Design Optimization

1. Lab 1 Optimization a Matrix Multiplier

- Non-pipeline matrix multiplication

```
#include "matrixmul.h"

void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
        #pragma HLS PIPELINE off
          //#pragma HLS LOOP_FLATTEN off
       // Iterate over the columns of the B matrix
       Col: for(int j = 0; j < MAT_B_COLS; j++) {
         #pragma HLS PIPELINE off
          res[i][j] = 0;
          // Do the inner product of a row of A and col of B
          Product: for(int k = 0; k < MAT_B_ROWS; k++) {
             #pragma HLS PIPELINE off
             res[i][j] += a[i][k] * b[k][j];
          }
       }
   }

}
```

Performance & Resource Estimates

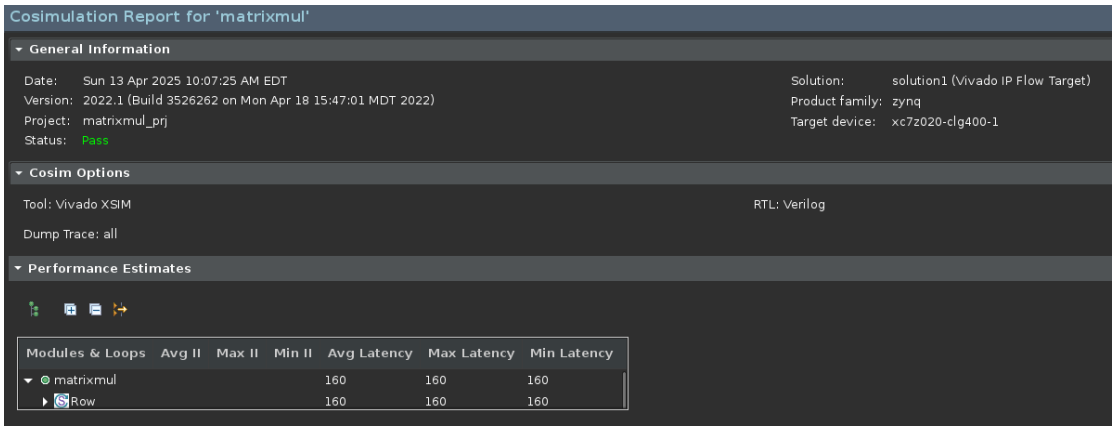
Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul				-	160	2.133E3	-	161	-	no	0	1	46	191	0
Row				-	159	2.120E3	53	-	3	no	-	-	-	-	-

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  5.638 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |      160|      160|  2.133 us|  2.133 us|  161|  161|       no|
    +---------+---------+----------+----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +--------------+---------+---------+----------+-----------+-----------+------+----------+
        |              |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        |   Loop Name  |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +--------------+---------+---------+----------+-----------+-----------+------+----------+
        |- Row         |      159|      159|        53|          -|          -|     3|        no|
        | + Col        |       51|       51|        17|          -|          -|     3|        no|
        |  ++ Product  |       15|       15|         5|          -|          -|     3|        no|
        +--------------+---------+---------+----------+-----------+-----------+------+----------+
```

Synthesis summary

Timeline Trace



Co-simulation

Col Latency = (15[iteration]+1[input]+1[output])*3[trip count] = 51

Row Latency = (51[iteration]+1[input]+1[output])*3 +1[input of Row]= 160

- Pipeline in Product loop

```
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
         #pragma HLS PIPELINE off
          //#pragma HLS LOOP_FLATTEN off
        // Iterate over the columns of the B matrix
        Col: for(int j = 0; j < MAT_B_COLS; j++) {
          #pragma HLS PIPELINE off
           res[i][j] = 0;
           // Do the inner product of a row of A and col of B
           Product: for(int k = 0; k < MAT_B_ROWS; k++) {
              #pragma HLS PIPELINE
              res[i][j] += a[i][k] * b[k][j];
           }
        }
   }

}
```

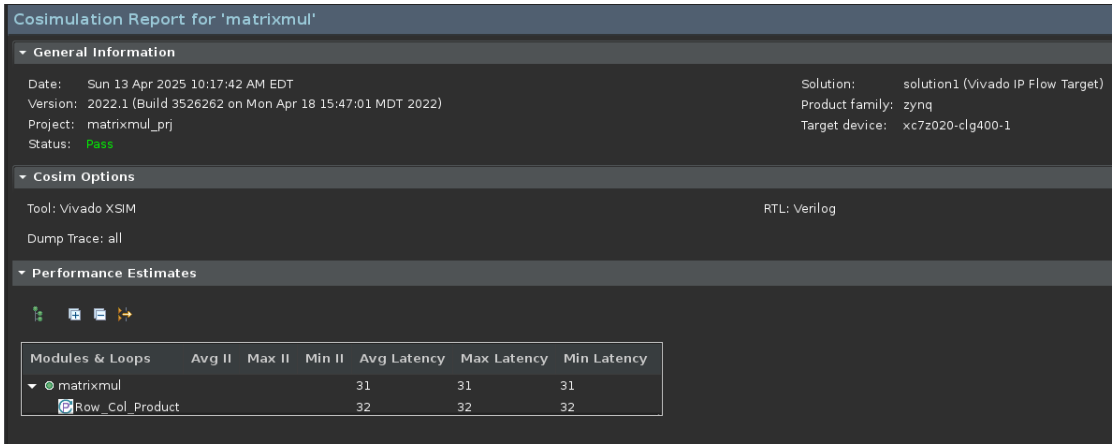
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul			-		33	440.000	-	34	-	no	0	1	241	387	0
Row_Col_Product			-		31	413.000	6	1	27	yes	-	-	-	-	-

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  9.546 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       33|       33|  0.440 us|  0.440 us|   34|   34|       no|
    +---------+---------+----------+----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +------------------+---------+---------+----------+-----------+-----------+-----+----------+
        |                  |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        |     Loop Name    |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +------------------+---------+---------+----------+-----------+-----------+-----+----------+
        |- Row_Col_Product |       31|       31|         6|          1|          1|   27|       yes|
        +------------------+---------+---------+----------+-----------+-----------+-----+----------+
```

Synthesis summary



Co-simulation

Product Latency = (27-1)*1 + 6 = 32

Pipeline example :

k=0 → [1] → [2] → [3] → [4] → [5] → [6] → 輸出

k=1 → [1] → [2] → [3] → [4] → [5] → [6] → 輸出

k=2 → [1] → [2] → [3] → [4] → [5] → [6] → 輸出

因為將 product loop 設定為 pipeline，可以發現 latency 有明顯降低，但是 resource 使用比起 non pipeline 上升許多。

- Pipeline in Col loop

```
#include "matrixmul.h"

void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
        #pragma HLS PIPELINE off
          //#pragma HLS LOOP_FLATTEN off
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < MAT_B_COLS; j++) {
        #pragma HLS PIPELINE
         res[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < MAT_B_ROWS; k++) {
            #pragma HLS PIPELINE off
            res[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul	II Violation		-	-	23	307.000	-	24	-	no	0	2	53	338	0
Row_Col	II Violation	Resource Limitation	-	-	21	280.000	6	2	9	yes	-	-	-	-	-

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  8.381 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       23|       23|  0.307 us|  0.307 us|   24|   24|       no|
    +---------+---------+----------+----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |           |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        | Loop Name |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |- Row_Col  |       21|       21|         6|          2|          1|     9|       yes|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
```

Synthesis summary

Cosimulation Report for 'matrixmul'

General Information

Date: Sun 13 Apr 2025 10:52:16 AM EDT
Version: 2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)
Project: matrixmul_prj
Status: Pass

Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1

Cosim Options

Tool: Vivado XSIM
RTL: Verilog
Dump Trace: all

Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
matrixmul				21	21	21
Row_Col				22	22	22

Co-simulation

Row_Col Latency = (9-1)*2 + 6 = 22

WARNING: [HLS 200-885] The II Violation in module 'matrixmul' (loop 'Row_Col'): Unable to schedule 'load' operation ('a_load_1', matrixmul.cpp:54) on array 'a' due to limited memory ports (II = 1).

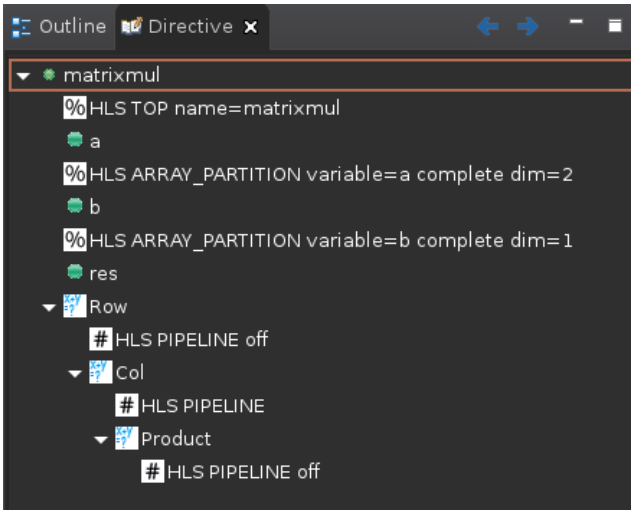
Initiation Interval 是 pipeline 想要達成的間隔（理想是 1），代表每個 cycle 都可以啟動一輪新的計算。HLS 嘗試讓迴圈 Row_Col 達成 II=1，但發現在同一個 cycle 內需要存取記憶體 array a 多次。預設一個 array 對應的是 single-port RAM，每個 cycle 只能做一次存取。res[i][j] += a[i][k] * b[k][j]; 這一行會在同一個 clock cycle 內同時存取多個 array，導致 memory port 不夠用。

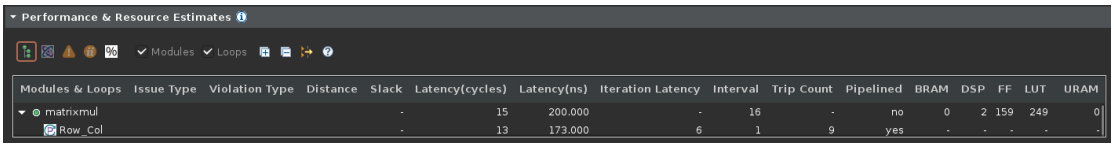
解決方法 : array partition 或者 reshaping

```
#include "matrixmul.h"

void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
        //#pragma HLS PIPELINE off
          //#pragma HLS LOOP_FLATTEN off
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < MAT_B_COLS; j++) {
        //#pragma HLS PIPELINE
         res[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < MAT_B_ROWS; k++) {
        //#pragma HLS PIPELINE off
            res[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

此外，如果不加入任何 pragma，HLS 也會自動將 Col loop 以 pipeline 合成，因此會出現相同的問題。

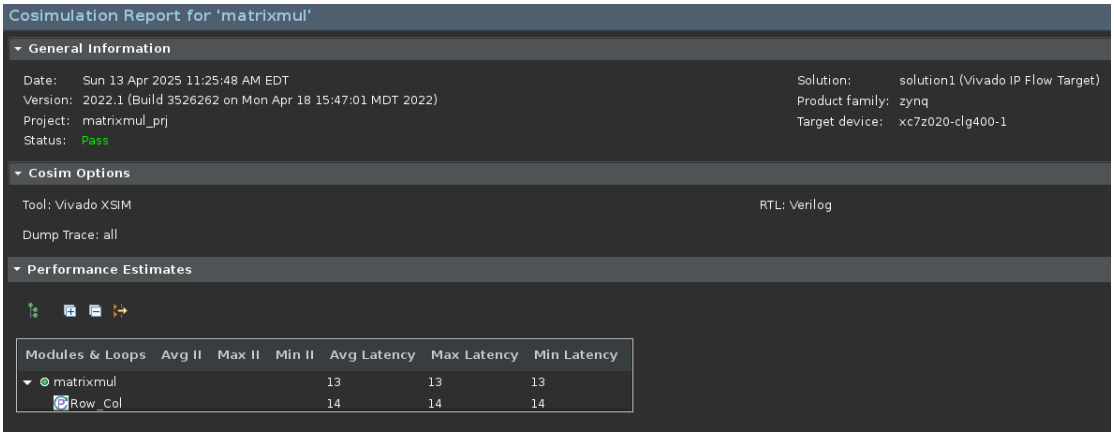
- Pipeline in Col loop & Array Partition






```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  8.592 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       15|       15|  0.200 us|  0.200 us|   16|   16|       no|
    +---------+---------+----------+----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |           |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        | Loop Name |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |- Row_Col  |       13|       13|         6|          1|          1|     9|       yes|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
```

Synthesis summary

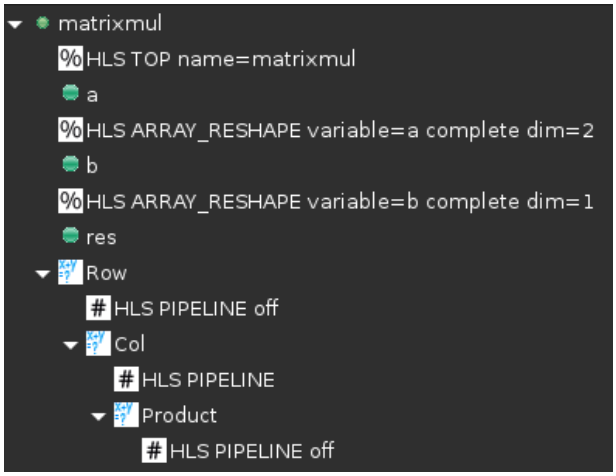


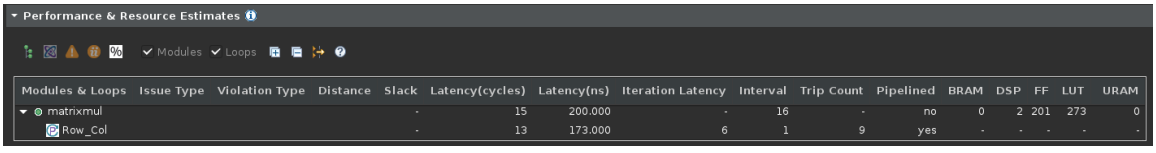
Co-simulation

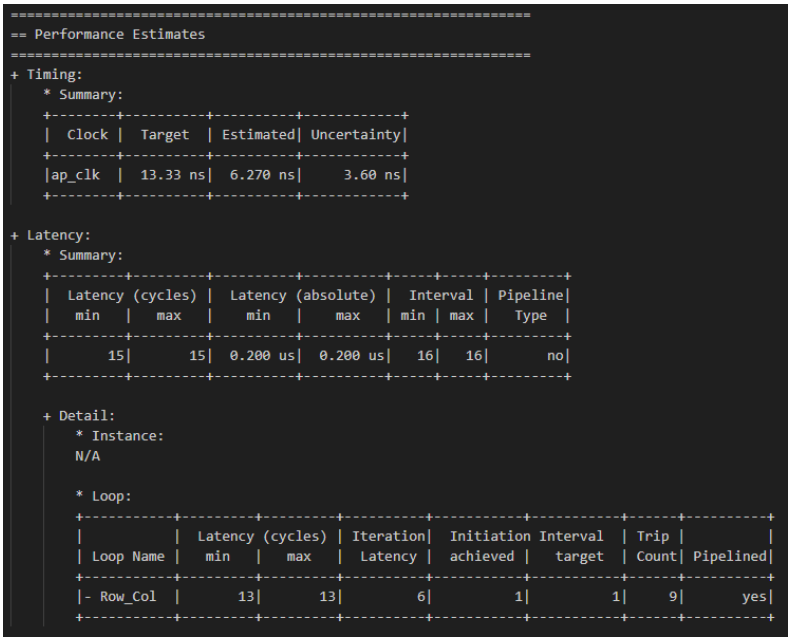
Row_Col Latency = (9-1)*1 + 6 = 14

利用 pragma 將 array a 和 array b 做 array partition，HLS 編譯器會嘗試將每一個元素放在不同的硬體資源上，使得在硬體上能夠並行處理這些元素，這樣就不需要透過 memory port 依序訪問 array 中的每個元素，可有效解決 limited memory ports，但所需硬體資源也因此上升。

- Pipeline in Col loop & Array Reshape

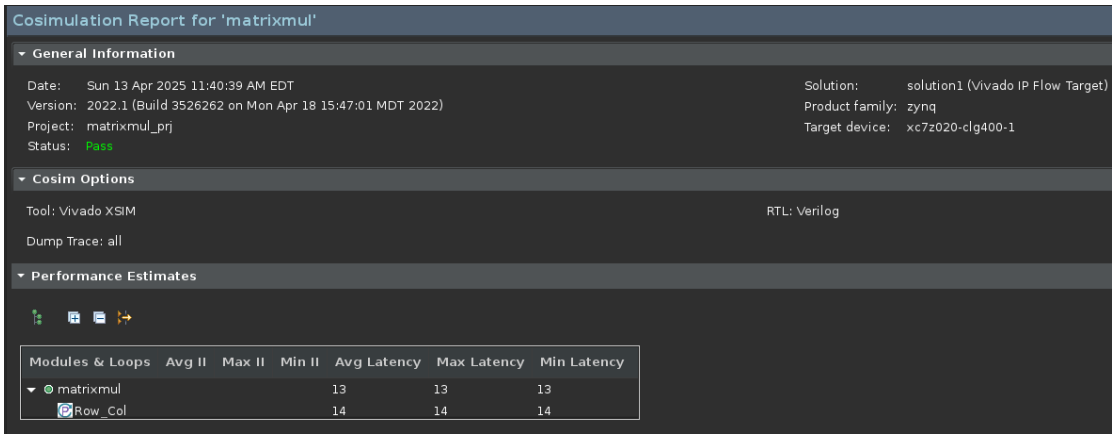






```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  6.270 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       15|       15|  0.200 us|  0.200 us|   16|   16|       no|
    +---------+---------+----------+----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |           |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        | Loop Name |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |- Row_Col  |       13|       13|         6|          1|          1|     9|       yes|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
```

Synthesis summary

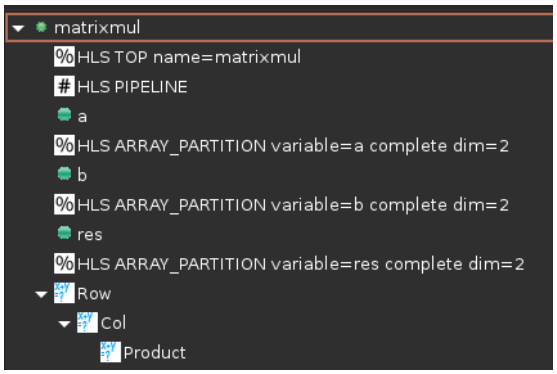


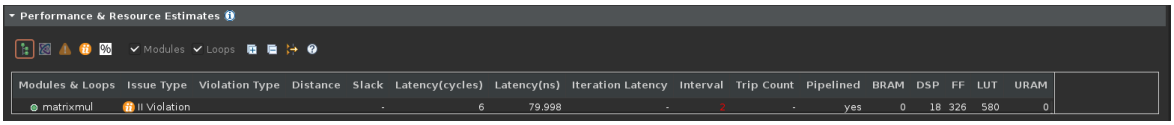
Co-simulation

Row_Col Latency = (9-1)*1 + 6 = 14

Array reshape 一樣可解決 limited memory ports 的問題，達成相同的 latency。差別在於，array reshape 是運用位寬較大的記憶體，一次讀取多筆資料，因此邏輯會變得較為複雜，導致 FF 和 LUT 使用量上升。

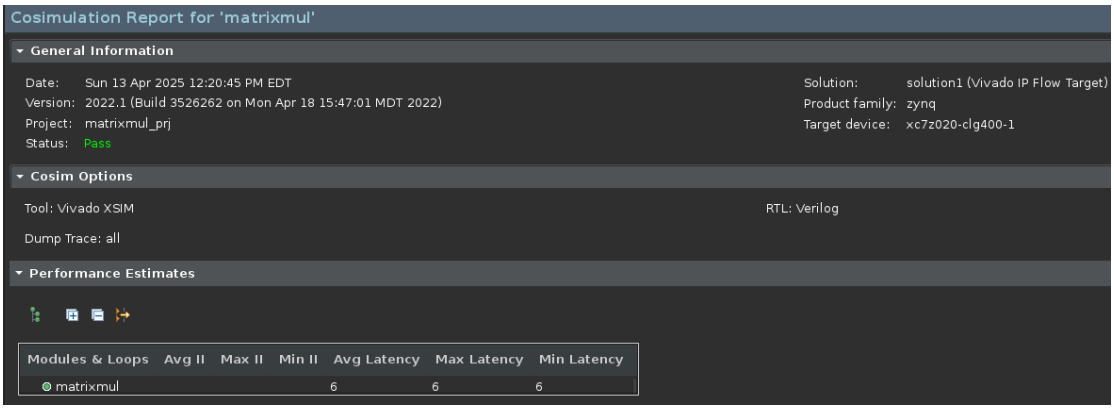
- pipeline at function level






```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  13.33 ns|  6.492 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+-----+-----+---------+
    |  Latency (cycles) |   Latency (absolute)  |  Interval | Pipeline|
    |   min   |   max   |    min    |    max    | min | max |   Type  |
    +---------+---------+-----------+-----------+-----+-----+---------+
    |        6|        6|  79.998 ns|  79.998 ns|    2|    2|      yes|
    +---------+---------+-----------+-----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        N/A
```

Synthesis summary



Co-simulation

Latency = 6

Function level pipeline 可讓 latency 大幅降低，但所需的硬體資源卻非常多。此外，仍有 limit memory port 的 warning 產生，雖然有使用 array partition，但是 HLS 好像沒有將 array b partition，導致 warning 產生，因此需放寬 II 以達成 pipeline。

2. LAB 2 C Code Optimized for I/O Access

```
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
#pragma HLS ARRAY_RESHAPE variable=b complete dim=1
#pragma HLS ARRAY_RESHAPE variable=a complete dim=2
#pragma HLS INTERFACE ap_fifo port=a
#pragma HLS INTERFACE ap_fifo port=b
#pragma HLS INTERFACE ap_fifo port=res
  mat_a_t a_row[MAT_A_ROWS];
  mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
  int tmp = 0;

  // Iterate over the rowa of the A matrix
  Row: for(int i = 0; i < MAT_A_ROWS; i++) {
    // Iterate over the columns of the B matrix
    Col: for(int j = 0; j < MAT_B_COLS; j++) {
#pragma HLS PIPELINE rewind
      // Do the inner product of a row of A and col of B
      tmp=0;
      // Cache each row (so it's only read once per function)
      if (j == 0)
        Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
          a_row[k] = a[i][k];

       // Cache all cols (so they are only read once per function)
     if (i == 0)
           Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
              b_copy[k][j] = b[k][j];

      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
        tmp += a_row[k] * b_copy[k][j];
      }
      res[i][j] = tmp;
    }
  }
}
```

- Explain the requirement for Stream/FIFO access

  在 HLS 中，預設的 array 是 map 到 RAM，需要多次讀取同一筆資料，這樣會可能會出現 limited memory port 的問題，如同 Lab1 所示。而透過 HLS 的 ap_fifo 介面，能以類似資料流（dataflow）的方式將資料串流至計算邏輯中，避免重複讀取，降低記憶體壅塞。此外，我們利用快取緩衝區

（cache buffer）將 A matrix 的 row 以及 B matrix 的 column 暫存，確保每筆資料僅被載入一次，大幅提升資料重用率，也有利於 pipeline 的進行。

- Observe MM access pattern, Figure 7-20

  由程式碼可看出，對於每一筆輸出 res[i][j]，會存取：

  第 i row 的 a：a[i][0], a[i][1], a[i][2]

  第 j column 的 b：b[0][j], b[1][j], b[2][j]

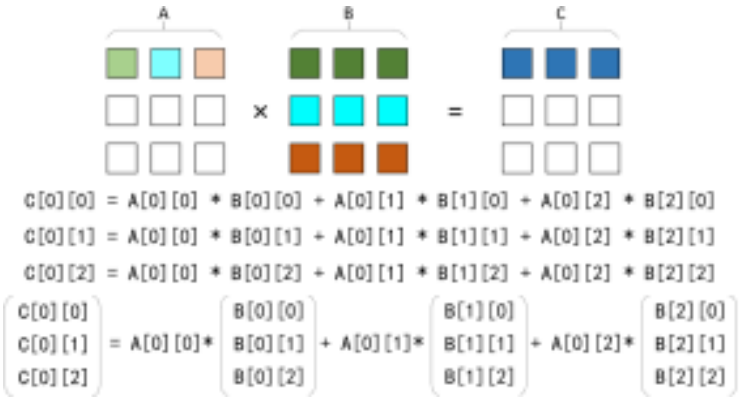
  a matrix 的 row 存取具有 temporal locality：對於每個 i 固定的情況下，a[i][k] 的三個元素會被連續讀取，並且在處理 res[i][0], res[i][1], res[i][2] 時會重複使用。

  b matrix 的 column 存取具有 spatial locality：在 j 固定時，b[k][j] 會被連續存取，且對於不同的 i，同一列的 b[k][j] 會被多次重複使用。

  因此：

  a[i][*] 可在第一個 j == 0 時先快取整列，後續重用。

  b[*][j] 可在第一個 i == 0 時快取整欄，後續重用。



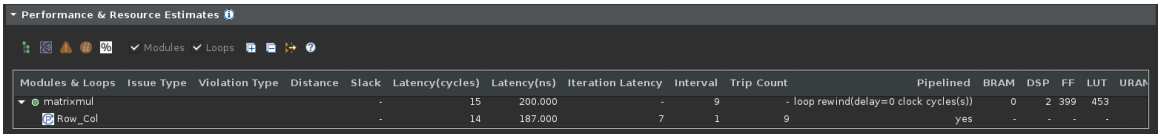
- Derive the code modification, explain it. Cache buffer is introduced and how it is used

  在進入每個 j（column）時：a_row[k] = a[i][k];：只在 j == 0 把 A 的一整 row cache 起來

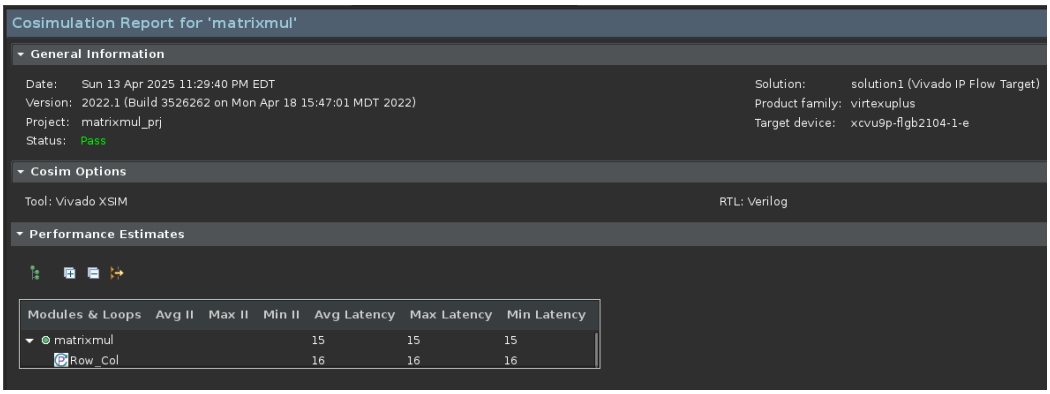
  在進入每個 i（row）時：b_copy[k][j] = b[k][j];：只在 i == 0 把 B 的一整 column cache 起來

每筆資料只從 FIFO 讀一次，把所有的 a 和 b 內容先一次性全部拉進來快取，後面再從快取做真正的 matrix multiply，可減少 memory access 頻率並提高 performance

- Compare latency and resource used for each optimization step




```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  | 13.33 ns|  3.251 ns|     3.60 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+-----------------------------------------+
    |  Latency (cycles) |  Latency (absolute) |  Interval |                Pipeline                 |
    |   min   |   max   |    min   |    max   | min | max |                  Type                   |
    +---------+---------+----------+----------+-----+-----+-----------------------------------------+
    |       14|       15| 0.187 us| 0.200 us|    9|    9| loop rewind stp(delay=0 clock cycles(s))|
    +---------+---------+----------+----------+-----+-----+-----------------------------------------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |           |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
        | Loop Name |   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
        |- Row_Col  |       14|       14|         7|          1|          1|     9|       yes|
        +-----------+---------+---------+----------+-----------+-----------+------+----------+
```



Latency = (9-1)*1+7 = 15

Resource 的部分因為有利用 cache 將讀入的資料存起來，因此，FF 和 LUT 都有明顯上升。