УДК 004.354.4:512.643:004.388

КОДИРОВКА МАТРИЦ УМНОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ НОВЫХ УСТРОЙСТВ FPGA (ПЛИС)
"ENCRYPTION OF MATRIX MULTIPLICATION USING NEWER FPGA DEVICES"

PhD Candidate: Haider I Mohsin, Prof. J.I.Batyrkanov
Kyrgyz state Technical University
liveoffice12@gmail.com

1. Abstract

Matrix multiplication is a fundamental buildingblock for many applications includingimage processing, coding, and digital signalprocessing. In this project weintended to give two ways of improvement of Encryption efficientmethodology for implementing integer and floating point matrixmultiplication using FPGAs. The architecture presented is based on new FPGA features and provides a significant reductionin total computation time and resource utilization over previous solutions.

2. Introduction

Many applications like image processing, coding, and digital signal processing have matrix multiplication operation as a fundamental one. Processors implement the matrixmultiplicationin $O(n^3)$running time. To reduce this complexity, many parallel methods have been developed [2] [3] [4]. While these parallel processing methods havereduced the total computation time, they are applicable only for small matrix dimensions since they require significant device resources. Recent advancesinFieldProgrammable Gate Array (FPGA) technology gave new possibilities for implementation ofmore efficient parallel matrix multiplication algorithms.

3. Matrix Multiplication Methods

Several methods have been explored to decrease the algorithmic complexity of the matrix multiplication operation [2] [3] [4]. Although theseprior works have systematically improved the resource utilization and operating frequency over prior designs, their algorithms all require intercommunication between the processing elements (PE's). While this improves the latency of the system, it is not delay or resourceoptimal due to data sharing between the PE's. In this project we will use the method that performs matrix multiplication with PE's that operate in isolation fromeach other [1].

New advances in FPGA technology allow moreefficient implementation of integer and floating-point multiplication and addition over previous generations ofdevices. For example, theXilinx Virtex-4 family introduced the dedicated DSP-48 block which consists of a$18 x18$ multiplier coupled with a 48-bit adder / accumulator [7]. Inaddition, the Virtex-4 family includes dedicated FIFOs which allow efficient implementation of both synchronous and asynchronous FIFOs without requiring additional logic resources.

3.1 The parallel model

In order to optimize FPGA architecture resource use, the data frominput matrices A and B

should be re-used. Optimal data re-use occurs when data is read frommemory for matrices*A* and *B* exactly once. By simultaneously reading one columnof 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 termof every element in output matrix *C*to be computed per clock cycle.This process is shown in Figure 1.

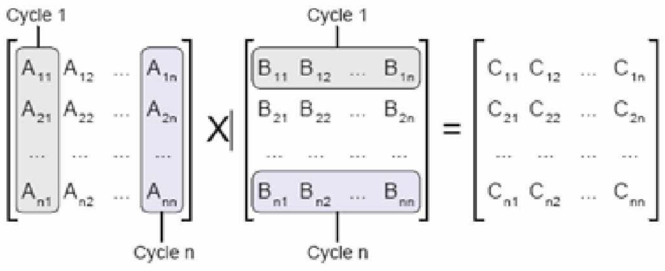

Figure 1: Parallel Matrix Multiplication Sequence

3.1.1 Multiplier Array Architecture

In order to meet the memory read requirements, an array structure of PE's is required. Figure 2 shows the array structure requiredwheninputmatrix*A* has *l* elements,and *B* has *m* elements.

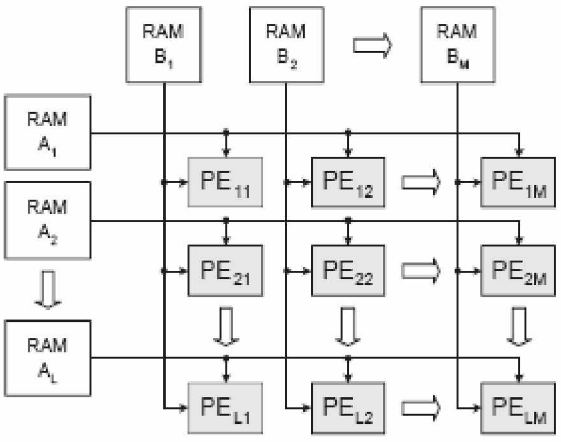

Figure 2: Matrix Multiplier Array

Inthisarrayconfiguration,*lm* product terms are produced each clock cycle fromthe*l*+ *m* matrix elements. In the sequel, we assume that $m = l$, allowing for a regular multiplier array structure. In all but the smallest of matrix dimensions n where $m = n$, it is assumed that $m = n/k$ where *k* is an integer. This array configuration allows the algorithmto be applied to matrices of non- square dimension.

3.1.2 Processing Element (PE) Structure

The structure of the PE is shown in Figure 3.

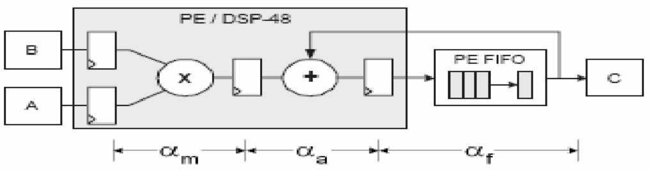

Figure 3: Processing Element (PE) structure

The PE structure consists of one input each from matrix*A* and *B*, a multiplier accumulator (MAC), and a result FIFO. The multiplier latency is denoted as a_m,while that of the adder and FIFO are denoted as $\alpha_a$and $\alpha_f$respectively. The inputs from matrices*A* and *B*, containing one word each per clock cycle, are implemented using dedicated routes-fromtheBlockRammemoryassociated with the multiplier. By having dedicated memory connections for each PE the routing and resource delay penalty is eliminated.

During the computation ofoutput matrix element$C_{ij}$the product term$A_{ik}\cdot B_{kj}$must be available at the output of the adder duringthe same clock cycle as the product term$A_{i(k+1)}\cdot B_{(k+1)j}$is available fromthe multiplier. The final product termrequires$a_a$oflatencyin the adderbefore it is stored in memory.

3.1.3 Memory Structure

The memory hierarchy is organized in the following way: input matrices$A$ and B, and output matrix C FIFO storage. Each matrix is partitioned into$m$ BlockRambanks. This structure has one bank of A and B feeding one PE array row and column respectively. Each bank of $A$stores$k = (n/m)$ words of each column in A, for every row of A. Similar, each bank of B stores $k =(n/m)$ words of each row in B, for every column of$B$. Because the data remains in FIFO memory only until used, additional data can be loaded fromexternal memory to accommodate array dimensions larger than internal storage would allow. Each BlockRamhas a depth$d$ based on the width of the internal data. To determine the number of BlockRams required per matrix word, the input data word width (W_{mat}) is divided by the BlockRamwidth (W_{bram})and rounded up. This is multiplied by the number of words in thematrix $n^2$and divided by the FIFO depth d. The total number of input BlockRams required for matrices A and B are as presented in Equation (1).

$$InputBlockRams = 2 \quad ((W_{mat}/W_{bram}) \quad (n^2)/d) \quad (1)$$

The number of PE BlockRams required is the width of the input word (W_{PE})divided by the Block-Ram(W_{bram})width and rounded up. Note that the datawidth at the output of the matrix MAC operation is wider than that ofthe input matrices, and will thus require more BlockRams per word. This value ismultiplied by the number ofPE words (k^2)and divided by the FIFO depth$d$. The total number of BlockRams required to storetheresultisgiveninEquation(2).

$$PEBlockRams = (W_{PE}/W_{bram}) \quad (k^2/d) \qquad (2)$$

3.2 System Latency and Computational Time

3.2.1 System Latency

In the analyzed method , we have $k= n/m$, which requires k^2clock cycles to read the input data which produces the product terms for one column of$A$multiplied by one row of B.With n rows and n columns in method's design, and each column of$A$ and B read simultaneously, there are nk^2 clock cycles required to read the input matrices from memory. This means that after $[(n - 1)k^2+ 1]$ clock cyclesthe elements $A_{1n}$and B_{n1}which are required for the final product termof C_{1n}will be read frommemory. Given that the multiplier latency is$a_m$and the adder latency is a_a,the first result C_{1n}will be presented to the PE FIFO after $[(n - 1)k^2+ 1]+a_m+ a_a$clock cycles. Because the PE FIFO also serves as the output matrix$C$ storage, no additional clock cyclesare requiredto store the result. The latency formula of thesystemis given in Equation (3).

$$L_m= [(n - 1)k^2+ 1]+a_m+ a_a(3)$$

3.2.2 Total Computational Time

The input matrix elements $A_{nn}$and B_{nn}required for the final product termof$C_{nn}$are read frommemoryafternk^2clock cycles. Given the multiplier latency of$a_m$and the adder latency ofa_a, the final result C_{nn}will be written tothe PE FIFO after$nk^2+a_m+a_a$eloek cycles. Again, with no addition time required for storing result$C_{nn}$to memory, the total computation time is given in Equation (4).

$$Total\ Computation\ Time = nk^2+ a_m+ a_a(4)$$

4. Proposed Improvement

By analyzing the described method for parallel matrix multiplication, we suggest two strategies that can improve the performance of the algorithm. Since the size of the numbers that are added in the PE unit is large, we could use a high-performance carrylook-ahead adder. This modificationwill decrease the value ofa,respectively decreasing the system latency (Ecuation 4). Another improvement could be usingpipelining in multiplication operation ofthe PE unit. By using a pipelined multiplier, we will decrease the $a_m$value fromEquation 4, which will also improve the systemlatency.

4.1 Carry – LookAhead Adder

In a carry – lookahead adder, all the carry outputs are calculated at once by specialized lookahead logic. The result is that instead of having to wait for the output to "ripple" up to the most significant bit, the entire result can becomputed with significantly less delay.

While we need to use large adders, we use the carry lookaheadmethod on the outputs of smaller adders (which themselvesmay be carry lookahead adders)to build up a larger result. For example, a 16-bit adder may be implemented as four 4-bit adders, connected in a carry lookahead configuration.

4.2 Pipelining

Using a pipelined multiplier can considerably increase the performance ofthe PE unit. Jia Di [5] proposed a 2-dimensional pipeline gating scheme. Results show that 16-bit multipliers using this technique have on average a 66%power saving and latency reduction of 47% over original design.

5. Conclusions

In this report we analyzed aEncryptionefficient implementation ofthe matrix multiplication algorithmfor fixed-point and floating-point designs [1]. It's advantage of recent FPGA technology to significantly reduce the resource utilizationand total computation time over previous methods. Algorithmproposed removes the intercommunication between parallel processing elements (PEs), and allows each PE tooperate inisolation. Also, this algorithmallows the implementationusingmatrices of arbitrary dimension.

6. References

1. Scott J. Campbell.Sunil P. Khatri, "Resource and delay efficient matrix multiplication using newer FPGA devices", *in Great Lakes Symposium on VLSI, 2006*
2. J. Jang, S. Choi, and V. Prasanna, "Area and time efficient implementations of matrix multiplication on FPGAs,," in *Proceedings, IEEE International Conference on FieldProgrammable Technology(FPT)*, pp. 93–100, Dec 2002.
3. A. Amira and F. Bensaali, "An FPGA basedparameterizablesystemformatrixproduct implementation," in*IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 75–79, Oct 2002.
4. L. Zhuo and V. Prasanna, "Scalable andmodular algorithms for floating-point matrix multiplication on FPGAs," in *Proceedings, International Parallel and Distributed Processing Symposium*, p. 92, Apr 2004.
5. Jia Di, J.S.Yuan, "Power-aware pipelined multiplier design based on 2-dimensional pipeline gating**"**,*inGreat Lakes Symposium on VLSI, 2003.*[6] http://www.wikipedia.org
6. Xil-inx,"XtremeDSPdesignconsiderations."www.xilinx.com.