FPGA-Based Multi-precision Architecture for Accelerating Large-Scale Floating-Point Matrix Computing

Longlong Zhang, Yuanxi Peng, Xiao Hu, Ahui Huang, Tian Tian

FPGA-based Multi-precision Architecture for Accelerating Large-scale Floating-point Matrix Computing

Longlong Zhang[1,2], Yuanxi Peng[1], Xiao Hu[2], Ahui Huang[1,2], and Tian Tian[2]

[1] State Key Laboratory of High Performance Computing, School of Computer, National University of Defense Technology
[2] Institute of Microelectronics, School of Computer, National University of Defense Technology
{pyx,xiaohu}@nudt.edu.cn

Abstract. Matrix computing plays a vital role in many scientific and engineering applications, but previous work can only handle the data with specified precision based on FPGA. This study first presents algorithms, data flows, and mapping strategies to match the hardware structure for matrix computing of different precisions. Then, we propose a unified multi-precision matrix computing unit core that can handle three precisions and three matrix operation modes and can be used as a coprocessor for large-scale matrix computing which has advantages of low storage and high efficiency. Finally, we build a complete matrix computing acceleration system and deploy it on FPGA using 128 processing elements (PEs). The experimental results show that the accelerator achieves a maximum frequency of 180 Mhz, and matrix computing of double-precision, single-precision, and half-precision floating-point data performs 46.1 GFLOPS, 92.1 GFLOPS, and 184.3 GFLOPS respectively, which is superior to other current designs in terms of application range and performance.

Keywords: large-scale matrix computing · accelerator · multi-precision floating-point · FPGAs.

1 Introduction

Matrix computing is widely used in the fields of computing science and engineering applications, such as signal processing [1], image processing [2], and convolutional neural networks [3]. In recent years, although matrix computing has achieved good performance on many acceleration platforms such as CPU, GPU, TPU, and FPGA, its computing performance is still the bottleneck for the performance improvement of the entire system.

Compared with other platforms, field-programmable gate arrays (FPGAs) have the advantage of designing computing structures and storage schemes for specific applications. Many studies have shown that FPGAs are superior to other

platforms in terms of performance and power consumption [4], and are suitable as a low-cost hardware accelerator for matrix computing.

There have been many solutions for matrix computing based on FPGAs. Some scholars have proposed algorithms and structures for fixed-point matrix multiplication [5] and matrix calculation [6]. A systolic array structure has achieved high data throughput and computing speed [7,8]. But its demand for the number of PEs and I/O bandwidth is very high. Another linear array structure is widely used in matrix multiplication [9-11]. Some studies have carried out different types of optimization based on the structure [12], such as storage optimization [13], I/O, and storage scheduling optimization [14]. This structure has also achieved better results in the acceleration of convolutional neural networks [15,16], multi-operation and continuous matrix computing [17].

However, matrix computing based on FPGAs still faces the following concerns. Firstly, the previous work was specifically designed for matrix computing with certain data precision. After determining the data type, the calculation structure can not handle the data with another precision. For example, the signal processing in the controller has a greater demand for matrix computing with different precisions, which requires a special hardware accelerator to speed up real-time computing capabilities. Secondly, most of the existing research was interested in matrix multiplication. In many engineering applications, such as the filtering algorithm, matrix addition and matrix subtraction are used together with matrix multiplication. Therefore, they are both important calculation models. Thirdly, consider that the matrix calculation accuracy in the deep learning model does not need to be too high. It is necessary to design a half-precision floating-point matrix calculation accelerator that satisfies performance and calculation efficiency.

This study aims to develop an efficient multi-precision matrix computing acceleration (MCA) unit core based on a unified architecture. The major contributions of this work are as follows:

- Based on the parallel block matrix computing algorithm and linear array, we present two parallel single-precision floating-point matrix computing unit. Through the splicing data and the mapping strategy from algorithm to calculation structure, each PE can complete two single-precision floating-point matrix computing in parallel, which truly improves the computing speed.
- The proposed multi-precision MCA unit core with a unified structure can handle three precisions (double-precision, single-precision, and half-precisionand three commonly used matrix operations (matrix multiplication, matrix addition and matrix subtractionwhich enhances the adaptability of data types.
- We develop the MCA system based on the DSP-FPGA platform. Compare to some state-of-the-art structures, the proposed system meets engineering needs and achieves better performance.

The remainder of this paper is organized as follows. In Section 2, the parallel block matrix computing algorithm and linear array are introduced. The details of single-precision, half-precision floating-point matrix computing, multi-precision

functional component, the MCA core and system are described in Section 3. The implementation results and discussion are shown in Section 4. Finally, Section 5 concludes the work.

2 The parallel block matrix computing algorithm and linear array

The matrix multiplication involved in this article is shown in Equation (1), where matrix A, matrix B, and matrix C are dense matrices of arbitrary size.

$$C_{M\times N} = A_{M\times K} + B_{K\times N} + C_{M\times N} \tag{1}$$

We introduce a parallel block matrix multiplication algorithm [17]. As shown in Algorithm 1, it includes three outer loops and two inner loops. The outer loops with loop variables T_p, T_{t1}, and t2 are used for the data transmission of the sub-matrix A and B, and the initialization of the sub-matrix C. The inner loops with loop variables p and t1 are used for the calculation in each PE. After the block processing, the sizes of the sub-matrix blocks A, B, and C are $S_p \times K$, $K \times S_{t1}$, and $S_p \times S_{t1}$, respectively. Parameters S_p and S_{t1} are also represent the number of PE and the depth of on-chip RAM in each PE.

Algorithm 1. Parallel block matrix multiplication.

for T_p = 0 to M-1, S_p do
  for T_{t1} = 0 to N-1, S_{t1} do
  Initialize data block C[T_p:T_p+S_p-1,T_{t1}:T_{t1}+S_{t1}-1] to zero;
  for t2 =0 to K-1 do
  Load data block A[T_p:T_p+S_p-1, t2];
  Load data block B[t2,T_{t1}:T_{t1}+S_{t1}-1];
  for p=T_p to T_p+S_p-1 do
  for t1=T_{t1} to T_{t1}+S_{t1}-1 do
  C[p,t1]=C[p,t1]+A[p,t2]*B[t2,t1];
  Store data block C[T_p:T_p+S_p-1,T_{t1}:T_{t1}+S_{t1}-1];

As shown in Figure 1, the structure of linear array corresponding to Algorithm 1 is usually composed of a set of PEs, a data transfer controller, and two-stage FIFOs. Each PE includes two registers for storing elements of sub-matrix A and B, a FIFO for buffering data flow a, an on-chip RAM block for storing a row of elements of sub-matrix C, and a multiply-accumulate functional component for matrix multiplication.

Before the calculation starts, all elements in the on-chip RAM block are initialized to zero in advance, and a column of elements from sub-matrix A is preloaded and distributed to the register a in each PE. When the elements from the sub-matrix B flow into the linear array by rows, the linear array starts parallel calculations. Then, each PE completes the $c^k = a \times b + c^{k-1}$ operation

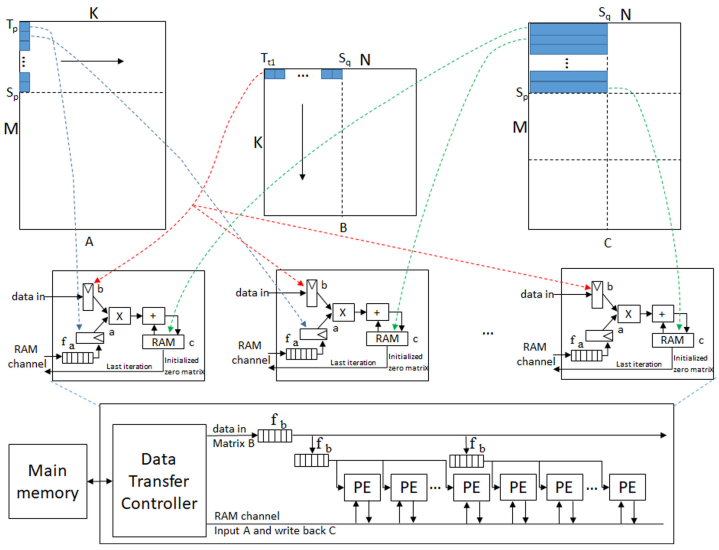

Fig. 1. Linear array architecture and computing process.

through the write-back mechanism and saves the intermediate result in the on-chip RAM block. After K iterations, the result of sub-matrix C is moved to the off-chip memory.

The above algorithm and structure can complete the functions of $A \times B \pm C$. When B=1, the addition and subtraction operations can be realized. It uses the identity matrix I to realize the matrix addition (subtraction) of sub-matrix A and sub-matrix C, as shown in Equation (2).

$$C_{M\times K} = A_{M\times K} + I_{K\times K} \pm C_{M\times K} \tag{2}$$

The matrix addition (subtraction) uses the same structure and similar data flow method as matrix multiplication. The sub-matrix C originally initialized to a zero matrix is loaded into the on-chip RAM block by the RAM channel in advance.

3 Multi-precision floating-point matrix computing based on a unified structure

To solve the problem of multiple data types for matrix computing, we first discuss how to design algorithms, data flows, and PE structures for single-precision and half-precision floating-point matrix computing based on the aforementioned double-precision floating-point matrix computing and linear array. Then, we introduce the corresponding floating-point multiply-accumulate (subtract) func-

tional component in each PE. Finally, we describe the multi-precision floating-point MCA unit core and take the core to build a multi-precision MCA system.

3.1 Two parallel single-precision floating-point matrix computing

In this section, a PE in the proposed structure can complete two single-precision floating-point data operations in the same clock cycles.

As shown in Algorithm 2, two parallel single-precision floating-point matrix multiplication uses the same block method, loop mode, and data flow as Algorithm 1. The difference is that the size of sub-matrix A and C becomes $2S_p \times K$ and $2S_p \times S_{t1}$, respectively. In the innermost loop, the original multiply-accumulate operation becomes two parallel multiply-accumulate operations.

Algorithm 2. Two parallel single-precision floating-point matrix multiplication.

```
Initialize data block C[0:2Sp-1,0:St1-1] to zero;
  for t2 =0 to K-1 do
    Load data block A[0:2Sp-1, t2];
    Load data block B[t2,0:St1-1];
      for p=0 to Sp-1 do
        for q=0 to St1-1 do
          C[2p,q]=C[2p,q]+A[2p,t2]*B[t2,q];
          C[2p+1,q]=C[2p+1,q]+A[2p+1,t2]*B[t2,q];
Store data block C[0:2Sp-1,0:St1-1];
```

Figure 2 shows the PE structure of several data types. The data sources in Figure 2(a) and Figure 2(b) are double-precision and single-precision floating-point data, respectively.

Generally, each PE has three source operands. In the structure of two parallel single-precision floating-point matrix multiplication, we use a data splicing mechanism when preparing the source operands. Two 32-bit single-precision floating-point data are spliced into a 64-bit source operand, that is, the two single-precision floating-point data are placed in the upper half and the lower half of the register, respectively.

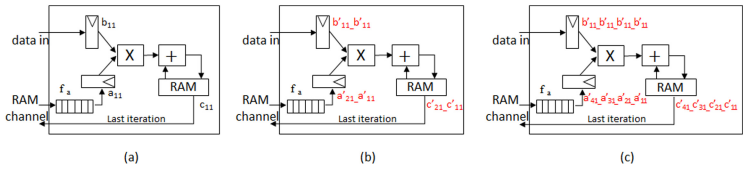

Fig. 2. The PE structure of matrix multiplication. (a) Double-precision floating-point. (b) Two parallel single-precision floating-point. (c) Four parallel half-precision floating-point.

Figure 3 describes the mapping process of single-precision floating-point matrix multiplication from algorithm to hardware structure. Firstly, the value of the sub-matrix C is initialized to zero matrix. Then, when loading a column of data ($2S_p$) from sub-matrix A, we stitch two adjacent single-precision data into a 64-bit operand and distribute it to the corresponding PE. When loading data row by row from the sub-matrix B, we also stitch two identical single-precision data into a 64-bit operand and distribute it to each PE. At the same time, the calculation of the linear array begins.

After the write-back mechanism and K iterations, the results are 64-bit operands. Each PE is responsible for calculating the two rows of sub-matrix C, where the lower 32 bits are the result of the previous row of sub-matrix C, and the upper 32 bits are the result of the subsequent row of sub-matrix C.

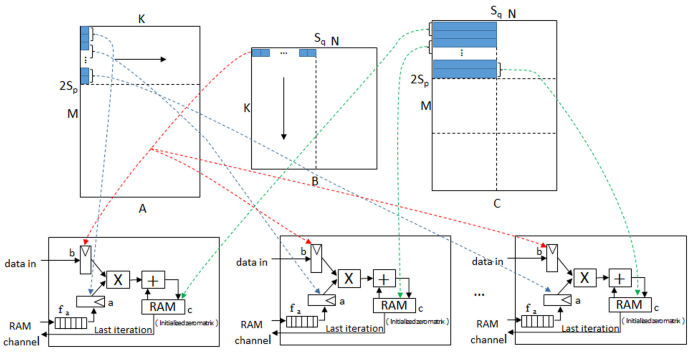

Fig. 3. The mapping process of two parallel single-precision floating-point matrix multiplication.

The single-precision floating-point matrix addition (subtraction) uses a similar algorithm and calculation structure with the single-precision floating-point matrix multiplication. The difference is that the sub-matrix B becomes an identity matrix, and the sub-matrix C participating in multiply-accumulate (subtract) operation needs to be imported into the on-chip RAM in advance.

3.2 Four parallel half-precision floating-point matrix computing

In half-precision floating-point matrix multiplication, one PE can complete four floating-point data operations in the same clock cycles. As shown in Algorithm 3, the size of sub-matrix A is $4S_p \times K$, the size of sub-matrix B is unchanged, and the size of sub-matrix C is $4S_p \times S_{t1}$. In the innermost loop, the original multiply-accumulate operation becomes four parallel multiply-accumulate operations.

As shown in Figure 2(c), when we prepare the source operand, four 16-bit half-precision floating-point data are spliced into a 64-bit source operand. For ex-

Algorithm 3. Half-precision floating-point matrix multiplication.

```
Initialize data block C[0:4Sp-1,0:St1-1] to zero;
  for t2 =0 to K-1 do
    Load data block A[0:4Sp-1, t2];
    Load data block B[t2,0:St1-1];
      for p=0 to Sp-1 do
        for q=0 to St1-1 do
          C[4p,q]=C[4p,q]+A[4p,t2]*B[t2,q];
          C[4p+1,q]=C[4p+1,q]+A[4p+1,t2]*B[t2,q];
          C[4p+2,q]=C[4p+2,q]+A[4p+2,t2]*B[t2,q];
          C[4p+3,q]=C[4p+3,q]+A[4p+3,t2]*B[t2,q];
Store data block C[0:4Sp-1,0:St1-1];
```

ample, half-precision floating-point source operands $a'_{41}a'_{31}a'_{21}a'_{11}$ and $b'_{11}b'_{11}b'_{11}b'_{11}$ are operated to obtain four half-precision operation results $c'_{41}c'_{31}c'_{21}c'_{11}$.

Similar to the process shown in Figure 3, when loading a column of data ($4S_p$) from sub-matrix A, we stitch four adjacent half-precision data into a 64-bit source operand and distribute it to the corresponding PE. When loading data row by row from the sub-matrix B, we also stitch four identical half-precision data into a 64-bit source operand and distribute it to each PE. Here, each PE is responsible for calculating four rows of sub-matrix C.

Besides, the half-precision floating-point matrix addition (subtraction) has similarities with the half-precision floating-point matrix multiplication.

3.3 Multi-precision floating-point multiply-accumulate (subtract) functional component

We adjust and optimize a self-designed and high-performance floating-point multiply-accumulate (subtract) functional component [18] to meet the needs of matrix calculation. Figure 4 shows the three precision floating-point data formats that comply with the IEEE 754 standard. According to the rules of data format, the multi-precision functional component can perform single-precision and half-precision floating-point operations by maximizing the logical structure of double-precision floating-point operation.

The multi-precision functional unit adopts a six-stage pipeline design which mainly includes the modules, such as operand preparation, mantissa multiplication, Multiply-add, normalization processing, exception judgment, and result selection.

After reading the operands, the functional unit separates the exponent and mantissa according to the format and performs corresponding operations respectively. Figure 5(a) shows that in half-precision calculation, the input 64-bit source operand is split into four half-precision floating-point format data.

Besides, we use four single-precision multipliers to perform each group of mantissa multiplications in parallel. The hardware overhead can be saved by multiplexing the multipliers. As shown in Fig. 5(b), when performing half-precision

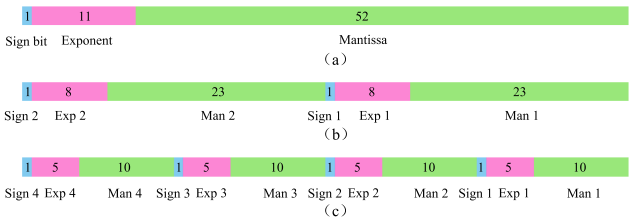

Fig. 4. Different precision floating-point formats used in the unit. (a) Double-precision format. (b) Single-precision format. (c) Half-precision format.

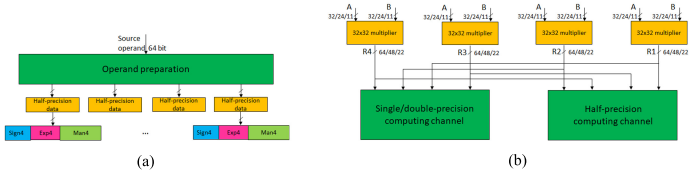

Fig. 5. (a) The process of half-precision data preparation. (b) Multiplexing multiplier in the mantissa multiplication module.

floating-point operations, the separated 4-way multiplication operation data are respectively sent to the upper 11 bits of the four 32*32 bit multipliers.

3.4 Implementation for multi-precision MCA unit core and MCA system

The MCA unit core supports matrices computing of arbitrary size. As shown in Figure 6, the core receives the matrix operation mode signal provided externally and decides whether to load or initialize the sub-matrix C. It can also select the appropriate precision mode according to the actual needs. Then we supply the corresponding operands and process the corresponding operands according to the precision mode.

In the process of implementing the multi-precision MCA unit core, we add logic for selecting data types, data splicing, and parts that cannot be reused, but the logic delay constraints meet the design requirements.

The overall structure of the system is shown in Figure 6. The MCA system uses the architecture of DSP + FPGA. Firstly, the DSP sends instructions and control signals to the coprocessor and transfers the data from the memory on the host side to the DDR on the coprocessor side by starting the SRIO module. Then, the coprocessor calculates autonomously. After the overall calculation is completed, the system starts the SRIO module again and returns the calculation result. Therefore, when the coprocessor is working, the DSP can execute other instructions to achieve independent acceleration.

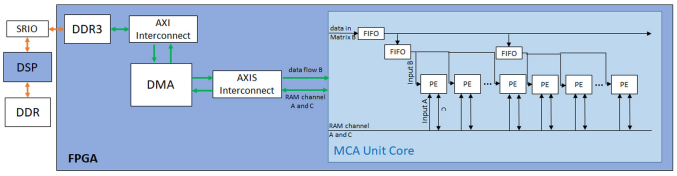

Fig. 6. The architecture of multi-precision MCA unit core and MCA system.

The coprocessor includes an MCA unit core and data transmission logic. The logic mainly includes the DMA module for selecting operation mode, controlling data flow, and transmitting data within the coprocessor. The connection between modules and the core uses the AXI protocol to facilitate communication between each other.

4 Experimental Results and Discussions

The MCA unit core is programmed in Verilog HDL, synthesized, and simulated using synthesis tools (Xilinx Vivado 2016). Then, we deployed the MCA system on the DSP + FPGA platform.

4.1 Synthesis results

We use the 585T development board as the target device and synthesize the MCA unit using 128 PEs under typical conditions. The delay of each stage of the pipeline meets the target delay of 550 ps and the total power consumption is 5.48W.

Different numbers of PEs directly affects the calculation efficiency and hardware overhead in the proposed structure. Table 1 shows the logical resource consumption of the MCA unit with different numbers of PEs. We can see that as the number of PEs increases, LUTs and Slice Registers increase. In addition to the number of PEs, the RAM block consumption is also related to the size of the sub-matrix block. Because the number of PEs determines the number of RAM blocks used, and the size of the sub-matrix block determines the depth of the RAM blocks.

Because the depth of the RAM block in the PE represents the number of times that the data of sub-matrix block A can be reused. To maximize the reuse of data, we can choose the depth of the RAM block according to the size of the sub-matrix B. In principle, the larger the better if the on-chip storage allows. We set the depth of the RAM block to 512 in the project.

For M×N rectangular matrices, due to the idea of block matrix, our storage requirement is only $2S_pS_{t1}$, where the parameters S_p and S_{t1} represent the number of rows and columns of the sub-matrix C, respectively. Therefore, this structure has the characteristic of low storage requirement in large-scale matrix computing.

Table 1. Resource consumption of different numbers of PEs.

PEs	LUTs	Slice Registers	BRAM(18K)
32	70130	58920	41
64	139860	117700	73
128	278960	234450	140

4.2 Performance analysis

Table 2 shows the relationship between the maximum operating frequency and peak performance of the multi-precision MCA unit with the number of PEs. When we set 128 PEs, the maximum operating frequency achieves 180 Mhz. At this time, the multi-precision MCA unit can complete 128 double-precision multiplication operations and 128 addition operations in one cycle. The peak performance of double-precision floating-point matrix computing, for example, can be estimated as $180Mhz \times 128FLOP \times 2 = 46.1GFLOPS$. It can also complete two times single-precision floating-point data and four times half-precision floating-point data at the same time. Therefore, single-precision floating-point and half-precision floating-point performance can reach 92.1 GFLOPS and 184.3 GFLOPS, respectively.

Table 2. Number of PEs, clock speed, and peak performance for double-precision/ single-precision / half-precision floating-point data.

PEs		32	64	128
Clock Speed(Mhz)		202	195	180
Peak performance(GFLOPS)	Double-precision	12.92	24.96	46.1
	Single-precision	25.85	49.92	92.1
	Half-precision	51.71	99.84	184.3

In large-scale matrix computing, we have to consider the time to initialize the system, the time to transfer data from external memory, and the calculation unit waiting for calculation data that may occur during the calculation process.

Firstly, for the time to initialize the system, the main consumption is preloading data. In multi-precision matrix multiplication and matrix addition (subtraction), the amount of preloading data are S_p and $S_p + S_p \times S_{t1}$, respectively. When the size of the matrix becomes larger, the ratio of data preload time to the total time will be small, and the calculation is more efficient.

Secondly, according to the characteristics of the algorithm and structure, we analyze the data transfer time in the calculation process of matrix computing. As shown in Equation (3), we use $\boldsymbol{B}_i^r$ and $\boldsymbol{C}_i^r$ to denote elements containing the ith row of matrix B and matrix C, respectively.

$$\boldsymbol{C}_1^r = a_{1,1}\boldsymbol{B}_1^r + a_{1,2}\boldsymbol{B}_2^r + \ldots + a_{1,k}\boldsymbol{B}_k^r \tag{3}$$

Let one PE calculates a row of elements from left to right. After k iterations, the first PE can complete the calculation of the first row of the matrix C. Then, the i^{th} PE is responsible for the calculation of the ith row of the matrix C. More generally, the extension to all rows of the matrix C is represented by Equation (4).

$$\boldsymbol{C}_i^r = a_{i,1}\boldsymbol{B}_1^r + a_{i,2}\boldsymbol{B}_2^r + \ldots + a_{i,k}\boldsymbol{B}_k^r \tag{4}$$

It can be seen from this formulation that all PEs can work in parallel without affecting each other. We can reuse the data $a_{i,k}$ by reasonably setting the number of elements in a row of matrix B, and at the same time, the latency of floating-point functional component and the data moving time can be hidden into the computing time, which effectively avoids the time of waiting data and ensures the pipeline of the structure.

4.3 Discussion

As shown in Table 3, we compare the proposed multi-precision matrix calculation acceleration unit with related work.

Compared to the double-precision floating-point matrix multiplication structure [13], our calculation structure merges more precisions and more matrix calculation modes. Compared with [17], although our proposed multi-precision MCA unit places fewer PEs, it does not affect the performance of the calculation unit. We can see from Table 3 that when calculating single-precision floating-point data, the performance is close to the structure [17]. When calculating half-precision floating-point data, the performance exceeds the structure [17].

Although the proposed structure based on the circulant matrix [6] has reached a high performance, it is only for fixed-point data. This can not meet the needs of most engineering calculations, such as the nonlinear Kalman filter algorithm and the extended Kalman filter algorithm that require floating-point matrix multiplication and matrix addition. Its size of the matrices is equal to the number of PEs, and it can only handle square matrices. Therefore, its processing capacity is limited. Besides, the computational performance of our structure on a half-precision floating-point (16bit) is better than that of the structure on fixed-point (18bit) [6]. Our preparation time for preloading data is significantly shorter than their preparation time for preloading and generating circulant matrix.

Another matrix computing system [19] with multiple accelerators attempts to set up separate computing arrays for different matrix operations. We can see that our structure is superior to the structure in both performance and energy efficiency.

5 Conclusions

This paper extends the matrix calculation to a multi-precision and multi-operation environment based on the block matrix computing algorithm and linear array.

Table 3. Performance and hardware overhead compared to related work.

	ours	[17]	[13]	[19]	[6]
Supported matrix sizes	$M \times N$	$M \times N$	$M \times N$	$M \times N$	$N \times N$
No. of PEs	128	256	256	N.A.	500
f(Mhz)	180	195	181	150	346
Performance(GFLOPS)	46.1/92.1/184.3	99.8	N.A.	76.8	173
Power(W)	5.48	5.24	N.A.	4.59	N.A.
Energy efficiency(GFLOPS/W)	8.4/16.81/33.63	19.05	N.A.	16.7	N.A.

Through adjusting data flow and reusing logic, the proposed multi-precision floating-point MCA unit can process half-precision, single-precision, and double-precision floating-point data at the same time. Then, we built the MCA system based on the MCA unit core. Compared with the existing matrix calculation structure, our matrix calculation unit can handle three kinds of precision and three modes of operation in a unified structure and features low storage and high efficiency. We plan to improve the arithmetic component to support more precise numerical calculations including fixed-point and extended double-precision in the future.

Acknowledgments This work was partially supported by the National Science and Technology Major Project (2017-V-0014-0066).

References

1. Amira, A., A. Bouridane, and P. Milligan, Accelerating Matrix Product on Reconfigurable Hardware for Signal Processing, in Proceedings of the 11th International Conference on Field-Programmable Logic and Applications. 2001, Springer-Verlag. p. 101–111
2. Bensaali, F., A. Amira, and A. Bouridane, Accelerating matrix product on reconfigurable hardware for image processing applications. Circuits, Devices and Systems, IEE Proceedings, 2005. 152: p. 236-246
3. Liu, Z.Q., et al., Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. Acm Transactions on Reconfigurable Technology And Systems, 2017. 10(3): p. 23
4. Jovanovic, Z. and V. Milutinovic, FPGA accelerator for floating-point matrix multiplication. IET Computers Digital Techniques, 2012. 6(4): p. 249-256
5. Sonawane, D., M.s. Sutaone, and I. Malek, Systolic architecture for integer point matrix multiplication using FPGA. 2009. 3822-3825.
6. Abbaszadeh, A., et al., An Scalable Matrix Computing Unit Architecture for FPGA, and SCUMO User Design Interface. Electronics, 2019. 8(1): p. 20.
7. Qasim, S.M., S.A. Abbasi, and B. Almashary. A proposed FPGA-based parallel architecture for matrix multiplication. in in Proc. IEEE Asia Pacific Conf. Circuits Syst. 2008.
8. Zhou, L.T., et al., Research on Systolic multiplication technology based on FPGA. Computer Engineering and Science, 2015.

9. Ju-Wook Jang, S.B. Choi, and V.K. Prasanna, Energy and time-efficient matrix multiplication on FPGAs. IEEE Transactions on Very Large Scale Integration Systems, 2005. 13(11): p. 1305-1319.
10. Zhuo, L. and V.K. Prasanna, Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on Reconfigurable Computing Systems. IEEE Transactions on Parallel and Distributed Systems, 2007. 18(4): p. 433-448.
11. Kumar, V.B.Y., et al., FPGA Based High Performance Double-Precision Matrix Multiplication. International Journal of Parallel Programming, 2010. 38(3): p. 322-338.
12. Dou, Y., et al., 64-bit floating-point FPGA matrix multiplication, in Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. 2005, Association for Computing Machinery: Monterey, California, USA. p. 86–95.
13. Wu, G.M., Y. Dou, and M. Wang, High performance and memory efficient implementation of matrix multiplication on FPGAs, in 2010 International Conference on Field-Programmable Technology. 2010: Beijing. p. 134–137.
14. Jia, X., G.M. Wu, and X.H. Xie, A High-Performance Accelerator for Floating-Point Matrix Multiplication, in 2017 15th Ieee International Symposium on Parallel and Distributed Processing with Applications. 2017, Ieee: New York. p. 396-402.
15. Qiao, Y.R., et al., FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency. Concurrency And Computation-Practice And Experience, 2017. 29(20): p. 20.
16. Shen, J., et al., Towards a Multi-array Architecture for Accelerating Large-scale Matrix Multiplication on FPGAs. 2018.
17. Zhang, L., et al., A Scalable Architecture for Accelerating Multi-Operation and Continuous Floating-Point Matrix Computing on FPGAs. IEEE Access, 2020. 8: p. 92469-92478.
18. Tian, T., The Research and Implementation of High Performance SIMD Floating-point Multiplication Accumulator Unit for FT-XDSP. National University of Defense Technology, 2013.
19. Wang, W.Q., et al., A Universal FPGA-based Floating-point Matrix Processor for Mobile Systems. Proceedings Of the 2014 International Conference on Field-Programmable Technology, 2014, New York: IEEE. 139-146.