International Journal of Computer Applications (0975 – 8887)
Volume 86 – No 8, January 2014

FPGA Implementation of Latency, Computational Time Improvements in Matrix Multiplication

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, like matrix multiplication, 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. FPGA based designs are usually evaluated using three performance metrics: speed (latency), area, and power (energy). Fixed point implementations in FPGA are fast and have minimal power consumption. With today's applications requiring ever higher computational throughputs, distributed memory approach is an effective solution for real-time applications. This application shows how to achieve higher computational throughput via parallel processing with the DSP processors. The matrix-vector multiplication applied to calculate linear convolution. This paper presents an FPGA-based hardware realization of matrix multiplication based on distributed memory approach architecture. 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.

Keywords
Latency, computational throughput, gate count, field-programmable gate array (FPGA).

1. 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 field-programmable gate arrays (FPGAs) [1] make them attractive as flexible and high-speed alternatives to DSPs [3] and ASICs. It is a highly procedure oriented computation [6], 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, this point 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, 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.

2. METHODOLOGY ADOPTED
Our approach change 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 A is a matrix of n rows where n = qp, p is the number of processors, and q . 1 is an integer. Matrix A has been segmented 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 same as 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 pretended 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.

3. DESING BLOCK
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. These 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. The processors in a shared-memory system share a common memory. Although data is easily accessible to any processor.

The unit has following main blocks

a. Memory blocks
b. Processing element
c. Adders counters

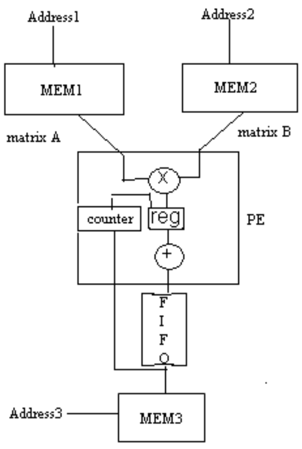

Fig: Block diagram with processing element

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 Lm, adder latency as La, processing element latency as Lpe and computational time of matrix multiplier is Tm. 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 Cij the product term Aik · Bkj must be available at the output of the adder during the same clock cycle as the product term Ai (k+1). B(k+1) j 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 3x3 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 NXN 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 NxN, 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.

4. ALGORITHM

- 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

  The data store in distributed memory shown by sample code of VHDL as:

```
PROCESS(clk)

    BEGIN
       IF (clk = '1' AND clk'EVENT) THEN

         IF (we = '1') THEN -- we is the memory read
    write enable signal
            mem1(conv_integer(addr1)) <= din1;--
    depends on address data can enter in memory

               mem2(conv_integer(addr2)) <= din2;--
    depends on address data can enter in
         ELSE
            dout1 <= mem1(conv_integer(addr1)); --
    depends on address data can outs
                                              dout2 <=
    mem2(conv_integer(addr2)); -- depends on address
    data can outs
         END IF;

       END IF;

    END PROCESS;
```

- Split the design in sub modules
- Multiply the row and column elements
- Accumulate the multiplier outputs and added results is store in distributed memory
- Interconnecting all the modules to complete the circuit.
- The various modules required for are called from the library and then are interconnected as required.
- Calculate the latency and throughput

5. SIMULATION ANALYSIS

Row and column counter is in used to count the number of input data that is read from the memory which is given to matrix arrangement logic. To read/write a data from the memory an address location is sent. Hence the data will become available at the each clock cycle. As the data are retrieved from the Block RAM start from the memory location of zero, for every nine clock-cycles the retrieved data is sent to the multiplier.

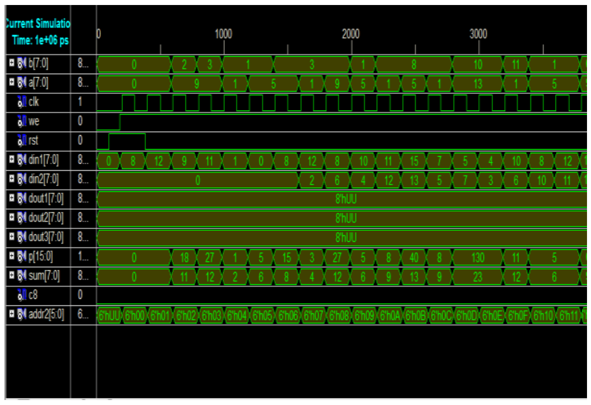

Fig 1 Matrix data write in memory when we=1

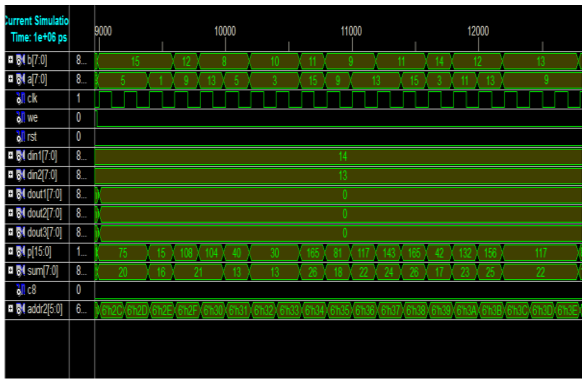

Fig 2 Matrix data read in memory when we=0

Data can be writing and read according the address given to memory1, memory2, memory3 as shown in our block diagram.

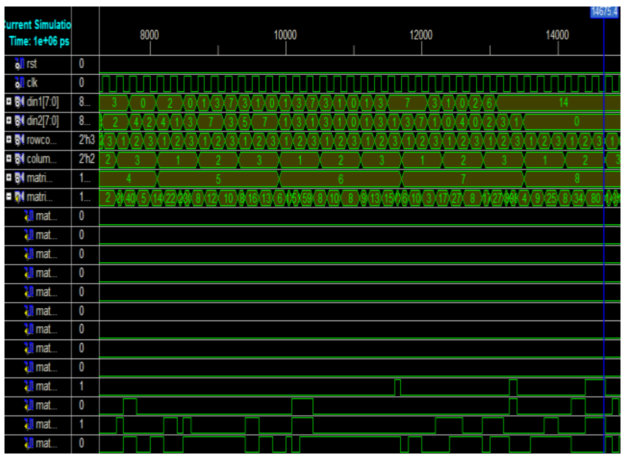

Fig 3 Matrix multiplication and matrix arrangement

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.

6. CONCLUSION

In this paper, we considered two different examples of matrix multiplier architecture where speed is the main constraint. The performance is evaluated by computing its execution time on simulator. The latency time between reading the first elements from the input matrices, A and B, and writing the first element C to the result matrix calculate is 115ns. 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 is calculates 1600ns. Hardware implementation results demonstrate that it can 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.

7. REFERENCES

[1] Syed M. Qasim, Shuja A. Abbasi, “Throughput Latency Implementation of Matrix Multiplication using Field Programmable Gate Array” IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 26, no.6, Nov. 2012.

[2] 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 Transaction on Geo Science and Remote Sensing, vol. 38, no. 4, July 2000.

[3] Davide Anastasia and Yiannus Andreopoulos, “Throughput-Distortion Computation Generaic Matrix Multiplication: Toward A Computation Channel for Digital Signal Processing System” IEEE Transaction on Signal Processing, vol.60, no.4, April 2012.

[4] Nan Zhang "A Novel Parallel Scan for Multicore Processors and Its Application in Sparse Matrix-Vector Multiplication" IEEE Transaction on Parallel and Distributed System, vol. 23, on. 3, March 2012.

[5] Bahram Hamraz, Nicholas HM Caldwell, and P. John Clarkson "A Matrix-Calculation-Based Algorithm for Numerical Change Propagation Analysis" IEEE Transaction on Engineering Management, vol. 60, no. 1, February 2013.

[6] Vasileios Karakasis, Theodoros Gkountouvas, Kornilios Kourtis, Georgios Goumas, Nectarios Koziris "An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication" IEEE Transaction on Parallel and Distributed System- 2012.