와 <그림 3>를 보면, 제안하는 알고리즘은 동일한 데이터에 대하여 묶음 수가 달라지면 압축률이 달라진다. 0의 비율뿐 아니라 0의 배치, 특히 연속성에 따라 묶음 안의 데이터가 모두 0일 확률이 달라지기 때문이다. 따라서 행렬 데이터를 압축하기 전에 시뮬레이션을 통하여 최적의 압축 단계 및 묶음 수를 결정한다.

III. 압축/해제 하드웨어 구조

3.1 시스템 구성

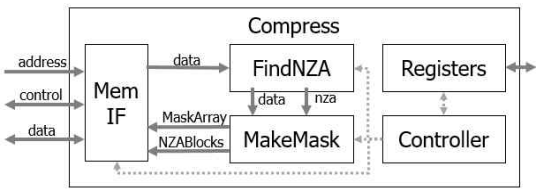
<그림 4>는 제안하는 압축 알고리즘을 적용한 딥러닝 시스템 구조의 예이다. 하드웨어 가속기를 포함한 인공지능 시스템에서는 CPU/GPU 수준에서 데이터를 하드웨어 가속기로 넘겨주고, 하드웨어 가속기에서 처리가 끝난 뒤 그 결과를 받게 된다. 이때 데이터는 메모리를 통하여 전달되기 때문에, 시간 지연 및 전력 소비가 증가한다. 따라서 CPU/GPU는 희소 행렬 데이터를 압축하여 메모리에 저장하고, 가속기에서 이를 읽어와 사용하기 전에 압축해제기를 거쳐 원래 데이터로 만들어준다. 반대로, 가속기의 결과 데이터는 압축기를 거쳐 압축시킨 후에 메모리에 저장한다. 따라서 <그림 4>와 같이 압축/해제기는 시스템 메모리와 하드웨어 가속기 전용 메모리 사이에 위치하게 된다.



<그림 4> 압축 알고리즘을 적용한 딥러닝 시스템의 예

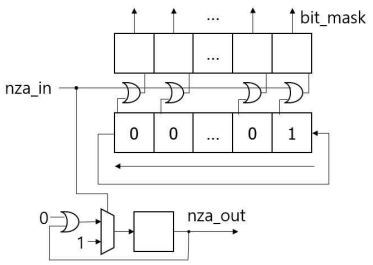
3.2 하드웨어 압축/해제기 구조

하드웨어 압축기는 가속기의 데이터를 시스템 메모리로 옮겨놓을 때 사용된다. 압축기는 <그림 5>와 같이 MemoryInterface, FindNZA, MakeMask, Registers, 그리고 Controller로 구성된다. 시스템에서 Registers에 압축 방법에 관한 정보를 설정하며, 압축기는 압축기의 상태 정보를 설정한다. Controller는 Registers의 정보를 바탕으로 MemoryInterface, FindNZA, MakeMask를 제어하여 압축한다. MemoryInterface는 가속기 버퍼의 데이터를 읽고, 시스템 메모리에 데이터를 저장하는 역할을 한다. FindNZA는 입력 데이터에 대하여 0과 0이 아닌 데이터를 구분한다. MakeMask는 시스템에서 정해준 묶음 비율에 따라 단계별로 위치 비트 마스크를 구성한다.



<그림 5> 압축기 블록도

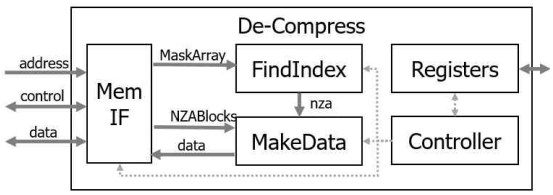
MakeMask는 시스템에서 지원하는 단계 수만큼 비트 마스크 생성기를 포함하고 있다. <그림 6>에서 보는 바와 같이 비트 마스크 생성기에는 최대 데이터/비트 묶음 수만큼의 시프트 레지스터와 비트 마스크를 저장하는 레지스터가 있다. 시프트 레지스터는 LSB를 1, 나머지 상위 비트들은 모두 0으로 초기화 한 뒤 데이터가 입력될 때마다 상위 비트로 시프트한다. 비트 마스크 레지스터는 0으로 초기화 된다. 하위 단계 비트 마스크 생성기에서 보낸 nza_in (0 혹은 1)과 시프트 레지스터의 값을 비트 별로 OR 연산하여 비트 마스크 레지스터의 값을 설정한다. 시프트 레지스터에서 1의 위치가 묶음 수의 위치가 되면 비트 마스크 레지스터의 비트 마스크 레지스터의 값을 출력한 뒤 시프트 레지스터와 비트 마스크 레지스터를 초기화 한다. 이 때 묶음 수 동안에 모든 입력 값이 0이었다면 nza_out을 0으로 출력하고 그렇지 않으면 1로 출력한다. nza_out은 상위 단계 비트 마스크 생성기의 nza_in으로 입력된다. 그리고 nza_in 값이 0이면 출력된 비트 마스크 값도 모두 0이므로 MaskArray에는 포함되지 않는다.



<그림 6> 비트 마스크 생성기 블록도

제안하는 압축 하드웨어는 곱셈/나눗셈 등 복잡한 연산기 없이 비교기와 시프트 레지스터 등 간단한 로직만으로 구성이 가능하다. 그리고 각 단계별 비트 마스크는 병렬적으로 생성되므로, 단계 수 만큼의 지연을 갖는다.

하드웨어 압축해제기는 시스템 메모리의 데이터를 가속기로 옮길 때 사용된다. 압축해제기는 <그림 7>과 같이 MemoryInterface, FindIndex, MakeData, Registers, 그리고 Controller로 구성된다. MemoryInterface는 압축기와 반대로 시스템 메모리의 압축 데이터를 읽고, 가속기의 버퍼에 압축 해제된 데이터를 저장하는 역할을 한다. FindIndex는 시스템에서 정해준 단계별 영역 묶음 비율에 맞추어 메모리에서 읽어온 MaskArray 정보를 해석하여 0의 위치와 0이 아닌 데이터의 위치를 찾아낸다. 그리고 MakeData는 NZABlocks의 데이터를 차례대로 출력하는데, FindIndex의 해석에 따라 0의 위치에서 0 데이터를 삽입하여 출력한다.



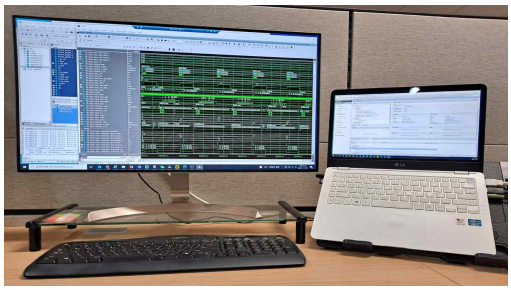
<그림 7> 압축해제기 블록도

FindIndex에는 각 단계별로 묶음 수만큼 수를 세는 카운터가 있으며, 아랫 단계의 카운터가 묶음 수만큼 증가하면 윗 단계의 카운터가 1씩 증가한다. 동시에 각 단계의 위치 비트 마스크를 병렬로 해석한다. 우선 순위에 따라 최상위 위치 비트 마스크의 LSB부터 차례대로 해석한다. 위치 비트 마스크의 LSB가 1이면 아래 단계의 위치 비트 마스크의 LSB를 확인하고, 0이면 0을 출력한다. 단, 최하위 위치 비트 마스크의 LSB가 1이면 1을 출력한다. 위치 비트 마스크는 현 단계의 카운터가 묶음 수 만큼 증가하고, 상위 위치 비트 마스크의 LSB가 1이면 LSB쪽으로 시프트하여, 항상 LSB가 현재 비트 위치에 대한 정보를 나타내도록 한다. 단, 최상위 위치 비트 마스크는 현 단계의 카운터가 묶음 수 만큼 증가하면 LSB쪽으로

시프트한다. 따라서 압축 해제기도 곱셈/나눗셈 등의 복잡한 연산 없이 비교기와 시프트 레지스터 등 간단한 로직만으로 구성이 가능하다.

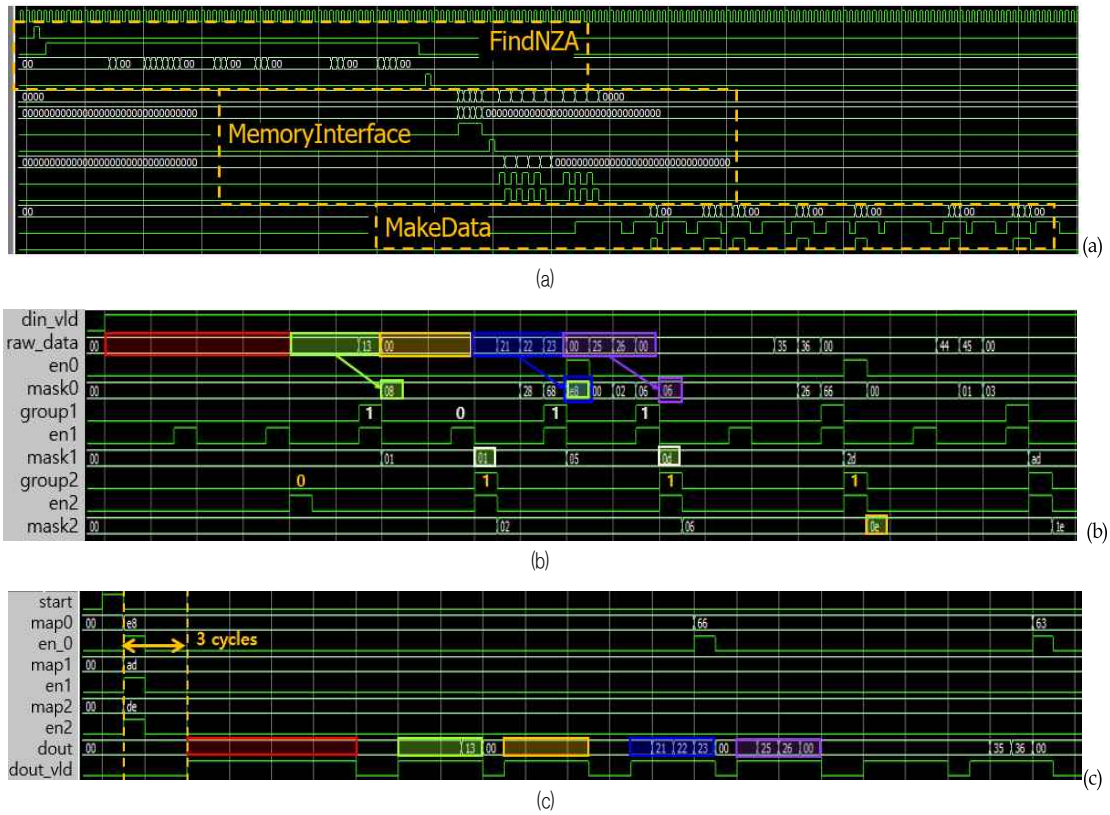
IV. 설계 및 결과

제안하는 압축 알고리즘에 따라 VerilogHDL을 이용하여 하드웨어 압축/해제기를 설계하고 동작을 검증하였다. <그림 8>은 설계 및 검증 환경을 보여준다. <그림 8>의 왼쪽은 ModelSim 설계툴로써, 하드웨어의 기능을 검증하기 위하여 사용되었다. 오른쪽은 Vivado 설계툴로써, 합성(synthesis) 가능한 설계임을 확인하기 위하여 사용되었다.



<그림 8> 압축/해제 하드웨어 블록의 시뮬레이션 개발 환경. (좌) Modelsim을 이용한 기능 검증, (우) Vivado를 이용한 합성 검증

압축 단계는 최대 4이며, 데이터/비트 묶음은 2, 4,



<그림 9> 압축/해제 하드웨어 블록의 시뮬레이션 결과 파형. (a) FindNZA 블록의 입력 신호, 압축/해제 블록의 MemoryInterface 입출력 신호, 그리고 MakeData의 출력신호, (b) 압축 블록 신호, (c) 압축 해제 블록 신호

8, 세 가지로 설정하였다. 압축 안 된 데이터의 최대 사이즈는 256 bytes로 설계하였다. 압축/해제기의 Registers 설정을 위해서는 APB 인터페이스를 가정하였다.

<그림 9>은 압축/해제 하드웨어 블록의 시뮬레이션 결과 파형이다. <그림 1>의 예제와 같이 8비트 데이터 8x8 matrix를 원형 데이터로 사용하였으며, 첫 번째는 4개씩, 그리고 두 번째, 세 번째는 각각 2개씩 데이터를 묶으며 위치 비트 마스크를 작성하였다. 이 때 0이 아닌 데이터의 값은 그 위치에 따라 상위 4 bits으로 행 번호를, 하위 4 bits로 열 번호를 나타내도록 설정하였다. 원형 데이터를 압축 블록에 입력으로 제공한 뒤 출력되는 압축 데이터를 다시 압축해제 블록으로 입력하여 해제된 데이터 값과 처음 입력한 값을 비교하였다.

<그림 9(a)>는 위에서부터 차례대로 FindNZA 블록의 입력 신호, 압축/해제 블록의 MemoryInterface 입출력 신호, 그리고 MakeData의 출력신호를 보여준다. <그림 9(b)>와 <그림 9(c)>는 각각 압축 블록 신호와 압축해제 블록 신호를 확대하여 보여준다. <그림 9(b)>에서 보는 바와 같이 FindNZA 블록에 din_vld 신호가 1일 때 8 bits 원형 데이터(raw_data)가 차례대로 입력된다. 첫 번째 위치 비트 마스크(mask0)는 연속 4개의 raw_data 중 0이 아닌 값이 있을 때 0이 아닌 값의 위치를 4 bits으로 표시한다. 두 번째 위치 비트 마스크(mask1)은 연속 2개의 mask0의 값 중 0이 아닌 값이 있을 때 그 값의 위치를 2 bits으로 표시한다. 마찬가지로 세 번째 위치 비트 마스크(mask2)는 연속 2개의 mask1의 값 중 0이 아닌 값이 있을 때 그 값의 위치를 2 bits으로 표시한다.

압축 해제 블록은 <그림 9(c)>에서 보는 바와 같이 mask0, mask1, mask2 정보를 입력으로 받아서 세 개의 mask 정보를 계층적으로 해석하여 데이터를 출력한다. 유효 데이터를 출력할 때마다 dout_vld 신호를 1로 설정하였으며, FindNZA의 입력과 동일함을 확인하였다.

Ⅴ. 결론

본 논문에서는 경량 인공지능 가속기를 위한 희소 행렬 압축 기법과 이를 적용한 압축/해제 하드웨어 구조를 제안하고 설계하였다. 단계별로 압축률을 선택할 수 있어서 데이터의 특성에 맞게 압축 정도를 결정할 수 있다. 시뮬레이션을 통하여 0이 50% 이상일 때 제안하는 방법이 ZVC와 CSR에 비하여 압축률이 좋음을 보였다. 압축/해제 하드웨어는 가속기와 시스템 메모리 사이에 위치하여 희소 행렬의 데이터 저장 용량을 낮추고 메모리 접근에 따른 시간 및 전력 소비를 감소시킨다. 또한 제안하는 0 제거 압축 알고리즘에 대하여 하드웨어 구현에 적합한 구조를 제시하여 복잡도 및 지연을 낮춘다. Verilog HDL로 설계하고 Modelsim으로 동작을 검증하여 딥러닝 가속기 하드웨어에 적용 가능함을 확인하였다. 본 0 제거 압축/해제 하드웨어 모듈은 딥러닝 가속기뿐 아니라 대규모 희소 데이터 처리기에 적용되어 저장 용량 및 전력 소비를 낮추며 처리 속도를 향상시킬 것으로 기대된다.

Acknowledgment

본 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (IITP- 2021-0-00875, 데이터센터-엣지 NPU 간 연합 추론/학습 기반 대규모 인공지능 응용/개발을 용이하게 하는 SW 프레임워크 개발, 50%) (IITP-2020-0-01077, IoT 다중 인터페이스 기반의 데이터센싱, 엣지컴퓨팅 분석 및 데이터공유 지능형 반도체 기술 개발, 50%)

참고문헌

[1] Lin, C. H. , Cheng, C. C., Tsai, Y. M., Hung, S. J., Kuo, Y. T., Wang, P. H., Tsung P. K., Hsu, J. Y., Lai, W. C., Liu, C. H., Wang, S. Y., Kuo, C. H., Chang, C. Y., Lee, M. H., Lin, T. Y. and Chen, C. C.,, "A 3.4-to-13.3TOPS/W 3.6TOPS Dual-Core Deep-Learning Accelerator for Versatile AI Applications in 7nm 5G Smartphone SoC," 2020 IEEE International Solid- State Circuits Conference(ISSCC), San Francisco, CA, USA, 2020, pp.134-136.

[2] 안세현, 강석주, "경량화된 딥러닝 구조를 이용한 실시간 초고해상도 영상 생성 기술," 방송공학회 논문지, 제26권 제2호, 2021, pp.167-174.

[3] 김진호, "딥러닝 신경망을 이용한 문자 및 단어 단위의 영문 차량 번호판 인식", 디지털산업정보학회 논문지, 제16권, 제4호, 2020, pp.19-28.

[4] 장중봉, 최승원, "채널 상태 정보를 이용한 딥 러닝 기반 실내 위치 확인 시스템", 디지털산업정보학회 논문지, 제16권, 제4호, 2020, pp.1-7.

[5] 이경하, 김은희, 딥러닝 모델 경량화 기술 분석, 발한국과학기술정보연구원, 대전, 2020. p.89.

[6] 오광일, 김성은, 배영환, 박경환, 권영수, "인공지능 뉴로모픽 반도체 기술 동향," 전자통신동향분석, 제35권, 제3호, 2020, pp.76-84.

[7] Chen, Y., Xie, Y., Song, L., Chen, F. and Tang, T., "A Survey of Accelerator Architectures for Deep Neural Networks," Engineering, Vol.6, 2020, pp.264–274.

[8] 추연호, 최영규, "딥러닝을 이용한 IOT 기기 인식 시스템", 반도체디스플레이기술학회지, 제18권, 제2호, 2019, pp.1-5.

[9] 김윤빈, "딥러닝을 활용한 IoT 센서 데이터 처리에 관한 연구", 고려대학교 대학원 컴퓨터정보학과 국내석사 학위 논문, 2019.

[10] 이용주, 문용혁, 박준용, 민옥기, "경량 딥러닝 기술 동향," 전자통신동향분석, 제34권, 제2호, 2019, pp.40-50.

[11] Han, S., Mao, H. and Dally, W. J., "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," International Conference on Learning Representations(ICLR), San Juan, Puerto Rico, 2016,

[12] Zhu, M. H. and Gupta, S., "To prune, or not to prune: exploring the efficacy of pruning for model compression," International Conference on Learning Representations(ICLR), Vancouver Canada, 2018.

[13] Kanellopoulos, K., Vijaykumar, N., Giannoula, C., Azizi, R., Koppula, S., Ghiasi, N. M., Shahroodi, T., Luna, J. G. and Mutlu, O., "SMASH: Co-designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations," Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), Columbus, OH, USA, 2019, pp.600-614.

[14] Park, Y., Kang, Y., Kim, S., Kwon, E. and Kang, S., "GRLC: Grid-based Run-length Compression for Energy-efficient CNN Accelerator," Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, Boston, MA, USA, 2020, pp.91–96.

[15] Liu, W. and Vinter, B., "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication," Proceedings of the 29th ACM International Conference on Supercomputing (ICS '15), New York, USA,

2015, pp.339-350.

[16] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," IEEE International Symposium on High Performance Computer Architecture(HPCA), Vienna, Austria, 2018, pp. 78-91.

[17] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," https://sparse.tamu. edu/

■ 저자소개 ■

김 선 희
Kim, Sun Hee

2016년 9월~현재 상명대학교 시스템반도체공학과 조교수
2005년 3월~2012년 5월 KETI 선임연구원
2002년 2월~2005년 2월 ETRI 연구원
2016년 2월 이화여자대학교 전자공학과 (공학박사)
2002년 2월 이화여자대학교 정보통신학과 (공학석사)
2000년 2월 이화여자대학교 전자공학과 (공학사)

관심분야 : WPAN, Neuromorphic system
E-mail : happyshkim@smu.ac.kr

신 동 엽
Shin, Dongyeob

2020년 9월~현재 KETI 선임연구원
2020년 8월 고려대학교 전기전자공학과 (공학박사)
2013년 8월 고려대학교 전기전자전파공학부 (공학사)

관심분야 : Neural Network Accelerator
E-mail : dongyeob.shin@keti.re.kr

임 용 석
Lim, Yong-Seok

2021년 1월 ~ 현재 한국전자기술연구원 스마트네트워크연구센터 센터장
2007년 12월 ~ 2020년 12월 한국전자기술연구원 스마트네트워크연구센터 책임연구원
2005년 7월 ~ 2007년 11월 ㈜엠진 책임연구원
2003년 3월 ~ 2005년 6월 삼성전기 선임연구원
2017년 12월 고려대학교 공학박사
2003년 2월 고려대학교 공학석사
2001년 2월 고려대학교 공학사

관심분야 : 무선전력전송, 인공지능/통신 SoC 설계, 임베디드 시스템
E-mail : busytom@keti.re.kr

논문접수일 : 2021년 10월 12일
수 정 일 : 2021년 11월 3일(1차)
2021년 11월 15일(2차)
게재확정일 : 2021년 11월 22일