Matrix-Matrix Multiplication Using Systolic Array Architecture in Bluespec

Team SegFault
Chaitanya Peddawad (EE11B096), Aman Goel (EE11B087), Dheeraj B (EE11B090)

Oct. 25, 2015

1 Theoretical Background

1.1 Matrix-Matrix Multiplication on Hardware

Computing matrix products is both a central operation in many numerical algorithms and potentially time consuming, making it one of the most well-studied problems in numerical computing. Various algorithms have been devised for computing C = AB, especially for large matrices. Mapping such algorithms to custom or general purpose hardware architecture is always a challenging task. By having a custom or ASIC hardware, the matrix-matrix multiplication can be implemented using least resources and can be accelerated to a large extent. Mapping the same algorithms on general purpose hardware, for example, implementing on general purpose Xilinx FPGA board always has inherent trade-offs such as area (power), time (maximum operating clock frequency), latency, hardware utilization efficiency and so on. The realistic way to compare two solutions would be to assign weights to each of these factors and choose a solution among multiple possible pareto optimal solutions.

1.2 Systolic Array Architecture

Systolic architectures (also referred to as systolic arrays) represent a network of processing elements (PEs) that *rhythmically* compute and pass data through the system. These PEs regularly pump data in and out such that a regular flow of data is maintained [1], [2]. As a result, systolic systems feature two important properties for VLSI design:

- **Modularity**: Various functional blocks which make up the larger system have well-defined functions and interfaces. Hence, the concept of modularity enables the parallelisation of the design process.
- **Regularity**: Hierarchical decomposition of a large system results in not only simple, but also similar blocks, as much as possible.

The systolic array may be used as a coprocessor in combination with a host computer where the data samples received from the host computer pass through the PEs and the final result is returned to the host computer (see Fig. 1). This operation is analogous to the flow of blood through the heart, thus the name "systolic".

Typically, all the PEs in a systolic array are uniform and fully pipelined, i.e., all communicating edges among the PEs contain delay elements, and the whole system usually contains only local interconnections [3]. However, some relaxations have been introduced to increase the utility of systolic arrays. These relaxations include use of not only local but also neighbor (near, but not nearest) interconnections, use of data broadcast operations, and use of different PEs in the system, especially at the boundaries. With these relaxations, a family of modular, regular, and efficient data-driven array architectures can be designed for DSP applications, one of which is matrix-matrix multiplication.

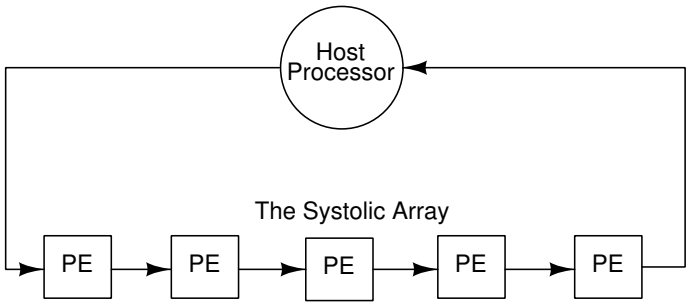


Figure 1: Basic principle of a systolic system

1.3 Systolic Array Design Methodology

We use the systolic architecture design methodology where many systolic architectures can be designed for any given regular iterative algorithm using linear mapping or projection techniques.

The dependency graph (DG) corresponds to a *space representation* where no time instance is assigned to any computation. Typically this corresponds to a t = 0 plane. The mapping technique transforms a space representation to a *space-time representation* where each node is mapped to a certain processing element and is scheduled to a certain time instance.

The systolic design methodology that we are adopting here maps a 3-dimensional DG to a 1D or 2D systolic architecture. Now we define the basic vectors involved in the systolic array design:

- **Projection vector** (also called iteration vector), $d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$: Two nodes that are displaced by d or multiples of d are executed by the same processor
- **Processor space vector**, $p^T = \begin{bmatrix} p_1 & p_2 \end{bmatrix}$: Any node with the index $I^T = \begin{bmatrix} i & j \end{bmatrix}$ would be executed by processor $p^T I = \begin{bmatrix} p_1 & p_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$
- **Scheduling vector**, $s^T = \begin{bmatrix} s_1 & s_2 \end{bmatrix}$: Any node with index I would be executed at time $s^T I$.
- **Hardware Utilization Efficiency**, $HUE = 1/\left|s^T d\right|$: This is because two tasks executed by the same processor are spaced $s^T d$ time units apart.

These aforementioned vectors must satisfy the **feasibility constraints** stated below:

- Processor space vector and the projection vector must be orthogonal to each other. If points A and B differ by the projection vector, i.e., $I_A - I_B$ is same as d, then they must be executed by the same processor. In other words, $p^T I_A = p^T I_B$. This leads to $p^T (I_A - I_B) = 0 \implies p^T d = 0$.
- If A and B are mapped to the same processor, then they cannot be executed at the same time, i.e., $s^T I_A \neq s^T I_B$, i.e., $s^T d \neq 0$.
- **Edge mapping**: If an edge e exists in the space representation or DG, then an edge $p^T e$ is introduced in the systolic array with $s^T e$ delays.

Given 2 matrices A and B, we can denote their product as $C = AB$, where A, B and C are $n \times n$ matrices. For n = 2, we have

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

These equations can be represented in a space representation as shown in Fig. 2.

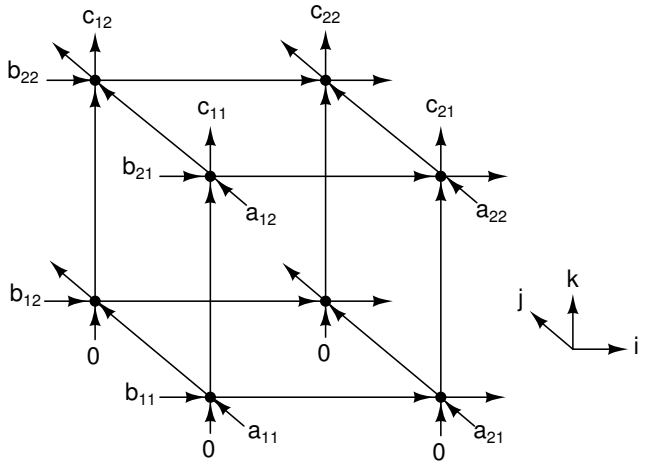


Figure 2: Systolic array architecture of the matrix product computation

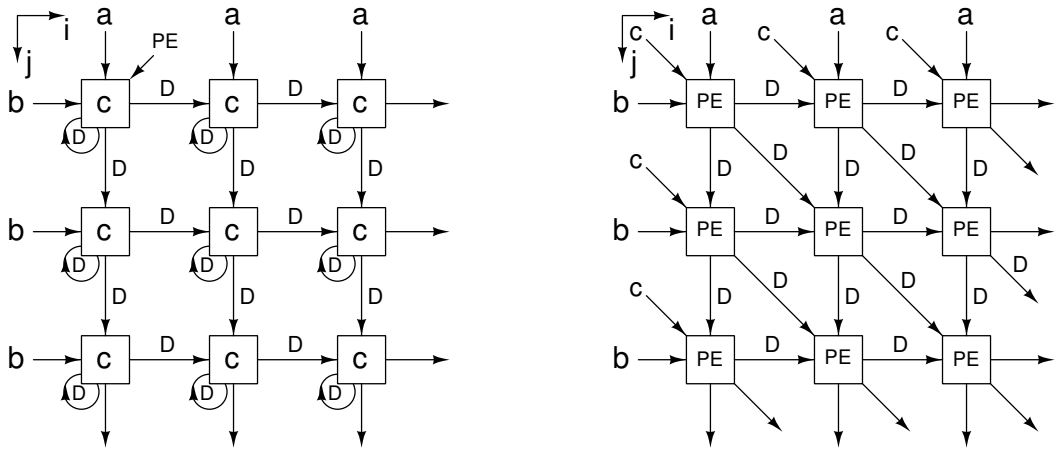
From the space diagram, we can write the iteration in standard output regular iterative algorithm (RIA) form as follows:

$$a(i,j,k) = a(i,j-1,k)$$
$$b(i,j,k) = b(i-1,j,k)$$
$$c(i,j,k) = c(i,j,k-1) + a(i,j,k)b(i,j,k)$$

With linear mapping, this 3D space representation is mapped onto 2D space to design 2D systolic arrays for matrix-matrix multiplication. With different choice of processor vector (d), projection vector (p^T) and scheduling vector (s^T) that satisfy the scheduling constraints, we get different edge mapping hence different systolic array architecture. Some of different possible solutions are derived in Tab. 1. Systolic array architecture, Arch 1 and Arch 2 using general processor elements are drawn in Fig. 3.

Table 1: Different solutions to systolic array architecture

Vector	**Arch 1**		**Arch 2**		**Arch 3**		**Arch 4**	
s^T	$\begin{bmatrix}1 & 1 & 1\end{bmatrix}$		$\begin{bmatrix}1 & 1 & 1\end{bmatrix}$		$\begin{bmatrix}1 & 1 & 1\end{bmatrix}$		$\begin{bmatrix}1 & 1 & 1\end{bmatrix}$	
p^T	$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$		$\begin{bmatrix}1 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}$		$\begin{bmatrix}1 & 0 & -1\\ 0 & 1 & 0\end{bmatrix}$		$\begin{bmatrix}1 & 0 & -1\\ 0 & 1 & -1\end{bmatrix}$	
d	$\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$		$\begin{bmatrix}1 & 1 & -1\end{bmatrix}^T$		$\begin{bmatrix}1 & 0 & 1\end{bmatrix}^T$		$\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$	
e	p^Te	s^Te	p^Te	s^Te	p^Te	s^Te	p^Te	s^Te
$a(0,1,0)$	$(0\ \ 1)$	1	$(0\ \ 1)$	1	$(0\ \ 1)$	1	$(0\ \ 1)$	1
$b(1,0,0)$	$(1\ \ 0)$	1	$(1\ \ 0)$	1	$(1\ \ 0)$	1	$(1\ \ 0)$	1
$c(0,0,1)$	$(0\ \ 0)$	1	$(1\ \ 1)$	1	$(-1\ \ 0)$	1	$(-1\ \ -1)$	1



(a) Arch 1 (b) Arch 2

Figure 3: Two-dimensional systolic array for matrix-matrix multiplication

2 Our Implementation: Design & Evaluation of Different Architectures

We implemented 4 different solutions for matrix-matrix multiplication, right from implementing on one processor element to implementing on 2D array of processor elements. The design supports muliple matrix-matrix multiplications in a single stretch in continuous fashion. We coded and simulated all four solutions in **Bluespec** to verify the accuracy of each of the solutions. We further synthesized the solutions to compare different trade-offs on hardware implementation each of them faced in terms of maximum operating clock frequency (change of critical path), area (hardware utilization) and theoretical throughput. The solutions are stated below:

- **Solution 1**: Perform matrix-matrix multiplication using 2D systolic array of processor elements.
- **Solution 2**: Perform matrix-matrix multiplication using linear 1D array of processor elements.
- **Solution 3**: Perform matrix-matrix multiplication using single processor element.
- **Solution 4**: Perform matrix-matrix multiplication using linear bidirectional 2D systolic array of processor elements.

Note that the matrix-matrix multiplication synthesis results were obtained for the 5×5 *matrices with performing 4 such multiplications one after other in a single run.*

2.1 Solution 1

This solution is implemented using 2D systolic array of processor elements, which is nothing but systolic array of sequential multiply and accumulate (MAC) units. In one step, K^2 MAC units performs K^2 multiplications of two numbers a_{ik} and b_{kj} and accumulation if applicable. But, since the outputs propagate form one PE to another, for multiplication of two $K \times K$ matrices to complete, the number of steps required is $3K - 2$. The architecture is shown in Fig. 4.

Synthesis Results

- Slice logic utilization:

  Number of slice registers: 5108/126800 $\implies$ 4%

  Number of slice LUTs: 51632/63400 $\implies$ 81%

  Number used as logic: 51632/126800 $\implies$ 81%

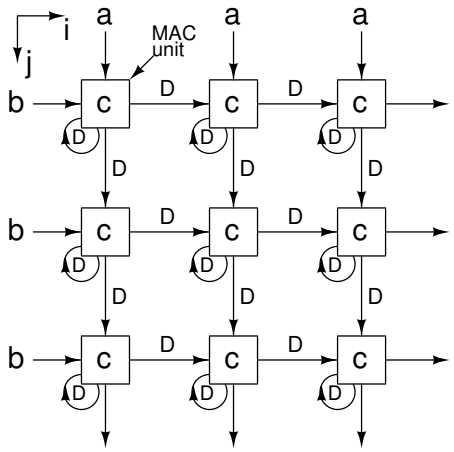
- Minimum period : 13.008 ns (Maximum frequency: 76.876 MHz).

- Critical path:

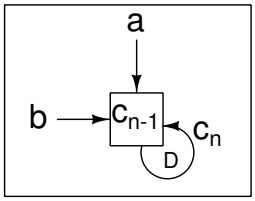
  From matNum_16 (FF) to out_66_1 (FF)

  Delay = 13.008 ns

  Levels of logic = 23



(a) Solution 1



MAC unit: $c_n = c_{n-1} + ab$

(b) Operation of MAC unit

Figure 4: Architecture for solution 1: $K = 3$

2.2 Solution 2

This solution is implemented using linear array of K processor elements, which is nothing but array of sequential multiply and accumulate (MAC) units. In one step, K MAC units performs K multiplications of two numbers a_{ik} and b_{kj} and accumulation if applicable. Hence, for multiplication of two $K \times K$ matrices, the number of steps required is K^2. The architecture is shown in Fig. 5.

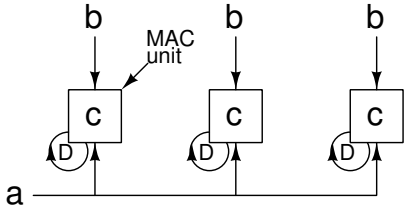


Figure 5: Architecture for solution 2: $K = 3$

Synthesis Results

- Slice logic utilization:

  Number of slice registers: 3689/126800 $\implies$ 2%

  Number of slice LUTs: 16986/63400 $\implies$ 26%

  Number used as logic: 16986/126800 $\implies$ 26%

- Minimum period : 12.008 ns (Maximum frequency: 83.278 MHz).

- Critical path:

  From matNum_16 (FF) to out_10_1 (FF)

  Delay = 12.008 ns

  Levels of logic = 11

2.3 Solution 3

This solution is implemented using only one processor element, which is nothing but sequential multiply and accumulate (MAC) unit. In one step, the MAC unit performs single multiplication of two numbers a_{ik} and b_{kj} and accumulation if applicable. Hence, for multiplication of two $K \times K$ matrices, the number of steps required is K^3. The architecture is shown in Fig. 6.

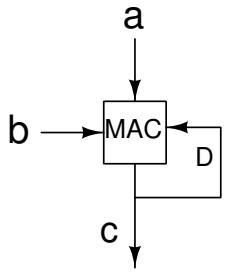


Figure 6: Architecture for solution 3

Synthesis Results

- Slice logic utilization:

  Number of slice registers: 3333/126800 $\Longrightarrow$ 2%

  Number of slice LUTs: 4798/63400 $\Longrightarrow$ 7%

  Number used as logic: 4798/126800 $\Longrightarrow$ 7%

- Minimum period : 10.938 ns (Maximum frequency: 91.424 MHz).

- Critical path:

  From matNum_16 (FF) to out_0_1 (FF)

  Delay = 10.938 ns

  Levels of logic = 10

2.4 Solution 4

This solution is implemented using K Bidirectional Linear Systolic Arrays (BLSA) of combinational PEs that takes 3 input a,b and c and produces output $c + ab$, described in [4]. Note that combinational PEs are used for the implementation and performance comparison against the sequential counterpart that has been implemented and used in first 3 solutions. It can be shown that for multiplication of two $K \times K$ matrices to complete, the number of steps required is $3K - 2$. Note that K columns of the output matrix are computed simultaneously, where one linear array computes one column each. Hence K BLSA structures are repeated regularly and work independent of each other. The architecture is shown in Fig. 7.

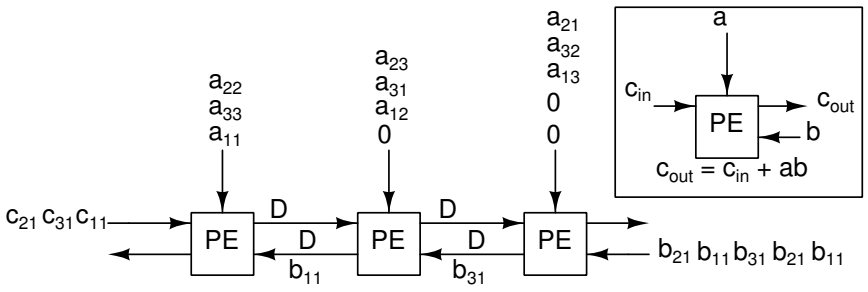


Figure 7: Architecture for solution 4: $K = 3$

Synthesis Results

- Slice logic utilization:

  Number of slice registers: 1207/126800 $\implies$ 0.95%

  Number of slice LUTs: 647/63400 $\implies$ 1%

  Number used as logic: 647/126800 $\implies$ 1%

- Minimum period : 9.374 ns (Maximum frequency: 106.657 MHz).

- Critical path:

  From macs_0_1/Mmult_ans_a_MUL_ans_b___d1 (DSP) to c_reg_0_1_31 (FF)

  Delay = 9.374 ns

  Levels of logic = 18

2.5 Pareto Curve: Face-off Between Solutions

Architecture	Slice Reg. Utilization	Logic LUTs Utilization	Clock Frequency	Throughput
Solution 1	4%	81%	76.87 MHz	$\frac{1}{3K-2}$
Solution 2	2%	26%	83.27 MHz	$\frac{1}{K^2}$
Solution 3	2%	7%	91.42 MHz	$\frac{1}{K^3}$
Solution 4	0.95%	1%	106.67 MHz	$\frac{1}{3K-2}$

Table 2: Trade-offs in different solutions ($K = 5$)

The trade-offs are clear from the Tab. 2. Note that solution 1 uses custom sequential MAC units while solution 4 uses default combinational processor elements, hence significant difference in hardware utilization and the clock period. Solutions 1-3 use sequential PEs hence are included in pareto curve where we evaluate the solutions based on trade-off between hardware utilization and throughput, or between clock freqeuncy and throughput. Either way, all three are pareto optimal. The pareto curve is drawn in Fig. 8.

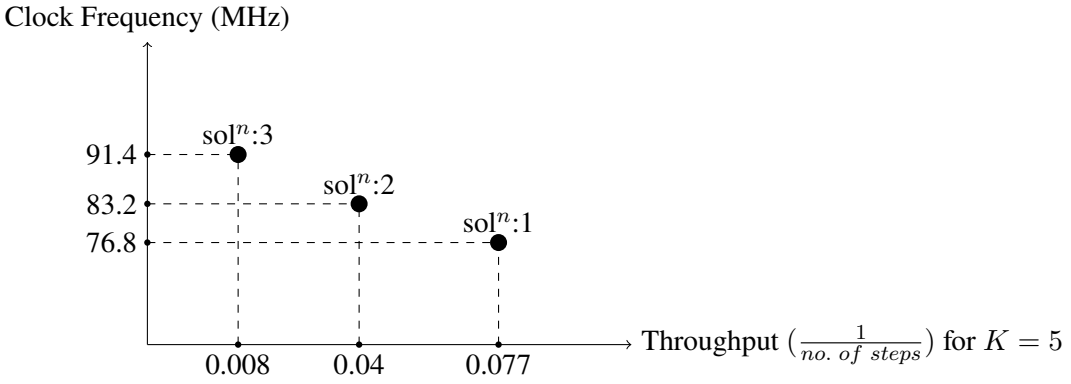


Figure 8: Pareto Curve (*not to scale*)

3 Conclusion

We have explored four different solutions to implement matrix-matrix multiplication. Selection of a solution to one particular application depends on several factors such as dimensions of matrix, constraints on throughput, resource utilization and makespan as demanded by the application. On a general perception we can say that for smaller matrices with stringent hardware utilization constraint, solution 3 performs

good. On the other hand to get better throughput at the cost of extra hardware, solution 1 will be more optimal compared to solution 3. Solution 2 assumes a middle ground between solution 1 and solution 3. However we can see that the best results are obtained when the processing element is implemented using combinational logic as in solution 4. Similar results can also be obtained from solution 1 if the basic PE is combinational. While solution 4 demands more complicated scheduling of the inputs, solution 1 requires a simple sequencing of inputs and can thus lead to interesting results if the basic PE of solution 1 is implemented using combinational logic.

References

[1] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," *Sparse Matrix Symposium, SIAM*, pp. 256–282, 1978.

[2] H. T. Kung, "Why systolic architectures ?" *IEEE Computers Magazine*, vol. 15, pp. 37–45, Jan. 1982.

[3] S. Y. Kung, "VLSI Array Processors," *Prentice Hall*, 1988.

[4] E. I. Milovanovic et. al, "Matrix Multiplication on Linear Bidirectional Systolic Arrays," *Ser. A: Appl. Math. Inform. and Mech*, vol. 2, no. 1, pp. 11–20, 2010.