CS411 project 3 Document

Release date: 2023.11.17
Due date : 2023.12.08

- GOAL
  - Integrate FPGA into CPU system to implement end-to-end object detection algorithm
    - [**STEP** 1] Extend FPGA INT8 matrix multiplication to INT16 matrix multiplication
    - [**STEP** 2] To run conv on FPGA INT16 matrix multiplication, quantize input activations and weights of conv, and dequantize output activations of conv
    - [**STEP** 3] Tiling matrix multiplication (software-wise)
    - [**STEP** 4] Tiling matrix multiplication (hardware-wise)
    - [**STEP** 5] Integrate STEP 1-4 to run end-to-end object detection algorithm

Description of each steps

- STEP 0
  - You should execute below instruction

```
mkdir -p ./data/weight
wget -O ./data/weight/yolov2-tiny-voc.weights https://pjreddie.com/media/files/yolov2-tiny-voc.weights
```

- STEP 1
  - Description: After project 2, you will be ready to run FPGA signed INT8 matrix multiplication. However, with a limited expression range of signed INT8 (-128 ~ 127), it may be hard to apply to real-world object detection algorithms. Therefore, it is time to extend FPGA INT8 matrix signed multiplication to INT16 signed matrix multiplication.

  - TODO: extend FPGA INT8 matrix multiplication to INT16 matrix multiplication

  - How to grade: Extend and synthesize verilog file, then run "test_STEP1.ipynb". If all of the tests pass, you will get full credits of STEP 1.

- STEP 2
  - Description: After step 1, you will be ready to run FPGA signed INT16 matrix multiplication. However, because yolo-tiny-v2 should be run with floating points format, you should quantize FP format into INT format. To make the problem easier, project 3 restricts the problem to a **convolution case**.

    Here are several specification of STEP 2
    - Convolution will be transformed into matrix multiplication via im2col
      - im2col: transform the input tensor into two dimensional matrix
      - im2col will be provided, which you do not care about it
    - Convert will be based on below formula
      - A: $input\ activation\ tensor,\ W$: $weight\ activation\ tensor$
      - $M\ =\ 2^m - 1$ (m = 8, 9, 10, … , 15 - TAs are used m = 12)
      - $D\ =\ max(max\ element(abs(A)),\ max\ element(abs(W)))$
      - $Quantize(x) := INT16(clip((x\ /\ D)\ *\ M,\ -M,\ M))$

        - $Dequantize_{matmul\ output}(x) := FP(x\ *\ (D\ /\ M)^2)$
    - Convolution will be transformed like below formula
        - Before: $Conv(A,\ W)$
        - After: $Dequantize_{matmul\ output}(Conv(Quantize(A), Quantize(W)))$

- TODO: to run conv on FPGA INT16 matrix multiplication
    - First, convert input activations and weights of conv into INT16 (quantize)
    - Second, convert output activations of conv into FP (dequantize)

- How to grade: Implement quantization / dequantization function within “lib/model.py”, then run “test_STEP2,3.ipynb”. If prediction.jpg shows reasonably, you will get full credits of STEP 2.

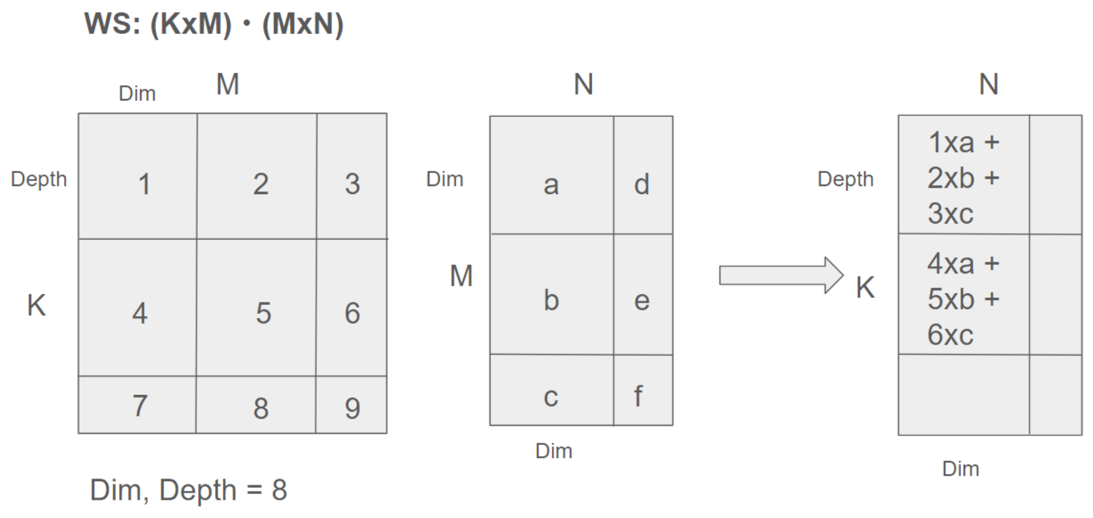
- STEP 3
    - Description: After step 2, you will be ready for small size (8x8) matrix multiplication. However, the actual matrix size is larger than 8x8, so you should do tiling technique. First, we will do tiling software-wise.

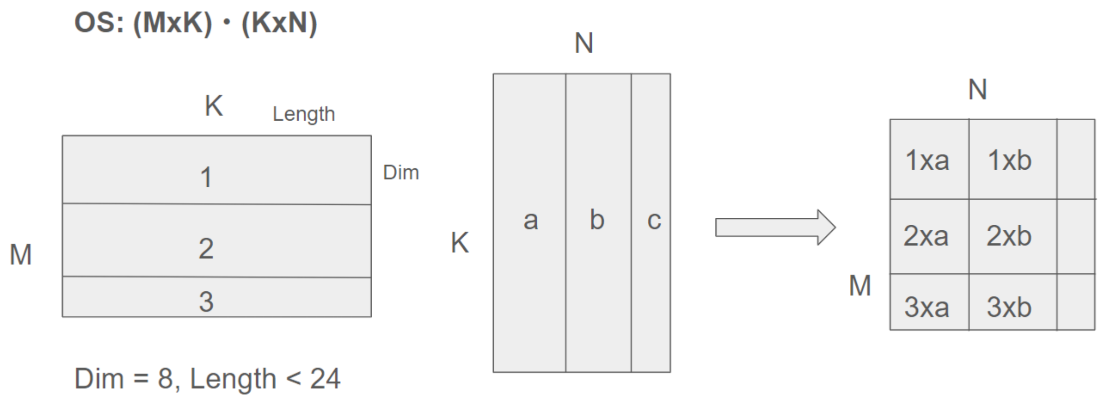
      Here are several specification of STEP 3
        - TILE WIDTH and HEIGHT should be 8

    - TODO: Tiling matrix multiplication (software-wise)

    - How to grade: Implement software-wise tiling function “matmul_CPU_tiling” within “lib/mytorch/conv.py”, then run “test_STEP2,3.ipynb”. If prediction.jpg shows reasonably, you will get full credits of STEP 3.

- STEP 4 [Run multiplication of larger matrices]
    - Description:
      [Goal]
      In this step, you will insert the large matrix into the FPGA (start implementation on your project 2 module). The matrix tiles for FPGA are already prepared in STEP3. In project 2, we perform matrix multiplication with size under 8 for each dimension. However, In project 3, your matmul_system can run the matrix multiplication with the size over 8. (for example, (24x24) x (24x24) matrix multiplication) Your module has to support **OS and WS mode**.
      [Accumulator]
      In weight stationary mode, you have to accumulate the partial sum matrix in the FPGA. Among the various ways, The way we choose is making the accumulator (new module). Accumulator can accumulate the partial sum from systolic array and send the final accumulated result to the output ram. [Constraint : DEPTH = 8]
      [Memory size : BRAM and ram(for each row and col)]
      To set the constraint for the tile size, we give you the memory size you should follow.
      [BRAM size = 2048, ram size > 256]
      The figure is recommended tiling for hardware.
        + NOW YOU DO NOT HAVE TO FOLLOW THE MEMORY OPTIMIZATION RULE IN PROJECT2.
        + You can padding the tiles and insert them into BRAM





- How to grade: After Implementation of FPGA module, then run the “test_STEP4.ipynb”. If you get 100 from the project3_test function, you can get full credits for STEP 4.

- STEP 5
  - Description: After step 5, you will be ready to run yolo-tiny-v2. Integrate all of the things which you have done.
  - TODO: Integrate STEP 1-4 to run end-to-end object detection algorithm
  - How to grade: Integrate STEP 1-4, then run “test_STEP5.ipynb”. If prediction.jpg shows reasonably like below, you will get full credits of STEP 5.
  - Notice:
    - For STEP5, it takes a long time (at least 10 min ~)
    - If you do not finish STEP4, you can use our bit / hwh file (CS411_teamN_step4) for STEP5

Submission and Scores

- Submission

    - Document (no limited)
    - Your Verilog codes (in folder name “verilog_code”)
    - Project folder (release.zip)
- Score
    - Design Document (20%)
        - Project STEP 2 (Quantization): Report result prediction.jpg for several m (m = 10, 11, … , 14)
        - Project STEP 3 (SW tiling): Explain how to implement
        - Project STEP 4 (HW tiling): Explain the improvement of your module compared to one of project 2.
            - ex1. What components do you add?
            - ex2. Any changes in FSM(controller)?
            - ex3. Overall data flow from jupyter(CPU) to FPGA and from FPGA to CPU; which layout do CPU stream large matrices into BRAM and how to do matmul within FPGA with BRAM data.
    - TEST (80 %)
        - For STEP 2**(5%)**, STEP 3**(15%)**, and STEP 5**(15%)**, If prediction.jpg shows reasonably, you will get full credits.
        - For STEP 1**(5%)** and STEP 4**(40%)**, you can check your score with test functions.