AAHLS-LabB Report

F08943009 張承洋

Briefly introduction to the algorithm or overall system

這次實驗的目的是實現 Matrix Multiplication (both matrix sizes = 8x8)，baseline 方法是利用三個 for loop 實現，順序為 row, col, and product Loop，並且在每一次 product Loop 前須將一個變數 ABij 初始化為 0，經過 product Loop 的 M (= matrix size) 個 iteration 後再 store 回 AB array。

```
#include "Brutal_design.h"



void matrixmul(int A[N][M], int B[M][P], int AB[N][P]) {
    row: for(int i = 0; i < N; ++i) {
        col: for(int j = 0; j < P; ++j) {
            int ABij = 0;
            product: for(int k = 0; k < M; ++k) {
                #pragma HLS PIPELINE
                ABij += A[i][k] * B[k][j];
            }
            AB[i][j] = ABij;
        }
    }
}
```

Explain what you observed and learned

1. 從 **Lab 1 @ Case: Pipelining Col Loop and reshaping input arrays (solution 4)**觀察，ARRAY_RESHAPE 提升 matrix A,B 提供的 memory bandwidth，但 **HLS 自動捨棄 BRAM 本身提供的 dual-port 功能**，只使用一個 port 並將 bit-width 提升 32x (from 32-bit to 1024-bit)，而非利用 dual-port 的特性將 bit-width 提升 16x。
2. 從 **Lab 1 @ Case: Pipelining Row Loop and reshaping input arrays (solution 5)** 觀察，**對 Col Loop 做 Unrolling 只提升對 B matrix 輸入的 bandwidth 要求**，而不影響對 A matrix 的 bandwidth 要求，因此此作法的瓶頸為 matrix B 的 bandwidth。
3. 從 **Lab 2** 觀察到**外積運算(outer product)能用 constant-bandwidth 讓 IP 維持運作，適合與 streaming dataflow 搭配使用**，能在硬體資源需求差不多的條件下，比起 Lab 1 單純使用 pipeline 的做法大幅降低 latency (700x)。
4. 從 **Lab 2** 當中的各個 solution 觀察到，streaming dataflow 與不同的 unrolling factor 搭配，能在運算延遲以及硬體資源之間達到 trade-off。
5. 從 **Lab 2 @ Analyzing BLOCK_SIZE and MATRIX_SIZE** 觀察，Streaming dataflow 能達到 scalable 的特性，然而須針對 BLOCK_SIZE and unrolling factor 進行嚴謹的 co-exploration 才能在確保 IP 維持良好的 scalability。

Lab 1 (Direct Implementation)

@ Case: No Directive (solution 1)

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
133185	133185	1.332 ms	1.332 ms	133185	133185	none

Detail

Instance

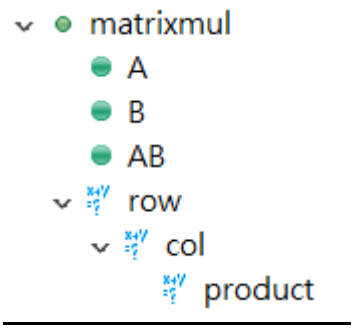
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row	133184	133184	4162	-	-	32	no
+ col	4160	4160	130	-	-	32	no
++ product	128	128	4	-	-	32	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	173	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	77	-
Register	-	-	193	-	-
Total	0	3	193	250	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0



Latency 計算方式: 最內層 Product Loop 的 iteration latency=4 並且 Trip Count =32，因此可計算出 Col Loop 當中的每一次 iteration 需要 4*32=128 cycles，並加上 loop in/loop out 的判斷，可計算出 Row Loop 當中的每一次 iteration 需要 (128+2)*32=4160 cycles，同樣道理，加上 loop in/loop out 的判斷，可計算出 Top function 需要(4160+2)*3=133184 cycles。

@ Case: Pipelining Product Loop (solution 2)

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
32773	32773	0.328 ms	0.328 ms	32773	32773	none

Detail

Instance

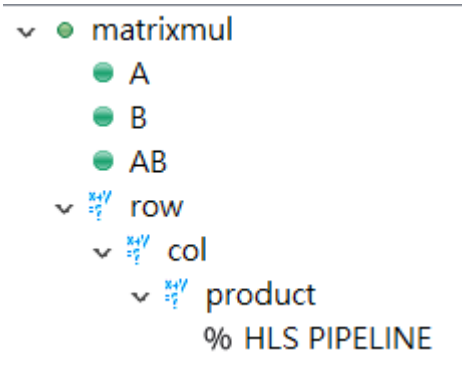
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row_col_product	32771	32771	5	1	1	32768	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	306	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	120	-
Register	0	-	475	128	-
Total	0	3	475	554	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	1	0



Pipeline 最內層 Loop 後，HLS 會自動對其外圈 Loop 做 Flattening (導致 row-col-product Loop)，Latency 計算方式與前 case 不同，最內層 row-col-product Loop 的 iteration latency=5，但 **II=1**(Trip Count =32768)，因此**前 5 個 cycle 之後每 cycle 都可以輸出運算結果**，可計算出 Top function 需要 5+32768=32773 cycles。雖然 latency 下降，但使用 pipeline 提升了對 FF (**2.46x**) and LUT (**2.21x**)的需求 (pipeline register)。

@ Case: Pipelining Col Loop (solution 3)

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
16389	16389	0.164 ms	0.164 ms	16389	16389	none

Detail

Instance

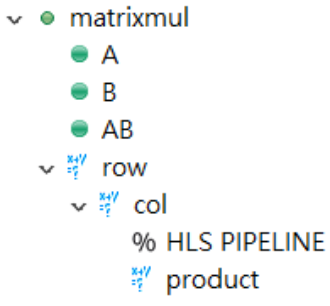
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row_col	16387	16387	20	16	1	1024	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	6	0	1102	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	496	-
Register	-	-	680	-	-
Total	0	6	680	1598	0
Available	280	220	106400	53200	0
Utilization (%)	0	2	~0	3	0



Pipeline Col Loop 後，HLS 會自動對其外圈 Loop 做 Flattening **並對內圈 Product Loop 做 Unrolling**，最內層 row-col Loop 的 iteration latency=20，但 **II=16**(Trip Count =1024)，因此**前 20 個 cycle 之後每 16 cycles 輸出運算結果**，可計算出 Top function 需要 20+16*1024=16404 cycles。雖然 latency 下降，但 Loop Unrolling 大幅提升了對硬體資源的需求(**DSP 2x**, **FF 3.52x**, **LUT 6.39x**)，除了 pipeline register 外，還需要 Loop Unrolling 平行化的運算力(DSP)需求。

無法做到 II=1 的原因是 matrix A,B 所提供的 memory bandwidth 不足，由於 HLS 對 Product Loop 做 Unrolling，可以理解成平行度提升為 32 倍(= matrix size)，然而 dual-port BRAM 能提供的 bandwidth 僅有 2，解釋了為何 II 只有 2 倍的下降量(32->16)，以及 DSP 的需求為何僅提升 2 倍。

@ Case: Pipelining Col Loop and reshaping input arrays (solution 4)

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

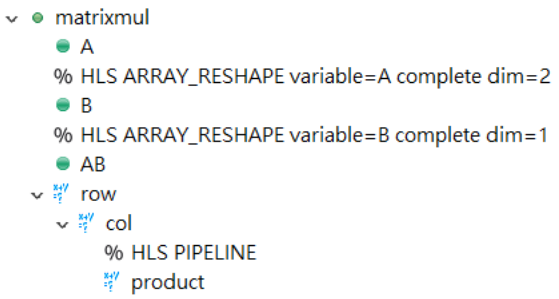
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row_col	1028	1028	6	1	1	1024	yes

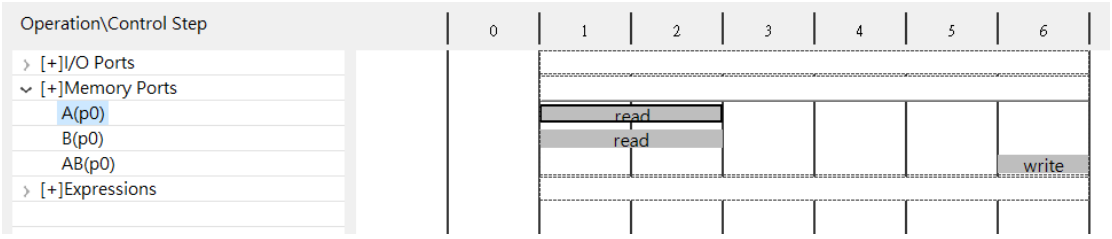
Utilization Estimates

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



根據**@ Case: Pipelining Col Loop (solution 3)** 的結果，**再透過 ARRAY_RESHAPE 提升 matrix A,B 所提供的 memory bandwidth**，從 II 的下降量($\frac{1}{16}$x)以及 DSP 的使用量(16x)看出，目前 input arrays 提供資料的速度已經可以跟上 Product Loop 平行化的程度。



Properties | Warnings | DRCs

Property	Value
Bit Width	1024
Cluster Group Number	-1
Delay	3.25
Line Number	17
Name	A_load
Opcode	load
Op Latency	1
Source File	Brutal_design.cpp
Topo Index	15

根據上圖也可以發現，matrix A,B 的 input port bit-width 大幅提升，此為 ARRAY_RESHAPE (Complete) 的效果。這裡還觀察到一個問題: **HLS 自動捨棄了 BRAM 本身提供的 dual-port 功能**，只使用一個 port 並將 bit-width 提升 32x (from 32-bit to 1024-bit)，而非利用 dual-port 的特性將 bit-width 提升 16x。

@ Case: Pipelining Row Loop and reshaping input arrays (solution 5)

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
518	518	5.180 us	5.180 us	518	518	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row	516	516	21	16	1	32	yes

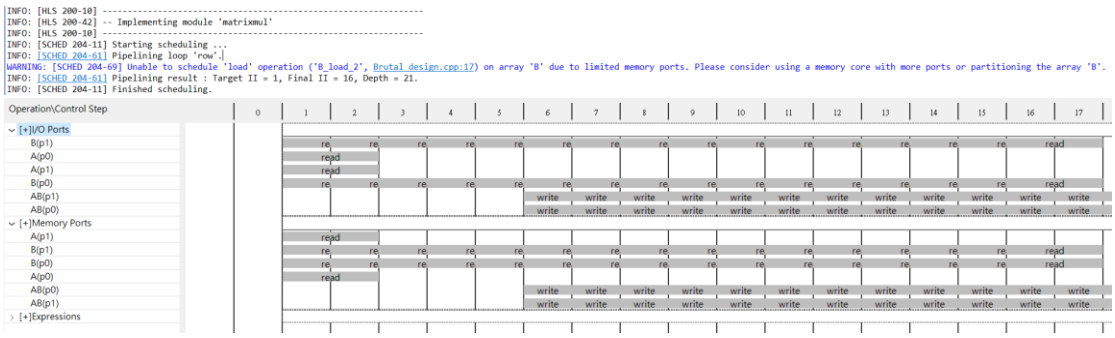
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	468	0	12537	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	630	-
Register	-	-	13742	-	-
Total	0	468	13742	13167	0
Available	280	220	106400	53200	0
Utilization (%)	0	212	12	24	0

- matrixmul
  - A
  - % HLS ARRAY_RESHAPE variable=A complete dim=2
  - B
  - % HLS ARRAY_RESHAPE variable=B complete dim=1
  - AB
  - row
    - % HLS PIPELINE
    - col
      - product

Pipeline Row Loop 後，HLS 會自動對 Product and Col Loop 做 Unrolling，然而 **row Loop 的 II 與前一個 case 的 II 相比大幅提升 (16x)**，即使 Trip Count 下降了 32 倍，但整體 Top function 的 latency 僅有 2 倍下降量，還需要考慮 Loop Unrolling 所帶來的硬體資源需求(**DSP 4.87x**, **FF 3.58x**, **LUT 6.66x**)，因此這個 case 並不是一個好的 design option。



此設計的瓶頸與前一個 case 類似，**皆是因為 matrix 所提供的 memory bandwidth 不足**，然而與前一個 case 不同的點是，**對 Col Loop 做 Unrolling 只會提升對 B matrix 輸入資料的 bandwidth 要求**，而不影響對 A matrix 輸入資料的 bandwidth 要求。因此從上圖可以看出此作法的瓶頸為 matrix B，而 matrix B 提供的 bandwidth 只比前一個 case 提升了 2 倍(利用 BRAM dual-port 的特點)，解釋了為何 II 只有 2 倍的下降量 (32->16)。

Lab 2 (Block-based Implementation)

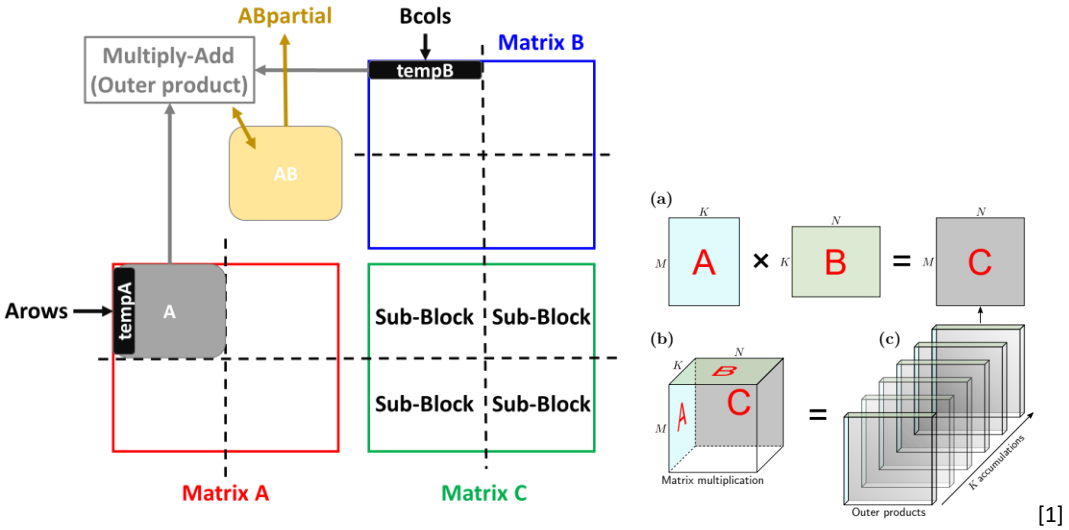
```
#include "BlockMatrix_design.h"


void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,blockmat &ABpartial, int it) {

    DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };

    #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];
            }
        }
    }

    partialsum: for(int k=0; k < SIZE; k++) {
        blockvec tempB = Bcols.read();
        ps_i:for(int i = 0; i < BLOCK_SIZE; i++) {
            ps_j:for(int j = 0; j < BLOCK_SIZE; j++) {
                #pragma HLS PIPELINE II=1

                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];
        }
    }
}
```

Partialsum 暫存運算結果，運算完畢後由 ABpartial進行 streaming output

A 為 Matrix A 的 cache，暫存 streaming in 的輸入資料 (Arows)

將A與streaming in 的輸入資料 (Bcols)做內積運算，將暫存運算結果維護在 AB

AB中的資料由 Abpartial 進行 streaming output

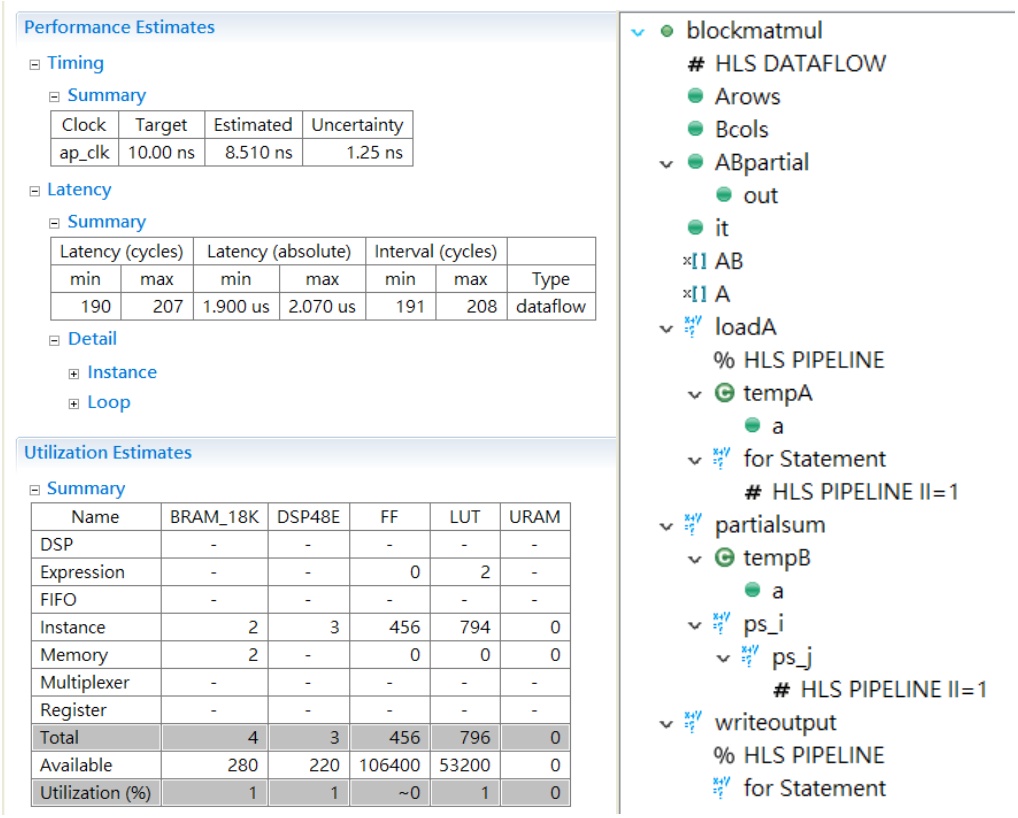


[1]

上圖為整個 IP 的資料流。首先利用 cache A 接收由 Arows 輸入的 streaming data，接著在每個 iteration 當中取出 cache A 當中的一個 column，並從 Bcols 接收 streaming data (tempB)，**將兩者進行外積運算(outer product)**，外積的好處是能夠基於 constant-bandwidth 讓 IP 維持運作 [1]，適合這次 lab 所使用的 streaming dataflow。外積運算過程中的 partial result 會暫存在 cache AB 中，運算結束後再將 cache AB 當中的資料由 ABpartial 進行 streaming output。

[1] Kung, H. T., Vikas Natesh, and Andrew Sabot. "Cake: matrix multiplication using constant-bandwidth blocks." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021.

@ Case: Stream Dataflow (solution 1)



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
190	207	1.900 us	2.070 us	191	208	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	2	3	456	794	0
Memory	2	-	0	0	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	4	3	456	796	0
Available	280	220	106400	53200	0
Utilization (%)	1	1	~0	1	0

由上圖當中的實驗結果可以看出，stream dataflow 能大幅降低硬體資源的需求，DSP 的使用量與 **Lab 1 @ Case: No Directive (solution 1)**相同，顯示 **streaming dataflow 在運算能力上並沒有特別要求**，但為了實現 cache 暫存資料，在 FF and LUT 的需求方面有比較大幅度的提升。

Cosimulation Report for 'blockmatmul'

Result

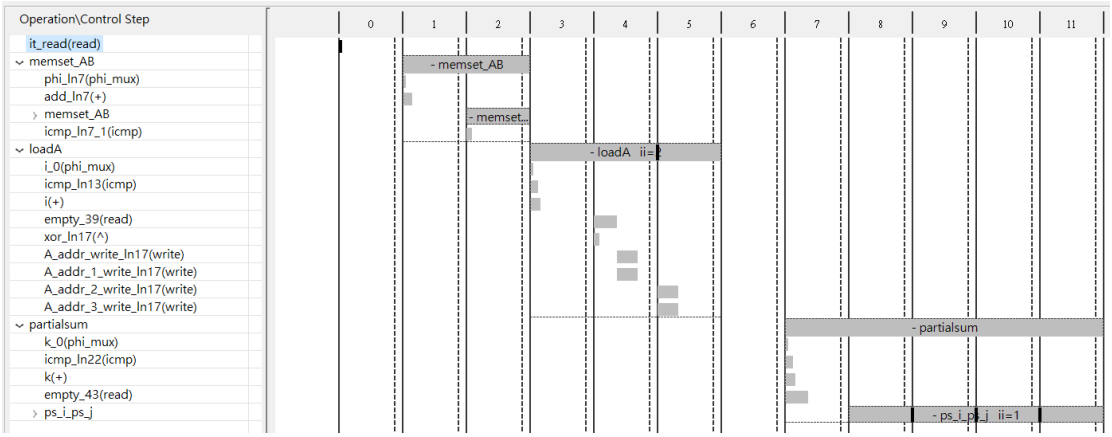
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	201	209	218	191	202	208

Export the report(.html) using the Export Wizard

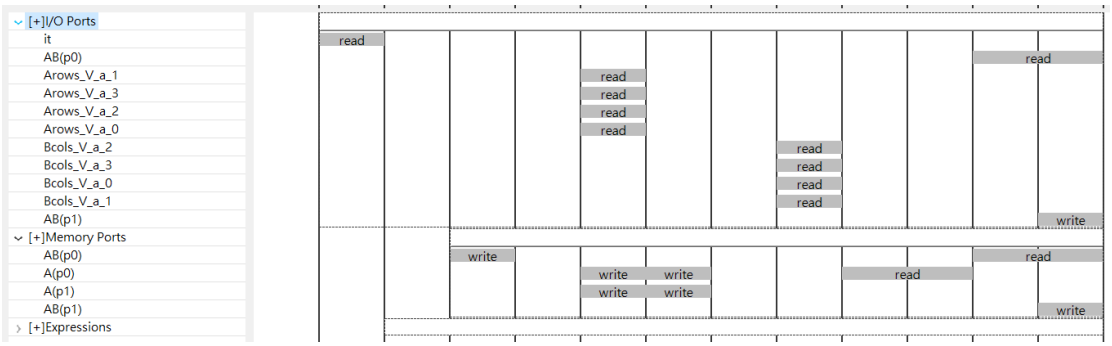
Co-sim 結果正確。

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
Loop_memset_AB_proc8	-	190~207	-	190 ~ 207	-
memset_AB	no	19	5	-	4
loadA	yes	16	3	2	8
partialsum	no	168	21	-	8



由上圖可以總結出 **Top function 當中 latency 的主要來源還是外積運算(partialsum)的累加**，這部分可以透過後面提到的 Loop unrolling 用較多的運算資源換取低運算延遲。



由 resource viewer 也可以看出，I/O port 當中的 Arows 與 Bcols 能夠在一個 cycle 當中提供 4 筆資料(= BLOCK_SIZE)給 IP 進行運算。但由於這個 case 只針對最內層 ps_j Loop 進行 pipeline，沒有要求 partialsum 以平行化的方式完成運算，因此對於 cache AB 的 memory port 的需求僅有一次讀寫。

@ Case: Stream Dataflow and Pipelining ps_i Loop (solution 2)

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
198	215	1.980 us	2.150 us	199	216	dataflow

Detail

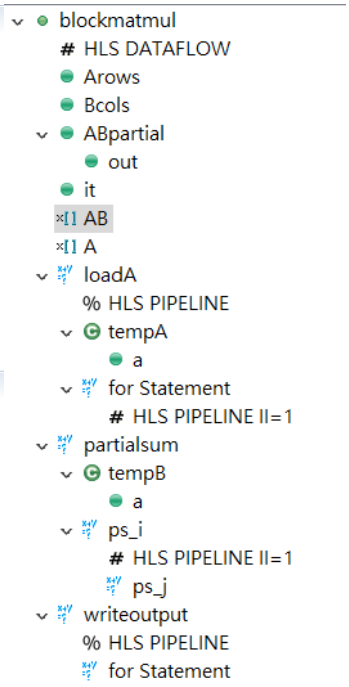
Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	2	12	547	935	0
Memory	2	-	0	0	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	4	12	547	937	0
Available	280	220	106400	53200	0
Utilization (%)	1	5	~0	1	0



Message

Synthesis

WARNING: [SCHED 204-69] Unable to schedule 'load' operation ('AB_load_2', BlockMatrix_design.cpp:35) on array 'AB'

WARNING: [SCHED 204-69] Unable to schedule 'load' operation ('AB_load_5', BlockMatrix_design.cpp:28) on array 'AB'

由上圖可以看出，光是針對 ps_i 進行pipeline (ps_j應進行 Loop Unrolling)，並不足以提升整體運算速度，原因是 **cache AB 所提供的memory bandwidth不足**

@ Case: Stream Dataflow and Pipelining ps_i Loop with RESHAPE (solution 3)

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
98	115	0.980 us	1.150 us	99	116	dataflow

Detail

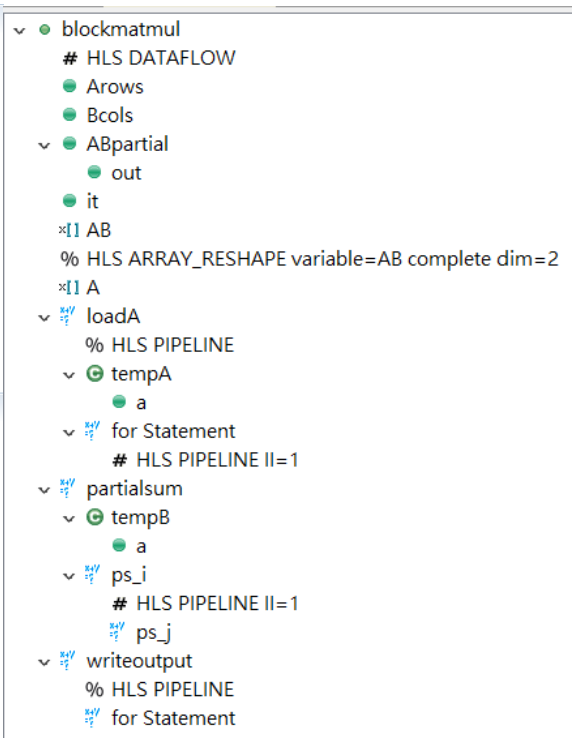
Instance

Loop

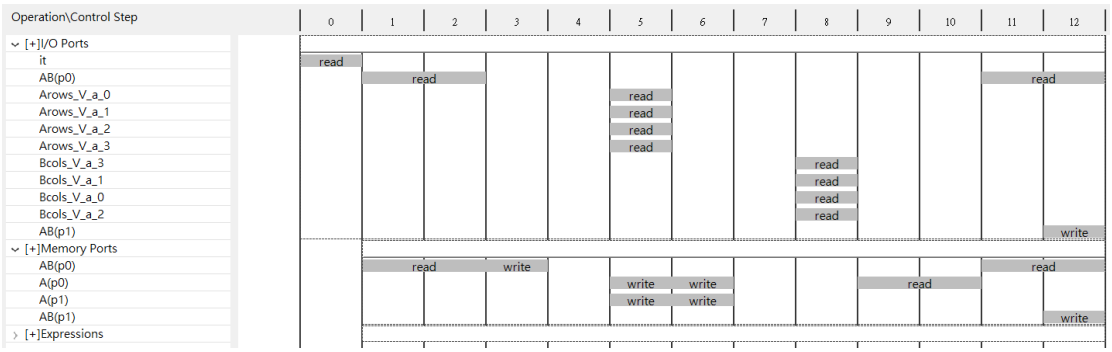
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	2	12	745	2444	0
Memory	8	-	0	0	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	10	12	745	2446	0
Available	280	220	106400	53200	0
Utilization (%)	3	5	~0	4	0



由於 ps_j 進行 Loop Unrolling，因此我選擇**針對 cache AB 的 dimension 2 進行 ARRAY_RESHAPE** (complete)，提供足夠的bandwidth使運算延遲下降約50%，而同時硬體資源的需求也提升(**DSP 4x**, **FF 1.63x**, **LUT 3.07x**)，DSP 需求的上升量推測與 BLOCK_SIZE (=4) 有關。



由resource viewer看出，針對 ps_j 進行 Loop Unrolling 後，要求partialsum以平行化的方式完成運算，因此對於cache AB的memory port的需求提升為兩次讀寫。

@ Case: Stream Dataflow and Pipelining partialsum with RESHAPE (solution 4)

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
15	15	0.150 us	0.150 us	13	13	dataflow

Detail

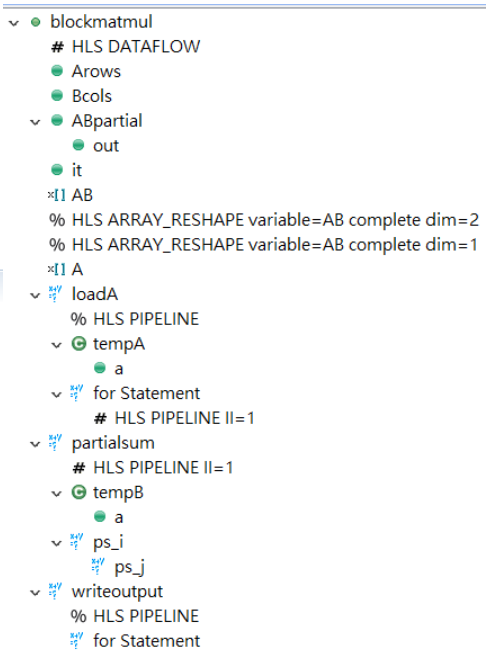
Instance

Loop

Utilization Estimates

Summary

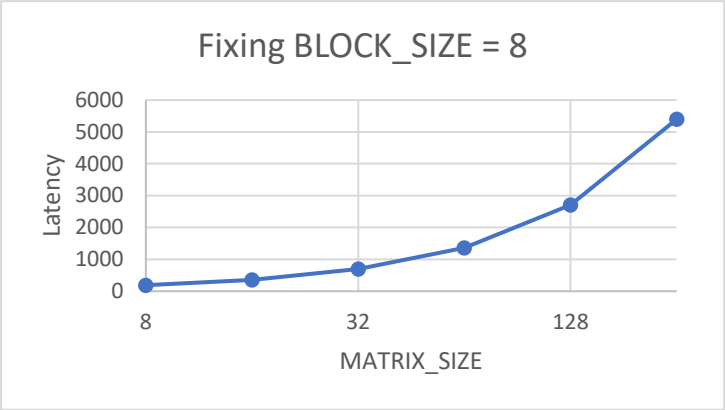
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	46	-
FIFO	29	-	556	858	-
Instance	0	48	1994	6277	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	54	-
Register	-	-	9	-	-
Total	29	48	2559	7235	0
Available	280	220	106400	53200	0
Utilization (%)	10	21	2	13	0



由於 ps_i and ps_j 皆進行 Loop Unrolling，因此我選擇**針對 cache AB 的 dimension 1 and 2 進行 ARRAY_RESHAPE** (complete)，提供足夠的bandwidth使運算延遲下降約**16x**，然而硬體資源的需求也大幅提升(**DSP 16x**, **FF 5.61x**, **LUT 9.08x**)，而 DSP 需求的上升量推測與BLOCK_SIZE^2 (=16)有關。

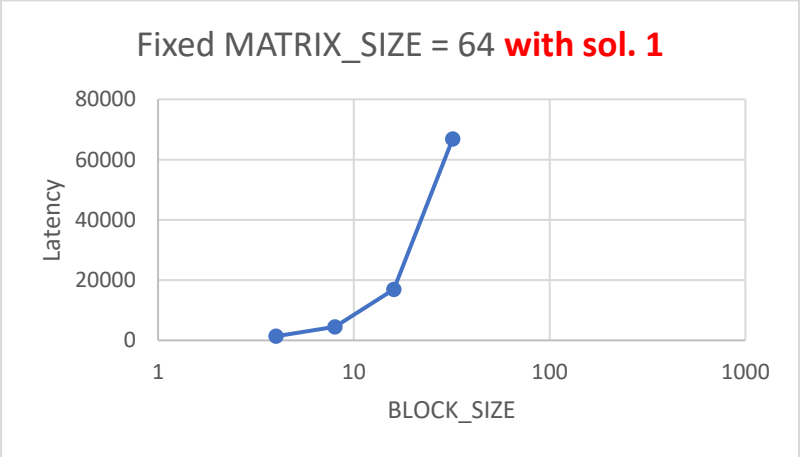
Analyzing BLOCK_SIZE and MATRIX_SIZE:

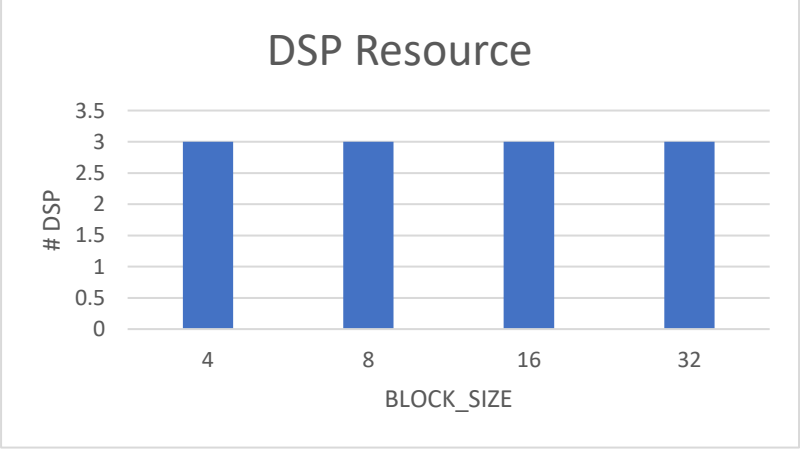
@ Case: Fixing BLOCK_SIZE=8 (using solution 1)



由上圖可以觀察出，固定BLOCK_SIZE = 8，Latency與MATRIX_SIZE大約呈現線性增長的關係 (橫軸為log scale)，然而實際上Naïve matrix multiplication的運算複雜度是$O(MATRIX_SIZE^3)$，顯示 Streaming dataflow 能達到 scalable 的特性。

@ Case: Fixing MATRIX_SIZE=64 (using solution 1)

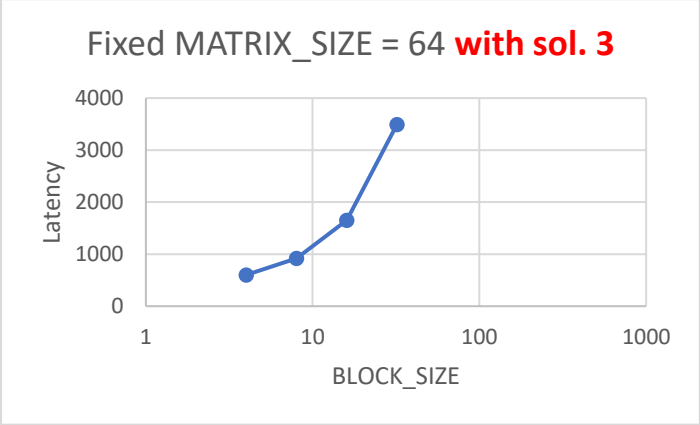


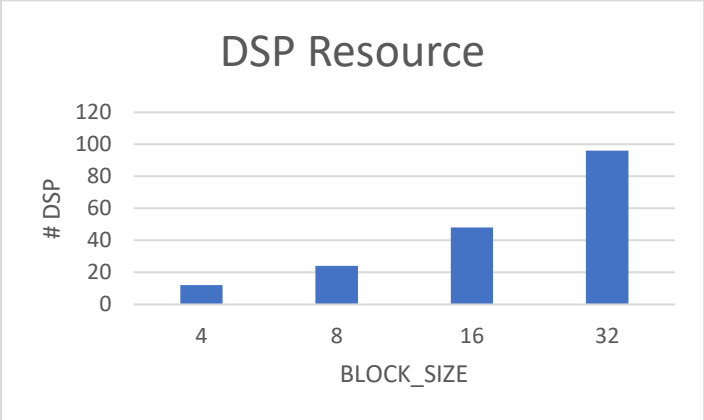


由上圖可以觀察出，固定MATRIX_SIZE = 64，Latency與BLOCK_SIZE大約呈現平方增長的關係 (橫軸為log scale)，主要原因推測是在 **Lab 2 @ Case: Stream Dataflow**

(solution 1)當中，運算資源 (DSP) 並不隨著BLOCK_SIZE增長，然而控制電路的複雜度以及 bit-width requirement 都隨著BLOCK_SIZE增長，導致大的BLOCK_SIZE無法帶來良好的scalability。

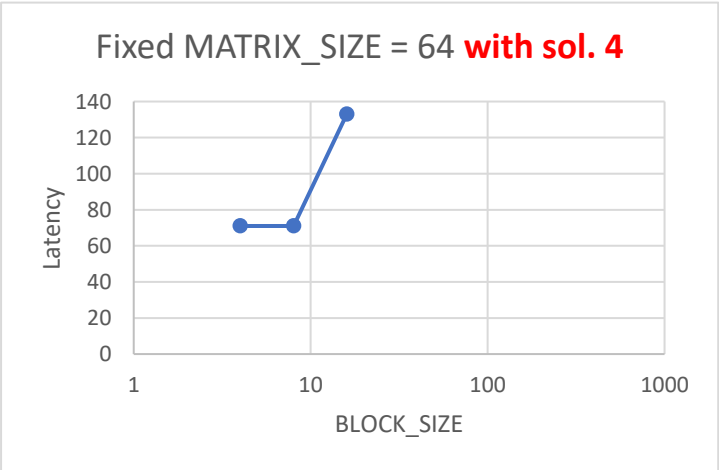
@ Case: Fixing MATRIX_SIZE=64 (using solution 3)

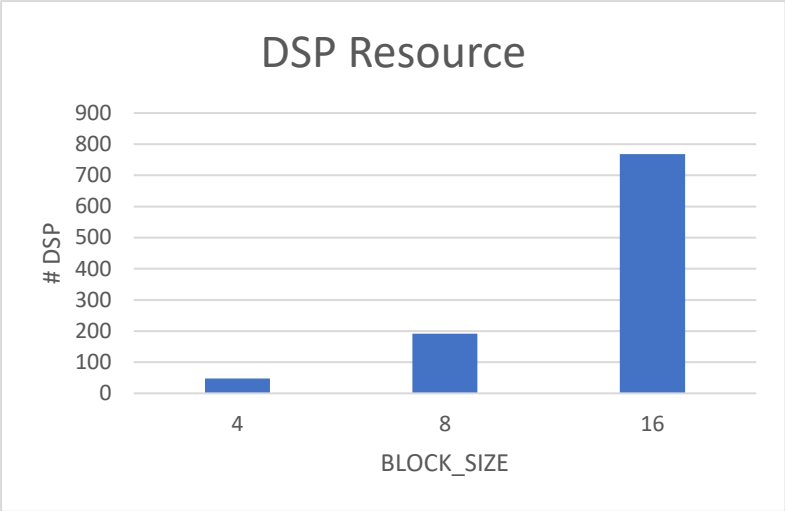




由上圖可以觀察出， Latency與BLOCK_SIZE大約呈現線性增長的關係 (橫軸為log scale)，主要原因推測是在 **Lab 2 @ Case: Stream Dataflow and Pipelining ps_i Loop with RESHAPE (solution 3)** 當中，運算資源 (DSP)隨著BLOCK_SIZE 線性增長，一定程度上改善 scalability。

@ Case: Fixing MATRIX_SIZE=64 (using solution 4)





由上圖可以觀察出， Latency與BLOCK_SIZE**沒有明顯關係**，主要原因推測是在 **Lab 2 @ Case: Stream Dataflow and Pipelining partialsum with RESHAPE (solution 4)**，運算資源 (DSP)隨著 BLOCK_SIZE 呈平方增長，大幅改善 scalability，可見在設計 Block-based Matrix Multiplication IP 時須針對 BLOCK_SIZE and unrolling factor 進行嚴謹的 co-exploration 才能在確保 IP 維持良好的 scalability。

GitHub link:

https://github.com/b04901056/AAHLS_LabB