



Matrix Multiplication and High-Level Synthesis on FPGA Clusters

Author: Hawra Alsaif

Supervisors: Dr. Maynor Ballina and Dr. Romina Molina

Date: August 25, 2025

Abstract

This project explores matrix multiplication algorithms and their optimization for FPGA-based acceleration using High-Level Synthesis (HLS) tools. The work began with introductory training in VIVADO, Python fundamentals, Git, and machine learning, followed by in-depth experimentation with different matrix multiplication techniques including the Standard Algorithm, Block Matrix Multiplication, Strassen's Algorithm, Winograd's Algorithm, and Partitioned/Streaming approaches. Performance was analyzed on FPGA hardware using Vitis HLS with varying directives (PIPELINE, UNROLL, ARRAY_PARTITION) and compared to CPU-based implementations. Results highlight the trade-offs between time complexity, resource utilization, and memory usage across algorithms. While Strassen's Algorithm showed superior speed in CPU profiling, Block and Partitioned Multiplication proved more suitable for FPGA deployment due to their efficiency in handling resource constraints. The final implementation deployed optimized HLS designs on an FPGA cluster, validated correctness, and enabled multi-node execution for parallel performance evaluation.

Introduction

Matrix multiplication is a fundamental operation in scientific computing, machine learning, and signal processing. Its computational cost grows rapidly with matrix size, making acceleration crucial for large-scale applications. Field-Programmable Gate Arrays (FPGAs) offer parallelism and reconfigurability, enabling high-throughput, low-latency implementations of linear algebra operations.

This project focused on:

I. Learning FPGA design tools (Vivado, Vitis HLS).

II. Exploring matrix multiplication algorithms and their performance trade-offs.

III. Implementing and optimizing these algorithms on FPGA hardware using HLS.

IV. Comparing FPGA results to CPU-based executions.

Through this research, I gained insight into how algorithmic choices and HLS directives (such as pipelining, unrolling, and array partitioning) affect FPGA resource utilization and timing, and how these optimizations translate to practical deployment on FPGA clusters.

Methodology

Training and Setup

- Weeks 1–3**:** Orientation, Vivado labs, Python fundamentals, GitHub version control, and introduction to HLS.
- Learned how to synthesize C/C++ code into RTL using Vitis HLS and apply optimization directives.
- Conducted first experiments with a Vector Adder design using PIPELINE, UNROLL, and ARRAY_PARTITION directives.

Machine Learning Module

- Gained exposure to classification tasks using neural networks.
- Observed effects of parameter changes on accuracy and convergence.

Matrix Multiplication Research

Studied multiple algorithms:

I. Standard Algorithm ($O(n^3)$) – Simple and FPGA-friendly.

II. Strassen's Algorithm (O(n^2.81)) – Reduces multiplications but requires extra memory; less FPGA-suited.

III. Block Matrix Multiplication – Divides matrices into sub-blocks, improving memory efficiency.

IV. Winograd's Algorithm – Uses pre-computation, modest improvements over naive approach.

V. Streaming/Partitioned Multiplication – Processes matrices in streams, reducing memory pressure; ideal for FPGA deployment.

A comparative table of time complexity, memory usage, and FPGA suitability was compiled.

Implementation and Profiling

- Profiling was conducted on CPU (multi-core up to 500 matrix size) and FPGA (via Vitis HLS synthesis).
- Directives applied in HLS to optimize performance:
    - PIPELINE – reduced latency per operation.
    - UNROLL – parallelized loop iterations.
    - ARRAY_PARTITION/RESHAPE – improved memory access efficiency.
- Exported optimized HLS designs into Vivado for synthesis, bitstream generation, and FPGA deployment.
- Deployed FPGA cluster implementation through Jupyter Notebook, validating results against expected matrix products.

Results

CPU Profiling

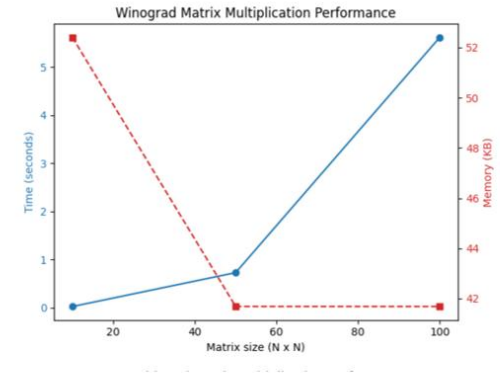
- Standard Algorithm: Longest runtime.
- Strassen's Algorithm: Fastest but required higher memory.
- Block and Partitioned Algorithms: Balanced trade-off between speed and memory efficiency.

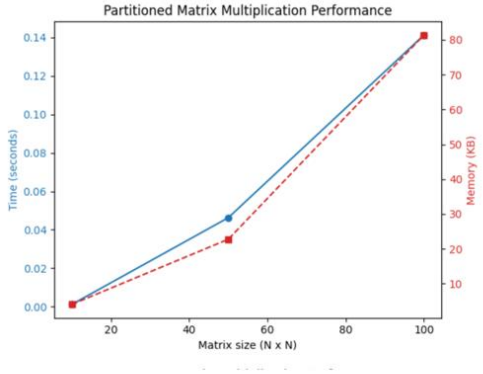
FPGA HLS Synthesis

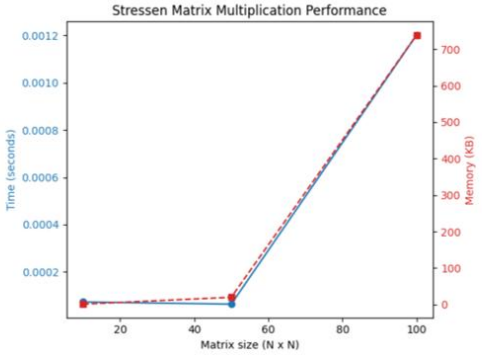
- PIPELINE directive: reduced cycle latency but increased resource usage.
- UNROLL: reduced latency but consumed more DSP and LUT resources.
- Optimal results achieved by applying PIPELINE only to core functions while minimizing unnecessary directives.
- Initial synthesis consumed >12,000 DSPs; optimized design reduced usage to 41 DSPs, enabling bitstream generation.

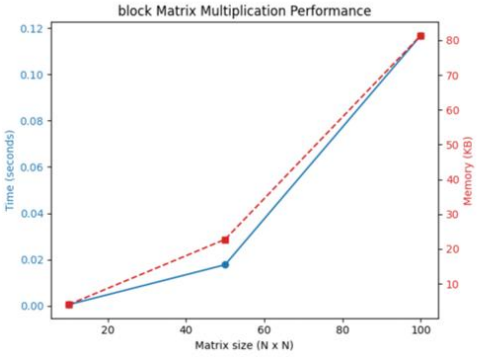
FPGA Cluster Deployment

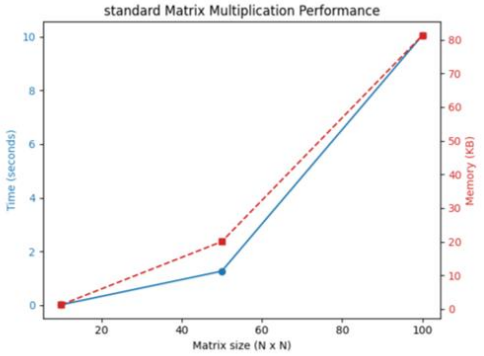
- Successfully implemented Standard and Partitioned Multiplication on FPGA clusters with matrix sizes up to 256.
- Block Matrix Multiplication encountered deployment issues beyond 128×128 due to incompatibilities in HLS-to-cluster workflow.
- Final implementation supported parallel execution across two FPGA nodes, reducing overall runtime.











Conclusion

This research highlights the interplay between algorithm design, HLS optimization, and FPGA resource constraints in implementing matrix multiplication. Key takeaways include:

- Partitioned/Streaming multiplication is the most FPGA-friendly algorithm due to its scalability and memory efficiency.
- Resource-aware directive application is critical; aggressive optimization may hinder synthesis.
- FPGA clusters enable scalable acceleration but require consistent design verification across tool chains (HLS → Vivado → Jupyter).

Future work can extend to sparse matrix multiplication, improved memory management, and integrating systolic array architectures for higher throughput.

References

I. Vivado Design Suite User Guide. Xilinx, 2024.

II. Xilinx Vitis High-Level Synthesis User Guide. Xilinx, 2024.

III. Strassen, V. “Gaussian Elimination is Not Optimal.” Numerische Mathematik, 1969.

IV. Winograd, S. “On Multiplication of 2×2 Matrices.” Linear Algebra and Its Applications, 1971.