Energy-Efficient Large-Scale Matrix Multiplication on FPGAs

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

Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Email: prasanna@usc.edu

***Abstract*—Energy efficiency has emerged as one of the key performance metrics in computing. In this work, we present an energy efficient design for large-scale matrix multiplication. As a baseline architecture, we use a highly optimized on-chip matrix multiplication architecture extended to support large matrices using external memory. Based on the matrix multiplication algorithm and the DRAM model, we present an efficient data layout for storing the input matrices. This data layout reduces the energy consumed by the external memory by minimizing the number of row activations in a DRAM. By exploiting the matrix multiplication algorithm, modular structure of the DRAM, and the high bandwidth between the on-chip and the external memory, we propose a memory activation schedule. This memory activation schedule is based on a realistic DRAM model and reduces the memory energy, which is the dominant energy of the design. Our proposed scheme improves the energy efficiency (defined as the number of operations per Joule) of the baseline architecture by $1.6\times$, $1.3\times$, and $1.2\times$ for 32K$\times 32$K 16-bit fixed point, 32K$\times 32$K single precision floating point, and 16K$\times 16$K double precision floating point matrix multiplication, 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.

Matrix multiplication is one of the key kernels in computing. Several architectures and algorithms [24], [4] for matrix multiplication on FPGA have been proposed over the years. For performing matrix multiplication using on-chip resources only, they can achieve up to 300 GFlops/sec for single precision matrices on a state-of-the-art device [13]. However, the limited on-chip memory restricts the matrix sizes that can be supported.

Large-scale matrix multiplication is used as a sub-routine in many important applications such as numerical computing, scientific computing [6]. DRAM with large memory sizes and low cost per bit of storage, can be a feasible option as external memory for storing large matrices. Previous solutions for large-scale matrix multiplication using external memory on FPGAs have focused on optimizing the throughput and area occupied by the design [4], [25]. However, they have not explored the design space for optimizing the energy efficiency of matrix multiplication.

Power is a key metric in computing today. The total system power is a major component of cost and availability. In this work, we explore large-scale matrix multiplication using external memory. To reduce the external memory energy, we present an efficient data layout for storing the input matrices based on the block/tiled matrix multiplication algorithm and DRAM model. This layout reduces the energy consumed by the external memory by minimizing the number of DRAM row activations. To reduce the memory energy of the design, based on a realistic DRAM model we present an activation schedule. This activation schedule exploits the matrix multiplication algorithm, modular structure of the DRAM, and the high bandwidth between the on-chip and the external memory. This paper makes the following contributions:

- A matrix multiplication design based on a realistic DRAM model to support large matrices (Section IV).
- Efficient data layout for storing matrices to minimize the energy dissipation of the design (Section IV-C).
- An activation schedule to reduce the memory energy, which is the dominant energy of the design (Section V).
- Significant energy efficiency improvement for 16-bit fixed point, single precision and double precision numbers (Section VI-C).

The rest of the paper is organized as follows. Section II covers the background and related work. Section III describes the DRAM model. Section IV describes the large-scale matrix multiplication architecture. It also describes the data layout used to store the matrices in external memory. Section V describes the proposed activation schedule. Section VI presents implementation details and experimental results. Section VII concludes the paper.

II. Background and Related Work

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 classical $O(n^3)$ complexity algorithm only. In the rest of the paper we denote this matrix multiplication as MM.

A. Related Work

To the best of our knowledge, there has been no previous work targeted at exploring the energy efficiency of large-scale MM using external memory on FPGAs. Most of the prior work

has mainly focused on optimizing the latency and the area of the design.

In [24], [25], we developed scalable algorithms considering various design parameters such as the amount of bandwidth, the on-chip storage, and the available configurable slices. Considering several resource constraints such as the size of on-chip memory, the number of PEs, we developed algorithms achieving optimal latency. Floating point MM using blocking technique was considered in [4]. A master-slave scheduling algorithm was presented for optimizing the throughput and area of the design.

There has been work in using external memory for other BLAS kernels on FPGA [7], [5]. For sparse-matrix vector multiplication kernel, a data reordering scheme was presented in [7]. This data reordering scheme improved the cache-misses when accessing the vector elements from the external memory. Most of the prior work in using external memory for other BLAS kernels on FPGA, was on optimizing the latency and the area of the designs. Also the techniques used in these works are directly not applicable for large-scale MM.

Non-linear data layouts such as recursive layouts [2] and block data layout [15] for MM have been explored for cache-based systems, to reduce cache pollution, cache conflict misses, and TLB misses. However, in this work, we explore the block data layout for reducing the energy consumed by the DRAM. In the context of compilers, estimating the required number of row activations in DRAM using the block data layout was presented in [11]. By using the block data layout, when compared with the canonical row/column data layouts, they reduce the energy consumed by the design, by reducing the number of row activations in the DRAM. However, in our proposed block data layout, by interleaving the data across the rows of different banks, we not only reduce the number of row activations but also the DRAM access time. By reducing the DRAM access time, we further reduce the energy consumed by the MM design by using our activation schedule scheme (Section V).

FPGA-based linear array algorithms for fixed-point MM are shown in [9]. These algorithms achieve optimal latency of $O(n^2)$ using n PEs. Using one of these linear array algorithms, we explored the design space for energy efficiency on the state-of-the-art FPGAs for fixed point MM in [12]. A buffer-management based activation schedule was proposed to reduce the memory energy of the on-chip MM. The design space for energy efficiency of floating point MM was explored in [13]. We developed an upper bound on the energy efficiency of any floating point MM implementation on a target device. We also extended our on-chip MM design to use external memory for performing large-scale MM.

In this paper, we extend the work in [12], [13] to explore large-scale matrix multiplication using external memory. Based on the block matrix multiplication algorithm and DRAM model we present an efficient data layout to reduce the energy consumed by the design. We also present a memory activation schedule based on a realistic DRAM model, to reduce the memory energy of the design.

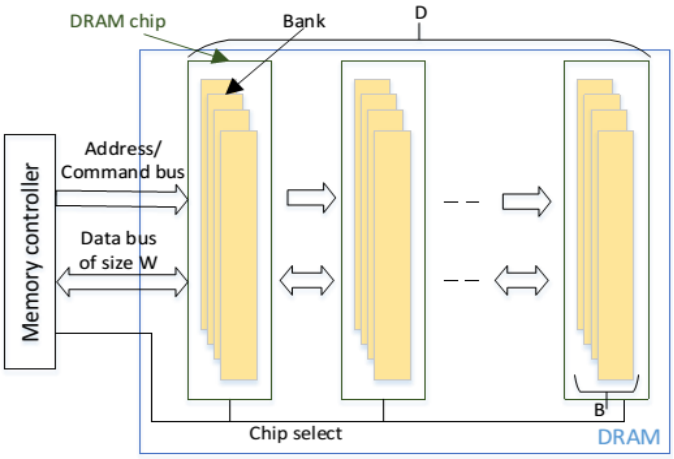

Fig. 1: Organization of the DRAM module

F	DRAM chip frequency
H	Number of columns in a DRAM row
I	Number of rows in a bank
G	Number of banks in a DRAM chip
D	Number of DRAM chips
w	Data access width of a DRAM chip

TABLE I: Parameters of the DRAM model

III. DRAM Model

In this work, due to limited amount of on-chip memory on FPGAs, external DRAMs are utilized to store the input matrices and the product matrix. In this section, we present the DRAM organization, access and power models that are used in this paper [18], [14].

A. DRAM Organization

Each DRAM chip is organized as several banks. Each bank is a two dimensional matrix of locations. At each addressable location ([Bank, Row, Column]), fixed number of data bits (usually 8-bits) are located. This fixed number of data bits is also the data access width of the chip. Several chips together form a rank. The data access width of the rank is the sum of the data access widths of all the chips in that rank. Several ranks together constitute a module. The data access width of the module is same as the data access width of the rank. Data from only one of the ranks can be transferred at a time.

A chip select signal is used to select the rank from which the data is accessed. DRAM chips in a given rank operate in unison while accessing the data. Also, a given data element can be accessed in a single cycle from DRAM by partitioning the data element and storing it among the DRAM chips in the same rank. In our DRAM model, we assume a data element can be accessed from DRAM in one cycle. In our DRAM model, concatenation of the corresponding rows of the chips in the rank is considered as one row. In our DRAM model, a rank consists of one DRAM chip with several banks, each bank consists of several rows. Fig. 1 shows the DRAM organization. Table I describes the parameters of the DRAM model. These parameters will be used in further sections of the paper.

B. DRAM Access Latencies

To perform a read/write at a location we first need to activate the row. After activating the row, read/write operations at a column index can be issued. An operation on an active row is defined as page-hit access. In page-hit access, as the row is already active, it is ready to service the request immediately.

Several time constraints are involved when accessing DRAM; time constraints used in this work are described here (latency values for the time constraints mentioned here are for 4Gb DDR3 SDRAM):

t_{RC}	Minimum time between issuing two successive row activations to the same bank, 46.1 ns.
C_{CCD}	Minimum number of cycles between successive accesses to the same bank and row, 4 cycles.

In matrix multiplication, the addresses of the accessed input data can be pre-determined. Also, DRAM technology supports pipelining of successive read/write requests to an active row. A read/write command to a location in a row initiates transfer of several contiguous locations (burst length, BL), transferring each location over a cycle. By pipelining the access requests and choosing $BL > C_{CCD}$ cycles, data can be accessed continuously from an active row in successive cycles without incurring any gaps in cycles.

C. DRAM Power Model

Total average power consumed by a DRAM chip is sum of several sub-component powers which are consumed during various operating conditions. These sub-component powers include background power, activate power, read power, write power, I/O termination power, and refresh power. The background power depends on the power mode in which the DRAM chip operates. A DRAM chip can operate in one of the two power modes, *standby mode* (or *active mode*) and *power-down mode*. Power-down mode is a power saving mode in which several commands like activate, read, write, precharge are disabled. In each mode DRAM chip can be in one of the two states, 1) either all the banks are closed/precharged or 2) one or more banks are active. In power-down mode and standby mode, the background current consumed when all the banks are closed/precharged, is less than the current consumed when one or more banks is active.

For activating a DRAM row, significant amount of power is consumed for a short duration of time. Average activation power of a memory access pattern depends on the number of row activations. After the DRAM row is active, read/write can be performed on it. Average read/write power dissipated for performing read/write operations on the DRAM, depends on the number of read/write operations performed. Also, power is consumed by the bus to drive the transfer of data between the DRAM and the memory controller, this is I/O termination power. The average I/O termination power depends on the total number of read/write operations performed on the DRAM. The final power component is refresh power, which is the power consumed for refreshing the capacitors in DRAM.

For the sake of analysis, in this paper we use a simple DRAM model.

1) A rank consists of one DRAM chip whose accesses width (w) is 64-bits. Parameters related to DRAM are shown in Table I.
2) Bandwidth between the DRAM and the FPGA is $2 \times F \times w$.
3) From an active DRAM row, k elements can be accessed over k cycles, $k >= 1$.
4) Let P_d be the power consumed by a DRAM chip when all its banks are pre-charged and it is operating in power-down mode.
5) Let P_s be the power consumed by a DRAM chip when it is operating in standby mode and any of its bank is active.
6) Let γ be the fraction of time a DRAM chip operates in standby mode for performing refresh. Typically γ is less than 5%, in distributed type refresh, for every 7.8μs, 350ns are used for performing refresh.

IV. Architecture

In this work, we adapt the architecture and algorithm proposed in [9] for on-chip MM core (denoted On-chip MM). This algorithm achieves an optimal latency of $O(m^2)$ using m PEs for $m \times m$ MM. Also the layout is simple as all the interconnections are localized and are within a PE or in between adjacent PEs only.

For $m \times m$ MM, On-chip MM consists of m identical PEs. Each PE includes a multiplier and adder, 4 registers, and 2 m-word local memories. The On-chip MM architecture can perform a $m \times m$ MM in $m^2 + 2m$ 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 m cycles behind matrix B in column major order, starting from the left most column. The partial results are accumulated at each PE in an auxiliary memory of size m elements, $CBuf$. Once the multiplication of A and B is completed, an m elements size memory at each PE, $COBuf$, is used to transfer the resulting matrix C in the column major order, starting from the left most column. The detailed algorithm is described in [9].

Algorithm 1 Block Matrix Multiplication Algorithm

```
{Matrices A and B are read in to on-chip memory in blocks of size m × m}
{Let A(i,j), B(i,j) and C(i,j) be the i^th row block and j^th column block in matrices A, B and C respectively}
for i = 1 to n/m do
    for j = 1 to n/m do
        Initialize output block C(i,j) to 0
        for k = 1 to n/m do
            Read block A(i,k) from DRAM
            Read block B(k,j) from DRAM
            Perform matrix multiplication on A(i,k) and B(k,j) blocks
            Add the output to C(i,j)
        end for
        Write C(i,j) to DRAM
    end for
end for
```

A. Large-Scale Matrix Multiplication

Large-scale MM can be efficiently performed using tiled/blocked MM. Detailed procedure for tiled/blocked MM is shown in Algorithm 1 [6]. The input matrices and the product

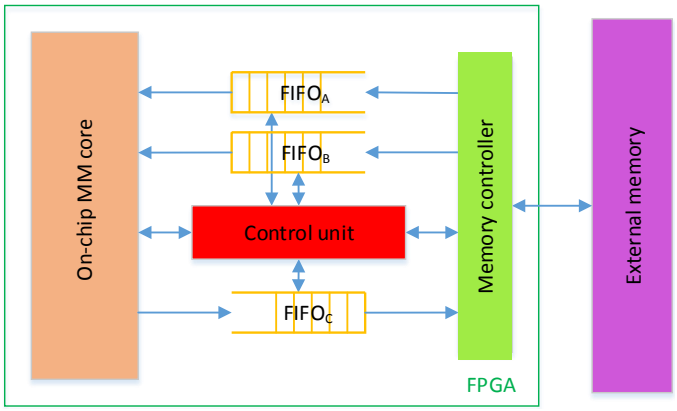

Fig. 2: Large-scale matrix multiplication architecture

matrix are stored in external DRAM. The overall architecture for performing large-scale MM is shown in Fig. 2.

As DRAM and the On-chip MM core can operate at different frequencies, FIFO's are used to synchronize the data access from the On-chip MM core to the DRAM. $FIFO_A$ and $FIFO_B$ are used to synchronize the reading of the input matrices A and B, respectively, from the DRAM memory to the On-chip MM core. $FIFO_C$ is used to synchronize the writing of the product matrix C, from the On-chip MM core to the DRAM 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 On-chip MM core and are accumulated with the results in the next iteration. The partial results are accumulated in $CBuf$ array at each PE of the On-chip MM core. Once a block of output sub-matrix C is produced, it is transferred to $COBuf$ array at each PE of the On-chip MM core; eventually it is streamed out to DRAM using the $FIFO_C$. During the streaming out process, the inputs for the next block MM can be streamed into the On-chip MM core.

Table II shows several parameters associated with the On-chip MM core. These parameters will be used in further sections of the paper.

$m \times m$	Size of the matrices multiplied by the On-chip MM core
f	Frequency of the On-chip MM core
FI_I	Size of $FIFO_A$ and $FIFO_B$
FI_O	Size of $FIFO_C$

TABLE II: Parameters of the On-chip MM core

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 O(n^3) MM algorithm, energy efficiency is given by $2n^3$/energy consumed by the design. Energy consumed by the design = time taken by the design $\times$ average power dissipation of the design. Alternatively energy efficiency

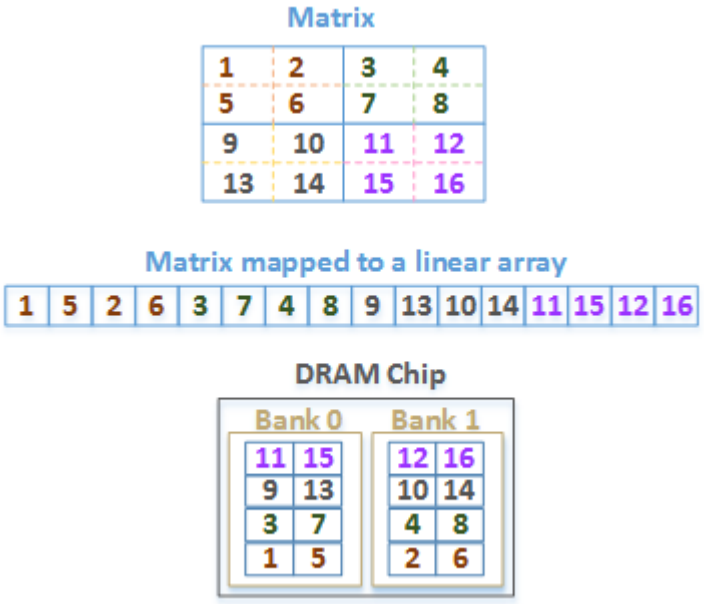

Fig. 3: Data layout for storing matrices

of the design is *Power efficiency* (number of operations per second/Watt).

C. Data layout

In block matrix multiplication, input matrices are accessed in two dimensional blocks. Canonical data layouts such as row/column-major layouts store the adjacent elements in row/column in contiguous locations. Therefore, elements of a block are not stored in contiguous locations. So, for accessing next row/column elements of a block, we may need to access another row of the chip. Also if the adjacent row/column of the block is stored in the same bank then t_{RC} delay needs to be incurred in accessing the data.

Using the block data layout [15], we present a data layout to store the input matrices in DRAM. To describe the data layout, we map the matrix elements to a linear array. Also the locations of a DRAM chip are given a linear index. The matrix elements in the linear array are stored in DRAM chips at the corresponding indexes.

Let b_i be the i^{th} bank of the DRAM chip, b_i^r be the r^{th} row of b_i bank, and $b_i^r[l]$ be the l^{th} location in the row b_i^r. The location $b_i^r[l]$ is given a linear index $G \times H \times b_i^r + H \times b_i + b_i^r[l]$, i. e. adjacent elements in a row of a bank are given contiguous indexes and the indexing of rows is interleaved across the banks.

As input and output matrices are transferred between DRAM and On-chip MM core in blocks of size $m \times m$, these matrices are divided into two dimensional blocks of size $m \times m$. An example of 4×4 matrix divided into 2×2 blocks is shown in Fig. 3. The adjacent blocks in a row (column) of matrices A and C (matrix B) are mapped to contiguous locations in the linear array. The adjacent block rows (columns) are mapped to contiguous locations in the linear array. The blocks of A and C matrices (B matrix) are stored in column-major format (row-major format). An example data layout for

a 4×4 A matrix divided into 2×2 blocks and stored in a DRAM chip with $G = 2$, $I = 4$, and $H = 2$ is shown in Fig. 3.

In this data layout, elements within the blocks are stored according to the input data access order of the program. This data layout exploits the spatial locality in the data accesses of the MM core. Unlike the row/column major layout, in this data layout, the elements in the block of a matrix are stored in contiguous locations. Therefore, for an activated row, the number of accesses to it are maximized. By maximizing the number of accesses to an activated row, the total number of row activations required are minimized; hence reducing the energy consumed by the DRAM. For $n \times n$ MM, the total number of DRAM row activations required by the proposed block-data layout is $2 \times (\frac{n}{m})^2 \times (\left\lceil \frac{(\frac{n}{m}) \times m^2 \times W}{H \times w} \right\rceil + 1) + (\frac{n^2 \times W}{H \times w})$, where W is the data-width of the matrix element. For row/column major layout, when the size of a DRAM row is divisible by the size of a row in a block, the total number of DRAM row activations is $2 \times (\frac{n}{m})^3 \times m + (\frac{n}{m})^2 \times m$.

In the row/column major layout, as the rows/columns of a block are not contiguously stored, the next accessed DRAM row can be in the same bank. The minimum time required for activating the row in the same bank is t_{RC}. In the worst case the next accessed row in the block can always be located in the same bank and the row activation time cannot be overlapped with the data access. In this case, the overhead of row activation times for rows in a block is $m \times t_{RC}$. With the proposed data layout the next accessed DRAM is another bank and row activation can be overlapped with the data access. Therefore, when compared to the row/column major layout our proposed data layout can reduce the time for accessing one block of data by up to $m \times t_{RC}$. Minimizing the DRAM access time helps us to reduce energy by using the activation schedule described in Section V.

D. Energy Dissipation Analysis

In this subsection, we present an asymptotic analysis for the energy dissipation. We consider the architecture and algorithm described in Section IV.

For $n \times n$ MM with $m \times m$ block size, number of block MM operations performed are $(n/m)^3$. Each block MM operation takes $m^2 + 2m$ cycles. Therefore, the total time for computation is $O(n^3/m)$. Total energy consumed by the arithmetic units is $O(n^3)$, where constant units of energy is assumed to be consumed for performing an arithmetic operation. Amount of on-chip memory is $O(m^2)$ and off-chip memory is $O(n^2)$. Energy consumed by the on-chip memory units is $O(n^3 m)$, where constant units of energy is assumed to be consumed in a cycle for a unit of storage. Energy consumed by the off-chip memory units is $O(n^5/m)$, where constant units of energy is assumed to be consumed in a cycle for a unit of storage in a DRAM chip which is operating in active power mode. Therefore, total energy consumed by the memory units is $O(n^3 \times (m + n^2/m))$. All the communication is in between the adjacent PEs, in between the components within the PEs, in between the on-chip and external memory, or in between adjacent units such as memory controller, FIFOs only. Therefore the energy consumed for communication is $O((n^3/m) \times m)$, where unit energy is assumed to be energy for transferring a word of data within a PE, between adjacent PEs or between adjacent components. Total energy consumed by the design is sum of the computation, memory, and communication energies. The total number of operations performed in MM is $2n^3$. Energy efficiency of the design is $O(m/n^2)$ as $n >> m$; memory energy limits the scalability of the large-scale MM design.

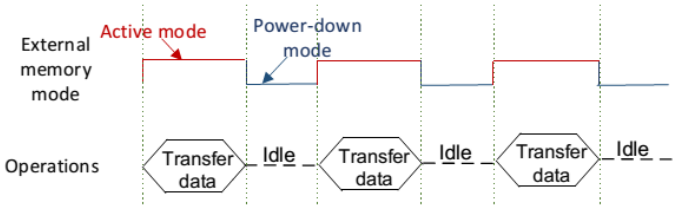

Fig. 4: Activation schedule for external memory

V. Memory Activation Schedule

In this section we present an optimization technique to reduce the memory energy. By suppressing the switching activity of the idle components, dynamic power can be saved. During a time interval, not all the components are accessed. In this subsection, we identify the time interval for several components in which their switching activity can be suppressed.

A. Activation Schedule for On-chip Components

In block matrix multiplication algorithm, output block is accumulated in the on-chip memory and written only once to the external memory. When computing the output block, the switching activity can be suppressed in several components that are used for transferring the output block from on-chip memory to external memory. $COBuf$ memory in each PE, used for transferring block of output, is accessed during the block operation where output block is produced. The output block is produced after multiplying last column block of a row in A with the last row block of a column in B. The $COBuf$ memory also needs to be active in the next block MM operation where the result is transferred to DRAM memory via the $FIFO_C$. In the other block MM operations the switching activity of $COBuf$ memory can be suppressed. Similarly, $FIFO_C$ is accessed only when transferring the output block matrix to the DRAM memory.

B. Activation Schedule for External Memory

For storing large matrices in external memory several DRAM chips may be required. Let $\frac{D}{3}$ DRAM chips be required for storing each of the matrices A, B, and C. In the block matrix multiplication algorithm, spatial locality in data accesses exists. From DRAM, matrices A and C (B) are accessed in blocks, in a row-major order (column-major order). Therefore, the matrices A and C (B) can be partitioned in to $\frac{D}{3}$ groups of adjacent rows of blocks and each such group can be stored in a DRAM chip to exploit the spatial locality in data accesses. For a matrix, when accesses to a group of adjacent rows of blocks is being performed, the corresponding DRAM chip can be operated in active mode and the remaining $(\frac{D}{3} - 1)$ DRAM chips storing the matrix can be operated in power-down mode. By operating the unaccessed DRAM chips in power-down mode, average background power of the DRAM

can be reduced by $3\times(1-\gamma)\times(\frac{D}{3}-1)\times(P_s-P_d)$. Due to this activation schedule, asymptotically the energy consumed by the external memory is reduced by a factor of D, to O($\frac{n^5}{m\times D}$).

The DRAM chips storing the group of block rows that are currently being accessed can also be switched to power-down mode. In a cycle, On-chip MM core may need to access at most one element from each of the matrices A, B, and C. If $F >= 3\alpha f$, then input data from the DRAM chip can be prefetched to FIFO's in FPGA and DRAM chip can be operated in power-down mode to save the background power. α depends on the overheads such as refreshing of DRAM chips, latencies involved when switching from one power mode to another. Fig. 4 shows this activation schedule for a DRAM chip. Similarly when transferring output data from the On-chip MM core to the DRAM, FIFO can be used to operate the DRAM chip storing the matrix C in power-down mode. For an on-chip FIFO of size FI_I for input matrix A (B), after FI_I amount of data has been transferred from the DRAM chip storing the matrix A (B) to the on-chip FIFO, the DRAM chip can be switched to power-down mode for $FI_I\times(\frac{1}{f}-\frac{1}{F})$ units of time. Similarly, for matrix C with FIFO size FI_O, after FI_O amount of data has been transferred from the On-chip MM core to the DRAM chip storing the matrix C, the DRAM chip can be switched to power-down mode for $FI_O\times(\frac{1}{f}-\frac{1}{F})$ units of time.

VI. Experimental results

A. Implementation Details

The On-chip MM core was implemented in Verilog, using Xilinx ISE 14.4, with Virtex-7 XC7VX690T and -3 speed grade as the target. For floating point MM implementations, Xilinx floating point addition and multiplication cores [17] were used. We used non-blocking interface, maximum latency and maximum DSP usage as configuration options when generating these cores. For fixed-point MM implementation, Xilinx multiply adder core [21] was used. When generating the core we choose configuration options to maximize frequency of the core. Floating point cores are IEEE-754 compliant with some minor deviations. To reduce power consumption, we implemented the FIFOs using BRAMs. As we are interested in the power consumed by the architecture, we considered only the dynamic power in our experiments. For on-chip MM, memory controller, and FIFOs, after the post place and route we measured the power using Xpower analyzer [20]. We used default toggle rates in Xpower analyzer in all our experiments. We measured the power by operating On-chip MM at 350MHz frequency for 16-bit fixed point MM, 300MHz for single precision floating point MM, and 250MHz for double precision floating point MM.are set to average toggle rates for most designs.

We considered 4Gb DDR3 SDRAM chips with 8-bits as data access width of the chip. Eight such chips are organized to form a 64-bit data access width rank. We used three ranks of DDR3 SDRAM chips. Each A, B, and C matrices are stored in one rank separately. We used a burst length of 8 to maximize the throughput. The operating frequency of the chip is set to 1066 MHz, supply voltage is set to 1.5V, DDR3 SDRAM speed grade is set to -187 and slow precharge exit mode is considered for maximizing power savings.

A_{Row}	Average time interval between row activations of a DDR3 SDRAM chip storing input matrices in the row major data layout
A_{Block}	Average time interval between row activations of a DDR3 SDRAM chip storing input matrices in the block data layout (described in Section IV-C)
W	Data-width of the matrix element
R_{DRAM}	Percentage of read cycles to the DDR3 SDRAM chip
W_{DRAM}	Percentage of write cycles to the DDR3 SDRAM chip
P_{idle}	Percentage of time DDR3 SDRAM chip is operating in power-down mode
T_{RI}	Time interval between two successive refresh intervals in a distributed type refresh
T_{idle}	Average amount of time in which DDR3 SDRAM chip can operate in power-down mode in a T_{RI} time interval

TABLE III: List of notations

B. External Memory Power Estimation

For estimating the external memory power, we used a realistic DDR3 SDRAM power model [14]. To estimate the power dissipated by a DDR3 SDRAM chip, average activation interval (number of row activations by time), percentage of read cycles from the DDR3 SDRAM chip, percentage of write cycles to the DDR3 SDRAM chip, and percentage of time the DDR3 SDRAM chip is in power-down mode, are required. These values are estimated as follows:

$$Time\ (T) = \frac{(\frac{n}{m})^3\times(m^2+2m)}{f}$$

$$A_{Row} = \frac{(\frac{n}{m})^3\times m}{T}$$

$$A_{Block} = \frac{(\frac{n}{m})^2\times(\left\lceil\frac{(\frac{n}{m})\times(m^2)\times W}{H\times w}\right\rceil+1)}{T}$$

$$R_{DRAM} = \frac{((\frac{n}{m})^3\times m^2\times W)\times 100}{2\times F\times T\times w}$$

$$W_{DRAM} = \frac{(n^2\times W)\times 100}{2\times F\times T\times w}$$

$$P_{idle} = \frac{T_{idle}\times 100}{T_{RI}}$$

the variables used above are described in Table III.

C. Experimental Results

In this subsection, we discuss the energy efficiency results of 1) baseline large-scale MM architecture (described in Section IV-A), 2) large-scale MM architecture with data layout optimization (described in IV-C), 3) large-scale MM architecture with memory activation schedule (described in V) enabled, and 4) large-scale MM architecture with both data layout optimization and memory activation schedule enabled. In block MM, as the block size increases the time required

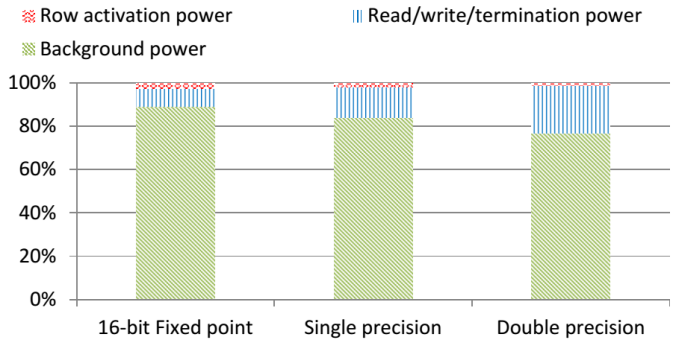

Fig. 5: External memory power profile for the baseline large-scale matrix multiplication

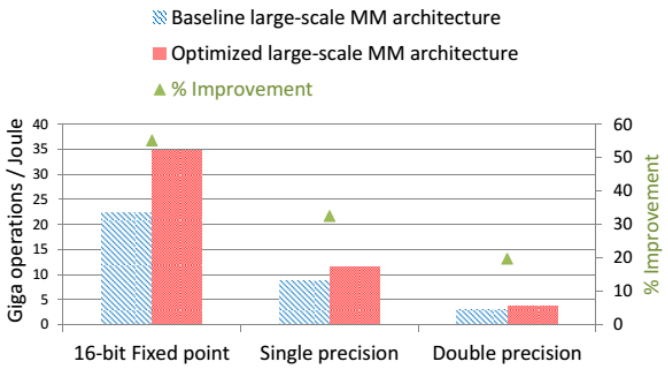

Fig. 6: Energy efficiency of large-scale matrix multiplication

for MM decreases (Section IV-D). Therefore, we consider large On-chip MM designs in powers of 2 and occupying around 50% of the available slices on the target device, so that there are enough slices for other components such as memory controller and FIFO's. For such On-chip MM designs, BRAM-based On-chip MM is more energy efficient than Distributed RAM-based On-chip MM [12], [13]. Therefore, we consider BRAM- based On-chip MM. We consider a block ($m \times m$ sub-matrix) of size 512×512 for 16-bit fixed point MM, a block of size 256×256 for single precision floating point MM, and a block of size 128×128 for double precision floating point MM. We consider large input matrix sizes in powers of 2 that fit in the considered external memory configuration. The size of input matrices for 16-bit fixed point MM is 32K×32K, for single precision floating point MM is 32K×32K and for double precision floating point MM is 16K×16K.

For 512×512 16-bit fixed point (256×256 single precision and 128×128 double precision) On-chip MM, post place and route results show that our implementation achieves 367 Gops/sec, (279 GFlops/sec and 67 GFlops/sec, respectively). It occupies $47,072$ slices ($56,404$ and $67,125$ slices, respectively), uses $2,560$ DSPs ($1,280$ and $1,792$ DSPs, respectively) and $1,024$ 18Kb BRAMs (512 18Kb BRAMs and 256 36Kb BRAMs, respectively).

The estimated dynamic power for 16-bit fixed point (single precision and double precision floating point) baseline large-scale MM when On-chip MM operating at frequency of 350MHz (300MHz and 250MHz) is 15.8W (17.2W and 19.8W). Figure 5 shows the power profile for external memory of baseline large-scale MM architecture. For the row-major data layout, a DRAM row in a chip is accessed for every row in a block. For the data layout proposed in Section IV-C, for most of the activated DRAM rows, all the elements in them are accessed. Therefore, data layout optimization reduces the number of DRAM row activations from 2×10^8 (1×10^9 and 5×10^8) to 2×10^6 (8×10^6 and 4×10^6) for 16-bit fixed point numbers (single precision and double precision floating point numbers, respectively). In matrix multiplication algorithm, as the adjacent elements in the row of a block are accessed, it has high spatial locality when accessing elements with row-major layout. Therefore, the activated row is repeatedly accessed and the fraction of DRAM row activation energy is small. Also, due to the high bandwidth between the DRAM and FPGA, the percentage of read/write cycles from DRAM is less. Dominant portion of the DRAM power is the background power.

Memory activation schedule described in Section V, reduces the background power by operating the DRAM in power-down mode when it is idle. For 32K×32K 16-bit fixed point (32K×32K single and 16K×16K double precision floating point) MM, the energy consumed by the DRAM is reduced by $2.2\times$ ($2\times$ and $1.8\times$, respectively). Also, with memory activation schedule for On-chip MM core, the energy consumed in the auxiliary memory components is reduced. For 16-bit fixed point MM, the energy of the On-chip MM core is reduced by $1.5\times$ ($1.2\times$ and $1.1\times$, respectively). For 16-bit fixed point numbers (single and double precision floating points numbers), using memory activation schedule reduced the total energy by $1.5\times$ ($1.3\times$ and $1.2\times$, respectively). Figure 6 shows the comparison of the energy efficiency results between the baseline and the optimized architectures with both data layout optimization and memory activation schedule. Energy efficiency is improved by $1.6\times$ ($1.3\times$ and $1.2\times$) for 32K×32K 16-bit fixed point MM (32K×32K single precision floating point MM and 16K×16K double precision floating point MM, respectively). As the matrix size increases, the number of DRAM chips required to store the matrix increases and hence the number of DRAM chips that can be operated in power-down mode using the memory activation schedule also increases. Therefore, as the matrix size increases, the energy efficiency improvements of the optimized architecture over the baseline architecture also increases.

VII. Conclusion

In this work, we explored the energy efficiency of large-scale matrix multiplication on FPGAs. Memory energy dominates the total energy of the design. Based on the matrix multiplication algorithm and the DRAM model, we presented an efficient data layout for storing the input matrices. This data layout reduced the energy consumed by the DRAM by minimizing the number of row activations in the DRAM. By exploiting the matrix multiplication algorithm, modular structure of the DRAM, and the high bandwidth between the on-chip and the DRAM, we proposed an activation schedule. This activation schedule reduced the energy consumed by the on-chip memory and the external memory. Our proposed activation schedule improved energy efficiency by $1.6\times$, $1.3\times$, and $1.2\times$ for 32K×32K 16-bit fixed point, 32K×32K single precision floating point and 16K×16K double precision floating point matrix multiplication, respectively. In the future, we plan to apply the activation schedule scheme to other dense linear algebra kernels for improving the energy efficiency.

REFERENCES

[1] A. Amira and F. Bensaali. An FPGA based parameterizable system for matrix product implementation. In Proc. of IEEE Workshop on Signal Processing Systems, pages 75-79, 2002.

[2] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast matrix multiplication. In IEEE TPDS, 13(11), 1105-1123, 2002.

[3] J. Choi, D. W. Walker, and J. J. Dongarra. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. In Concurrency: Practice and Experience 6.7: 543-570, 1994.

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

[5] Y. Dou, J. Zhou, X. Chen, Y. Lei, and J. Xu. FPGA accelerating three QR decomposition algorithms in the unified pipelined framework. In Proc. of IEEE FPL, pages 410-416, 2009.

[6] G. H. Golub, and F. V. L. Charles. Matrix computations. Vol. 3. JHU Press, 2012.

[7] D. Gregg, C. Mc Sweeney, C. McElroy, F. Connor, S. McGettrick, D. Moloney, and D. Geraghty. FPGA based sparse matrix vector multiplication using commodity DRAM memory. In Proc. of IEEE FPL, pp. 786-791, 2007.

[8] J. W. Hong, and H. T. Kung. I/O complexity: The red-blue pebble game. In Proc. of STOC, ACM, 1981.

[9] J. W. Jang, S. B. Choi, and V. K. Prasanna. Energy-and time-efficient matrix multiplication on FPGAs. In IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 13.11: 1305-1319, 2005.

[10] Z. Jovanovi, and V. Milutinovic. FPGA accelerator for floating-point matrix multiplication. In IET Computers & Digital Techniques 6.4: 249-256, 2012.

[11] H. S. Kim, N. Vijaykrishnan, M. Kandemir, E. Brockmeyer, F. Catthoor, and M. J. Irwin. Estimating influence of data layout optimizations on SDRAM energy consumption. In Proc. of ACM ISLPED, pages 40-43, 2003.

[12] M. K. Kiran, L. Hoang, and V. K. Prasanna. Energy efficient architecture for matrix multiplication on FPGAs. In Proc. of IEEE FPL, 2013.

[13] M. K. Kiran, L. Hoang, and V. K. Prasanna. Evaluating energy efficiency of floating point matrix multiplication on FPGAs. In Proc. of IEEE HPEC, 2013.

[14] Micron DDR3 SDRAM System-Power Calculator, http://www.micron.com/products/support/power-calc.

[15] N. Park, W. Liu, V. K. Prasanna, and C. Raghavendra. Efficient matrix multiplication using cache conscious data layouts. In Prof. of HPCMO User Group Conference, 2000.

[16] R. Scrofano, S. Choi, and V. K. Prasanna. Energy efficiency of FPGAs and programmable processors for matrix multiplication. In Proc. of IEEE FPT, pages 422-425, 2002.

[17] Xilinx logicore ip floating−point operator v6.0, http://www.xilinx.com/support/documentation/ip_documentation/floating_point/v6_0/ds816_floating_point.pdf.

[18] 4Gb: x4, x8, x16 DDR3 SDRAM. http://www.micron.com/products/dram/ddr3-sdram

[19] TN-41-01: Calculating memory system power for DDR3, http://www.micron.com/∼/media/Documents/Products/Technical%20Note/DRAM/TN41_01DDR3_Power.pdf.

[20] Xpower analyzer, http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm.

[21] Xilinx logicore ip multiply adder v2.0, http://www.xilinx.com/support/documentation/ip_documentation/xbip_multadd ds717.pdf.

[22] Xilinx power tools tutorial, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ug733.pdf.

[23] Xilinx XPower estimator user guide, http://www.xilinx.com/support/documentation/user_guides/ug440.pdf.

[24] L. Zhuo and V. K. Prasanna. Scalable and modular algorithms for floating-point matrix multiplication on FPGAs. In Proc. of 18th IPDPS, page 92. IEEE, 2004.

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