Hardware System Design

Term Project V2

이다운 2016-13919

1. Overview

Term project V0에서는 행렬 곱연산을 위한 floating point mac array를 보드의 하드웨어 가속기에 올려 구현하는 것을 하였고 V1에서는 구현한 array를 성능향상을 위해 transfer data를 quantization을 하여 구현하였다. 이번 V2에서도 마찬가지로 V1에서 성능 향상을 위한 구현으로 cpu 상에서 연산을 줄이는 것이다. 한번에 거대한 행렬 곱을 실시할 수 없으므로 행렬을 타일링 하여 연산을 쪼개서 하는 방법을 사용해왔다. 기존에 타일링은 보드의 cpu에서 했는데 이 타일링 과정을 cpu에서 하지 않고 fpga 하드웨어에서 하도록 구현하는 것이 이번 프로젝트의 목표다.

2. 파일 및 실행방법

ETL에 업로드한 zip파일속 zynq.bit 파일이나 제출한 SD카드에 있는 zynq.bit 파일을 sd카드에 넣고 보드와 pc를 연결 후 보드를 킨다. 그리고 /sdcard/hsd_v2 디렉토리로 이동한 뒤에 sudo date로 현재 날짜를 설정한 뒤 sudo make로 컴파일 해준다. 그리고 sudo bash benchmark.sh커맨드로 벤치마크를 실행시키면 된다.

hsd_v2 - sw코드 / v_v – Verilog 프로젝트 / result 폴더 – 결과 사진

3. 구현

```
float sz = (comp->act_min)*(-1.0);
float act_scale = (comp->act_max + sz)/act_bits_max; // TODO calculate the scale factor
int act_offset = (sz/act_scale); // TODO calculate the zero-offset
quantize(input, qinput, num_input, act_bits_min, act_bits_max, act_offset, act_scale);

int weight_bits_min = 0;
int weight_bits_max = (1<<(comp->weight_bits-1))-1;

sz = (comp->weight_min)*(-1.0);
float weight_scale = (comp->weight_max + sz)/weight_bits_max; // TODO calculate the scale factor
int weight_offset = (sz/weight_scale); // TODO calculate the zero-offset
quantize(large_mat, qlarge_mat, num_input*num_output, weight_bits_min, weight_bits_max, weight_offset, weight_scale);
```

먼저 fpga cpp 코드부터 살펴보겠다. 먼저 data를 보내기전 quantizie 과정부터 거쳐준다. V1과 마찬가지로 먼저 scale과 offset부터 계산한 뒤 scale을 각 배열의 원소들을 나눠 주는 방식으로 quantize를 한다.

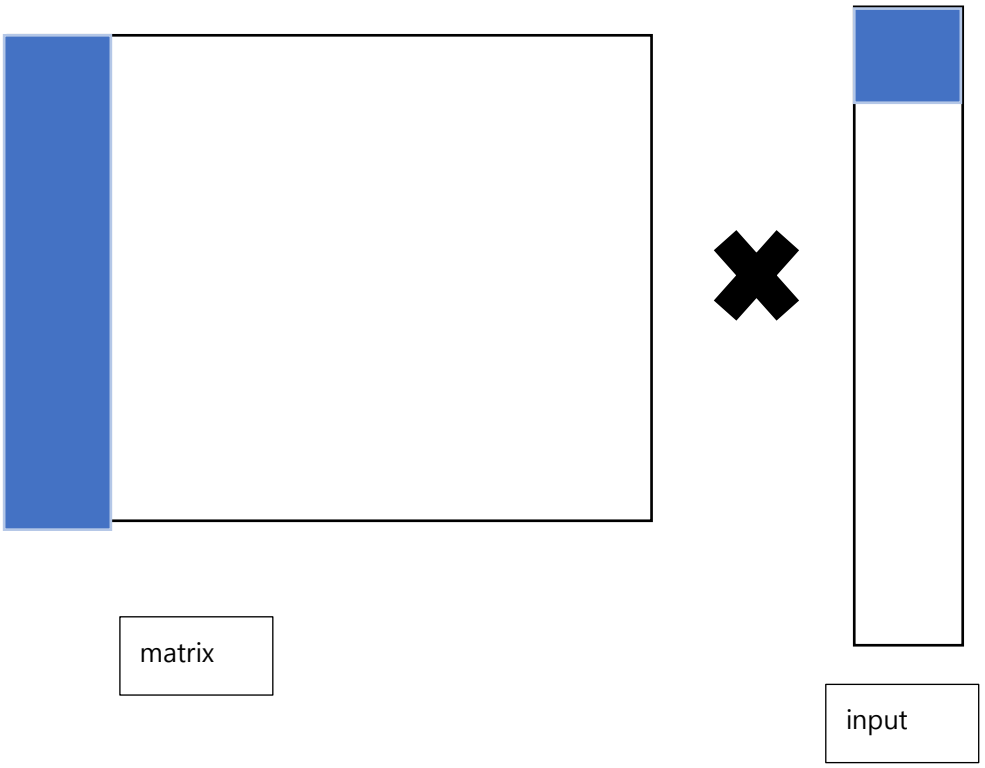
```
*(output_+2) = num_input;
*(output_+3) = num_output;

memset(vec, 0, (num_output + 1) * num_input * sizeof(int) );


memcpy(vec, qinput, sizeof(int)*num_input );
memcpy(vec+num_input, qlarge_mat, sizeof(int)*num_input*num_output );

const int* ret = this->qblockMV(comp);

for(int i=0; i<num_output; i++)
  qoutput[i] =ret[i];


dequantize(qoutput, output, num_output, 0, act_scale*weight_scale);
```

quantization을 끝냈으니 이제 하드웨어로 data를 보내면 된다. 타일링은 fpga 하드웨어 위에서 하므로 타일링 과정없이 memcpy 함수로 qinput, qlarge_mat 데이터를 그대로 옮겨준다. 그런데 하드웨어에서 받아오는 data의 inpu, output의 수를 모르므로 이를 알려주기 위해 *(output_+2), (output_+3) 포인터에 값을 입력해 정보를 보내준다. 이렇게 보낸 데이터는 myipS00_AXI의 slv_reg로 보내지게 된다.

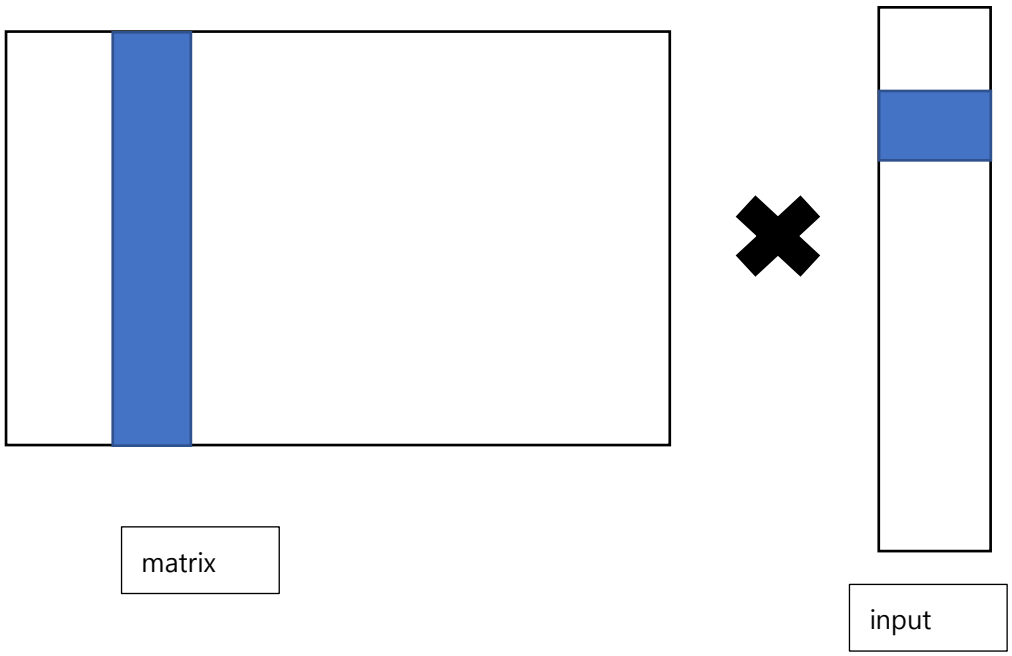
```
always@(posedge S_AXI_ACLK)begin

    if(reset ==1 || resett ==1)begin
        vecon <=0;
    end

    if(start ==1)begin
      num_in <= num_input;
       num_out <= num_output;
     //   num_in <= 32'd300;
    //    num_out <= 32'd100;
        vecon <= 1;
    end

    if(vecon ==1 )begin
        vecon <=0;
        matpos <= num_in+32'd4;
    end


end
```

이제 verilog를 보면, 먼저 start 신호가 들어옴으로 모듈이 시작된다. start 신호가 들어오면 아까 slv_reg로 보낸 num_input, num_output 정보를 내부 레지스터에 저장한다. 그리고 vecon 신호를 주어 vector값을 bram에서부터 읽어온다.

모든 layer의 데이터를 한번에 처리해야 하는데 한번에 reg에 올리기에는 너무 크기가 크다. 그래서 타일링을 실시하여 계산을 쪼개서 해주어야 한다. V0, V1에서 cpp 코드상에서 구현 했던 m_size_와 v_size_를 받아 타일링을 하는 방식을 verilog 상에서 구현하기에는 non_blocking 등 고려해야할 상황이 많으므로 너무 복잡하다. 그래서 아래와 같이 타일링을 하였다.



먼저 위 그림과 같이 matrix와 input이 BRAM상에 존재 할 테다. 여기서 matix는 가장 첫 열만, input 은 가장 첫 행만 데이터를 reg로 읽어온다. 여기서 파란색 표시가 reg로의 이동을 의미한다. 그리고 읽어온 데이터 들만 값을 곱 연산을 해주고 output 배열 레지스터에 저장해둔다.

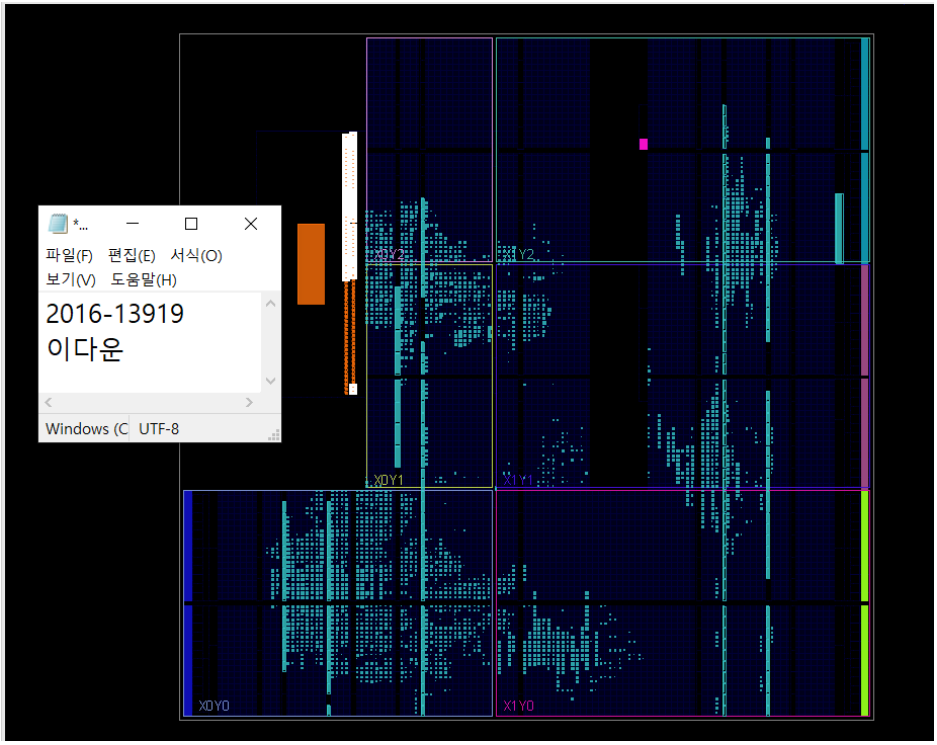


그리고 위 사진처럼 다음 matrix 열과 input 행을 내부 reg로 read해준다. 그리고 마찬가지로 reg로 읽어온 값들만 곱 연산을 해주고 결과를 output 배열에 누적해준다.

이런 방식으로 끝 열까지 하나씩 하는 방법으로 타일링을 구현해 주었다. 위 설명을 보면 알 수 있듯이 vec 로드, mat 로드, 과정을 반복해준다. 아래와 같이 자릿수를 의미하는 outpoint가 끝에 도달하면 과정을 중단하고 값을 bram에 저장해준다.

```
if(save ==1)begin
    if(outpoint < num_in -1)begin // goto mat
        outpoint <= outpoint + 32'd1;
        nextmat <=1;
    end
    else begin  // finish
        save1 <= 1;
        s_c <= 0;
    end
end

if(save1==1) begin
    we<= 4'b1111;
    wrdata <= out[s_c];
    addr2 <= addr3;
    addr3 <= addr3 +32'd4;
    s_c <= s_c + 32'd1;
    if(s_c == num_out)begin
        save1 <= 0;
        s_c <= 0;
        addr2 <=0;
        outpoint<=0;
        addr3 <= 0;
        done1 <= 1;
```

4. 결과



먼저 보드에 implementation한 결과다. v0에 비교하면 확실히 하드웨어 부담은 줄었들었지만 v1과 비교하면 큰 차이는 없어 보인다.

```
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 741,
 'm_size': 64,
 'total_image': 100,
 'total_time': 24.945292999999992,
 'v_size': 64}

=> Accuracy should be 1.0

[*] Arguments: Namespace(a_bits=8, m_size=64, network='cnn', num_test_images=100, quantized=True, run_type='cpu', v_size=64, w_bits=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
random: nonblocking pool is initialized
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 741,
 'm_size': 64,
 'total_image': 100,
 'total_time': 53.016566,
 'v_size': 64}

=> Accuracy should be 1.0

[*] Arguments: Namespace(a_bits=8, m_size=64, network='cnn', num_test_images=100, quantized=True, run_type='fpga', v_size=64, w_bits=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 678,
 'm_size': 64,
 'total_image': 100,
 'total_time': 5.006910000000005,
 'v_size': 64}

=> Accuracy should be 1.0

xilinx@debian-zynq:/sdcard/hsd_v2$
```


bench 마크 결과다 똑같이 이미지 100개를 돌렸는데 하드웨어에서 타일링을 한결과 시간이 5초정도 밖에 안걸렸다. 이는 v0,v1과 비교하면 거의 1/6로 단축된 것이다.

5. Discussion

이번 V2 프로젝트를 통하여 하드웨어 가속기로 성능을 cpu 연산과 비교했을 때 폭발적으로 증가시킬 수 있음을 알게 되었다. 타일링 구현은 cpu 코드상에서 구현하는 것이 훨씬 간단하여 결과 소모시간에 큰 차이가 없을 것이라고 생각했는데 예상과 반대로 1/6로 시간이 많이 감소하였다. 시간관계상 구현은 못했지만 더 해볼만 한 최적화는 32bit로 데이터를 cpu와 fpga 하드웨어와 transfer하였는데 int 타입대신 char 타입을 사용하여 8bit로 줄어들게 하는 것이다. 비트수가 줄어든 만큼 시간도 빨라질 것이라고 예상된다.

V2 프로젝트를 마지막으로 하드웨어실습 및 설계 수업이 끝나게 되었다. 하드웨어 설계 수업을 따라가기 위해 컴퓨터 구조적 지식과 소프트웨어 지식 모두 총동원하여 했었던 것 같다. 나는 소프트웨어 쪽이 적성에 맞다고 생각하지만 하드웨어 설계를 통해 컴퓨터 전체에 대해 지식이 넓어 지고 많은 것이 남게 된 수업이었다.

6.code

```
initial begin
    we <= 0;
    addr <= 0;
    vecsize <=0;
    v_c <=0;

    matsize <=0;
    m_c <=0;

    outpoint <= 0;
    save <= 0;
    addr2 <= 0;
    addr3 <= 0;
    resett <= 0;


end
```

```
always@(posedge S_AXI_ACLK)begin

    if(reset ==1 || resett ==1)begin
        vecon <=0;
    end

    if(start ==1)begin
      num_in <= num_input;
       num_out <= num_output;
     //   num_in <= 32'd300;
    //    num_out <= 32'd100;
        vecon <= 1;
    end

    if(vecon ==1 )begin
        vecon <=0;
        matpos <= num_in*32'd4;
    end


end
```

```
always@(posedge S_AXI_ACLK)begin

    if(reset ==1 || resett==1)begin
        addr <=0;
        vecsize <=0;
        v_c <= 0;
        firstmat <= 0;
        vecin <= 0;
    end
    if(nextmat==1)begin
        addr<= outpoint*32'd4;
        vecin <=1;
    end

    if(vecon ==1) begin
        addr<= outpoint*32'd4;
        vecin <=1;
    end

    if(vecin ==1)begin
        vec <= BRAM_RDDATA;
        vecin <=0;
        firstmat <= 1;

    end

    if(firstmat ==1) firstmat <=0;

end
```

```
always@(posedge S_AXI_ACLK)begin

    if(reset ==1 || resett ==1)begin
        addr1 <= 0;
        matsize <=0;
        m_c <=0;
        cal <=0;
        firstmat1 <= 0;
        firstmat2 <= 0;

    end


    if(firstmat ==1)begin
        addr1 <= matpos + outpoint*32'd4;
        firstmat1 <= 1;
        matsize <=0;
    end

//    if(nextmat ==1)begin
//        addr1 <= matpos + 4*outpoint * num_in;
//        firstmat1 <= 1;
//        matsize <=0;
//        m_c <= 0;
//    end
```

```
    if(firstmat1 ==1)begin
        addr1 <= addr1+ num_in*32'd4;
        matsize <= matsize + 32'd1;
        firstmat2<=1;
        if(matsize == num_out-32'd1 ) firstmat1 <= 0;
    end

    if(firstmat2==1 || firstmat1 ==1) begin
        mat[m_c] <= BRAM_RDDATA;
        m_c <= m_c + 32'd1;
        if(m_c ==  num_out-32'd1)begin
            m_c <= 0;
            firstmat2 <= 0;
            cal <=1;
        end

    end
    if(cal ==1) cal <= 0;

end
```

```
always@(posedge S_AXI_ACLK)begin
    if(reset ==1 || resett ==1 )begin

        i<=0;
        cal1<=0;
        cal2<=0;
        for(iii=0; iii<64; iii=iii+1)begin
            out[iii] <= 32'd0;
            temp[iii] <= 32'd0;
        end


    end

    if(cal ==1)begin
        cal1<=1;
        i<=0;

    end
```

```
    if(cal1 ==1)begin
        for(iii=0; iii<64; iii=iii+1)begin
            temp[iii] <= vec * mat[iii];
        end
        cal1 <= 0;
        cal2 <= 1;
    end

    if(cal2 ==1)begin
        for(iii=0; iii<64; iii=iii+1)begin
            out[iii] <= temp[iii] + out[iii];
        end
        cal2 <= 0;

        if(1==1)begin
            save <= 1;
            i <= 0;
        end
        else begin
            i <= i+1;
            cal1 <= 1;
        end
    end

    if(save ==1) save <= 0;

end
```

```
always@(posedge S_AXI_ACLK)begin

    if(reset ==1 || resett ==1)begin
        outpoint <= 32'd0;
        save1 <= 0;
        addr2 <= 0;
        addr3 <= 0;
        s_c <= 0;
        done1 <= 0;
        nextmat <=0;
        wrdata <= 0;
        we <= 4'b0000;
        done11<=0;
        done111<=0;
    end

   if(save ==1)begin
        if(outpoint < num_in -1)begin // goto mat
            outpoint <= outpoint + 32'd1;
            nextmat <=1;
        end
        else begin  // finish
            save1 <= 1;
            s_c <= 0;
        end
       .
    if(save1==1) begin
        we<= 4'b1111;
        wrdata <= out[s_c];
        addr2 <= addr3;
        addr3 <= addr3 +32'd4;
        s_c <= s_c + 32'd1;
        if(s_c == num_out)begin
            save1 <= 0;
            s_c <= 0;
            addr2 <=0;
            outpoint<=0;
            addr3 <= 0;
            done1 <= 1;
            resett <=1;
            we <= 4'b0000;
        end

    end

    if(done1 ==1) done1 <= 0;
    if(nextmat ==1) nextmat <=0;
    if(resett ==1) resett<=0;

end
```