

# High Speed Matrix Multiplication Implementation Using Field Programmable Gate Array

**Shriyashi Jain**

M.TECH (DC)  
SSSIST, Sehore, M.P., India

**Neeraj Kumar**

Department of Electronics and  
Communication,  
SSSIST, Sehore, M.P. India

**Jaikaran Singh**

Department of Electronics and  
Communication,  
SSSIST, Sehore, M.P. India

**Mukesh Tiwari**

Department of Electronics and  
Communication,  
SSSIST, Sehore, M.P. India

## Abstract

Matrix operations are commonly used in almost all areas of scientific research. Matrix multiplication has significant application in the areas of graph theory, numerical algorithms, signal processing, and digital control. Matrix multiplication is a computationally intensive problem, especially the design and efficient implementation on an FPGA where resources are very limited, has been more demanding. In this paper, we implement an architecture that is capable of handling matrices of variable sizes. This design minimize the gate count, area, improvements in latency, computational time, throughput for performing matrix multiplication and reduce the number of multiplication and additions hardware required to get the matrices multiplied on commercially available FPGA devices. The hardware design in our work to multiply two numbers is use the multiplier unit used for multiplying two numbers in a single clock cycle. This increases the speed of the computation. The system is simple to implement and is highly scalable, the system can be scaled with simple repetition of the hardware and with no changes in the algorithm. Our approach converts matrix multiplication in programmable processors into a computation channel, when increasing the processing throughput, the output noise (error) increases due to computational errors caused by exceeding the machine-precision limitations.

**Keywords**—Multiplier, Xilinx Software, Vhdl Language, FPGA, Latency.

## Introduction

Matrix multiplication is frequently used operation in a wide variety of graphics, image processing, robotics, and signal processing applications. The increases in the density and speed of FPGAs make them attractive as flexible and high-speed alternatives to DSPs and ASICs. It is a highly procedure oriented computation, there is only one way to multiply two matrices and it involves lots of multiplications and additions. But the simple part of matrix multiplication is that the evaluation of elements of the resultant elements can be done independent of the other, these points to distributed memory approach.

In this paper, we propose an architecture that is capable of handling matrices of variable sizes our designs minimize the gate count, area, improvements in latency, computational time, and throughput for performing matrix multiplication and reduces the number of multiplication and additions hardware required to get the matrices multiplied on commercially available FPGA devices. The hardware design in our work to multiply two numbers is use the multiplier unit used for multiplying two numbers in a single clock cycle. This increases the speed of the computation. The system is simple to implement and is highly scalable, the system can be scaled with simple repetition of the hardware and with no changes in the algorithm.

## Methodology adopted

Our approach converts matrix multiplication in programmable processors into a computation channel, when increasing the processing throughput, the output noise (error) increases due to computational errors caused by exceeding the machine-precision limitations. Let  $n = qp$ , where  $n$  is the number of rows of matrix  $A$ ,  $p$  is the number of processors, and  $q$  is an integer. Matrix  $A$  has been partitioned into  $p$  regions with each region containing  $q$  rows and being assigned to the local-memory (LM) of each processor. Matrix  $B$  is made available to all the processors. The data-partitioning scheme is similar to the shared-memory approach. The differences are the extra time required for data distribution/collection via message passing and the fact that all computations are done in the LM of each processor with no memory-access conflict involved.

In this implementation, it is assumed that only processor 0 has access to matrix  $A$  and  $B$ . Processor 0 acts as a host processor responsible for broadcasting the needed data to each of the other processors and waiting for the vector results from the other processors.[1][2]

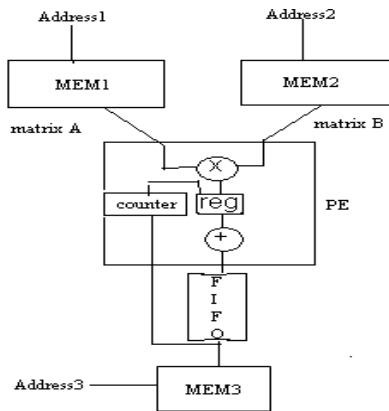
These methods have focused on using complex PE's to both store and transfer intermediate results throughout the processing array. While this improves the latency of the system, it is not delay or resource optimal due to data sharing between the PE's. To the

best of our knowledge, our method is the first to perform matrix multiplication with PE's that operate in isolation from each other.

The unit has following main blocks

1. Memory blocks
2. Control unit
3. Matrix arrangement of multiplier units
4. Adders and counters

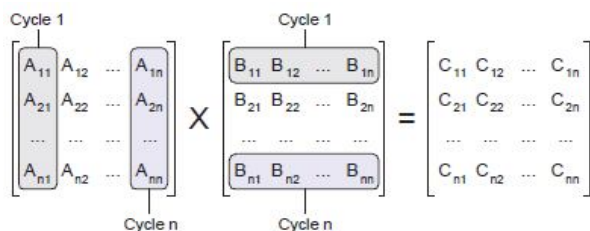
The architecture for the matrix multiplier is as shown below,



In order to optimize FPGA architecture resource use, the data from input matrices A and B should be re-used. Optimal data re-use occurs when data is read from memory for matrices A and B exactly once. By simultaneously reading one column of matrix A and one row of matrix B, and performing all multiply operations based on those values before additional memory reads, optimal data re-use occurs. Data read in this sequence allows one partial product term of every element in output matrix C to be computed per clock cycle.

## Design Implementation

The block diagram of our design consists of three memory blocks. The matrix A and Matrix B is stored in memory 1 and memory2 respectively depends on the address line. This matrix bytes are use for matrix multiplication. Here we can use distributed memory approach. In a distributed-memory system, each processor has only local memory, and information is exchanged as messages between processors. In contrast, the processors in a shared-memory system share a common memory. Although data is easily accessible to any processor.



The latency is defined as the time between reading the first elements from the input matrices, A and B, and writing the first element C to the result matrix. The total computation time is the

time elapsed between reading the first elements from the input matrices, A and B, and writing the final result matrix element C to memory. The PE structure consists of one input each from matrix A and B, a multiplier, adder and a result FIFO. The multiplier latency is denoted as  $L_m$ , adder latency as  $L_a$ , processing element latency as  $L_{pe}$  and computational time of matrix multiplier is  $T_m$ . The inputs from matrices A and B, containing one byte each per clock cycle, are implemented using dedicated routes from the Block Ram memory associated with the multiplier.

By having dedicated memory connections for each PE, multiplexing between several inputs sources is not required. During the computation of output matrix element  $C_{ij}$  the product term  $A_{ik} \cdot B_{ki}$  must be available at the output of the adder during the same clock cycle as the product term  $A_{i(k+1)} \cdot B_{(k+1)i}$  is available from the multiplier. The multipliers give the multiplied output at the end of the ON time of the current clock cycle, these outputs have to be added to get the resultant element. For handling a  $3 \times 3$  matrix the multiplier outputs have to be added column wise and we get one resultant element at the end of addition. Our design requires only one processing element because our method utilizes the built-in hardware FIFOs in the FPGA, and also because we utilize the same Block Ram for the local PE memory and for storing the result matrix C. In the multiplication of two  $N \times N$  matrices the evaluation of each resultant element results in  $N$  number of multiplications and  $N-1$  additions. If we were to handle matrices of order with a maximum order of  $N \times N$ , we require the  $N$  number of multiplications that go into the evaluation of the elements. This is achieved by setting up the counter, so that on every evaluation of partial sum it is stored into the register. The counter is decremented every clock cycle and if the value is not zero then the enable signal for the feedback buffer is high, hence the partial sum is added along with fresh set inputs. If the counter decrements to zero then all the  $N$  multiplications have been taken into account and hence the output buffer is enabled while the feedback is disabled, for the next clock cycle both the buffers are disabled so that the partial sum of next number is loaded into the register. We have taken a separate counter in this block for simplicity of understanding, the control of the buffers can easily be done by the control circuit itself.

- Enter the data to be process in distributed memory as per address.
- Read the individual row elements of first matrix and that of column for second matrix
- 

```

PROCESS(clk)
BEGIN
IF (clk = '1' AND clk'EVENT) THEN
IF (we = '1') THEN
mem1(conv_integer(addr1))<= din1;
mem2(conv_integer(addr2))<= din2;
ELSE
dout1 <= mem1(conv_integer(addr1));
dout2 <= mem2(conv_integer(addr2));
END IF;

```



provide a throughput improved frames per second which is sufficient for many image and video processing applications. Finally, we conclude that for multiplication of large matrices, memory based architecture is quite efficient whereas, for small and medium sized matrix multiplication, systolic array techniques prove to be quite efficient as demonstrated by the implementation results..

## **References**

- [1] Shu-Qing Li, Chi Hou Chan, Leung Tsan "Parallel Implementation of the Sparse-Matrix/Canonical Grid Method for the Analysis of Two-Dimensional Random Rough Surfaces (Three-Dimensional Scattering Problem) on a Beowulf System" IEEE Transactions On Geoscience And Remote Sensing, Vol. 38, No. 4, July 2000
- [2] Nan Zhang "A Novel Parallel Scan for Multicore Processors and Its Application in Sparse Matrix-Vector Multiplication" IEEE Transactions On Parallel And Distributed Systems, Vol. 23, No. 3, March 2012
- [3] Bahram Hamraz, Nicholas HM Caldwell, and P. John Clarkson "A Matrix-Calculation-Based Algorithm for Numerical Change Propagation Analysis" IEEE Transactions On Engineering Management, Vol. 60, No. 1, February 2013
- [4] Vasileios Karakasis, Theodoros Gkountouvas, Kornilios Kourtis, Georgios Goumas, Nectarios Koziris "An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication" IEEE Transactions On Parallel And Distributed Systems- 2013