FPGA-Based Design and Realization of Fixed and Floating Point Matrix Multipliers: A Review

SYED MANZOOR QASIM*, SHUJA AHMAD ABBASI AND BANDAR ALMASHARY

Department of Electrical Engineering, College of Engineering, King Saud University, Riyadh 11421, Saudi Arabia

Matrix multiplication is a computationally-intensive and fundamental matrix operation in many algorithms used in scientific computations. It serves as the basic building block for many signal, image processing, graphics and robotic applications. To improve the performance of these applications, a high performance matrix multiplier is required. Traditionally, matrix multiplication operation is either realized as software running on fast processors or on dedicated hardware (Application Specific Integrated Circuits (ASICs)). Software based matrix multiplication is slow and can become a bottleneck in the overall system operation. However, hardware (Field Programmable Gate Array (FPGA)) based design of matrix multiplier provides a significant speed-up in computation time and flexibility as compared to software and ASIC based approaches respectively. In this paper, we present a study of reported work from past ten years, covering various aspects of FPGA-based design and realization of matrix multipliers. A study of fixed and floating-point designs of matrix multiplier is also presented. Finally, some applications where matrix multiplication is the core operation are highlighted and discussed.

Keyword: Field Programmable Gate Array (FPGA), Hardware Realization, Matrix Multiplier, Review.

1 INTRODUCTION

The advancement of Field Programmable Gate Arrays (FPGAs) in the recent years, allowing multimillion gates on a single chip, together with sophisticated
Electronic Design Automation (EDA) tools has allowed the implementation of complex and computation-intensive algorithms in efficient and cost-effective way. During the last decade, the logic density, functionality and speed of FPGA have improved considerably. Modern FPGAs are now capable of running at speed of 500 MHz and beyond [1]. Another important feature of FPGAs is their potential for dynamic reconfiguration [2]; that is, reprogramming part of the device at run time so that resources can be reused through time multiplexing.

The increased interest in FPGAs for real-time applications, such as wireless communications, image processing and image reconstruction, medical imaging, network security, and signal processing justifies the effort in the design of efficient and high-performance matrix multiplier where either fixed or floating point real number representations are exploited for calculations [3]. Since matrix multiplier is a key primitive in many such applications, the use of high performance matrix multiplier will thus ameliorate the performance of these computations.

The computational complexity of matrix multiplication algorithm is $O(n^3)$ on a sequential processor and $O(n^3/p)$ on a parallel system with $p$ processors. Traditionally, matrix multiplication operation is often performed by parallel processing systems which distribute computations over several processors to achieve significant speedup gains. There are many realizations of matrix multiplication. These realizations mainly differ in terms of algorithms or the hardware platforms. During the last few years, research efforts towards realizing and accelerating the matrix multiplication operation using reconfigurable hardware (FPGA) have been attempted [4–22].

FPGA-based design and realization of double precision floating-point matrix multiplier is an emerging area of research because of its application in many scientific computing applications. Research work related to the realization of matrix multiplier on FPGAs are scattered throughout the literature. In this paper, an attempt is made to organize and present a survey of reported work from past ten years dealing with various aspects of FPGA-based design and realization of matrix multipliers.

The remainder of this paper is structured as follows. Section 2 discusses the important features of FPGA. Section 3 provides a survey of various techniques and approaches for the design and realization of fixed and floating-point matrix multiplier on FPGAs. A brief description of the application domains where the matrix multiplier plays an important role is provided in section 4. Finally, section 5 summarizes the conclusions.

### 2 FPGA OVERVIEW

FPGAs have emerged to be a mainstream technology for realizing digital systems. It is defined as a matrix of logic elements which can be interconnected in any desired configuration to implement a given application. An
FPGA is generally composed of three types of elements: configurable logic blocks (CLBs), input/output blocks (IOBs), and programmable interconnects. In addition, advanced FPGA architectures include dedicated blocks such as RAM, hardwired multipliers, multiply-accumulate unit, high-speed clock management circuitry, and serial transceivers, embedded hard processor cores such as PowerPC or ARM, and soft processor cores such as NIOS or Microblaze [1].

FPGAs offer the design flexibility of a software and speed of hardware (ASICs). Early FPGA devices were mainly used for fixed-point applications. However, with rapid advances in VLSI technology, current FPGAs contain enough hardware resources required for implementing floating point operations [22].

In order to realize an algorithm on an FPGA it must be programmed first. To achieve this, a design methodology is adopted. Usually, design entry is done using schematic or hardware description language (HDL) such as Very High Speed Integrated Circuit HDL (VHDL) or Verilog. The objective is to make the system description independent of the physical hardware such that it can be used on other FPGAs and even on ASICs. Once a design has been completed it is simulated to verify the correct operation. A vendor specific netlist file which is essentially text based descriptions of the schematic is generated from the design and is mapped onto the FPGA using synthesis, place and route and optimization tools. Mapping produces a bit-stream file that is used to program the FPGA.

3 DESIGN OF MATRIX MULTIPLIERS

FPGA based designs are usually evaluated using three key performance indicators (KPIs): speed (latency), area, and power (energy). Fixed point implementations in FPGA are fast and consume less power. Additionally, a fixed point matrix multiplier requires fewer gates in an FPGA or ASIC than its floating-point counterpart. However, the limitation of fixed point number is that very large and very small numbers cannot be represented and the range is limited to bit-width of the number. Extensive previous work has been done towards the design and realization of an FPGA based system for the computation of fixed point matrix multiplication.

3.1 Fixed Point Matrix Multiplier

Amira et al. presented a high throughput architecture based on systolic array for bit level matrix multiplication [4]. Baugh-Wooley algorithm is adopted for the design of serial-parallel matrix multiplier. The design based on the systolic array has been realized on a Xilinx XCV1000E Virtex-E FPGA.

Amira et al. designed a parameterizable system for 8-bit fixed point matrix multiplication using FPGA [5]. Their design used both systolic architecture
and distributed arithmetic design methodology for the implementation of matrix multiplication. The architecture proposed in this paper was targeted to Xilinx XCV2000E of Virtex-E FPGA family. The results presented in this paper showed better performance than the architecture presented in [4] in terms of area and speed.

For \( n = 4 \), distributed arithmetic based design used 57 Slices as compared to 72 slices used in [4] and operated at a maximum frequency of 166.47 MHz as compared to 58.302 MHz used in [4]. Distributed Arithmetic based design provides better performance in terms of speed and area as compared to systolic array based design. The I/O bandwidth required by the design is directly proportional to the problem size. The designs presented in [4–5] were restricted to smaller matrices.

For multiplying large matrices (\( n = 128, 256 \) and 512), Bensaali et al. designed an FPGA based partitioning engine [6]. The designed partitioning engine partitions the input matrices into smaller sub-matrices. Partial products generated from these sub-matrices are stored in buffer. Finally, all the partial products are accumulated to get the final result.

In [7], Mencer et al. implemented the matrix multiplication on Xilinx XC4000E FPGA. Their designs employ bit serial multipliers using Booth encoding. They focused on tradeoffs between area and maximum running frequency with parameterized circuit generators. Their design was improved by Amira et al. in [8] using modified booth encoder multiplication along with Wallace tree addition. For \( n = 4 \), 296 CLBs were used to achieve a maximum operating frequency of 60 MHz using Xilinx XCV1000E FPGA.

Jang et al. improved the design in [7] and [8] in terms of area, speed [9] and energy [10] by taking advantage of data reuse. They reduced the latency for computing matrix product by employing internal storage registers in the processing element (PE). Their algorithms need \( n \) multipliers, \( n \) adders, and total storage of size \( n^2 \) words. For \( 4 \times 4 \) matrix multiplication, the latency of the design in [7] is 0.57 \( \mu \)s, while the design in [9–10] uses 0.15 \( \mu \)s utilizing 18% less area as compared to [7].

The design and implementation of a high performance, fully parallel matrix multiplier core was presented by Belkacemi et al. [11]. The designed core is parameterized and scalable in terms of the matrix dimensions (i.e., number of rows and columns) and the input data word length. Fully floor planned FPGA configurations are generated automatically, from high-level descriptions of the matrix multiplication operation, in the form of EDIF netlist in less than 1 sec. These are specifically optimized for Xilinx Virtex FPGAs. By exploiting the abundance of logic resources in Xilinx Virtex FPGAs (LUTs, fast carry logic, shift registers, flip flops etc.), a fully parallel implementation of the matrix multiplier core is achieved; with a full matrix result being generated every clock cycle. A \( 3 \times 3 \) matrix multiplier instance consumes 2,448 Virtex slices and can run at 175 MHz on an XCV1000E-6 Virtex-E chip.
Traditionally, the KPIs for FPGA based designs have been speed and area. However, with the proliferation of portable, mobile devices, it has become increasingly important that the systems are also energy efficient and consume less power. In FPGA devices, major chunk of power is consumed by the programmable interconnects, while the remaining power is consumed by the clocking, logic, and I/O blocks. Another source of power dissipation in FPGAs is resource utilization and switching activity [12]. Research efforts towards the design of energy efficient matrix multiplier have been reported in [10], [13–15].

Most of the previous work in fixed point matrix multipliers focused only on increasing the speed and reducing the area. Choi et al. developed novel designs and architectures for FPGAs which minimized the power consumption along with latency and area [14–15]. They used linear systolic architecture to develop energy efficient designs. For linear systolic array, the amount of storage per processing element affects the system wide energy. Thus, they used maximum amount of storage per processing element and minimum number of multipliers to obtain energy-efficient matrix multiplier.

Partial reconfigurability feature was exploited for the first time for the computation of matrix multiplication by Jianwen et al. in [16]. Partially reconfigurable devices offer the possibility of changing the design implementation without stopping the whole execution process. The matrix multiplier was implemented in Xilinx Virtex-II device, which supports partial reconfiguration. The design was evaluated in terms of latency and area and it was found that area is reduced by 72% – 81% for matrix sizes between $3 \times 3$ and $48 \times 48$ as compared to [9] and the performance further improves for larger matrices.

### 3.2 Floating Point Matrix Multiplier

Many high performance computing applications such as weather forecasting, computational fluid dynamics etc. require high performance floating point matrix multipliers. Previously, implementing floating point matrix multiplier in FPGAs was not feasible because it consumed a major portion of the resources available on FPGAs [17]. However, with the recent advancement made in FPGA technology [18], it is feasible to implement floating point matrix multipliers. The performance metric used for floating point matrix multiplier is Mega floating point operations per second (MFLOPS) and GFLOPS. Research efforts to achieve this have started recently and some promising results have been reported in literature [19–22]. In this sub-section, we will provide a survey on the state-of-the-art FPGA implementation of floating point matrix multiplier.

Research initiative was started by the authors in [19] who studied the effect of the floating point multiplier and adder on the performance of matrix multiplication. It was concluded that floating point matrix multiply requires more space than fixed point matrix multiply because the floating point adder
consumes more resources than the fixed point adder. They used four 4062XL-3 FPGAs achieving a performance of 192 MFLOPS for multiplying $1024 \times 1024$ matrices as compared to a theoretical peak of 264 MFLOPS.

In [18] Underwood et al. discussed matrix multiplication as part of the BLAS (Basic Linear Algebra Subprograms). They studied the performance of FPGA on computing double precision (64-bit) floating point matrix multiplications and compared it with that of general purpose processors.

Dou et al. proposed a parallel algorithm for matrix multiplication in [20], which consists of master processor and multiple slave processors. The master controls the slave processors. It distributes the data from the source matrices to the slaves and each slave computes different blocks of the resultant matrix in parallel. This algorithm is suitable for matrices of large dimensions.

The authors of [22] extended the architecture in [9] for floating point matrix multiplication. They proposed two algorithms in which the number of floating point units and the storage size are independent of the problem size. In [22], Zhuo et al. provided parameterized and optimized designs for floating point matrix multiplication. The design was implemented on Xilinx Virtex-II Pro device and Cray XD1 high performance reconfigurable computing system. The designs achieved a sustained performance of 2.06 GFLOPS on a single node of XD1. It was concluded that the algorithms are suitable for matrix multiplication of various sizes and achieve floating-point performance comparable with general-purpose processors.

4 APPLICATION DOMAINS

There are many applications which can be expressed as a matrix multiply or a subset of matrix multiply. Some of these applications include image and signal processing which are summarized as follows:

4.1 Linear Back-Projection
Linear back-projection (LBP) is an image reconstruction algorithm used in Electrical Capacitance Tomography (ECT). LBP algorithm is formulated as a matrix-vector multiplication. It is a computationally intensive process because it involves multiplication of large matrix with a vector [23].

4.2 Color Space Conversion
Color space conversion is very important in many types of image processing applications including video compression. This operation consumes up to 40% of the entire processing power. Color space conversion can be expressed as matrix-vector multiplication. Techniques which efficiently implement this conversion are desired [24].
4.3 3D Affine Transformation

3D affine transformations are the transformations that involve rotation, scaling, shear and translation [25]. A matrix can represent an affine transformation and a set of affine transformations can be combined into a single overall affine transformation. 3D affine transformations are modeled using large matrix multiplication.

4.4 Estimation of Higher-Order Cross Moments

Higher-order cross moments are commonly used in digital signal processing. It has a wide applicability in many fields such as sonar, radar, seismic data processing, adaptive filtering, blind equalization, array processing, speech and image processing, motion estimation and biomedical signal processing [26]. The demand for efficient architectural designs for fast computation of higher-order cross moments is becoming increasingly significant as these statistics grow in importance as a signal processing tool. For fast estimation of these statistics, the problem is formulated as a series of matrix multiplication operation to take advantage of the inherent parallelism of matrix multiplier and parallel hardware.

4.5 Time Frequency Spectral Analysis

The problem of analyzing nonstationary signals has attracted considerable attention during the past three decades due to its large number of applications in diverse fields such as speech and image processing, biostatistical signal processing, radar, medicine, and seismology. For each application, the analysis results in a set of mathematical equations that can be used to understand the behavior of the signal. For fast estimation of time frequency spectrum [27], the problem is formulated as a series of matrix multiplication operation to take advantage of the inherent parallelism of matrix multiplier and parallel hardware.

4.6 Wireless Communication

Space-time block code (STBC) is used to improve the reliability and throughput of wireless communication systems. The channel estimation of STBC involves matrix multiplication operation [28]. Beamforming is another application where matrix multiplier is the core operation and plays a very important role in the overall system design [29].

5 CONCLUSIONS

In this paper we have presented a survey on the contributions of FPGA in realizing fixed and floating point matrix multipliers. An extensive literature survey covering various aspects of design and realization of matrix multiplication operation has been presented. It is concluded that by utilizing the
special features of advanced FPGAs, the computation time, hardware resource utilization, and power consumption can be significantly reduced. The current research trend is towards realizing a high-performance double-precision floating point matrix multiplier on FPGA.

ACKNOWLEDGMENT

The authors would like to thankfully acknowledge the financial support provided by the Research Center of College of Engineering at King Saud University.

REFERENCES


