Evaluating Energy Efficiency of Floating Point Matrix Multiplication on FPGAs

Kiran Kumar Matam  
Computer Science Department  
University of Southern California  
Email: kmatam@usc.edu

Hoang Le and Viktor K. Prasanna  
Ming Hsieh Department of Electrical Engineering  
University of Southern California  
Email: {hoangle, prasanna}@usc.edu

Abstract—Energy efficiency has emerged as one of the key performance metrics in scientific computing. In this work, we evaluate the energy efficiency of floating point matrix multiplication on the state-of-the-art FPGAs. We implement a modular design parameterized with the problem size and the type of on-chip storage. To understand the efficiency of our implementations, we estimate the peak energy efficiency of any matrix multiplication implementation. Our on-chip matrix multiplication core achieves up to 7.07 and 2.28 GFlops/Joule for single and double precision arithmetic, respectively. Our implementations sustain up to 73% and 84% of the peak energy efficiency for single and double precision arithmetic, respectively. Using an optimal on-chip matrix multiplication core, we also model and estimate the energy efficiency of large-scale matrix multiplication using external DRAM. Our designs for large-scale matrix multiplication achieve energy efficiency of 5.21 and 1.60 GFlops/Joule for single and double precision, respectively.

I. INTRODUCTION

State-of-the-art FPGAs offer high operating frequency, unprecedented logic density and a host of other features. As FPGAs are programmed specifically for the problem to be solved, they can achieve higher performance with lower power consumption than general-purpose processors. Therefore, FPGA is a promising implementation technology for computationally intensive applications such as signal, image, and network processing tasks [10], [5].

Floating point matrix multiplication is one of the key kernels in scientific computing. It is used as a building block for many fundamental linear algebra kernels. Several architectures and algorithms [2], [15] for floating point matrix multiplication have been proposed over the years. Among them, linear array architectures with simple layout can map suitably on to the FPGA [16]. On a state-of-the-art device they can achieve up to 300 GFlops/sec for single precision and are also scalable [16]. Several linear array architectures have been proposed [3], [15] to optimize for latency, throughput, and area. However, they have not explored the design space for energy efficiency.

Power is a key metric in scientific computing today. The total system power is a major component of cost and availability. In this work, we explore the design space of floating point matrix multiplication on FPGAs. Next, we determine an upper bound on the energy efficiency of any matrix multiplication implementation. To understand the efficiency of our implementations, the sustained energy efficiency of our implementations is compared with the peak energy efficiency of the device. Using a given floating point cores, we also model and estimate the energy efficiency of the large-scale matrix multiplication using external DRAM. This paper makes the following contributions:

1) A parameterized floating point matrix multiplication implementation. Parameters are problem size, and type of memory on FPGA (Section III).
2) Evaluation of the effect of using various types of storage available on FPGA on the energy efficiency of the floating point matrix multiplication (Section IV-D).
3) An upper bound on the energy efficiency of any matrix multiplication implementation on a given target device (Section III-A).
4) Implementations which can sustain up to 73% and 84% of the peak energy efficiency for single and double precision arithmetic, respectively (Section IV-D).
5) A model and an estimate of the energy efficiency of large-scale matrix multiplication using external DRAM (Section V). For 8Kx8K matrix multiplication, we estimate an energy efficiency of 5.21 and 1.60 GFlops/Joule for single and double precision, respectively.

The rest of the paper is organized as follows. Section II covers the background and related work. Section III describes the architecture and algorithm used in the work. It also describes the architecture used to estimate the peak energy efficiency of any matrix multiplication implementation. Section IV presents the experimental results and performance evaluation. Section V presents the model and energy efficiency estimation results for large-scale matrix multiplication using external DRAM. Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Background

Given two matrices $A, B$ of size $n \times n$, the product $AB$, denoted $C$, is defined as: $C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$, $1 \leq i, j \leq n$. In this paper we consider the well known three nested loop $O(n^3)$ complexity algorithm only. In the rest of the paper we denote matrix multiplication, single precision, and double precision as MM, SP, and DP, respectively.

Many architectures and algorithms have been proposed to compute the product matrix $C$ [3], [15]. In this work, we adapt the Algorithm 1 proposed in [6], as it can be efficiently implemented on a linear array architecture.

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

To the best of our knowledge, there has been no previous work targeted at exploring the design space for energy efficiency of floating point MM on FPGAs. Most of the prior work has mainly focused on optimizing the latency and the area of the design.

In [15], [16], we developed scalable algorithms considering various design parameters such as the amount of bandwidth, on-chip storage, configurable slices available. Considering several resource constraints such as the size of on-chip memory, the number of PEs, we derived algorithms achieving optimal latency.

Floating point MM on a linear array architecture was considered in [3]. They presented a master-slave scheduling algorithm for block matrix multiplication. A bit-level algorithm for block matrix multiplication. A bit-level algorithm was implemented on FPGAs in [1]. A disadvantage of this architecture is that it requires an I/O bandwidth that is proportional to the problem size. This requirement limits the scalability of the algorithm. In [7], FPGAs were used as accelerators when computing floating point MM. In this context, they proposed a optimization for FPGAs when input matrices are given one after the other in a streaming fashion.

The algorithm performs $n \times n$ matrix multiplication in $n^2 + 2n$ cycles using 3 I/O ports and $n$ processing elements, each having a multiplier, an adder, 4 registers, and 2 local memories of $n$ words. In this work, we carefully pipeline the architecture to increase the frequency.

Algorithm 1 Operation of each PE

1: {During $t = 1$ to $n$}
2: for all $j, 1 \leq j \leq n$, do in parallel do
3: $PE_j$ shifts data in BU right to $PE_{j+1}$
4: end for
5: {During $t = n + 1$ to $n^2 + n$}
6: for all $j, 1 \leq j \leq n$, do in parallel do
7: $PE_j$ shifts data in A, BU right to $PE_{j+1}$
8: if $BU = B_{k,j}$ then
9: copy it into BL or BM (alternatively)
10: end if
11: if $A_{\text{reg}} = A_{i,k}$ then
12: $C_{i,j} = C_{i,j} + A_{i,k} \times B_{k,j}$
13: $(B_{k,j}$ is in either BM or BL)
14: $(C_{i,j}$ is in Cbuf)
15: end if
16: end for
17: {During $t = n^2 + 1$ to $2n^2$}
18: for all $j, 1 \leq j \leq n$, do in parallel do
19: $PE_j$ stores input $C_{in}$ in Cbuf
20: end for
21: {During $t = n^2 + 1$ to $n^2 + n$}
22: $PE_j$ output data $C_{i,j}$ to $PE_{j-1}$
23: {During $t = n^2 + n + 1$ to $2n^2 + n$}
24: $PE_j$ output data Cbuf to $PE_{j-1}$

III. Architecture

In this work we adapt the architecture and algorithm proposed in [6] for floating point matrix multiplication. This algorithm achieves an optimal latency of $O(n^2)$ using $n$ PEs for $n \times n$ matrix multiplication. The amount of storage within each PE is linearly proportional to the problem size. Also the layout is simple as all the inter-connections are localized and are within a PE or in between adjacent PEs only. Fig. 1 shows the overall architecture. Fig. 2 shows the architecture of a PE.

Algorithm 1 shows the operations performed by each PE. $PE_j, 1 \leq j \leq n$, denotes the $j^{th}$ PE from the left. $PE_j$ computes the $j^{th}$ column of $C$. In the $k^{th}$ iteration, $k^{th}$ row of $A$ and $k^{th}$ column of $B$ traverse in order from $PE_1$ to $PE_n$. $PE_j$ computes $A_{i,k} \times B_{k,j}$ and accumulates in $C_{i,j}$ which is stored in Cbuf array, which is used to store intermediate output values. After the output elements are computed, each $PE_j$ transfers them to $PE_{j-1}$ using Cbuf array. As the output transfer of one MM can be overlapped with the computation of another MM, this architecture can perform MM when input matrices are given one after the other in a streaming fashion.

A. Minimal Architecture

The energy efficiency of any MM implementation is upper bounded by the inherent peak performance of the platform. This depends on the target device and the IP cores used.
to perform the arithmetic operations. We measure this peak energy efficiency by using a “minimal” architecture for the processing element under ideal conditions. Thus, we ignore all the overheads such as memory energy, I/O, access to memory, cache and other buffers that may be employed by an implementation. This minimal architecture performs only multiplication and addition as these are the basic operations in any MM design. The other components such as routing, memory will only increase the energy consumption. Such a minimal architecture is shown in Fig. 3.

We use this peak energy efficiency as an upper bound on the performance of any algorithm and architecture for MM and compare the sustained performance of an implementation against this bound. Note that as the FP cores improve, we can expect a corresponding increase in the sustained performance of our algorithm and the architecture for MM. Also note that this peak performance bound also applies to any other floating point based computations (matrix vector product, matrix decomposition, etc.) that employ the above floating point cores.

IV. EXPERIMENTS AND EVALUATION

A. Implementation Details

The architecture was implemented in Verilog, using Xilinx ISE 14.4, with Virtex-7 XC7VX690T with -3 speed grade as the target. Xilinx floating point addition and multiplication cores [12] were used in our implementation. We used non-blocking interface, maximum latency and maximum DSP usage as configuration options when generating these cores. These cores are IEEE-754 compliant with some minor deviations. As we are interested in the power consumed by the architecture, we considered only the dynamic power in our experiments. After the post place and route we measured the power using Xpower estimator [14]. All our results are reported at operating frequencies unless otherwise mentioned. We used 50% toggle rates in all our experiments, which is the worst case for arithmetic-intensive modules [13].

B. Performance Metric

We consider energy efficiency as the metric for performance evaluation. 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 MM algorithm, energy efficiency is given by $2n^3 / \text{energy consumed by the design}$. Energy of the design $= \text{time taken by the design} \times \text{average power dissipation of the design}$. Alternatively, Energy efficiency of the design is Power efficiency (number of operations per second/Watt).

C. Peak Energy Efficiency

The minimal architecture described in Section III-A is used to estimate the peak energy efficiency of any MM implementation. Based on the experiments using the implementation details mentioned in Section IV-A, this minimal architecture can operate at a maximum frequency of 581 MHz (405 MHz), consumes 120 mW (297 mW) when operating at maximum frequency, and occupies 299 slices (748 slices) for single precision (double precision) arithmetic.

Energy efficiency for this design is given by $2 \times \text{frequency/power consumed by the design}$. For identifying the frequency at which this design achieves peak energy efficiency, we measured the power consumed by the design at intervals of 50 MHz in the range of 1 to maximum frequency. Peak energy efficiency is observed at 550 MHz and 400 MHz frequency for single and double precision arithmetic, respectively. The peak energy efficiency at 50% toggle rate is 9.70 GFlops/Joule (7.27 GFlops/Joule) for single precision (double precision) arithmetic.

D. Performance Results

In this subsection we discuss the experiments with the floating point MM design described in Section III. We denote the architecture using BRAM and Distributed RAM as BRAM-based architecture and Dist.-RAM based architecture, respectively. We consider matrix sizes in powers of two till the largest possible MM design that can fit on the target FPGA. As the $Cbuf$ array (partial results array) is read and written in a round robin fashion, matrix column size should be greater than the latency of the floating point addition so that the $C_{i,j}$ (partial result) from $Cbuf$ array is read after it is updated with the partial output value ($A_{i,k} \times B_{k,j}$). So we chose the minimum matrix size such that it is greater than the latency of the floating point addition.

For $512 \times 512$ single precision ($128 \times 128$ double precision) BRAM-based architecture, post place and route results show that our implementation achieves 279 GFlops/sec (67 GFlops/sec) and consumes 39 W (30 W) of dynamic power when operating at peak frequency of 274 MHz (264 MHz), occupies 99,585 slices (67,125 slices), uses 2,560 DSPs (1,792 DSPs) and 1,024 18 Kb BRAMs (256 36 Kb BRAMs).

For an $n \times n$ MM, the number of operations performed is $2n^3$. The proposed architecture completes it in $n^2 + 2n$ 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 floating point 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 $O(n^4)$, 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.
A. Architecture

Large-scale matrix multiplication can be efficiently performed using tiled/blacked matrix multiplication. Detailed procedure for tiled/blacked matrix multiplication is shown in Algorithm 2 [4]. In our design, the tile (or core) size is chosen to maximize energy efficiency. Due to limited amount of on-chip memory on FPGAs, external DDR3 SDRAMs are utilized to store the input matrices and the product matrix. The overall architecture for performing large-scale MM is shown in Fig. 10.

There are 2 SDRAM banks, one to store the input matrices A, B, and the other to store the product matrix C. Each
Fig. 10: Large-scale matrix multiplication architecture

SDRAM bank is controlled by a memory controller, which handles communication with the high-speed interface of the DDR3. We chose to implement DDR3 SDRAM utilizing the maximum allowed burst length of 8 to ensure maximum data throughput.

The major drawbacks of DDR3 SDRAM are the long access latency and the periodic refreshes (required to retain the content of the SDRAM). Both of these issues can be alleviated by taking advantage of the intrinsic high bandwidth of SDRAM, combined with the use of a suitable FIFO (First In First Out) queues. FIFOs are required to connect (or synchronize) producers and consumers with different data rates. In our case, FIFO 1 synchronizes the reading of input matrices A, B from DDR3 memory to the MM core. FIFO 2 synchronizes the writing of output product matrix C from the MM core to the DDR3 memory.

The control unit coordinates the operation of the entire design. As the timing of the operations in MM can be pre-determined, control unit can be implemented using several counters. During a block operation, the partial results are kept in the MM core and are accumulated with the results in the next iteration. Once a block operation finishes, the product sub-matrix C is streamed out from the MM core to FIFO 2, and is written into the DDR3 eventually. During the streaming out process, the inputs of the next block iteration can be streamed in.

B. Implementation on FPGAs

DDR3 SDRAM memory is organized in pages, each of size 8 KB. Data within the same page can be accessed at the peak bandwidth offered by the DDR3 SDRAM (about 6.4 GB/sec); this is called page hit. The input data should be arranged based on their access order to maximize the page hit rate. As mentioned in Section III, our MM core accepts input matrices A in column major order and input matrix B in row major order. Therefore, matrices A and B are stored in the external SDRAM in column and row major, respectively.

Current DDR3 SDRAM chips are available in 1 GB, 2 GB, 4 GB and 8 GB sizes. These chips have 8 K refresh cycles (NRC), 50.625 ns refresh cycle time (TRC), and 64 ms refresh interval (TREF). The minimum burst refresh time is TREF = TRC × NRC. Let F be the operating frequency of our MM core. The number of clock cycles required to refresh SDRAM is NCycles = TREF × F. During the refresh time, the first FIFO needs to have enough data to feed to the MM core. Therefore, FIFO 1 must be able to store at least Lmin = 2 × NCycles elements (a pair of {A and B elements} is required by the MM core per cycle). Depending on the block size and the target FPGA device, these FIFOs can be implemented using either on-chip BRAM, off-chip SRAM, or a combination of both. Let W denote the data width (32 for SP and 64 for DP). The size of SRAM needed to implement this FIFO is

\[ S_{SRAM} = L_{min} \times W = T_{RC} \times N_{RC} \times F \times 2 \times W \]

If \( F = 300 \text{ MHz} \), then \( S_{SRAM} = 50.625 \text{ ns} \times 8 \text{K} \times 300\text{M} \times 2 \times W = 243\text{K} \times W \) bits. Hence, the size of SDRAM needed to implement FIFO 1 is 7.78 Mb and 15.55 Mb for single and double precision, respectively.

On the other hand, FIFO 2 only requires to store the entire partial result \( C_{i,j} \), in the worst case. The size of SRAM needed for FIFO 2 is \( m^2 \times W \). Thus, the size of this FIFO is 2 Mb for single precision (with \( m = 256 \)) and 1 Mb for double precision (with \( m = 128 \)).

C. Energy Efficiency Estimation

The total power consumption of the design can be computed as:

\[ P_{MM} = P_{core} + P_{(DDR3)} + P_{(DDR3-Ctrl)} + P_{(FIFO)} + P_{(FIFO2)} + P_{CU} \]

The power consumption of SRAM is estimated using the on-chip BRAM of FPGA, whereas the power consumption of the external DDR3 SDRAM is calculated using Micron DDR3 power calculator [9] for 1 GB target chip. With this chip, our design can handle matrices consisting of 256 M and 128 M SP and DP elements, respectively. The DDR3 memory controllers can be implemented using Xilinx Memory Interface Generator (user interface mode). We assume that the power consumption of the control unit is negligible as it is a simple state machine. We use 256 × 256 (128 × 128) MM architecture operating at 300 MHz (250 MHz) as on-chip MM core for single precision (double precision) arithmetic. Based on these, we obtain the following power consumption values:

\[ P_{core} = 21.61 \text{ W (for } 256 \times 256 \text{ SP MM)} \]
\[ P_{core} = 28.67 \text{ W (for } 128 \times 128 \text{ DP MM)} \]
\[ P_{DDR31} = P_{DDR32} = 1.3 \text{ W} \]
\[ P_{FIFO1} = 2.40 \text{ W (for SP MM)} \]
\[ P_{FIFO2} = 617 \text{ mW (for } 256 \times 256 \text{ SP MM)} \]
\[ P_{FIFO1} = 4.77 \text{ W (for DP MM)} \]
\[ P_{FIFO2} = 308 \text{ mW (for } 128 \times 128 \text{ DP MM)} \]
\[ P_{DDR3-Ctrl1} = P_{DDR3-Ctrl2} = 1.57 \text{ W} \]

The time taken by the design is given by the time taken to compute one output block \( \times \) the number of output blocks. Sequence of events over time for computing one block of output matrix is shown in Fig. 11. The time taken to compute one output block is \((n/m) \times (m^3 + 2m))/F\). So the time taken by large-scale MM design is \((n/m)^3 \times (m^2 + 2m))/F\). Energy efficiency can be calculated as follows:

\[ \text{Energy efficiency} = \frac{2n^3/(P_{MM} \times \text{time})}{(P_{MM} \times (n/m)^3 \times (m^2 + 2m))} \]
Using the above equation for $8192 \times 8192$ MM, the energy efficiency is 5.21 and 1.60 GFlops/Joule for SP and DP, respectively.

**D. Performance comparison**

We compare our estimated energy efficiency of large-scale MM on FPGAs with the energy efficiency of floating point MM on C2075 GPUs. C2075 GPU has 448 cores, each operating at 1.15 GHz frequency, and 6 GB of DDR5 SDRAM memory. The peak floating point performance for the device is 1.03 TFlops/sec and 515 GFlops/sec for single and double precision, respectively. Power consumed by the floating point MMIs on GPU has been reported in [8]. For $8K \times 8K$ MM, the energy efficiency values are 3.51 and 1.65 GFlops/Joule for single precision and double precision, respectively. Single precision MM on FPGAs is $1.48 \times$ more energy efficient than that on GPUs. Double precision MM on GPUs is $1.03 \times$ more energy efficient than MM on FPGAs.

**VI. Conclusion**

In this work we explored the energy efficiency of floating point matrix multiplication on FPGAs. Memory power consumption limits the scalability of the designs. We proposed a minimal architecture to estimate the peak energy efficiency of any matrix multiplication implementation. As the layout is simple our implementations can achieve up to 73% and 84% of the peak energy efficiency for single and double precision arithmetic, respectively. We also extended our on-chip matrix multiplication core to estimate the energy efficiency of the large-scale matrix multiplication using external DRAM. In the future, we plan to study optimizing memory energy for improving energy efficiency of matrix multiplication designs.

**References**


