FPGA-Based Acceleration of Matrix Multiplication Using a PYNQ Heterogeneous Computing Framework

Donovan Jones, Suxia Cui, Lujun Zhai, Prairie View A&M University,
Mubbashar Khan, Central State University,
Shujun Yang, Xiang Zhao, Alabama A&M University

Abstract

❖ This work looks at using FPGAs to speed up compute-heavy tasks through a mixed computing setup on the PYNQ-Z1 platform. A custom matrix-multiplication accelerator is created using High-Level Synthesis (HLS) and installed as an FPGA overlay. The system uses AXI memory-mapped interfaces and Direct Memory Access (DMA) to allow smooth data transfer between the ARM processor and the programmable logic. Performance is measured by comparing the FPGA implementation to a CPU-based NumPy implementation with different matrix sizes. The experimental results show that execution time improves because of parallel hardware execution. They also reveal how communication overhead affects the overall system performance. This study highlights key trade-offs in hardware and software design for FPGA-accelerated computing.

Introduction & Background

❖ The rapid growth of artificial intelligence and data-intensive applications has led to a higher demand for high-performance computing solutions that can efficiently handle large-scale numerical operations. Many of these applications depend on matrix multiplication, which is a task that requires significant computation and benefits from parallel execution. While central processing units (CPUs) offer flexibility and are easy to program, they often struggle to fully exploit fine-grained parallelism. Field-Programmable Gate Arrays (FPGAs) provide a promising alternative by allowing the creation of custom hardware architectures designed for specific algorithms. Unlike general-purpose processors, FPGAs enable designers to develop highly parallel data paths, which improves performance for targeted workloads. Modern FPGA platforms, like the PYNQ-Z1, support heterogeneous computing by combining an embedded processor with programmable logic. This integration allows software and hardware to work together in a single system. The PYNQ framework makes FPGA development easier by enabling hardware control through Python, making it more user-friendly for software-oriented users. Communication between the processor and FPGA occurs through AXI-based interfaces and Direct Memory Access (DMA), which streamlines data transfer. This research investigates FPGA-based acceleration by implementing and evaluating a matrix-multiplication accelerator in this heterogeneous computing setup.

Motivation

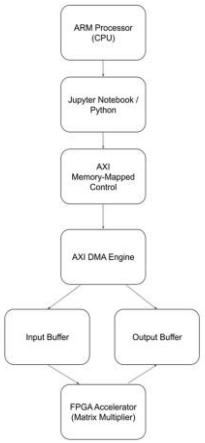
❖ The rising demands of artificial intelligence and data-heavy applications have revealed the limits of traditional CPU systems, especially for tasks that require many parallel operations, like matrix multiplication. FPGAs can deliver significant performance benefits through their customizable parallel hardware, but their use is often constrained by complex design and development processes. This creates a disconnect between the potential of FPGA acceleration and its real-world application. The goal of this research is to close this gap by exploring an easier way to use FPGA acceleration through the PYNQ platform. By using Python-based controls and high-level design tools, this work shows how hardware and software can be effectively designed together. The study aims to offer practical insights into speeding up compute-intensive tasks while also highlighting the trade-offs in mixed computing systems.

Methodology

❖ **System Architecture and Setup**: A diverse computing system is built on the PYNQ-Z1 platform, which combines an ARM processor with FPGA programmable logic. The processor runs Linux and executes Python programs through Jupyter Notebook. The FPGA fabric implements custom hardware accelerators. The PYNQ framework enables easy interaction with the FPGA using Python, simplifying system control. Communication between software and hardware happens through AXI memory-mapped interfaces for control and AXI Direct Memory Access (DMA) for effective data transfer between system memory and the FPGA

❖ **Hardware Accelerator Design:** A matrix multiplication accelerator is built using High-Level Synthesis (HLS) in C/C++. The design implements the standard multiply-accumulate operation and is improved for parallel execution via loop pipelining and loop unrolling. First, functional correctness is checked through C-level simulation before synthesis. Next, the design is turned into a hardware IP core and integrated into a Vivado block design with AXI interfaces. Finally, a bitstream and a hardware description file are created for deployment onto the FPGA.

System Architecture of the PYNQ-Based FPGA Acceleration Framework



FPGA Acceleration Workflow for Matrix Multiplication

❖ **Software Integration and Execution:** The generated bitstream is loaded onto the FPGA with the PYNQ Python framework. Python scripts control the accelerator, allocate contiguous memory buffers, and initialize input data with NumPy. Input matrices are transferred to the FPGA via DMA, where the accelerator performs parallel matrix multiplication. The results are then sent back to system memory. Synchronization between the processor and FPGA is maintained using DMA control signals to ensure correct execution.

❖ **Performance Measurement and Benchmarking:** Performance is evaluated by comparing the FPGA-based implementation with the CPU-based version using NumPy. Execution time is measured using Python timing functions, which include both computation and data transfer delays. Experiments are conducted with different matrix sizes, including 8×8, 32×32, 64×64, and 128x128 to test scalability. The CPU implementation acts as a baseline for measuring performance gains from hardware acceleration.

❖ **Analysis and Validation:** The results are examined by comparing execution times and calculating speedup as the ratio of CPU execution time to FPGA execution time. This analysis looks at the benefits of parallel computation on FPGA hardware and how data transfer overhead affects overall performance. Additionally, Vivado synthesis reports evaluate hardware resource use, including LUTs, DSP blocks, and BRAM. We verify correctness by comparing FPGA outputs with NumPy results, ensuring proper computation across all test cases.

Expected Results

❖ The FPGA-based matrix multiplication accelerator is expected to show better performance than the CPU-based NumPy implementation, especially for larger matrix sizes. The parallel execution capabilities of the FPGA hardware should reduce computation time as the matrix size grows, leading to a noticeable speedup compared to the CPU. However, performance improvements may be limited for smaller matrices due to the FPGA. As the workload size increases, the benefits of parallelism are likely to exceed the costs of communication, leading to greater performance gains. We also expect increased use of FPGA resources, like DSP blocks. These results will reveal important trade-offs between computation efficiency and data transfer overhead in mixed computing systems..

Discussion

❖ The expected performance improvements from FPGA-based acceleration mainly come from programmable logic's ability to take advantage of parallelism with custom hardware designs. Unlike CPU-based systems that process instructions one after the other, the FPGA accelerator can carry out several multiply-accumulate operations at the same time. This parallel execution is especially useful for matrix multiplication, where separate operations can be computed in parallel. This reduces computation time as the matrix size grows.

❖ However, overall system performance is influenced not only by computation speed but also by data transfer overhead between the processor and the FPGA. Communication through AXI DMA introduces latency that can limit performance gains, especially for smaller matrix sizes where transfer time may dominate total execution time. This highlights a key trade-off in heterogeneous computing systems: while FPGAs can significantly accelerate computation, the efficiency of data movement is critical to overall system performance.

❖ As matrix sizes grow, the computational workload becomes more significant. This means the advantages of parallel hardware execution can outweigh communication costs. It indicates that FPGA acceleration performs best for large, compute-intensive tasks common in artificial intelligence and signal processing applications. We also need to consider hardware resource usage when looking at system efficiency. More parallelism often demands greater use of FPGA resources, such as DSP blocks and logic units. This can limit scalability due to device limitations.

❖ Overall, this work highlights the need to balance computation, communication, and resource use in hardware/software co-design. Understanding these trade-offs is crucial for effectively using FPGA-based acceleration in real-world systems.

Acknowledgement

❖ This research is partially supported by the National Science Foundation under Grant CNS-2411979, EES-2436203; Apple NSI; and the Prairie View A&M University Faculty RISE Program.