

Lab #B

Sparse Matrix Vector Multiplication

<https://github.com/kaiiiz/hls-spmv>

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

Sparse Matrix

$$\begin{matrix} & m=5 \\ n=4 & \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix} \\ & \text{nnz}=6 \end{matrix}$$

sparsity: $6/20 = 30\%$

nnz: number of non-zero

Sparse Matrix Vector Multiplication

$$\begin{matrix} n=4 \\ \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \begin{matrix} m=5 \\ \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \end{matrix}$$

nnz=6

nnz: number of non-zero

Naive Vector Multiplication:

- Multiplication: $N * M = 20$
- Addition: $N * (M - 1) = 16$

Sparse Matrix Vector Multiplication:

- Multiplication: 6
- Addition: 3

Sparse Matrix Encoding - COO (Coordinate list)

$$\begin{matrix} & m=5 \\ n=4 & \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix} \\ & \text{nnz}=6 \end{matrix}$$

(a) Original

$$A = \{ (0, 0, 2), (0, 3, 1), (2, 3, 1), \\ (2, 4, 4), (3, 1, 2), (3, 3, 1) \}$$

$$\begin{array}{ll} \text{row_index} & [0 \quad 0 \quad 2 \quad 2 \quad 3 \quad 3] \\ \text{col_index} & [0 \quad 3 \quad 3 \quad 4 \quad 1 \quad 3] \\ \text{value} & [2 \quad 1 \quad 1 \quad 4 \quad 2 \quad 1] \end{array}$$

(b) COO

Sparse Matrix Encoding - CSR (Compressed Sparse Row)

$$\begin{matrix} & m=5 \\ n=4 & \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix} \\ & \text{nnz}=6 \end{matrix}$$

(a) Original

$$A = \{ (0, 0, 2), (0, 3, 1), (2, 3, 1), \\ (2, 4, 4), (3, 1, 2), (3, 3, 1) \}$$

$$\begin{array}{l} \text{row_index} \begin{bmatrix} 0 & 0 & 2 & 2 & 3 & 3 \end{bmatrix} \\ \text{col_index} \begin{bmatrix} 0 & 3 & 3 & 4 & 1 & 3 \end{bmatrix} \\ \text{value} \begin{bmatrix} 2 & 1 & 1 & 4 & 2 & 1 \end{bmatrix} \end{array}$$

(b) COO

$$\begin{array}{l} \text{row_length} [2 \quad 0 \quad 2 \quad 2] \\ \text{row_index} \begin{bmatrix} 0 & 2 & 2 & 4 & 6 \end{bmatrix} \\ \text{value} \begin{bmatrix} 2 & 1 & 1 & 4 & 2 & 1 \end{bmatrix} \\ \text{col_index} \begin{bmatrix} 0 & 3 & 3 & 4 & 1 & 3 \end{bmatrix} \end{array}$$

(c) CSR

SpMV - CSR

```
1 #include "spmv.h"
2
3 void spmv(int rowPtr[NUM_ROWS+1],
4           int columnIndex[NNZ],
5           DTYPE values[NNZ],
6           DTYPE y[SIZE],
7           DTYPE x[SIZE])
8 {
9     for (int i = 0; i < NUM_ROWS; i++) {
10         DTYPE y0 = 0;
11         for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
12             y0 += values[k] * x[columnIndex[k]];
13         }
14         y[i] = y0;
15     }
16 }
```

$$\begin{matrix} & m=5 \\ n=4 & \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix} \\ & nnz=6 \end{matrix}$$

row_index	[0 2 2 4 6]					
value	[2 1 1 4 2 1]					
col_index	[0 3 3 4 1 3]					

Test data

- Matrix
 - Type: float
 - Size: 256 x 256
 - Sparsity: 5%
 - NNZ: 3277 (Size * Sparsity)
- Input
 - Type: float
 - Size: 256
- Output
 - Type: float
 - Size: 256

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- **Basic Optimization**
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

Basic Optim. - Pipeline

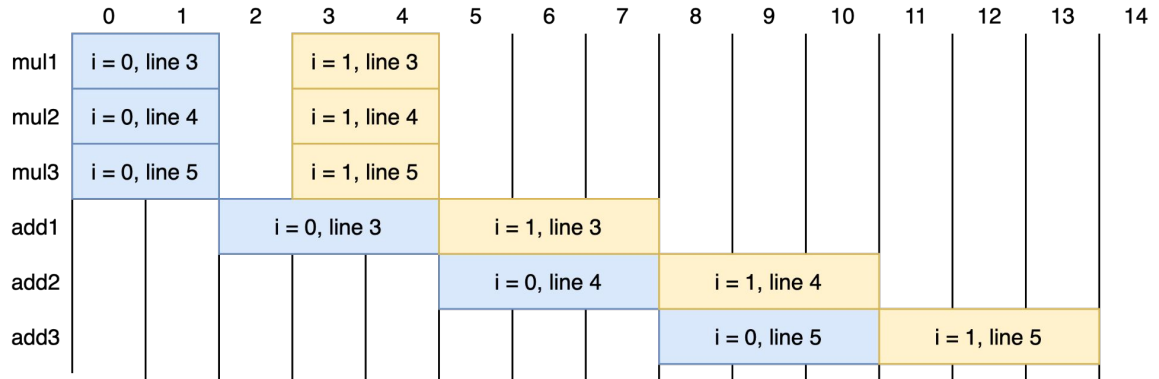
```
1 #include "spmv.h"
2
3 void spmv(int rowPtr[NUM_ROWS+1],
4           int columnIndex[NNZ],
5           DTYPE values[NNZ],
6           DTYPE y[SIZE],
7           DTYPE x[SIZE])
8 {
9     for (int i = 0; i < NUM_ROWS; i++) {
10         DTYPE y0 = 0;
11         for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
12             y0 += values[k] * x[columnIndex[k]];
13         }
14         y[i] = y0;
15     }
16 }
```

Problem: Can not perform pipeline between rows, since the inner loop can not be fully unrolled.

Basic Optim. - Explanation of outer loop pipeline

```
1  for (int i = 0; i < NUM_ROWS; i++) {  
2      DTYPE y0 = 0;  
3      y0 += values[rowPtr[i]] * x[columnIndex[rowPtr[i]]];  
4      y0 += values[rowPtr[i] + 1] * x[columnIndex[rowPtr[i] + 1]];  
5      y0 += values[rowPtr[i] + 2] * x[columnIndex[rowPtr[i] + 2]];  
6      y[i] = y0;  
7  }
```

} If fully unrolled is possible

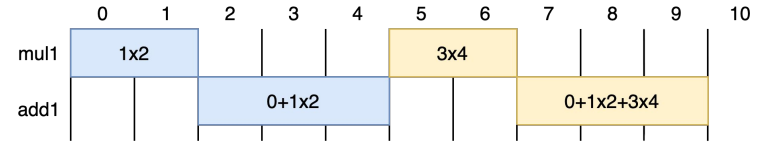


It's easy to determine hardware resources and easy to schedule If fully unrolled is possible.

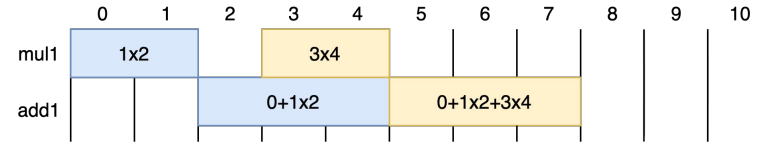
However, the number of iteration is determined in runtime.

Basic Optim. - Inner Loop Pipeline

```
1 #include "spmv.h"
2
3 void spmv(int rowPtr[NUM_ROWS+1],
4           int columnIndex[NNZ],
5           DTYPE values[NNZ],
6           DTYPE y[SIZE],
7           DTYPE x[SIZE])
8 {
9     for (int i = 0; i < NUM_ROWS; i++) {
10         DTYPE y0 = 0;
11         for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
12             #pragma HLS pipeline
13             y0 += values[k] * x[columnIndex[k]];
14         }
15         y[i] = y0;
16     }
17 }
```



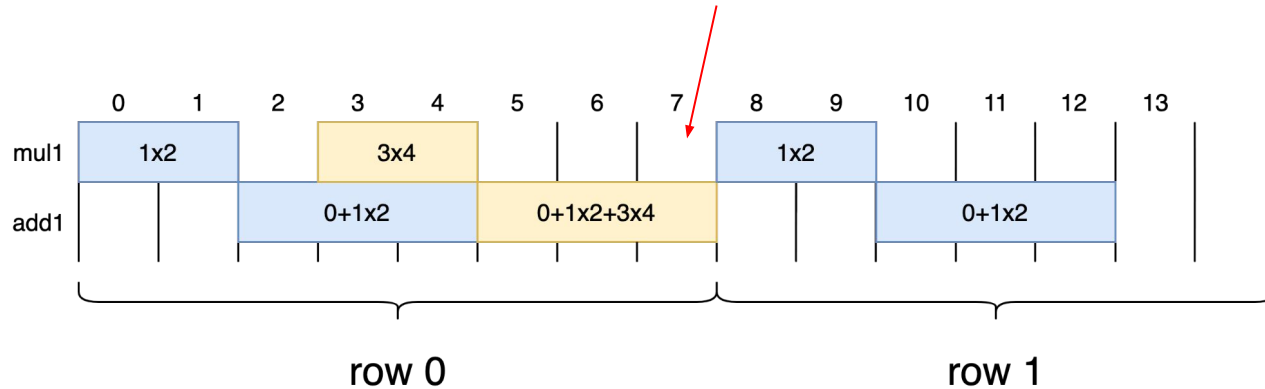
Before inner-loop pipeline



After inner-loop pipeline

Disadvantage of inner loop pipeline

pipeline flush is required between different rows



Basic Optim. - Pipeline - Result - Latency

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.353 ns	0.62 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	none

Detail

+ Instance

- Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	?	?	?	-	-	256	no
+ L2	?	?	18	9	1	?	yes

fadd: 9 cycles
fmul: 5 cycles

Problem: Trip count information is determined in runtime
(need loop_tripcount directive or using co-sim)

Basic Optim. - Pipeline - Result - Latency

Cosimulation Report for 'spmv'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	60011	60011	60011	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

no optimization

Cosimulation Report for 'spmv'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	32822	32822	32822	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

pipeline inner loop

Basic Optim. - Pipeline - Result - Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	85	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	151	-
Register	-	-	337	-	-
Total	0	5	812	985	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

no optimization

Utilization Estimates

Summary

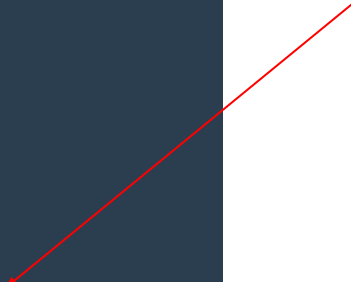
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	87	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	122	-
Register	-	-	333	-	-
Total	0	5	808	958	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

pipeline inner loop

Basic Optim. - Automatic Partial Loop Unrolling

```
1 #include "spmv.h"
2
3 void spmv(int rowPtr[NUM_ROWS+1],
4           int columnIndex[NNZ],
5           DTYPE values[NNZ],
6           DTYPE y[SIZE],
7           DTYPE x[SIZE])
8 {
9     for (int i = 0; i < NUM_ROWS; i++) {
10         DTYPE y0 = 0;
11         for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
12 #pragma HLS pipeline
13 #pragma HLS UNROLL factor=9
14             y0 += values[k] * x[columnIndex[k]];
15         }
16         y[i] = y0;
17     }
18 }
```

Problem: Loop bound is not determined, so the inner loop can only be partial unrolled.



Basic Optim. - Automatic Partial Loop Unrolling - Result

Cosimulation Report for 'spmv'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	32822	32822	32822	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

pipeline

Cosimulation Report for 'spmv'

Result

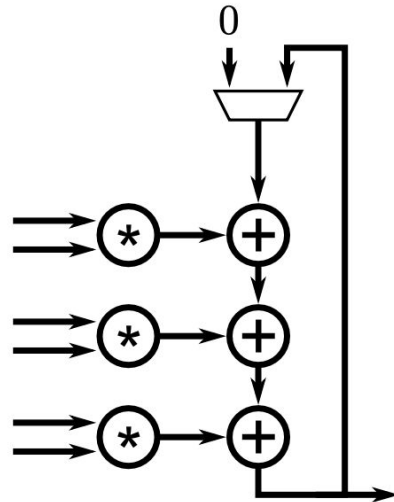
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	42510	42510	42510	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

pipeline + auto partial loop unrolling

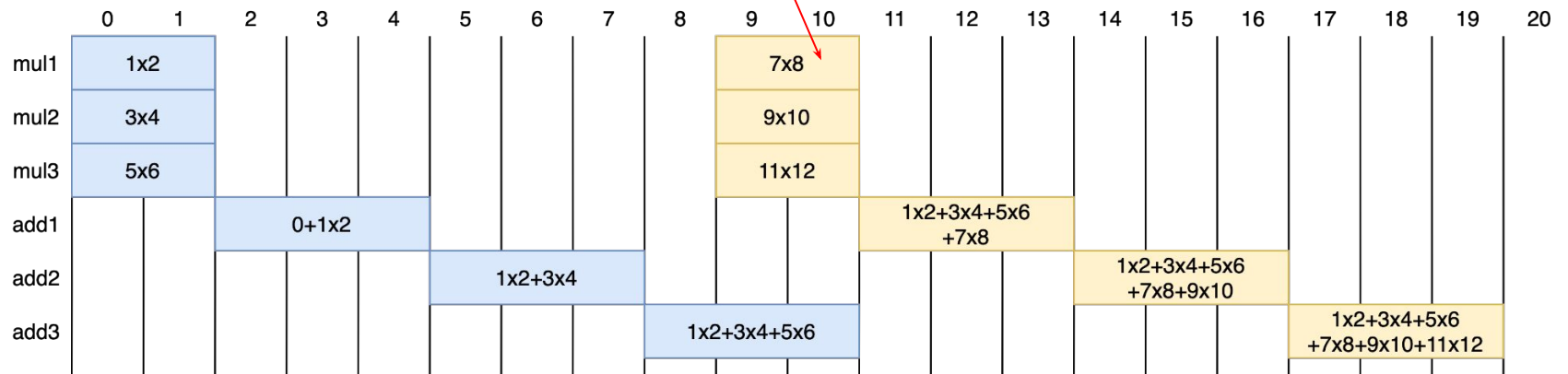
Behavior Automatic Partial Loop Unrolling

To preserve the order of the calculation (for floating point precision), automatic loop partial unrolling will conservative synthesize sequential addition.



need to wait feedback result

unroll factor = 3

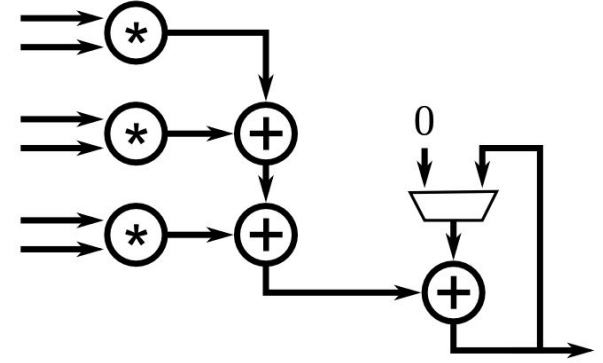


Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

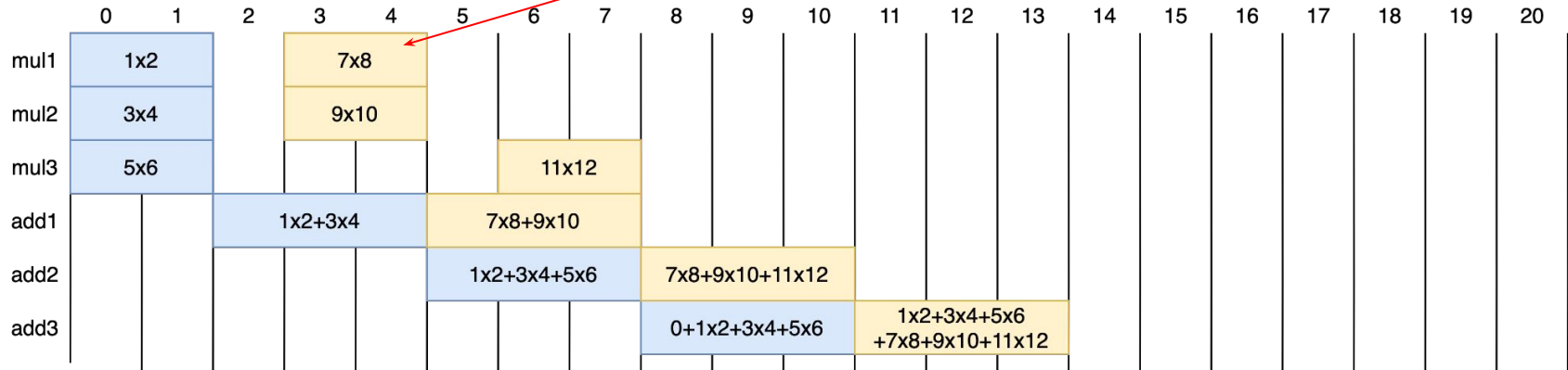
Manual Partial Loop Unrolling

Reduce the length of recurrence chain from 3 to 1.

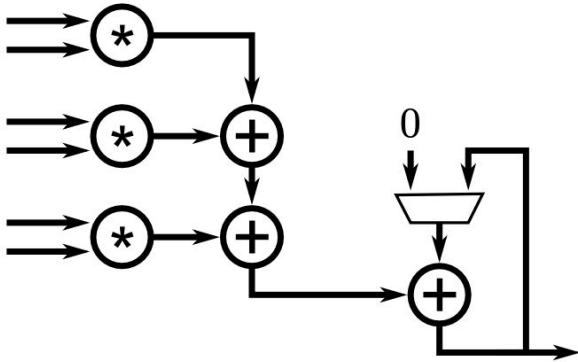


no need to wait feedback result

unroll factor = 3



Manual Partial Loop Unrolling



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

Basic Optim. - Manual Partial Loop Unrolling - Latency

Cosimulation Report for 'spmv'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	32822	32822	32822	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

pipeline

Cosimulation Report for 'spmv'

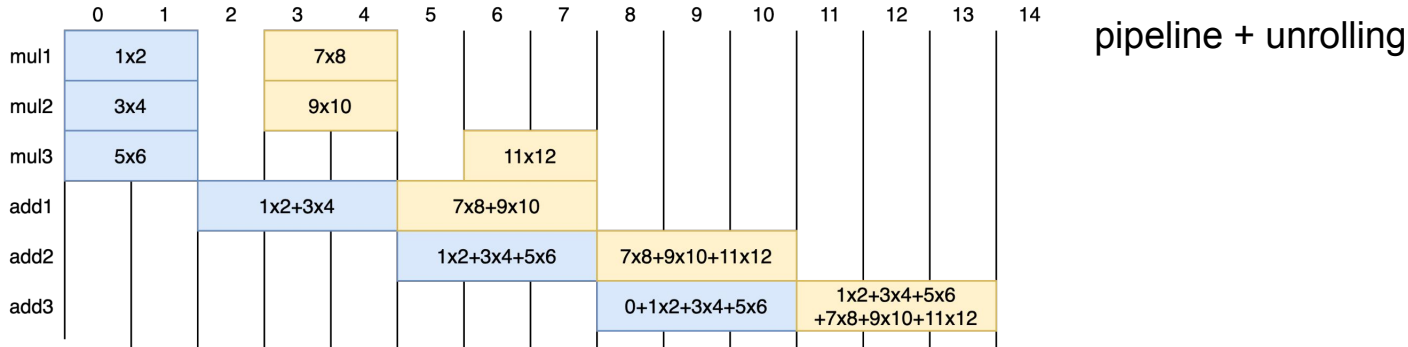
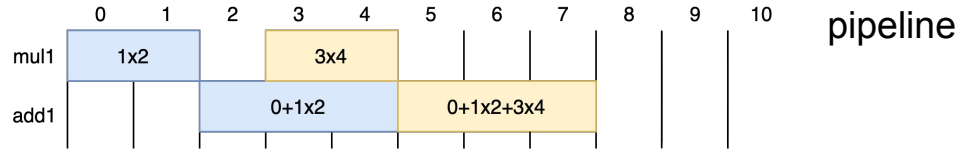
Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	28430	28430	28430	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

pipeline + manual partial loop unrolling

Basic Optim. - Manual Partial Loop Unrolling - Trade-off



- Unrolling: **The depth of the pipeline** of the inner loop is much longer.
 - The number of cycles to flush the pipeline to start a new iterations of the outer L1 loop is much larger.
- Fewer non-zero elements in each line would favor the first implementation, while more non-zero elements in each line would favor the second implementation

Basic Optim. - Manual Partial Loop Unrolling - Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	87	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	122	-
Register	-	-	333	-	-
Total	0	5	808	958	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

pipeline

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	799	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	508	-
Register	0	-	2878	448	-
Total	0	5	3353	2504	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	3	4	0

pipeline + manual partial loop unrolling

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- **Increase parallelism between rows: Naive stream implementation**
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

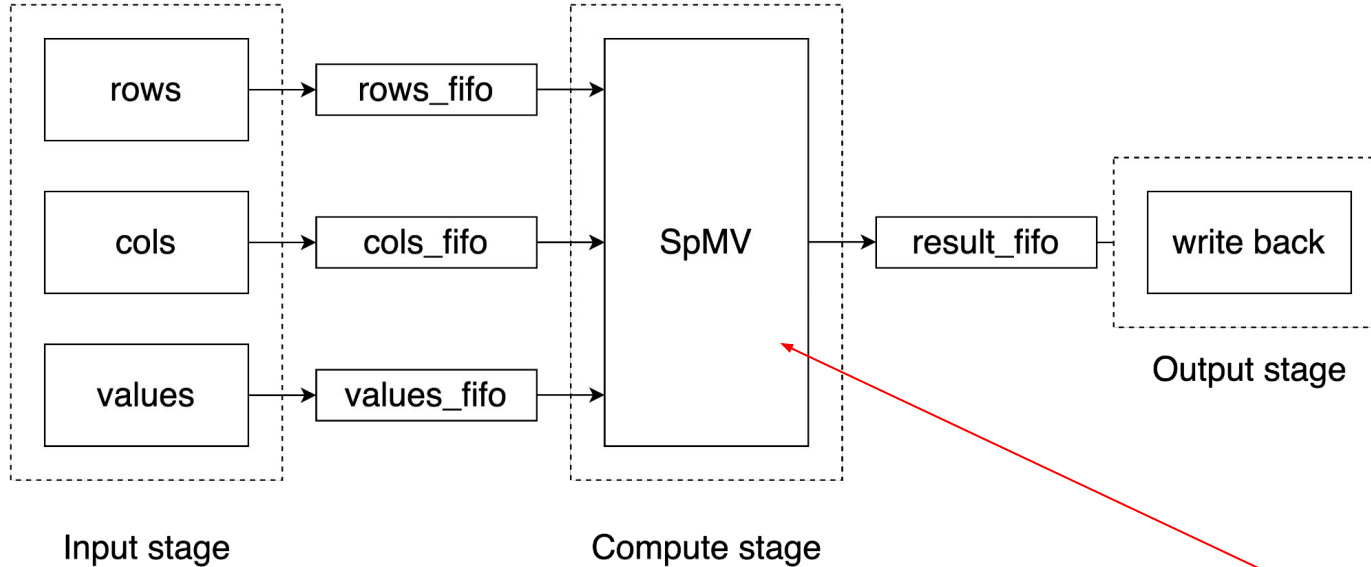
Streaming Dataflow in SpMV

A Streaming Dataflow Engine for Sparse Matrix-Vector Multiplication using High-Level Synthesis

Mohammad Hosseinabady, and Jose Luis Nunez-Yanez

@University of Bristol, 2019

Naive Stream - Concept

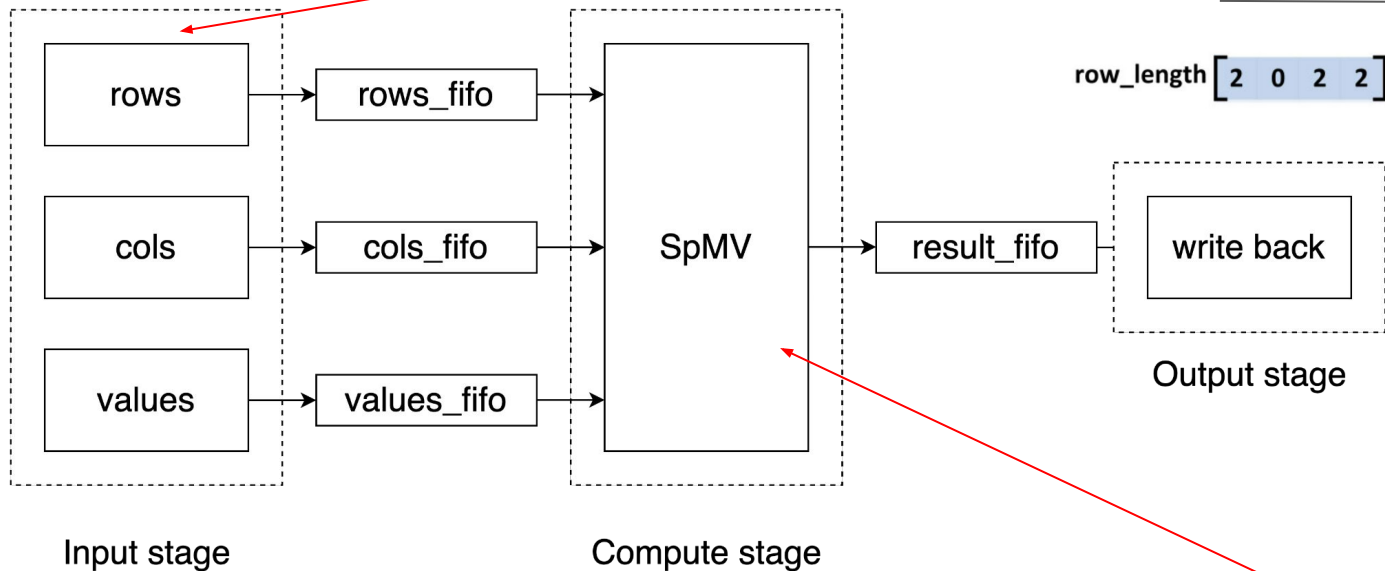


Easy to pipeline, and the values of compute stage aren't restricted by row.

Naive Stream - MCSR

Using MCSR
(store NNZ in each row)

CSR	row_index	[0 2 2 4 6]
	value	[2 1 1 4 2 1]
	col_index	[0 3 3 4 1 3]

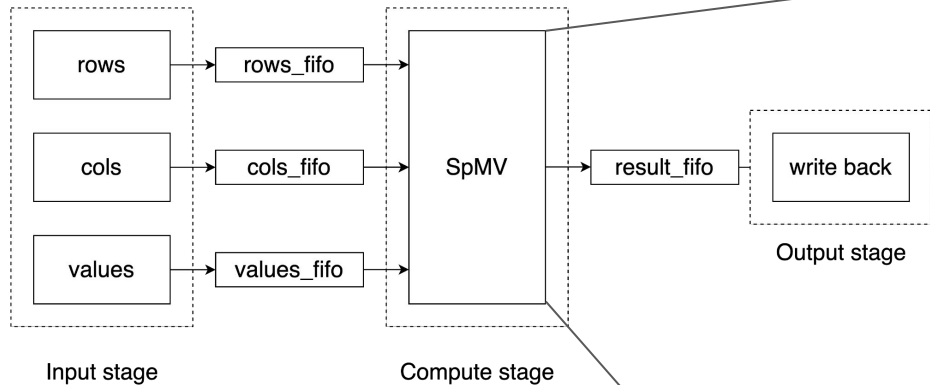


row_length	[2 0 2 2]	value	[2 1 1 4 2 1]
		col_index	[0 3 3 4 1 3]

MCSR

Easy to pipeline, and the values of compute stage aren't restricted by row.

Naive Stream - Implementation



Easy to pipeline, and the values of compute stage aren't restricted by row.

```
for (int i = 0; i < NNZ; i++) {  
    #pragma HLS PIPELINE  
    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;  
    }  
}
```

Naive Stream - Latency

Cosimulation Report for 'spmv'

Result

	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	30026	30026	30026	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

NNZ: 3277

- No optimization: 60011
- Pipeline inner loop: 32822
- Pipeline inner loop + unrolling: 28430

Naive Stream - Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	87	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	122	-
Register	-	-	333	-	-
Total	0	5	808	958	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

pipeline

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	116	-
FIFO	-	-	-	-	-
Instance	0	5	1071	1672	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	129	-
Register	0	-	282	64	-
Total	1	5	1353	1981	0
Available	280	220	106400	53200	0
Utilization (%)	~0	2	1	3	0

Detail

Instance

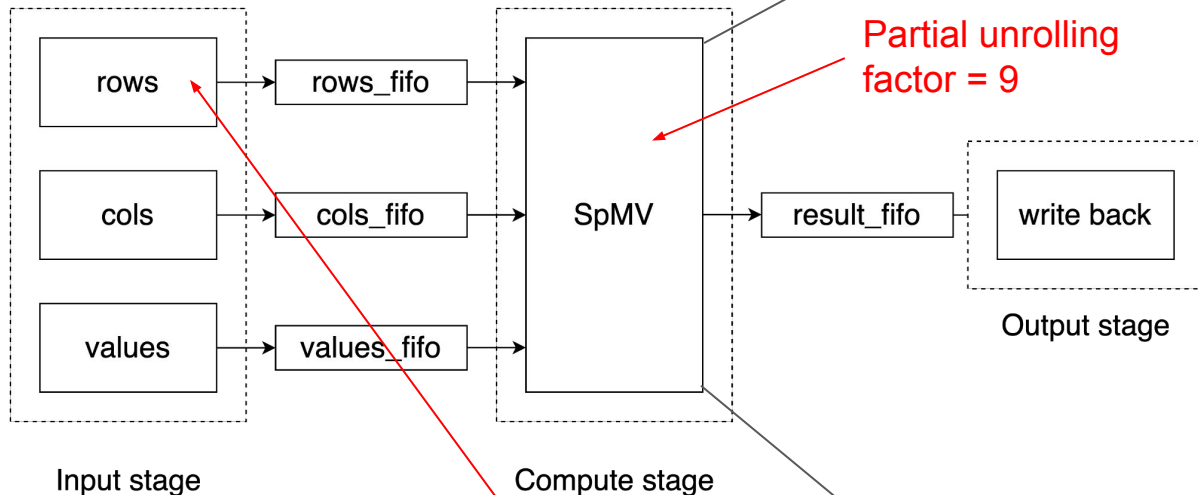
Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
grp_spmv_kernel_fu_130	spmv_kernel	0	5	1071	1672	0
Total	1	0	5	1071	1672	0

naive stream

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- **Put it all together: Fast stream implementation**
- Experience of using Silexica
- Conclusion

Fast Stream



```
for (int i = 0; i < new_nnz; i += II) {  
    #pragma HLS PIPELINE  
    if (row_length_pad == 0) {  
        row_length_pad = rows_length_pad[k];  
        row_length = rows_length[k++];  
        row_counter = 0;  
        sum = 0;  
    }  
  
    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;  
    for (int j = 0; j < II; j++) {  
        sum_tmp += term[j];  
    }  
    sum += sum_tmp;  
  
    row_length_pad -= II;  
    if (row_length_pad == 0) {  
        results_fifo << sum;  
    }  
}
```

Fast Stream SpMV - Latency

Cosimulation Report for 'spmv'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	5816	5816	5816	NA	NA	NA

Export the report(.html) using the [Export Wizard](#)

NNZ: 3277

- No optimization: 60011 (baseline)
- Pipeline inner loop: 32822
- Pipeline inner loop + unrolling: 28430
- Naive stream: 30026

Zero padding has a little overhead, but it's 10x faster than baseline

Utilization

	BRAM_18K	DSP48E	FF	LUT	URAM
No optim.	0	5	812	985	0
Pipeline	0	5	808	958	0
Pipeline + Unrolling	0	5	3353	2504	0
Naive Stream	1	5	1353	1981	0
Fast Stream	2	5	9070	6267	0

Fast Stream SpMV - Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	85	-
FIFO	-	-	-	-	-
Instance	-	5	475	749	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	151	-
Register	-	-	337	-	-
Total	0	5	812	985	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	1	0

no optimization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	284	-
FIFO	-	-	-	-	-
Instance	0	5	7152	5461	-
Memory	2	-	0	0	0
Multiplexer	-	-	-	266	-
Register	0	-	1918	256	-
Total	2	5	9070	6267	0
Available	280	220	106400	53200	0
Utilization (%)	~0	2	8	11	0

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
grp_spmv_kernel_fu_214	spmv_kernel	0	5	4869	3723	0
spmv_srem_32ns_5ng8j_U34	spmv_srem_32ns_5ng8j	0	0	2283	1738	0
Total		2	5	7152	5461	0

fast stream

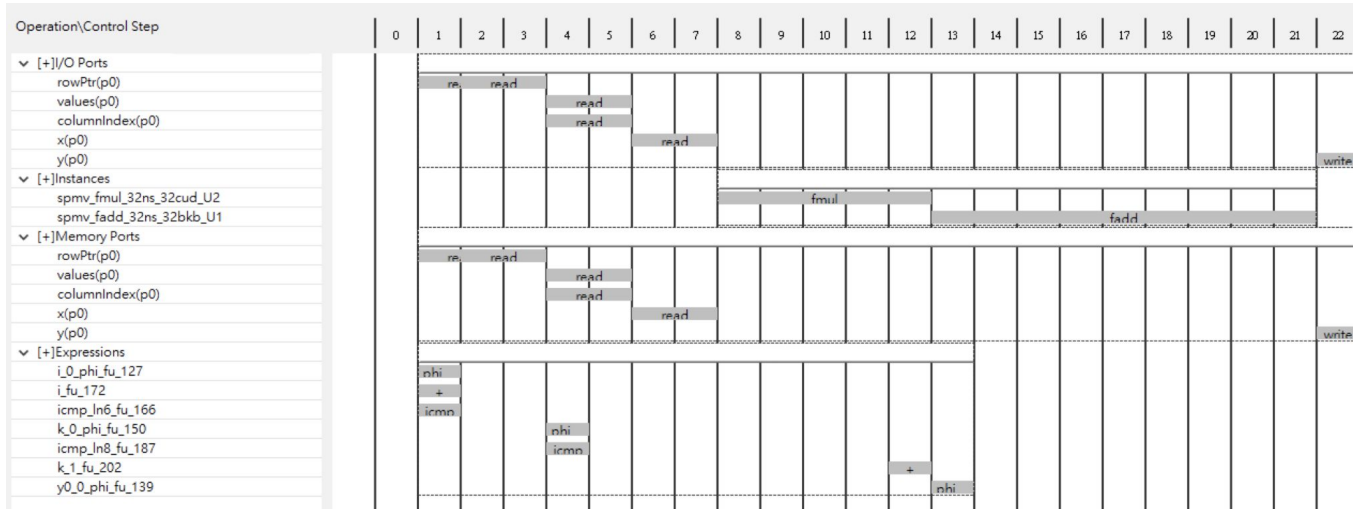
Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

Experience of using Silexica

Some important features in Vivado HLS but can't find in Silexica:

- analysis view (it's useful to analysis initial interval and resource utilization)
- co-simulation



Experience of using Silexica

Anyway... I imported fast stream HLS project to Silexica, and tried to get some optimization suggestions:

- `array_partition` **and** `array_reshape` for both array arguments and variables
- `loop_tripcount`
- `pipeline`

Experience of using Silexica

```
2   for (int i = 0; i < new_nnz; i+=II) {  
3   // Loop annotated with HLS directives by SLX  
4   #pragma HLS loop_tripcount min=485 max=485  
5   }  
6  
7   // Loop annotated with HLS directives by SLX  
8   #pragma HLS pipeline  
9   }  
10  #pragma HLS PIPELINE
```

NNZ = 3277, new_nnz >= 3277



duplication?



Experience of using Silexica

```
void spmv_kernel(..., DTYPE y[SIZE], ...) {  
    y = answer  
}  
  
void spmv(..., DTYPE y[SIZE], ...) {  
    // Function annotated with HLS directives by SLX  
    #pragma HLS array_reshape variable=y complete  
  
    (padding preprocessing)  
    spmv_kernel(..., y, ...)  
}
```

Need inline?

ERROR: [XFORM 203-711] Function 'spmv_kernel' failed dataflow checking: only top module can have a return value from a dataflow, please convert return statement into a pointer argument.

Function 'spmv_kernel' failed dataflow checking: only top module can have a return value from a dataflow, please convert return statement into a pointer argument.

Outline

- Introduction of Sparse Matrix Vector Multiplication (SpMV)
- Basic Optimization
- Increase parallelism in one row: Manual partial loop unrolling
- Increase parallelism between rows: Naive stream implementation
- Put it all together: Fast stream implementation
- Experience of using Silexica
- Conclusion

Conclusion

- Introduction of compressed matrix (COO, CSR, MCSR)
- Some optimization techniques:
 - Pipeline
 - Pipeline + Partial Loop Unroll
 - Can increase parallelism in one row
 - Naive Stream
 - Can increase parallelism across multiple rows
 - Fast Stream
 - Naive stream + partial loop unroll
- Fast stream implementation can perform SpMV with about 10x faster than baseline implementation

Takeaways

Q1: Why the outer loop of the naive CSR multiplication can not be pipelined?

Q2: Why the traditional approach can only apply partial unrolling?