Comparison between Systolic Array and Vector Multiplier based on FPGA Implementation

1st DuHyeon Kim
School of Electrical Engineering
Korea University
Seoul, Republic of Korea
kdhluck@naver.com

***Abstract*—This paper compares systolic arrays and vector multipliers on the Xilinx xcu250 FPGA for matrix multiplication. Systolic arrays excel in timing consistency and scalability due to inherent pipelining, while vector multipliers require complex pipelining, leading to higher resource usage and limited scalability. The findings highlight trade-offs in resource efficiency and performance for matrix multiplication workloads.**

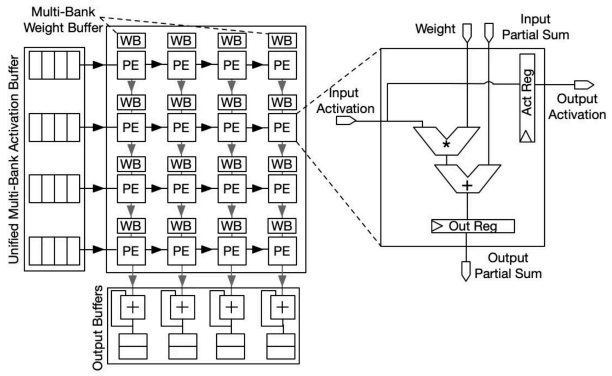
I. Introduction

The rapid rise of machine learning accelerators targeting data centers has led to a surge in hardware innovations. NVIDIA has introduced versatile GPGPUs capable of large-scale LLM training and inference, while major tech companies like Google, AWS, and Meta are focusing on developing proprietary AI chips tailored to their specific service requirements. This paper explores the comparative performance of systolic arrays and vector multipliers when implemented on FPGA platforms, offering insights into their potential roles in this evolving landscape.

The initial verification and analysis of such ML accelerators are often conducted rapidly on FPGA platforms. In this study, we implement a weight-stationary basic systolic array and an adder-tree-based vector multiplier on Xilinx's xcu250 UltraScale+ architecture FPGA. Through this implementation, we analyze timing, power characteristics, and FPGA resource utilization, providing a comprehensive evaluation of the fundamental performance and efficiency of MXU (Matrix Unit) building blocks.

II. Preliminaries

A. Systolic Array

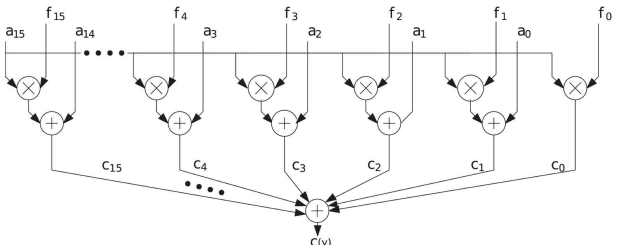


A systolic array is a control-minimized methodology based on processing elements (PEs). Each PE contains a MAC (Multiply-Accumulate) unit and performs computations as long as data flows are provided in sync with the timing requirements. The key advantage of systolic arrays lies in their minimal data control overhead.

Systolic arrays are typically implemented using either the weight stationary (WS) or output stationary (OS) approach. In the WS approach, weights are held within the PEs while outputs are propagated outside the array. In contrast, the OS approach retains the output (partial sum) within the PEs, allowing weights and input activations to flow through the array.

In this paper, we adopt the weight stationary approach, leveraging the TPUv1 architecture as a reference. A simplified TPU module is implemented, including basic control logic and buffers, to analyze its performance.

B. Vector Multiplier (Adder-tree based)



A vector multiplier operates by producing an output vector directly from a vector input without requiring additional data setup. It simply processes the input data through multipliers to generate the output. For matrix multiplication, where summation of elements is required, an adder-tree-based architecture is typically employed. This structure is commonly used in GPUs, which execute thousands of vector multipliers in parallel using a large number of threads, with a focus on managing and controlling the resulting computations efficiently.

In this paper, we implement a pure vector multiplier architecture without pipelining, focusing on its fundamental operation. Considerations for pipelining are reserved for future exploration and discussion.

C. FPGA Features

FPGAs are versatile hardware platforms that provide configurable logic resources, making them ideal for prototyping and implementing custom architectures. At the core of FPGA functionality is the Configurable Logic Block (CLB), which includes Look-Up Tables (LUTs) and registers. LUTs are small memory elements used to implement combinational logic functions, while registers support sequential operations. In the Xilinx xcu250 UltraScale+ FPGA, LUTs can operate as logic elements or memory components, such as distributed RAM, enabling a wide range of design possibilities.

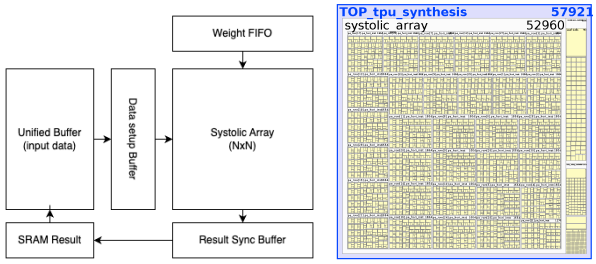
The xcu250 UltraScale+ FPGA offers extensive resources, including 1,728,000 LUTs, 3,456,000 registers, and 2,688 block RAM tiles. In this study, only a small fraction of these resources is used, highlighting the resource efficiency of the

implemented architectures while leaving ample room for scalability and optimization.

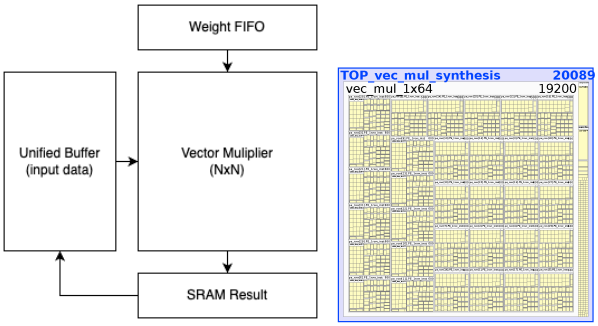
III. Tested Architecture

In this paper, the tested architecture consists of three Buffers (result, input, and weight) and control units as the basic components. Two types of MXUs, namely systolic arrays and vector multipliers, are integrated into the design. The architecture is implemented to perform matrix multiplication for square matrices of sizes 8, 16, 32, and 64. Functionality is validated, and the design undergoes synthesis, placement, and routing using Vivado, with timing constraints (.xdc) applied for accurate evaluation.

A. Systolic Array Architecture



B. Adder tree based Vector Multiplier



C. Key Flags (Vivado)

In this implementation, Vivado flags were used to control synthesis and ensure design integrity:

- **dont_touch**: Applied to prevent optimization of registers (weight reg, din reg) in PEs.
- **use_dsp**: Enabled for wires carrying partial sums and multiplications to utilize DSP units.
- **ram_style**: Set to block for buffer memories to allow consistent and objective resource utilization comparisons.

IV. Implementation Results

A. FPGA Utilizations

TABLE I. Systolic Array Utilization

# Utils	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
LUT	202	656	1502	4125
FF	2643	11091	42724	167434
BRAM	14	38	132.5	491.5
DSP	64	256	1024	4096

Fig. 1. Utilization of systolic array based on different Matrix size.

Figure Lab

TABLE II. Vector Multiplier Utilization (not pipelined)

# Utils	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
LUT	38	87	169	325
FF	522	2060	8207	32787
BRAM	12	38	132.5	491.5
DSP	72	304	1248	5056

Fig. 2. Utilization of Vector Multiplier array based on different Matrix size.

B. Timing (Max Delay)

TABLE III. Max Delay Path (not pipelined)

T (ns)	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
Systolic Array	4.995	4.904	4.443	6.433
Vector Multiplier (non-pipelined)	9.383	13.413	24.552	54.936
Vector Multiplier (pipelined)	6.16	8.986	8.509	22.722
Vector Multiplier (pipelined + DSP Flag)	6.16	8.986	9.44	25.555

Fig. 3. Maximum Data Path Delay

C. Vector Multiplier pipelined

TABLE IV. Vector Multiplier Utilization (pipelined)

# Utils	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
LUT	38 (38)	87 (87)	9129 (169)	36166 (325)
FF	1546 (522)	6156 (2060)	30735 (8207)	122899 (32787)
BRAM	12 (12)	38 (38)	132.5 (132.5)	491.5 (491.5)
DSP	96 (72)	384 (304)	1152 (1248)	4608 (5056)

Fig. 4. 1stage pipelined (N=8, 16) / 2stage pipelined (N=32)

TABLE V. Max Delay Path (pipelined)

T (ns)	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
Systolic Array	4.995	4.904	4.443	6.433
Vector Multiplier (pipelined)	6.16	8.986	8.509	22.722

Fig. 5. Original systolic array / Pipelined Vector multiplier critical delay

TABLE VI. Max Delay Path (pipelined + DSP flag)

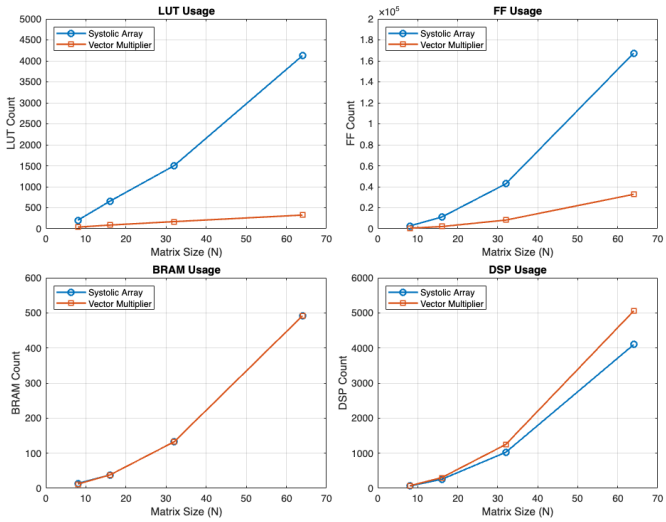
T (ns)	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
Systolic Array	4.995	4.904	4.443	6.433
Vector Multiplier (pipelined)	6.16	8.986	9.44	25.555

Fig. 6. Original systolic array / Pipelined Vector multiplier critical delay

# Utils	Matrix Size (NxN)			
	N = 8	*N = 16*	*N = 32*	*N = 64*
LUT	38 (38)	87 (87)	169 (169)	326 (325)
FF	1546 (522)	6156 (2060)	30735 (8207)	122899 (32787)
BRAM	12 (12)	38 (38)	132.5 (132.5)	491.5 (491.5)
DSP	96 (72)	384 (304)	1664 (1248)	6656 (5056)

V. Comparisons

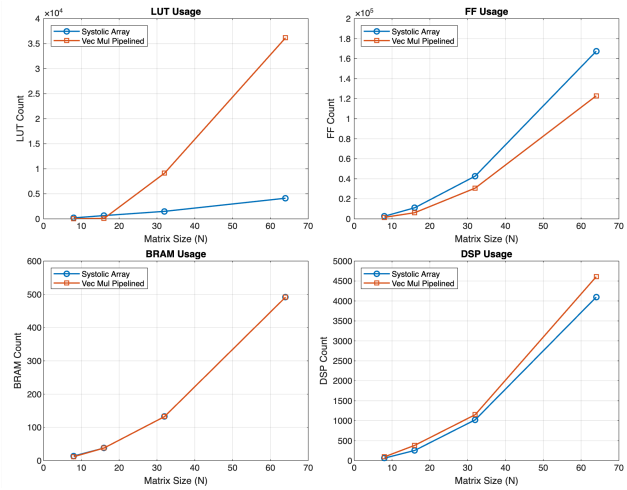
A. Non Pipelined Utilization



For a non-pipelined vector multiplier, the FPGA resource utilization is significantly lower compared to a systolic array. This difference arises because the systolic array inherently incorporates pipelining within its architecture and requires additional logic to set up input data and align output data for results, leading to higher resource consumption.

However, as the matrix dimensions increase, the non-pipelined vector multiplier exhibits a nearly twofold increase in maximum delay with each size increment. In contrast, the systolic array maintains a relatively consistent delay regardless of the matrix size due to its efficient dataflow structure.

B. Pipelined Utilization

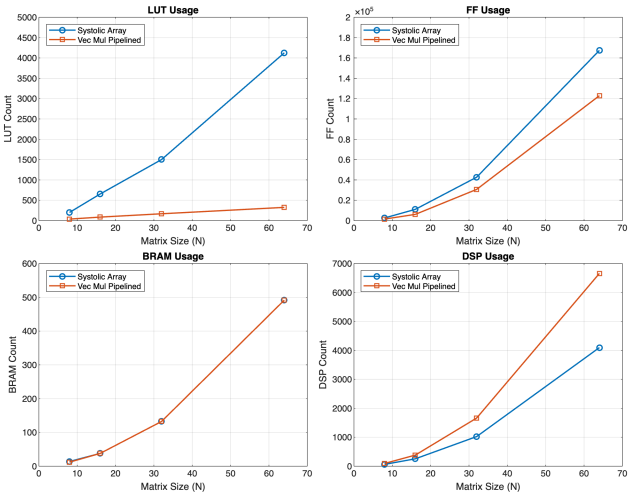


To enable a more valid comparison, the vector multiplier was modified to include pipelining. For smaller matrices (N=8, N=16), a single-stage pipeline was introduced just before the adder tree. For larger matrices (N=32), an additional pipeline stage was added at the level 2 adder of the adder tree, resulting in a two-stage pipeline. This modification allowed the design to meet the 100 MHz timing constraint successfully.

After introducing pipelining, the comparison between the pipelined vector multiplier and the systolic array revealed that the overall FPGA resource utilization became more comparable. Notably, DSP and LUT utilization increased significantly in the pipelined vector multiplier. The DSP increase is attributed to path division within the ALU caused by pipelining in the adder tree. Similarly, the LUT usage increased, as certain paths that could not fully leverage DSP resources defaulted to LUT-based CLBs. This highlights a key challenge: as matrix dimensions grow, the pipelined vector multiplier requires careful consideration of timing and additional pipeline stages, which negatively impact design scalability compared to the systolic array.

The increased pipelining in the vector multiplier significantly raises FPGA utilization due to longer data paths, yet its throughput remains less than half that of the systolic array. This highlights the inherent limitations of vector multipliers for large-scale matrix multiplication, as pipelining improves timing at the cost of higher resource usage and reduced scalability compared to systolic arrays.

C. Pipelined Utilization + DSP Flag in pipelined Path



To reduce excessive LUT usage in pipelined paths, the use_dsp flag was applied. The higher LUT utilization in systolic arrays compared to vector multipliers is mainly due to the data setup controller, not the computation units.

Using DSP units lowered LUT usage but significantly increased DSP utilization, highlighting a trade-off between these resources. The sharper increase in LUT utilization when DSPs are not used suggests that DSPs are more cost-effective for such tasks.

Ultimately, pipelining in vector multipliers remains resource-intensive and technically challenging, regardless of whether LUTs or DSPs are used. This reinforces the complexity and cost of implementing effective pipelining strategies.

VI. Conclusion

This implementation enabled a comparative analysis of FPGA utilization and critical delay between systolic arrays and vector multipliers. The results highlight the scalability and optimized latency of systolic arrays, driven by their dataflow-based pipelined architecture for matrix multiplication. In contrast, the complexity of pipelining strategies required for vector multipliers to achieve comparable latency underscores the inherent advantages of systolic arrays in large-scale matrix operations.

In conclusion, systolic arrays, with their simplified control logic, are well-suited for large-scale matrix multiplications. On the other hand, vector multipliers are more effective when deployed in smaller units within large MXUs, as seen in GPU architectures, where efficient logic controls numerous parallel multipliers. This distinction provides valuable insight into the design trade-offs and the optimal use cases for each architecture.

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *44th International Symposium on Computer Architecture (ISCA)*, Toronto, Canada, June 2017.

[2] H. T. Kung, "Why Systolic Architectures?" *IEEE Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[3] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," *IEEE Micro*, DOI: 10.1109/MM.2021.3058217, 2021.