

Optimizing Matrix Multiplication on FPGAs using Spatial Parallelism and High-Level Synthesis

Henrique Gregory Gimenez¹, Edson Toshimi Midorikawa¹,
Felipe Valencia de Almeida¹, Liria Matsumoto Sato¹

¹ Departamento de Engenharia de Computação e Sistemas Digitais
da Escola Politécnica – Universidade de São Paulo (USP)

{henriquegregory, emidorik, fvalencia, liria.sato}@usp.br

Abstract. *This paper investigates the optimization of blocked matrix multiplication on the Alveo U50 FPGA using High-Level Synthesis (HLS). The proposed accelerator applies spatial parallelism by through up to four independent Compute Units (CUs) and is evaluated against a multi-threaded CPU implementation. Results show that PCIe transfer overhead limits FPGA performance for small workloads, but scalability becomes advantageous for large matrices. For 4096×4096 matrices, the 4-CU design achieves a $4.03\times$ speedup over a single-threaded CPU and matches the performance of an optimized 4-thread CPU implementation while consuming $2.36\times$ less power and dissipating $2.46\times$ less energy. These results highlight the FPGA as a power-efficient solution for large-scale matrix multiplication.*

1. Introduction

As hardware development grows increasingly complex, High-Level Synthesis (HLS) has become an important design methodology [Kastner et al. 2018]. By translating high-level languages such as C/C++ into hardware descriptions, HLS enables designers to focus on architecture rather than implementation details. FPGAs are ideal targets for HLS-generated designs due to their flexibility, parallelism, and energy efficiency.

This work extends the previous comparative study from [Gimenez and Midorikawa 2025] that evaluated baseline and blocked matrix multiplication algorithms on PYNQ-Z2 board. The objective of this paper is to investigate the blocked matrix multiplication by exploring spatial parallelism through the instantiation of multiple Compute Units (CUs) on the Alveo U50 FPGA.

2. Related Works

[Gorlani and Plessl 2021] applied HLS directives to implement a three-dimensional systolic array on Intel Stratix 10 FPGAs, showing how HLS can abstract the complexity of resources optimization. [Hosseiniabady and Nunez-Yanez 2020] used HLS optimizations, including loop pipelining and dataflow, to synthesize a streaming computation engine for embedded FPGAs, achieving higher energy efficiency than GPUs. More recently, [Favaro et al. 2026] proposed kernel-level HLS optimizations for sparse matrix multiplication on the Alveo U280 FPGA.

In contrast to these approaches, the present work uses the High-Level Synthesis tools not only to optimize the internal pipeline of blocked matrix multiplication but also to orchestrate spatial parallelism across the accelerator architecture.

3. Methodology

The FPGA design was synthesized using AMD Vitis/Vivado 2024.1 and evaluated against an optimized, block-based C implementation on an Intel Core i5-11400 CPU (up to 4.40 GHz) using OpenMP with 1, 2, and 4 threads. Automated via Python and the PyXRT API, the benchmark tested square matrices from 4×4 to 4096×4096 . Speedup, standard deviation, and 95% confidence intervals (via Student's t-distribution) were computed from five runs per configuration. Power consumption was obtained through the Vivado *Power Report* for the FPGA and the Linux `perf` command for the CPU.

Operating on integer matrices randomly generated via NumPy, the HLS kernel architecture processes 128×128 data blocks: an optimal size, as smaller blocks (64×64) severely degrade FPGA performance while larger ones (256×256) exceed available hardware resources. To maximize throughput, the `DATAFLOW` directive overlaps AXI4 memory transfers with computation (via AXI4 memory-mapped interfaces and internal FIFO streams), while `ARRAY_PARTITION` maps local matrices to individual registers, complemented by `PIPELINE` and `UNROLL` directives on nested loops. To further increase spatial parallelism, the architecture dynamically instantiates up to four independent Compute Units. Mapped to separate High Bandwidth Memory (HBM) banks, these CUs process independent matrix sub-blocks concurrently while reducing memory bandwidth bottlenecks. The hardware architecture of the proposed matrix multiplication accelerator is illustrated in Figure 1.

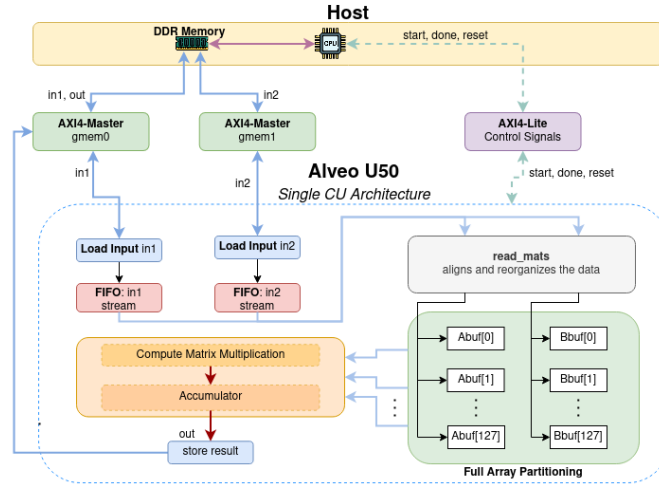


Figure 1. Hardware Accelerator Architecture of a single Matrix Multiplication Compute Unit (CU)

4. Experimental Results

The hardware resource utilization for the Alveo U50 platform is detailed in Table 1. The instantiation of four parallel CUs, alongside the base platform routing, consumes 255,653 LUTs and 1,540 DSP slices, leaving ample logic and arithmetic capacity on the device. With 85% of the available blocks consumed, Block RAM (BRAM) represents the highest resource utilization. This is a direct consequence of the array partitioning and internal block buffering required to sustain the dataflow pipeline and achieve an Initiation Interval (II) of 1. Thus, scaling beyond four CUs was not implemented, as it would exceed the available BRAM limit for the Alveo U50 FPGA board.

Table 1. Resource Utilization Comparison of Matrix Multiplication Across Different Configurations on Alveo U50

Resource	1 CU	4 CUs	4 CUs + Alveo U50 Platform	Available
LUT	25,485 (3.4%)	101,538 (13.6%)	255,653 (34.3%)	744,497
LUTASMEM	9,345 (2.4%)	37,380 (9.5%)	52,595 (13.4%)	392,749
REG	33,915 (2.2%)	136,437 (8.7%)	359,696 (22.8%)	1,576,025
BRAM	201 (17.3%)	804 (69.1%)	989.5 (85.1%)	1,163
URAM	0 (0%)	0 (0%)	4 (0.6%)	636
DSP	384 (6.5%)	1,536 (25.8%)	1,540 (25.9%)	5,948

Table 2 compares the execution times of the CPU (1, 2, and 4 threads) and the FPGA accelerator (1, 2, and 4 CUs) across various matrix sizes. For small workloads ($N \leq 128$), the CPU outperforms the FPGA due to the dominant PCIe transfer overhead. However, as the matrix size increases, the FPGA exhibits performance gains and spatial scalability, where increasing the number of CUs yields an almost linear reduction in execution time. At 4096×4096 , the 4-CU architecture achieves a $4.03\times$ speedup over the single-threaded CPU, matching the performance of the optimized 4-thread CPU baseline. Furthermore, statistical analysis for this peak workload indicates high execution stability, with a maximum relative error margin of 2.3% for the FPGA, compared to 5.5% for the CPU.

Table 2. Execution Time (ms) by Size and Hardware Configuration

Size	Avg FPGA (ms)			Avg CPU (ms)		
	1 CU	2 CUs	4 CUs	1 Thread	2 Threads	4 Threads
16×16	1.05	–	–	0.01	0.01	0.04
128×128	1.12	–	–	1.51	0.81	0.76
256×256	9.30	4.87	–	12.10	6.21	3.49
512×512	70.21	37.19	22.85	91.61	48.71	24.11
1024×1024	571.74	310.36	184.36	800.42	402.65	209.29
2048×2048	4447.23	2402.31	1473.26	6412.28	3179.43	1669.16
4096×4096	35279.14	19034.62	11547.17	46527.73	23476.43	12018.44

As illustrated in Figure 2(a), both platforms exhibit performance scalability for 4096×4096 matrices. The CPU achieves a $3.87\times$ speedup with four threads, while the FPGA yields a $3.06\times$ speedup with four Compute Units (CUs). Although the CPU exhibits slightly better linear scaling, the multi-CU FPGA architecture presents a substantial advantage in power and energy efficiency. As shown in Figure 2(b), CPU execution scales from 41.68 W / 1939.28 J (1 thread) to 70.08 W / 842.25 J (4 threads), whereas the FPGA scales from 21.70 W / 765.49 J (1 CU) to 29.69 W / 342.84 J (4 CUs). Beyond the figure, expanding the CPU to 8 and 12 threads reaches speedups of $4.56\times$ and $5.79\times$ versus a single CPU thread (dissipating 808.51 J), yet the 4-CU FPGA remains significantly more energy-efficient.

5. Conclusion

This paper showed that High-Level Synthesis (HLS) is an effective approach for accelerating block-based matrix multiplication on the AMD Alveo U50 FPGA. By leveraging spatial parallelism with up to four Compute Units, the accelerator achieved a $4.03\times$

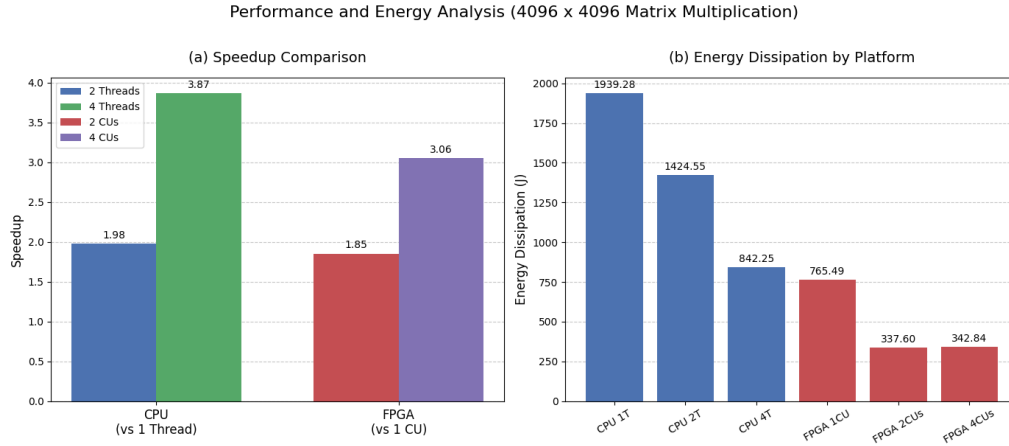


Figure 2. Performance and energy analysis for a 4096×4096 matrix multiplication: (a) Speedup comparison between CPU (thread scaling) and FPGA (CU scaling) and (b) Energy dissipation by platform.

speedup over a single-threaded CPU and matched the performance of an optimized 4-thread implementation for 4096×4096 matrices, while consuming $2.36 \times$ less power and achieving a $2.46 \times$ reduction in energy dissipation.. Although PCIe transfer overheads limited performance gains for small workloads, the scalability of the FPGA architecture showed advantageous for large-scale matrix computations. Source code, benchmarks, and full extracted data are available at: <https://tinyurl.com/yfmdxz23>

As future works, we intend to: (i) apply the spatial parallelism technique to accelerate the polynomial matrix multiplications of the Kyber Post-Quantum Cryptography scheme and (ii) explore the influences of HLS pragmas to analyze the trade-offs between hardware resource utilization and power efficiency.

References

- Favaro, F., Dufrechou, E., Oliver, J. P., and Ezzatti, P. (2026). Optimizing high-level synthesis sparse matrix-vector kernel for alveo fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1.
- Gimenez, H. and Midorikawa, E. (2025). Análise comparativa de síntese de alto nível para algoritmos de multiplicação de matrizes em fpga. In *Anais da XVI Escola Regional de Alto Desempenho de São Paulo*, pages 1–4, Porto Alegre, RS, Brasil. SBC.
- Gorlani, P. and Plessl, C. (2021). High level synthesis implementation of a three-dimensional systolic array architecture for matrix multiplications on intel stratix 10 fpgas.
- Hosseinabady, M. and Nunez-Yanez, J. L. (2020). A streaming dataflow engine for sparse matrix-vector multiplication using high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(6):1272–1285.
- Kastner, R., Matai, J., and Neuendorffer, S. (2018). Parallel programming for fpgas. *CoRR*, abs/1805.03648.