ESE 532 Homework 5

Sheil Sarda

Partner: Liwen Zhong

1. Initial CPU implementation and HLS Kernel.

a.

The visualized application timeline is shown below:

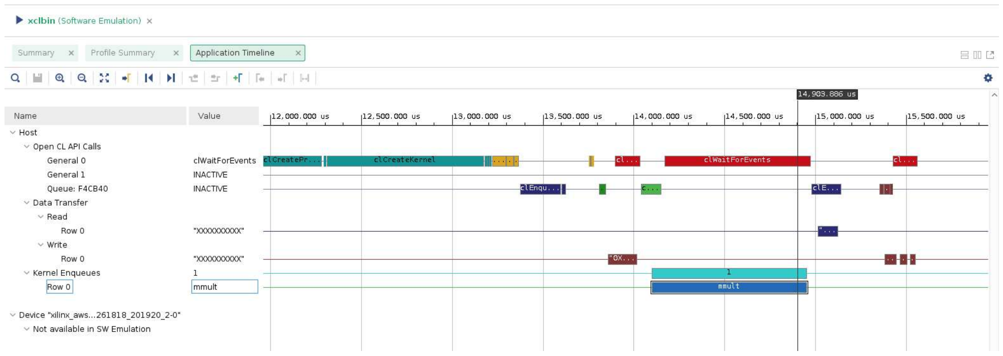


Figure 1

The latency of mmult kernel is 0.85060 ms.

b.

```
Starting C simulation ...
/opt/Xilinx/Vitis/2020.1/bin/vitis_hls /home/centos/hw5/solution1/csim.tcl
INFO: [HLS 200-10] Running
'/opt/Xilinx/Vitis/2020.1/bin/unwrapped/lnx64.o/vitis_hls'
INFO: [HLS 200-10] For user 'centos' on host 'ip-172-31-76-23.ec2.internal'
(Linux_x86_64 version 3.10.0-1127.10.1.el7.x86_64) on Tue Oct 13 13:47:57 UTC 2020
INFO: [HLS 200-10] On os "CentOS Linux release 7.7.1908 (Core)"
INFO: [HLS 200-10] In directory '/home/centos'
Sourcing Tcl script '/home/centos/hw5/solution1/csim.tcl'
INFO: [HLS 200-10] Opening project '/home/centos/hw5'.
INFO: [HLS 200-10] Adding design file
'ese532_code/hw5/hls/MatrixMultiplication.cpp' to the project
INFO: [HLS 200-10] Adding design file 'ese532_code/hw5/hls/MatrixMultiplication.h'
to the project
INFO: [HLS 200-10] Adding test bench file 'ese532_code/hw5/hls/Testbench.cpp' to
the project
INFO: [HLS 200-10] Opening solution '/home/centos/hw5/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 8ns.
INFO: [HLS 200-10] Setting target device to 'xcvu9p-flga2104-1-i'
INFO: [HLS 200-1505] Using flow_target 'vitis'
```

```
Resolution: For help on HLS 200-1505 see www.xilinx.com/html_docs/xilinx2020_1/hls-
guidance/200-1505.html
INFO: [HLS 200-435] Setting 'config_flow -target' configuration: config_interface -
m_axi_latency=64
INFO: [HLS 200-435] Setting 'config_flow -target' configuration: config_interface -
m_axi_alignment_byte_size=64
INFO: [HLS 200-435] Setting 'config_flow -target' configuration: config_interface -
m_axi_max_widen_bitwidth=512
INFO: [HLS 200-435] Setting 'config_flow -target' configuration: config_interface -
default_slave_interface=s_axilite
INFO: [HLS 200-435] Setting 'config_flow -target' configuration: config_rtl -
register_reset_num=3
INFO: [HLS 200-1464] Running solution command: config_compile -dump_cfg=false
INFO: [HLS 200-1464] Running solution command: config_compile -name_max_length=80
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [HLS 200-1464] Running solution command: config_compile -
no_signed_zeros=false
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [HLS 200-1464] Running solution command: config_compile -pipeline_loops=0
INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or
its parent loop when its trip count is less than or equal 0.
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [HLS 200-1464] Running solution command: config_compile -pipeline_style=stp
INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or
its parent loop when its trip count is less than or equal 0.
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [HLS 200-1464] Running solution command: config_compile -
pragma_strict_mode=false
INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or
its parent loop when its trip count is less than or equal 0.
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [HLS 200-1464] Running solution command: config_compile -
unsafe_math_optimizations=false
INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or
its parent loop when its trip count is less than or equal 0.
```

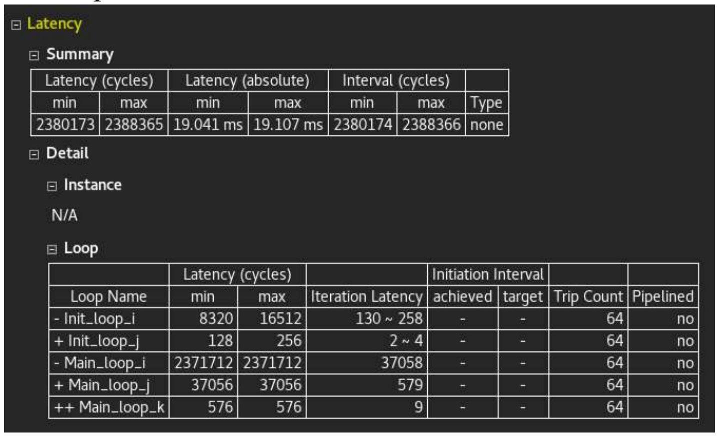
```
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1173] Reordering floating point operations aggressively.
INFO: [XFORM 203-1176] Optimizing floating point comparison without checking NaN.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 8ns.
INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or
its parent loop when its trip count is less than or equal 0.
INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its
signedness.
INFO: [XFORM 203-1173] Reordering floating point operations aggressively.
INFO: [XFORM 203-1176] Optimizing floating point comparison without checking NaN.
INFO: [XFORM 203-1161] The maximum of name length is set into 80.
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   Compiling ../../../../ese532_code/hw5/hls/Testbench.cpp in debug mode
   Compiling ../../../../ese532_code/hw5/hls/MatrixMultiplication.cpp in debug mode
   Generating csim.exe
TEST PASSED
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

c.

We are doing matrix multiplying in our application. To test the functionality of the code, we test the code by giving it a test input, just like how we test a circuit with a test input signal. The Testbench.cpp does the job by giving two test matrix and compare the actual output with the expected output.

d.

The synthesis report is shown below:



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2380173	2388365	19.041 ms	19.107 ms	2380174	2388366	none

Detail

Instance

N/A

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	8320	16512	130 ~ 258	-	-	64	no
+ Init_loop_j	128	256	2 ~ 4	-	-	64	no
- Main_loop_i	2371712	2371712	37058	-	-	64	no
+ Main_loop_j	37056	37056	579	-	-	64	no
++ Main_loop_k	576	576	9	-	-	64	no

Figure 2

The expected latency is 19.041 ~ 19.107ms.

e.

The resource utilization estimates are shown below:

Name	BRAM	DSP	FF	LUT	URAM
Total	76	5	7600	5532	0

f.

The time schedule of multiplication (the yellow box) is shown below:

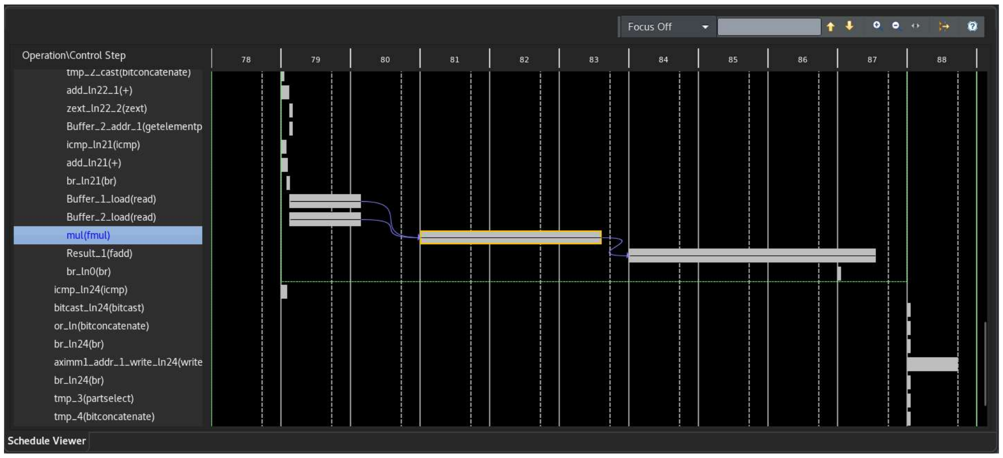


Figure 3

The multiplication takes approximately 2.6 cycles.

g.

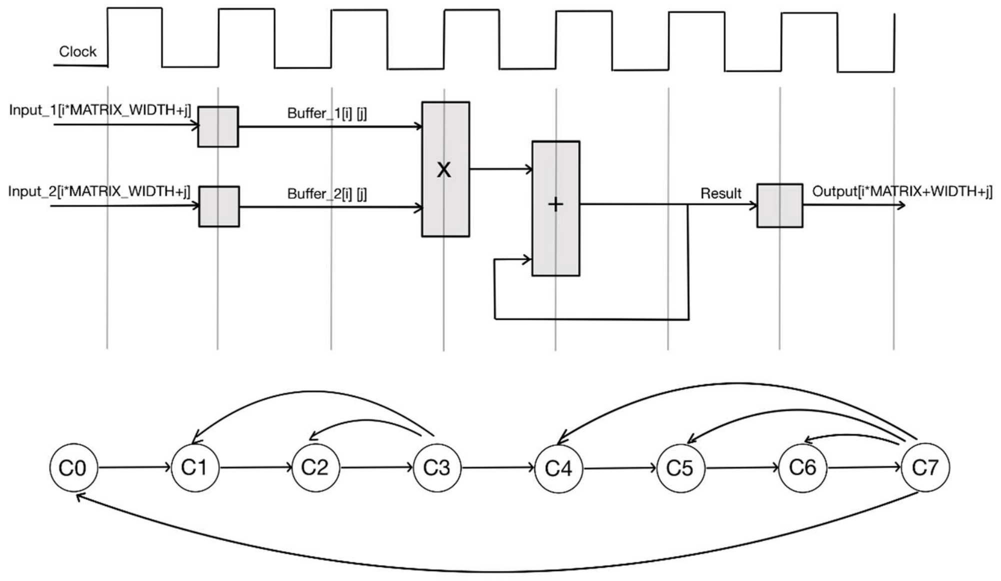


Figure 4

h.

There are many factors that slow down the accelerator. For example, the loops are unpipelined and unparalleled. The reading and writing to the public memory also take a relatively long time.

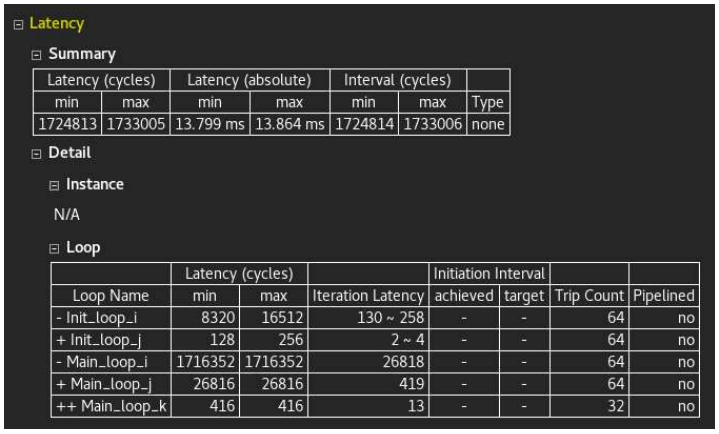
2. HLS Kernel Optimization: Loop Unrolling

a.

The code is shown below:

```
Main_loop_k: for (int k = 0; k < MATRIX_WIDTH; k++) {
    #pragma HLS unroll factor=2
    Result += Buffer_1[i][k] * Buffer_2[k][j];
}
```

The synthesis report is shown below:



⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1724813	1733005	13.799 ms	13.864 ms	1724814	1733006	none

⊟ Detail

⊟ Instance

N/A

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	8320	16512	130 ~ 258	-	-	64	no
+ Init_loop_j	128	256	2 ~ 4	-	-	64	no
- Main_loop_i	1716352	1716352	26818	-	-	64	no
+ Main_loop_j	26816	26816	419	-	-	64	no
++ Main_loop_k	416	416	13	-	-	32	no

Figure 5

The expected latency is 13.799 ~ 13.861ms.

Loops in the C/C++ functions are kept rolled by default. When loops are rolled, synthesis creates the logic for one iteration of the loop, and the RTL design executes this logic for each iteration of the loop in sequence. A loop is executed for the number of iterations specified by the loop induction variable. Using the UNROLL pragma we can unroll loops to increase data access and throughput.

b.

The unrolled loop will look like the following:

```
Main_loop_k: for (int k = 0; k < MATRIX_WIDTH; k+=2) {
    Result += Buffer_1[i][k] * Buffer_2[k][j];
    if (k + 1 >= MATRIX_WIDTH) break;
    Result += Buffer_1[i][k + 1] * Buffer_2[k + 1][j];
}
```

c.

The time schedule of multiplication (the yellow box) is shown below:

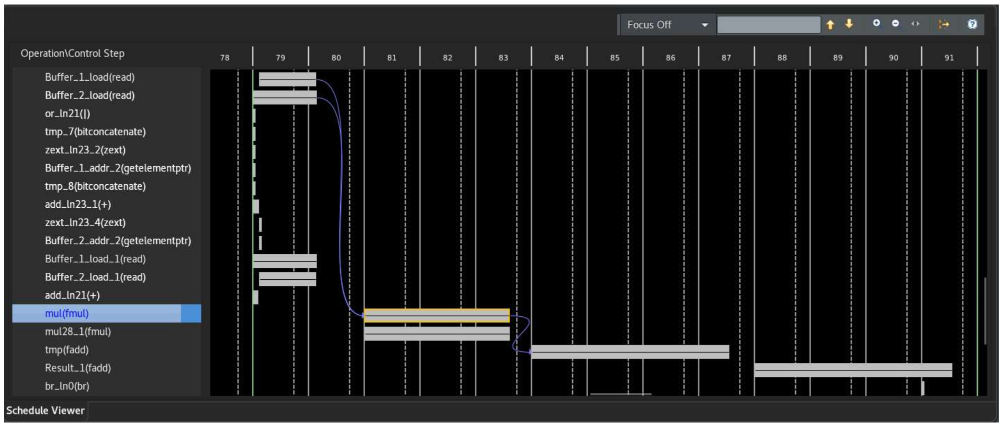


Figure 6

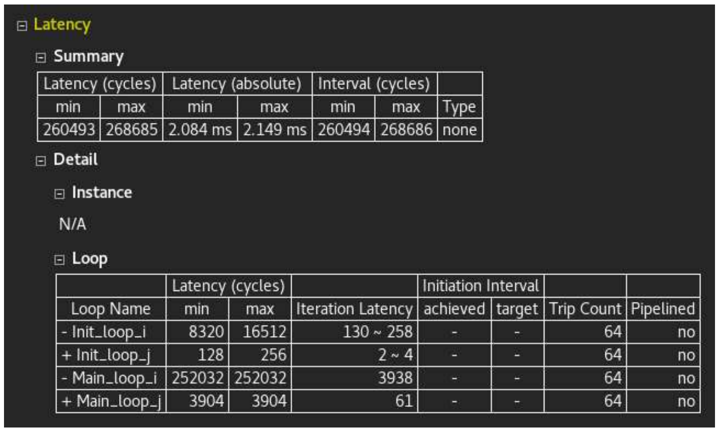
We can see the buffer loading (Buffer_1_load_1 and Buffer_2_load_2) operations and multiplication (fmul) operations are shared by multiple operations.

d.

The code is shown below:

```
Main_loop_k: for (int k = 0; k < MATRIX_WIDTH; k++) {
    #pragma HLS unroll
    Result += Buffer_1[i][k] * Buffer_2[k][j];
}
```

The synthesis report is shown below. The latency estimation is shown correctly and is 2.084 ~ 2.149ms.



⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
260493	268685	2.084 ms	2.149 ms	260494	268686	none

⊟ Detail

⊟ Instance

N/A

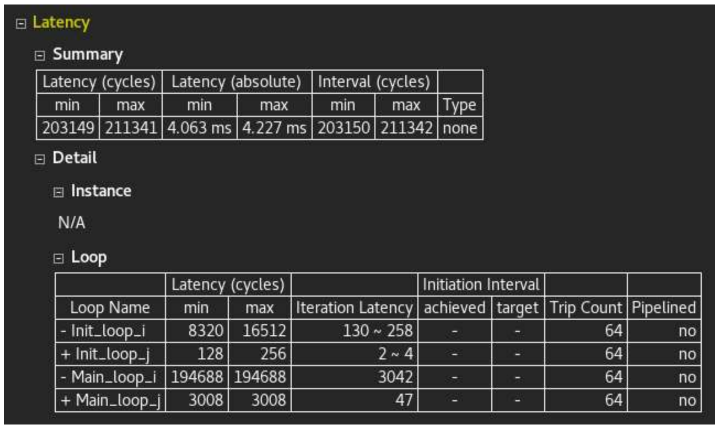
⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	8320	16512	130 ~ 258	-	-	64	no
+ Init_loop_j	128	256	2 ~ 4	-	-	64	no
- Main_loop_i	252032	252032	3938	-	-	64	no
+ Main_loop_j	3904	3904	61	-	-	64	no

Figure 7

e.

The synthesis report is shown below. The estimated latency is 4.063 ~ 4.227ms.


Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
203149	211341	4.063 ms	4.227 ms	203150	211342	none

Detail

Instance

N/A

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	8320	16512	130 ~ 258	-	-	64	no
+ Init_loop_j	128	256	2 ~ 4	-	-	64	no
- Main_loop_i	194688	194688	3042	-	-	64	no
+ Main_loop_j	3008	3008	47	-	-	64	no

Figure 8

f.

The resource utilization estimates are shown below:

Name	BRAM	DSP	FF	LUT	URAM
Total	76	14	11517	8760	0

g.

The results show that unrolling loops does not parallelize floating-point additions.

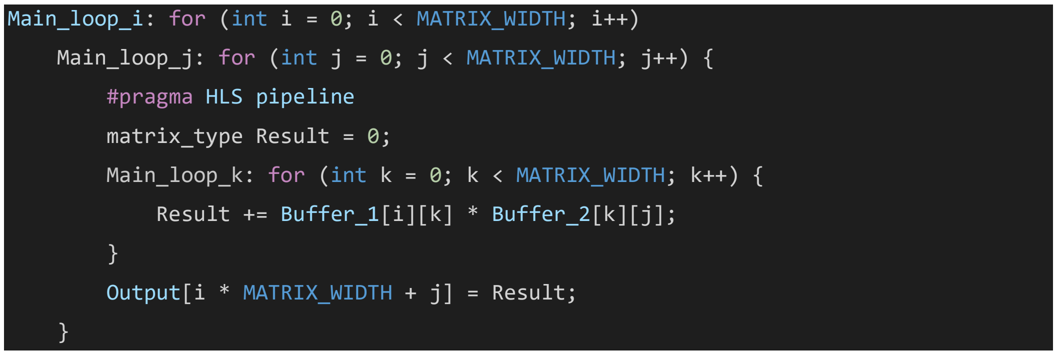
f.

Based on the performance of the accelerators in 1d and 2d, we choose the one in 2b for more parallelism and can achieve the throughput of approximately 500 operations/s.

3. HLS Kernel Optimization: Pipelining

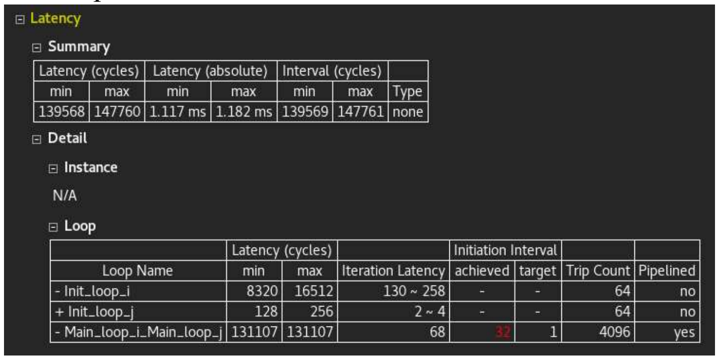
a.

The code is shown below:



```
Main_loop_i: for (int i = 0; i < MATRIX_WIDTH; i++)
    Main_loop_j: for (int j = 0; j < MATRIX_WIDTH; j++) {
        #pragma HLS pipeline
        matrix_type Result = 0;
        Main_loop_k: for (int k = 0; k < MATRIX_WIDTH; k++) {
            Result += Buffer_1[i][k] * Buffer_2[k][j];
        }
        Output[i * MATRIX_WIDTH + j] = Result;
    }
```

The synthesis report is shown below:



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
139568	147760	1.117 ms	1.182 ms	139569	147761	none

Detail

Instance

N/A

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	8320	16512	130 ~ 258	-	-	64	no
+ Init_loop_j	128	256	2 ~ 4	-	-	64	no
- Main_loop_i_Main_loop_j	131107	131107	68	32	1	4096	yes

Figure 9

We achieved an II of 32 cycles.

b.

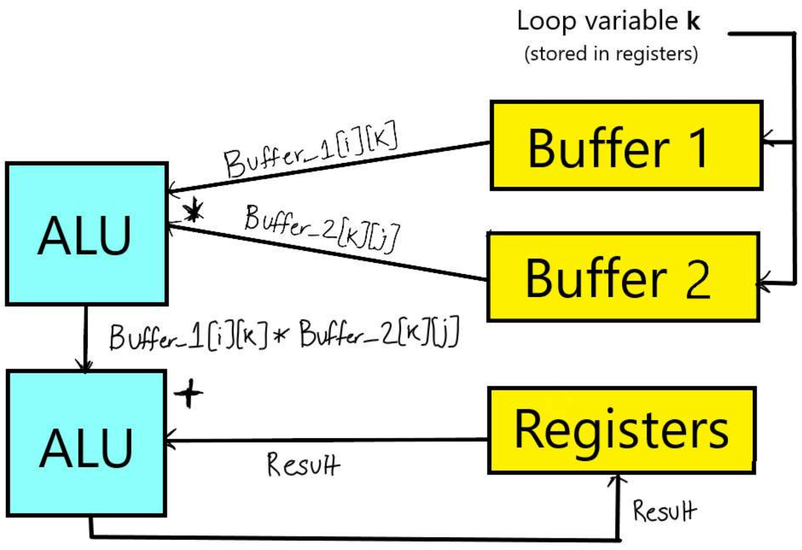


Figure 10

c.

4096 words/cycle

d.

The minimal initiation interval is limited by the number of words loaded from the buffer per cycle. We can achieve better performance by increasing the loading speed.

e.

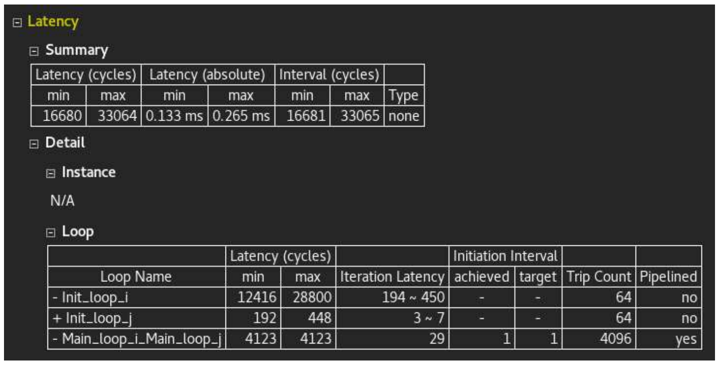
Arrays are implemented as block RAM which only has a maximum of two data ports. This can limit the throughput of a read/write (or load/store) intensive algorithm. The bandwidth can be improved by splitting the array (a single block RAM resource) into multiple smaller arrays (multiple block RAMs), effectively increasing the number of ports.

f.

The code is shown below:

```
void mmult(const matrix_type Input_1[MATRIX_WIDTH * MATRIX_WIDTH],
        const matrix_type Input_2[MATRIX_WIDTH * MATRIX_WIDTH],
        matrix_type Output[MATRIX_WIDTH * MATRIX_WIDTH]) {
#pragma HLS INTERFACE m_axi port=Input_1 bundle=aximm1
#pragma HLS INTERFACE m_axi port=Input_2 bundle=aximm2
#pragma HLS INTERFACE m_axi port=Output bundle=aximm
    matrix_type Buffer_1[MATRIX_WIDTH][MATRIX_WIDTH];
    matrix_type Buffer_2[MATRIX_WIDTH][MATRIX_WIDTH];
#pragma HLS array_partition variable=Buffer_1 block factor=32 dim=0
#pragma HLS array_partition variable=Buffer_2 block factor=32 dim=0
    Init_loop_i: for (int i = 0; i < MATRIX_WIDTH; i++)
        Init_loop_j: for (int j = 0; j < MATRIX_WIDTH; j++) {
            Buffer_1[i][j] = Input_1[i * MATRIX_WIDTH + j];
            Buffer_2[i][j] = Input_2[i * MATRIX_WIDTH + j];
        }
    Main_loop_i: for (int i = 0; i < MATRIX_WIDTH; i++)
        Main_loop_j: for (int j = 0; j < MATRIX_WIDTH; j++) {
            #pragma HLS pipeline
            matrix_type Result = 0;
            Main_loop_k: for (int k = 0; k < MATRIX_WIDTH; k++) {
                Result += Buffer_1[i][k] * Buffer_2[k][j];
            }
            Output[i * MATRIX_WIDTH + j] = Result;
        }
}
```

The synthesis report is shown below:



⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
16680	33064	0.133 ms	0.265 ms	16681	33065	none

⊟ Detail

⊟ Instance

N/A

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Init_loop_i	12416	28800	194 ~ 450	-	-	64	no
+ Init_loop_j	192	448	3 ~ 7	-	-	64	no
- Main_loop_i_Main_loop_j	4123	4123	29	1	1	4096	yes

Figure 11

The estimated latency is 0.133 ~ 0.265ms.

g.

Name	BRAM	DSP	FF	LUT	URAM
Total	90	318	299579	884981	0

h.

Completed.

4. Vitis Analyzer

a.

Completed.

b.

Completed.

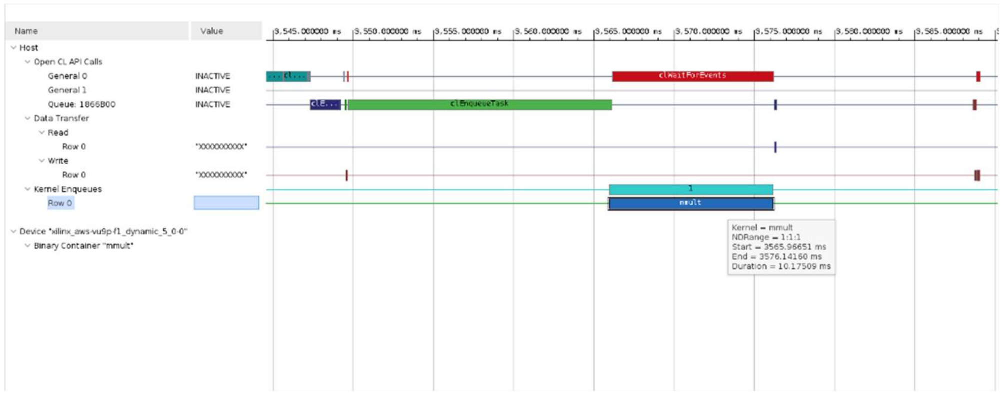
c.

Completed.

d.

The latency of the mmult kernel is 1.02ms.

e.



Kernel = mmult
NDRange = 1:1:1
Start = 3565.96651 ms
End = 3576.14160 ms
Duration = 10.17509 ms

Figure 12

5. Breakeven and Net Acceleration

a.

$$S_{fpga} = \frac{T_{seq}}{T_{fpga}} = 8.33$$

b.

$$T_{setup} = 23.5ms$$

c.

$$T_{setup} = 0.45ms$$

d.

T_{seq} should be more than 24.1ms.

e.

$$T_{seq} = K_{seq} N^2$$

Where K_{seq} is the constant for the processor.

f.

$$T_{fpga} \approx K_{fpga} N$$

Where K_{fpga} is the constant for the FPGA.

g.

$$T_{transfer=} K_{transfer} N^2$$

Where $K_{transfer}$ is the constant for the FPGA.

h.

We have:

$$T_{accel} = T_{seq}$$

Solve the equation and we get the minimum integer N = 578.

i.

To achieve a 10× speedup, the minimum N is 4170.

j.

$$S_{fpga} = \frac{T_{seq}}{T_{fpga}} = 34736.1$$

k.

Large invocation of the accelerators will largely reduce N. N = 411.

6. Reflection

a.

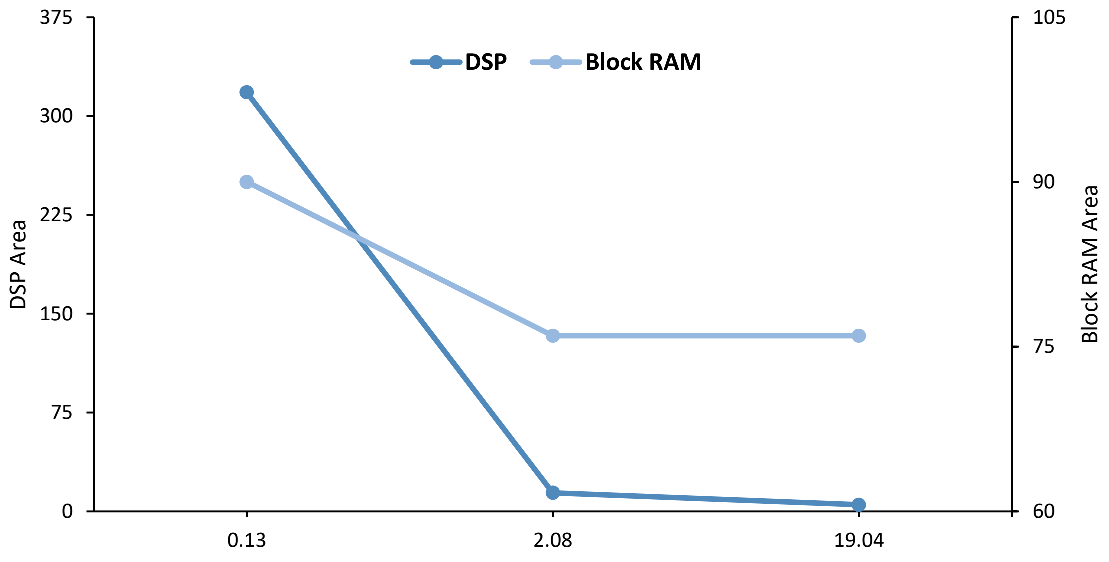
Specific optimization sequence:

- Identify bottleneck (Part 1)
- Loop unrolling (Part 2)
- Loop pipelining (Part 3)
- Array partitioning (Part 3)

b.

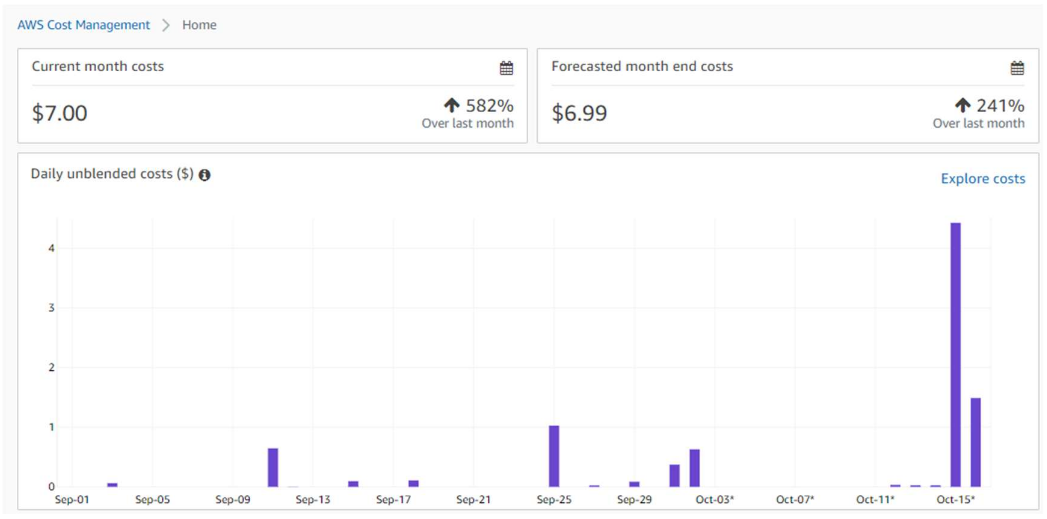
Area time plot for the following designs with a curve for DSPs and Block RAMs:

- Resource estimates from Baseline
- Resource estimates for complete unrolled loop with 20ns clock
- Resource estimates from Pipeline and Partitioned Arrays



c.

Total credits used thus far: $10



Current month costs

$7.00 ↑ 582% Over last month

Forecasted month end costs

$6.99 ↑ 241% Over last month

Daily unblended costs ($)