AAHLS Lab B Report – Sparse Matrix-Vector Multiplication

R11943109 何國瑋

Introduction

Sparse matrix is a matrix with only few non-zero elements in it. Some operations like multiplication become inefficient due to the excessive number of zero elements in these matrices. The matrix-vector multiplication is one of the operations that can be implemented more efficiently using different matrix representations and algorithms.

The matrix representation used in this lab is as shown in the following figure from [1]

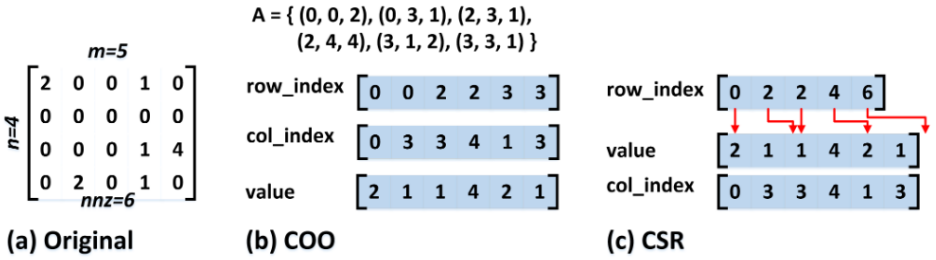


Fig. 1: Sparse matrix representation example ($n = 4$, $m = 5$, $nnz = 6$) (a) The original matrix, (b) The coordinate format (c) The Compressed Row format

The COO (coordinate) format represents the non-zero values and their coordinates in a matrix. The CSR (compressed row) format further simplifies the COOformat by sorting the row index and only record at which element the row index should be increased by 1.

The main idea is to multiply the non-zero elements in the same row by their corresponding multiplier in the vector and sum them together row-by-row.

Implementation 1 - “spmv” from pp4fpgas Examples

Introduction to the overall system

The top function “spmv”, as shown in the following code snippets, simply performs the multiplications row by row and obtains the summation when the multiplications in the same row are done.

```
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
    L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
#pragma HLS unroll factor=8
#pragma HLS pipeline
            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
```

The optimization like unroll, pipeline can be controlled using the #pragma statement here.

Implementation Details

To create the block design an run the test on pynq board, the interface should be defined as the following using #pragma statements.

```
8  #pragma HLS INTERFACE s_axilite port=rowPtr
9  #pragma HLS INTERFACE s_axilite port=columnIndex
10 #pragma HLS INTERFACE s_axilite port=values
11 #pragma HLS INTERFACE s_axilite port=y
12 #pragma HLS INTERFACE s_axilite port=x
13 #pragma HLS INTERFACE s_axilite port=return
```

The block design and bitstream can then be generated following the flow in Lab 1.

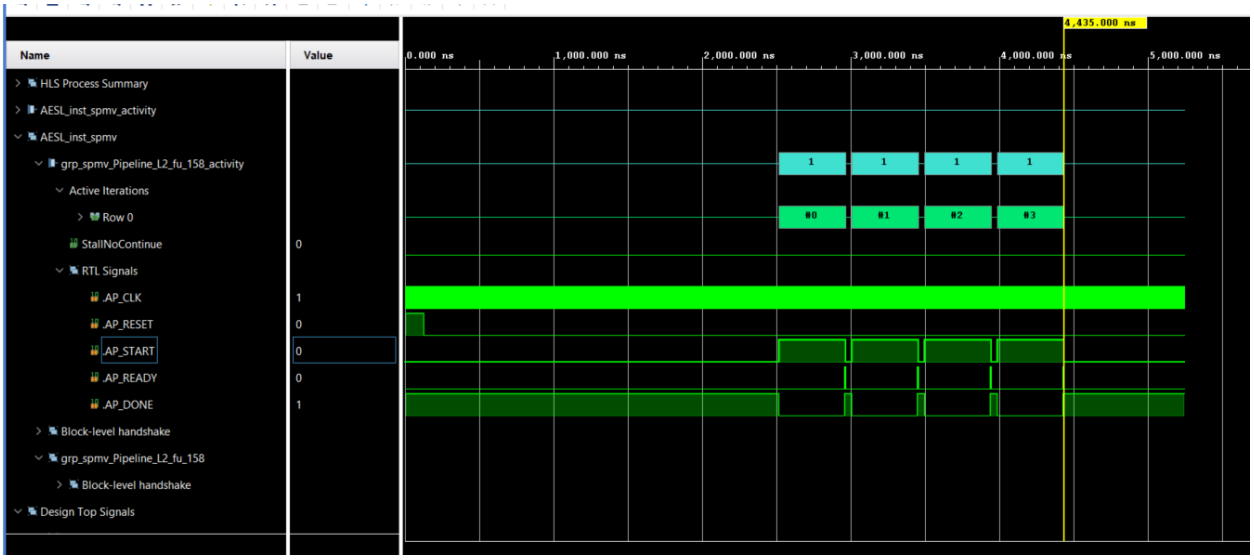
Observation

The following matrix is the input data used in testbench, given in the source code. The matrix is multiplied by the vector [1, 2, 3, 4].

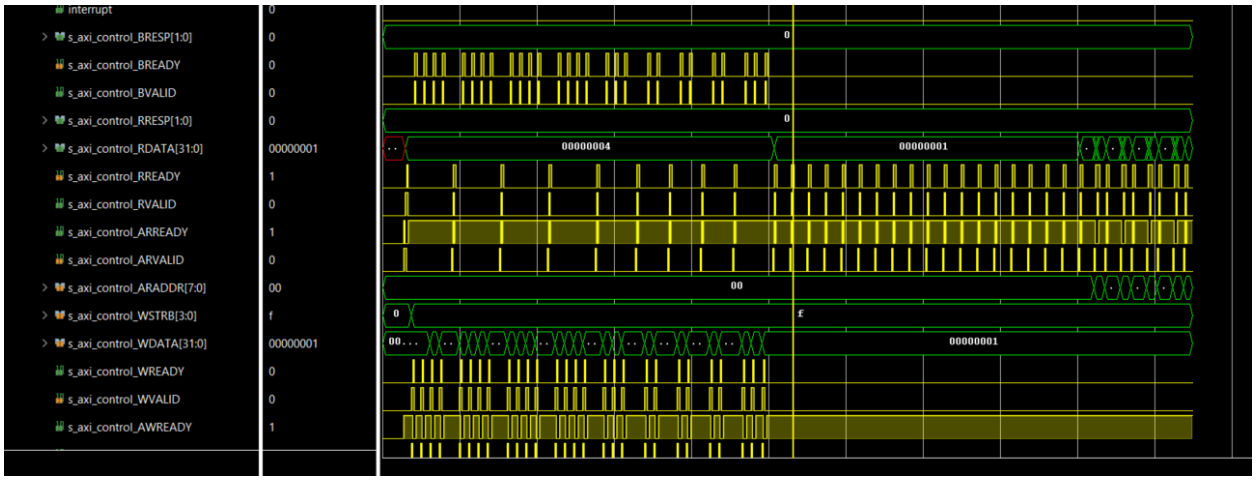
a) Matrix M

3	4	0	0
0	5	9	0
2	0	3	1
0	4	0	6

The timeline of co simulation result is shown in the following figure. The latency is about 4500 ns.



The read/write activity of data can be seen in the following timeline.



The result of running on pynq is shown in the following figure.

```
        regIP.write(0xc0 + 4*i, struct.pack('f', x[i]))
    regIP.write(0x00, 0x01)
    timeKernelStart = time()
    while (regIP.read(0x00) & 0x4) == 0x0:
        continue
    timeKernelEnd = time()
    y = []

    print("Result")
    print("============================")

    for i in range(SIZE):
        res = regIP.read(0x10+4*i)
        res = struct.pack('i', res)
        res = struct.unpack('f', res)[0]
        print(res)
        y.append(res)

    print("============================")
    print(f"time: {timeKernelEnd-timeKernelStart}")
    print("Exit process")
```

```
Entry: /usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py"
Result
============================
11.0
37.0
15.0
32.0
============================
time: 6.0558319091796875e-05
Exit process
```

Problem Encountered

I tried to enable configurable matrix size and number of non-zero values. But the code didn't pass the C-simulation due to some error in testbench.

```
INFO: [SIM 211-2] ************** CSIM start **************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   Compiling ../../../spmv.cpp in debug mode
   Generating csim.exe
@E Simulation failed.
ERROR: [SIM 211-100] CSim failed with errors.
INFO: [SIM 211-3] ************** CSIM finish **************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 3.36 seconds; current allocated memory: 260.430 MB.
4
    while executing
"source D:/Documents/HLS/LabB/hls/spmv_base/solution1/csim.tcl"
    invoked from within
"hls::main D:/Documents/HLS/LabB/hls/spmv_base/solution1/csim.tcl"
    ("uplevel" body line 1)
    invoked from within
"uplevel 1 hls::main {*}$newargs"
    (procedure "hls_proc" line 16)
    invoked from within
```

I didn't understand the error message, and the code even successfully returned zero when I checked the code step by step using the debugger. I googled the error message [SIM 211-100] and found some discussion about this issue involving the installation of build-essentials.

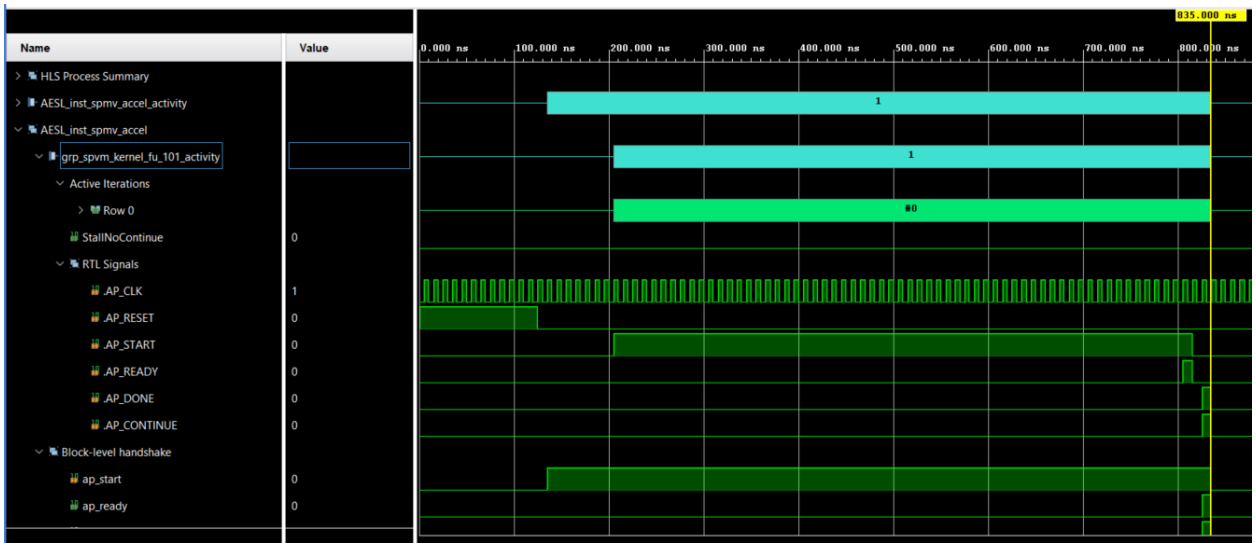
Implementation 2 - “fast_stream” from [1]

Introduction to the overall system

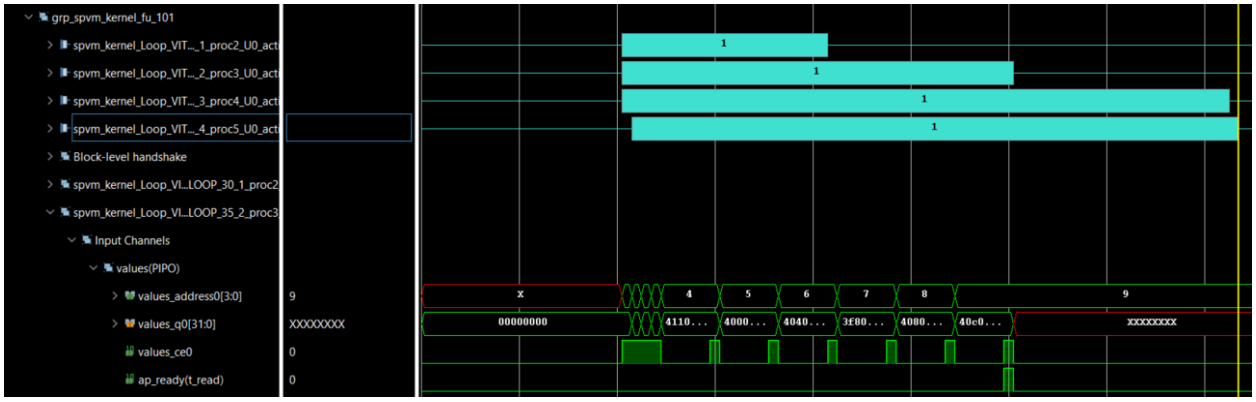
The proposed algorithm uses the concepts of loop pipelining, dataflow, and data streaming to optimize the performance. The streaming dataflow utilizes the memory bandwidth of FPGA efficiently.

Observation

The timeline of co-simulation result is shown in the following figure. The latency is about 800 ns and is way faster than the first implementation.



The following timeline shows the read/write behavior of the streaming dataflow.



Problem Encountered

The biggest problem encountered is how to modify the source code of this work so that it can be tested with the same input data as the first implementation. The modification involves redefining

some variables and datatypes. Also, the CSR representation of sparse matrix is slightly changed by replacing the row pointer with the difference of its elements.

GitHub Link

https://github.com/allen1236/AAHLS_LabB

Reference

[1] M. Hosseinabady and J. L. Nunez-Yanez, "A Streaming Dataflow Engine for Sparse Matrix-Vector Multiplication Using High-Level Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1272-1285, June 2020, doi: 10.1109/TCAD.2019.2912923.