ECE 8893 Parallel Programming for FPGAs

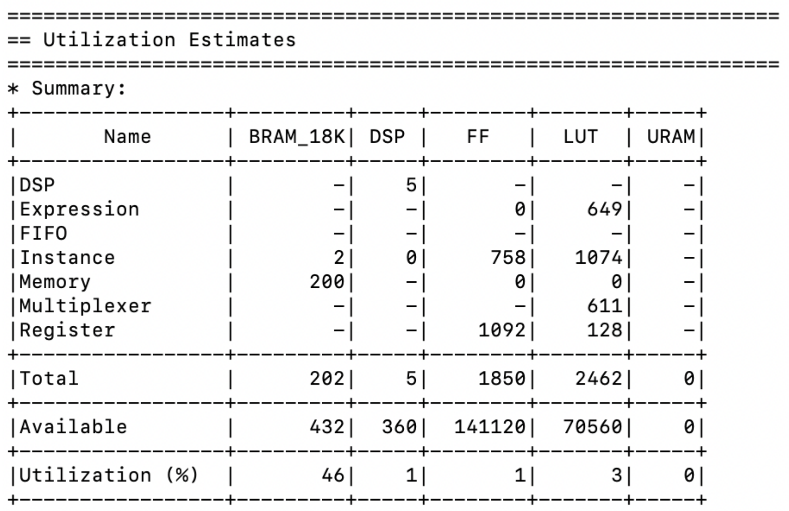
Lab 1
Matrix Multiplication Acceleration on FPGA

1. Introduction:

This aim of this Lab is to design an accelerator for matrix multiplication. In the base implementation shown below, the dimension of matrix A is M x N (100 x 200) and that of matrix B is (N x K) 200 x 300.

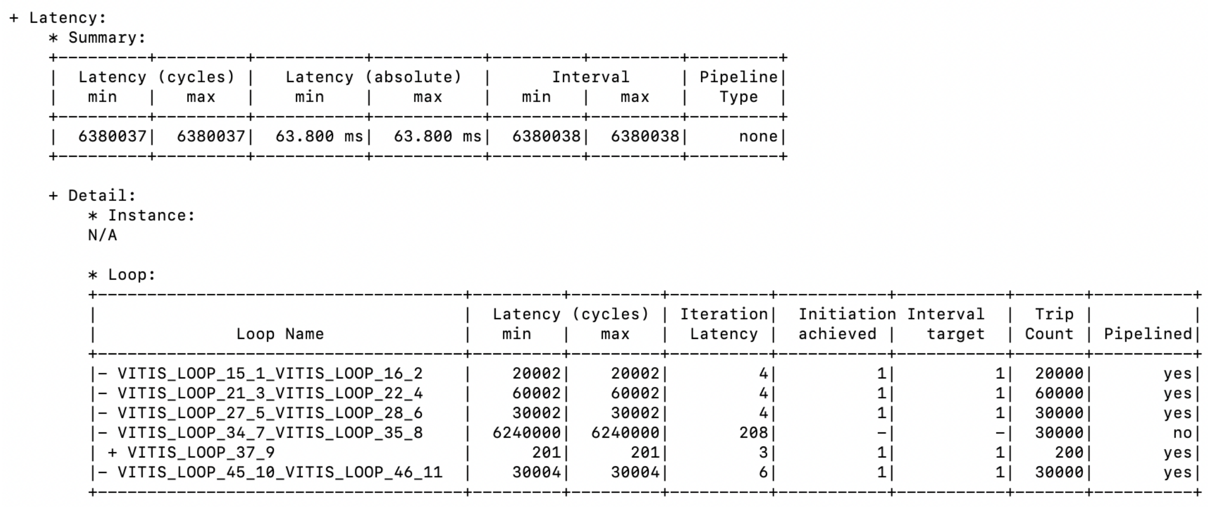
```
    // Compute the matrix multiplication
    for(int i = 0; i < M; i++) {
        for(int j = 0; j < K; j++) {

            for(int p = 0; p < N; p++) {
                MatC[i][j] += MatA[i][p] * MatB[p][j];
            }

        }
    }
```

The latency and resource utilization for the base implementation are:



```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-----+--------+-------+-----+
|       Name      | BRAM_18K| DSP |   FF   |  LUT  | URAM|
+-----------------+---------+-----+--------+-------+-----+
|DSP              |        -|    5|       -|      -|    -|
|Expression       |        -|    -|       0|    649|    -|
|FIFO             |        -|    -|       -|      -|    -|
|Instance         |        2|    0|     758|   1074|    -|
|Memory           |      200|    -|       0|      0|    -|
|Multiplexer      |        -|    -|       -|    611|    -|
|Register         |        -|    -|    1092|    128|    -|
+-----------------+---------+-----+--------+-------+-----+
|Total            |      202|    5|    1850|   2462|    0|
+-----------------+---------+-----+--------+-------+-----+
|Available        |      432|  360|  141120|  70560|    0|
+-----------------+---------+-----+--------+-------+-----+
|Utilization (%)  |       46|    1|       1|      3|    0|
+-----------------+---------+-----+--------+-------+-----+
```

Fig 1.1 Utilization estimates of base matrix mul



```
+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  Latency (cycles) |  Latency (absolute)  |      Interval     | Pipeline|
    |   min   |   max   |    min    |    max    |   min   |   max   |   Type  |
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  6380037|  6380037|  63.800 ms|  63.800 ms|  6380038|  6380038|     none|
    +---------+---------+-----------+-----------+---------+---------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +--------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |                                      |  Latency (cycles) | Iteration|  Initiation Interval  | Trip  |          |
        |               Loop Name              |   min   |   max   |  Latency |  achieved |   target  | Count | Pipelined|
        +--------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |- VITIS_LOOP_15_1_VITIS_LOOP_16_2     |    20002|    20002|         4|          1|          1|  20000|       yes|
        |- VITIS_LOOP_21_3_VITIS_LOOP_22_4     |    60002|    60002|         4|          1|          1|  60000|       yes|
        |- VITIS_LOOP_27_5_VITIS_LOOP_28_6     |    30002|    30002|         4|          1|          1|  30000|       yes|
        |- VITIS_LOOP_34_7_VITIS_LOOP_35_8     |  6240000|  6240000|       208|          -|          -|  30000|        no|
        | + VITIS_LOOP_37_9                    |      201|      201|         3|          1|          1|    200|       yes|
        |- VITIS_LOOP_45_10_VITIS_LOOP_46_11   |    30004|    30004|         6|          1|          1|  30000|       yes|
        +--------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
```

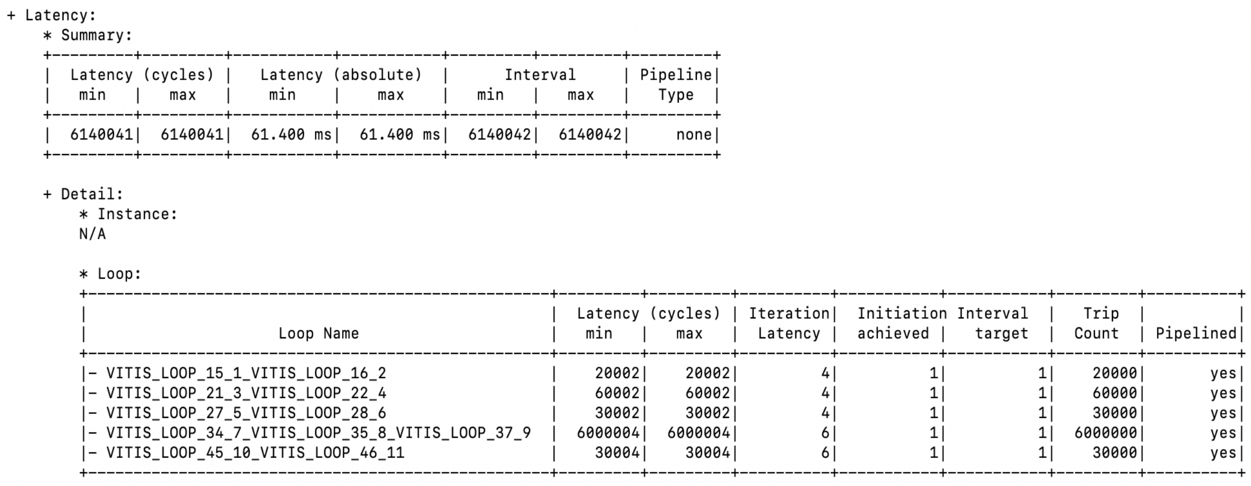
Fig 1.2 Performance estimates of base matrix mul

2. Acceleration Techniques:

1. **Loop reordering**: It can be observed from the above estimates that the innermost loop (VITIS_LOOP_37_9) in the base matrix multiplication is not pipelined due to loop-carry dependency as MatC[i][j] result must be populated before reading the result which takes more than 1 cycle. Hence, reordering the loop is beneficial in this case to eliminate the II violation.

```
// Compute the matrix multiplication
for(int i = 0; i < M; i++) {
    for(int p = 0; p < N; p++) {

        for(int j = 0; j < K; j++) {
        #pragma HLS PIPELINE II=1
            MatC[i][j] += MatA[i][p] * MatB[p][j];
        }
    }
}
```

From figure 2.1 below, it can be observed that loop reordering helps in pipelining the innermost loop in matrix multiplication and helps in reducing the latency by **239,996 cycles**.



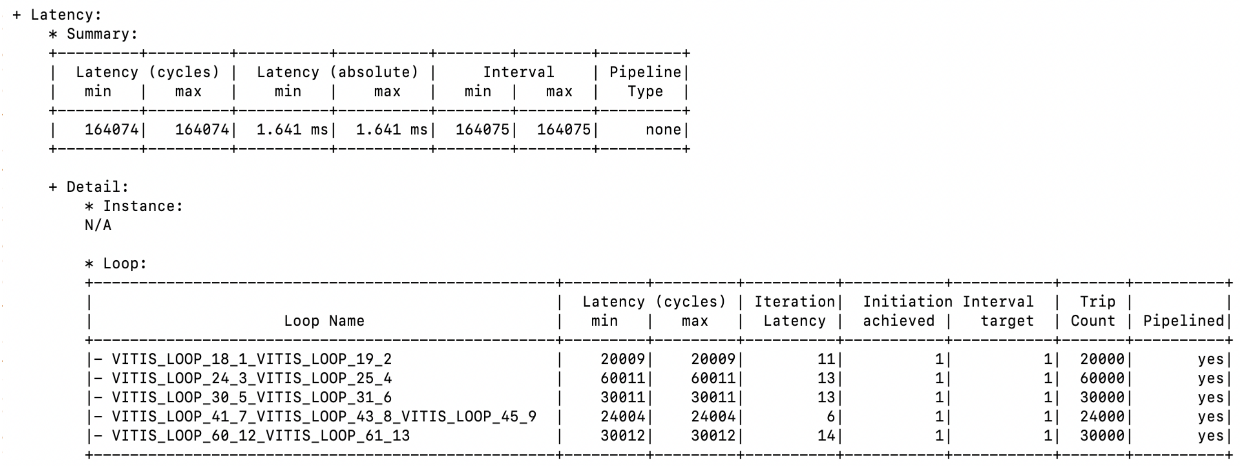
```
+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  Latency (cycles) |  Latency (absolute)   |      Interval     | Pipeline|
    |   min   |   max   |    min    |    max    |   min   |   max   |   Type  |
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  6140041|  6140041|  61.400 ms|  61.400 ms|  6140042|  6140042|     none|
    +---------+---------+-----------+-----------+---------+---------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+---------+----------+
        |                                                    |  Latency (cycles) | Iteration|  Initiation Interval  |  Trip   |          |
        |                      Loop Name                     |   min   |   max   |  Latency |  achieved |   target  |  Count  | Pipelined|
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+---------+----------+
        |- VITIS_LOOP_15_1_VITIS_LOOP_16_2                   |    20002|    20002|         4|          1|          1|    20000|       yes|
        |- VITIS_LOOP_21_3_VITIS_LOOP_22_4                   |    60002|    60002|         4|          1|          1|    60000|       yes|
        |- VITIS_LOOP_27_5_VITIS_LOOP_28_6                   |    30002|    30002|         4|          1|          1|    30000|       yes|
        |- VITIS_LOOP_34_7_VITIS_LOOP_35_8_VITIS_LOOP_37_9   |  6000004|  6000004|         6|          1|          1|  6000000|       yes|
        |- VITIS_LOOP_45_10_VITIS_LOOP_46_11                 |    30004|    30004|         6|          1|          1|    30000|       yes|
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+---------+----------+
```

Fig. 2.1 Performance estimate after loop reordering

2. **Loop Tiling**: Loop reordering gives an opportunity to unroll the inner most loop, provided matrices A, B, and C are partitioned. However, due to large size of the matrices, partitioning and unrolling causes scalability issues. Hence, loop tiling technique is used to achieve parallelism within smaller tiles. Matrices A, B, and C are partially and cyclically partitioned based on the dimension of the tile as shown below:

```
#pragma HLS array_partition variable=MatA dim=1 factor=25 cyclic
#pragma HLS array_partition variable=MatB dim=2 factor=10 cyclic
#pragma HLS array_partition variable=MatC dim=1 factor=25 cyclic
#pragma HLS array_partition variable=MatC dim=2 factor=10 cyclic
    // Compute the matrix multiplication
    for(int i = 0; i < M; i += D)
    {
        for(int p = 0; p < N; p++)
        {
            for(int j = 0; j < K; j += W)
            {
                #pragma HLS PIPELINE II=1
                for(int ii = 0; ii < D; ii++)
                {
                    FIX_TYPE a = MatA[i + ii][p];
                    for(int jj = 0; jj < W; jj++) {
                        MatC[i + ii][j + jj] += a * MatB[p][j + jj];
                    }
                }
            }
        }
    }
```

Loop tiling technique helps in achieving a speedup of D x W (D = depth, W = width of the tile). For example, when D = 25 and W = 10, the number of cycles for matrix multiplication is reduced to 24004 from 6000004, thus giving a **250x speedup for matrix multiplication**.

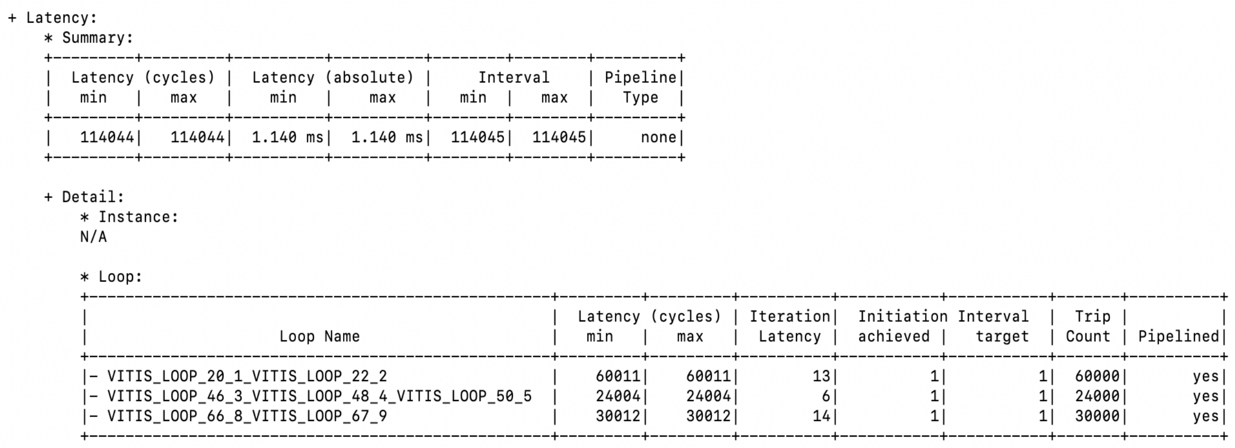


```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   164074|   164074|  1.641 ms|  1.641 ms|  164075|  164075|     none|
    +---------+---------+----------+----------+--------+--------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |                                                    |  Latency (cycles) | Iteration|  Initiation Interval  |  Trip |          |
        |                      Loop Name                     |   min   |   max   |  Latency |  achieved |   target  | Count | Pipelined|
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |- VITIS_LOOP_18_1_VITIS_LOOP_19_2                   |    20009|    20009|        11|          1|          1|  20000|       yes|
        |- VITIS_LOOP_24_3_VITIS_LOOP_25_4                   |    60011|    60011|        13|          1|          1|  60000|       yes|
        |- VITIS_LOOP_30_5_VITIS_LOOP_31_6                   |    30011|    30011|        13|          1|          1|  30000|       yes|
        |- VITIS_LOOP_41_7_VITIS_LOOP_43_8_VITIS_LOOP_45_9   |    24004|    24004|         6|          1|          1|  24000|       yes|
        |- VITIS_LOOP_60_12_VITIS_LOOP_61_13                 |    30012|    30012|        14|          1|          1|  30000|       yes|
        +----------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
```

Fig 2.2 Performance estimates after loop tiling (D = 25, W = 10)

3. **Loop Fusion:** In Fig 2.2 above, it can be observed that computation is no longer a bottleneck. The read/write operations from DRAM to BRAM take 85% of the cycles while actual computation takes 15% of the cycles. This is because loading matrices A and B and initializing matrix C happens in a sequential manner. Different DRAM ports can be used for all three matrices to allow parallel access and loop fusion can be applied to reduce the DRAM to BRAM transfer latency.

```
    int max_m_n = ((M >= N) ? M : N);
    int max_n_k = ((N >= K) ? N : K);
    for(int i = 0; i < max_m_n; i++)
    {
        for(int j = 0; j < max_n_k; j++)
        {
            if((i < M) && (j < N))
            {
                MatA[i][j] = MatA_DRAM[i][j];
            }

            if((i < N) && (j < K))
            {
                MatB[i][j] = MatB_DRAM[i][j];
            }

            if((i < M) && (j < K))
            {
                MatC[i][j] = 0;
            }
        }
    }
```



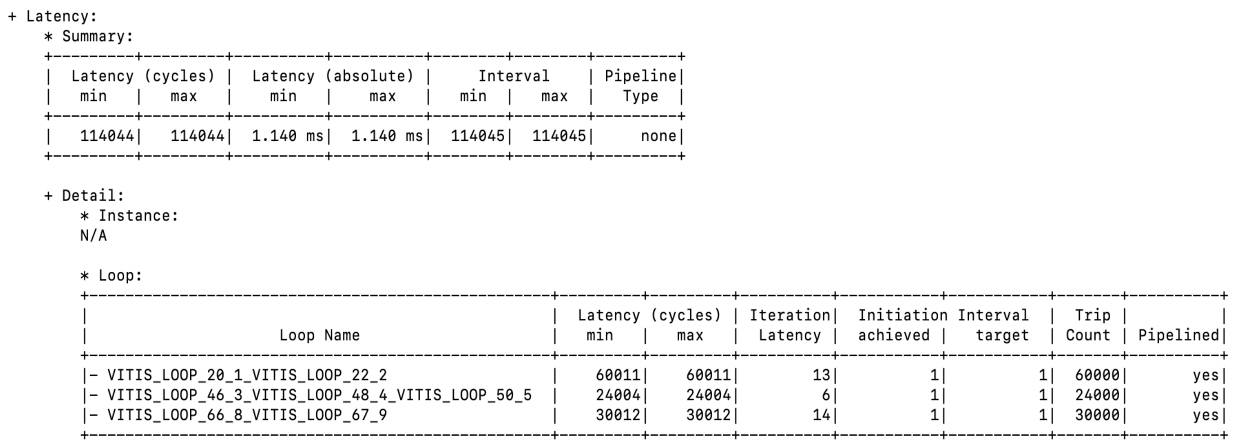
```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   114044|   114044|  1.140 ms|  1.140 ms|  114045|  114045|     none|
    +---------+---------+----------+----------+--------+--------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |                                                 |  Latency (cycles) | Iteration|  Initiation Interval  |  Trip |          |
        |                    Loop Name                    |   min   |   max   |  Latency |  achieved |   target  | Count | Pipelined|
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |- VITIS_LOOP_20_1_VITIS_LOOP_22_2                |    60011|    60011|        13|          1|          1|  60000|       yes|
        |- VITIS_LOOP_46_3_VITIS_LOOP_48_4_VITIS_LOOP_50_5 |    24004|    24004|         6|          1|          1|  24000|       yes|
        |- VITIS_LOOP_66_8_VITIS_LOOP_67_9                |    30012|    30012|        14|          1|          1|  30000|       yes|
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
```

Fig 2.3 Performance estimates after loop fusion (D = 25, W = 10)

From Fig 2.3 above, it can be observed that loop fusion leads to further reduction of **50,030 cycles**.

4. Optimized Latency and resource utilization of the final design:

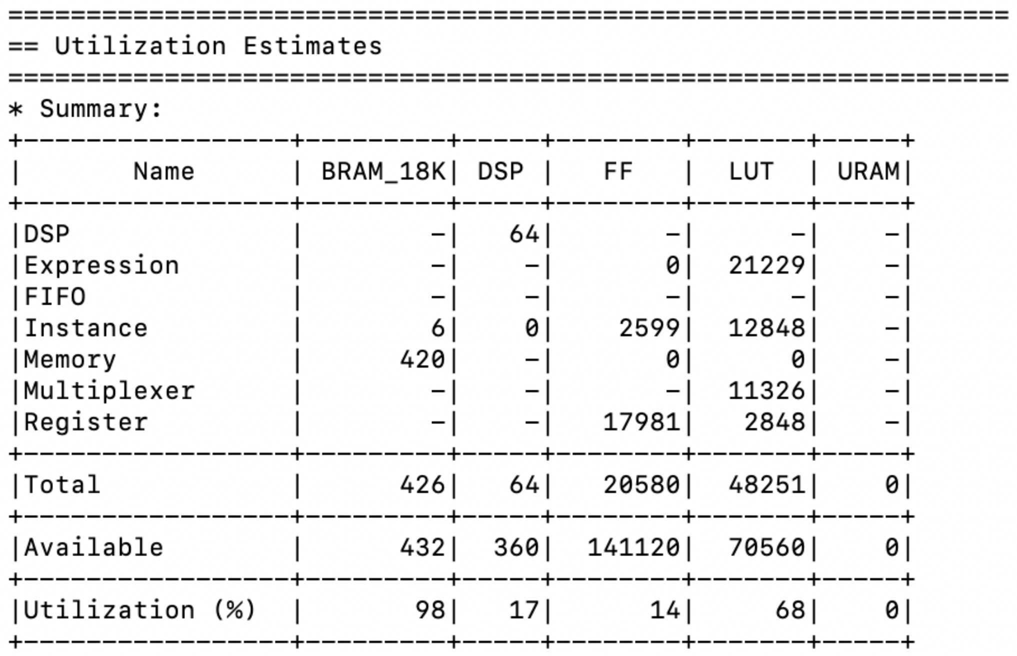
The final design achieves parallelism in tiles of D = 25, W = 10. The performance and utilization estimates can be seen below:



```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   114044|   114044|  1.140 ms|  1.140 ms|  114045|  114045|     none|
    +---------+---------+----------+----------+--------+--------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |                                                 |  Latency (cycles) | Iteration|  Initiation Interval  |  Trip |          |
        |                    Loop Name                    |   min   |   max   |  Latency |  achieved |   target  | Count | Pipelined|
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
        |- VITIS_LOOP_20_1_VITIS_LOOP_22_2                |    60011|    60011|        13|          1|          1|  60000|       yes|
        |- VITIS_LOOP_46_3_VITIS_LOOP_48_4_VITIS_LOOP_50_5 |    24004|    24004|         6|          1|          1|  24000|       yes|
        |- VITIS_LOOP_66_8_VITIS_LOOP_67_9                |    30012|    30012|        14|          1|          1|  30000|       yes|
        +-------------------------------------------------+---------+---------+----------+-----------+-----------+-------+----------+
```

Fig 4.1 Performance estimates for final design (Tile size D = 25, W = 10)

The latency of the optimized design is 114044. Thus, an overall speed up of 55.94 (6380037/114044) is gained as compared to the base implementation.



```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-----+--------+-------+-----+
|       Name      | BRAM_18K| DSP |   FF   |  LUT  | URAM|
+-----------------+---------+-----+--------+-------+-----+
|DSP              |        -|   64|       -|      -|    -|
|Expression       |        -|    -|       0|  21229|    -|
|FIFO             |        -|    -|       -|      -|    -|
|Instance         |        6|    0|    2599|  12848|    -|
|Memory           |      420|    -|       0|      0|    -|
|Multiplexer      |        -|    -|       -|  11326|    -|
|Register         |        -|    -|   17981|   2848|    -|
+-----------------+---------+-----+--------+-------+-----+
|Total            |      426|   64|   20580|  48251|    0|
+-----------------+---------+-----+--------+-------+-----+
|Available        |      432|  360|  141120|  70560|    0|
+-----------------+---------+-----+--------+-------+-----+
|Utilization (%)  |       98|   17|      14|     68|    0|
+-----------------+---------+-----+--------+-------+-----+
```

Fig 4.2 Utilization estimates for final design (Tile size D = 25, W = 10)

5. Design Space Exploration:

In order to analyze the tradeoff between latency and resource utilization, depth and width of the tiles can be changed. The following table summarizes the latency and resource utilization for different depth and width combinations:

Depth (D)	Width (W)	Latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)
5	10	210044	61	6	4	18
10	15	130044	73	10	5	41
25	10	114044	98	17	14	68
10	30	110044	107	15	13	72
20	20	105044	135	17	17	95

It can be observed from the above table that although high depth and width give a lower latency, they have a very high resource utilization which is not an optimal solution. Design selection should be made depending on which metric out of latency and area we care about.

6. Effect of quantization:

The following table summarizes the effect of quantization on resource utilization:

AP Fixed Type	Depth (D)	Width (W)	Latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)
<32, 10>	25	10	114044	98	17	14	68
<24, 8>	25	10	114044	91	156	13	70
<32, 10>	10	15	130044	73	10	5	41
<24, 8>	10	15	130044	63	94	5	42

While quantization does reduce the result precision, from the above table it can be observed that lower quantization also results in slightly lower BRAM usage at the cost of extremely high DSP usage while latency remains the same for a given <D, W> pair.

7. Observations:

While tiling helps in achieving a speedup of D x W for matrix multiplication, it can be observed that after a point increasing D and W does not result in high speed up as memory becomes a bottleneck. Even after loop fusion, about 78.9% of the cycles (90,023 cycles) are spent in loading and storing data from DRAM to BRAM whereas only 21.06% cycles (24004 cycles) are spent in the actual computation. Thus, further optimization techniques such as data streaming can be applied to reduce the memory overhead – i.e., begin computation as soon as first few elements of A and B are available instead of waiting for all elements of A and B to be loaded in BRAM.

Secondly, in the current design fanout is high as D elements from matrix A are multiplied with W elements in matrix B, which can be improved.

Finally, for an M x K matrix, D and W must be factors of M and K respectively to maintain functional correctness. Thus, there is a lot of constraint in the incremental step size of D and W. Moreover, if M and K are not factorizable, tiling becomes slightly complicated.