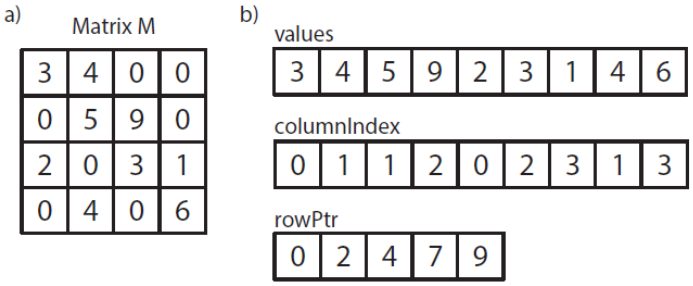
HLS Self-paced Learning Project
[pp4fpga] Sparse Matrix Vector Multiplication

r08943154 謝承翰

1. Sparse Matrix Vector Multiplication

   A normal matrix vector multiplication is a 2D array multiplied by a 1D array. A simple implementation is to enumerate all the elements for multiplication. For a sparse matrix, the multiplication is better in the CRS representation.



   In this figure, the matrix M can be represented as three arrays on the right hand side. The array “values” stores the values which are not zero in M. The array “columnIndex” contains the index of the column corresponding to the values in the array “values”. And the array “rowPtr” indicates the start and end position of a single row’s values.

   For the CRS representation, the matrix vector multiplication can be written as

```
#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;
    }
}
```

   The first for loop enumerate rows and the second for loop use rowPtr to enumerate the values in a column. Finally, the values are multiplied by vector and summed up.

2. HLS

(1) Loop_tripcount pragma

If we synthesis the code directly, the latency information can be reported by vivado HLS tool. However, we cannot get latency report in this case.

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

Since the execution time of the second loop cannot be determined, the HLS tool cannot report the latency. So we need to add a pragma as

```
#pragma HLS loop_tripcount min=1 max=4 avg=2
```

to tell the tool that the time of execution with min, max and average. After inserting this pragma, the baseline code is reported as

- Latency
  - Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
57	189	0.570 us	1.890 us	57	189	none

  - Detail
    - Instance
    - Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	56	188	14 ~ 47	-	-	4	no
+ L2	11	44	11	-	-	1 ~ 4	no

This report can estimate the latency with the given informations.

(2) Optimization Pragmas

In the textbook, a series of pragmas are suggested for us to try.

	L1	L2
Case 1	-	-
Case 2	-	pipeline
Case 3	pipeline	-
Case 4	unroll=2	-
Case 5	-	pipeline, unroll=2
Case 6	-	pipeline, unroll=2, cyclic=2
Case 7	-	pipeline, unroll=4
Case 8	-	pipeline, unroll=4, cyclic=4
Case 9	-	pipeline, unroll=8
Case 10	-	pipeline, unroll=8, cyclic=8
Case 11	-	pipeline, unroll=8, block=8

Since we suppose the maximum iteration of loop 2 is 4, we compare the unrolling until 4.

Utilization Estimates

	origin_loop_tripcount	L2_pipeline	L2_pipeline_unroll2	L2_pipeline_unroll4
BRAM_18K	0	0	0	0
DSP48E	5	5	5	5
FF	563	563	729	999
LUT	878	883	1026	1345
URAM	0	0	0	0

The resource of pipeline and loop unrolling is larger than the original version because the control logic and the duplicate of the operations.

Latency

		origin_loop_tripcount	L2_pipeline	L2_pipeline_unroll2	L2_pipeline_unroll4
Latency (cycles)	min	57	57	101	181
	max	189	117	141	181
Latency (absolute)	min	0.570 us	0.570 us	1.010 us	1.810 us
	max	1.890 us	1.170 us	1.410 us	1.810 us
Interval (cycles)	min	57	57	101	181
	max	189	117	141	181

Pipeline helps improve the latency of the original design, but unrolling does not. First, we may think this can be solved by array partition.

Latency

		L2_pipeline_unroll2	L2_pipeline_unroll2_cyclic2	L2_pipeline_unroll4	L2_pipeline_unroll4_cyclic4
Latency (cycles)	min	101	101	181	185
	max	141	141	181	185
Latency (absolute)	min	1.010 us	1.010 us	1.810 us	1.850 us
	max	1.410 us	1.410 us	1.810 us	1.850 us
Interval (cycles)	min	101	101	181	185
	max	141	141	181	185

Utilization Estimates

	L2_pipeline_unroll2	L2_pipeline_unroll2_cyclic2	L2_pipeline_unroll4	L2_pipeline_unroll4_cyclic4
BRAM_18K	0	0	0	0
DSP48E	5	5	5	5
FF	729	729	999	1166
LUT	1026	1218	1345	2426
URAM	0	0	0	0

However, the report shows that unrolling only increases the resources but has no latency improvement. From the synthesis report, the reason is that the II is 5 in the implementation.

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	56	116	14 ~ 29	-	-	4	no
+ L2	10	25	11	5	1	1 ~ 4	yes

This is caused by the multiplication operation and the

(3) Data Type

In the testcase, all the values are integer, but the type to store the values are floating points. Once we change the type to int, the II can be 1 easily.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
33	45	0.330 us	0.450 us	33	45	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	32	44	8 ~ 11	-	-	4	no
+ L2	4	7	5	1	1	1 ~ 4	yes

(4) Code rewriting for loop unrolling

For the unroll version, we can rewrite the code. We separate the multiplication and the addition into two different loops. The pipeline can be more efficient.

```
void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
        DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
#pragma HLS ARRAY_PARTITION variable=rowPtr cyclic factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=columnIndex cyclic factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=values cyclic factor=4 dim=1

L1: for (int i = 0; i < NUM_ROWS; i++) {
    DTYPE yt[SIZE];
#pragma HLS ARRAY_PARTITION variable=yt complete dim=1
    DTYPE y0 = 0;
    L2_1: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {
#pragma HLS loop_tripcount min=1 max=4 avg=2
#pragma HLS unroll factor=4
#pragma HLS pipeline
            yt[k] = values[k] * x[columnIndex[k]];
        }
    L2_2: for (int k = 0; k < SIZE; k++) {
#pragma HLS unroll
            y0 += yt[k];
        }
        y[i] = y0;
    }
}
```

The synthesis report shows that II=2 in this code.

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
137	137	1.370 us	1.370 us	137	137	none

⊟ Detail

⊞ Instance

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	136	136	34	-	-	4	no
+ L2_1	9	9	8	2	1	1	yes

3. Conclusion
   (1) loop_tripcount can help us determine the iteration of a loop
   (2) We can first use optimization pragmas to do some optimization
   (3) When the II cannot meet the goal, can analyze the II and rewrite the code to separate some operation for a better pipeline.
   (4) Codes and report are available on my github https://github.com/qmo1222/MSOC_HLS
4. Reference
   (1) Textbook and codes in https://github.com/KastnerRG/pp4fpgas