Tiled Matrix Multiplication using Systolic Array

By Krithika Kandasamy, Ram Jaswanth and Disha Gulur

Georgia Tech

Problem Statement

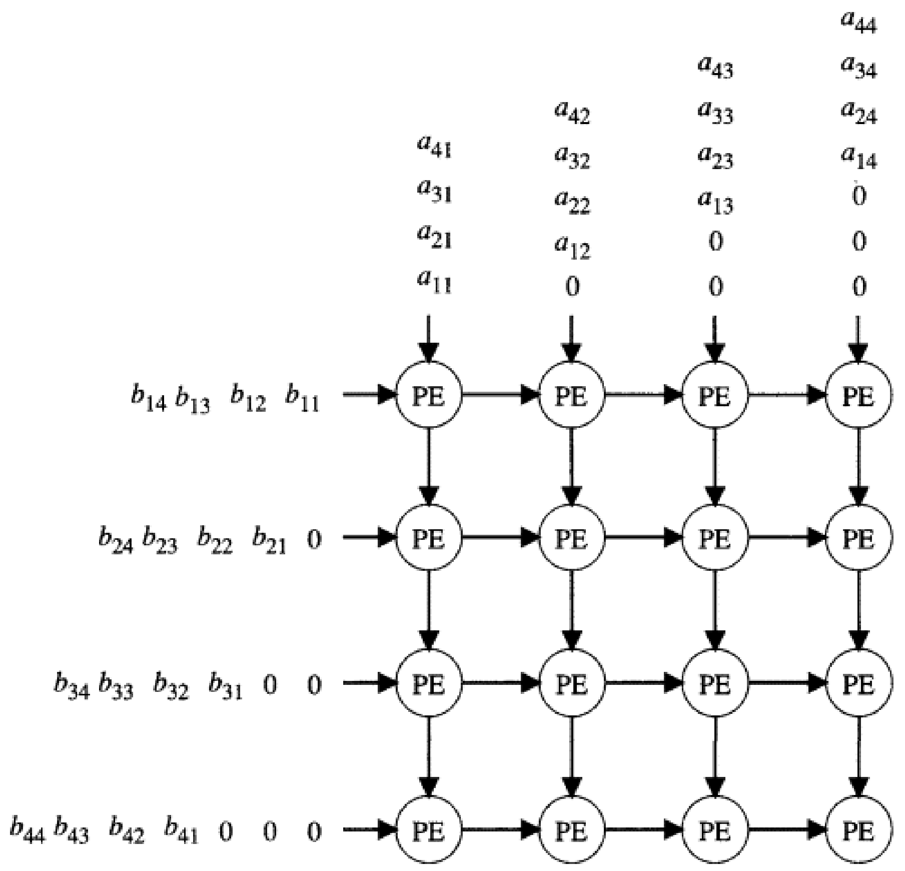
- The sequential implementation of matrix multiplication is a very time-consuming task especially for large matrices.

- Systolic Array implementation of matrix multiplication is one approach to gain computational speed-up.

- Our project aims at accelerating matrix multiplication operation of large input matrices – 512x512 matrix multiplication.

Systolic Array Design

Design on paper



a_{44} a_{34} a_{24} a_{14} 0 0 0
a_{43} a_{33} a_{23} a_{13} 0 0
a_{42} a_{32} a_{22} a_{12} 0
a_{41} a_{31} a_{21} a_{11}
b_{14} b_{13} b_{12} b_{11}
b_{24} b_{23} b_{22} b_{21} 0
b_{34} b_{33} b_{32} b_{31} 0 0
b_{44} b_{43} b_{42} b_{41} 0 0 0
PE

Reference- Evolutionary Mapping Techniques for Systolic Computing System

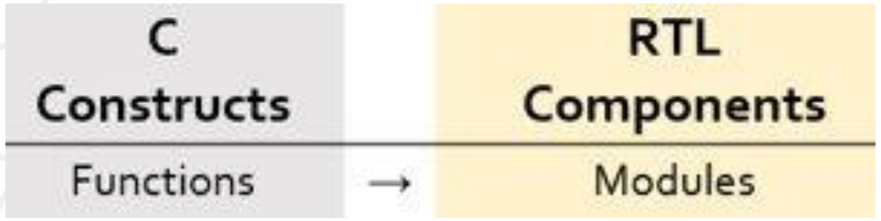
HLS Implementation?

- How to implement the processing elements?
- How is data transferred between the processing elements?
- How to overlap the read, write and compute operations of the processing element?

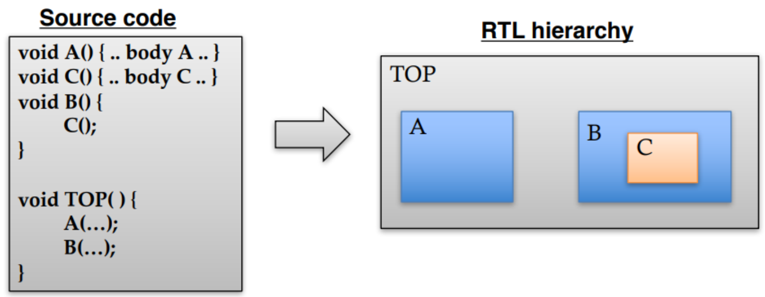
HLS Implementation

How to implement the processing elements?

- C functions translate to modules.



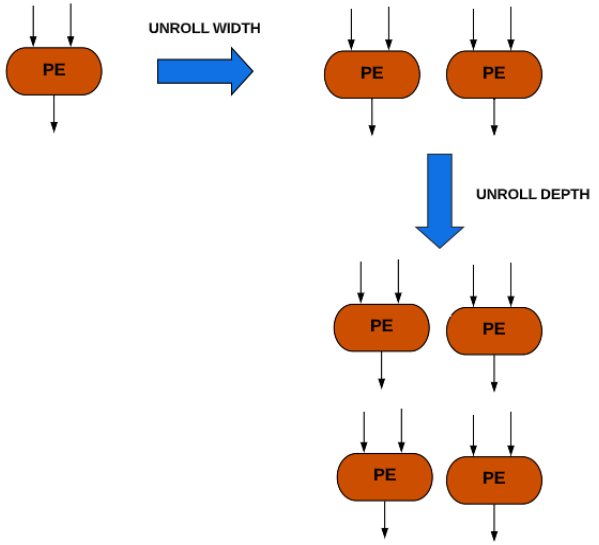
C Constructs		RTL Components
Functions	→	Modules



Source code

```
void A() { .. body A .. }
void C() { .. body C .. }
void B() {
        C();
}

void TOP( ) {
        A(...);
        B(...);
}
```

- Unrolling is scaling!

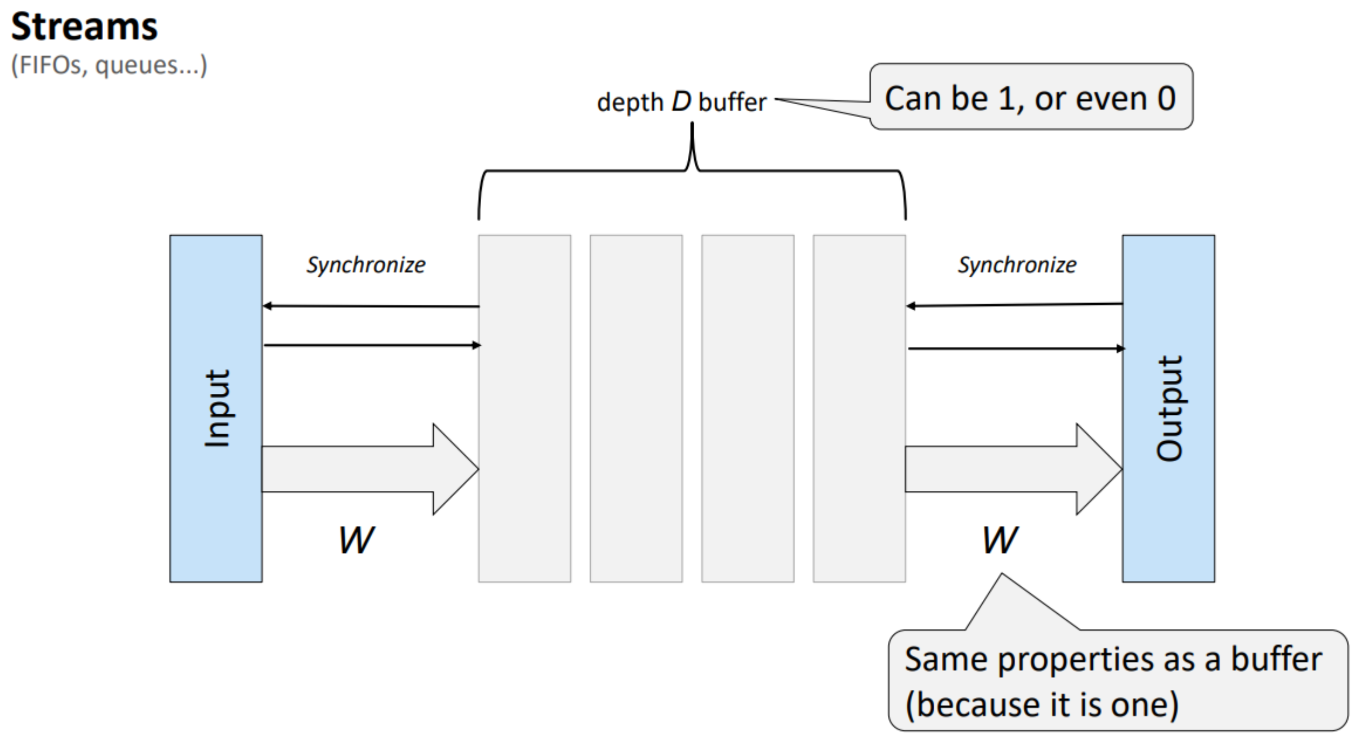


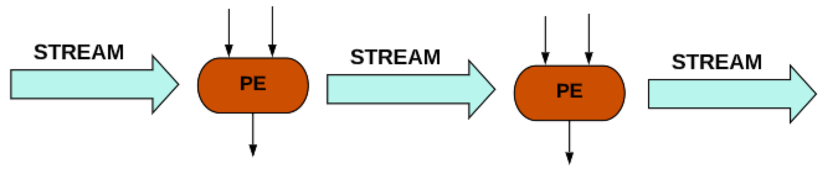
Has more meaning in FPGA than instruction-based architectures – Every "instruction" is a separate hardware.

HLS Implementation

Data transfer between the processing elements

- Synchronous interface
- **Asynchronous interface**



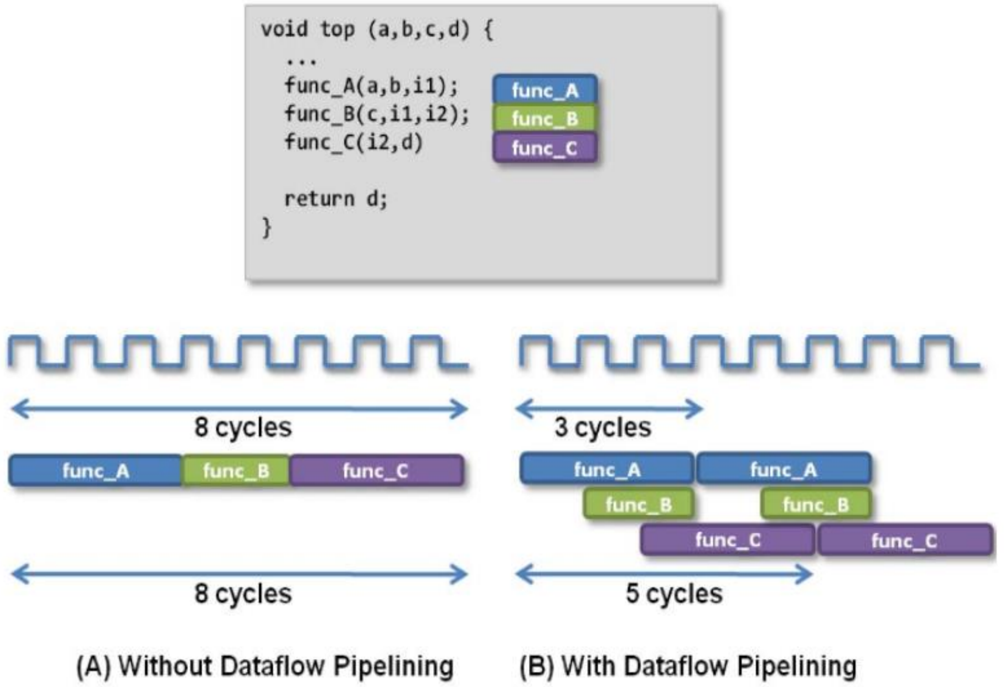


Data streaming between the PEs

HLS Implementation

How to overlap execution in the processing element?

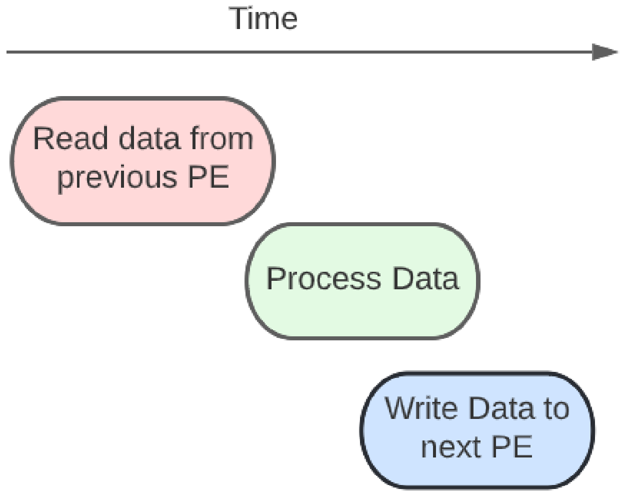
- Data flow to the rescue!



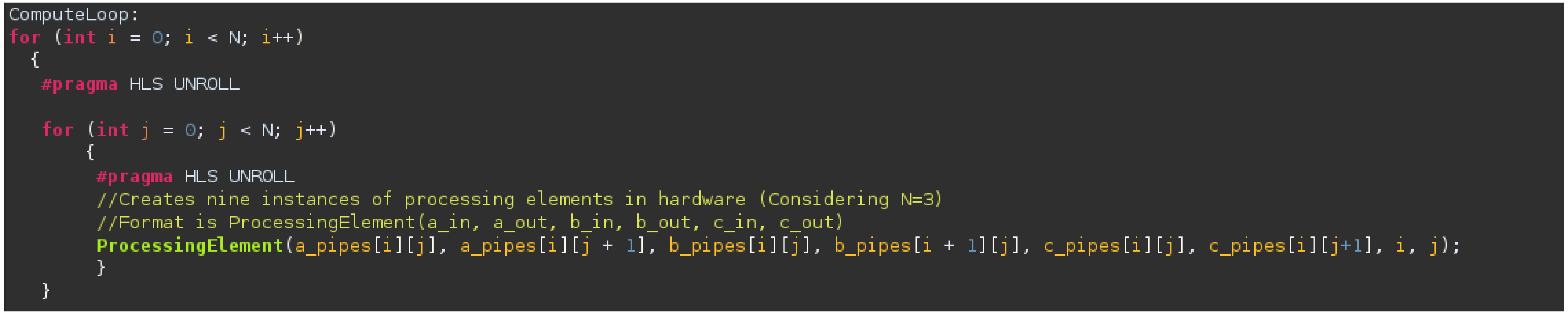
```
void top (a,b,c,d) {
  ...
  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

  return d;
}
```

Pipelining in the function level

Inside the processing element

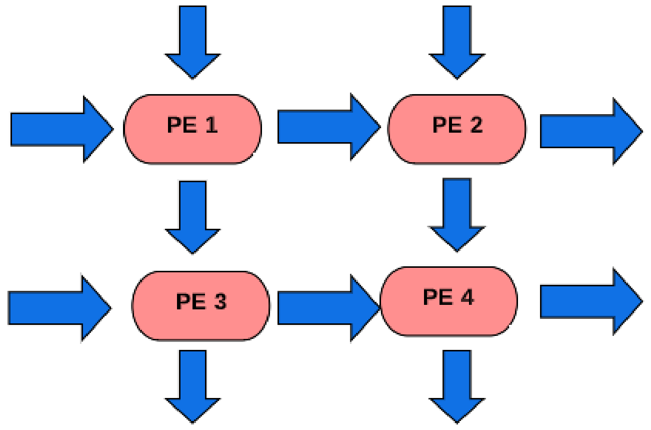


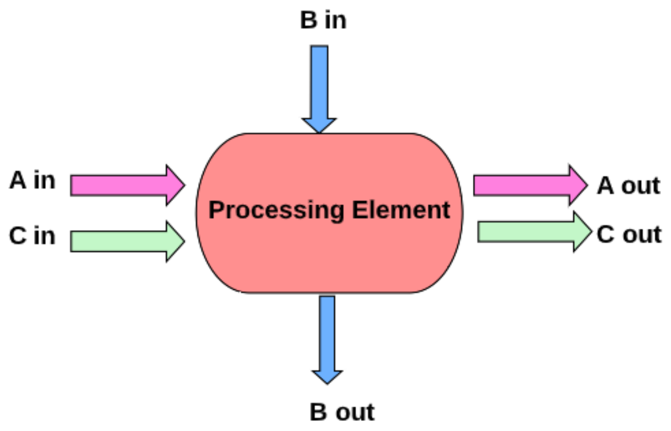
HLS Implementation



```
ComputeLoop:
for (int i = 0; i < N; i++)
  {
   #pragma HLS UNROLL

   for (int j = 0; j < N; j++)
       {
        #pragma HLS UNROLL
        //Creates nine instances of processing elements in hardware (Considering N=3)
        //Format is ProcessingElement(a_in, a_out, b_in, b_out, c_in, c_out)
        ProcessingElement(a_pipes[i][j], a_pipes[i][j + 1], b_pipes[i][j], b_pipes[i + 1][j], c_pipes[i][j], c_pipes[i][j+1], i, j);
        }
   }
```

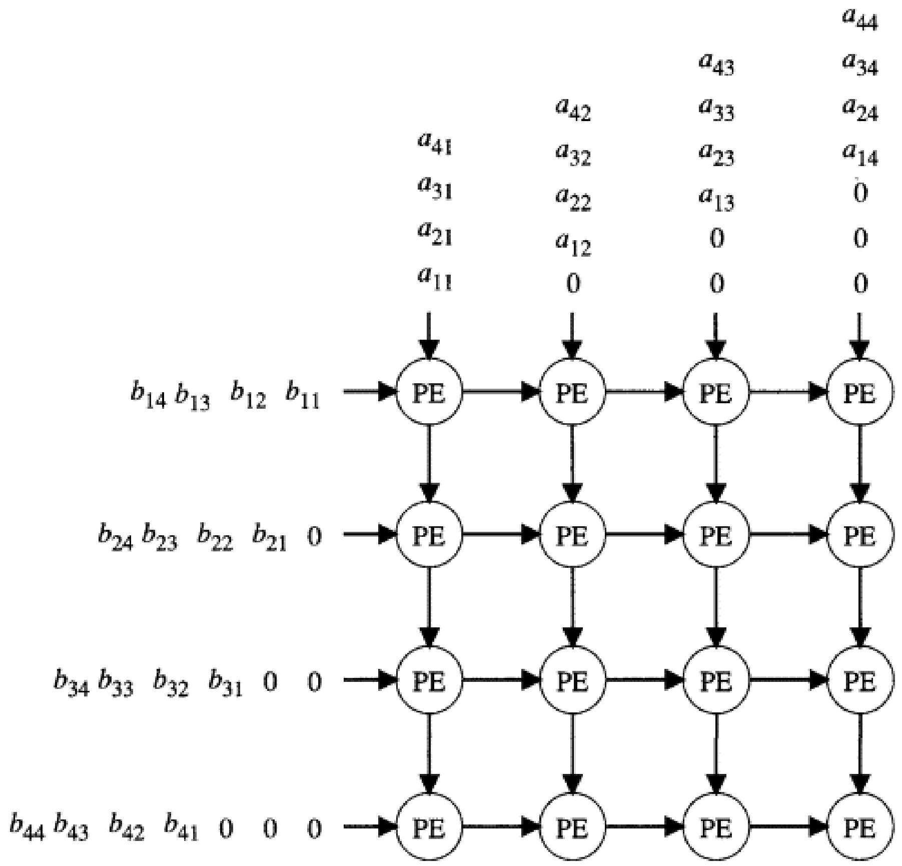
- Each PE is a function.
- 2D array of streams for data transfer.





Systolic Array Design – Putting it all together

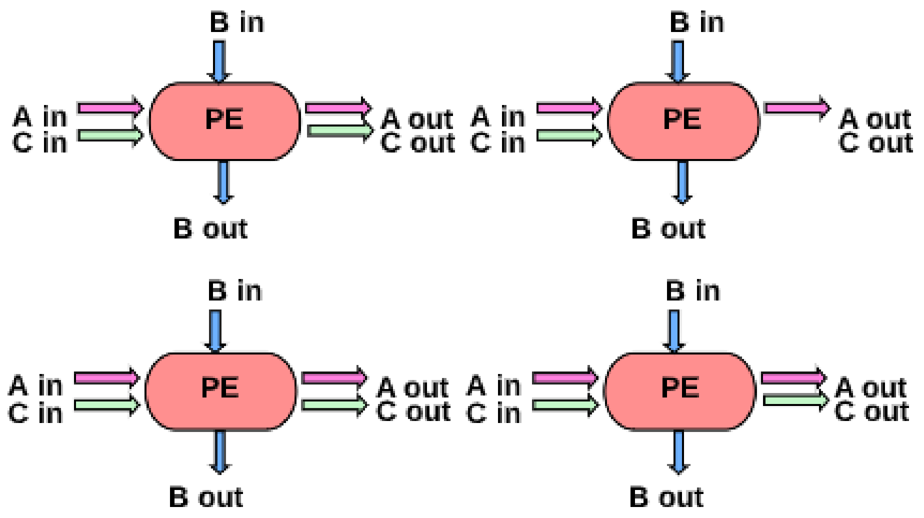
Design on paper (4x4 systolic array)



a_{44}
a_{43} a_{34}
a_{42} a_{33} a_{24}
a_{41} a_{32} a_{23} a_{14}
a_{31} a_{22} a_{13} 0
a_{21} a_{12} 0 0
a_{11} 0 0 0
b_{14} b_{13} b_{12} b_{11}
b_{24} b_{23} b_{22} b_{21} 0
b_{34} b_{33} b_{32} b_{31} 0 0
b_{44} b_{43} b_{42} b_{41} 0 0 0
PE

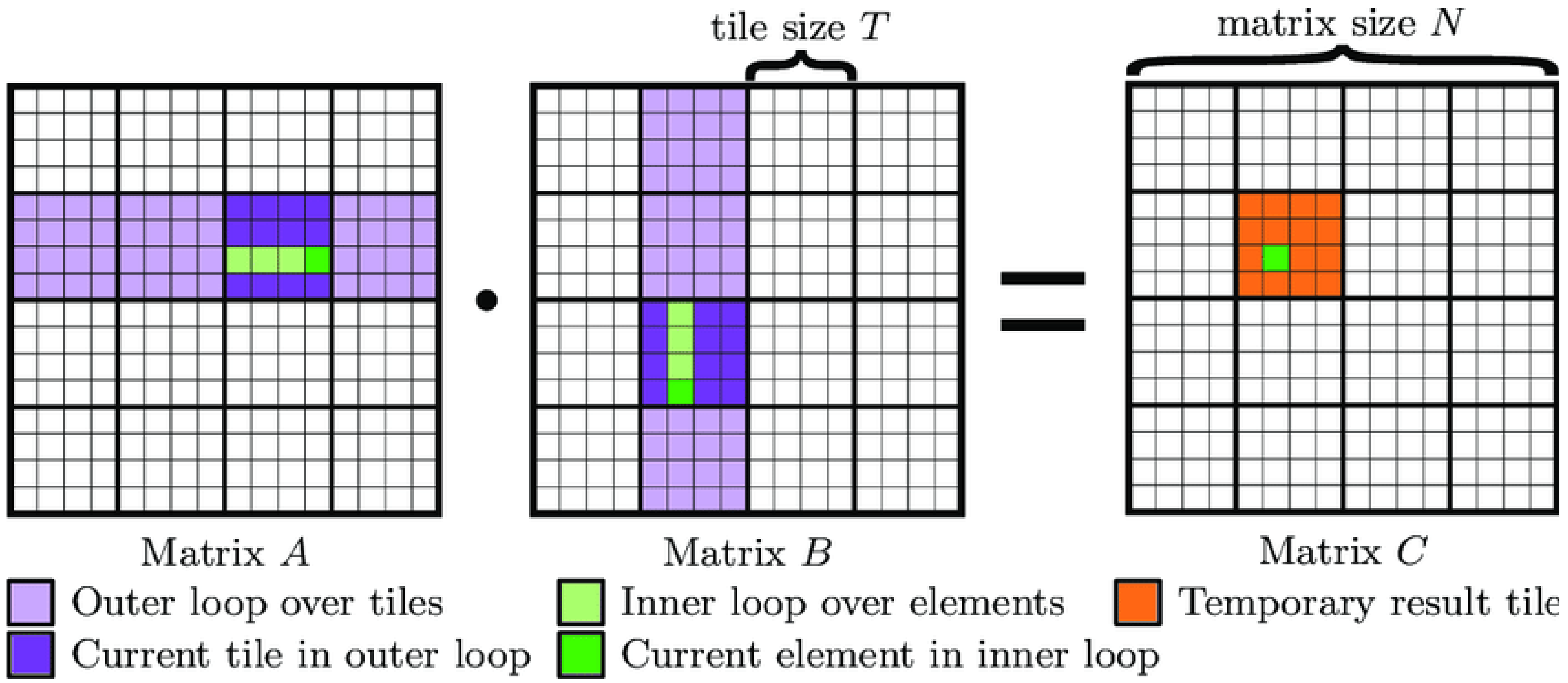
Reference- Evolutionary Mapping Techniques for Systolic Computing System

HLS Implementation (2x2 systolic array)

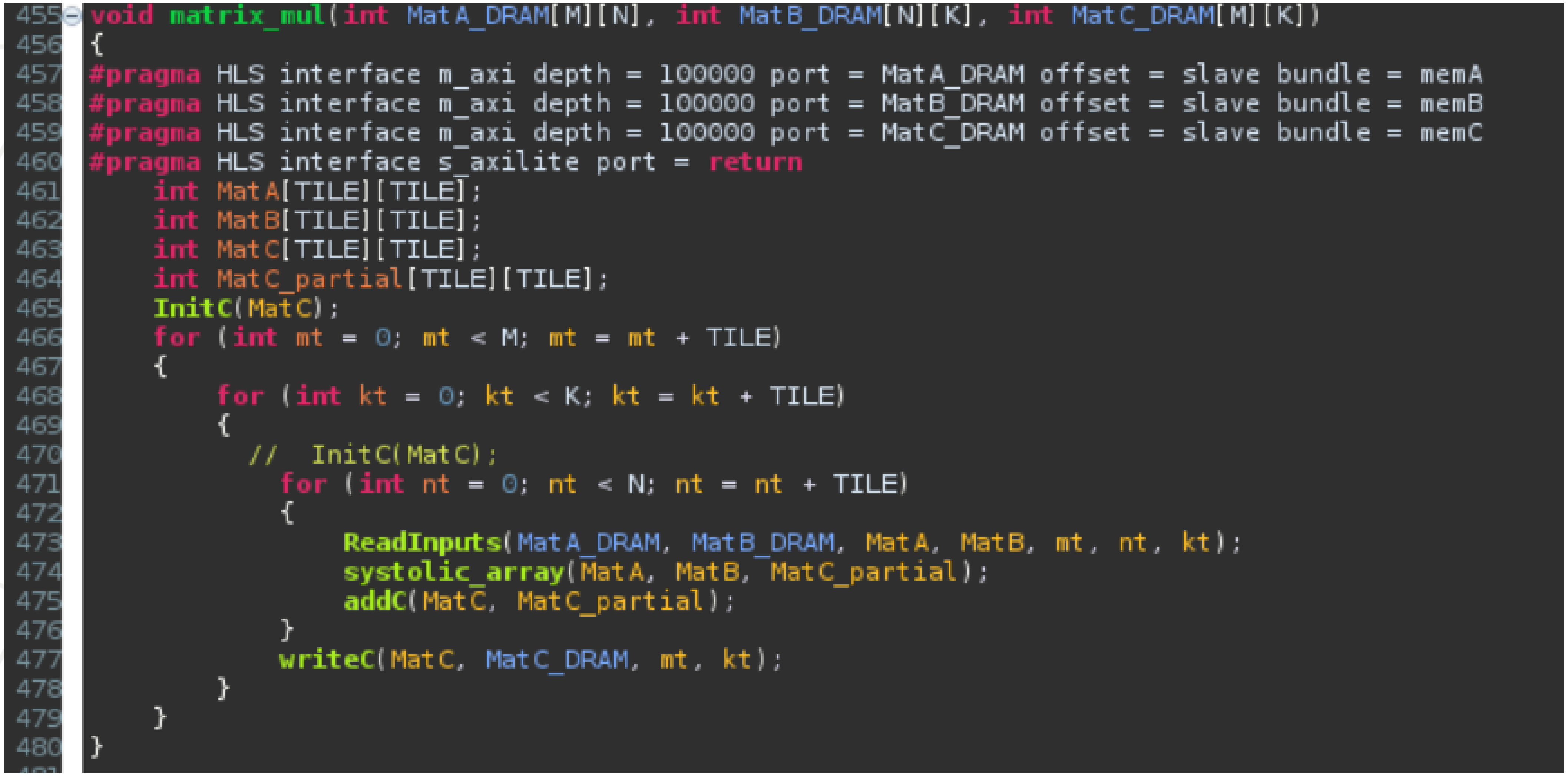


Tiling Matrix Multiplication

- Output stationary!



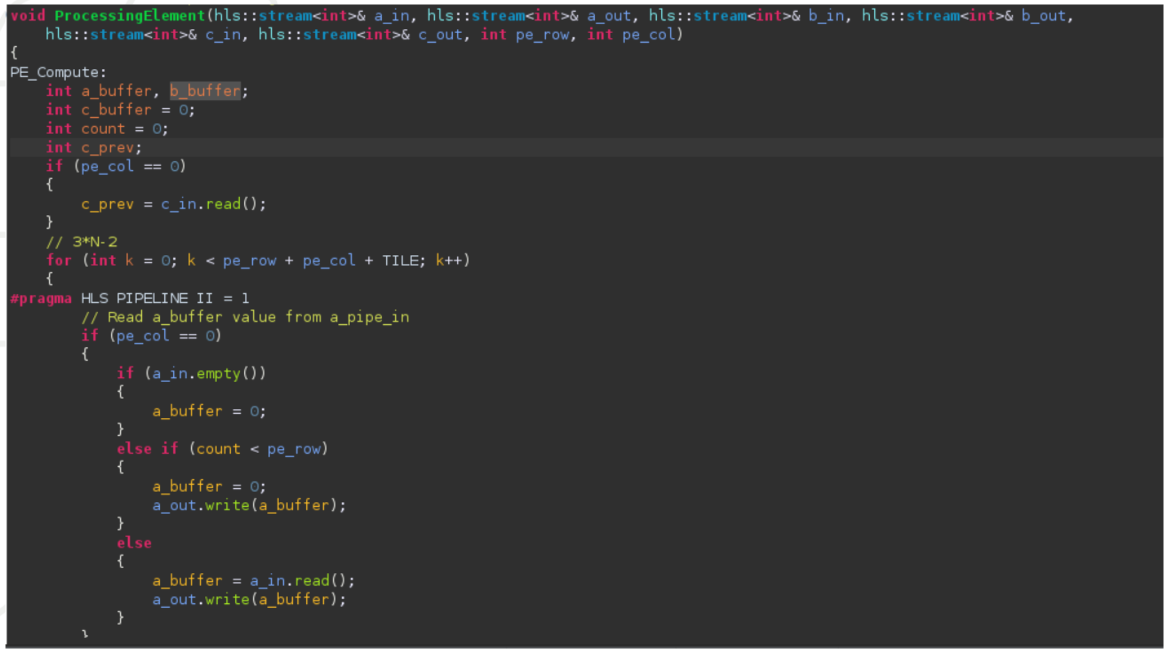
BASELINE



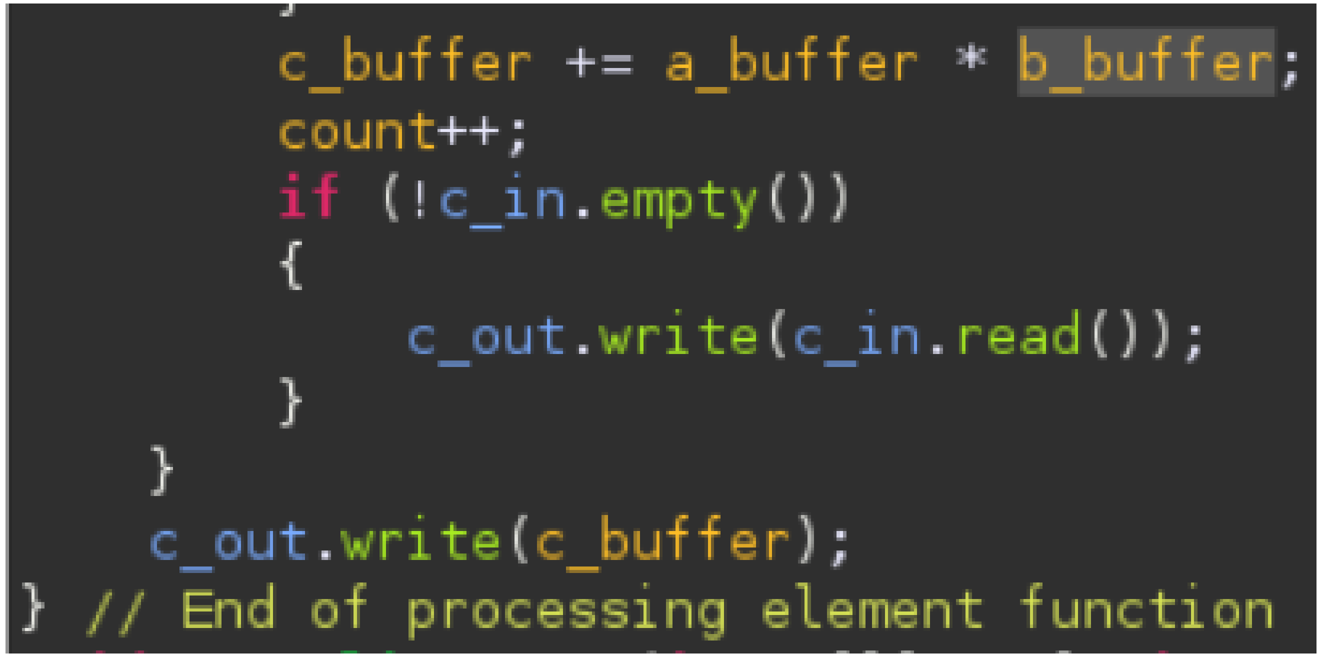
```
void matrix_mul(int MatA_DRAM[M][N], int MatB_DRAM[N][K], int MatC_DRAM[M][K])
{
#pragma HLS interface m_axi depth = 100000 port = MatA_DRAM offset = slave bundle = memA
#pragma HLS interface m_axi depth = 100000 port = MatB_DRAM offset = slave bundle = memB
#pragma HLS interface m_axi depth = 100000 port = MatC_DRAM offset = slave bundle = memC
#pragma HLS interface s_axilite port = return
    int MatA[TILE][TILE];
    int MatB[TILE][TILE];
    int MatC[TILE][TILE];
    int MatC_partial[TILE][TILE];
    InitC(MatC);
    for (int mt = 0; mt < M; mt = mt + TILE)
    {
        for (int kt = 0; kt < K; kt = kt + TILE)
        {
          //  InitC(MatC);
            for (int nt = 0; nt < N; nt = nt + TILE)
            {
                ReadInputs(MatA_DRAM, MatB_DRAM, MatA, MatB, mt, nt, kt);
                systolic_array(MatA, MatB, MatC_partial);
                addC(MatC, MatC_partial);
            }
            writeC(MatC, MatC_DRAM, mt, kt);
        }
    }
}
```

LATENCY --------> 1.8 seconds

Optimization – II
Pipeline



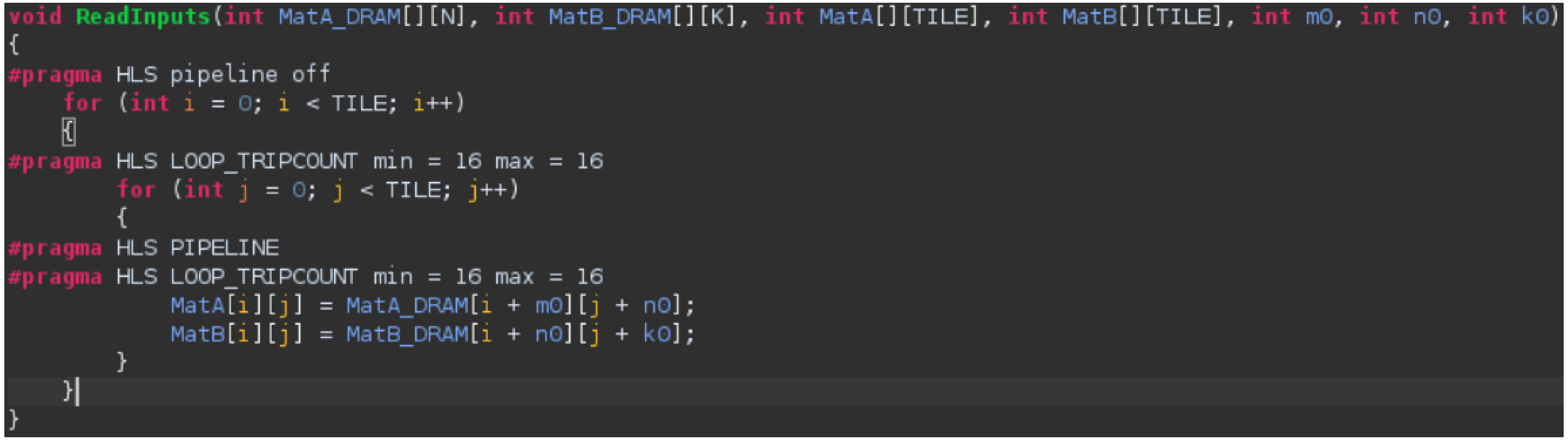
```cpp
void ProcessingElement(hls::stream<int>& a_in, hls::stream<int>& a_out, hls::stream<int>& b_in, hls::stream<int>& b_out,
    hls::stream<int>& c_in, hls::stream<int>& c_out, int pe_row, int pe_col)
{
PE_Compute:
    int a_buffer, b_buffer;
    int c_buffer = 0;
    int count = 0;
    int c_prev;
    if (pe_col == 0)
    {
        c_prev = c_in.read();
    }
    // 3*N-2
    for (int k = 0; k < pe_row + pe_col + TILE; k++)
    {
#pragma HLS PIPELINE II = 1
        // Read a_buffer value from a_pipe_in
        if (pe_col == 0)
        {
            if (a_in.empty())
            {
                a_buffer = 0;
            }
            else if (count < pe_row)
            {
                a_buffer = 0;
                a_out.write(a_buffer);
            }
            else
            {
                a_buffer = a_in.read();
                a_out.write(a_buffer);
            }
        }
```



```cpp
            c_buffer += a_buffer * b_buffer;
            count++;
            if (!c_in.empty())
            {
                c_out.write(c_in.read());
            }
        }
        c_out.write(c_buffer);
} // End of processing element function
```

- Each PE element performs a matrix multiplication and accumulation operation.
- Within the PE these operations are pipelined across different cycles.

Optimization – II
Pipeline + Loop Fusion

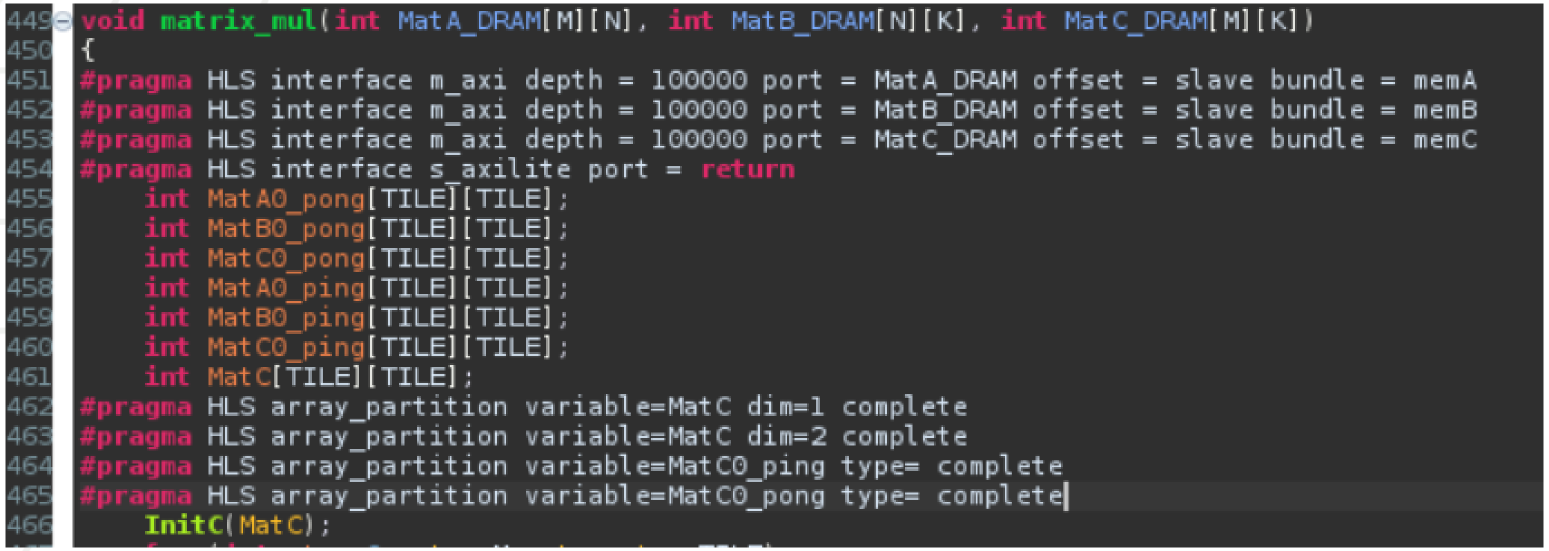


```
void ReadInputs(int MatA_DRAM[][N], int MatB_DRAM[][K], int MatA[][TILE], int MatB[][TILE], int m0, int n0, int k0)
{
#pragma HLS pipeline off
    for (int i = 0; i < TILE; i++)
    {
#pragma HLS LOOP_TRIPCOUNT min = 16 max = 16
        for (int j = 0; j < TILE; j++)
        {
#pragma HLS PIPELINE
#pragma HLS LOOP_TRIPCOUNT min = 16 max = 16
            MatA[i][j] = MatA_DRAM[i + m0][j + n0];
            MatB[i][j] = MatB_DRAM[i + n0][j + k0];
        }
    }
}
```

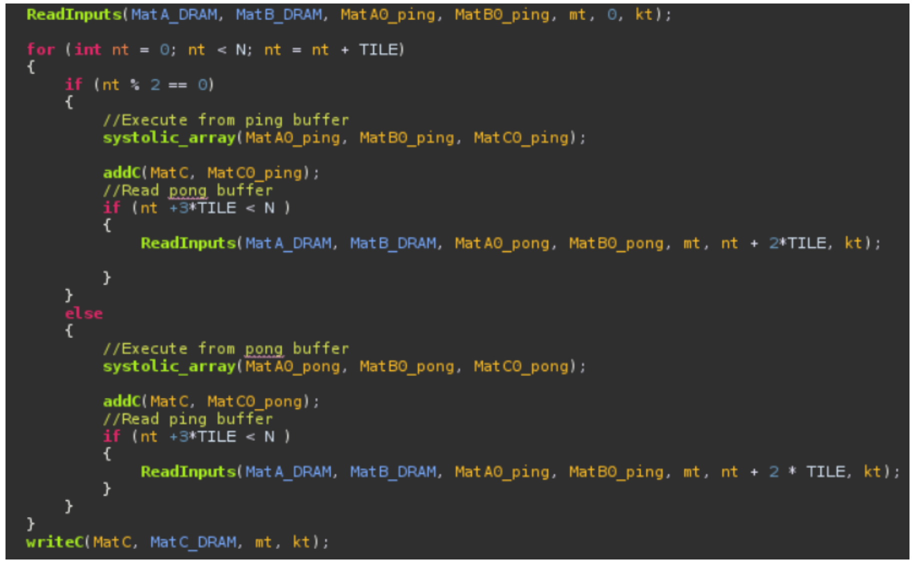
- Pipelining is also performed during memory access operations
- Since we use 512 x 512 matrices both the arrays can be accessed in the same loop

LATENCY --------> 0.348 sec

Optimization – III
Ping Pong Buffers



```
void matrix_mul(int MatA_DRAM[M][N], int MatB_DRAM[N][K], int MatC_DRAM[M][K])
{
#pragma HLS interface m_axi depth = 100000 port = MatA_DRAM offset = slave bundle = memA
#pragma HLS interface m_axi depth = 100000 port = MatB_DRAM offset = slave bundle = memB
#pragma HLS interface m_axi depth = 100000 port = MatC_DRAM offset = slave bundle = memC
#pragma HLS interface s_axilite port = return
    int MatA0_pong[TILE][TILE];
    int MatB0_pong[TILE][TILE];
    int MatC0_pong[TILE][TILE];
    int MatA0_ping[TILE][TILE];
    int MatB0_ping[TILE][TILE];
    int MatC0_ping[TILE][TILE];
    int MatC[TILE][TILE];
#pragma HLS array_partition variable=MatC dim=1 complete
#pragma HLS array_partition variable=MatC dim=2 complete
#pragma HLS array_partition variable=MatC0_ping type= complete
#pragma HLS array_partition variable=MatC0_pong type= complete
    InitC(MatC);
```

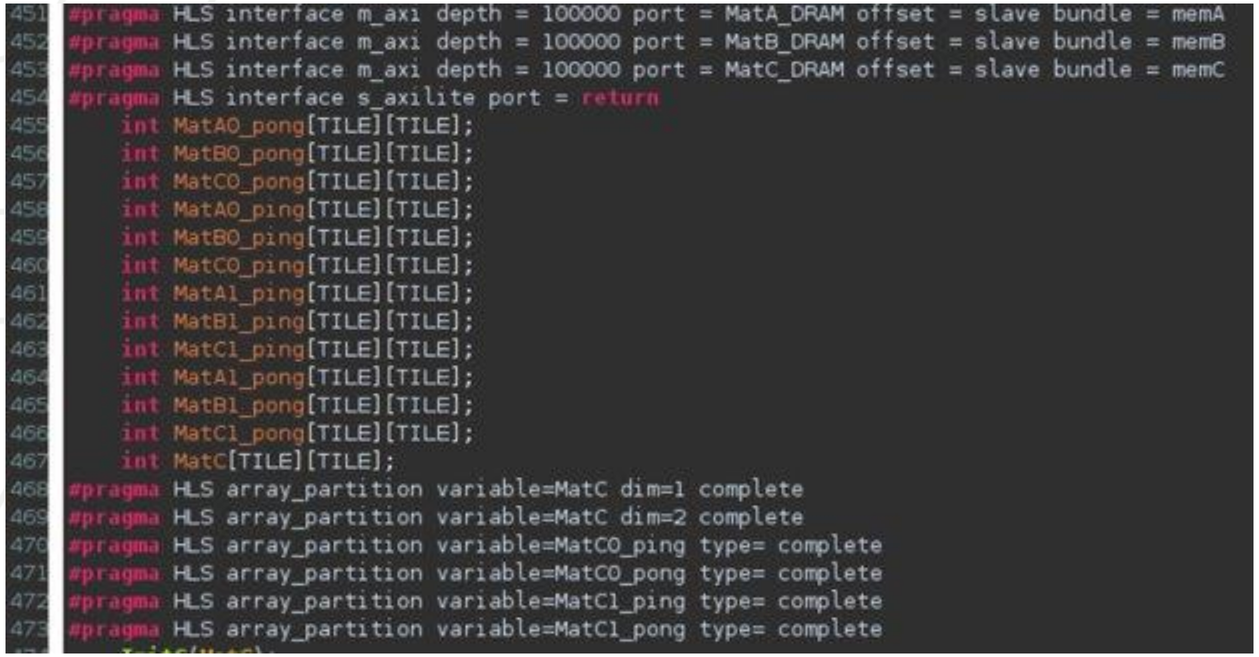


```
ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_ping, MatB0_ping, mt, 0, kt);

for (int nt = 0; nt < N; nt = nt + TILE)
{
    if (nt % 2 == 0)
    {
        //Execute from ping buffer
        systolic_array(MatA0_ping, MatB0_ping, MatC0_ping);

        addC(MatC, MatC0_ping);
        //Read pong buffer
        if (nt +3*TILE < N )
        {
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_pong, MatB0_pong, mt, nt + 2*TILE, kt);

        }
    }
    else
    {
        //Execute from pong buffer
        systolic_array(MatA0_pong, MatB0_pong, MatC0_pong);

        addC(MatC, MatC0_pong);
        //Read ping buffer
        if (nt +3*TILE < N )
        {
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_ping, MatB0_ping, mt, nt + 2 * TILE, kt);
        }
    }
}
writeC(MatC, MatC_DRAM, mt, kt);
```

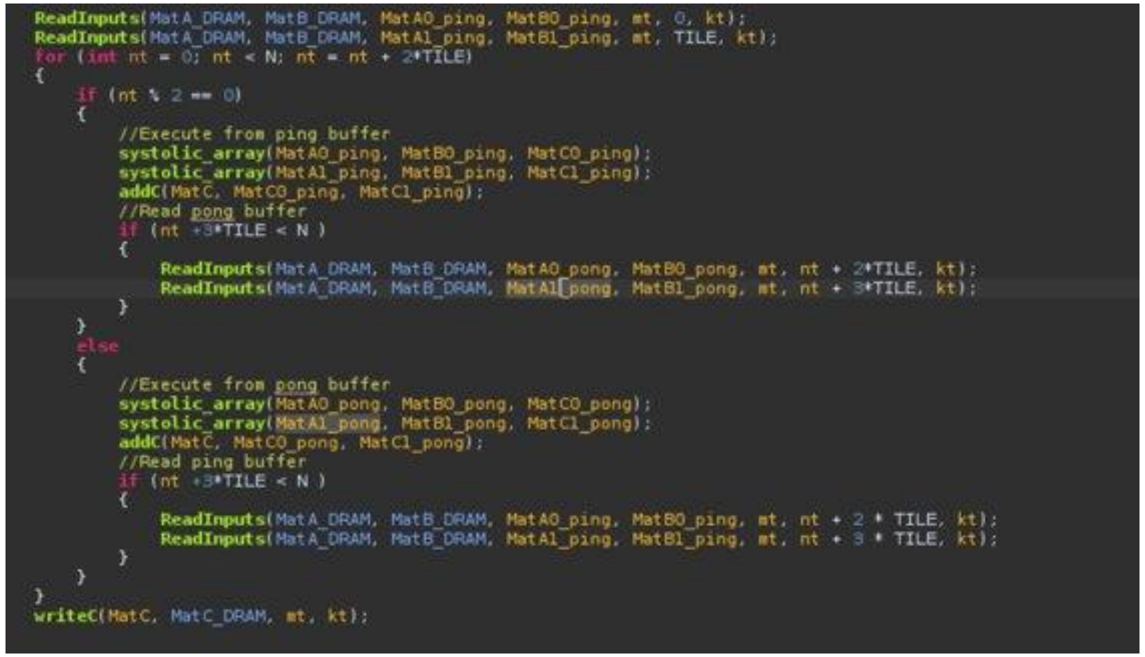
- Ping Pong Buffers are used to overlap read operation(for the next tile) and systolic array (current tile) operation.

LATENCY --------> 0.269 sec

Optimization – IV
2 Parallel Systolic Arrays



```
#pragma HLS interface m_axi depth = 100000 port = MatA_DRAM offset = slave bundle = memA
#pragma HLS interface m_axi depth = 100000 port = MatB_DRAM offset = slave bundle = memB
#pragma HLS interface m_axi depth = 100000 port = MatC_DRAM offset = slave bundle = memC
#pragma HLS interface s_axilite port = return
    int MatA0_pong[TILE][TILE];
    int MatB0_pong[TILE][TILE];
    int MatC0_pong[TILE][TILE];
    int MatA0_ping[TILE][TILE];
    int MatB0_ping[TILE][TILE];
    int MatC0_ping[TILE][TILE];
    int MatA1_ping[TILE][TILE];
    int MatB1_ping[TILE][TILE];
    int MatC1_ping[TILE][TILE];
    int MatA1_pong[TILE][TILE];
    int MatB1_pong[TILE][TILE];
    int MatC1_pong[TILE][TILE];
    int MatC[TILE][TILE];
#pragma HLS array_partition variable=MatC dim=1 complete
#pragma HLS array_partition variable=MatC dim=2 complete
#pragma HLS array_partition variable=MatC0_ping type= complete
#pragma HLS array_partition variable=MatC0_pong type= complete
#pragma HLS array_partition variable=MatC1_ping type= complete
#pragma HLS array_partition variable=MatC1_pong type= complete
```

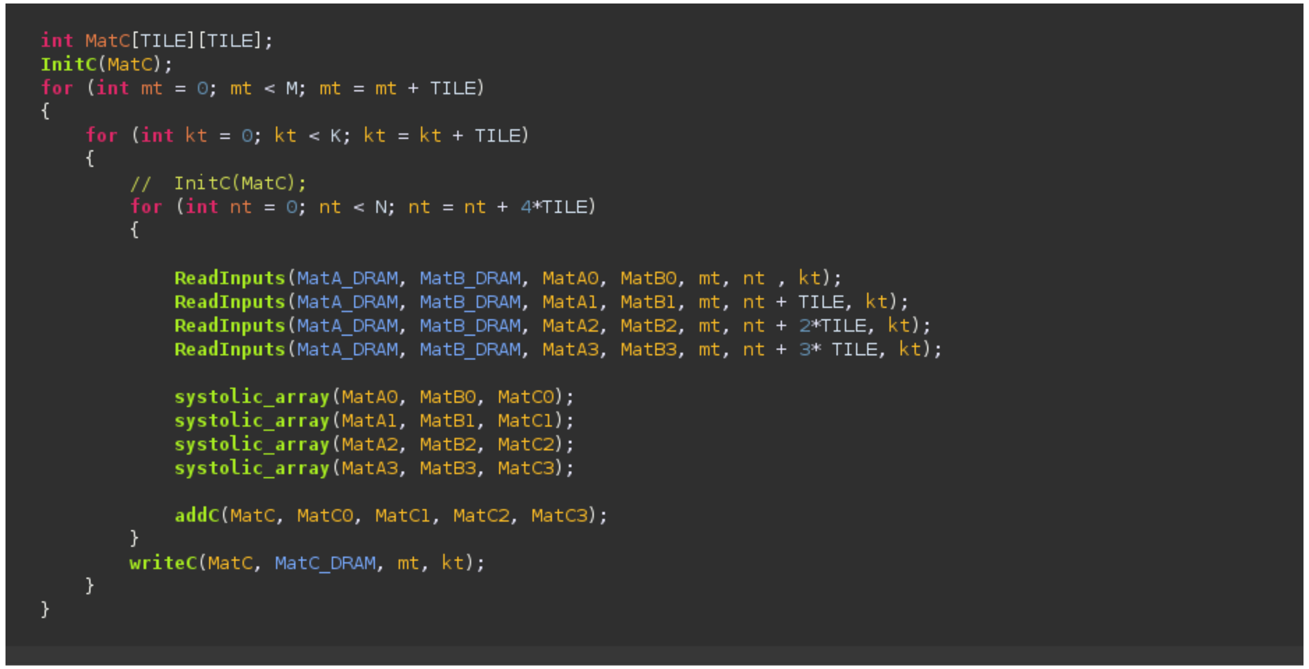


```
ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_ping, MatB0_ping, mt, 0, kt);
ReadInputs(MatA_DRAM, MatB_DRAM, MatA1_ping, MatB1_ping, mt, TILE, kt);
for (int nt = 0; nt < N; nt = nt + 2*TILE)
{
    if (nt % 2 == 0)
    {
        //Execute from ping buffer
        systolic_array(MatA0_ping, MatB0_ping, MatC0_ping);
        systolic_array(MatA1_ping, MatB1_ping, MatC1_ping);
        addC(MatC, MatC0_ping, MatC1_ping);
        //Read pong buffer
        if (nt +3*TILE < N )
        {
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_pong, MatB0_pong, mt, nt + 2*TILE, kt);
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA1_pong, MatB1_pong, mt, nt + 3*TILE, kt);
        }
    }
    else
    {
        //Execute from pong buffer
        systolic_array(MatA0_pong, MatB0_pong, MatC0_pong);
        systolic_array(MatA1_pong, MatB1_pong, MatC1_pong);
        addC(MatC, MatC0_pong, MatC1_pong);
        //Read ping buffer
        if (nt +3*TILE < N )
        {
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA0_ping, MatB0_ping, mt, nt + 2 * TILE, kt);
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA1_ping, MatB1_ping, mt, nt + 3 * TILE, kt);
        }
    }
}
writeC(MatC, MatC_DRAM, mt, kt);
```

- Start two systolic arrays at the same time.
- Integrate Ping Pong buffer operation.

LATENCY --------> 0.222 sec (2 Systolic Arrays)

Optimization – V
4 Parallel Systolic Arrays



```
int MatC[TILE][TILE];
InitC(MatC);
for (int mt = 0; mt < M; mt = mt + TILE)
{
    for (int kt = 0; kt < K; kt = kt + TILE)
    {
        //  InitC(MatC);
        for (int nt = 0; nt < N; nt = nt + 4*TILE)
        {

            ReadInputs(MatA_DRAM, MatB_DRAM, MatA0, MatB0, mt, nt , kt);
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA1, MatB1, mt, nt + TILE, kt);
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA2, MatB2, mt, nt + 2*TILE, kt);
            ReadInputs(MatA_DRAM, MatB_DRAM, MatA3, MatB3, mt, nt + 3* TILE, kt);

            systolic_array(MatA0, MatB0, MatC0);
            systolic_array(MatA1, MatB1, MatC1);
            systolic_array(MatA2, MatB2, MatC2);
            systolic_array(MatA3, MatB3, MatC3);

            addC(MatC, MatC0, MatC1, MatC2, MatC3);
        }
        writeC(MatC, MatC_DRAM, mt, kt);
    }
}
```

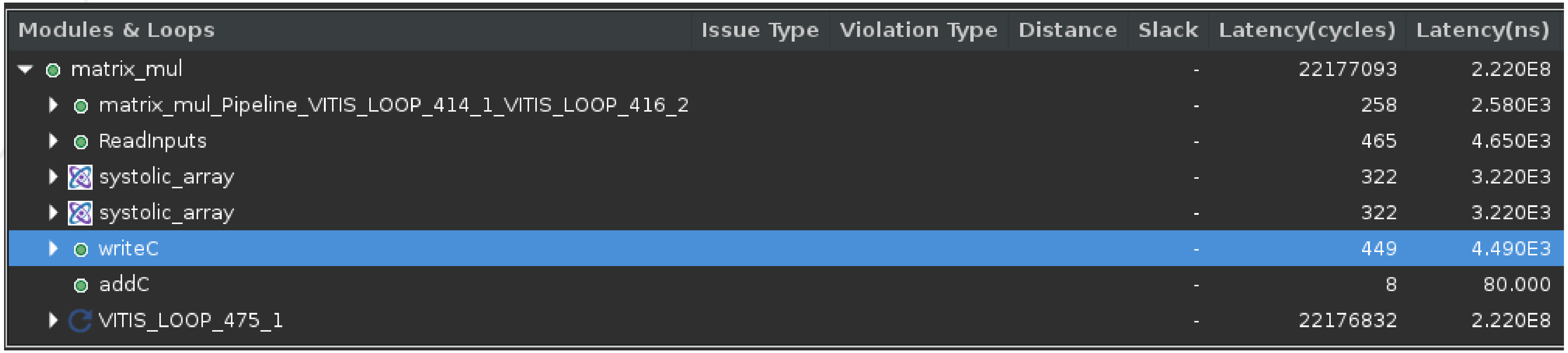
- Start 4 systolic_arrays at the same time.
- Removed the Ping Pong Buffer implementation as the resource utilization is above 100%

LATENCY --------> 0.189 sec (4 Systolic Arrays)

Georgia Tech

Results

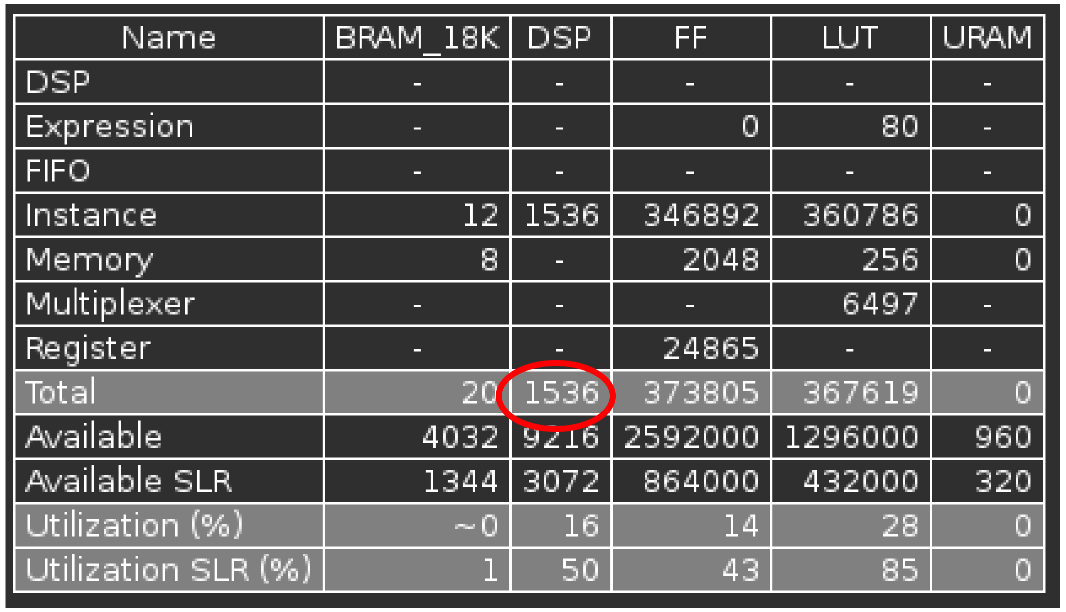
Latency of the design



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)
matrix_mul				-	22177093	2.220E8
matrix_mul_Pipeline_VITIS_LOOP_414_1_VITIS_LOOP_416_2				-	258	2.580E3
ReadInputs				-	465	4.650E3
systolic_array				-	322	3.220E3
systolic_array				-	322	3.220E3
writeC				-	449	4.490E3
addC				-	8	80.000
VITIS_LOOP_475_1				-	22176832	2.220E8

➢ Memory access is the bottleneck!

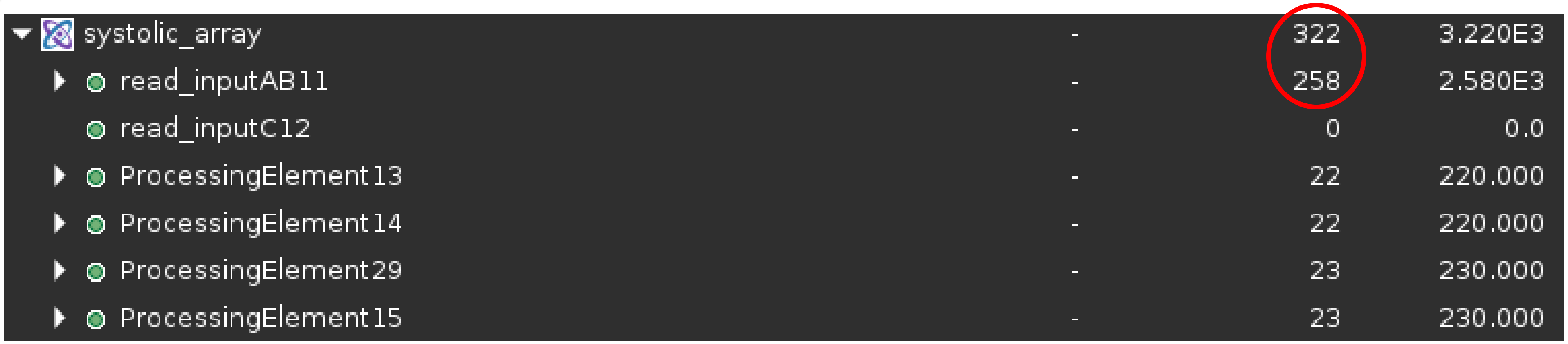
Resource Utilization of the design



Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	80	-
FIFO	-	-	-	-	-
Instance	12	1536	346892	360786	0
Memory	8	-	2048	256	0
Multiplexer	-	-	-	6497	-
Register	-	-	24865	-	-
Total	20	1536	373805	367619	0
Available	4032	9216	2592000	1296000	960
Available SLR	1344	3072	864000	432000	320
Utilization (%)	~0	16	14	28	0
Utilization SLR (%)	1	50	43	85	0

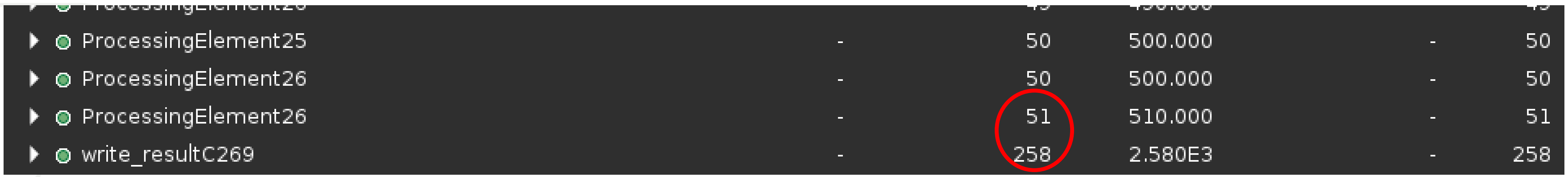
Results

PEs in the initial columns



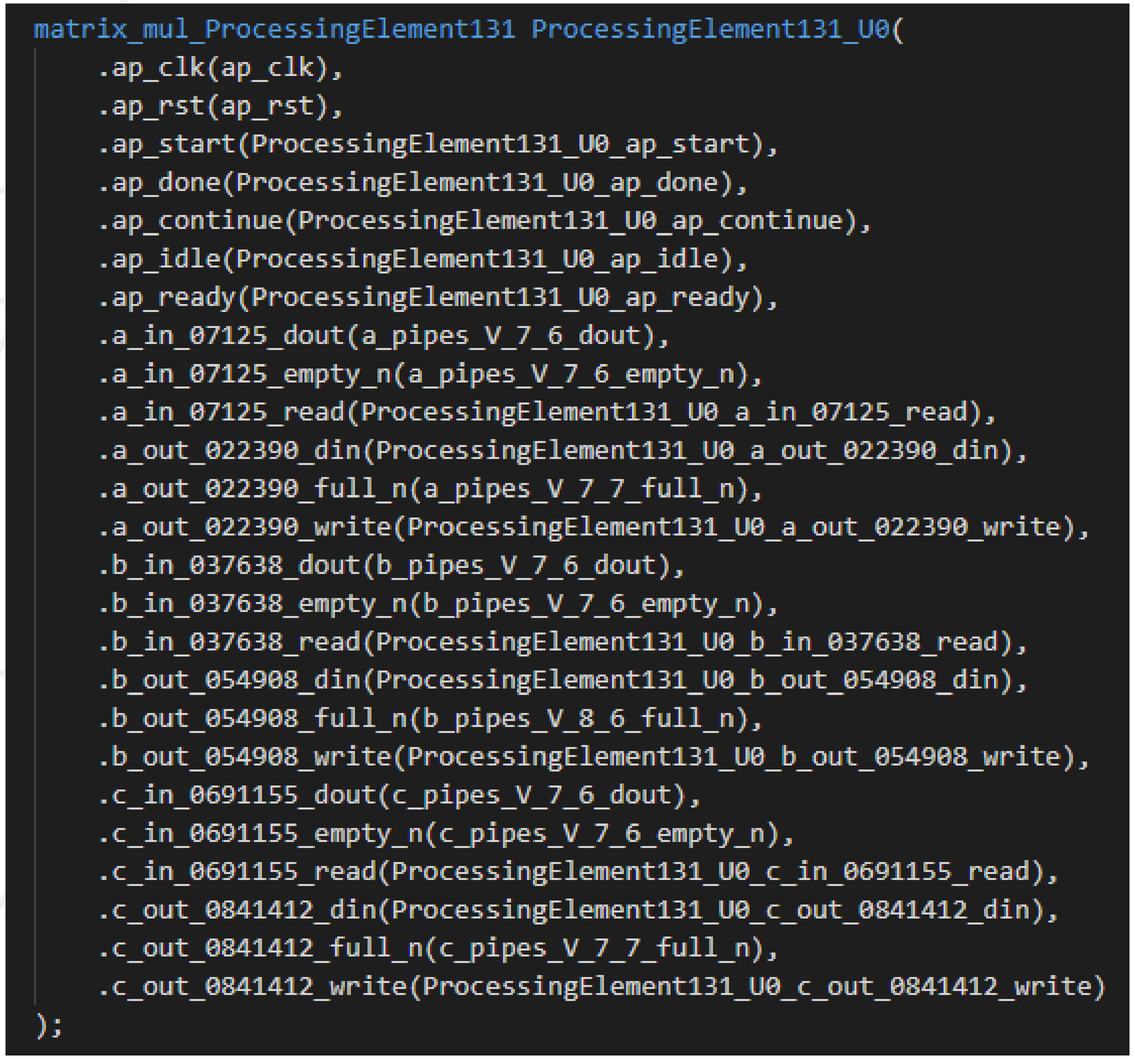
```
▾ systolic_array                    -     322    3.220E3
    ▸ read_inputAB11                -     258    2.580E3
      read_inputC12                 -       0        0.0
    ▸ ProcessingElement13           -      22    220.000
    ▸ ProcessingElement14           -      22    220.000
    ▸ ProcessingElement29           -      23    230.000
    ▸ ProcessingElement15           -      23    230.000
```

PEs in the final rows and columns



```
▸ ProcessingElement25      -     50    500.000      -     50
▸ ProcessingElement26      -     50    500.000      -     50
▸ ProcessingElement26      -     51    510.000      -     51
▸ write_resultC269         -    258    2.580E3      -    258
```

Results

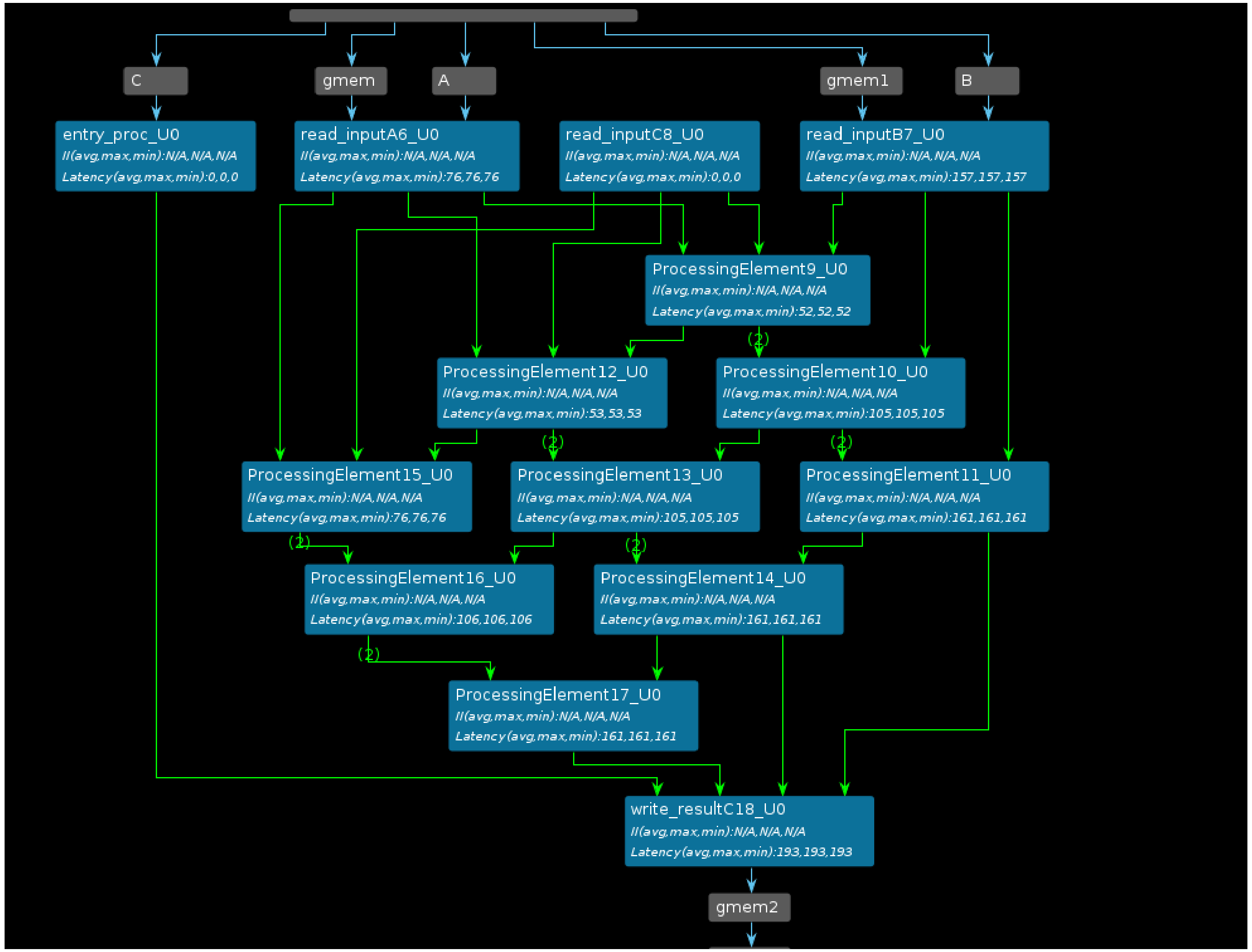


```
matrix_mul_ProcessingElement131 ProcessingElement131_U0(
    .ap_clk(ap_clk),
    .ap_rst(ap_rst),
    .ap_start(ProcessingElement131_U0_ap_start),
    .ap_done(ProcessingElement131_U0_ap_done),
    .ap_continue(ProcessingElement131_U0_ap_continue),
    .ap_idle(ProcessingElement131_U0_ap_idle),
    .ap_ready(ProcessingElement131_U0_ap_ready),
    .a_in_07125_dout(a_pipes_V_7_6_dout),
    .a_in_07125_empty_n(a_pipes_V_7_6_empty_n),
    .a_in_07125_read(ProcessingElement131_U0_a_in_07125_read),
    .a_out_022390_din(ProcessingElement131_U0_a_out_022390_din),
    .a_out_022390_full_n(a_pipes_V_7_7_full_n),
    .a_out_022390_write(ProcessingElement131_U0_a_out_022390_write),
    .b_in_037638_dout(b_pipes_V_7_6_dout),
    .b_in_037638_empty_n(b_pipes_V_7_6_empty_n),
    .b_in_037638_read(ProcessingElement131_U0_b_in_037638_read),
    .b_out_054908_din(ProcessingElement131_U0_b_out_054908_din),
    .b_out_054908_full_n(b_pipes_V_8_6_full_n),
    .b_out_054908_write(ProcessingElement131_U0_b_out_054908_write),
    .c_in_0691155_dout(c_pipes_V_7_6_dout),
    .c_in_0691155_empty_n(c_pipes_V_7_6_empty_n),
    .c_in_0691155_read(ProcessingElement131_U0_c_in_0691155_read),
    .c_out_0841412_din(ProcessingElement131_U0_c_out_0841412_din),
    .c_out_0841412_full_n(c_pipes_V_7_7_full_n),
    .c_out_0841412_write(ProcessingElement131_U0_c_out_0841412_write)
);
```

A simple instantation of a PE module in the matrix_mul.v module (Verilog)

FPGA Implementation

- Implemented 3x3 systolic array

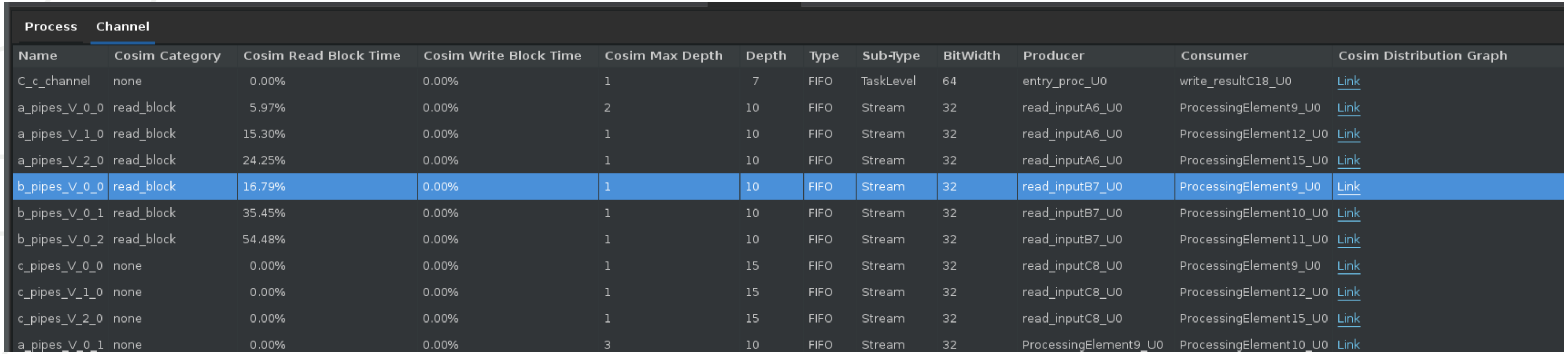

C
gmem
A
gmem1
B
entry_proc_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):0,0,0
read_inputA6_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):76,76,76
read_inputC8_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):0,0,0
read_inputB7_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):157,157,157
ProcessingElement9_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):52,52,52
ProcessingElement12_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):53,53,53
ProcessingElement10_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):105,105,105
ProcessingElement15_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):76,76,76
ProcessingElement13_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):105,105,105
ProcessingElement11_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):161,161,161
ProcessingElement16_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):106,106,106
ProcessingElement14_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):161,161,161
ProcessingElement17_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):161,161,161
write_resultC18_U0
II(avg,max,min):N/A,N/A,N/A
Latency(avg,max,min):193,193,193
gmem2
(2)

Data flow viewer

Georgia Tech

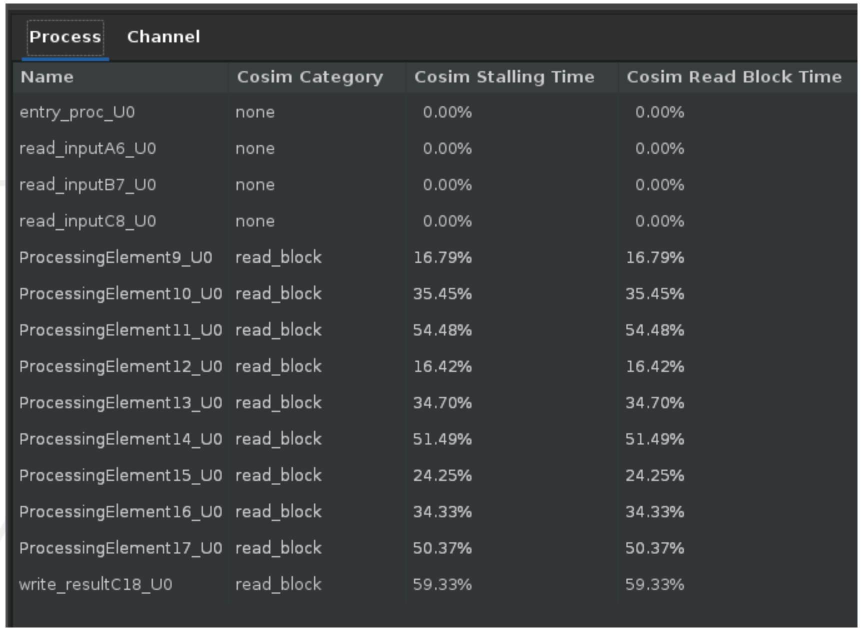
FPGA Implementation

Data flow properties



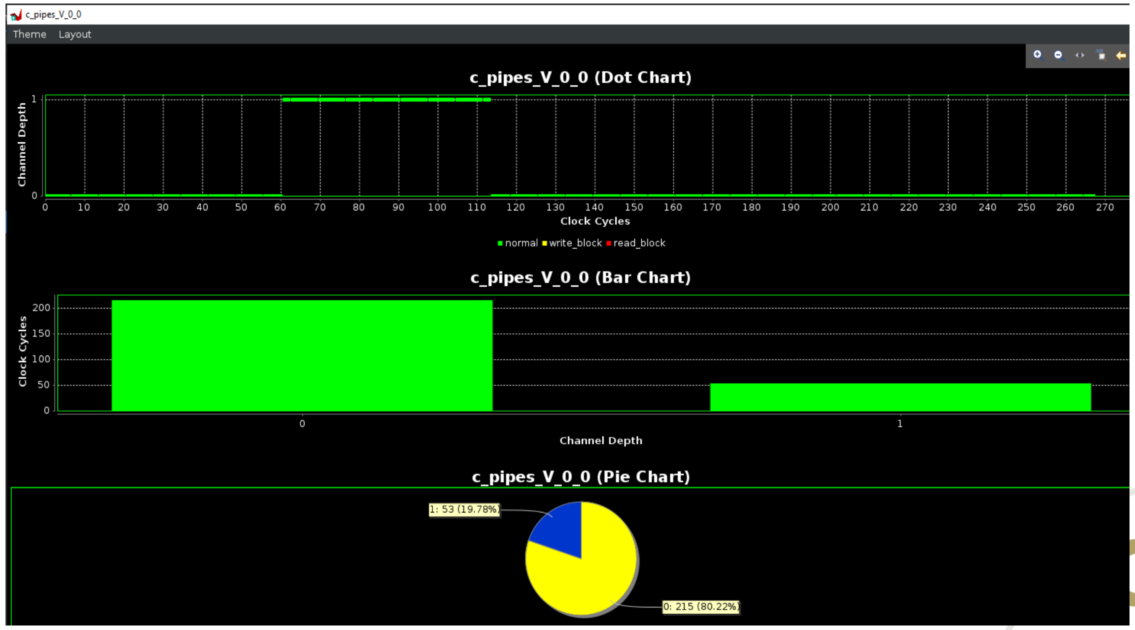
Process Channel

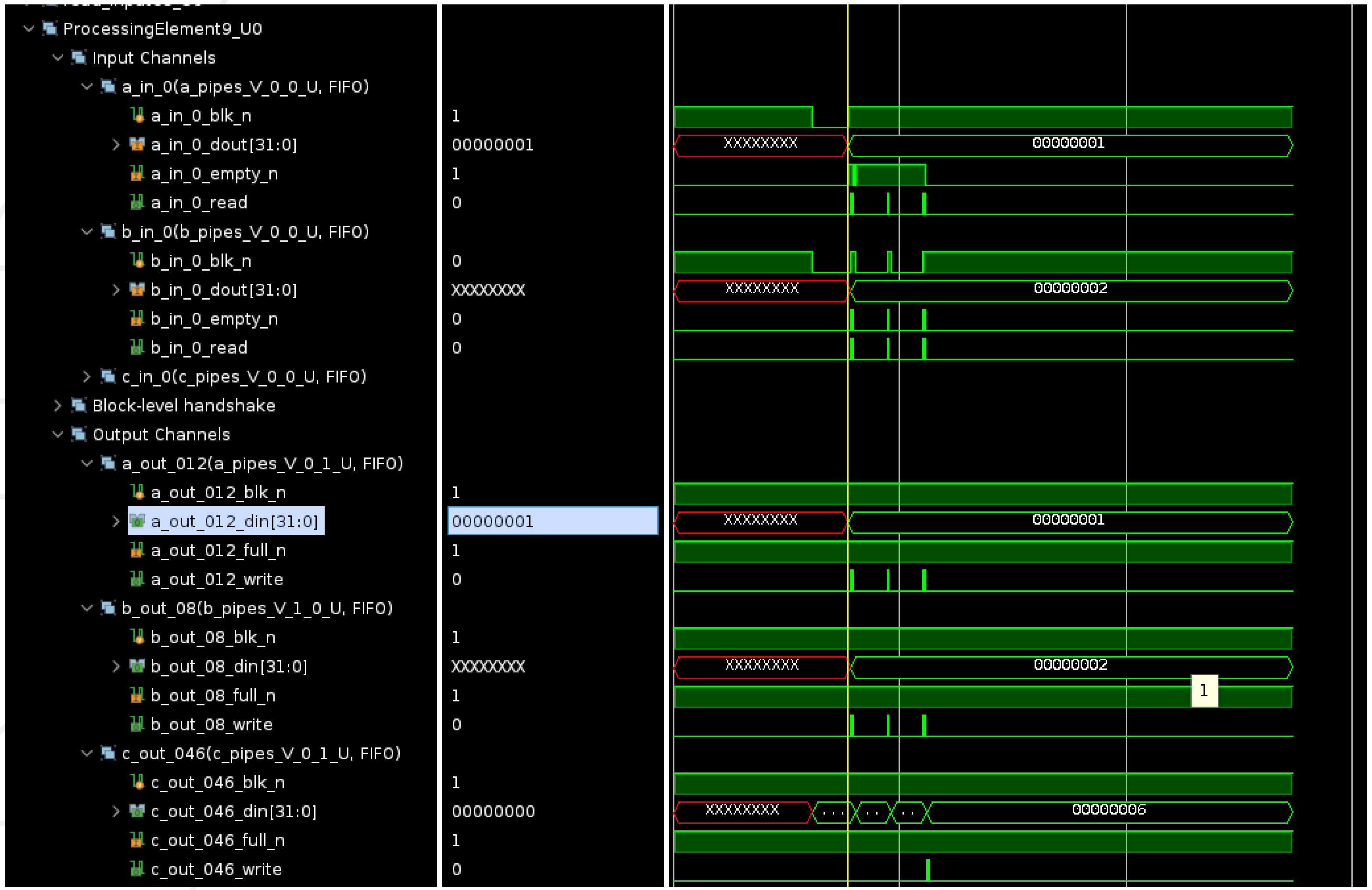
Name	Cosim Category	Cosim Read Block Time	Cosim Write Block Time	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
C_c_channel	none	0.00%	0.00%	1	7	FIFO	TaskLevel	64	entry_proc_U0	write_resultC18_U0	Link
a_pipes_V_0_0	read_block	5.97%	0.00%	2	10	FIFO	Stream	32	read_inputA6_U0	ProcessingElement9_U0	Link
a_pipes_V_1_0	read_block	15.30%	0.00%	1	10	FIFO	Stream	32	read_inputA6_U0	ProcessingElement12_U0	Link
a_pipes_V_2_0	read_block	24.25%	0.00%	1	10	FIFO	Stream	32	read_inputA6_U0	ProcessingElement15_U0	Link
b_pipes_V_0_0	read_block	16.79%	0.00%	1	10	FIFO	Stream	32	read_inputB7_U0	ProcessingElement9_U0	Link
b_pipes_V_0_1	read_block	35.45%	0.00%	1	10	FIFO	Stream	32	read_inputB7_U0	ProcessingElement10_U0	Link
b_pipes_V_0_2	read_block	54.48%	0.00%	1	10	FIFO	Stream	32	read_inputB7_U0	ProcessingElement11_U0	Link
c_pipes_V_0_0	none	0.00%	0.00%	1	15	FIFO	Stream	32	read_inputC8_U0	ProcessingElement9_U0	Link
c_pipes_V_1_0	none	0.00%	0.00%	1	15	FIFO	Stream	32	read_inputC8_U0	ProcessingElement12_U0	Link
c_pipes_V_2_0	none	0.00%	0.00%	1	15	FIFO	Stream	32	read_inputC8_U0	ProcessingElement15_U0	Link
a_pipes_V_0_1	none	0.00%	0.00%	3	10	FIFO	Stream	32	ProcessingElement9_U0	ProcessingElement10_U0	Link



Process Channel

Name	Cosim Category	Cosim Stalling Time	Cosim Read Block Time
entry_proc_U0	none	0.00%	0.00%
read_inputA6_U0	none	0.00%	0.00%
read_inputB7_U0	none	0.00%	0.00%
read_inputC8_U0	none	0.00%	0.00%
ProcessingElement9_U0	read_block	16.79%	16.79%
ProcessingElement10_U0	read_block	35.45%	35.45%
ProcessingElement11_U0	read_block	54.48%	54.48%
ProcessingElement12_U0	read_block	16.42%	16.42%
ProcessingElement13_U0	read_block	34.70%	34.70%
ProcessingElement14_U0	read_block	51.49%	51.49%
ProcessingElement15_U0	read_block	24.25%	24.25%
ProcessingElement16_U0	read_block	34.33%	34.33%
ProcessingElement17_U0	read_block	50.37%	50.37%
write_resultC18_U0	read_block	59.33%	59.33%

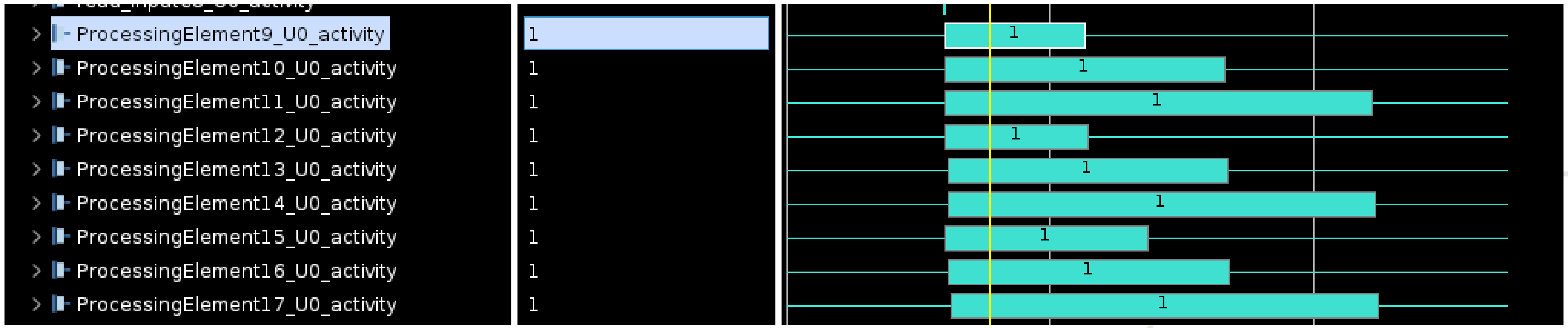




```
ProcessingElement9_U0
  Input Channels
    a_in_0(a_pipes_V_0_0_U, FIFO)
      a_in_0_blk_n            1
      a_in_0_dout[31:0]       00000001
      a_in_0_empty_n          1
      a_in_0_read             0
    b_in_0(b_pipes_V_0_0_U, FIFO)
      b_in_0_blk_n            0
      b_in_0_dout[31:0]       XXXXXXXX
      b_in_0_empty_n          0
      b_in_0_read             0
    c_in_0(c_pipes_V_0_0_U, FIFO)
  Block-level handshake
  Output Channels
    a_out_012(a_pipes_V_0_1_U, FIFO)
      a_out_012_blk_n         1
      a_out_012_din[31:0]     00000001
      a_out_012_full_n        1
      a_out_012_write         0
    b_out_08(b_pipes_V_1_0_U, FIFO)
      b_out_08_blk_n          1
      b_out_08_din[31:0]      XXXXXXXX
      b_out_08_full_n         1
      b_out_08_write          0
    c_out_046(c_pipes_V_0_1_U, FIFO)
      c_out_046_blk_n         1
      c_out_046_din[31:0]     00000000
      c_out_046_full_n        1
      c_out_046_write         0
```

Waveform viewer for one of the PEs

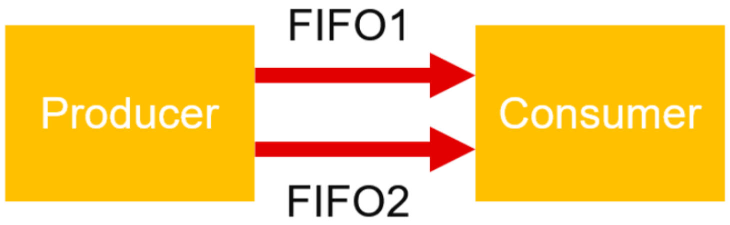
Activity time Of the PEs



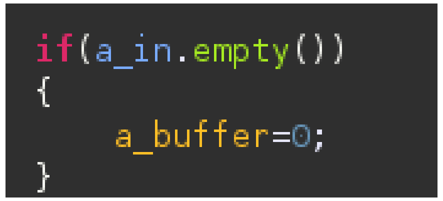
```
ProcessingElement9_U0_activity     1
ProcessingElement10_U0_activity    1
ProcessingElement11_U0_activity    1
ProcessingElement12_U0_activity    1
ProcessingElement13_U0_activity    1
ProcessingElement14_U0_activity    1
ProcessingElement15_U0_activity    1
ProcessingElement16_U0_activity    1
ProcessingElement17_U0_activity    1
```

FPGA Implementation - Points to note

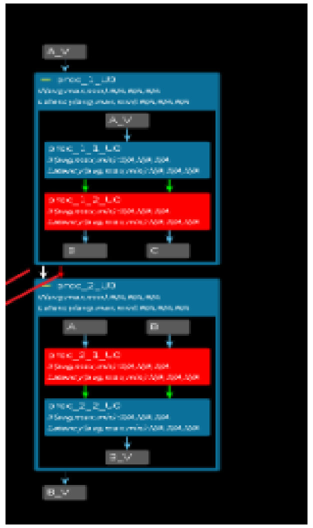
- Each stream should have one consumer and one producer.
- Insufficiently sized FIFOs can cause deadlocks



- Blocking statement calls – Stall the flow (Can check the Cosim stall time)



```
if(a_in.empty())
{
    a_buffer=0;
}
```



Scope for future work

- Address the memory accessing bottleneck using techniques like **data packing, ping pong buffers, etc.**
- Try implementing a larger systolic array to increase the resource utilization as it is available.
- Scale our design to multiply **asymmetric matrices.**
- We would like to extend our systolic array implementation to convolution as a follow-on work.