HLS 기반 딥러닝 가속 하드웨어의 ISA 확장을 통한 성능 향상

(Performance Improvement by Extending ISA of a HLS Based Deep Learning Accelerator)

권 용 인*, 김 영 주, 유 미 선, 이 제 민, 김 태 호
한국전자통신연구원
(Yongin Kwon, Young-Joo Kim, Misun Yu, Jemin Lee, Taeho Kim)
(Electronics and Telecommunications Research Institute)

Abstract : As the emerging field of deep learning, implementing deep learning accelerators using High-level Synthesis (HLS) makes it easier to run low-power and high-performance deep learning operations on FPGA. An HLS based deep learning accelerator, VTA, is the three-stage-architecture which is designed to run 'Load', 'Compute' and 'Store' modules sequentially and not allowed to transfer output data to input buffer without DRAM operations. This results in low utilization of processing elements and high power consumption. In this paper, we extend ISA (Instruction Set Architecture) of VTA to define 'MOV' instruction which transfers the data of output buffer to input buffer directly and implement it using pure HLS code. As a result, the proposed ISA and implementation improve the data transferring speed by 184% with the same FPGA resource usage.

Keywords : FPGA, VTA, HLS, ISA, Deep Learning

Ⅰ. 서 론

최근 딥러닝 기술을 통해 높은 정확도의 이미지 인식과 분류 등이 개발됨에 따라, 임베디드 시스템 상에서도 이를 수행하기 위해 저전력 고성능의 실행이 요구되고 있다. 이에 따라 CPU와 GPU에서의 딥러닝 연산 수행의 한계점을 극복하고자, 다양한 형태의 딥러닝 가속 하드웨어가 개발되고 있고, FPGA 상에서의 하드웨어 구현은 다향한 딥러닝 모델에 맞춰 빠른 하드웨어 설계와 테스트를 가능케 한다.

특히 High-level Synthesis (HLS)를 사용한 FPGA 하드웨어 구현은 C 언어와 같은 상위레벨 언어를 사용하여 Register Transfer Language (RTL)로의 자동 변환을 지원하기 때문에 개발 시간이 단축되고, Verilog나 VHDL에 대한 지식과 숙련이 없이도 개발 및 최적화가 가능하도록 해준다.

대부분의 딥러닝 가속 하드웨어는 딥러닝 연산 중 가장 큰 비중을 차지하는 행렬곱 연산을 병렬화 및 가속화 함으로써 딥러닝 연산을 가속화한다. 딥러닝 가속 하드웨어의 연산에 필요한 입력 데이터는 DRAM에 적재되어있기 때문에 상대적으로 느린 DRAM 접근 속도를 해결하기 위해 하드웨어 내부에 속도가 빠른 메모리 버퍼를 두어 연산기에 빠른 속도로 데이터를 공급해준다. 이 내부 버퍼는 크기가 비교적 작기 때문에 재사용을 하거나, 연산기가 동작하는 동안 다음 사용할 데이터를 미리 읽어 오는 방법으로 DRAM 병목현상을 줄인다. 따라서 DRAM 연산을 최소화 하는 것이 성능 향상에 큰 영향을 주며, 또한 DRAM 연산은 내부 버퍼 연산보다 전력 소모가 크기 때문에 전력 소모량에 민감한 임베디드 시스템 상에서는 가능한 줄이는 것이 유리하다. [1]

딥러닝 가속 하드웨어의 설계 방법 중, 다양한 딥러닝 모델을 하드웨어 상에서 유연하기 동작시키고 컴파일러 개발을 용이하게 하기 위해

*Corresponding Author (yongin.kwon@etri.re.kr)
권용인, 김영주, 유미선, 이제민, 김태호: 한국전자통신연구원(ETRI)
※이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-00769,인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발).

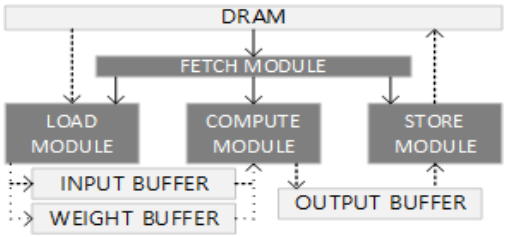

그림 1. VTA 하드웨어 아키텍처
Fig. 1. The hardware architecture of VTA

ISA(Instruction Set Architecture)기반 설계 방법이 있다. 본 논문에서는 ISA 기반 딥러닝 가속 하드웨어인 VTA(Versatile Tensor Accelerator)를 대상으로 ISA를 분석하고 인스트럭션 추가를 통해 DRAM 연산을 최적화하고 성능을 비교한다.

Ⅱ. VTA 하드웨어

VTA[2]는 FPGA 보드 상에서 구현이 가능한 딥러닝 가속 하드웨어로 오픈소스 딥러닝 컴파일러인 TVM[3]의 한 컴포넌트로 개발되었다. VTA는 HLS 언어로 작성된 인스트럭션 세트 기반의 아키텍처로, 전체 구조는 그림 1과 같다.

VTA는 네 개의 모듈(Fetch, Load, Compute, Store)로 구성되어 있다. fetch 모듈은 DRAM으로부터 인스트럭션을 읽어 이를 수행할 다른 모듈(Load, Compute, Store)로 전달하는 역할을 한다. Load 모듈에서는 DRAM으로부터 데이터를 읽어 Input 버퍼와 Weight버퍼에 저장하고, Compute 모듈은 이를 이용하여 GEMM이나 ALU연산을 수행한 뒤 Output 버퍼에 그 결과를 저장한다. Store 모듈은 Output 버퍼의 데이터를 DRAM에 저장하는 역할을 한다. 따라서 VTA에서는 데이터가 Load, Compute, Store 순서로 이동 및 연산되고, Output 버퍼에 있는 데이터를 다시 Input 버퍼로 읽어들이기 위해서는 Store 모듈을 통해 DRAM으로 저장을 한 뒤 Load 모듈에서 이를 다시 읽어들여야 한다. DRAM연산 시간을 숨기고(Hiding) Compute 모듈의 활용률을 높이기 위해 입력데이터를 분할하여 로드하고 재사용을 늘리며 3개의 모듈을 파이프라이닝(Pipeling) 등을 함으로써 수행시간 성능 하락을 최소화 한다.

VTA의 인스트럭션은 총 128 bitwidth를 사용하여 구성된다. 이는 fetch 모듈이 사용하는 AXI 인터페이스의 bandwidth와 일치하기 때문에 하드웨어

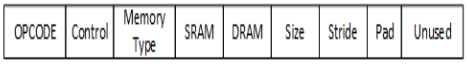

그림 2. VTA 메모리 인스트럭션 세트
Fig. 2. VTA's memory instruction set

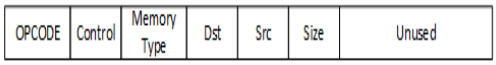

그림 3. VTA MOV 인스트럭션 세트
Fig. 3. VTA's MOV instruction set

가 효율적으로 인스트럭션을 읽어들일 수 있게 한다. 인스트럭션 세트의 Opcode는 총 세가지로, 메모리 인스트럭션, GEMM 연산 인스트럭션, ALU 연산 인스트럭션으로 나뉘고, 인스트럭션 세트은 Opcode와 Control 표현을 위해 7 bit가 할당되어 있다. 나머지 121 bit의 영역에는 DRAM 주소나 SRAM주소, 연산의 반복 수 등을 Opcode에 따라 다르게 표현한다. 그림 2는 메모리 인스트럭션의 구조를 나타낸다. HLS코드 상에 정의된 Memory Type은 총 네가지로 각기 다른 내부 버퍼를 가리키며 이를 위해 2 bit가 할당되어있다.

Ⅲ. MOV 인스트럭션 구현

Output 버퍼의 데이터를 DRAM을 통하지 않고 바로 Input 버퍼로 전달하여 수행 성능과 전력 소모량을 최적화 하기 위해 MOV 인스트럭션을 HLS 코드상에 구현한다.

1. 인스트럭션 세트 정의

본 논문에서는 MOV 인스트럭션을 ISA에 포함하기 위해 Opcode는 기존의 Load 인스트럭션을 그대로 사용한다. 대신, Memory Type 영역을 사용하여 MOV를 나타내는데, 이를 위해 bitwidth를 3으로 늘릴 필요가 있다. SRAM와 DRAM 영역을 활용하여 각각 Src 및 Dst 주소를 표현하고, Size 영역은 그대로 사용하여 전송 할 사이즈를 표현한다.

2. 인스트럭션 구현

그림 4는 MOV 인스트럭션을 구현한 HLS코드를 나타낸다. HLS 코드 상에서는 MOV 인스트럭션의 성능을 예측하기 힘들다. 이는 루프 내부의 언롤

```
if (insn.generic.opcode == VTA_OPCODE_LOAD) {
  if (insn.mem.memory_type == VTA_MEM_ID_MOVE) {
    for (int i = 0; i < insn.mem.x_size; i++) {
      inp_mem[i][0] = out_mem[i][0];
}}}
```

그림 4. MOV 인스트럭션 HLS 구현

Fig. 4. HLS implementation of MOV instruction

```
if (insn.generic.opcode == VTA_OPCODE_LOAD) {
  if (insn.mem.memory_type == VTA_MEM_ID_MOVE) {
    #pragma HLS UNROLL factor=2
    for (int i = 0; i < insn.mem.x_size; i++) {
      inp_mem[i][0] = out_mem[i][0];
    }}}
```

그림 5. '#pragma'를 용한 언롤 팩터 표현

Fig. 5. The expression of unroll factor using '#pragma'

```
if (insn.generic.opcode == VTA_OPCODE_LOAD) {
  if (insn.mem.memory_type == VTA_MEM_ID_MOVE) {
    for (int i = 0; i < insn.mem.x_size/2; i++) {
      inp_mem[i*2][0] = out_mem[i*2][0];
      inp_mem[i*2+1][0] = out_mem[i*2+1][0];
    }
    if (insn.mem.x_size%2==1) {
      inp_mem[insn.mem.x_size-1][0]
        = out_mem[insn.mem.x_size-1][0];
    }}}
```

그림 6. 명시적인 언롤 팩터 표현

Fig. 6. The explicit expression of unroll factor

(Unroll) 정도에 따라 다른데, 그림 5와 같이 '#pragma'를 이용하여 언롤 팩터(Unroll Factor)를 지정하거나, 그림 6와 같이 언롤된 코드 형태로 명시할 수 있다. 본 논문에서는 팩터 값에 따른 합성 결과를 분석하고 실험을 통해 성능을 검증한다.

Ⅳ. 실험 및 분석

1. 실험 환경

본 논문의 실험에 사용한 FPGA 보드는 Ultra96-V2-G로, Xilinx사의 Zynq UltraScale+ MPSoC EG 디바이스가 탑재되어있다. 해당 디바이스에는 Quad-core ARM Cortex-A53 CPU, LPDDR4 2GB를 구성하고 있으며, 가용한 FPGA 자원으로는 360개의 DSP Slice, 7.6Mb의 Block RAM, 71K의 LUT 등이 있다.

Xilinx사의 Vivado HLS 도구를 사용하여 수정한 HLS 코드를 비트스트림 합성을 하였으며 Vivado에서 제공해주는 합성결과 분석을 사용하여 리소스 사용량과 각 오퍼레이션의 지연시간, 전력소모 등을 비교하였다.

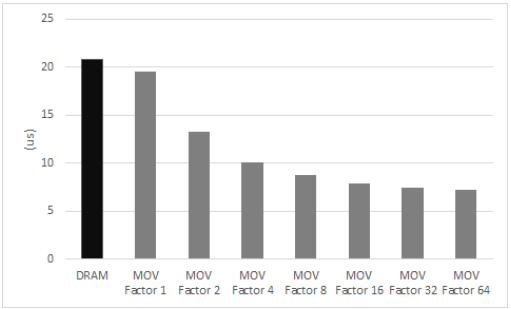

그림 7. 언롤 팩터에 따른 MOV 인스트럭션 수행 시간

Fig. 7. The execution time of MOV instruction with different unroll factors

추가된 MOV 인스트럭션의 성능을 비교하기 위해 32kB 크기의 Output 버퍼 데이터를 Input 버퍼로 전송하는 실험을 진행하여, DRAM을 경유할 경우와 각 언롤 팩터 값에 따른 변화를 비교하였다.

2. 실험 결과

그림 7은 언롤 팩터에 따른 MOV 인스트럭션의 수행시간을 나타낸다. 언롤 팩터가 1인 경우, DRAM을 통하여 데이터를 전송하는 것과 수행시간이 크게 차이나지 않았다. 그럼에도 불구하고 전력소모량은 크게 감소했을 것으로 예측된다. 언롤 팩터를 늘려감에 따라 수행시간이 더 빨라지는 것을 알 수 있고, 16 이후로는 포화하는 것을 알 수 있다.

비트스트림 합성 결과 FPGA 자원 활용률 또한 MOV의 유무와 언롤 팩터 값에 따라 큰 차이가 없었다. 특히 DSP Slice와 BRAM의 사용량은 항상 같은 값을 유지하였고 GEMM 연산에 따른 전력 소모량과 수행시간에 영향을 주지 않았다.

Ⅴ. 결 론

본 논문에서는 HLS로 구현된 인스트럭션 세트 기반 딥러닝 가속 하드웨어의 ISA를 확장함으로써 메모리 연산을 가속화 하고 전력소모량을 줄이는 것을 보였다. 이와 같이 FPGA의 가용 자원으로 기존 연산의 성능을 떨어뜨리지 않는 범위 내에서 ISA를 추가 확장하여 딥러닝 가속 하드웨어 상에서의 딥러닝 모델 수행을 최적화할 수 있을 것이다.

참 고 문 헌

[1] Yu-Hsin Cehn, et al. “Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks”, Proc. Int’l Symp. on Computer Architecture.

[2] VTA, https://tvm.apach.org/vta

[3] Chen, Tianqi, et al. "TVM: An automated end-toend optimizing compiler for deep learning.", OSDI. 2018.

[4] Mingzhen Li, et al. “The Deep Learning Compiler: A Comprehensive Survey”, arXiv:2002.03794, 2020