FPGA Design and Implementation of Dense Matrix-Vector Multiplication for Image Processing Application

Syed M. Qasim, *Member, IAENG*, Ahmed A. Telba, and Abdulhameed Y. AlMazroo

Abstract—**Matrix-vector multiplication is a computationally intensive and kernel operation used in many image processing applications. This paper presents a preliminary Field Programmable Gate Array (FPGA) design and implementation of dense matrix-vector multiplication for use in an image processing application. The design is optimized for speed which is the main requirement for such applications. The design has been implemented on Virtex-4 FPGA using Xilinx ISE 9.2i and the performance is evaluated by computing the execution time on FPGA. FPGA implementation results demonstrate that it can provide a maximum throughput of 16970 frames per second utilizing only 14% Virtex-4 slices and 57% DSP48 blocks which is quite adequate for most real-time image processing applications.**

Index Terms—**Field Programmable Gate Array (FPGA), Hardware Implementation, Image Processing, Matrix-Vector multiplication.**

I. Introduction

Computationally intensive algorithms used in image and signal processing, multimedia, telecommunications, cryptography, networking and high performance computing (HPC) domains in general were first realized using software running on Digital Signal Processors (DSPs) or General Purpose Processors (GPPs). Significant speed–up in computation time can be achieved by assigning complex computation intensive tasks to hardware and by exploiting the parallelism in algorithms [1].

Recently, Field Programmable Gate Arrays (FPGAs) have emerged as a platform of choice for hardware implementation of computation intensive algorithms [1]–[13]. Especially, when the design at hand requires very high performance, designers can benefit from high density and high performance FPGAs instead of costly multicore Digital Signal Processing (DSP) systems. FPGAs enable a high degree of parallelism and can achieve orders of magnitude speedup over GPPs [7]. This is as a result of the increasing embedded resources on FPGA.

FPGA have the benefits of the hardware speed and the software flexibility; also they have a price/performance ratio much more favorable than Application Specific Integrated Circuits (ASICs). Since the major resources for implementing computation intensive algorithms are embedded on FPGA, latency associated with device communication has been eliminated. However, these embedded resources are limited hence it is important to use these resources efficiently.

The last decade has seen ever increasing application areas for FPGAs. Modern FPGAs currently accommodate more than ten million gates with clock rates up to 600 MHz [13]. Example application areas include single chip replacements for old multichip technology designs, DSP, image processing, multimedia applications, high–speed communications and networking equipment such as routers and switches, the implementation of bus protocols such as Peripheral Component Interconnect (PCI), microprocessor glue logic, coprocessors and controllers [13].

Most of the computation intensive algorithms such as those used in image processing application involve dense or sparse matrix–vector multiplication as the kernel operation. It has been implemented using novel algorithms and technologies to achieve high performance [14]–[16]. In this paper, we present a preliminary design and FPGA implementation of dense matrix–vector multiplication for use in an image processing application.

The remainder of this paper is organized as follows. Section II presents a brief overview of the FPGA technology. The mathematical formulation of the design is presented in section III. Section IV presents the hardware design and FPGA implementation results of the matrix–vector multiplier. Finally, concluding remarks and scope for future work are discussed in section V.

II. FPGA Technology Overview

FPGAs are digital integrated circuits (ICs) that belong to a family of programmable logic devices (PLDs). An FPGA chip includes Input Output Blocks (IOBs) and the core programmable fabric. The IOBs are located around the periphery of the chip, providing programmable I/O connections and support for various I/O standards. The core programmable fabric consists of programmable logic blocks also called Configurable Logic Blocks (CLBs) and programmable routing architectures [17].

Many different architecture and programming technologies have evolved to provide better designs that make FPGAs economically viable and an attractive alternative to ASICs. Modern FPGAs have superior logic density, low chip cost and performance specifications comparable to low end

Manuscript received March 23, 2010. This work was financially supported by Research Center in the College of Engineering, King Saud University under grant no. 11/430.

Authors are with the Department of Electrical Engineering, College of Engineering, King Saud University, Riyadh 11421, Saudi Arabia (emails: smanzoor@ksu.edu.sa, atelba@ksu.edu.sa and almazroo@ksu.edu.sa).

microprocessor. With multimillion programmable gates per chip, current FPGAs can be used to implement digital systems capable of operating at frequencies up to 600 MHz. In many cases, it is possible to implement an entire system using a single FPGA. This is very economical for specialized applications that do not require the performance of custom hardware [17].

Significant technological advancements have led to architectures that combine FPGA's logic blocks and interconnect matrices, with one or more microprocessors, embedded Intellectual Property (IP) cores, memory blocks, DSP blocks integrated on a single chip to facilitate the implementation of Programmable System–on–a–Chip (PSoC) designs [18]–[19].

Examples of PSoC are the Xilinx Virtex–II Pro, Virtex–4, Virtex–5 and Virtex–6 FPGA families, which include one or more hard-core PowerPC processors embedded along with the FPGA's logic fabric [20]–[22]. Alternatively, soft processor cores that are implemented using part of the FPGA logic fabric are also available. Many soft processor cores are now available such as: Xilinx 32–bit MicroBlaze [23] and PicoBlaze, and the Altera Nios and the 32–bit Nios II processor [17].

III. Mathematical Formulation

Matrix–vector multiplication is computationally intensive and typical routine used in many image processing applications. It requires several multiply and accumulate (MAC) units. In DSPs, the overall performance is limited by the number of multiplications and additions that could be done in parallel. DSPs take several clock cycles to perform all the necessary MAC operations. However, modern FPGAs, on the other hand are equipped with large number of hardware resources embedded in the FPGA fabric itself such as DSP48 blocks, multipliers, Block RAMs, etc [17]. It can provide higher and more efficient processing rates required by such applications if the algorithm is coded in a way to utilize these embedded resources efficiently. The objective of this paper is to realize a large and dense matrix–vector multiplier for an image processing application [24].

We represent the vector C as $(C_1, C_2 ... C_m)^T$ and vector G which represents the image data. According to the application, we want to multiply matrix S with vector C represented by the following equation

$$C=SG \qquad (1)$$

where, S is a Jacobian matrix. In the discrete form, it is required to find the unknown vector G from the known vector C, while S is treated as a constant matrix for simplicity. We can represent G by the following relationship

$$G=S^TC \qquad (2)$$

where, S^T is the transpose of S. Replacing S^T by A, mathematically; the above equation is approximated by the following relationship

$$G=AC \qquad (3)$$

The key idea here is to calculate G using (3). The dimension of the given matrices depends on the application, which, in this case is summarized in table 1.

Table 1: Matrix Dimensions

Matrix Symbol	Matrix Dimension
A	1024×28
C	28×1
G	1024×1

IV. Hardware Design and FPGA Implementation

In this section, we present the details of the hardware design for implementing matrix–vector multiplication on FPGA. As can be seen from (3), the image processing algorithm reduces to matrix–vector multiplication. For efficient implementation and maximum speed-up, integer arithmetic is utilized. Since the floating–point arithmetic unit consumes more silicon real estate of FPGA and are slower as compared to integer arithmetic, we used integer arithmetic for the design.

The design involves the computation of G = AC, where A is a matrix, C and G are vectors as summarized in table 1. It is required to calculate vector G. The matrix–vector multiplication is performed by broadcasting rows of matrix A and multiplying the corresponding column elements of vector C [25]. The sequence of operations involved in the computation of matrix–vector multiplication is as follows:

1) Reading the individual row elements of matrix A and the individual column elements of vector C.
2) Storing them in internal buffers row and column wise respectively.
3) Multiplying the row and column elements.
4) Accumulating the multiplier output and writing back the results to the output buffers.

The input and output buffers are implemented on the FPGA. The matrix–vector multiplication typically involves MAC operations. The MAC unit consists of a multiplier and accumulator. The row and the column elements are supplied as the two inputs to the multiplier. The output of the multiplier is directly given to the accumulator as one of the inputs. The previous output of the accumulator is fed back as the second input.

The MAC unit takes each element of the matrix A in row major format and each element of vector C, multiplies them and adds the result to the running total. This process is repeated till the last element of row A and column C. The values are fed in a sequential manner. If the reset signal is asserted high, the contents of registers A and C are cleared.

After a delay, as determined by the implementation results, the first element of vector G is available at the serial output and this output is stored in on–chip memory as shown in figure 1. This operation is repeated and the process continues until all the rows of matrix A are processed. Finally, the output vector G is available with all the elements stored in the memory locations. A simplified diagram of the processing element for matrix–vector multiplication is shown in figure 1. Figure 2 presents the simulation result for matrix–vector multiplication using Xilinx edition ModelSim XE III 6.4b simulator.

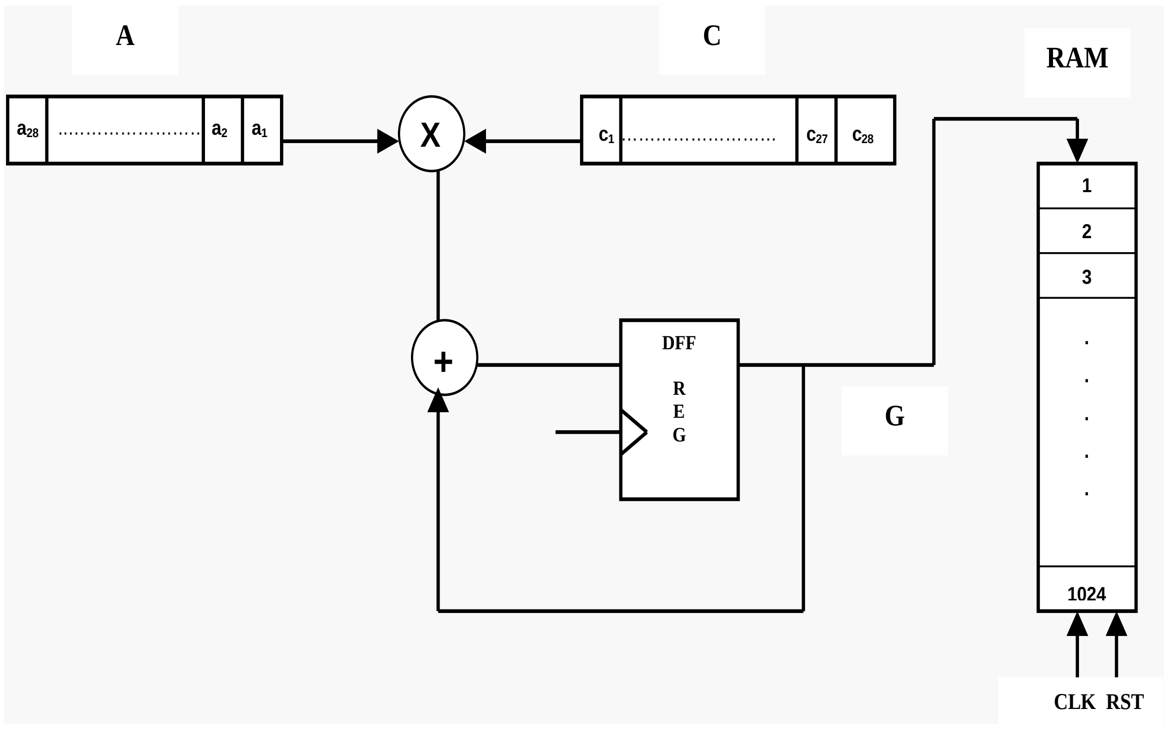
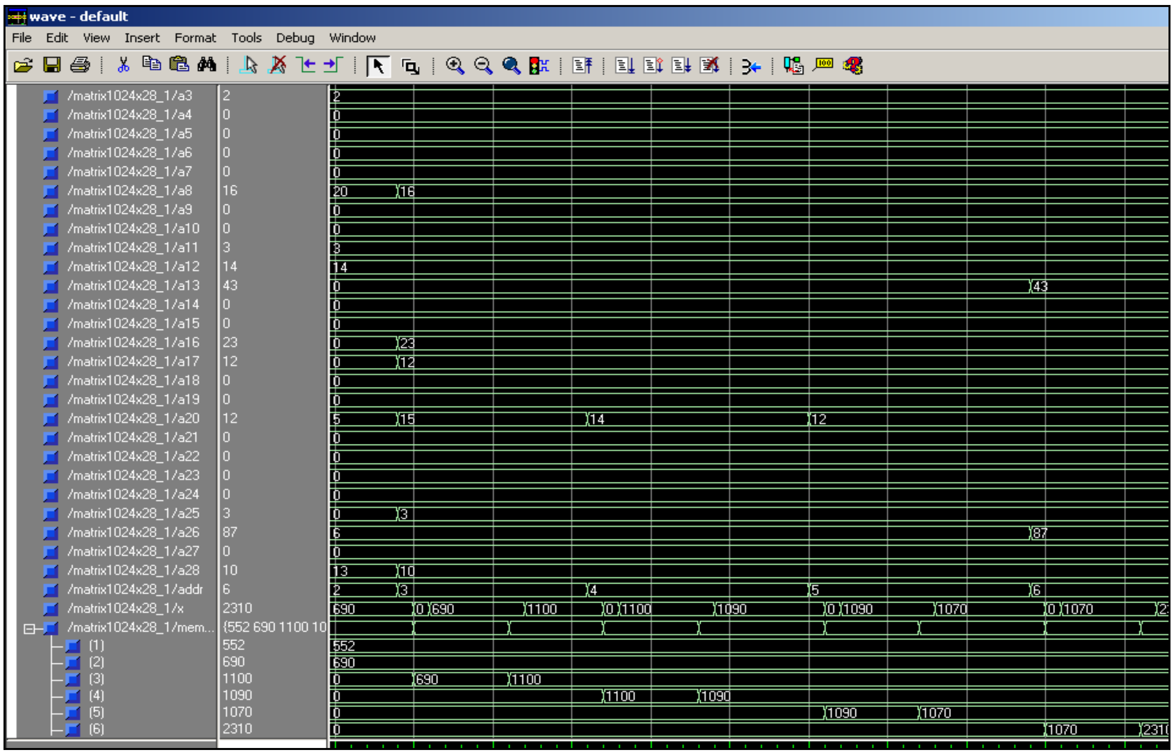


Figure 1: Matrix-vector multiplication processing architecture



/matrix1024x28_1/a3	2
/matrix1024x28_1/a4	0
/matrix1024x28_1/a5	0
/matrix1024x28_1/a6	0
/matrix1024x28_1/a7	0
/matrix1024x28_1/a8	16
/matrix1024x28_1/a9	0
/matrix1024x28_1/a10	0
/matrix1024x28_1/a11	3
/matrix1024x28_1/a12	14
/matrix1024x28_1/a13	43
/matrix1024x28_1/a14	0
/matrix1024x28_1/a15	0
/matrix1024x28_1/a16	23
/matrix1024x28_1/a17	12
/matrix1024x28_1/a18	0
/matrix1024x28_1/a19	0
/matrix1024x28_1/a20	12
/matrix1024x28_1/a21	0
/matrix1024x28_1/a22	0
/matrix1024x28_1/a23	0
/matrix1024x28_1/a24	0
/matrix1024x28_1/a25	3
/matrix1024x28_1/a26	87
/matrix1024x28_1/a27	0
/matrix1024x28_1/a28	10
/matrix1024x28_1/addr	6
/matrix1024x28_1/x	2310
/matrix1024x28_1/mem...	{552 690 1100 10
(1)	552
(2)	690
(3)	1100
(4)	1090
(5)	1070
(6)	2310

Figure 2: FPGA simulation results of matrix-vector multiplication

In order to evaluate the performance of FPGA–based implementation, the algorithm was coded in VHDL and implemented on Virtex–4 (xc4vlx200ff1513, speed grade: –11) family using Xilinx ISE 9.2i tool. The design was synthesized into Virtex–4 FPGA optimized for speed. The hardware resource utilization is summarized in table 2.

As shown in table 2, 14% of the slices and 57% DSP48s are utilized leaving a plenty of room to implement more parallel processors on the same FPGA chip. The results listed in table 2 were obtained using Xilinx ISE 9.2i tool. The optimization setting for ISE is for maximum clock speed. The total processing time using Virtex–4 FPGA is found to be 58.93 µs; this is equivalent to a throughput of 16970 frames per second. The results indicate the feasibility of using FPGA for real–time high speed image processing applications using this matrix–vector multiplication.

Table 2: FPGA Resource Utilization

Resources	**Used/Available**	**Utilization**
Slices	1,3010 out of 89088	14%
Four-input LUTs	9612 out of 178176	5%
DSP48s	55 out of 96	57%
Maximum Frequency	17.376 (MHz)	-

V. Conclusions and Future Work

Most of the algorithms which are used in DSP, image and video processing, computer graphics and vision and high performance supercomputing applications have matrix operation as the kernel operation. In this paper, we have presented a preliminary design of dense matrix–vector multiplication. The design has been implemented on a Xilinx Virtex–4 FPGA device and the performance is evaluated by computing its execution time on FPGA. Hardware implementation results demonstrate that it can provide a throughput of 16970 frames per second which is sufficient for many real–time image and video processing applications.

Some recommendations to continue this work in future are outlined below:

1) Implementing the architecture of matrix–vector multiplication using floating point arithmetic instead of integer. This will further enhance the design by making it suitable for other high performance computing applications, where the current trend is to use double precision floating point numbers.
2) FPGA–based standalone module is presented to enhance the computation time of the matrix–vector multiplication. However, the communication time between the FPGA coprocessor and host PC is not taken into consideration. The introduction of parallel and/or pipelined coprocessor along with an embedded processor of FPGA can reduce the computational time depending on the level of parallelism introduced.
3) Exploration of domain–specific Coarse Grained Reconfigurable Architecture (CGRA) for implementing computationally intensive matrix–vector multiplication.

Acknowledgment

The authors would like to thank Dr. Mohammed AlEsheikh and the staff of Research center, College of Engineering, King Saud University for their support.

References

[1] S. Ogrenci, A. K. Katsaggelos, and M. Sarrafzadeh, “Analysis and FPGA Implementation of Image restoration under resource constraint,” *IEEE Trans. on Computers*, Vol. 52, No. 3, pp. 390-399, 2003.

[2] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, “Implementing an OFDM Receiver on the RaPiD Reconfigurable Architecture,” *IEEE Trans. on Computers*, Vol. 53, No. 11, pp. 1436-1448, 2004.

[3] G. R. Goslin, “A Guide to Using Field Programmable Gate Arrays for Application-Specific Digital Signal Processing Performance,” *Microelectronics Journal*, Vol. 28, Issue 4, pp. 24-35, 1997.

[4] J. Isoaho, J. Pasanen, O. Vainio, and H. Tenhunen, “DSP System Integration and Prototyping with FPGAs,” *Journal of VLSI Signal Processing*, Vol. 6, pp. 155-172, 1993.

[5] A. G. Ye and D. M. Lewis, “Procedural Texture Mapping on FPGAs,” in Proc. of ACM/SIGDA 7th Intl. Symp. on Field Programmable Gate Arrays, pp. 112-120, 1999.

[6] S. Knapp, “Using Programmable Logic to Accelerate DSP Functions,” http://www.xilinx.com/appnotes/dspintro.pdf.

[7] J. Ma, “Signal and Image processing via Reconfigurable Computing,” in Proc. of the First Workshop on Information and Systems Technology, 2003.

[8] F. Otto and Z. Pavel, “Hardware Accelerated Imaging Algorithms,” in Proc. of AUTOS’2002 Automatizace systému, pp. 165-171, 2002.

[9] L. Batina, S. B. Ors, B. Preneel, and J. Vandewalle, “Hardware architectures for public key cryptography,” *Integration, the VLSI Journal*, Vol. 34, pp. 1-64, 2003.

[10] D. Johnson, K. Gribbon, D. Bailey, and S. Demidenko, “Implementing Digital Signal Processing Algorithm’s in FPGA’s: Digital Spectral Warping,” in Proc. of 9th Electronics New Zealand Conf., pp. 72-77, 2002.

[11] K. Compton and S. Hauck, “Reconfigurable Computing: A Survey of Systems and Software,” *ACM Computing Surveys*, Vol. 34, No. 2, pp. 171-210, 2002.

[12] R. Tessier and W. Burleson, “Reconfigurable Computing for Digital Signal Processing: A survey,” *Journal of VLSI Signal Processing*, Vol. 28, No. 3, pp. 7-27, 2001.

[13] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, “Reconfigurable Computing: architectures and design methods,” *IEE Proc. of Computer Digital Techniques*, Vol. 152, No. 2, pp. 193-207, 2005.

[14] N. Fujimoto, “Faster matrix-vector multiplication on GeForce 8800GTX,” in Proc. of the 22nd IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS), pp. 1-8, 2008.

[15] N. Fujimoto, “Dense matrix-vector multiplication on the CUDA architecture,” *Parallel Processing Letters*, Vol. 18, No. 4, pp. 511-530, 2008.

[16] M. Delorimier and A. Dehon, “Floating-Point sparse matrix-vector multiply for FPGAs,” in Proc. of the IEEE Intl. Symposium on Field Programmable Gate Arrays, pp. 75-85, 2005.

[17] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, “Features, Design Tools and Application Domains of FPGAs,” *IEEE Trans. on Industrial Electronics*, Vol. 54, No. 4, pp. 1810-1823, 2007.

[18] G. Stitt and F. Vahid, “Energy advantages of microprocessor platforms with on-chip configurable logic,” *IEEE Design and Test of Computers*, Vol. 19, No. 6, pp. 36-43, 2002.

[19] A. Ansari, P. Ryser, and D. Isaacs, “Accelerated System Performance with APU-enhanced processing,” *Xcell Journal*, First quarter 2005.

[20] Xilinx Inc, Virtex-II platform FPGA Data Sheet, 2005.

[21] Xilinx Inc, Virtex-4 multiplatform FPGA, 2005.

[22] Xilinx Inc, Virtex-5 multiplatform FPGA, May 2006.

[23] Xilinx Inc, MicroBlaze Soft Processor Core, 2005.

[24] D. F. Garcia-Nocetti, J. G. N. Gamio, and L. Aguilar, “Parallel Realization of the Linear Back-projection Algorithm for Capacitance Tomography using TMS320C6701 DSP,” in Proc. of 3rd World Congress on Industrial Process Tomography, pp. 648-653, 2003.

[25] J.-Y. Blanc and D. Trystram, “Implementation of parallel numerical routines using broadcast communication schemes,” in Joint Intl. Conf. on Vector and Parallel Processing, Lecture Notes in Computer Science, Springer Verlag, Vol. 457, pp. 467-478, 1990.