ENERGY EFFICIENT ARCHITECTURE FOR MATRIX MULTIPLICATION ON FPGAS

Kiran Kumar Matam[†]*, Hoang Le*[‡]*, and Viktor K. Prasanna*[‡]

[†] Computer Science Department, [‡] Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA 90007
Email: {kmatam, hoangle, prasanna}@usc.edu

ABSTRACT

Energy efficiency has emerged as one of the key performance metrics. In this work, we first implement a baseline architecture for matrix multiplication, parameterized with the number of processing elements and the types of storage memory. We map this architecture onto a state-of-the-art Field Programmable Gate Array (FPGA). A design space is generated to demonstrate the effect of these parameters on the energy efficiency (defined as number of operations per Joule). We determine that on-chip memory constitutes the largest amount of power consumption among all the components. To improve energy performance, we propose a memory activation schedule. Using this scheme, the proposed optimized design achieves 2.2x and 1.33x improvement with respect to Energy×Area×Time (EAT) and energy efficiency, respectively, compared with the state-of-the-art matrix multiplication core.

1. INTRODUCTION

Matrix multiplication is one of the frequently used kernels in a wide variety of applications. Several parallel architectures and algorithms have been proposed over the years. These solutions are optimized for either latency, throughput, or area. Also, most of the prior work [5, 1] for matrix multiplication on FPGA platforms has mainly focused on developing algorithms and architectures that occupy less area and achieve high sustained GFLOPS/sec. To the best of our knowledge, there has been no previous work targeted at exploring the design space for energy efficiency of matrix multiplication on FPGAs.

In this paper we explore matrix multiplication with respect to energy efficiency. We use a linear array architecture for matrix multiplication [2]. For $n \times n$ matrix multiplication, the linear array architecture can achieve optimal latency of $O(n^2)$ with n PEs. Also the layout is simple as all the inter-connections are localized and are within a PE or in between adjacent PEs only. We use energy efficiency (defined as number of operations per Joule) and composite Energy×Area×Time (EAT) as the performance metrics. This paper makes the following contributions:

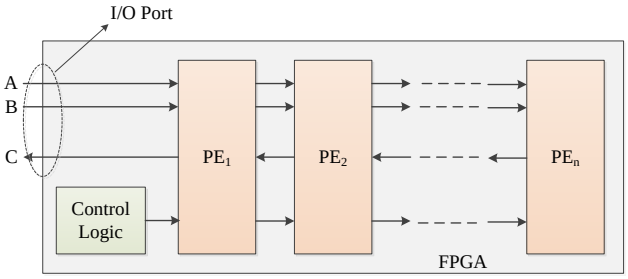

Fig. 1. Overall linear array matrix multiplication architecture

1. A parameterized implementation of the linear array architecture for matrix multiplication. Implementation parameters include: the number of processing elements, and the types of storage memory (Section 2.1).
2. A design space that demonstrates the effect of the types of storage memory on the energy efficiency metric (Section 3).
3. A cache-based architecture that reduces the power consumption of memory using a memory activation schedule (Section 2.2).
4. Demonstrate improved energy performance for various problem sizes (Section 3.5).
5. Significant energy efficiency improvement compared with the state-of-the-art matrix multiplication core (Section 3.6).

The rest of the paper is organized as follows. Section 2 describes the baseline architecture and the proposed cache-based architecture. Section 3 presents the experimental results and performance evaluation. Section 4 concludes the paper.

2. ARCHITECTURE AND OPTIMIZATIONS

Given two matrices A, B of size $n \times n$, the product AB, denoted C, is computed as: $C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$, $1 \leq$

This work has been funded by DARPA under grant number HR0011-12-2-0023.

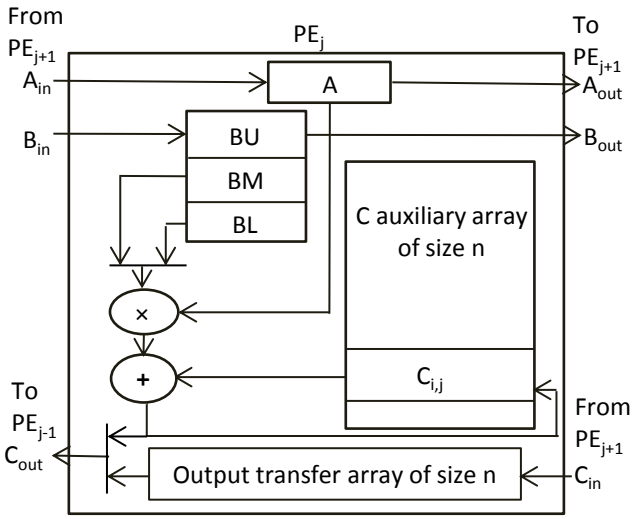

Fig. 2. Architecture of a PE

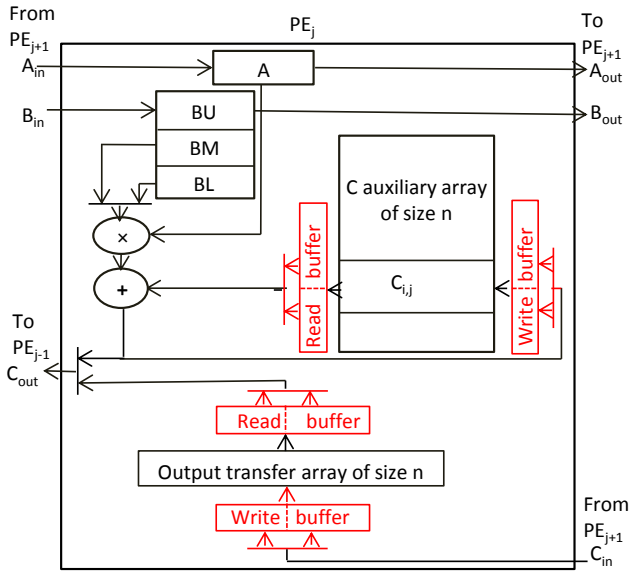

Fig. 3. Block diagram of a PE in the cache-based design

$i, j \leq n$. Throughout this paper, the designs are based on $O(n^3)$ complexity matrix multiplication algorithm.

2.1. Baseline Architecture

Fig. 1 shows the overall architecture [2]. Fig. 2 shows the architecture of a PE. The design consists of a control unit and a data unit. For $n \times n$ matrix multiplication, the data unit consists of n identical PEs. Each PE includes a multiply adder, 4 registers, and 2 n-word local memories. The control unit controls the operations of these PEs.We assume that the input matrices are stored outside the FPGA and the input data is fed to on-chip via I/O ports.

The architecture can perform a $n \times n$ matrix multiplication in $n^2 + 2n$ cycles. To compute $C = AB$, matrix B is fed into the design in row major order, starting from the top row. Matrix A is fed into the design n cycles behind matrix B in column major order, starting from the left most column. Once the multiplication of A and B is completed, the resulting matrix C is outputted in the column major order, starting from the left most column.

The architecture is parameterized with implementation parameters, such as the number of processing elements (PEs) and the types of storage memory (distributed RAM or block RAM). These parameters allow us to switch between various architectures without having to recode the entire design.

2.2. Cache-Based Architecture

Our analysis on the power consumption of each component of the baseline architecture (more details in Section 3.3 and Fig. 5 in Section 3.4) shows that local memory constitutes the largest amount of power consumption among all the components. Therefore, reduction in power consumption of the local memory makes a greater impact on the power consumption of the entire system. The algorithm has spatial locality; words are accessed in ascending order, starting with word 0. From these observations we can reduce the power consumption of the local memory if we can access k words in one clock cycle ($1 < k \leq n$), and suppress the switching activity in the local memory in the subsequent $k - 1$ clock cycles.

We propose a cache-based design to reduce power consumption of local memories. We use a memory of size $n/k \times k$ ($height \times width$). For the sake of simplicity, we can choose k such that n is divisible by k. In this case, the height of the new memory is exactly n/k. Two k-word registers are introduced, one for the read operation and one for the write operation. The read register is configured as parallel-in-serial-out shift register. In a read operation, this register reads in k words from the memory, then outputs one word per clock cycle. During the shift time, the switching activity in memory is suppressed. On the other hand, the write register is configured as serial-in-parallel-out shift register. For every clock cycle, the write register shifts the coming word into its parallel register. At the k^{th} cycle, the output (k words) is written into the memory. Note that the memory is not written during the window of $k - 1$ cycles. Fig. 3 shows the proposed cache-based architecture for $k = 2$. In this figure, $Read\ buffer$ is the read register, and $Write\ buffer$ is the write register.

Our approach of using buffers to suppress the switching activity in memory over certain cycles is also applicable in algorithms other than matrix multiplication. If an algorithm has read and write operations accessing adjacent memory locations, then the data can be held in the buffers and the access to memory can be done only once in a window of k cycles.

3. PERFORMANCE EVALUATION

We denote the baseline architecture (Section 2.1) using BRAM and distributed RAM as *BRAM-based architecture* and *Dist.-RAM based architecture*, respectively.

3.1. Experimental Setup

The baseline and cache-based architectures were implemented in Verilog, using Xilinx ISE 14.4, with Xilinx Virtex-7 XC-7VX690T with -3 speed grade as the target. Xilinx's multiply adder core [4] was used in all our implementations. When generating the cores we choose configuration options to maximize frequency of the cores. The local memory is configured as dual-ported (one read port and one write port), and can either be implemented using distributed RAM or block RAM. The designs were verified by post place-and-route simulation. The input matrices for the simulation were randomly generated and had an average switching activity of 50%. All the reported results are post place-and-route simulation results. We used the VCD file (value change dump file) as input to Xilinx XPower Analyzer to produce accurate power dissipation estimation. The operating frequency of all the evaluated designs were set to be 250 MHz for power evaluation.

3.2. Performance Metrics

We consider the following two metrics for performance evaluation.

1. *Energy Efficiency* is defined as the number of operations per unit energy consumed. When multiplying two $n \times n$ matrices using well-known three nested loop matrix multiplication algorithm, *energy efficiency* is given by $2n^3$/*energy consumed by the design*. *Energy of the design = time taken by the design* $\times$ *average power dissipation of the design*. Alternatively *Energy efficiency* of the design is *Power efficiency* (number of operations per second/Watt).

2. *Energy*$\times$*Area*$\times$*Time (EAT)* is measured as the product of three important metrics, time, area, and energy of the design. Number of slices occupied by the design is considered as the area of the design. A BRAM or a Multiply and Add unit is counted as 16 slices [2].

3.3. Energy Dissipation Analysis

For an $n \times n$ MM, the number of operations performed is $2n^3$, time is n^2+2n cycles. The total amount of local memory is $2n^2$ elements, the number of arithmetic units is n. The number of IO pins is three times the data width of the elements. So asymptotically energy consumed by the arithmetic units is $O(n^3)$, where unit energy is assumed to be consumed for performing a fixed point arithmetic operation. Energy consumed by the memory units is $O(n^4)$, where unit energy is assumed to be consumed in a cycle for a unit of storage. As all the communication is in between the adjacent PEs or in between the components within the PEs only, energy consumed for communication is $O(n^3)$, where unit energy is assumed to be energy for transferring a word of data within a PE or between adjacent PEs. Total energy consumed by the design is sum of the computation, memory, and communication energy. Therefore, as the problem size grows, the energy efficiency is $O(1/n)$; energy consumed by the memory limits the scalability of the design.

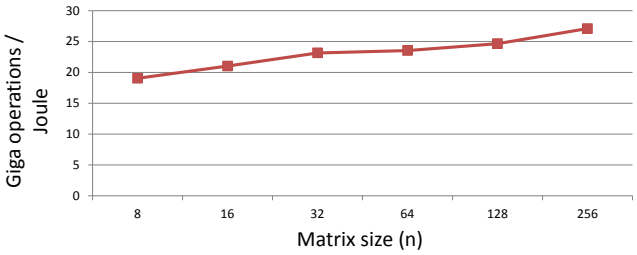

Fig. 4. Energy efficiency of *BRAM-based architecture*

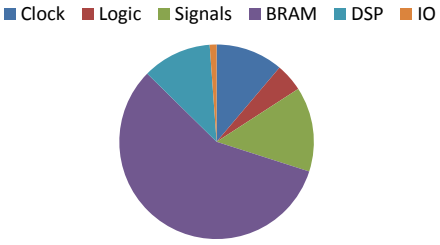

Fig. 5. % power consumed by the components for the 128×128 *BRAM-based architecture*

3.4. Design Space of Baseline Architecture

In this subsection we discuss the experiments we perform on the *BRAM-based architecture* and *Dist.-RAM based architecture*. In these experiments we fix the data width at 8, and vary the number of PEs from 8 to 256.

Fig. 4 shows the energy efficiency of the *BRAM-based architecture* designs. Fig. 5 shows the % power consumption of the various components for 128×128 MM. Dominant portion of the energy is consumed by the BRAM. Each PE requires two 18 Kb BRAMs for all the considered problem sizes. Therefore, the amount of storage used increases linearly for the considered problem sizes. As the matrix size increases, percentage of memory utilized in a BRAM increases while the IO power remains constant resulting in an increase in energy efficiency.

Fig. 6 shows the energy efficiency of the *Dist.-RAM based architecture* designs. Initially, as the matrix size increases, the percentage of IO power in the total power de-

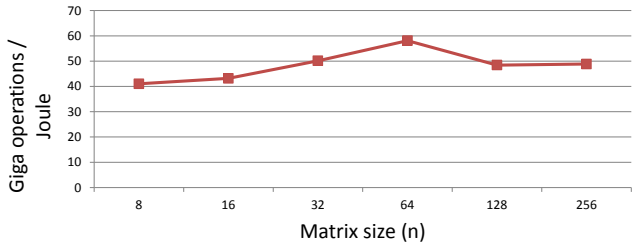

Fig. 6. Energy efficiency of *Dist.-RAM based architecture*

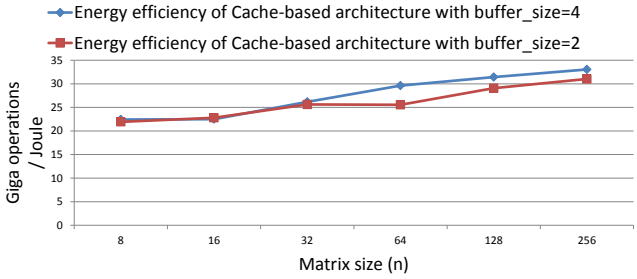

Fig. 7. Energy efficiency of *Cache-based architecture*

creases and hence the energy efficiency increases. For matrix sizes greater than 64 we observed that routing becomes complex and hence the energy efficiency decreases.

3.5. Cache-Based Architecture

In this section, we discuss the experiments on the cache-based architecture discussed in Section 2.2 (denoted *Cache-based architecture*). We vary the number of PEs from 8 to 256, fix the data width as 8, and use BRAM as storage memory. Let $buffer\_size$ be the number of words of the read register (Section 2.2) used as a buffer before the memory. For the considered problem sizes, two 18 Kb BRAMs are required in each PE. For 18 Kb BRAMs, the maximum port width that can be configured is 36. In the case of 8-bit data width and $buffer\_size > 4$, the number of 18 Kb BRAMs used in each PE is greater than 2 resulting in a decrease in energy efficiency of *Cache-based architecture*. Hence, we consider $buffer\_sizes$ of 2 and 4 in our experiments. Fig. 7 shows the energy efficiency of *Cache-based architecture*. Similar to the *BRAM-based architecture*, as the number of PEs increases, the percentage of memory utilized in BRAM increases while the IO power remains constant resulting in an increase in energy efficiency. When compared to our *BRAM-based architecture*, the *Cache-based architecture* on the average improves 1.17x with respect to energy efficiency.

3.6. Performance Comparison

Xilinx LogiCORE IP Linear Algebra Toolkit (LAT) [3] provides a highly optimized implementation of matrix multiplication. However, a direct comparison with their work may not be apt as the Xilinx matrix multiplication core is

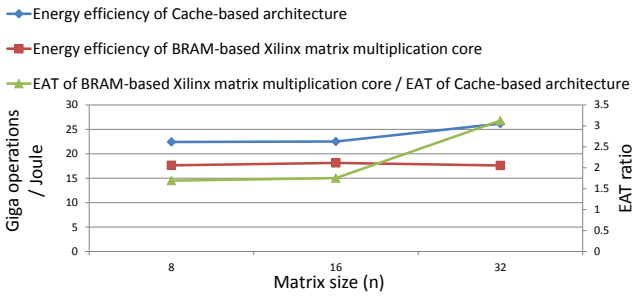

Fig. 8. Comparison between *Cache-based architecture* and Xilinx matrix multiplication core for EAT and energy efficiency

AXI4-stream compliant. We compare between EAT and energy efficiency of our *Cache-based architecture* with that of BRAM-based Xilinx matrix multiplication design. Xilinx matrix multiplication core can multiply only up to 32×32 matrices. Fig. 8 shows the comparison for 8-bit data width. Our *Cache-based architecture* improves on the average 2.2x and 1.33x with respect to EAT and energy efficiency, respectively.

4. CONCLUSION

In this work, we explored the matrix multiplication for energy efficiency. Analytically we determined that memory energy limits the scalability of matrix multiplication. We then explored the design space generated by the types of memory storage. We studied the power consumption of different components and proposed a cache based optimization to reduce the dominant BRAM energy. Compared to the state-of-the-art matrix multiplication core, our *Cache-based architecture* performs 2.2x and 1.33x for EAT and energy efficiency, respectively. In the future, we plan to study the suitability of the memory activation schedule technique using cache optimization on other kernels.

5. REFERENCES

[1] Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev. 64-bit floating-point fpga matrix multiplication. In *Proceedings of the 2005 ACM/SIGDA FPGA*, pages 86–95, 2005.

[2] J.-W. Jang, S. B. Choi, and V. K. Prasanna. Energy-and time-efficient matrix multiplication on fpgas. *IEEE Transactions on VLSI Systems*, 13(11):1305–1319, 2005.

[3] Xilinx. Logicore ip linear algebra toolkit v1.0. *http://www.xilinx.com/support/documentation/ip_documentation/linear_algebra_toolkit/v1_0/ds829_linalg_toolkit.pdf*.

[4] Xilinx. Logicore ip multiply adder v2.0. *http://www.xilinx.com/support/documentation/ip_documentation/xbip_multadd_ds717.pdf*.

[5] L. Zhuo and V. K. Prasanna. Scalable and modular algorithms for floating-point matrix multiplication on reconfigurable computing systems. *IEEE TPDS*, 18(4):433–448, 2007.