

Energy- and Time-Efficient Matrix Multiplication on FPGAs

Ju-Wook Jang, *Member, IEEE*, Seonil B. Choi, *Member, IEEE*, and Viktor K. Prasanna, *Fellow, IEEE*

Abstract—We develop new algorithms and architectures for matrix multiplication on configurable devices. These have reduced energy dissipation and latency compared with the state-of-the-art field-programmable gate array (FPGA)-based designs. By profiling well-known designs, we identify “energy hot spots,” which are responsible for most of the energy dissipation. Based on this, we develop algorithms and architectures that offer tradeoffs among the number of I/O ports, the number of registers, and the number of PEs. To avoid time-consuming low-level simulations for energy profiling and performance prediction of many alternate designs, we derive functions to represent the impact of algorithm design choices on the system-wide energy dissipation, area, and latency. These functions are used to either optimize the energy performance or provide tradeoffs for a family of candidate algorithms and architectures. For selected designs, we perform extensive low-level simulations using state-of-the-art tools and target FPGA devices. We show a design space for matrix multiplication on FPGAs that results in tradeoffs among energy, area, and latency. For example, our designs improve the energy performance of state-of-the-art FPGA-based designs by 29%–51% without any increase in the area–latency product. The latency of our designs is reduced one-third to one-fifteenth while area is increased 1.9–9.4 times. In terms of comprehensive metrics such as Energy-Area-Time, our designs exhibit superior performance compared with the state-of-the-art by 50%–79%.

Index Terms—Algorithm design, configurable hardware, energy-delay tradeoff, field-programmable gate array (FPGA), linear array, matrix multiplication, performance estimation.

I. INTRODUCTION

DRAMATIC increases in the density and speed of field-programmable gate arrays (FPGAs) make them attractive as flexible and high-speed alternatives to DSPs and ASICs [7], [12], [16], [25]. Indeed, FPGAs have become an attractive fabric for the implementation of computationally intensive applications such as signal, image, and network processing tasks used in mobile devices [11], [18], [21]. Matrix multiplication is a frequently used kernel operation in a wide variety of graphics, image processing, robotics, and signal processing applications. Several signal and image processing operations can be reduced

to matrix multiplication. Most of the previous work on matrix multiplication on FPGAs focuses on latency optimization [1], [15], [17], [22]. However, since mobile devices typically operate under various computational requirements and energy constrained environments, energy is a key performance metric in addition to latency and throughput [3]. Hence, in this paper, we develop designs that minimize the energy dissipation. Our designs offer tradeoffs between energy, area, and latency for performing matrix multiplication on commercially available FPGA devices. Recent efforts by FPGA vendors have resulted in rapid increases in the density of FPGA devices. Hence, we also develop a design that attempts to further minimize the energy dissipation and latency in exchange for an increase in area, to take advantage of further increases in FPGA density.

Our effort is focused on algorithmic techniques to improve energy performance, instead of low-level (gate-level) optimizations. We evaluate various alternative designs at the algorithmic level (with accompanying architectural modifications) on their energy performance. For this purpose, we construct an appropriate energy model based on the methodology proposed in [8] to represent the impact of changes in the algorithm on the system-wide energy dissipation, area, and latency. The modeling starts by identifying parameters whose values change depending on the algorithm and have significant impact on the system-wide energy dissipation. These parameters depend on the algorithm and the architecture used and the target FPGA device features. We derive closed-form functions representing the system-wide energy dissipation, area, and latency in terms of the key parameters.

The energy, area, and latency functions provide us with a high level view on where to look for possible savings in system-wide energy, area, and latency. These functions allow us to make tradeoffs in the early design phase to meet the constraints. Using the energy function, algorithmic- and architectural-level optimizations are made. To illustrate the performance gains, extensive low-level simulations using Xilinx ISE 4.1i and ModelSim 5.5 e, and Virtex-II as an example target FPGA device, are then performed. Xilinx XPower is used on the simulation data to verify the accuracy of the energy and area estimated by the functions. Our optimized algorithm and architecture (Corollary 1 in Section III, for example) save 51% of the system-wide energy dissipation for matrices of sizes 15×15 , when compared with the design from the state-of-the-art Xilinx library [26]. Latency is reduced by a factor of 15 while area is increased by a factor of 9.4.

To pursue the possibility of further reduction in system-wide energy dissipation and latency in exchange for an increase in area, we also develop an algorithm and architecture (Theorem 1

Manuscript received November 10, 2003; revised July 26, 2004. This work was supported by the National Science Foundation under Award CCR-0311823. The work of J.-W. Jang was supported by the Ministry of Information and Communication of Korea under the Human Resource Development for IT SoC Key Architect.

J.-W. Jang is with the Department of Electronic Engineering Sogang University, Seoul, Korea (e-mail: jjang@sogang.ac.kr).

S. B. Choi is with Intel Corporation, Chandler, AZ 85248 USA (e-mail: seonil@halcyon.usc.edu).

V. K. Prasanna is with the Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, CA 90089-2562 USA (e-mail: prasanna@ganges.usc.edu).

Digital Object Identifier 10.1109/TVLSI.2005.859562

in Section III) with an increased number of MACs. Low-level simulations show further reduction in the system-wide energy dissipation and latency. For example, for matrices of size 12×12 , the system-wide energy dissipation is reduced by an additional 40%, resulting in 69% reduction when compared with the design from the Xilinx library [26]. The latency and area reduce and increase by factors of 23 and 11.8, respectively.

The remainder of the paper is organized as follows. Section II summarizes the related work in the literature. Algorithms and architectures for energy-efficient implementation are presented in Section III. An energy model specific to our implementation is described in Section IV. It includes extracting key parameters from our algorithm and architecture to build a domain-specific energy model and deriving functions to represent system-wide energy dissipation, area, and latency. Section IV-B shows the optimization procedure for our algorithms and architectures in an illustrative way. Analysis of the tradeoffs between system-wide energy, area, and latency is also provided. Section V provides implementation details and explains the simulation method along with its statistical representativeness. Section VI analyzes the performance of our algorithms and architectures through various known metrics in addition to the system-wide energy dissipation. Section VII concludes the paper.

II. RELATED WORK

To the best of our knowledge, there has been no previous work targeted at energy-efficient implementation of matrix multiplication on FPGAs.

Mencer *et al.* [17] implemented matrix multiplication on the Xilinx XC4000E FPGA device. Their design employs bit-serial MACs using Booth encoding. They focused on tradeoffs between area and maximum running frequency with parameterized circuit generators. For the specific example of 4×4 matrix multiplication, 954 CLBs are used to achieve a maximum running frequency of 33 MHz.

Amira *et al.* [1] improved the design in [17] using the Xilinx XCV1000E FPGA device. Their design uses modified Booth-encoder multiplication along with Wallace tree addition. The emphasis was once again on maximizing the running frequency. For the specific example of 4×4 matrix multiplication, 296 CLBs are used to achieve a maximum running frequency of 60 MHz. Area/speed or, equivalently, the number of CLBs divided by the maximum running frequency was used as a performance metric.

Even though our designs mainly target the tradeoffs among energy dissipation, area, and latency along with algorithmic level energy optimization, they also improve the designs in [17] and [1] in terms of the area/speed metric. The area/speed metrics for the designs in [17] and [1], and for our design are 14.45, 4.93, and 2.35, respectively. For fair comparison, translation of the number of CLBs for different FPGA devices is performed on the basis of the equivalent amount of logic. For example, 140 CLBs of the Xilinx XC2V1500 used in our design of 4×4 matrix multiplication to achieve a running frequency of 166 MHz can be translated into 280 CLBs of the Xilinx XCV1000E FPGA device used in [1].

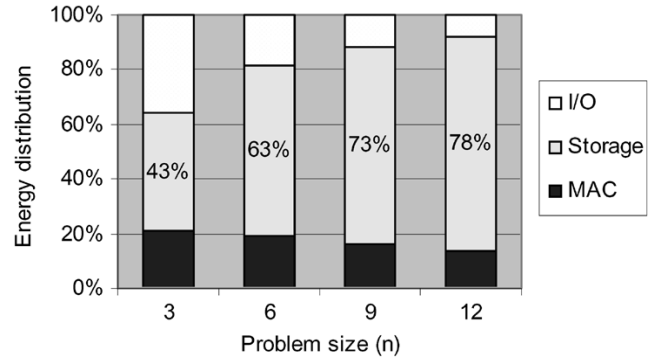


Fig. 1. Energy distribution of the design proposed in [22].

Kumar and Tsai [22] achieved the theoretical lower bound for latency for matrix multiplication with a linear systolic design. They provide tradeoffs between the number of registers and the latency. Their work focused on reducing the leading coefficient for the time complexity. Our work focuses on minimizing energy dissipation under constraints for area and latency. We significantly reduce the number of registers involved in the movement of intermediate results and elements of input matrices. $n^2 + (6n^2/s)$, $1 \leq s \leq n$, registers of 8-b words are involved in the data movement for $n \times n$ matrix multiplication in [22]. In our design, only $2n$ registers of 8-b words are involved in the systolic data movement (based on Theorem 1). Extra $2n$ registers of 8-b words are required to store copies and are not involved in the systolic data movement. Their work is not implemented on FPGAs.

The most appropriate reference design with which the performance of our designs should be compared comes from Xilinx [26]. The state-of-the-art design from Xilinx library performs matrix multiplications for limited sizes (3×3). Xilinx XPower [26] can be used to measure the power dissipation of designs implemented on Xilinx FPGA devices. For a fair comparison, we use the same design environment, the same target device, and the same power measurement tool. Details of the simulations can be found in Section VI. Xilinx just provides a point design optimized at the gate level. Our work constructs a design space spanned by possible design choices in our algorithm.

III. ENERGY-EFFICIENT ALGORITHMS/ARCHITECTURES FOR MATRIX MULTIPLICATION

For performance comparison purposes, we have implemented the latency-optimal systolic design [22] on FPGA devices. The energy distribution profile of the design reveals that much of the total energy is dissipated in the registers (see Fig. 1). For example, 78% of the energy is used in the registers for 12×12 matrix multiplication.

By identifying the energy hot spot, we propose new energy-efficient algorithms and architectures for matrix multiplication. We present our algorithms and architectures in two theorems and two corollaries. Pseudocode for cycle-specific data movement, the detailed architectures, and a snapshot of an example computation are also shown. Theorem 1 improves the latency-optimal algorithm for matrix multiplication [22] in terms of the number of registers used in the designs. Our design has optimal

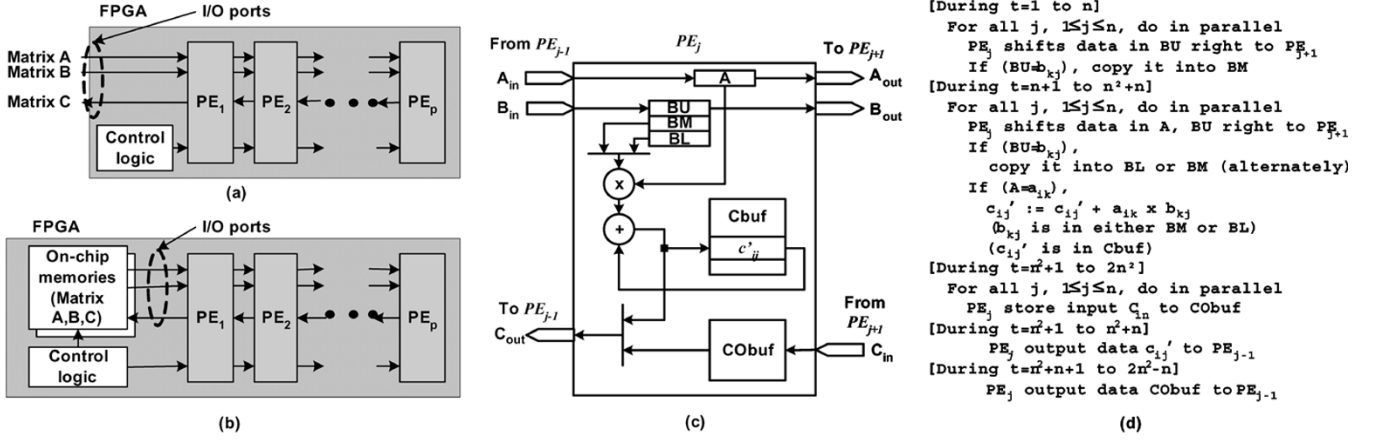


Fig. 2. (a) Off-chip design and (b) on-chip design. (c) Architecture of PE_j used in Theorem 1. (d) Algorithm used in Theorem 1.

time complexity with a leading coefficient of 1 for matrix multiplication on a linear array. Theorem is extended to Corollary 1 for tradeoffs among energy dissipation, area, and latency. Corollary 1 is used to identify energy-efficient designs under latency and area constraints.

The second algorithm is developed to exploit further increases in the density of FPGA devices to realize improvements in energy dissipation and latency (Theorem 2). It uses more MACs and I/O ports. Corollary 1 and Theorem 2 are integrated into Corollary 2. Corollary 2 provides more comprehensive tradeoffs among energy dissipation, area and latency than Corollary 1.

Based on the location of the input and output matrices, we have two design scenarios: off-chip design and on-chip design [see Fig. 2(a) and (b)]. In the off-chip design, we assume that the input matrices are stored outside the FPGA. I/O ports are used for data access. While we assume that the input matrices are stored in an external memory outside the FPGAs, we do not include the energy used by the external memory. However, in the on-chip design, we store all input and output matrices in an on-chip memory of the FPGA devices. The on-chip memory refers to an embedded memory in FPGAs. For example, a Block SelectRAM in the Xilinx Virtex-II devices can be used for the on-chip memory. Thus, the energy used by the on-chip memory is included in the on-chip design.

Theorem 1: $n \times n$ matrix multiplication can be performed in $n^2 + 2n$ cycles using 3 I/O ports and n processing elements (PEs), each having a MAC (MAC-and-accumulator), 4 registers, and 2 local memories of n words (Fig. 2(a) and (b) shows a linear array connecting the PEs and Fig. 2(c) shows a PE).

Proof: The algorithm in Fig. 2(d) and the architecture in Fig. 2(a)–(c) are devised to compute $c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj}$ for all i, j . a_{ik} , b_{kj} , and c_{ij} represent elements of the $n \times n$ matrices A, B , and C . PE_j denotes the j th PE from the left in Fig. 2(a), $j = 1, 2, \dots, n$. PE_j computes column j of matrix C , $c_{1j}, c_{2j}, \dots, c_{nj}$, which is stored in the local memory Cbuf. In Phase k , column k of matrix A ($a_{ik}, 1 \leq i \leq n$) and row k of matrix B ($b_{kj}, 1 \leq j \leq n$) traverse $PE_1, PE_2, PE_3, \dots, PE_n$ in order and allow PE_j to update $c'_{ij} = c'_{ij} + a_{ik} \times b_{kj}$, where c'_{ij} represents the intermediate value of c_{ij} . Once b_{kj} arrives at PE_j , a copy of b_{kj} resides in PE_j until $a_{1k}, a_{2k}, a_{3k}, \dots, a_{nk}$

pass through PE_j . We observe that the following two essential requirements should be satisfied: 1) since a_{ik} stays at each PE_j for just one cycle, b_{kj} should arrive at PE_j no later than a_{ik} , for any $i, 1 \leq i \leq n$ and 2) once b_{kj} arrives at PE_j , a copy of b_{kj} should reside in PE_j until a_{nk} arrives. We show how these two essential requirements for our systolic implementation are satisfied with a minimal number of registers. In addition, we evaluate the number of cycles required to finish the operation and the amount of local memory per PE. An illustrative snapshot for $n = 3$ is provided for more clarity.

- 1) b_{kj} should arrive at PE_j no later than a_{ik} , for any $C, 1 \leq i \leq n$: matrix B is fed to the lower I/O port of PE_1 [see Fig. 2(c)] in row major order ($b_{11}, b_{12}, b_{13}, \dots, b_{1n}, b_{21}, b_{22}, \dots$). Matrix A is fed to the upper I/O port of PE_1 in column major order ($a_{11}, a_{21}, a_{31}, \dots, a_{n1}, a_{12}, a_{22}, \dots$), n cycles behind matrix B . For example, a_{11} is fed to the upper I/O port of PE_1 in the same cycle as b_{21} is fed to the lower I/O port of PE_1 . The number of cycles required for b_{kj} to arrive at PE_j is $(k-1)n + 2j - 1$. a_{ik} requires $n + (k-1)n + i + j - 1$ cycles to arrive at PE_j . The requirement is satisfied since $(k-1)n + 2j - 1 \leq n + (k-1)n + i + j - 1$ for all i and j . For example, we show how b_{2n} (the last element of matrix B in phase 2) arrives at PE_n no later than a_{12} (the first element of matrix A in phase 2) for $c'_{1n} = c'_{1n} + a_{12} \times b_{2n}$. b_{2n} needs $3n - 1$ cycles. a_{12} needs $3n$ cycles.
- 2) Once b_{kj} arrives at PE_j , a copy of b_{kj} should reside in PE_j until a_{nk} arrives: we show how to minimize the number of registers to store copies of b_{kj} ($k = 1, 2, \dots, n$) in PE_j , for each j . We prove that two registers [denoted BM and BL in Fig. 2(c)] are sufficient to hold b_{kj} at PE_j (to store two consecutive elements, $b_{(k+1)j}$ and b_{kj}). For example, when b_{34} arrives at PE_4 , b_{14} is in BL and b_{24} is in BM. If we can prove that a_{n1} has arrived at PE_4 , b_{34} can replace b_{14} in BL. Note that b_{14} is no longer needed in PE_4 after $c'_{n4} = c'_{n4} + a_{n1} \times b_{14}$ is performed using a_{n1} . In general, b_{kj} is needed until a_{nk} arrives at PE_j in the $\{n + (k-1)n + n + j - 1\}$ -th cycle. $b_{(k+2)j}$ arrives at PE_j in the $\{(k+1)n + 2j - 1\}$ -th cycle. Since $(k+1)n + 2j - 1 > n + (k-1)n + n + j - 1$ for all

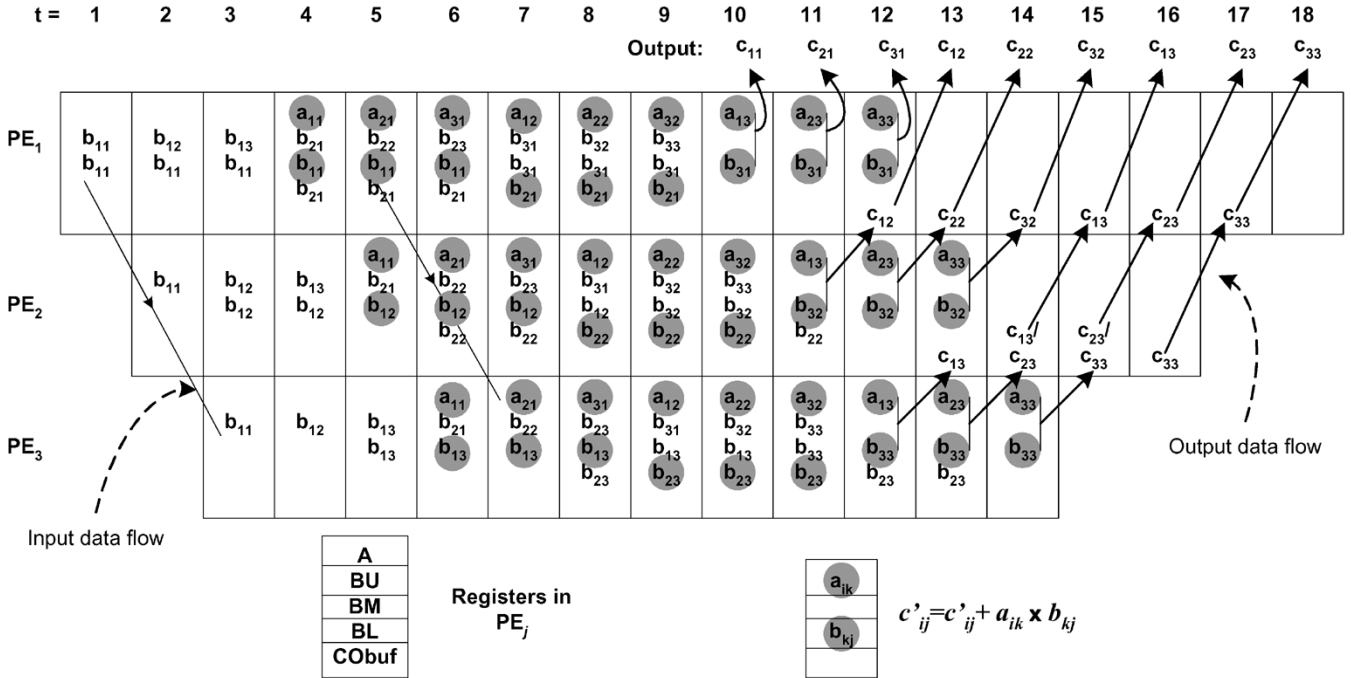


Fig. 3. Snapshot of the data flow for 3×3 matrix multiplication (Theorem 1).

j, k , and n , b_{kj} can be replaced when $b_{(k+2)j}$ arrives at PE_j . Also, the time difference between $b_{(k+2)j}$ and b_{kj} is $\{(k+1)n + 2j - 1\} - \{n + (k-1)n + n + j - 1\} = j$, which has a minimum value of 1. Hence, in the worst case, $b_{(k+2)j}$ arrives one cycle after b_{kj} is no longer required, which means that $b_{(k+2)j}$ can replace b_{kj} . This also shows that $b_{(k+2)j}$ cannot arrive while $b_{(k+1)j}$ is used since $b_{(k+2)j}$ barely arrives after b_{kj} is no longer required. This proves that PE_j needs at least two temporary registers, BM and BL, to hold $b_{kj}(k = 1, 2, \dots, n)$.

- 3) $n^2 + 2n$ cycles are needed to complete the matrix multiplication. The computation finishes one cycle after a_{nn} arrives at PE_n , which is the $\{n + (n-1)n + n + n - 1\}$ -th or $\{n^2 + 2n - 1\}$ th cycle.
- 4) On completion of the above matrix multiplication, column j of the resulting output matrix C is in Cbuf of PE_j , for $1 \leq j \leq n$. To move the matrix C out of the array, we use an extra local memory CObuf and two ports Cout and Cin in each PE. Once column $j + 1$ of the resulting output matrix C is available in the Cbuf of PE_{j+1} , it is moved to CObuf of PE_j for $1 \leq j \leq n - 1$. Column 1 is moved out of the array via Cout of PE_1 . Columns 2 to n are moved out of the array in a pipelined fashion through CObufs of PE_{n-1} to PE_1 . Using CObuf, the moving of the output matrix C for the current matrix multiplication out of the array can be overlapped with the next matrix multiplication involving Cbuf. Without the overlapping, it takes extra $n^2 - 2n$ cycles to move the resulting output matrix C out of the array. Note that the first element of the output matrix, c_{11} is available after $n^2 + 1$ -th cycle and it takes $n^2 - 1$ cycles to move the rest of the output matrix out of the array. ■

A snapshot of the execution of the algorithm is provided for $n = 3$ in Fig. 3. It shows the contents of the registers, A, BU, BM,

and BL of each PE during each cycle of the matrix multiplication process. For example, a_{31}, b_{23}, b_{11} , and b_{21} stay in PE_1 during cycle 6. a_{31} and b_{11} (in the dark circles) are used to update $c_{31} = c_{31} + a_{31} \times b_{11}$. Note that b_{11} is no longer needed after this update and hence can be replaced by b_{31} , which arrives in cycle 7. Elements of matrix A stay in register A for one clock cycle and pass through while elements of matrix B are prefetched into registers, BU, BM, and BL of each PE in the linear array and stay until they are no longer needed.

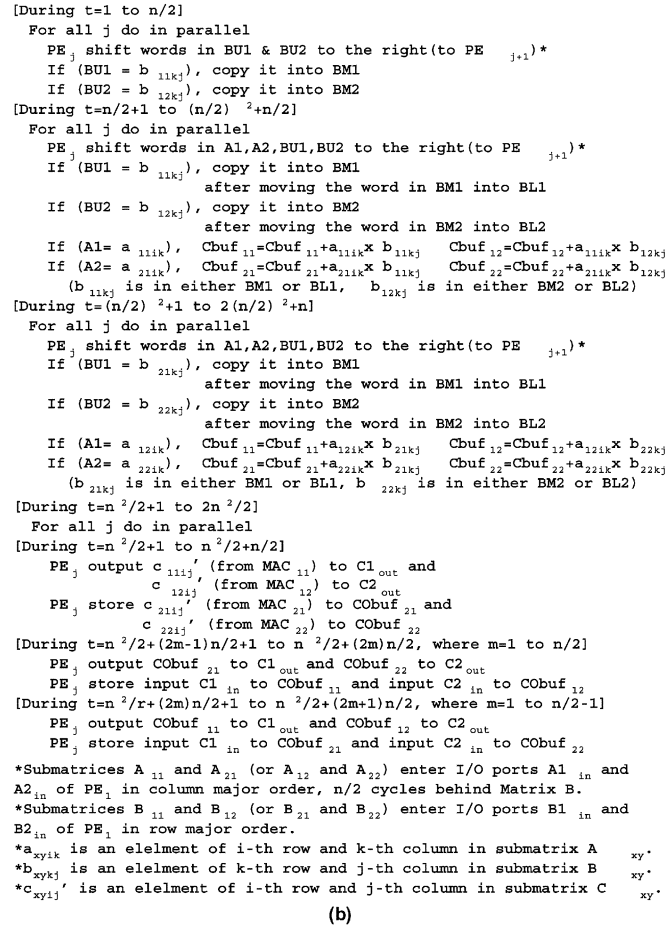
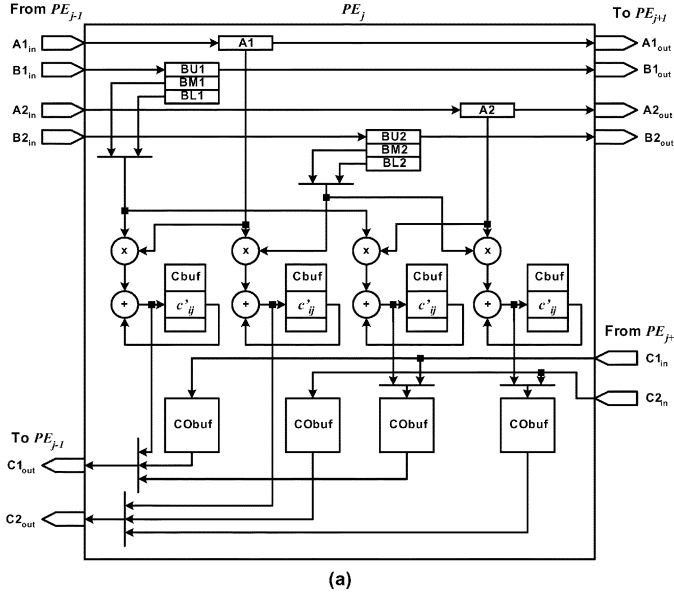
Corollary 1: $n \times n$ matrix multiplication can be performed in $(rn^2 + 2r^2n)$ cycles using three I/O ports and (n/r) PEs, each having one MAC, two local memories of (n/r) words, and four registers, where n is divisible by r .

Proof: $n \times n$ matrix multiplication can be decomposed into $r^3(n/r) \times (n/r)$ matrix multiplications, assuming that n is divisible by r . Using Theorem 1 with n replaced by (n/r) , the proof follows. ■

Corollary 1 provides tradeoffs between area and latency. Larger values for r reduces the number of PEs, which results in less area. However, it increases the number of cycles to complete the matrix multiplication. Combined with power and area estimation of modules, Corollary 1 provides tradeoffs among energy dissipation, area, and latency.

Theorem 2: $n \times n$ matrix multiplication can be performed in $((n^2/r) + (2n/r))$ cycles using $3r$ I/O ports and (n/r) PEs, each having r^2 MACs, $2r^2$ local memories of (n/r) words, and $4r$ registers [Fig. 4(a) shows a PE for $r = 2$], where n is divisible by r .

A detailed proof is provided in the Appendix. Here, we show only the basic idea. Corollary 1 decomposes a $n \times n$ matrix multiplication into $r^3(n/r) \times (n/r)$ matrix multiplications and performs them in a serial manner to reduce the area with increased latency. Theorem 2 performs the $r^3(n/r) \times (n/r)$ matrix multiplications in a parallel manner to reduce the latency with in-


 Fig. 4. (a) Architecture of PE_j . (b) Algorithm for Theorem 2.

creased area. A direct application of Theorem 1 to reduce the latency by the factor of r^2 would increase the resource r^2 times. This means $3r^2$ I/O ports and (n/r) PEs with r^2 MACs, $4r^2$ registers, and $2r^2$ local memories of (n/r) words per PE. We devise a new data movement scheme to reduce the number of registers and I/O ports by the factor of r , saving greatly the area and the energy consumption. Once a submatrix of size $(n/r) \times (n/r)$

from A or B is loaded, it is shared by the r MACs instead if a MAC during its stay in each PE without any increase in latency. The registers consume a considerable amount of energy due to frequent switching of intermediate results. Besides, we overlap the input of two subsequent submatrices to reduce the latency from $(n^2/r) + 2n$ to $(n^2/r) + (2n/r)$ cycles. Fig. 4(a) shows our architecture for $r = 2$ and Fig. 4(b) shows the accompanying algorithm. Refer to the Appendix for more details.

Corollary 1 and Theorem 2 combined together widen the design space where the tradeoff among area, latency, and energy dissipation is possible. It must be noted that the number of MACs is a key parameter to determine the whole architecture. Based on the number of MACs, Theorem 2 and Corollary 1 can be combined into Corollary 2.

Corollary 2: $n \times n$ matrix multiplication can be performed in $\max((n^3/m^3), (n/m)) \times \min(n^2 + 2n, m^2 + 2m)$ cycles using m MACs, $2m$ local memories of $(\min(m^2, n^2)/m)$ words, $4 \min(n, m)$ registers, and $3 \max((m/n), 1)$ I/O ports, where n^2 is divisible by m and $1 \leq m \leq n^2$.

Proof: For $1 \leq m \leq n$, the proof follows from Corollary 1 by setting $r = (n/m)$. For $n \leq m \leq n^2$, the proof follows from Theorem 2 by setting $r = (m/n)$. ■

Smaller values for m reduce the number of modules such as MACs, registers, and I/O ports used in the design, resulting in a lesser area but the latency is seen to increase. Combined with the latency of a design and the area and the power dissipation of the modules, Corollary 2 provides tradeoffs among energy dissipation, area, and latency for $1 \leq m \leq n^2$. Corollary 2 provides a more comprehensive set of tradeoffs than Corollary 1 or Theorem 2 since the number of MACs used varies within a wide range for a given problem size n . Note that Corollary 1 and Theorem 2 provide tradeoffs among energy dissipation, area, and latency for $1 \leq m \leq n$ and $n \leq m \leq n^2$, respectively, and hence can be viewed as subsets of Corollary 2. A more detailed analysis of all designs with respect to energy, area, and latency is presented in Section IV-C.

IV. PERFORMANCE MODELING AND OPTIMIZATION

Given the goal of algorithmic-level optimization of energy performance for matrix multiplication on FPGA devices, we need an energy model to represent the impact of individual algorithmic-level choices on the energy performance. Based on this model, we make the design tradeoffs to obtain energy-efficient designs. The candidate designs are implemented in Section V-A.

A. Domain-Specific Energy Model

Our approach for the performance modeling is to use a domain-specific energy model [8]. The model is applicable only to the design domain spanned by the family of algorithms and architectures being evaluated. The family represents a set of algorithm-architecture pairs that exhibit a common structure and similar data movement. The domain is a set of point designs resulting from unique combinations of algorithm- and architecture-level changes. The domain-specific energy model abstracts the energy dissipation to suit the design domain. The abstraction is independent of the commonly used levels such as gate,

register, or system level. Rather, it is based on the knowledge of the family of algorithms and architectures. The parameters are extracted considering their expected impact on the total energy performance. For example, if the number of MACs and the number of registers change values in a domain and are expected to be frequently accessed, a domain-specific energy model is built using them as key parameters. The parameters may include elements at the gate, register, or system level as needed by the domain. It is a knowledge-based model that exploits the knowledge of the designer about the algorithm and the architecture.

We also use the knowledge to derive functions that represent energy dissipation, area, and latency. Beyond the simple complexity analysis, we make the functions as accurate as possible by incorporating implementation and target device details. For example, if the number of MACs is a key parameter, we implement a sample MAC on the target FPGA device to estimate its average power dissipation. Random input vectors, as many as are needed for the desired confidence interval [13], are generated for simulation. A power function representing the power dissipation as a function of m , the number of MACs, is generated. This power function is obtained for each module related to the key parameters. Based on the designer's optimization goal and the time available for design, a balance needs to be struck between accuracy and simple representation of the functions. The estimation error of the functions derived in this paper ranges from 3.3% to 7.4%. Since the model is intended for algorithmic-level analysis in the early stage of the design, the error is considered satisfactory.

Our family of architectures and algorithms for matrix multiplication forms a domain and we limit algorithm-level exploration for energy optimization to the design space spanned by this domain. The family of architectures and algorithms in Figs. 2 and 4, and the parameters in Table I represent the design space. We build two domains for Corollary 1 and Theorem 2. Two parameters, n and r , are used. In Corollary 1, n denotes the size of input matrices. r is introduced for block multiplication using submatrices of size (n/r) . In Theorem 2, r determines the number of I/O ports ($3r$), the number of MACs (r^2), and the submatrices of size (n/r) . Due to the nature of our algorithms, the number of each key module depends only on these two parameters.

We identify registers of 8-b and 16-b words, MACs, SRAMs (distributed SelectRAMs in the Xilinx devices), and BSRAMs (Block SelectRAMs in the Xilinx Virtex-II devices) [26] as key modules. Choosing specific values for the parameters in Table I results in a design point in the design space. For example, $n = 24$, $p = 6$, $\text{reg} = 4$, $m = 1$, $\text{sram} = 2$, $K_b = 2$, and $K_{io} = 0$ represents a design where 24×24 matrix multiplication is implemented using six PEs with four registers, one MAC, and two SRAMs per PE. The input and output matrices are stored in two ($\lceil 2 \times 24 \times 24 / 1024 \rceil = 2$) BSRAMs on the device and no I/O ports are used.

An energy model specific to the domain is constructed at the module level by assuming that each module of a given type (register, multiplier, SRAM, BSRAM, or I/O port) dissipates the same power independent of its location on the chip. This model simplifies the derivation of system-wide energy dissipation functions. The energy dissipation for each module can be

TABLE I
RANGE OF PARAMETERS FOR XILINX XC2V1500

Parameter	Range	FPGA constraints
Problem size (n)	2,3,4,...	
No. of PEs (p)	n/l , n is divisible by l , l is integer	
No. of registers/PE (reg)	b^{2k+2} , $b = 2, 3, 4, \dots$ ($0 \leq k \leq \log_b n$)	8/16-bit registers
No. of MACs/PE (m)	b^{2k}	2-stage pipeline, embedded
No. of SRAMs/PE (sram)	$\lceil nb^k / 16 \rceil$	16 words minimum
No. of BSRAMs/PE (K_b) (on-chip design)	$\lceil 2n^2 / 1024 \rceil$	1024 16-bit words minimum
No. of I/O ports (K_{io}) (off-chip design)	$3b^k$	8/16 bits

determined by counting the number of cycles the module stays in each power state and low-level estimation of the power used by the module in the power state, assuming average switching activity. Additional details of the model can be found in [8].

Table II lists the key parameters and the number of each key module in terms of the two parameters for each domain. In addition, it shows the latencies which also depend on the parameters. By choosing specific values for the parameters in Table II, a different design is realized in the design space. For example, a design with $n = 16$ and $r = 4$ represents a design where 16×16 matrix multiplication is implemented using four PEs with four registers, one MAC, and one SRAM per PE.

B. Functions to Estimate Energy, Area, and Latency

Functions that represent the energy dissipation, area, and latency are derived for Corollary 1 and Theorem 2. The energy function of a design is approximated to be $\sum_i t_i P_i$, where t_i and P_i represent the number of active cycles and average power for module i . For example, P_{Mult} denotes the average power dissipation of the multiplier module. The average power is obtained from low-level power simulation of the module. The area function is given by $\sum_i A_i$, where A_i represents the area used by module i . In general, these simplified energy and area functions may not be able to capture all of the implementation details needed for accurate estimation. However, we are concerned with algorithmic-level comparisons, rather than accurate estimation. Moreover, our architectures are simple and have regular interconnections, and so the error between these functions and the actual values based on low-level simulation is expected to be small. In Section V-B, we evaluate the accuracy of the energy and area functions. The latency functions is obtained easily because the theorems and corollaries already give us the latency in clock cycles for the different designs.

Table II shows the number of modules used by the designs for $n \times n$ matrix multiplication with 8-b input precision and 16-b output precision. For the off-chip design, I/O ports are used to fetch elements from outside the FPGA. In the on-chip design, BSRAMs of 1024 16-b words are used for on-chip storage of input matrices. SRAMs are CLB-based memory blocks used for storing intermediate results. The power and area values of

TABLE II
NUMBER OF MODULES USED AND THE LATENCY OF VARIOUS DESIGNS

Domain	Key parameters (range)	No. of PEs	No. of register/PE	No. of MAC/PE	No. of SRAM/PE	No. of BSRAM	No. of I/O ports	Latency (cycles)
Corollary 1	$n, r (r \leq n)$ * n divisible by r	$\frac{n}{r}$	4	1	2	$\frac{2n^2}{1024}$	3	$rn^2 + 2r^2n$
Theorem 2	$n, r (r \leq n)$ * n divisible by r	$\frac{n}{r}$	$4r$	r^2	$2r^2$	$\frac{2n^2}{1024}$	$3r$	$\frac{n^2}{r} + \frac{2n}{r}$
Corollary 2	$n, m (1 \leq m \leq n^2)$ * n^2 divisible by m	$\min\left(m, \frac{n^2}{m}\right)$	$\max\left(4, \frac{4m}{n}\right)$	$\max\left(1, \frac{m^2}{n^2}\right)$	$\max\left(2, \frac{2m^2}{n^2}\right)$	$\frac{2n^2}{1024}$	$\max\left(3, \frac{3m}{n}\right)$	$\max\left(\frac{n^3}{m^2}, \frac{n}{m}\right) \min(n^2 + 2n, m^2 + 2m)$

TABLE III
ENERGY AND TIME PERFORMANCE MODELS

Corollary 1	
Metric	Performance model
Latency (cycles)	$L_{Cor1} = r^3 \left\{ (n/r)^2 + n/r \right\}$
Effective latency (cycles)	$L_{Cor1} = r^3 (n/r)^2$
Energy (on-chip)	$E_{Cor1} = L_{Cor1} \left\{ (n/r)(P_{Mult} + P_{Add} + 2P_{SRAM} + 4P_{RS} + 4P_{R16}) + \left\lceil \frac{2n^2}{1024} \right\rceil P_{BSRAM} + (n/r)P_{offset} \right\}$
Energy (off-chip)	$E_{Cor1} = L_{Cor1} \left\{ (n/r)(P_{Mult} + P_{Add} + 2P_{SRAM} + 4P_{RS} + 4P_{R16}) + 2P_I + P_O + (n/r)P_{offset} \right\}$
Area (on-chip)	$A_{Cor1} = (n/r)(A_{Mult} + A_{Add} + 2A_{SRAM} + 4A_{RS} + 4A_{R16} + A_{offset})$ and $\left\lceil \frac{2n^2}{1024} \right\rceil$ BSRAMs
Area (off-chip)	$A_{Cor1} = (n/r)(A_{Mult} + A_{Add} + 2A_{SRAM} + 4A_{RS} + 4A_{R16} + A_{offset})$ and two 8-bit input ports, one 16-bit output port
Theorem 2	
Metric	Performance model
Latency (cycles)	$L_{Thm2} = n^2 / r + 2n / r$
Effective latency (cycles)	$L_{Thm2} = n^2 / r$
Energy (on-chip)	$E_{Thm2} = L_{Thm2} \left\{ nr(P_{Mult} + P_{Add} + 2P_{SRAM}) + n(4P_{RS} + 4P_{R16}) + \left\lceil \frac{2n^2}{1024} \right\rceil P_{BSRAM} + (n/r)P_{offset} \right\}$
Energy (off-chip)	$E_{Thm2} = L_{Thm2} \left\{ nr(P_{Mult} + P_{Add} + 2P_{SRAM}) + n(4P_{RS} + 4P_{R16}) + 2rP_I + rP_O + (n/r)P_{offset} \right\}$
Area (on-chip)	$A_{Thm2} = nr(A_{Mult} + A_{Add} + 2A_{SRAM}) + n(4A_{RS} + 4A_{R16}) + (n/r)A_{offset}$ and $\left\lceil \frac{2n^2}{1024} \right\rceil$ BSRAMs
Area (off-chip)	$A_{Thm2} = nr(A_{Mult} + A_{Add} + 2A_{SRAM}) + n(4A_{RS} + 4A_{R16}) + (n/r)A_{offset}$ and $2r$ 8-bit input ports, r 16-bit output ports

each module are shown in Table IV. For example, P_{SRAM} is the average power used by SRAM (16-b word), where x is the number of entries. In the actual implementation of a SRAM, the number of its entries should be multiples of 16. P_{offset} denotes the remaining power dissipation of a PE (after the modules have been accounted for), and takes care of glue logic and control logic. Similar numbers representing the area of each module are also obtained. A_{offset} denotes the area of a PE that accounts for glue logic and control logic. The latencies are obtained in terms of seconds by dividing them by the clock frequency. Using Table II, functions that represent energy, area, and latency for Corollary 1 and Theorem 2 are shown in Table III. Functions for other designs can be obtained in the same way. An average switching activity of 50% for input data to each module at a running frequency of 150 MHz is assumed. Multiply operation is performed using dedicated embedded multipliers available in the Virtex-II device.

Note that throughput is important, since many applications for matrix multiplication process a stream of data. Our design in Corollary 1 is a pipelined architecture, with the first (n/r) cycles of the computations on the next set of data being overlapped

TABLE IV
POWER AND AREA FUNCTIONS FOR VARIOUS MODULES

Module	Power Function (mW)	Area Function (slice)
Block multiplier (8x8 bit)	$P_{Mult} = 12.50$	$A_{Mult} = 16^*$
Adder (8 bit)	$P_{Add} = 2.77$	$A_{Add} = 4$
SRAM (16-bit word, x number of entries)	$P_{SRAM} = 0.126 \left\lceil \frac{x}{16} \right\rceil + 2.18$	$A_{SRAM} = 18.44 \left\lceil \frac{x}{16} \right\rceil + 16.40$
BSRAM (16 bit, 1024 entries)	$P_{BSRAM} = 16.37$	$A_{BSRAM} = 16^*$
Register (8 bit)	$P_{RS} = 2.12$	$A_{RS} = 4$
Register (16 bit)	$P_{R16} = 2P_{RS}$	$A_{R16} = 8$
Output port (16 bit)	$P_O = 70$	
Input port (8 bit)	$P_I = 10$	

* Block multiplier or BSRAM uses area equivalent to 16 slices.

with the last (n/r) cycles of the computations on the current set of data. Thus for a stream of matrices, an $(n/r) \times (n/r)$ sub-matrix can be processed every $(n/r)^2$ cycles. Thus, the *effective latency* becomes $(n/r)^2$, which is the time between the arrivals of the first and last output data of the current computation. Hence, the design in Corollary 1 is a throughput-oriented design since one output is available every clock cycle for a stream of

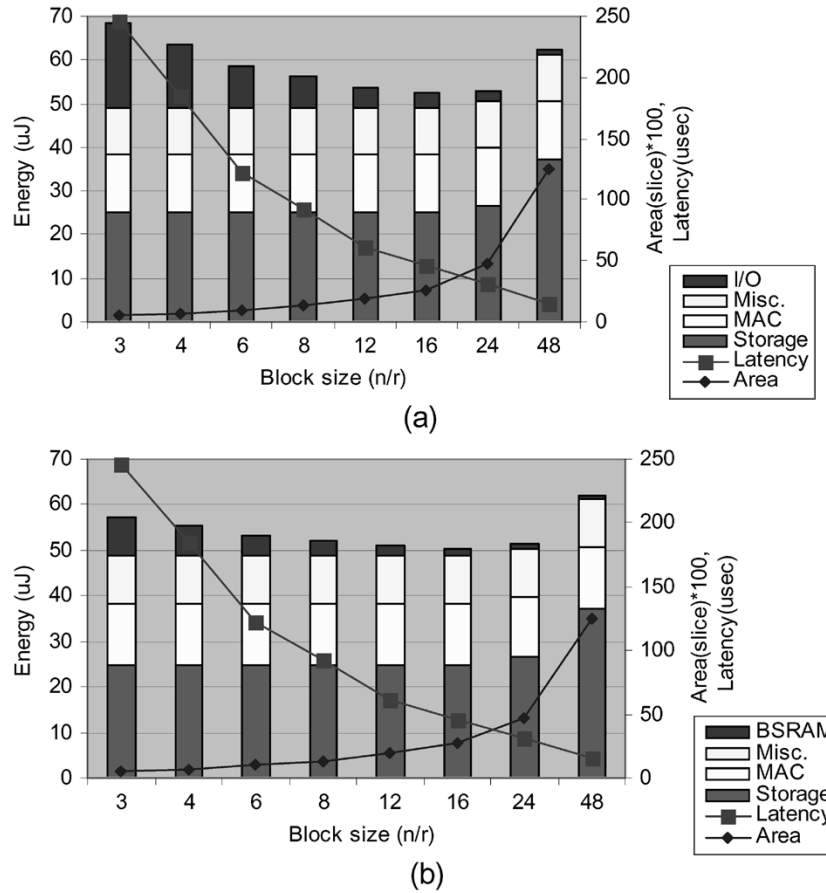


Fig. 5. Energy, area, and latency tradeoffs of Corollary 1 as a function of the block size (n/r). (a) Off-chip and (b) on-chip design for $n = 48$.

matrices. The design in Theorem 2 is also throughput-oriented since r output data items are available every clock cycle. Its effective latency becomes (n^2/r) .

C. Tradeoffs Among Energy, Area, and Latency

The functions in Table III are used to identify tradeoffs among energy, area, and latency. For example, Fig. 5 illustrates the tradeoffs among energy, area, and latency for 48×48 matrix multiplication for the off-chip and on-chip designs of Corollary 1. It can be used to choose energy-efficient designs to meet given area and latency constraints. For example, if 800 slices are available and the latency should be less than 6000 cycles ($36 \mu\text{s}$), an energy-efficient design is obtained using $(n/r) = 4$. The energy dissipation, area, and latency for such a design, evaluated using the functions in Table III, are $6.85 \mu\text{J}$, 524 slices, and 5400 cycles ($32.4 \mu\text{s}$), respectively. Fig. 5 shows that, as the block size (n/r) increases, the area increases and the latency decreases, because the degree of parallelism increases. While the energy dissipation decreases to $(n/r) = 15$ or 16 , it starts increasing afterwards. The reason for this behavior is as follows. The energy used by the local storages, Cbuf and CObuf, is $2n^3(0.126[(1/16)(n/r)] + 2.18)$ and is hence proportional to $O(n^4/r)$. The energy used by the rest of modules, except I/O, are proportional to $O(n^3)$. The energy for I/O is proportional to $O(rn^2)$. As (n/r) increases (r decreases), the energy used by I/O decreases relatively faster, and thus the total

energy decreases. However, after $(n/r) > 16$, the energy used by the local storage becomes the dominant factor. This helps us to identify the optimal block size for energy-efficient matrix multiplication.

Tradeoff analysis for the on-chip model also shows similar behavior. The on-chip design uses BSRAMs instead of I/O ports. Since the energy used in I/O ports is more than the energy used in the BSRAMs, the energy used in the on-chip design is less than the energy used in the off-chip design. However, the choice between the off-chip and on-chip design depends on the situation—whether the matrix multiplication is stand-alone or a part of an application (e.g., an application consists of multiple kernels).

Theorem 2 provides asymptotic improvement in energy and latency performance in the on-chip model. As shown in Table II, asymptotically, the energy dissipated in the BSRAMs and the latency of the Xilinx reference design increase as $O(n^5)$ and $O(n^3)$, respectively, assuming a unit of energy is used per cycle for retaining a word in the BSRAM. Energy dissipation and latency for the designs based on Theorem 1 and [22] increase as $O(n^4)$ and $O(n^2)$, respectively, under the same assumptions. Theorem 2 improves these complexities to $O(n^4/r)$ and $O(n^2/r)$, respectively, where (n/r) is the block size for block multiplication and n is divisible by r with $r \geq 1$. Further increases in the density of FPGAs can be used to increase the number of multipliers and hence nr , leading to asymptotic reduction in energy dissipation and latency.

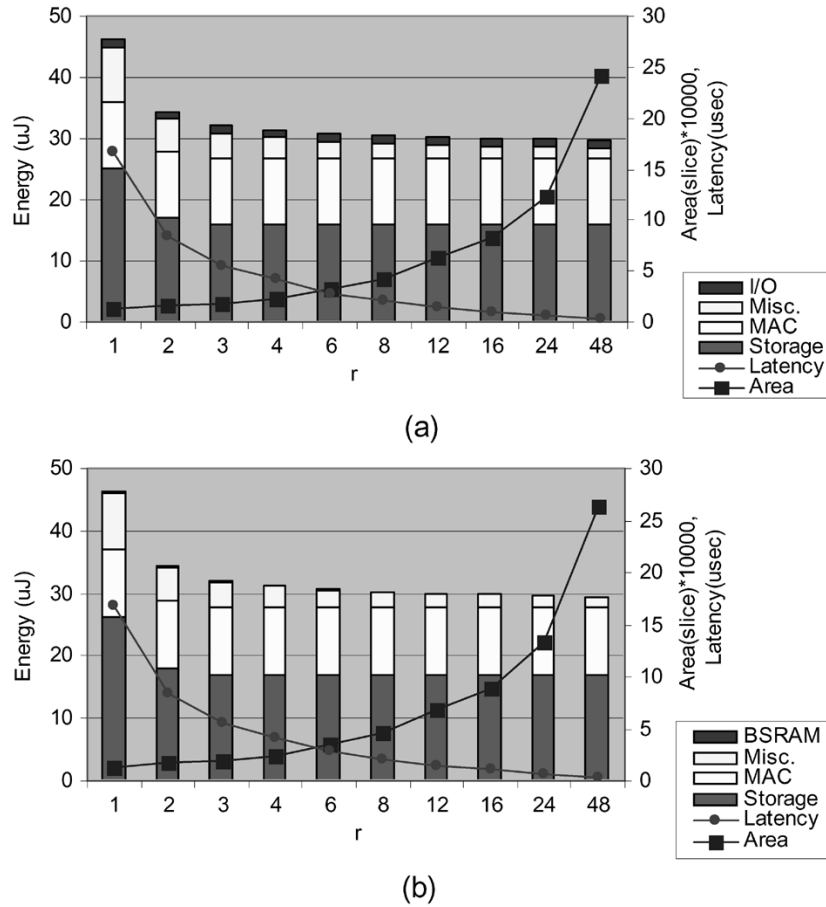


Fig. 6. Energy, area, and latency tradeoffs of Theorem 2 as a function of r . (a) Off-chip and (b) on-chip design for $n = 48$.

Fig. 6 shows the tradeoffs among energy, area, and latency for Theorem 2. As the value of r increases (or block size (n/r) decreases), the area increases and the latency decreases. However, the energy dissipation continuously decreases. Thus, the designs based on Theorem 2 reach the minimal point of energy dissipation when the block size is the smallest unlike the designs based on Corollary 1. Note that the local storages consists of registers, Cbufs, and CObufs. The energy used by the registers in the designs based on Theorem 2 is $O(n^3/r)$ while the energy used by the registers in the designs based on Corollary 1 is $O(n^3)$ for the same problem size. Thus, the energy used by the registers in the designs based on Theorem 2 decreases as r increases while the energy used by the registers in the designs based on Corollary 1 is constant. The same analysis applies to the energy complexity of BSRAMs.

D. Other Optimization Techniques for Energy Efficiency

To optimize the energy performance of our design, we employ several energy-efficient design techniques [9]. One such technique is *architecture selection*. FPGAs give the designer the freedom to map almost any architecture onto hardware. Different architectures have varying energy performances, latencies, and throughputs. In our design, we have chosen a linear array of processing elements. In FPGAs, long interconnects dissipate a significant amount of power [24]. Therefore, for energy-efficient designs, it is beneficial to minimize the number of long interconnects. A linear array of PEs accomplishes this goal.

Each processing element communicates only with its nearest neighbors, minimizing the use of long wires. Additionally, the linear array architecture facilitates the use of two more techniques: *parallel processing* and *pipelining*. Both parallel processing and pipelining decrease the effective latency of a design. Parallel processing does so by increasing the amount of resources while pipelining does so by increasing the resource utilization. By decreasing effective latency, both techniques can lead to lower energy dissipation. However, these techniques can also increase the power dissipation, which can have a negative effect on the energy dissipation. The designer must reach a compromise between low latency and high power in order to achieve a low-energy design. Another technique that we employ is the choosing of the appropriate *bindings*. In an FPGA, there can be many possible mappings of the computation and storage elements to the actual hardware. For example, in the Xilinx Virtex-II, the storage Cbuf can be implemented as registers, a distributed SelectRAM, or a Block SelectRAM. Each of these types of storage dissipates a different amount of energy and can lead to implementations with wide variation in energy dissipation. When the number of entries >64 , a Block SelectRAM is used since it is energy-efficient as a large memory; otherwise, a distributed SelectRAM is used. Similar decisions can be made for other elements of the design, such as choosing multiplication unit. The architecture of the target Virtex-II FPGA offers two options to implement a multiplier: 1) Block Multiplier, which is an ASIC-based embedded multiplier and 2) slices to build a

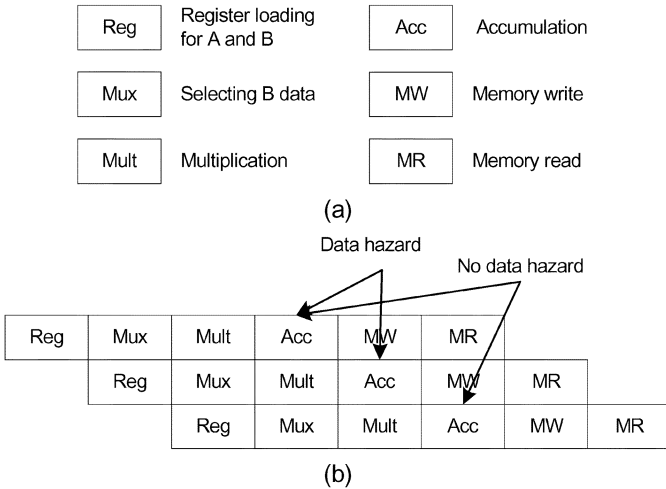


Fig. 7. (a) Pipelining stages. (b) Data hazard.

configured multiplier. By default, the Xilinx ISE XST tool uses the Block Multiplier for multiplication. In our designs, all of the multiplications are performed using the Block Multipliers since they are energy-efficient when any of the inputs is not constant.

V. DESIGN SYNTHESIS AND SIMULATION

Based on the high-level performance estimation, the chosen designs are implemented and simulated to obtain the accurate results. Our target device is Xilinx Virtex-II which is a high-performance platform FPGA from Xilinx [26]. We have chosen the XC2V1500 and XC2V3000 models with a -5 speed grade for comparison. These models have 48 and 96 18×18 -b Block Multipliers, respectively.

A. Implementation Details

Based on the observation in Section IV-C, we implemented the designs using VHDL in the Xilinx ISE 4.1i environment. All parameters are specified using “Generic” variables in VHDL syntax. By changing the values of the Generic variables, different numbers and types of modules are instantiated (see Table II), and eventually the whole architecture is synthesized accordingly. Note that the design for Theorem 1 has one parameter n , and Corollary 1 and Theorem 2 have two parameters n and r . These are only necessary parameters for design synthesis. For the off-chip design, all matrix data are fed via I/O ports. Thus, the total energy includes the energy used for I/O. For the on-chip design, the data are stored in the BSRAMs first and fed into the first PE from the BSRAM. However, the energy used to store the data in BSRAM via I/O ports is not included unlike the case of the off-chip design.

The whole design as well as the design for a PE are pipelined. Fig. 7(a) shows the pipeline stages of each PE and Fig. 7(b) shows the pipelining in each PE. Cbuf is implemented using a dual-port SRAM. Note that there is data feedback which is required for accumulation of intermediate values for matrix C . To accumulate the intermediate values, one data comes from the MAC and the other one comes from Cbuf. If $n > 2$, the memory read for intermediate value occurs after the value is written to Cbuf. However, if $n \leq 2$ for Theorem 1 or $(n/r) \leq 2$ for Theorem 2, data hazard occurs. Since two clock cycles

are required between the memory write and read, the distance between read-after-writes (RAWs) has to be two clock cycles to prevent the data hazard.

Other than the data paths for all designs, control logic is also parameterized based on n and r . The first approach for control logic is that one centralized control logic generator provides most of the control signals for the first PE and then the rest of the PEs receive the delayed control signals from their respective previous PEs. Since all PEs perform the same operations in a systolic manner, the delayed control signals are well suited in our design. Another approach is to have one control logic generator in each PE, which means that distributed control logic generators are used. In both approaches, the control logic generator consists of simple counters and combinational logic. The centralized control logic uses less area but more interconnects compared with the distributed control logic. In our design, the difference in the energy dissipation of two approaches is negligible. Thus, we choose the mixture of distributed and centralized controls in our designs. Each PE of the off-chip design for Theorem 1 and Theorem 2 has 6 control signals from the control logic (i.e., centralized control). The centralized control signals are generated from the control logic (outside PE) and are fed to the first PE. Note that only the first PE gets the control signals and the rest of PEs use them in a pipelining manner [see Fig. 2(a) and (b)]. Then, all signals are passed to the next PE with one or two clock delays. Address generation for Cbuf and CObuf are generated inside PE in a distributed manner. The first control signal CtRegLoad is used to load the input data to BM or BL. It is asserted every (n/r) cycles (see Fig. 3). CtMuxToMult is used to determine which BM or BL is multiplied with A. This signal is identical to CtRegLoad except for the first (n/r) cycles. CtMultCe and CtRamWe are the enable signals for the multiplier and SRAM. Both are asserted when the computation starts. CtFlush is asserted after one set computation for $(n/r) \times (n/r)$ matrix multiplication is completed and a new set of computation starts. Cbuf needs to be flushed before the next result is stored. It is asserted at $(n/r)^2 + (n/r) + 3$ and is on for (n/r) cycles and is off for $(n/r)^2 - (n/r)$ cycles. In the data path, using this signal, we accumulate the intermediate values for matrix C with previous intermediate values or 0 (flushing effect). CtOutMux is used to determine whether the results come from an accumulator or CObuf. It triggers the data pulling from the current CObuf to the previous PE. In fact, the current PE sends the data of CObuf to the previous PE. It is asserted at $(n/r)^2 + 4$ and is on for (n/r) cycles and is off for $(n/r)^2 - (n/r)$ cycles. For the on-chip design, the control logic includes the additional signals such as address generation for on-chip memory. In addition, the energy used by the control logic is separately measured and accounts for about 10% of the designs.

B. Simulation Method

Using the high-level model defined in Section IV-B, several designs can be obtained by optimizing on latency, area, or energy. In this paper, we attempt to arrive at minimal energy designs. The candidate designs are implemented in VHDL. These designs are synthesized using Xilinx Synthesis Technology (XST) in Xilinx ISE 4.1i. The place-and-route file (.ncd file) is

TABLE V
PERFORMANCE COMPARISON OF VARIOUS OFF-CHIP DESIGNS AGAINST THE XILINX DESIGN

Design	Metric	Matrix size					
		3×3	6×6	12×12	15×15	24×24	48×48
Xilinx design	Energy(nJ)	24.4	195.4	1563	3053	12506	100049
	Latency(usec)	0.18	1.44	11.52	22.50	92.16	737.28
	Area(slices)	251	251	251	251	251	251
	EATx1E-12	1.1	70.6	4520	17243	289293	18514777
	Block size (n/r)	3	6	12	15	12	12
Proposed design (Corollary 1)	Energy(nJ)	17.3	110.0	795	1509	6361	50892
	(reduction, %)	29%	44%	49%	51%	49%	49%
	Latency(usec)	0.06	0.24	0.96	1.50	7.68	61.44
	(speedup, times)	3.00	6.00	12.00	15.00	12.00	12.00
	Area(slices)	477	949	1894	2364	1894	1894
	(increase, times)	1.90	3.78	7.55	9.42	7.55	7.55
	EATx1E-12	0.5	25.1	1446	5351	92534	5922165
Design based on [22]	Block size (n/r)	3	6	6	5	8	8
	Energy(nJ)	31.1	158.8	1271	2729	8983	71862
	(reduction, %)	-27%	19%	19%	11%	28%	28%
	Latency(usec)	0.10	0.30	2.36	5.86	13.17	105.40
	(speedup, times)	1.87	4.88	4.88	3.84	7.00	7.00
	Area(slices)	451	1124	1124	877	1698	1698
	(increase, times)	1.80	4.48	4.48	3.49	6.76	6.76
EATx1E-12	1.4	52.7	3372	14016	200950	12860824	

obtained for Virtex-II XC2V1500 and XC2V3000 (speed grade –5). The input test vectors for the simulation are randomly generated such that their average switching activity is 50%. ModelSim 5.6 b, from Mentor Graphics, is used to simulate the designs and generate simulation results (.vcd file). These two files are then provided to the Xilinx XPower tool to evaluate the average power dissipation. Energy dissipation is further obtained by multiplying the average power with the effective latency.

While measuring the power dissipation of the designs on an actual device can provide the most accurate energy dissipation values, XPower is chosen because it is one of the most widely available tools. Also, the values from XPower are known to be reasonably accurate. Using tools such as XPower, high accuracy is achieved if the switching activities of individual logic interconnect, clocks, and I/O are taken into account to obtain their power dissipations. In our estimates, these switching activities are determined based on the actual designs and the switching activity of input vectors during the simulation (that is, no default switching activity is assumed). Based on XPower, our design for matrix multiplication dissipates 29%–51% less energy than that of the Xilinx design (see Table V). The performance improvement is well above the accuracy margin of XPower.

The estimates from Section IV-B are also against the actual values based on synthesized designs to test the accuracy of the performance estimation functions. We observe that the estimation error of our functions ranges from 3.3% to 7.4% for energy dissipation and 3.3% to 4.1% for area. The estimation method (the domain-specific modeling) used in this paper and its accuracy are extensively discussed in [8].

To address the dependency of energy dissipation on the input matrices, matrices are randomly generated and fed to our design and the Xilinx design for 3 × 3 matrix multiplication. The following equation is employed to estimate the confidence interval for our simulation:

$$\bar{x} \pm z_{\alpha/2} \frac{s}{\sqrt{M}} \tag{1}$$

The confidence interval is the interval into which the real average (over all possible input matrices in this example) falls with a certain probability (confidence) [13]. \bar{x} , $z_{\alpha/2}$, s , and M

represent the average energy over (randomly generated) sample input matrices, the standard normal percentile, the standard deviation, and the number of sample matrices, respectively. The probability that the real average energy dissipation belongs to the interval in (1) is $1 - \alpha$ [13].

Fig. 8(a) compares the energy dissipation over 50 randomly generated 3 × 3 input matrices of our design with that of the Xilinx design. The 95% confidence intervals are compared in Fig. 8(b). Based on 95% confidence intervals, the average energy dissipation of our design for 3 × 3 input matrices is 7.81 nJ (32%) less than that of the Xilinx design. All designs in the paper follow the simulation method based on this confidence interval.

VI. DESIGN ANALYSIS AND PERFORMANCE COMPARISON

Using the functions in Table III, energy-efficient designs are identified for various sizes of matrices around (an arbitrary) area constraint of 2000 slices. This is about 25% of the total available area in XC2V1500. The area does not necessarily increase with the size of the input matrices. The identified designs are implemented and simulated, as described in Section V-B. Table V compares the performance of our designs against the Xilinx design and a best-known latency-optimal design [22] on a linear array for various sizes of matrices. All values for energy and area are based on low-level simulation. All designs include control logic, and the on-chip designs includes on-chip memories (BSRAM) and its address generation logic.

Xilinx provides an optimized design for 3 × 3 matrix multiplication only. The three rows of matrix A are stored in three registers, and the entire matrix B is stored in another set of nine registers. A single multiplier and an adder are used to perform multiplication and accumulation to obtain one row of matrix C . This is repeated with the two more rows from matrix A .

The Xilinx design uses only 251 slices while a XC2V1500 device can hold up to 7800 slices in addition to memory banks (BSRAMs) and I/O ports. For $n > 3$, we used block matrix multiplication using the 3 × 3 design. The Xilinx design is run at 150 MHz. The comparisons are based on individual metrics as well as more comprehensive metrics, such as energy × area × latency (EAT) product [2]. The area is the number of slices used in the design. An embedded multiplier or a BSRAM is counted as 16 slices each since they occupy the equivalent area. Fig. 5 shows that our design is able to trade off the area with the energy and the latency. For example, our design for 3 × 3 matrix occupies only 1.9 times of the area used by the Xilinx design (see Table V). The latency is reduced to one third and the energy is reduced by 29%. However, our design is a parallel design and, hence, uses more area than the serial design from Xilinx.

The largest reduction in energy dissipation is 51% and can be obtained by using $(n/r) = 15$ or 16. Our designs improve the performance of the Xilinx reference design by 29%–51% with respect to energy and a factor of 3–15 with respect to latency while increasing the area by a factor of 1.9–9.4. Xilinx design saves power by slowing the clocks of the registers which read/writes data once in a several cycles. Our improvement comes threefold. First, we greatly reduce the latency by parallelizing the design since energy is power multiplied by latency.

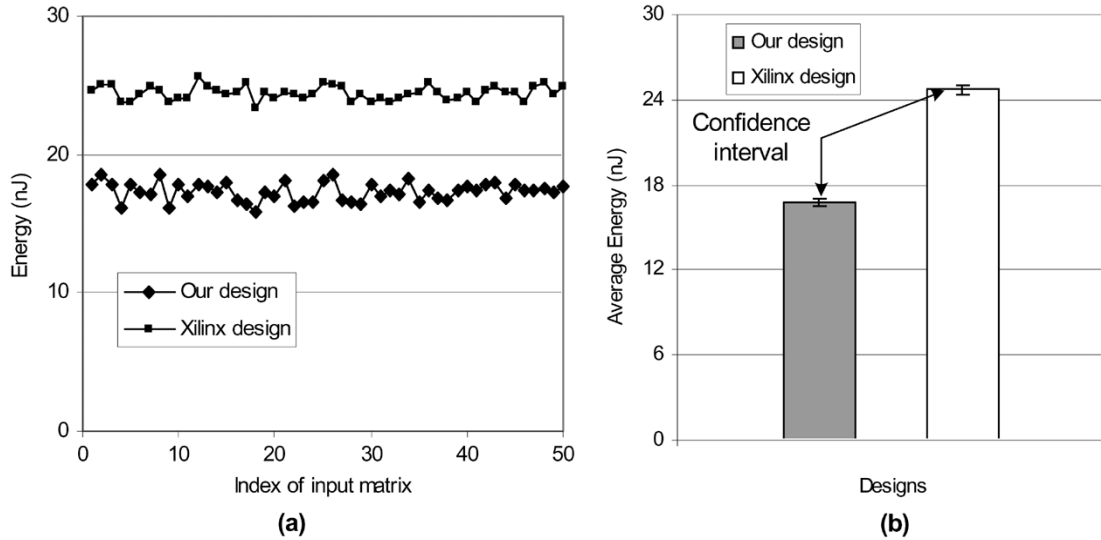


Fig. 8. Comparison between our design (based on Theorem 1) and Xilinx design for 3×3 matrix multiplication. (a) Energy dissipation for randomly generated matrices. (b) Average energy dissipation with confidence intervals.

TABLE VI
PERFORMANCE COMPARISON OF VARIOUS ON-CHIP DESIGNS

Design	Metric	Matrix size					
		3×3	6×6	12×12	15×15	24×24	48×48
Proposed (Corollary 1)	Block size (n/r)	3	6	12	15	12	12
	Energy(nJ)	16.0	105.9	775	1510	6200	49598
	Latency(usec)	0.06	0.24	0.96	1.50	7.68	61.44
	Area(slices)	434	861	1699	2083	1699	1699
	EATx1E-12	0.4	21.9	1264	4717	80896	5177370

Second, we reduce the amount of switching by reducing the number of registers on the systolic path. Third, we use the prediction functions to choose the most promising design in the domain-specific design space.

The designs based on [22] with the same latency as ours reduce the energy dissipation by 27%–31% when compared with the Xilinx design. Analysis of energy and area functions reveals that the performance improvement of our designs over the Xilinx design due to the reduction in latency and energy-efficient binding. In terms of energy–area–time (EAT), our designs based on Corollary 1 offer superior performance by 50%–79% compared with the Xilinx design.

The latency of our design ranges from 1/3–1/15 of the Xilinx design, resulting in less energy consumption. The leakage energy (leakage power \times latency) is proportional to the product of area and latency. The area–latency product of our design is smaller than that of the Xilinx designs, and thus leakage energy of our designs is smaller as compared to the Xilinx designs.

Table VI shows the energy, area, and latency of on-chip designs in Corollary 1 for various problem sizes. While the comparison of off-chip and on-chip designs is not fair, it is useful to analyze the effect of I/O ports and on-chip memory. For the off-chip design, data are fed via the I/O ports of the FPGA. The power used by a 16-b I/O port is 80 mW. The input ports use less power than the output ports since the output ports need to handle a large fan-out. For the on-chip design, all data are stored in BSRAMs. The power dissipation of the read and write operations on a single-port BSRAM with 50% access rate is 35 mW.

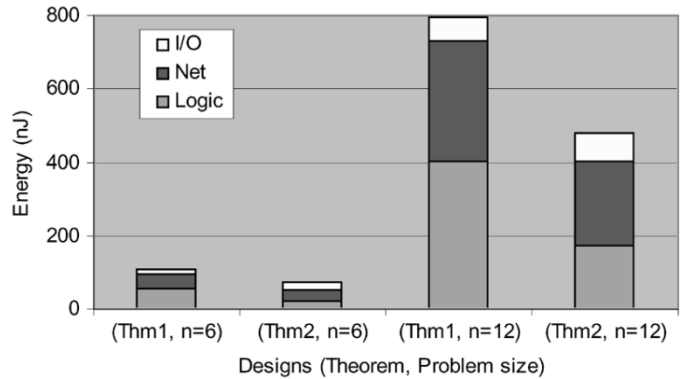


Fig. 9. Energy distribution over logic, net, and I/O for Theorem 1 and 2.

Thus, data access from a BSRAM is more energy-efficient than from an I/O port. However, we likely have a combination of both situations, where we read the data from input ports and store the result in BSRAMs for further computation. The final result would be obtained after several computations and output via output ports. Thus, the design tradeoffs have to be performed at a system level where multiple kernels are integrated.

Fig. 9 shows the energy distribution among logic, net, and I/O ports for Theorems 1 and 2. The figures are based on the off-chip designs. Logic energy represents the energy used by the combinational logic in the design. Net energy is the energy used by interconnect wires, and I/O energy by input/output ports. Note that the algorithm of Theorem 2 uses more energy in the nets than in the logic, unlike Theorem 1. For example, the ratio of (logic energy)/(net energy) is 1.21 for Theorem 1 and 0.75 for Theorem 2. The reason is that Theorem 2 has a more complex design for a PE and uses more interconnects between PEs. In the low-level simulation, Theorem 1 runs at 150 MHz. However, Theorem 2 runs at 143 MHz or less due to a more complex design involving more interconnect wires.

Another factor that affects the energy dissipation in Theorem 2 is the parameter, r , which determines the degree of parallelism. Table VII shows the results from the

TABLE VII
PERFORMANCE COMPARISON OF VARIOUS OFF/ON-CHIP DESIGNS FOR
THEOREM 2

Design	Metric	Matrix size							
		6x6	8x8	12x12			16x16		
Theorem 2 (Off-chip)	Block size(n/r)	2	2	2	3	4	2	4	
	Frequency(MHz)	143	143	143	117	127	143	130	
	Energy(nJ)	73.4	158.9	481	503	430	1212	1062	
	Latency(usec)	0.13	0.22	0.50	0.41	0.28	0.90	0.49	
	Area(slices)	1669	2233	3358	5879	6592	4472	8850	
	EATx1E-12	15.4	79.4	813	1213	804	4851	4627	
Theorem 2 (On-chip)	Block size(n/r)	2	2	2	3	4	2	4	
	Frequency(MHz)	143	143	143	100	110	143	110	
	Energy(nJ)	64.8	146.6	456	480	412	1113	1225	
	Latency(usec)	0.13	0.22	0.50	0.48	0.33	0.90	0.58	
	Area(slices)	1827	2416	3558	6670	6770	4700	9203	
	EATx1E-12	14.9	79.3	818	1535	913	4684	6559	

low-level simulation of the off-chip and on-chip designs for Theorem 2. As r increases, the usage of logic and interconnect increases and the latency decreases by a factor of r . Since the latency decreases as r increases, we can arrive at a design with reduced energy by decreasing r . If $r = 4$ for $n = 16$, we use a larger device, XC2V3000, instead of the XC2V1500, since the XC2V1500 can hold only 48 multipliers, while our design requires 64 multipliers. We do not use CLB-based multipliers due to low energy efficiency.

VII. CONCLUSION

New algorithms and architectures were developed for matrix multiplication to significantly reduce the energy dissipation and latency, when compared with the state-of-the-art FPGA-based designs. These improve the latency-optimal design and provide tradeoffs among energy, area, and latency. In our methodology, “energy hot spots,” which dissipate most of the energy, are identified through energy estimation functions. Algorithmic-level optimizations are performed to reduce the dissipation in these “energy hot spots” without an increase in latency. Low-level simulations were also performed to evaluate the chosen designs. The validity of our approach is also illustrated extensively with other designs such as discrete cosine transform [23], fast Fourier transform [10], and signal processing applications [9].

While we have shown many techniques for energy-efficient matrix multiplication, many FPGAs (e.g., Virtex-II/pro) do not provide low-level power features such as control for multiple power states or lower static power. When more features such as dynamic voltage scaling with dynamic frequency scaling are available, our design techniques can easily be ported to the newer FPGAs and exploit the new features at the algorithmic level effectively.

APPENDIX

Proof: [Proof of Theorem 2]

The $n \times n$ matrices A, B , and C are divided into r^2 submatrices, each of size $(n/r) \times (n/r)$, assuming n is divisible by r . Let $A_{xy}, B_{xy}, C_{xy}, 1 \leq x, y \leq r$, denote the submatrices. Then, we have $C_{xy} = \sum_{k=1}^r A_{xk} \times B_{ky}, 1 \leq x, y \leq r$. We compute $C_{xy} = \sum_{k=1}^r A_{xk} \times B_{ky}$ in parallel for all $x, y, 1 \leq x, y \leq r$

using r^2 MACs per PE. $A_{xk} \times B_{ky}$ for each $x, y, 1 \leq x, y \leq r$, can be performed using 3 I/O ports and (n/r) PEs, each having one MAC, 4 registers, and two local memories of (n/r) words per PE, as stated in Theorem 1. Since the computations for all the submatrices of matrix C need to be performed in parallel in each PE, we need to duplicate the resources of a PE by a factor of r^2 , which is the number of the submatrices of C to be computed in parallel. This would require $3r^2$ I/O ports and each PE to have r^2 MACs, $4r^2$ registers, and $2r^2$ local memories of (n/r) words. We show how the number of registers and I/O ports can be reduced to $4r$ registers per PE and $3r$ I/O ports. PE $_j$ denotes the j -th PE from the left in Fig. 4(a), $j = 1, 2, \dots, (n/r)$. PE $_j$ computes column j of all submatrices $C_{xy}, 1 \leq x, y \leq r$. An MAC is used to update column j of each submatrix $C_{xy}, 1 \leq x, y \leq r$, requiring a total of r^2 MACs per PE.

- 1) For each pair of x and $y, 1 \leq x, y \leq r$, we show how $C_{xy} = \sum_{k=1}^r A_{xk} \times B_{ky}$ can be performed in $((n^2/r) + (2n/r))$ cycles using (n/r) PEs with an MAC and 4 registers per PE and 3 I/O ports. C'_{xy} represents the intermediate results for C_{xy} . Using Theorem 1, we can perform $C'_{xy} = C_{xy} + A_{xk} \times B_{ky}$ for any specific combination of x, k , and $y, 1 \leq x, k, y \leq r$, in $((n^2/r^2) + (2n/r))$ cycles using the aforementioned PEs. For each pair of x and $y, 1 \leq x, y \leq r, C_{xy} = \sum_{k=1}^r A_{xk} \times B_{ky}$ is obtained by performing $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ in a serial manner with k increasing from 1 to r . A preliminary analysis reveals that this would take $((n^2/r^2) + (2n/r)) \times r = ((n^2/r) + 2n)$ cycles. However, a close look at the data movement in the Proof of Theorem 1 reveals that the input of the last column of submatrix A_{xk} to the array can be overlapped with input of the first row of submatrix $B_{(k+1)y}$ for $k = 1, 2, \dots, r - 1$. Using the overlapping, $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ for $k = 1, 2, 3, \dots, r$ can be performed in a pipelined fashion, taking (n^2/r^2) cycles for each k . At the start, (n/r) cycles are needed to prefetch the first row of submatrix B_{1y} . At the end, after the last column of submatrix A_{xr} is input, (n/r) additional cycles are needed to move it through the array of (n/r) PEs to complete the updates for C_{xy} . This leads to an execution time of $(n/r) + r \times (n^2/r^2) + (n/r) = (n^2/r) + (2n/r)$ cycles, instead of $((n^2/r) + 2n)$ cycles.
- 2) We show how $C_{xy} = \sum_{k=1}^r A_{xk} \times B_{ky}$ can be performed in parallel for all pairs of x and $y, 1 \leq x, y \leq r$, in $((n^2/r) + (2n/r))$ cycles using (n/r) PEs with r^2 MACs, $2r^2$ local memories of (n/r) words, and $4r$ registers per PE, and $3r$ I/O ports. C'_{xy} is the intermediate result for C_{xy} . In stage $k, 1 \leq k \leq r, C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ is performed in parallel for all $1 \leq x, y \leq r$. I/O ports, IOA $_1, IOA_2, \dots, IOA_r$ are used to feed submatrices $A_{1k}, A_{2k}, \dots, A_{rk}$ simultaneously to the array. I/O ports IOB $_1, IOB_2, \dots, IOB_r$ are used to feed submatrices $B_{k1}, B_{k2}, \dots, B_{kr}$ simultaneously to the array. An MAC, MAC $_{xy}$, is used to perform $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ for each pair of x and $y, 1 \leq x, y \leq r$. Note that elements of A_{xk} can be shared among r MACs, MAC $_{xy}, 1 \leq y \leq r$, in each PE while elements of B_{ky} can be shared among r MACs, MAC $_{xy}, 1 \leq x \leq r$, in

each PE. For example, in any stage $k, 1 \leq k \leq r$, all the r MACs, $\text{MAC}_{xy}, 1 \leq y \leq r$, in parallel perform $C'_{xy} = C_{xy} + A_{xk} \times B_{ky}$ and need A_{xk} . This sharing allows us to reduce the number of registers required per PE from $4r^2$ to $4r$ and the number of I/O ports from $3r^2$ to $3r$. A direct application of Theorem 1 without sharing would use 4 registers per PE and 3 I/O ports to feed each MAC, requiring $4r^2$ registers per PE and $3r^2$ I/O ports to the array. Column j of submatrix C'_{xy} is updated by MAC_{xy} and stored in a local memory of size (n/r) , Cbuf_{xy} , for each $x, y, 1 \leq x, y \leq r$, in PE_j .

- 3) One set of local memories (Cbufs) of (n/r) words in PE_j is used to store intermediate results for column j of submatrix C_{xy} for any $x, y, 1 \leq x, y \leq r$. Another set of local memories (CObufs) of (n/r) words is required to buffer the final results from PE_{j+1} to PE_j . Thus, a total of $2r^2$ local memories of (n/r) words per PE are required. Starting from the $((n^2/r) + 1)$ -th cycle, n^2 results of c'_{ij} are generated from all PEs during the next $n + (n/r) - 1$ cycles. CObuf is necessary in each PE otherwise the final c'_{ij} from one matrix multiplication would be overwritten by the intermediate c'_{ij} of the next matrix. PE_j stores either the output from PE_{j+1} via ports $\text{C1}_{in}, \text{C2}_{in}, \dots, \text{Cr}_{in}$ or c'_{ij} from MAC_{xy} of PE_j in CObufs . For the first (n/r) cycles, the final c'_{ij} at PE_j is output to PE_{j-1} via ports $\text{C1}_{out}, \text{C2}_{out}, \dots, \text{Cr}_{out}$. For the next $(n^2/r) - (n/r)$ cycles, the stored c'_{ij} of PE_j in CObuf is output to PE_{j-1} . The outputs from PE_1 are the resulting matrix C . Starting from the $((n^2/r) + 3)$ -th cycle, PE_j stores the final c'_{ij} of PE_{j+1} via ports $\text{C1}_{in}, \text{C2}_{in}, \dots, \text{Cr}_{in}$ in CObufs for (n^2/r) cycles.
- 4) Fig. 4(a) shows our architecture for $r = 2$ and Fig. 4(b) shows the accompanying algorithm. Let A_{xy}, B_{xy} , and $C_{xy}, 1 \leq x, y \leq 2$, denote submatrices, each of size $(n/2) \times (n/2)$. In the first stage, $C'_{xy} = A_{x1} \times B_{1y}$, are performed in parallel for all $1 \leq x, y \leq 2$ by feeding the elements of A_{11}, A_{21}, B_{11} , and B_{12} through the 4 input ports. In the second stage, $C'_{xy} = C'_{xy} + A_{x2} \times B_{2y}$ is performed in parallel for all $1 \leq x, y \leq 2$. Each stage takes $(n^2/4) + n$ cycles from Theorem 1. Since overlapping is possible between the end of the first phase and the start of the second phase, the total number of cycles for both stages combined is $(n^2/2) + n$, as explained before. For larger values of r , there is greater parallelism within the PEs and hence the execution time greatly reduces. ■

ACKNOWLEDGMENT

The authors wish to thank H. Krishnaswamy for helpful discussions and comments on the earlier draft of this work. The authors would also like to thank J. Ou for editorial assistance.

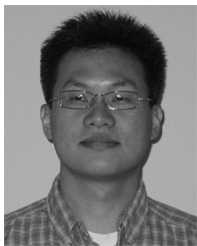
REFERENCES

- [1] A. Amira, A. Bouridane, and P. Milligan, "Accelerating matrix product on reconfigurable hardware for signal processing," in *Proc. 11th Int. Conf. Field-Programmable Logic and Its Applications (FPL)*, 2001, pp. 101–111.
- [2] B. Bass, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, Mar. 1999.
- [3] J. Becker, T. Pionteck, and M. Glesner, "DRAM: A dynamically reconfigurable architecture for future mobile communication applications," in *Proc. Field-Programmable Logic and Its Applications (FPL)*, 2002, pp. 312–321.
- [4] BenPRO Virtex TM-II Pro DIME-II Module [Online]. Available: http://www.nallatech.com/solutions/products/embedded_systems/dime2/trc_modules/benpro/
- [5] A. Bogliolo, L. Benini, and G. Micheli, "Regression-based RTL power modeling," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 5, no. 3, pp. 337–372, 2000.
- [6] B. L. Bowerman and R. T. O'Connell, *Linear Statistical Models—An Applied Approach*, 2nd ed. Pacific Grove, CA: Brooks/Cole, 1990.
- [7] G. Brebner and N. Bergman, "Reconfigurable computing in remote and harsh environments," in *Proc. Field-Programmable Logic and Its Applications (FPL)*, 1999, pp. 195–204.
- [8] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid energy estimation of reconfigurable architectures," *J. Supercomputing*, vol. 26, no. 3, pp. 259–281, Nov. 2003.
- [9] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy efficient signal processing using FPGAs," in *Proc. Field Programmable Gate Array (FPGA)*, 2003, pp. 225–234.
- [10] S. Choi, G. Govindu, J.-W. Jang, and V. K. Prasanna, "Energy-efficient and parameterized designs of fast fourier transforms on FPGAs," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, 2003, pp. 521–524.
- [11] O. D. Fidanci, D. Poznanovic, K. Gaj, T. El-Ghazawi, and N. Alexandridis, "Performance and overhead in a hybrid reconfigurable computer," in *Proc. Reconfigurable Architecture Workshop (RAW)*, 2003, pp. 176–183.
- [12] J. Frigo, M. Gokhale, and D. Lavenier, "Evaluation of the streams-C C-to-FPGA compiler: An applications perspective," *Proc. Field Programmable Gate Arrays (FPGA)*, pp. 134–140, 2001.
- [13] R. Hogg and E. Tanis, *Probability and Statistical Inference*, 6th ed. Upper Saddle River, NJ: Prentice-Hall, 2001, pp. 656–657.
- [14] J.-W. Jang, S. Choi, and V. K. Prasanna, "Energy efficient matrix multiplication on FPGAs," *Proc. Field-Programmable Logic and Its Applications (FPL)*, pp. 534–544, 2002.
- [15] H. T. Kung and C. E. Leiserson, "Systolic arrays for (VLSI)," *Introduction to VLSI Systems*, 1980.
- [16] W. J. C. Melis, P. Y. K. Cheung, and W. Luk, "Image registration of real-time broadcast video using the UltraSONIC reconfigurable computer," *Proc. Field Programmable Logic and Its Applications (FPL)*, pp. 1148–1151, 2002.
- [17] O. Mencer, M. Morf, and M. J. Flynn, "PAM-Blox: High performance FPGA design for adaptive computing," in *Field-Programmable Custom Computing Machines (FCCM)*, 1998, pp. 167–174.
- [18] P. Master and P. M. Athanas, "Reconfigurable computing offers options for 3G," *Wireless Syst. Des.*, vol. 4, no. 1, pp. 20–27, 1999.
- [19] ML310 Virtex-II Pro Development Platform [Online]. Available: http://www.xilinx.com/univ/ML310/ml310_mainpage.html
- [20] Model-Based Integrated Simulation [Online]. Available: <http://milan.usc.edu>
- [21] W. Najjar, W. Böhm, B. Draper, J. Hammes, R. Rinker, R. Beveridge, M. Chawathe, and C. Ross, "From algorithms to hardware—A high-level language abstraction for reconfigurable computing," in *Proc. IEEE Computer*, vol. 36, Aug. 2003, pp. 63–69.
- [22] V. K. P. Kumar and Y. Tsai, "On synthesizing optimal family of linear systolic arrays for matrix multiplication," *IEEE Trans. Comput.*, vol. 40, no. 6, pp. 770–774, 1991.
- [23] R. Scrofano, J.-W. Jang, and V. K. Prasanna, "Energy-Efficient discrete cosine transform on FPGAs," in *Proc. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2003, pp. 215–221.
- [24] L. Shang, A. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in *Proc. Field-Programmable Gate Arrays (FPGA)*, 2002, pp. 157–164.
- [25] N. Srivastava, J. L. Trahan, R. Vaidyanathan, and S. Rai, "Adaptive image filtering using run-time reconfiguration," in *Proc. Reconfigurable Architectures Workshop (RAW)*, 2003.
- [26] Xilinx Application Note: Virtex-II Series and Xilinx ISE 4.1i Design Environment (2001). [Online]. Available: <http://www.xilinx.com>
- [27] XPower [Online]. Available: http://toolbox.xilinx.com/docsan/xilinx5/help/xpower/html/b_getting_started/b_overview.htm



Ju-Wook Jang (M'05) received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea in 1983, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Seoul, in 1985, and the Ph.D. degree in electrical engineering from the University of Southern California (USC), Los Angeles, in 1993.

From 1985 to 1988 and 1993 to 1994, he was with Samsung Electronics, Suwon, Korea, where he was involved in the development of a 1.5-Mb/s video codec and a parallel computer. Since 1995, he has been with Sogang University, Seoul, Korea, where he is currently a Professor. His research interests are in the development of application-specific algorithms for energy-efficient design of computation intensive signal processing applications. He has also built systems for videoconferencing, streaming, home networks and ad hoc networks using protocols like RTP, SIP, multicast, and IPv6.



Seonil B. Choi (S'94–M'05) received the B.S. degree from Korea University, Seoul, Korea, in 1994 and the M.S. and Ph.D. degrees from the University of Southern California, Los Angeles, in 1996 and 2004, respectively, all in electrical engineering.

Since 2004, he has been with the Intel Corporation, Chandler, AZ. His research interests are in developing high-performance and energy-efficient designs for image and video signal processing applications on FPGA-based platforms and on ASIC. His other research interests are in designing FPGA-based prototyping platforms and developing validation methods for consumer electronics.



Viktor K. Prasanna (M'84–SM'91–F'96) received the B.S. degree in electronics engineering from Bangalore University, Bangalore, India, the M.S. degree from the Indian Institute of Science, Bangalore and the Ph.D. degree in computer science from The Pennsylvania State University, University Park, in 1983.

Currently, he is a Professor with the Department of Electrical Engineering as well as in the Department of Computer Science, University of Southern California (USC), Los Angeles. He is also an Associate Member of the Center for Applied Mathematical Sciences (CAMS) at USC. He served as the Division Director for the Computer Engineering Division during 1994–1998. His research interests include parallel computation, computer architecture, VLSI computations, and high-performance computing for signal and image processing, and vision. He has published extensively and consulted for industries in the above areas. He serves on the Editorial Boards of the *Journal of Parallel and Distributed Computing* and Elsevier's *Journal on Pervasive and Mobile Computing*.

Dr. Prasanna has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He is the Steering Co-Chair of the International Parallel and Distributed Processing Symposium [merged IEEE International Parallel Processing Symposium (IPPS) and the Symposium on Parallel and Distributed Processing (SPDP)] and is the Steering Chair of the International Conference on High Performance Computing (HiPC). He was the Program Chair of the IEEE/ACM First International Conference on Distributed Computing in Sensor Systems (DCOSS '05) meeting. He has served on the Editorial Boards of the PROCEEDINGS OF THE IEEE, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. Currently, he is the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS. He was the founding Chair of the IEEE Computer Society Technical Committee on Parallel Processing.