HLS LabB Report

胡晉瑄

Overview

- Introduction to Sparse Matrix Vector Multiplication
- Optimization
  - Baseline
  - Partial Unroll
  - Streaming
  - fast_streaming
- Overall result
- Summary

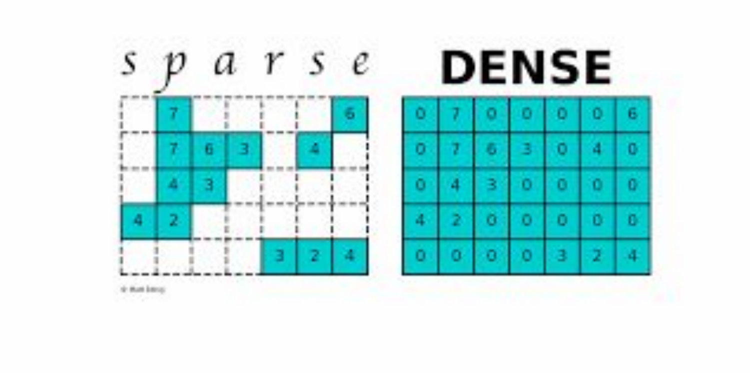
Why SpMV?

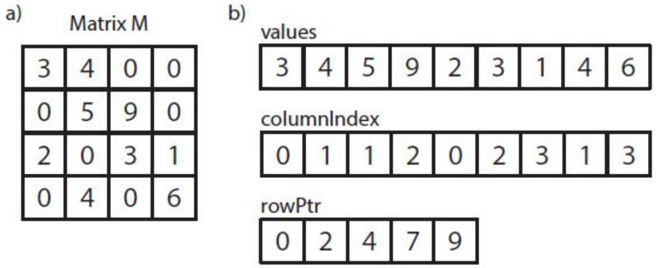
Pros of sparse format:

- Reduction of memory footprint
  - too many zeros
- Reduction of execution time
  - without zero calculation
- Scalable representation of matrix
  - size grows by NNZ

Applications:

- Graph computation
- PageRank
- …

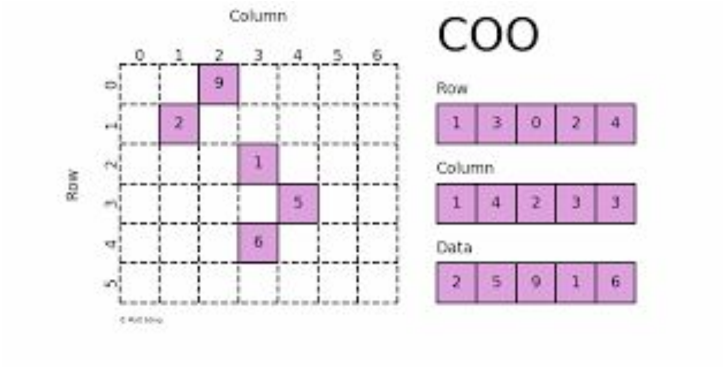


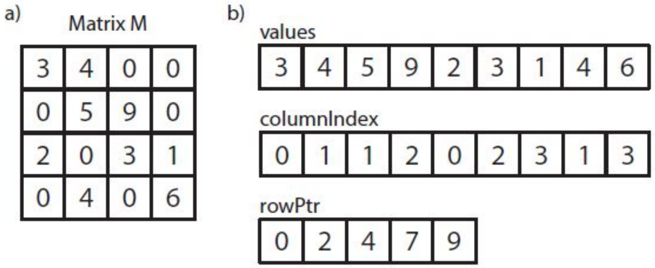


Sparse Format

1. COO (Coordinate list)
2. CSR (Compressed sparse row)
3. CSC (Compressed sparse column)
4. …

We will focus on CSR in this presentation



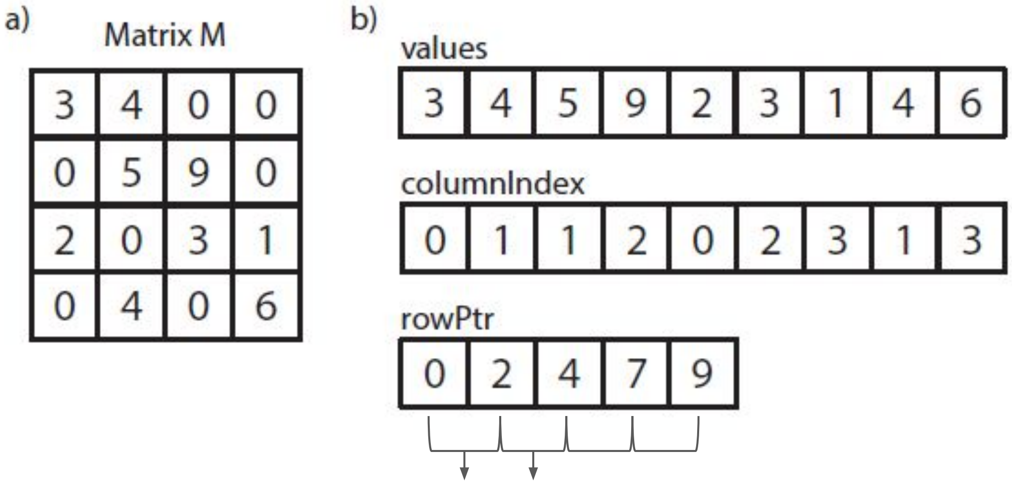


CSR format

Sparse Format - CSR

Data structure

1. values
   - holds NZ in raster order
2. col
   - holds NZ col index
3. rowPtr
   - encode row info.
   - # element in row
   - correponding index in val/col



row0 row1

2 elements
val/col index : 0~1

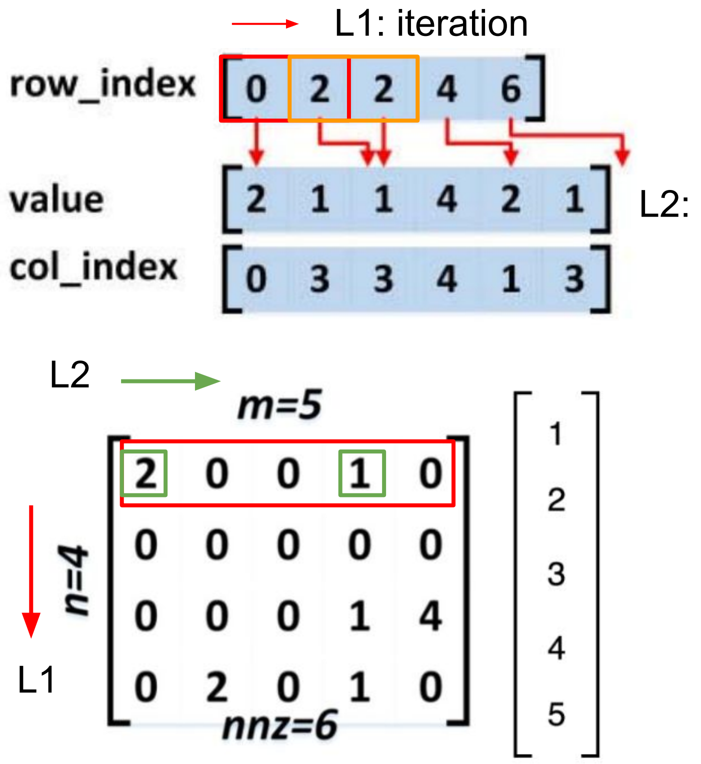
HLS Design

Test Environment

- Adj. Matrix:
  - size: 256*256
  - type: float
  - sparsity: 5%
  - NNZ = 3277 (size*sparsity)
- Input: X
  - size: 256
  - type: float
- Ouput: Y
  - size: 256
  - type: float

Baseline

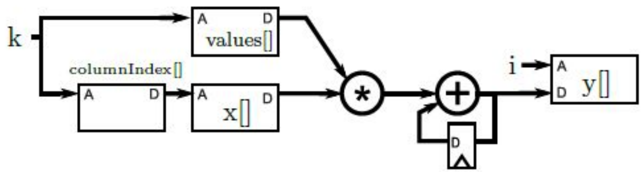
```
#include "spmv.h"
void spmv(  int rowPtr[NUM_ROWS+1],
            int columnIndex[NNZ],
            DTYPE values[NNZ],
            DTYPE y[SIZE],
            DTYPE x[SIZE])
{
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
        L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {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



nnz: number of non-zero

Baseline - Architecture

```
#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;
    }
}
```



Baseline - Timing

```c
#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;
    }
}
```

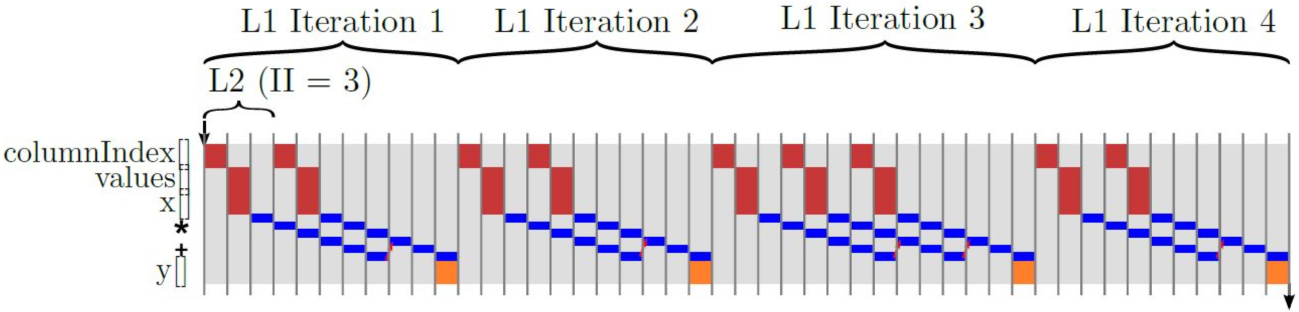
II = 5 、iteration Latency = 13

Modules & Loops	Vi	Di	s)	s)	Iteration Latency	Interval	Trip Count	Pipelined
spmv			-	-	-	-	-	no
L1			-	-	-	-	256	no
spmv_Pipeline_L2			-	-	-	-	-	no
L2			-	-	13	5	-	yes

Cosim Latency: 19458

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
spmv				19458	19458	19458
L1						
spmv_Pipeline_L2	75	127	27	72	123	23
L2	75	127	47	73	124	24

II: unknown
latency: 19458

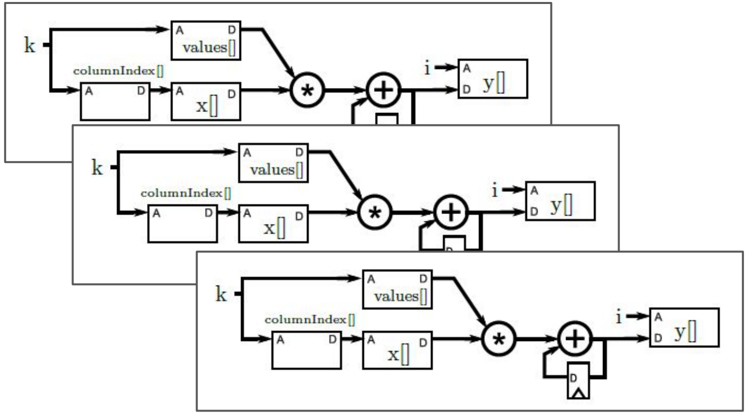


floating point operation: II=5

Partial Unroll - Architecture

```
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
    L2_1: for (int k = rowPtr[i]; k < rowPtr[i+1]; k+=S) {
#pragma HLS pipeline II=S
            DTYPE yt = values[k] * x[columnIndex[k]];
            L2_2: for (int j = 1; j < S; j++) {
                if (k + j < rowPtr[i + 1]) {
                    yt += values[k+j] * x[columnIndex[k+j]];
                }
            }
            y0 += yt;
        }
        y[i] = y0;
    }
}
```

S



Partial Unroll - Timing

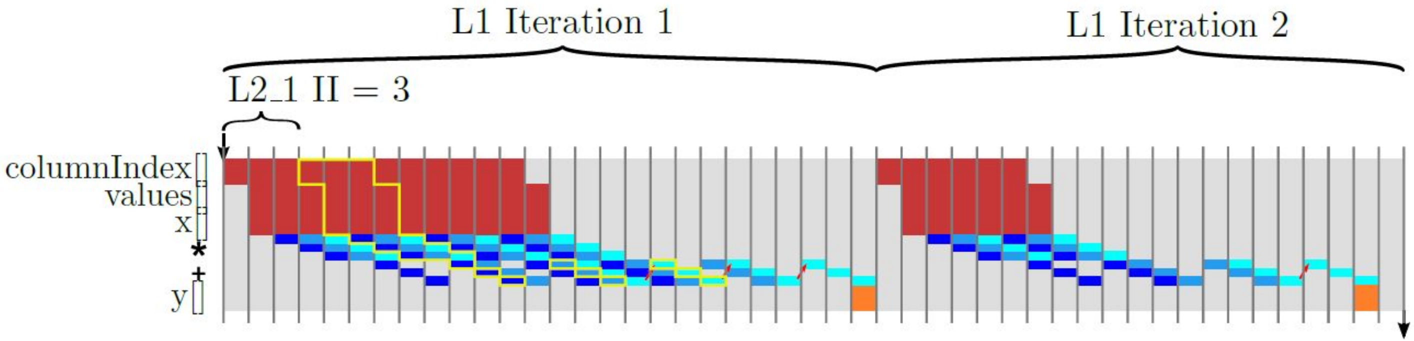
```
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
    L2_1: for (int k = rowPtr[i]; k < rowPtr[i+1]; k+=S) {
#pragma HLS pipeline II=S
            DTYPE yt = values[k] * x[columnIndex[k]];
            L2_2: for (int j = 1; j < S; j++) {
                if (k + j < rowPtr[i + 1]) {
                    yt += values[k+j] * x[columnIndex[k+j]];
                }
            }
            y0 += yt;
        }
        y[i] = y0;
    }
}
```

II = 5 、iteration Latency = 37

Modules & Loops	Issı	Vi	[	ack	cles)	ns)	Iteration Latency	Interval	Trip Count	Pipelined
spmv				54	-	-	-	-	-	no
L1				-	-	-	-	-	256	no
spmv_Pipeline_L2_1				54	-	-	-	-	-	no
L2_1		Re		-	-	-	37	5	-	yes

Cosim Latency: 13007

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
spmv				13007	13007	13007
L1				13007	13007	13007
spmv_Pipeline_L2_1	50	61	41	46	57	37
L2_1	50	61	46	47	58	38

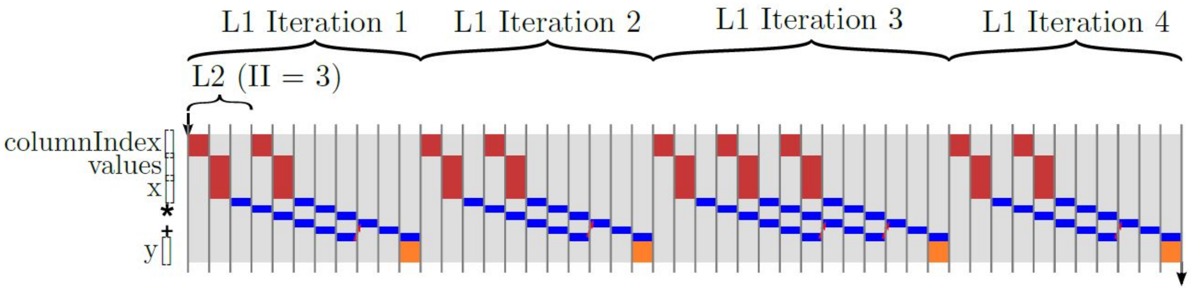


Baseline v.s. Partial Unroll

(L2 latency + (row_nnz-1)*II)

- L2 latency: 13
- II: 5
- 13+ (row_nnz-1)*5

latency 較短，但需要做較多次

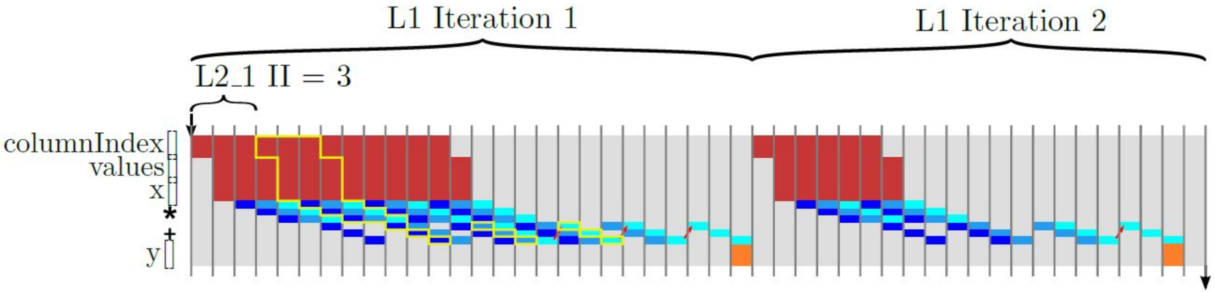


結論: 如果row_nnz較多，適合unroll，反之則適合baseline

(L2_1 latency + (row_nnz/S - 1)*II)

- L2_1 latency: 37
- II: 5
- S = 5
- 37+ (row_nnz/5-1)*5

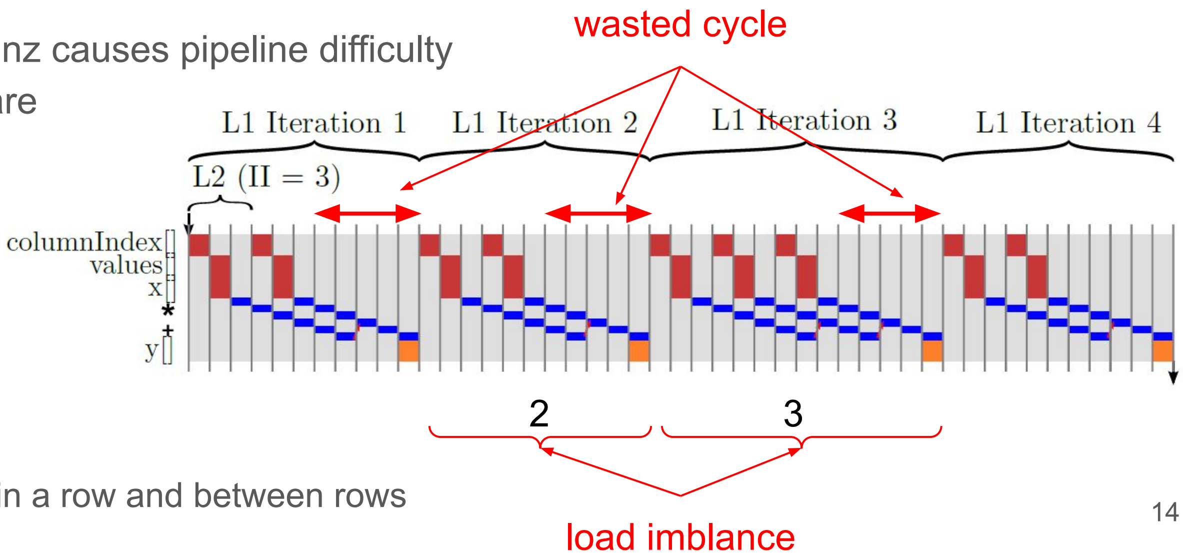
latency 較長，做較少次



Streaming - Motivation

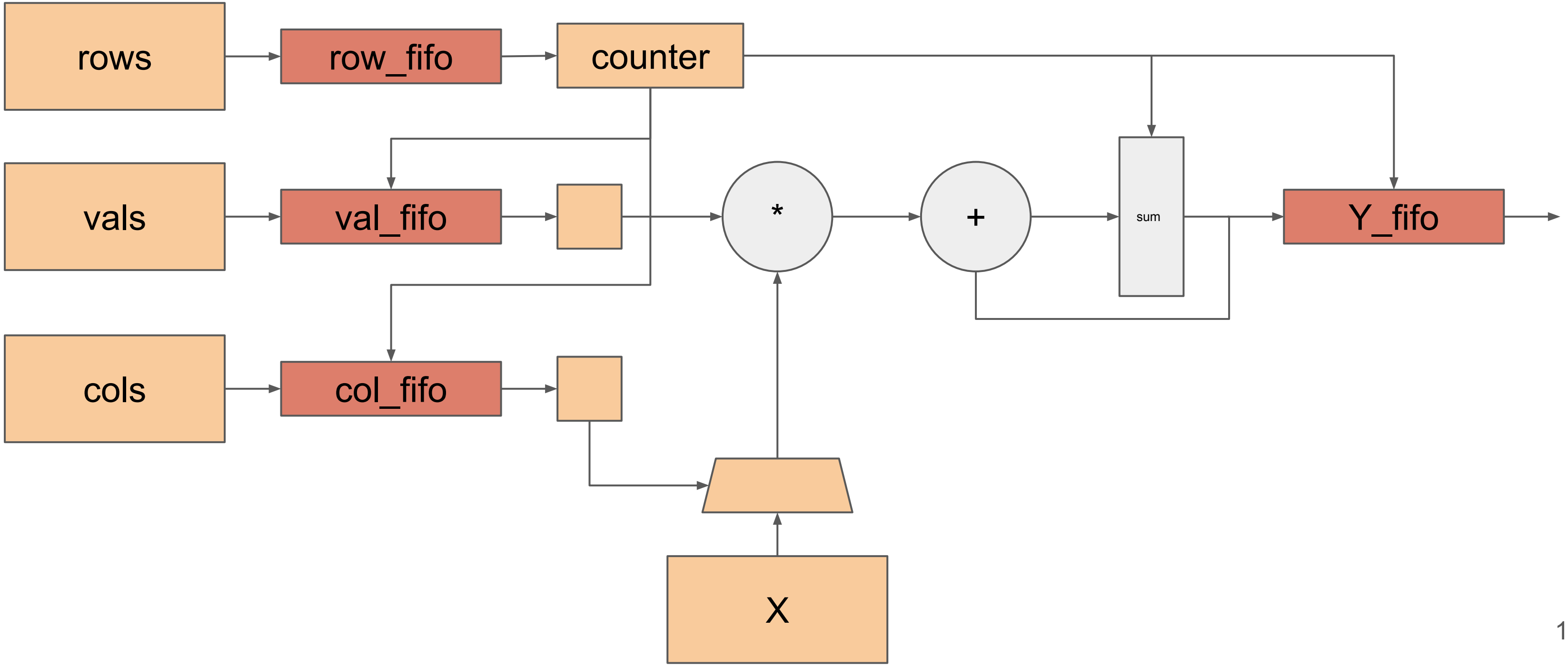
Paper: A Streaming Dataflow Engine for Sparse Matrix-Vector Multiplication Using High-Level Synthesis

- Bad memory bandwidth
  - SpMV is a memory-bound algorithm with irregular memory access
  - Lots of wasted cyle between L1 Iteration
- Load imblance
  - Different number of row_nnz causes pipeline difficulty
  - Hard to fully unroll hardware



❖ Streaming exploits the parallelism in a row and between rows

Streaming - Architecture



Streaming - Timing

```
COM: for (int i = 0; i < NNZ; i++) {
#pragma HLS PIPELINE II = 4
        if (col_left == 0) {
            col_left = rows_fifo.read();
            sum = 0;
        }
        value = values_fifo.read();
        col   = cols_fifo.read();
        sum   += value * x[col];
        col_left--;
        if (col_left == 0) {
            results_fifo << sum;
        }
    }
```

II = 4、COM Latency = 13

Modules & Loops	Is	Vi	D	Latency(cycles)	Latency(ns)	Iteration Latency	Interval
spmv				13642	1.360E5	-	13643
spmv_Pipeline_1				258	2.580E3	-	258
spmv_Pipeline_RS				259	2.590E3	-	259
spmv_kernel				13120	1.310E5	-	13119
spmv_kernel_Loop_rows_fifo_proc2				258	2.580E3	-	258
spmv_kernel_Loop_values_fifo_proc3				3279	3.279E4	-	3279
spmv_kernel_Loop_COM_proc4				13118	1.310E5	-	13118
COM		M	1	13116	1.310E5	13	4
spmv_kernel_Loop_Y_proc5				258	2.580E3	-	258

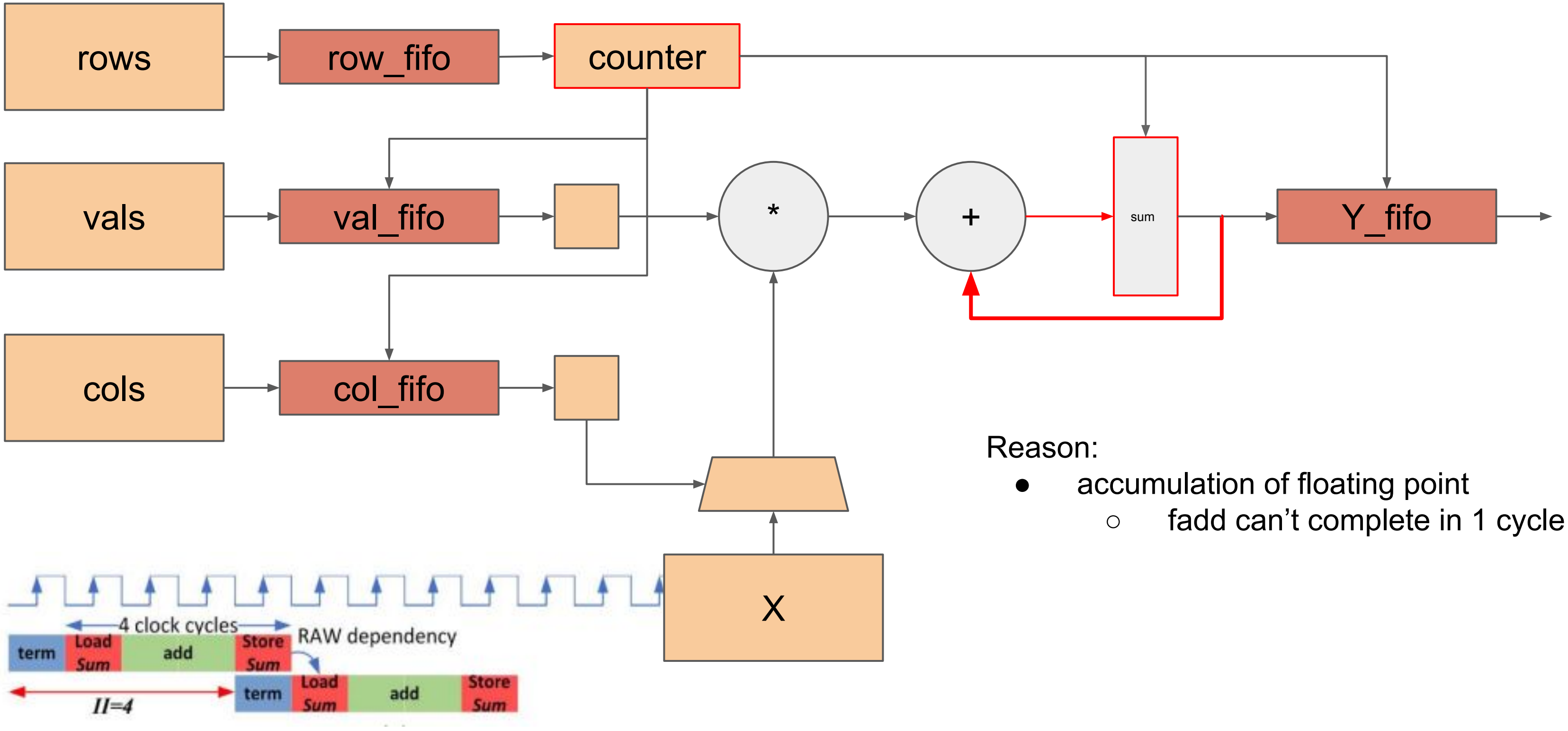
Cosim Latency: 13636

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
spmv				13636	13636	13636
spmv_Pipeline_1				256	256	256
spmv_Pipeline_RS				257	257	257
spmv_kernel				13118	13118	13118
spmv_kernel_Loop_rows_fifo_proc2				12977	12977	12977
spmv_kernel_Loop_values_fifo_proc3				13099	13099	13099
spmv_kernel_Loop_COM_proc4				13117	13117	13117
spmv_kernel_Loop_Y_proc5				13117	13117	13117

II = 4， why?

we expect achieving II = 1 using streaming architecture

Why II = 4



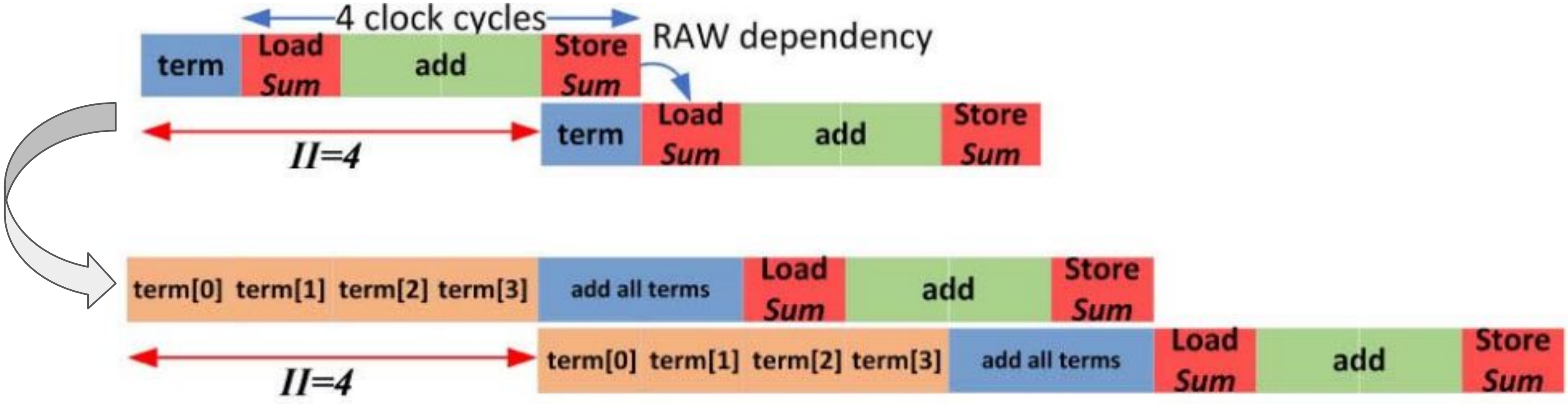
Reason:

- accumulation of floating point
  - fadd can't complete in 1 cycle

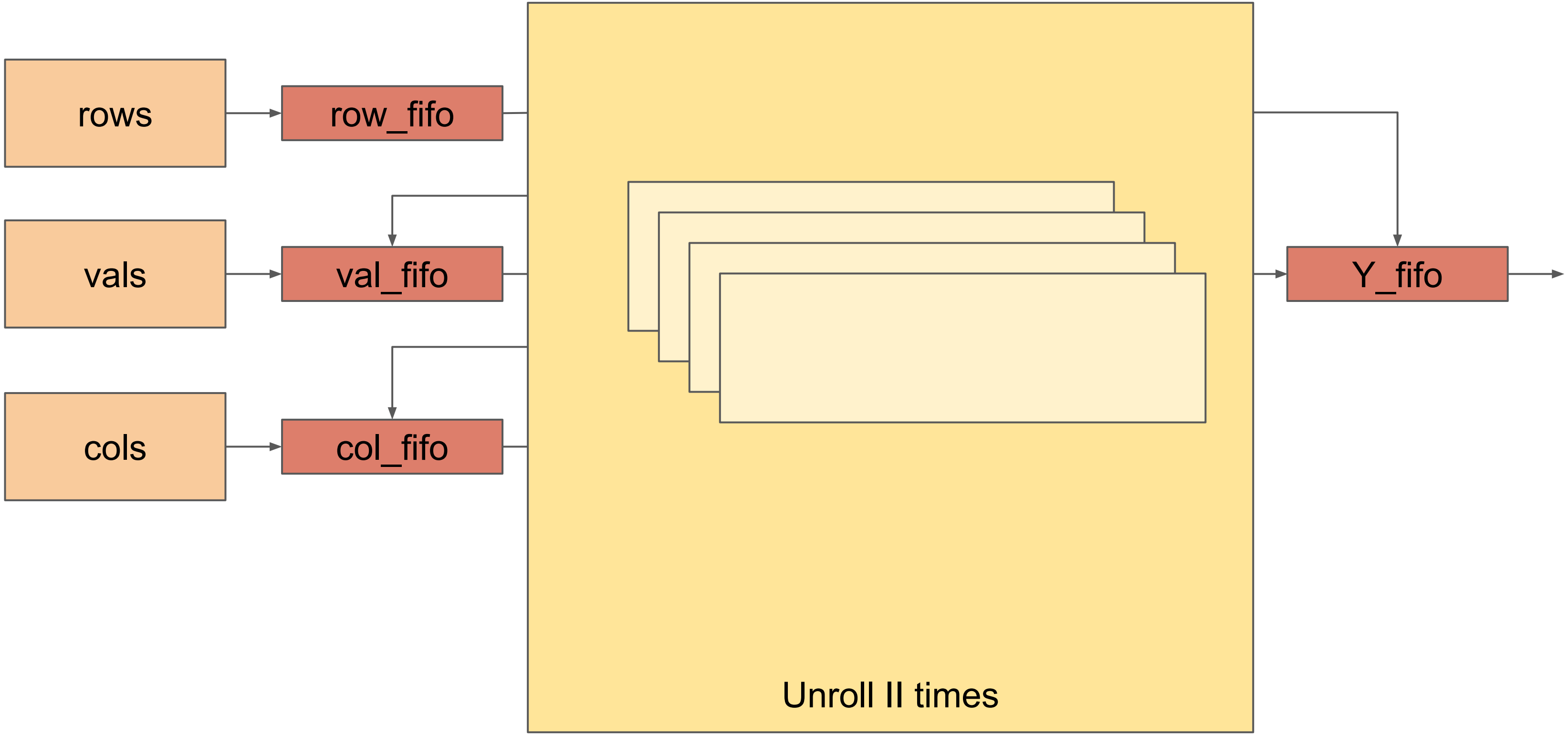
Fast Stream

Concept

- Overcome the high II bottleneck in COM process
  - processing multiple data to compensate negative impacts
  - 既然II=4了 不如就多算點吧



Fast Stream - Architecture



Preprocessing - Zero Padding

In order to work correctly, padding is necessary

- Padding will cause some penalty, but impact small

```
    int rows_length_pad[NUM_ROWS];
    int new_nnz = 0;
    for (int i = 0; i < NUM_ROWS; i++) {
#pragma HLS PIPELINE
        int r = rows_length[i];
        int r_diff = r % II;
        if (r == 0) {
            rows_length_pad[i] = II;
            new_nnz += II;
        } else if (r_diff != 0) {
            rows_length_pad[i] = r + (II - r_diff);
            new_nnz += r + (II - r_diff);
        } else {
            rows_length_pad[i] = r;
            new_nnz += r;
        }
    }
```

1	1	1	1
1			
2	2	2	
3	3		
4	4	4	4
4	4	4	

Fast Stream - Timing

```
TERM: for (int j = 0; j < II; j++) {
    row_counter++;
    if (row_counter > row_length) {
        term[j] = 0;
    } else {
        value = values_fifo.read();
        col   = cols_fifo.read();
        term[j] = value * x[col];
    }
}

DTYPE sum_tmp = 0;
SUM_TMP: for (int j = 0; j < II; j++) {
    sum_tmp += term[j];
}
sum += sum_tmp;
```

II = 6、COM Latency = 42

Modules & Loops	Latency(cycles)	Iteration Latency	Interval	Trip Count	Pipelined
spmv	-	-	-	-	no
spmv_Pipeline_1	258	-	258	-	no
spmv_Pipeline_VITIS_LOOP_76_1	259	-	259	-	no
spmv_Pipeline_VITIS_LOOP_83_2	260	-	260	-	no
spmv_kernel	-	-	-	-	dataflow
spmv_kernel_Loop_FIFO_proc2	3279	-	3279	-	no
spmv_kernel_Loop_COM_proc3	-	-	-	-	no
spmv_kernel_Loop_COM_proc3_Pipeline_COM	-	-	-	-	no
COM	-	42	6	-	yes
spmv_kernel_Loop_Y_proc4	258	-	258	-	no

Cosim Latency: 6313

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
spmv				6313	6313	6313
spmv_Pipeline_1				256	256	256
spmv_Pipeline_VITIS_LOOP_76_1				257	257	257
spmv_Pipeline_VITIS_LOOP_83_2				258	258	258
spmv_kernel				5535	5535	5535

Overall Result

Performance NNZ=3277	latency (cycle)	Improved factor
Baseline	19458	1
Partial unroll	13007	1.49
Naive stream	13636	1.42
Fast stream	6313	3.08

Resource(%) NNZ=3277	DSP	FF	LUT	BRAM(個)
Baseline	2	1	1	0
Partial unroll	2	1	3	0
Naive stream	5	1	3	1
Fast stream	5	2	6	3

Summary

We introduce sparse format 、SpMV operation and optimizations on SpMV

Sparse format

- Reduce memory footprint、execution time and scalable representation

Optimizations

1. Use unroll to exploit parallelism
2. Use streaming to overcome the irregularity between rows
3. Combine them all together

The final result is 3x faster than the baseline

Backup slide

pipeline v.s. partial unroll