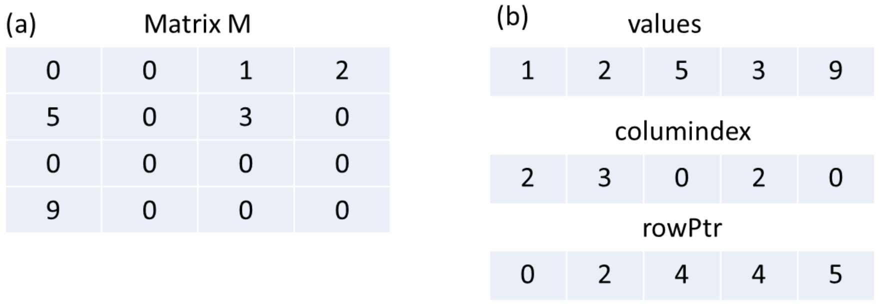
Application Acceleration with High-Level Synthesis

110062703 梁浩祥

● Overall system

Sparse matrix refers to a specific matrix type containing most of zero data. A lot of multiplication will be wasted when we multiply this type of matrix with the usual matrix multiplication method and standard data structure since most of the answers are zero. Hence, we should use an advanced data structure named “compressed row storage”(CRS). Figure 1.1 shows an example of 4*4 matrix M. Figure 1.1(a) is the normal representation of this matrix as a two-dimensional array of 16 elements. Figure 1.2(b) shows the CRS representation. CRS representation has three arrays. The values array holds the values of non-zero elements in the matrix. The columnindex and rowPtr array encode information regards to the location of those non-zero data. Columnindex stores the column of each non-zero data, and rowPtr stores cumulative the number of non-zero data in each row. Applying the CRS format will certainly introduce some increase in book-keeping information when the number of zero data is not enough in the matrix. But this negative effect will be compromised when the sparsity increase.

To be more precise, we can look at Figure 1.1. Figure 1.1(a) shows how the sparse matrix is stored in normal format. Only (0,2), (0,3), (1,0), (1,2), (3,1) have data, but it consumes a 4*4 storage, which is a lot of waste. On the other hand, we can separate the matrix into three arrays, values, columindex, and rowPtr. Values array holds the value of M. Columindex array holds the column index of those non-zero data. RowPtr array holds the cumulative number of data with the first elements always being zero, and the last element will always be the total number of data in M. Since row1 has two non-zero data, rowPtr[1] is 2. And row 2 also have two non-zero data, so the rowPtr[2] = rowPtr[1] + 2.



(a) Matrix M

0	0	1	2
5	0	3	0
0	0	0	0
9	0	0	0

(b) values

1	2	5	3	9

columindex

2	3	0	2	0

rowPtr

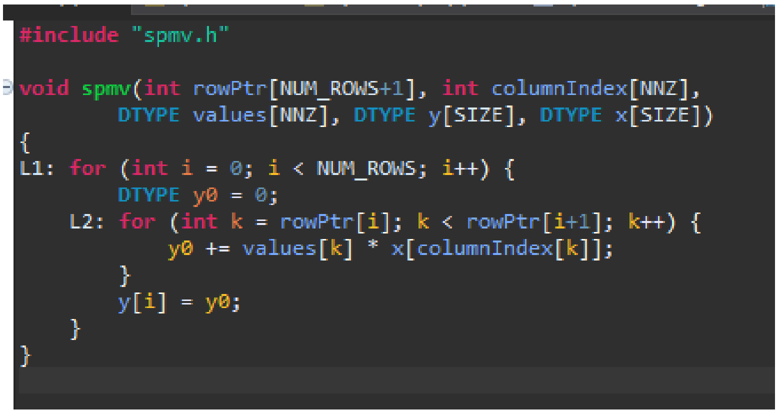
0	2	4	4	5

Figure 1.1 CRS demostration

When we want to calculate the CRS format data, we start from row0 and use columindex to achieve data and check whether there is still have non-zero data by rowPtr. Until there is no non-zero data anymore.

Implementation

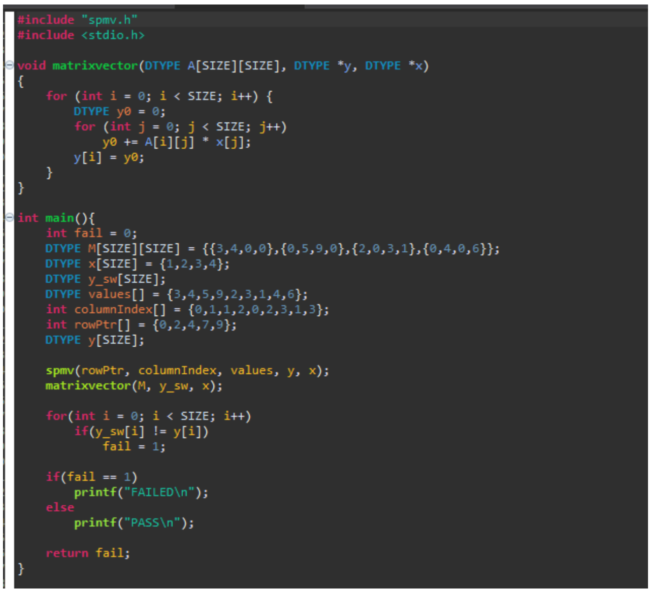
Based on our introduction of the CRS format, we can design a multiplication method as Figure 2.1. This algorithm contains two for loop. The first for loop will traverse all number of rows, and the second will multiply values from the y matrix and the corresponding columindex in the x matrix until we reach all non-zero data in this row.



```
#include "spmv.h"

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
          DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
    L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
}
```

Figure 2.1 spmv.cpp

And we need to design a testbench to verify our idea. Figure 2.2 shows the testbench design. We will compare our result from our algorithm and the restult from typical matrix multiplication. If the result is the same, we will get a PASS!.

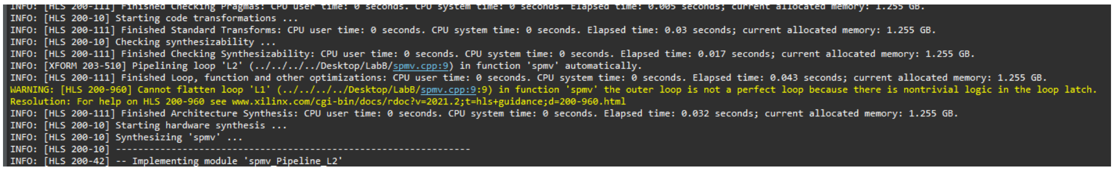


```
#include "spmv.h"
#include <stdio.h>

void matrixvector(DTYPE A[SIZE][SIZE], DTYPE *y, DTYPE *x)
{
    for (int i = 0; i < SIZE; i++) {
        DTYPE y0 = 0;
        for (int j = 0; j < SIZE; j++)
            y0 += A[i][j] * x[j];
        y[i] = y0;
    }
}

int main(){
    int fail = 0;
    DTYPE M[SIZE][SIZE] = {{3,4,0,0},{0,5,9,0},{2,0,3,1},{0,4,0,6}};
    DTYPE x[SIZE] = {1,2,3,4};
    DTYPE y_sw[SIZE];
    DTYPE values[] = {3,4,5,9,2,3,1,4,6};
    int columnIndex[] = {0,1,1,2,0,2,3,1,3};
    int rowPtr[] = {0,2,4,7,9};
    DTYPE y[SIZE];

    spmv(rowPtr, columnIndex, values, y, x);
    matrixvector(M, y_sw, x);

    for(int i = 0; i < SIZE; i++)
        if(y_sw[i] != y[i])
            fail = 1;

    if(fail == 1)
        printf("FAILED\n");
    else
        printf("PASS\n");

    return fail;
}
```

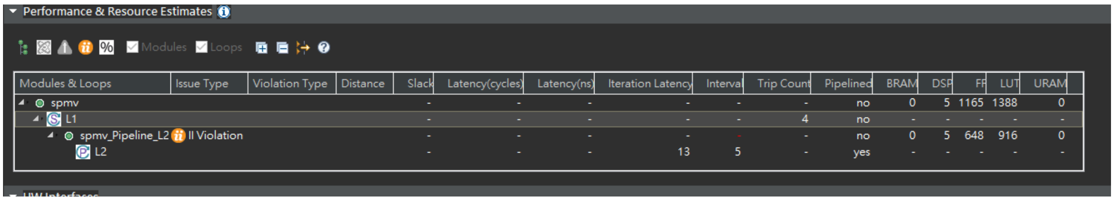
Figure 2.2 spmv testbench

- ## Analyze

If we didn’t apply any deriptive and run the synthesis, we will get the baseline model. We will notice that Vitis will automatilly pipline the L2 with the iteration latency is 13, interval is 5, trip count is 4, FF usage is 1165, and LUT is 1388. And we can see the program use one mul and one add for the baseline.



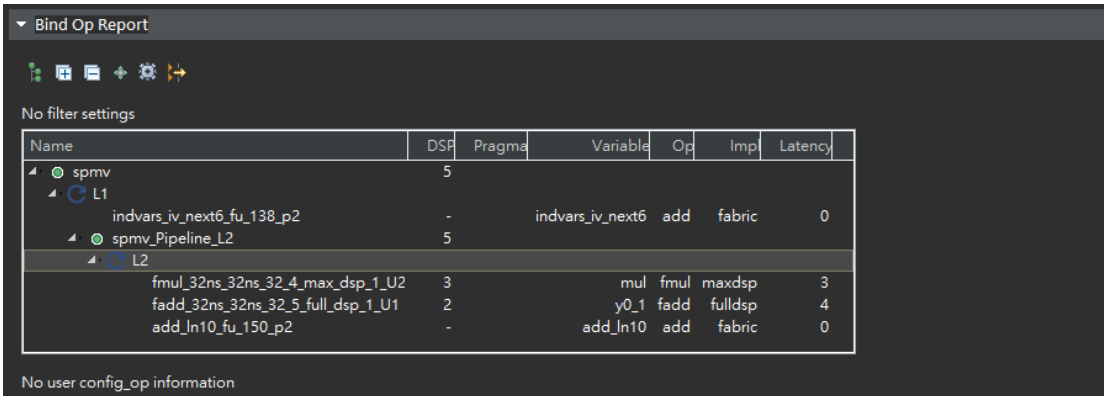
```
INFO: [HLS 200-111] Finished Checking Pragmas: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.005 seconds; current allocated memory: 1.255 GB.
INFO: [HLS 200-10] Starting code transformations ...
INFO: [HLS 200-111] Finished Standard Transforms: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.03 seconds; current allocated memory: 1.255 GB.
INFO: [HLS 200-10] Checking synthesizability ...
INFO: [HLS 200-111] Finished Checking Synthesizability: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.017 seconds; current allocated memory: 1.255 GB.
INFO: [XFORM 203-510] Pipelining loop 'L2' (../../../../Desktop/LabB/spmv.cpp:9) in function 'spmv' automatically.
INFO: [HLS 200-111] Finished Loop, function and other optimizations: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.043 seconds; current allocated memory: 1.255 GB.
WARNING: [HLS 200-960] Cannot flatten loop 'L1' (../../../../Desktop/LabB/spmv.cpp:9:9) in function 'spmv' the outer loop is not a perfect loop because there is nontrivial logic in the loop latch.
Resolution: For help on HLS 200-960 see www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=hls+guidance;d=200-960.html
INFO: [HLS 200-111] Finished Architecture Synthesis: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.032 seconds; current allocated memory: 1.255 GB.
INFO: [HLS 200-10] Starting hardware synthesis ...
INFO: [HLS 200-10] Synthesizing 'spmv' ...
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [HLS 200-42] -- Implementing module 'spmv_Pipeline_L2'
```



Performance & Resource Estimates

☑ Modules ☑ Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
spmv				-	-	-	-	-	-	no	0	5	1165	1388	0
L1				-	-	-	-	-	4	no	-	-	-	-	-
spmv_Pipeline_L2	II Violation			-	-	-	-	-	-	no	0	5	648	916	0
L2				-	-	-	13	5	-	yes	-	-	-	-	-



Bind Op Report

No filter settings

Name	DSP	Pragma	Variable	Op	Impl	Latency
spmv	5					
L1						
indvars_iv_next6_fu_138_p2	-		indvars_iv_next6	add	fabric	0
spmv_Pipeline_L2	5					
L2						
fmul_32ns_32ns_32_4_max_dsp_1_U2	3		mul	fmul	maxdsp	3
fadd_32ns_32ns_32_5_full_dsp_1_U1	2		y0_1	fadd	fulldsp	4
add_ln10_fu_150_p2	-		add_ln10	add	fabric	0

No user config_op information

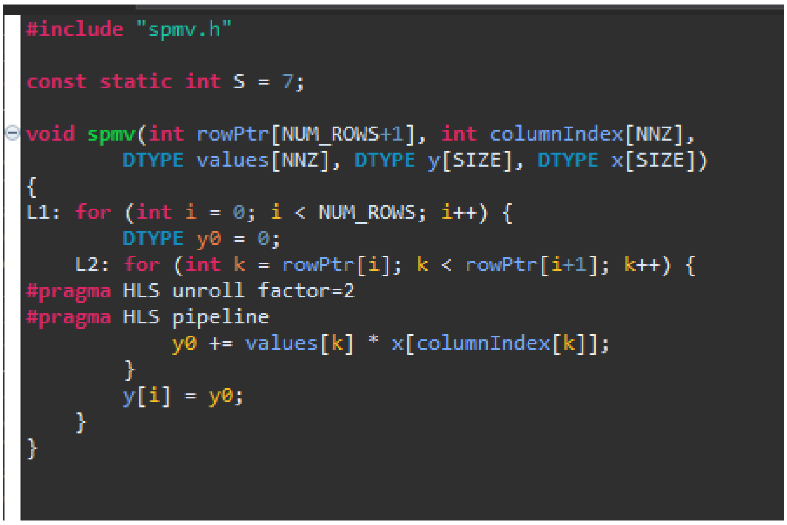
- ## Optimization

After we got the baseline model we can try to optimize the performance. I decide to try pipline, unroll, cyclic, block deriptive to optimize the performance.

case	L1	L2	time	Iteration latency	Interval	Trip count	DSP	DD	LUT
Baseline	x	x	7.256	13	5	4	5	1165	1388
1	x	pipline	7.256	13	5	4	5	1165	1388
2	pipline	x	7.256	12	5	4	5	1165	1345
3	Unroll 2	x	7.256	13	5	2	5	1466	1799
4	x	Pipline Unroll 2	7.256	18	10	4	5	1466	1655
5	x	Pipline Unroll 2 Cyclic 2	7.256	18	10	4	5	1660	1874

6	x	Pipline Unroll 4	7.256	28	20	4	5	1677	2120	
7	X	Pipline Unroll 4 Cyclic 4	7.247	27	20	4	5	2285	2728	
8	X	Pipline Unroll 8	7.256	48	40	4	5	2065	2988	
9	X	Pipline Unroll 8 Cyclic 8	7.561	47	40	4	5	3399	4724	
10	x	Pipline Unroll 8 block 8	7.274	47	40	4	5	3969	4293	

As we can see, the best performance is case3, which has a lower trip count. But one thing I don’t understand is as pipelining and unrolling L2, the iteration latency and interval, instead of decrease, increase. My suspicion is since variables in L2 are not constant, the tool thinks there is dependence inside the loop. Therefore, the tool generates double resources but still has to wait for the previous operation, making latency double and more resources usage. Also, we have an example of how I set deriptive as figure 2.2. About cyclic array partition and block array partition we have an example as figure 2.3.



```
#include "spmv.h"

const static int S = 7;

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
        DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
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=2
#pragma HLS pipeline
            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
}
```

Figure 2.2 example of how to use deriptive.

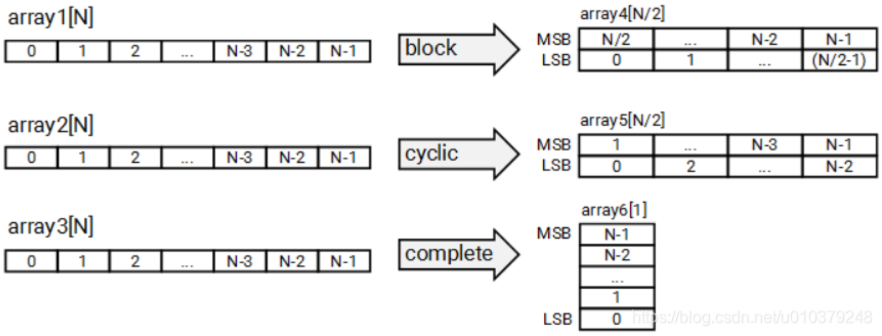


Figure 2.3 cyclic array example

I also try to set pipline II to two, but it will cause the time violation.

- ## Observered and Learned

In this lab, I learn how to use Vitis HLS and set deriptives to optimize the performance. And unlike the Stratus HLS flow, Vitis HLS has many disscussion and open realeased documentation online, which is very helpful for us to deeperly understand how to use this tool and making me want to switch from Stratus flow to Vitis HLS flow.

- ## Problems

The main problem is I don’t know how to parallel L2 loop. I try to modify the structure but not helping.

- Github : https://github.com/Liannthu/HLS_LabB_no9