An Energy-Efficient FPGA-based Matrix Multiplier

Yiyu Tan
RIKEN Advanced Institute for Computational Science
7-1-26 Minatojima-minami-machi, Chuo-ku
Kobe, Hyogo, Japan
tan.yiyu@riken.jp

Toshiyuki Imamura
RIKEN Advanced Institute for Computational Science
7-1-26 Minatojima-minami-machi, Chuo-ku
Kobe, Hyogo, Japan
imamura.toshiyuki@riken.jp

***Abstract*—Matrix multiplication is a fundamental operation of numerical linear algebra, and applied widely in high performance computing to solve scientific and engineering problems. It requires computer systems have huge computing capacity and data throughputs as problem size is increased, and consumes much more power. In this research, an OpenCL-based matrix multiplier is presented to improve energy efficiency. When data are single precision floating-point, and matrix dimension is 16384×16384, the matrix multiplier implemented by the FPGA board DE5a-NET achieves 240.34 GFLOPs in data throughput and 19.64 GFLOPs/W in energy efficiency, which are 296 times and 1964 times over the software simulation carried out on a PC with 32 GB DDR4 RAMs and an AMD processor Ryzen 7 1700 running at 3.0 GHz, respectively.**

Keywords—Matrix multiplication, FPGA, OpenCL.

I. Introduction

Matrix multiplication is the fundamental building block of numerical linear algebra, and greatly affects the high performance computing (HPC) applications. Even now, how to improve the performance of matrix multiplication at hardware and software level is still an open problem although many methods have already been developed to speed up computation through parallel programming in supercomputers or cluster systems. Unfortunately, because of the power wall and memory wall, performance and power improvements due to technology scaling and instruction level parallelism in general-purpose processors have ended. And the guiding metric to evaluate the performance of computing systems is evolved from computing power (FLOPs: floating-point operations per second) to energy efficiency (computing power/power consumption: FLOPs/W) at post Moore's era. How to reduce power consumption while increasing performance is one of the core concerns in future HPC systems. It is well known that heterogeneous systems with hardware specialization in the form of GPUs, FPGAs, and ASICs, offer a promising path toward major leaps in processing capability while achieving high energy efficiency. Particularly, FPGA, due to its high energy efficiency and reconfigurability, has been applied in data center and cloud computing in recent years [1-3].

The FPGA-based approaches accelerate matrix multiplication by exploiting spatial and temporal parallelism through circuit design techniques. Furthermore, memory units can be arranged close to arithmetic units to reduce data access overhead, data are kept by on-chip memory, and processing elements (PEs) communicate each other through routing wires instead of message passing to minimize memory bandwidth. In the earlier accelerators, PEs are arranged based on the onedimensional linear array architecture [4-10]. Data are exchanged between adjacent PEs directly or through broadcasting [8]. The main drawback of such architecture is the system parallelism, scalability, and data movement are limited. System is extended by cascading more PEs in series, which prolongs the system data path. To address this, the microarchitectures with the twodimensional systolic array were introduced in [11-13]. The twodimensional arrangement of PEs has been proven to be scalable relative to the ratio of problem size to local memory size, and different types of interconnects can be adopted to speed up data access, such as row/column broadcasting buses.

On the other hand, FPGA-based systems are usually designed by hardware descriptor language (HDL) in the past. The HDL-based design flow requires designers to have broad hardware knowledge, and developing period is long because of system verification and debugging. Particularly, it is big challenge to implement and debug system I/O interface, such as DDR memory controller, PCIe, and so on. Although modern FPGA integrates reconfigurable fabrics and hard-core processors on the same chip, and makes such implementation/debugging much easier, it is still timeconsuming. To overcome these disadvantages of the HDL-based design flow, high level synthesis is becoming popular in recent years, such as SDAccel from Xilinx, and OpenCL from Intel. The Intel FPGA SDK for OpenCL provides an environment including software and device drivers to exchange data between FPGA and host machine. It allows users to abstract away the traditional HDL-based FPGA design flow, and only focus on designing the kernel by using OpenCL programming language. The related I/O interfaces and drivers are generated by the compiler automatically. Holland proposed high-level synthesis optimization strategies to maximize the utilization of the DSPs and BRAMs in blocked matrix multiplication [14]. In this study, an OpenCL-based accelerator for matrix multiplication is presented and its performance is evaluated. The major contributions of this work are as follows.

(1) An OpenCL-based accelerator for blocked matrix matrix multiplication.

(2) Detail performance evaluation of the prototype implemented by using the DE5a-NET FPGA board, including data throughput, energy efficiency, and hardware resources.

The remainder of this paper is organized as follows. The system design and implementation are introduced in Section

II. In Section III, performance evaluation results are presented, followed by conclusions drawn in Section IV.

II. System Design and Implementation

A matrix-matrix multiplication can be theoretically viewed as a collection of matrix-vector multiplications, which are carried out independently. Consequently, it is easily implemented through cascading matrix-vector multiplication modules together to speed up computation. However, the onchip block RAMs and DSP blocks inside FPGA constraints the number of matrixvector multiplication modules. To reduce the requirement of memory block RAMs, blocked matrix multiplication algorithm is generally applied to divide the matrix into small sub-matrices, and inter-row and intra-row parallelism are put on the submatrices multiplications to speed up computation.

A. System Design

In the blocked matrix multiplication, for example, matrix A multiplies matrix B, a block of matrix A along rows, and a block of matrix B along columns, are read into the local buffers (block RAMs inside FPGA) and multiplications are carried. After computations inside a block are finished, another blocks are read into, and multiplications are performed. This procedure is repeated until the product of matrix A and B is obtained. The whole system is designed by using OpenCL, and the multiplications inside blocks are performed by the processing elements defined by the execution model of the OpenCL, in which an N dimensional index space (NDRange) is defined, each work item inside the index space executes the kernel, and each work item may execute on one or more processing elements. The pseudo codes of the kernel are shown in Fig. 1. The matrices A and B are stored by row-major in the host, and are read into the system along the rows and columns, respectively. As shown in Fig. 1, two local buffers, A_local and B_local, are defined to store the block data read from the onboard DDR memory (lines 6 and 7). Along with reading data (lines 16 and 17), the blocks of the matrix B are transposed and stored in the local buffer (line 17) so that consecutive data are accessed to speed up data access during computation (line 22). The computation loop is unrolled completely (line 20), and the related pipeline is extended by replicating the multipliers and adders to increase the number of parallel operations and improve pipeline throughput. Furthermore, computations will not start until all data in blocks are read into the local buffers (line 19), and calculation results are written into the on-board DDR memory after computation is completed (line 27).

B. System Implementation

To verify the proposed matrix multiplier and evaluate its performance, the system is implemented by a FPGA board DE5a-NET from Terasic [15], which includes an Altera Arria 10 GX FPGA 10AX115N2F45E1SG and 8 GB on-board DDR3 memory. The FPGA contains 1518 hardened single-precision floating-point multipliers, 427,200 ALMs, and 53 Mb M20K memory. The on-board memory is arranged at two independent channels with each being 4 GB. The kernel code is compiled into the intermediate representations, applied necessary optimizations, converted to Verilog, and then performed synthesis, placement and routing to generate the final FPGA bitstream by the Intel SDK for FPGA.

Before computation, the matrices A and B are written into the different banks of the on-board memory from the host machine through PCIe bus. Therefore, the data of matrices A and B are accessed independently through different channels. The product, namely matrix C, is firstly stored into the same memory bank as the matrix A, and finally written back to the host machine. When the block size is 128×128, the dimension of both matrices is 16,384×16,384, data are single-precision floating-point, and the number of work items within a work group being executed in a single instruction multiple data (SIMD) manner is 4, the system consumes 92,002 ALMs, 520 DSP blocks, and 1774 RAM blocks, and runs at 235 MHz.

Pseudo codes for blocked matrix multiplication

```
1: __kernel void matrix_mult( __global float *restrict C,
2: __global float *A, __global float *B,
3: int A_width, int B_width)
4: {
5: //defining local storage for blocks of matrices A and B
6: __local float A_local[BLOCK_SIZE][BLOCK_SIZE];
7: __local float B_local[BLOCK_SIZE][BLOCK_SIZE];
8: // calculating loop started and stopped points
9 a_start = A_width*BLOCK_SIZE*block_y;
10: a_end = a_start + A_width - 1;
11: b_start = BLOCK_SIZE * block_x;
12: sum = 0.0f;
13: // starting computation
14: for (i=a_start, j=b_start; i<=a_end; i+=BLOCK_SIZE, j+=
(BLOCK_SIZE*B_width)) {
15: // load the matrices into local memory, and perform B<=BT
16: A_local[local_y][local_x] = A[i+A_width*local_y+local_x];
17: B_local[local_x][local_y] = B[j+B_width*local_y+local_x];
18: // waiting for the entire blocks to be loaded.
19: barrier(CLK_LOCAL_MEM_FENCE);
20: #pragma unroll
21: for (k = 0; k< BLOCK_SIZE; ++k) {
22: sum += A_local[local_y][k] * B_local[local_x][k]; }
23: // waiting for completion of computation.
24: barrier(CLK_LOCAL_MEM_FENCE);
25: }
26 // storing result into matrix C
27: C[get_global_id(1)*get_global_size(0)+get_global_id(0)]=sum;
28: }
```

Fig. 1. OpenCL code for blocked matrix multiplication

III. Performance Evaluation

The matrix multipliers with different parameters are implemented by using the DE5a-NET FPGA board, and their performance is evaluated. As a comparison, the same systems are programed by using C programming language, and executed on a PC with AMD Ryzen 7 processor running at 3 GHz. The design environment and technology specification of the FPGA and CPU are presented in Tables I. The execution time, data throughput, hardware resource utilization, and power consumption are measured, and the related system performance are estimated. During evaluation, data are single precision floating-point, and the matrices are square (n×n). The C codes 515 in the software simulation are compiled by the gcc with the option -O3, mcmodel=large, and - fopenmp to use 16 threads in the PC.

TABLE I. TECHNOLOGY SPECIFICATION

	FPGA	CPU
Model	Arria 10 GX (10AX115N2F45E1SG)	AMD Ryzen 7 1700
Cores	1518 DSP blocks	8 cores (16 threads)
Clock frequency	About 300 MHz	3.0 GHz
On-chip memory	6.25 MB block RAMs	L1 cache: 768 KB L2 cache: 4 MB L3 cache: 16 MB
Fabrication	20 nm	14 nm
Operating system	CentOS 7.0	CentOS 6.8
Compiler	Intel SDK for FPGA 16.1	gcc 4.4.7
Programming language	OpenCL	C

A. Data Throughput

Fig. 2 shows the data throughput in the case of different problem sizes in the FPGA system and software simulation on the PC. The block size of matrix multiplication is 128×128, and the SIMD is 4. In Fig. 2, as the matrix scale is increased, the data throughput is firstly increased, and finally stays at about 240 GFLOPs even if the problem size is further increased in the FPGA system. In contrast, it fluctuates and decreases to the 0.81 GFLOPs when the problem size is 16,384 in the software simulation. When the matrix scale is 16,384, the data throughput of the FPGA system is about 296 times over that of the software simulation. In the FPGA system, matrix elements are firstly written into the on-board DDR memory from the host, and then blocks of the matrices are read into the local buffers from the onboard DDR memory, computations are carried out, and finally the results are written back to the host machine from the onboard memory through PCIe bus. Hence, the execution time consists of computation time and the time taken by data transfer between the host and the FPGA board. In general, the data transfer between the host and FPGA board dominates the operations when matrix scale is small while computations become the main operations as the matrix scale is increased. For example, the computation time is about 98.38% and 29.23% of the execution time when the matrix scale is16384×16384 and 128×128, respectively.

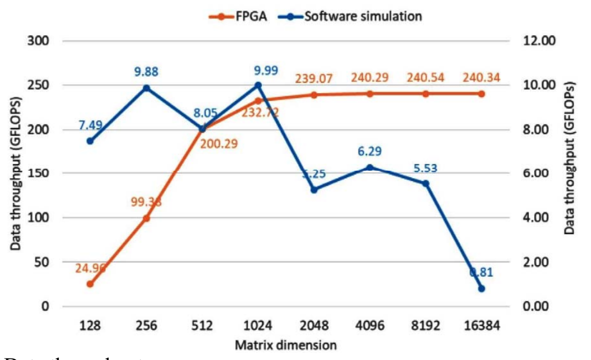

Fig. 2. Data throughput

B. Energy Efficiency

To estimate the energy efficiency of the FPGA system and the software simulation on the PC, the current and voltage of the host machine and the PC are measured every 200 ms when the FPGA system and the PC is idle and active. The power consumption is calculated by multiplying the voltage and the current difference. And the energy efficiency is obtained by using equation 1.

$$P_{efficiency} = \frac{E_{data\_throughput}}{Vx(I_{active} - I_{idle})} \quad (1)$$

where E is the data throughput, V is the voltage, and I is the current. Fig. 3 shows the energy efficiency of the FPGA system and the software simulation, and the speedup gain in the case of different problem sizes. As shown in Fig. 2, the data throughput is dropped significantly as the matrix dimension is increased in the software simulation. This results in the energy efficiency being decreased sharply from 9.60 GFLOPs/W to 0.01 GFLOPs/W. However, the energy efficiency is changed relatively smoothly in the FPGA system. The highest and lowest energy efficiency occur at the matrix scale being 1024 and 16,384, which are 30.49 GFLOPs/W and 19.64 GFLOPs/W, respectively. As shown in Fig. 2, the data throughput is almost fixed after the matrix dimension is larger than 1,024 in the FPGA system while it is increased sharply when the matrix dimension is increased from 128 to 1024. As a result, the FPGA system achieves the highest and lowest energy efficiency in the case of the problem size being 1024 and 16,384, respectively. On the other hand, due to the sharp drop of the energy efficiency in the software simulation, the speedup gain of the energy efficiency of the FPGA system over the software simulation is increased as the matrix scale is increased. When matrices are 16384×16384, the energy efficiency of the FPGA system is about 1964 times over the software simulation even if the fabrication technology of the Arria 10 is 20 nm while that of the processor Ryzen 7 1700 is 14 nm.

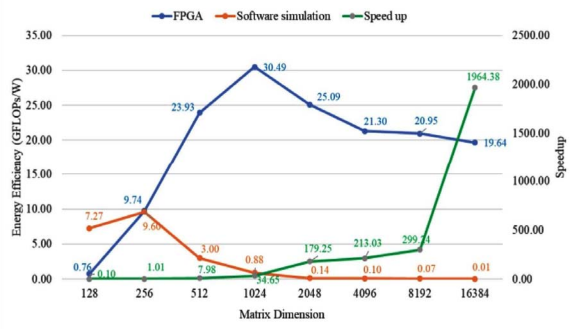

Fig. 3. Energy efficiency and speedup gain

C. Hardware Resources

In the current design, blocks of both matrices are read into the local buffers from the on-board DDR memory at each iteration, and the number of iterations is calculated according to the block size and matrix scale. Consequently, the required hardware resources are determined by the block size. Even if the matrix dimension is different, it only affects the number of iterations, and has small impacts on the hardware resource consumption. As the block size is increased, the hardware resources are required more, especially DSP blocks, which is applied to implement floating-point multipliers and adders. Table II presents the hardware resource consumption and clock frequency in the case of different block sizes when matrices are 16384×16384 and the SIMD is 4. In Table II, when the block size is 128×128, the DSP blocks are used about 34% while the block RAMs are consumed 65% of the capacity in the FPGA. If the block size is further increased to 256×256, although the data throughput can be enhanced by using more DSP blocks, the required block RAMs will be beyond the FPGA capacity. Thus, the size of the block RAMs affects system performance significantly in the current design, and techniques to reuse data and increase the number of processing elements may be adopted to reduce memory requirement and increase data throughput. On the other hand, as the block size is increased, the clock frequency is degraded because of long data path. When the block size is increased from 64×64 to 128×128, the clock frequency is reduced about 9%.

TABLE II. HARDWARE RESOURCE AND CLOCK FREQUENCY

Block Size	Logic Utilization (in ALMs)	Block Memory Bits	RAM Blocks	DSP Blocks	Clock Frequency (MHz)
8×8	51600(12%)	3779088(7%)	382(14%)	40/1518(3%)	281
16×16	53363(12%)	3879440(7%)	427(16%)	72(5%)	296
32×32	56712(13%)	4274704(8%)	524(19%)	136(9%)	295
64×64	70657(17%)	6317200(11%)	753(28%)	264(17%)	259
128×128	92002(22%)	14747408(27%)	1774(65%)	520(34%)	235

IV. Conclusion

Matrix multiplication is a fundamental operation of numerical linear algebra, and widely applied to solve science and engineering problems. In this research, an OpenCL-based matrix multiplier is designed and implemented by using a FPGA board DE5a-NET, and its performance is evaluated. Compared with the software simulation on a PC with 32 GB RAMs and an AMD Ryzen 7 1700 processor, the FPGA-based matrix multiplier achieves much higher data throughput and energy efficiency. On the other hand, although high-level synthesis approach can lighten the design complexity, system optimization depends on the compiler's performance. This needs developers to have rich programming techniques to design the desired systems by using pragmas and optimization options. In future work, the techniques to reduce the requirement of the on-chip block RAMs and increase the utilization of DSP blocks will be investigated to improve system performance.

Acknowledgment

Thanks for Intel's donation of the FPGA board DE5a-NET through University Program.

References

[1] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, et al, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services", the 41st annual international symposium on Computer architecture, 2014, pp. 13-24.

[2] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, et al, "A Cloud- Scale Acceleration Architecture", the 49th Annual IEEE/ACM International Symposium on Microarchitecture , 2016 .

[3] Jian Ouyang; Shiding Lin; Wei Qi; Yong Wang; Bo Yu; Song Jiang, "SDA: Software-defined accelerator for large-scale DNN systems", 2014 IEEE Hot Chips Symposium (HCS).

[4] Ling Zhuo and Viktor K. Prasanna, "High-Performance Designs for Linear Algebra Operations on Reconfigurable Hardware", IEEE Transactions on Computers, Vol. 57, No. 8, 2008, pp. 1057-1072

[5] Yong Dou S. Vassiliadis G. K. Kuzmanov G. N. Gaydadjiev, "64-bit Floating-Point FPGA Matrix Multiplication", ACM/SIGDA 13th international symposium on Field-programmable gate arrays, 2005, pp. 86-95.

[6] Guiming Wu, Yong Dou, and Miao Wang, "High Performance and Memory Efficient Implementation of Matrix Multiplication on FPGAs", 2010 International Conference on Field-Programmable Technology (FPT), 2010, pp. 134-137.

[7] Kiran Kumar Matam, Hoang Le, and Viktor K. Prasanna, "Energy Efficient Architecture for Matrix Multiplicaiton on FPGAs", IEEE International Conference on Field Programmable Logic and Applications (FPL), 2013, pp.1-4.

[8] V. B. Y. Kumar, S. Joshi, S. B. Patkar, and H. Narayanan, "FPGA based high performance double-precision matrix multiplication," International Journal of Parallel Programming, Vol. 38, 2010, pp. 322–338.

[9] Z. Jovanovic, and V. Milutinovic, "FPGA accelerator for floating-point matrix multiplication", IET Computers & Digital Techniques, Vol. 6, No. 4, 2012, pp. 249–256.

[10] Ju-Wook Jang, Seonil B. Choi, and Viktor K. Prasanna, "Energy and Time-Efficient Matrix Multiplication on FPGAs", IEEE Transactions on Very Large Scale Integration Systems, Vol. 13, No. 11, 2005, pp. 1305- 1319.

[11] Ardavan Pedram et al, "Algorithm, Architecture, and Floating-point Unit Codesign of a Matrix Factorization Accelerator", IEEE Transactions on Computers, Vol. 63, No. 8, 2014, pp. 1854-1867.

[12] Ardavan Pedram et al, "Codesign Tradeoffs for High-Performance, Low- Power Linear Algebra Architectures", IEEE Transactions on Computers, Vol. 61, No.12, 2012, pp. 1724-1736.

[13] Roberto Perez-Andrade, Cesar Torres-Huitzil, and Rene Cumplido, "Processor arrays generation for matrix algorithms used in embedded platforms implemented on FPGAs", Microprocessors and Microsystems, Vol. 39, 2015, pp.576–588.

[14] E. H. D'Hollander, "High-Level Synthesis Optimization for Blocked Floating-Point Matrix Multiplication", ACM SIGARCH Computer Architecture News, 2016, pp. 74-79.

[15] http://www.terasic.com.tw/cgibin/page/archive.pl?Language=English &No=526