Systolic Array Matrix Multiplication Accelerator

Alexandru Puşcaşu*†, Cătălin Bogdan Ciobanu*†, Octavian Buiu*
*National Institute of Research and Development on Microelectronics, Romania
{alexandru.puscasu, catalin.ciobanu, octavian.buiu}@imt.ro
†Transilvania University of Braşov, Romania
alexandru.puscasu@student.unitbv.ro, catalin.ciobanu@unitbv.ro

***Abstract*—Systolic arrays are a simple solution to accelerate matrix multiplication. Matrix multiplication is a common operation used in artificial intelligence. We designed a loosely-coupled matrix multiplier with a 2D systolic array. The accelerator is configured with matrices parameters. The host raises a start flag to start an operation and when is completed another flag is raised by the accelerator. The accelerator has dedicated memory interfaces. The accelerator was designed in SytemVerilog. The design was synthesized targeting an AMD/Xilinx VCU128 FPGA to measure the area utilization for various 2D systolic array dimensions. Another tests counted the clock cycles for the matrix multiplication. To reduce the total operation time, we proposed a parallel data flow. The design works on different clock domains: for the interfaces and the internal logic. Our experimental results suggest that our accelerator is up to 3.1X faster for a 8×8 Systolic Array and up to 145.5X faster for a 64×64 Systolic Array compared to baseline RISC-V processor.**

***Index Terms*—Systolic Array, SystemVerilog, RISC-V**

I. Introduction

Systolic arrays (SA) are parallel processing elements (PE) interconnected together in a 2D matrix. For matrix multiplication, there are two input matrices. The result element at position *i, j* is the dot product between *i-th* line of first matrix and *j-th* column of second matrix. Based on a 2D architecture, a systolic array has two types of pipeline directions: lines and columns. The lines pipelines have data from the first matrix in a line-by-line order. The columns pipelines use the second matrix in column-by-column order [1].

Figure 1 illustrates a 2D SA for matrix multiplication, presenting the data flows in the matrix structure. The input matrices are 2×2 and the systolic array is 2×2. The A matrix is processed in line-by-line order, and its data flows through systolic array rows. The B matrix is processed in column-by-column order, and its data flows through the systolic array columns. SA require a strict timing in order to have valid data in all PEs. To meet those requirements, some zero elements are inserted at the beginning.

The PE presented in Figure 2 implements two functions. The multiply-add-carry (MAC) function multiplies two inputs and adds the result using an accumulator. The second function is to store data elements and send them to the next stages [2]. This element has a passive role, as a computation node, and an active role, as an element in a pipeline.

The main contributions of this work are:

- A matrix multiplication accelerator micro-architecture based on SA;

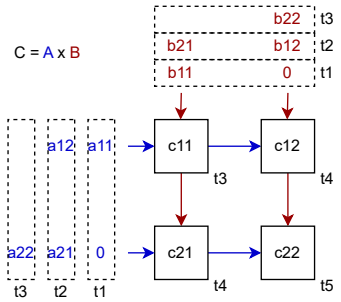

Fig. 1. Example 2D systolic array timing for matrix multiplication

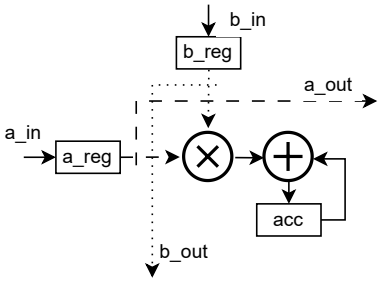

Fig. 2. 2D systolic array processing element example

- Overlapping the data processing and communication to reduce total execution time;
- SystemVerilog implementation targeting an AMD/Xilinx VCU128 FPGA (Field Programmable Gate Array);
- Performance of our accelerator. We observed speedups ranging from 3.1X to 145.5X depending on SA size and matrix size.

The remainder of this work is organized as follows: related work is presented in Section II. Section III describes the implementation of the accelerator. The experimental setup is described in Section IV and the results are presented in Section V. Section VI concludes this work.

II. Related Work

Systolic Arrays research started in 1982 as a solution that increased computation speed without increasing the memory bandwidth [3]. The main advantage is the versatility of systolic arrays for various applications. Furthermore, it has a great

2024 International Semiconductor Conference (CAS) | 979-8-3503-5207-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/CAS62834.2024.10736842

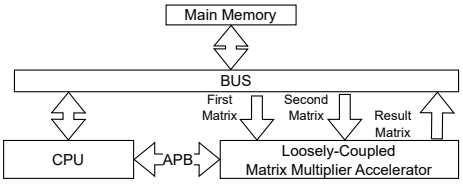

Fig. 3. A use case for our loosely-coupled matrix multiplier accelerator

replication factor, making SAs suitable for Very Large Scale Integration (VLSI) [3].

SAs are useful as hardware for inference in deep learning. Google uses SAs for the first version of the Tensor Processing Unit (TPU), employing an array of 256x256 8-bit integer PEs [4]. The second generation TPU has two cores featuring 128×128 8-bit integer SAs [5].

The main focus in modern SA research is to reduce memory accesses and to improve data reuse by optimizing the data flow. In [6] the authors accelerated the convolution operation and analyzed some existing data flows: no local data, weight stationary and output stationary. Based on that, the authors proposed a new data flow: row stationary. As an improvement of the previous data flow, [7] adds a shared storage for PEs.

Systolic array research on 3D-ICs is presented in [8] and [9]. The authors implement the SAs in 3D and enable data reuse along the third dimension. This approach achieved performance improvements while using less energy.

Other studies optimize the MAC of the PE. In [10] the authors implement the unit on a Radix-8 multiplier, and test it on 8, 16 and 32 bit integer numbers. The power is reduced by up to 30%, the area is reduced by up to 16.7%, but the delay of the array is increased.

In [11] the authors analyzed the SA architecture and split a large SA in multiple smaller arrays. Each small array has its own buffers and extra buffers are shared between all arrays. This memory topology is similar to cache levels. The authors use this approach to reduce systolic array management components, leading to power and area optimizations. On the data flow side, all systolic arrays use the same chunk of data in a small amount of time.

III. Implementation

Our Register Transfer Level (RTL) implementation is written in SystemVerilog. The accelerator has four interfaces: one Advance Peripheral Bus (APB) for configuration and three interfaces for memory access for the tree matrices (two inputs and one output). The memory interfaces are of type Request-Acknowledge. A possible configuration for the usage of the accelerator is presented in Figure 3. The CPU and accelerator use the same address space. The CPU configures the accelerator.

The accelerator was designed to work with 8-bit integers inputs and results on 16-bit integers. The data memory interfaces are 256 bits wide and addresses are 16-bit. The accelerator requires the input matrix dimensions and the start addresses of the matrices. An additional register is used for state. It has

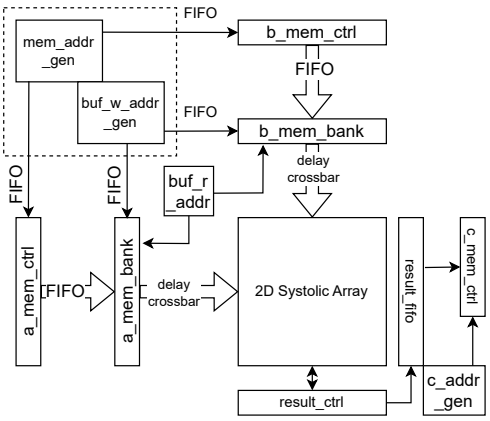

Fig. 4. 2D SA matrix multiplication accelerator micro-architecture

one start bit, one end bit and one bit for invalid input. The input dimension needs to be a multiple of 32. The accelerator has seven 16-bit registers.

The accelerator employs three clock domains: configuration interface clock, memory bus clock and internal logic clock. Between those clock domains there is high data traffic, and we design it to overcome the metastability problems. Those transitions are facilitated by asynchronous First-In-First-Out (FIFOs) and synchronizations using flip-flops.

The system has two parameters to configure the SA dimensions at compile time.

A. Architecture

The accelerator has two main components: Register File (RF) and SA. The RF is connected to APB and is configured by the user and passes the matrix parameters to the SA controller. The SA design is more complex, including the 2D SA, memory banks and control logic.

The micro-architecture of the SA is shown in Figure 4. To get the data from main memory, an Address Generation (AG) module generates addresses for input matrices. One matrix is addressed in a row-wise and the other column-wise. The AG modules send addresses to the memory controller modules. A FIFO is used to store addresses because of the different clock domains. After the data is received, a FIFO stores the data and another component writes it in the buffer memory in the correct order for the SA.

Buffers are used to store input data for the SA. When the multiplication starts, the array needs to be fed with data. The memory banks are implemented using simple dual port memories (one read and one write port).

Because a PE has two latency lines, we had to synchronize those latencies at the input. To achieve that, every input line needs to be delayed by one clock cycle by the line ahead of it. Our solution was to implement a delay crossbar. It has multiple independently delay lines, and every line adds one extra delay to the line above.

In the SA, the PE stores the partial result. Because data arrives at every clock cycle, we added an extra signal to

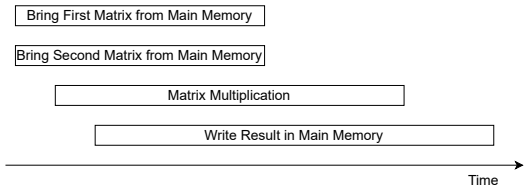

Fig. 5. Task concurrency of accelerator algorithm.

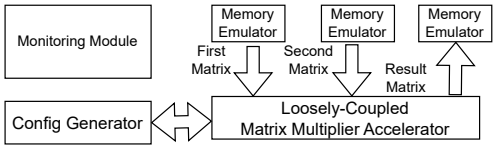

Fig. 6. Simulation environment of the accelerator. The monitoring module inspects all interfaces and internal signals of the accelerator.

emulate the reset. This signal selects which result to store in the result flip-flop (FF). The options are: the result of the multiplication, without the partial result, or the result of the addition, with the partial result.

Another component allows capturing the results and control the PE reset. This component has a counter and when the results are ready, they are sent to the FIFO. Another component generates the right address for the result. Those two data elements are received by another memory controller that writes the results back to main memory.

B. Algorithm

Many components in our design work independently in parallel, including reading and write data from main memory. When data is available, the SA is started and stops when all data is computed. Our algorithm fully exploits this parallelism. The accelerator fetches some of the input data and starts the SA. The algorithm also reuses data when possible.

The second matrix is completely buffered during execution. We buffer a few lines from the first matrix using a circular buffer while they are processed. The accelerator computes the result matrix line-by-line. When results are available, they are written to main memory. The algorithm timing diagram is shown in Figure 5.

Before the SA starts computing, the accelerator first reads *SA Height* lines of the first matrix and first *SA Width* columns of the second matrix. In parallel, the accelerator continues to read data from main memory.

IV. Experimental set up

In this section, we analyze the FPGA utilization for various SA dimensions. We define some metrics for the accelerator.

A. FPGA utilization

For the FPGA utilization, we use target AMD/Xilinx VCU128 FPGA. This board features [12]:

- 2852K logic cells;
- 8GB of High Bandwidth Memory (HBM) Dynamic RAM (Random Access Memory);
- 9024 DSP slices;
- 340.9 Mb of Block RAM + Ultra RAM.

We use Xilinx Vivado 2022.2 for synthesis. We employed Xilinx IPs for async FIFOs, memory banks and MAC. The MAC was replaced with a Digital Signal Processor (DSP) macro with 2 functions: $A \times B$ and $A \times B + C$. We replaced the partial result flip-flops with the one inside the DSP macro. Those changes were intended to allow the design to reach high clock frequencies.

To simplify the timing analysis, we create a wrapper which has three clocks: 500MHz for the core logic, 50MHz for configuration bus and 100MHz for memory buses.

B. Metrics

We defined two metrics: the number of requests per every memory controller and number of clock cycles for matrix multiplication. We measure these metrics on an HDL (Hardware Description Language) simulator. We measured the number of requests for every memory interface. This will help if we want to calculate the memory access time on various memory access delays and bus utilizations. The number of clock cycles for matrix multiplication starts when the first data is being read from the memory bank and stops when the last result is sent to the result FIFO. With this metric, we could calculate the runtime for this operation at various clock frequencies.

The simulation environment is composed of: a configuration module, memory emulators and a monitoring unit. A module will configure the accelerator, start the operation and wait for operation to complete. The memory emulator has two functions: for the input matrices to generate input data based on request address, and for the result matrix to follow the protocol and check the results. The simulation environment is illustrated in Figure 6.

For our baseline, we use GEM5 v23.1.0 [13] to simulate a single-core, four pipeline stages RISC-V processor without any extensions. The GEM5 cache configuration is presented in Table I. We implemented a matrix multiplication algorithm with $O(n^3)$ complexity in C and compiled it with *riscv-gnu-gcc* version 2024.04.12.

Cache	Type	Size [KB]	Latency [cycles]
L1 Instruction	2-way associative	16	2
L1 Data	2-way associative	64	2
L2	8-way associative	256	20

TABLE I
GEM5 cache configuration

V. Experimental results

Table II presents the synthesis results. The results are expressed as percentages of the total available resources. The number of Look Up Tables (LUTs) increases linearly with the SA size. The number of DSPs increases quadratically with the SA dimensions. The usage of BRAM does not vary so much because the FIFO capacities remain constant. The

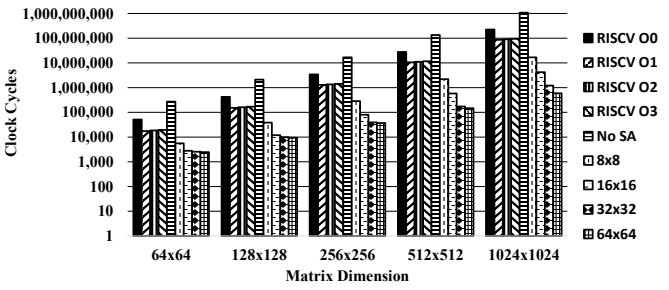

Fig. 7. Number of clock cycles for square input matrices.

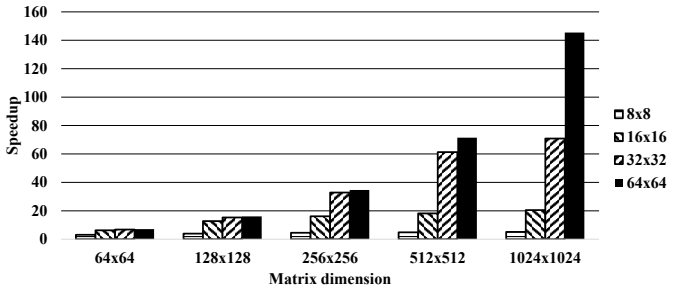

Fig. 8. Systolic Array spedup compared to RISC-V core GCC -O1

BRAM utilization increases at a low rate. A systolic array of 128x128 is not feasible because the number of DSPs available on the target FPGA board is too small.

Systolic Array	LUTs[%]	FF [%]	BRAM [%]	DSP [%]
8x8	0.94	1.45	2.03	0.73
16x16	1.90	2.93	2.13	2.86
32x32	4.05	5.40	2.33	11.35
64x64	9.71	12.89	2.60	45.39

TABLE II
FPGA UTILIZATION FOR MULTIPLE SYSTOLIC ARRAY DIMENSIONS.

Figure 7 shows results for matrix sizes ranging from 64×64 to 1024×1024, software implementations with GCC optimizations ranging from O0 to O3 and SA sizes ranging from 8×8 to 64×64. The O1 is the fastest GCC configuration.

The accelerator without SA (no SA), only one multiplication unit, is up to 1817 time slower than an implementation with SA. For example, on a large matrix of 1024×1024, the speedup is between 64 and 1817.

We analyze four systolic array dimensions: 8x8, 16x16, 32x32 and 64x64, and the two input matrices are square and have the same dimensions. The results are shown in Figure 7. When the size of the matrix is doubled in both directions (e.g., 16x16 to 32x32), the number of clock cycles increases by a factor of 8. When the systolic array size increases by a factor of 4, the clock cycles are reduced four times.

Our baseline implementation is the RISC-V processor with gcc -O1. As showed in Figure 8 the SA accelerator speedups the operation up to 145.5X times, for a 64×64 SA and 1024×1024 matrix size.

VI. CONCLUSIONS

We implemented a fully synthesizable matrix multiplier accelerator clocked at 500MHz, augmented with a Systolic Array dimensions ranging from 8×8 to 64×64. Our proposed data flow overlaps computations with the main memory accesses to reduce execution time. The experimental results suggest that our accelerator is up to 145.5X faster than a baseline RISC-V processor.

Future work includes integrating a RISC-V core and our matrix accelerator on the same FPGA, define the compiler support for the accelerator and compare our design against other matrix multiplication accelerators.

Acknowledgements This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI - UEFISCDI, project number PN-IV-P8-8.1-PME-2024-0022, within PNCDI IV.

The ISOLDE project, nr. 101112274 is supported by the Chips Joint Undertaking and its members Austria, Czechia, France, Germany, Italy, Romania, Spain, Sweden, Switzerland.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Sixth. Morgan Kaufmann, 2019, ISBN: 978-0-12-811905-1.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V edition: The hardware software interface*, Second. The Morgan Kaufmann, 2021, ISBN: 978-0-12-820331-6.

[3] H.-T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[4] N. P. Jouppi, C. Young, N. Patil, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[5] [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.

[6] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[7] C. Xin *et al.*, "Cosy: An energy-efficient hardware architecture for deep convolutional neural networks based on systolic array," in *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, 2017, pp. 180–189.

[8] H.-T. Kung *et al.*, "Mapping systolic arrays onto 3d circuit structures: Accelerating convolutional neural network inference," in *SiPS 2018*, IEEE, 2018, pp. 330–336.

[9] H. Kung *et al.*, "Systolic building block for logic-on-logic 3d-ic implementations of convolutional neural networks," in *ISCAS 2019*, IEEE, 2019, pp. 1–5.

[10] K. Inayat *et al.*, "Factored systolic arrays based on radix-8 multiplication for machine learning acceleration," *IEEE Transactions on VLSI Systems*, 2024.

[11] S. Choi *et al.*, "Savector: Vectored systolic arrays," *IEEE Access*, 2024.

[12] Xilinx, *Virtex UltraScale+ HBM VCU128 FPGA evaluation kit*. [Online]. Available: xilinx.com/products/boards-and-kits/vcu128.html.

[13] A. Bardsley, *gem5: Minor CPU Model*. [Online]. Available: https://www.gem5.org/documentation/general_docs/cpu_models/minor_cpu.