MSOC Fall 2020 (self-paced **3**)
R08943099 吳亭儀

Matrix Multiplication

1. HLS C-sim/Synthesis/Cosim (Screenshot + brief intro) (1%)

矩陣乘法是一種將兩個矩陣合併為第三個矩陣的二進制運算。運算本身可以描述為對兩個矩陣的向量作線性運算。矩陣乘法的最常見形式稱為矩陣乘積。當矩陣 A 的尺寸為 n x m 而矩陣 B 的尺寸為 m x p 時，矩陣乘積 AB 創建一個 n x p 矩陣。

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} (\mathbf{AB})_{11} & (\mathbf{AB})_{12} & \cdots & (\mathbf{AB})_{1p} \\ (\mathbf{AB})_{21} & (\mathbf{AB})_{22} & \cdots & (\mathbf{AB})_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ (\mathbf{AB})_{n1} & (\mathbf{AB})_{n2} & \cdots & (\mathbf{AB})_{np} \end{bmatrix}$$

where the operation $(\mathbf{AB})_{ij}$ is defined as $(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$.

◎圖 1 Matrix Multiplication Operation

```
#include "matrixmultiplication.h"

void matrixmul(int A[N][M], int B[M][P], int AB[N][P]) {
  #pragma HLS ARRAY_RESHAPE variable=A complete dim=2
  #pragma HLS ARRAY_RESHAPE variable=B complete dim=1
  /* for each row and column of AB */
  row: for(int i = 0; i < N; ++i) {
    col: for(int j = 0; j < P; ++j) {
      #pragma HLS PIPELINE II=1
      /* compute (AB)i,j */
      int ABij = 0;
    product: for(int k = 0; k < M; ++k) {
        ABij += A[i][k] * B[k][j];
      }
      AB[i][j] = ABij;
    }
  }
}
```

◎圖 2 Baseline Code

The Screenshot is for the Baseline Code

```
Comparing against output data
*******************************************
PASS: The output matches the golden output!
*******************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

◎圖 3 C-sim

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.742 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1030	1030	10.300 us	10.300 us	1030	1030	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	96	0	1804	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	75	-
Register	0	-	3757	96	-
Total	0	96	3757	1975	0
Available	280	220	106400	53200	0
Utilization (%)	0	43	3	3	0

◎圖 4 C-Synthesis(需要非常多的 DSP48E → Bad)

Cosimulation Report for 'matrixmul'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	1030	1030	1030	NA	NA	NA

Export the report(.html) using the Export Wizard

◎圖 5 Cosim

2. Improvement – throughput, area (1%)

因為是矩陣乘法必然會有很多的乘法跟並行的運算，所以可以先從減少乘法(DSP48 →面積)，以及利用 Pipeline 的運算來減少 Latency。

Solution1: Change the location of the pipeline directive

1a. Insert the pragma in row loop

```
row: for(int i = 0; i < N; ++i) {
  #pragma HLS PIPELINE II=1
```

1b. Insert the pragma in inter loop

```
    product: for(int k = 0; k < M; ++k) {
#pragma HLS PIPELINE II=1
```

Performance Estimates

Timing

Clock		Baseline	solution1a	solution1b
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.742 ns	8.742 ns	8.510 ns

Latency

		Baseline	solution1a	solution1b
Latency (cycles)	min	1030	518	32775
	max	1030	518	32775
Latency (absolute)	min	10.300 us	5.180 us	0.328 ms
	max	10.300 us	5.180 us	0.328 ms
Interval (cycles)	min	1030	518	32775
	max	1030	518	32775

Utilization Estimates

	Baseline	solution1a	solution1b
BRAM_18K	0	0	0
DSP48E	96	468	3
FF	3757	13742	4656
LUT	1975	13167	13522
URAM	0	0	0

◎圖 6 Compare Report (Result shows the location of pipeline directive in Baseline is best).
In solution1a the use of DSP48E is more than can use.

Solution2: Change the array reshape directives

2a. Remove all array reshape directives

Unable to schedule 'load' operation on array 'A' due to limited memory ports. Please consider using a memory core with more ports or partitioning the array 'A'.

Unable to schedule 'load' operation on array 'B' due to limited memory ports. Please consider using a memory core with more ports or partitioning the array 'B'.

```
12      product: for(int k = 0; k < M; ++k) {
13        ABij += A[i][k] * B[k][j];
14      }
15      AB[i][j] = ABij;
```

◎圖 7 因為 line14，所以必須將 Array Reshape

2b. Edit the dim

```
#pragma HLS ARRAY_RESHAPE variable=A complete dim=2
#pragma HLS ARRAY_RESHAPE variable=B complete dim=0
```

2c. Edit the dim

```
#pragma HLS ARRAY_RESHAPE variable=A complete dim=2
#pragma HLS ARRAY_PARTITION variable=B complete dim=0
```

Performance Estimates

Timing

Clock		Baseline	solution2b	solution2c
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.742 ns	8.742 ns	8.742 ns

Latency

		Baseline	solution2b	solution2c
Latency (cycles)	min	1030	1031	1030
	max	1030	1031	1030
Latency (absolute)	min	10.300 us	10.310 us	10.300 us
	max	10.300 us	10.310 us	10.300 us
Interval (cycles)	min	1030	1031	1030
	max	1030	1031	1030

Utilization Estimates

	Baseline	solution2b	solution2c
BRAM_18K	0	0	0
DSP48E	96	96	96
FF	3757	69314	4781
LUT	1975	14654	6615
URAM	0	0	0

◎圖 8 Compare Report(The improvement is fail)

<u>Solution3: Block Matrix Multiplication</u>

Separate origin matrix into submatrix:

$$\mathrm{A}=\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$A_{11}=\begin{bmatrix} a & b \\ e & f \end{bmatrix}, A_{12}=\begin{bmatrix} c & d \\ g & h \end{bmatrix}, A_{21}=\begin{bmatrix} i & j \\ m & n \end{bmatrix}, A_{22}=\begin{bmatrix} k & l \\ o & p \end{bmatrix}$$

◎圖 9 Block based Matrix

```
#include "block_mm.h"
void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
        blockmat &ABpartial, int it) {
#pragma HLS DATAFLOW

  int counter = it % (SIZE/BLOCK_SIZE);
  static DTYPE A[BLOCK_SIZE][SIZE];
  if(counter == 0){ //only load the A rows when necessary
    loadA: for(int i = 0; i < SIZE; i++) {
      blockvec tempA = Arows.read();
      for(int j = 0; j < BLOCK_SIZE; j++) {
        #pragma HLS PIPELINE II=1
        A[j][i] = tempA.a[j];
      }
    }
  }
  DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
  partialsum: for(int k=0; k < SIZE; k++) {
    blockvec tempB = Bcols.read();
    for(int i = 0; i < BLOCK_SIZE; i++) {
      for(int j = 0; j < BLOCK_SIZE; j++) {
        AB[i][j] = AB[i][j] +  A[i][k] * tempB.a[j];
      }
    }
  }
  writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
    for(int j = 0; j < BLOCK_SIZE; j++) {
      ABpartial.out[i][j] = AB[i][j];
    }
  }
}
```

◎圖 10 block_mm.cpp

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.742 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1030	1030	10.300 us	10.300 us	1030	1030	none

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	96	0	1804	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	75	-
Register	0	-	3757	96	-
Total	0	96	3757	1975	0
Available	280	220	106400	53200	0
Utilization (%)	0	43	3	3	0

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.510 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
7052	7404	70.520 us	74.040 us	7050	7050	dataflow

Detail

- Instance
- Loop

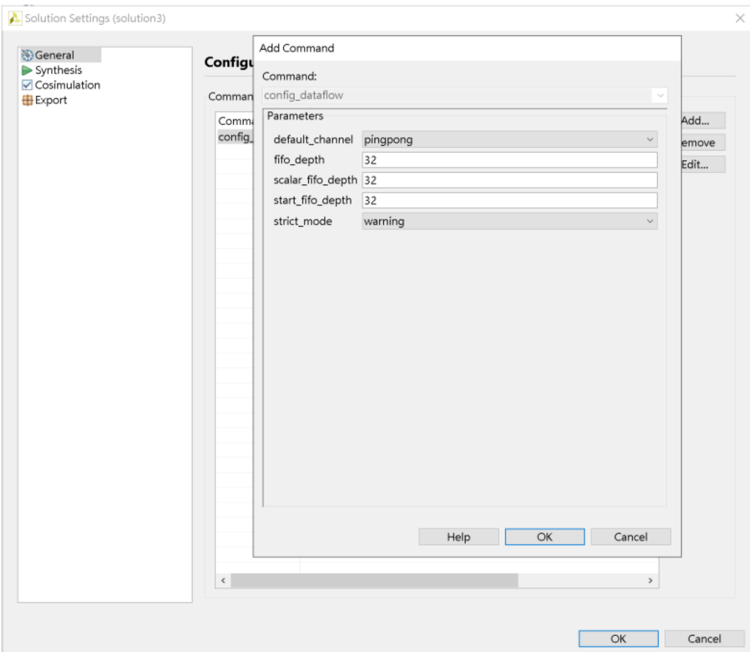
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	46	-
FIFO	2	-	50	44	-
Instance	-	3	766	863	-
Memory	2	-	0	0	0
Multiplexer	-	-	-	54	-
Register	-	-	9	-	-
Total	4	3	825	1007	0
Available	280	220	106400	53200	0
Utilization (%)	1	1	~0	1	0

◎圖 11 Compare Report(Now just need 1DSP48E)

在原本的 code 會無法進行 co-sim，將 DataFlow 即可執行:



◎圖 12 DataFlow 問題修正

3. Github submit (1%)

https://github.com/r08943099/MSOCFall2020

4. Complexity (1% - Instructor/ TA decide)