IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 18, 2018, accepted December 13, 2018, date of publication December 24, 2018,
date of current version January 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889558

MT-DMA: A DMA Controller Supporting Efficient Matrix Transposition for Digital Signal Processing

SHENG MA[1,2], YUANWU LEI[2], LIBO HUANG[2], AND ZHIYING WANG[2], (Member, IEEE)
[1]State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China
[2]College of Computer, National University of Defense Technology, Changsha 410073, China

Corresponding author: Yuanwu Lei (yuanwulei@nudt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672526, Grant 61572508, Grant 61872374, and Grant 61472435, in part by the Research Project of NUDT under Grant ZK17-03-06, and in part by the Science and Technology Innovation Project of Hunan Province under Grant 2018RS3083.

ABSTRACT Matrix transposition plays a critical role in digital signal processing. However, the existing matrix transposition implementations have significant limitations. A traditional design uses load and store instructions to accomplish matrix transposition. Depending on the amount of load/store units, this design typically transposes up to one matrix element per clock cycle. More seriously, this design cannot perform matrix transposition and data calculations in parallel. Modern digital signal processors integrate the support for matrix transposition into the direct memory access (DMA) controller; the matrix can be transposed during data movements. It allows the parallel execution of matrix transposition and data calculations. Yet, its bandwidth utilization is limited; it can only transfer one matrix element per clock cycle. To address the limitations of the existing designs, we propose matrix transposition DMA (MT-DMA), to support efficient matrix transposition in DMA controllers. It can transpose multiple matrix elements per clock cycle to improve the bandwidth utilization. Compared with the existing designs, MT-DMA achieves a maximum 23.9 times performance improvement for micro-benchmarks. It is also more energy efficient. Since MT-DMA effectively hides the latency of matrix transposition behind data calculations, it performs very closely to an ideal design for real applications.

INDEX TERMS Digital signal processing, DMA controller, matrix transposition, ping-pong scheme.

I. INTRODUCTION

The performance of numerous important digital signal processing algorithms, including linear algebra routines, audio processing and image processing kernels, depends heavily on the efficiency of matrix transposition. For example, three of the six steps in the most widely used Cooley-Tukey Fast Fourier Transform (FFT) algorithm involve matrix transposition [1]–[3]. Also, matrix transposition constitutes a significant portion in the synthetic-aperture radar (SAR) image processing [4]. Although the permutation pattern of matrix transposition is simple, its poor temporal and spatial locality dramatically improve the difficulty of achieving efficient implementations, especially for large matrices.

A traditional design uses load and store instructions to accomplish matrix transposition. Each element of the source matrix is first loaded from the source address, and then stored in the destination address according to the transpose permutation. The performance of this approach is limited. Depending on the amount of load/store units, it generally transposes at most one element per clock cycle. More seriously, this approach occupies the precious load/store units, and prohibits the simultaneous execution of computation tasks. The matrix transposition and computation tasks can only be executed serially.

To alleviate this issue, several digital signal processors (DSPs) and general purpose processors, including Texas Instruments (TI) TMS320C6678 [5], TMS320C54x [6], TMS320C6000 [7], FT-Matrix [8], Godson-3 [9] and Godson-T [10], integrate the support for matrix transposition into direct memory access (DMA) controllers. Since the DMA controller can work independently of the computation units, this design allows the parallel execution of matrix transposition and computation tasks. Yet, these DMA controllers have a limitation; they can only transfer one matrix element per clock cycle. It is due to that the matrix elements with continuous source addresses are transposed into non-continuous destination addresses. This is acceptable when the bit width of one element is comparable to the bit width of on-chip bus. However, modern processors extensively deploy wide on-chip bus to boost the system throughput and latency. For example, Intel's Xeon [11], Larrabee [12] and Xeon Phi processors [13] all leverage 512-bit width on-chip bus. With such a wide on-chip bus, only transferring one 32-bit or 64-bit element at each clock cycle significantly wastes the bandwidth.

In this paper, we propose a novel design, matrix transposition DMA (MT-DMA), to support efficient matrix transposition in DMA transfers. Unlike previous designs, MT-DMA can transpose multiple matrix elements at each clock cycle; this significantly improves the bandwidth utilization. The key module of MT-DMA is the multi-bank transposition buffer (MBTB). We propose the block matrix transposition using the MBTB. A large matrix is partitioned into several basic matrices. The transposition of a large matrix is achieved by sequentially transposing all basic matrices. The transposition of a basic matrix is accomplished by conducting row-wise write operations and column-wise read operations in the MBTB. In order to read out all data of the same column in one clock cycle, a row shift write strategy is applied to distribute multiple data of the same column into different banks. To further improve the performance, MT-DMA uses two MBTBs to deploy the ping-pong scheme to overlap write and read operations. When one MBTB performs write operations, the other one performs read operations.

Our evaluation results show that MT-DMA performs best in both micro-benchmarks and real applications. For micro-benchmarks, it achieves at most 23.9 times performance improvement against existing designs. Although MT-DMA consumes more power than existing designs due to its high bandwidth utilization, it is much more energy efficient than existing designs. Existing designs consume up to 7.64 times more energy than MT-DMA. For real applications, MT-DMA can effectively hide the latency of matrix transposition behind data calculations, and achieves a performance very close to an ideal design. Its hardware overhead is minor and acceptable considering the high performance improvement provided by MT-DMA.

In summary, we make the following main contributions:

- Observe that existing designs for matrix transposition have significant limitations due to the waste of on-chip bandwidth.
- Propose an efficient matrix transposition design which can transpose multiple matrix elements per clock cycle to improve the bandwidth utilization.
- Compared with existing designs, the proposed MT-DMA significantly improve the performance and is much more energy efficient.

II. BACKGROUND AND MOTIVATION

In this section, we first describe the system architecture of X-DSP, which is leveraged as an example platform to demonstrate the proposed MT-DMA. Then, we analyze the limitation of existing designs.

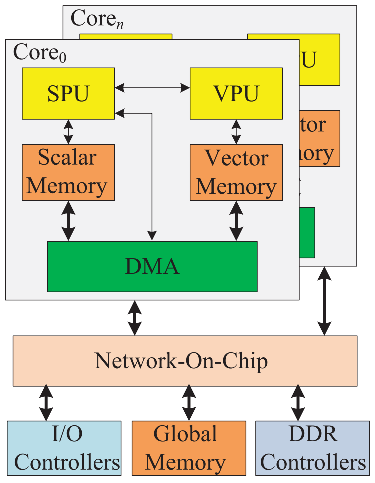


FIGURE 1. The system architecture of X-DSP.

A. SYSTEM ARCHITECTURE

The X-DSP is a multi-core DSP designed for high performance processing of applications for software defined radio (SDR) wireless base station. Fig. 1 shows the system architecture. This architecture has some similarities to other widely used DSPs, including TI's TMS320C6678 [14], AnySP [15] and FT-Matrix [8]. Multiple processing cores, I/O controllers and DDR controllers are connected with a network-on-chip (NoC). We apply a 512-bit wide ring bus to boost the throughput. A 4 MB on-chip global memory (GM) is deployed to be shared by all cores. The GM can be also configured as a memory-side shared cache. The core includes a scalar processing unit (SPU) and a vector processing unit (VPU) to perform the scalar task and vector computation, respectively. It applies a very long instruction word (VLIW) instruction architecture; multiple instructions can be issued to independent execution units at the same time. A 64 KB scalar memory and a 768 KB vector memory are employed for the SPU and VPU, respectively. Each core adopts a DMA controller to move data among different memories, including the scalar memory, vector memory, global memory and off-chip memory. The DMA controller works independent of the SPU and VPU; when the SPU and VPU perform computation tasks, the DMA controller can transfer data simultaneously.

B. MATRIX TRANSPOSITION BY LOAD/STORE UNITS

The most straight forward approach to accomplish the matrix transposition is using load and store instructions.

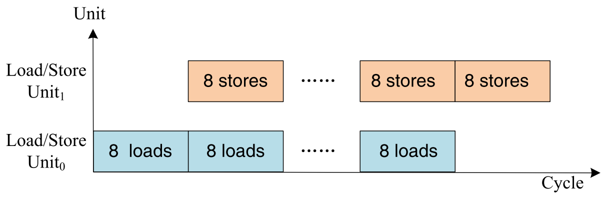


FIGURE 2. The space-time graph of matrix transposition using two load/store units in the VPU.

This approach sequentially loads each matrix element from the source address, and then stores the element into the destination address according to the transpose permutation. The performance of this design depends on the amount of load/store units. Use the X-DSP as an example. The VPU has two load/store units. The latency of load operation is 8 clock cycles, and the latency of store operation is 4 clock cycles. We can use one load/store unit to perform load operations, and use the other unit to perform store operations. As shown in Fig. 2, after 8 clock cycles of the pipeline setup delay, the VPU can transpose one matrix element per clock cycle. In contrast, due to the low performance requirement for scalar computing, the SPU has only one load/store unit; it can only transpose one matrix element every two clock cycles.

A more serious limitation of this approach is that it occupies the load/store units, and prohibits the parallel execution of computation tasks since the computation operand cannot be loaded. For example, before conducting the butterfly operation in FFT computation, two complex numbers are needed to be loaded from the memory. These load operations cannot be executed until the matrix transposition releases the load/store units. Thus, the computation tasks and matrix transposition can be only executed serially.

C. MATRIX TRANSPOSITION BY TRADITIONAL DMA CONTROLLERS

To address the limitation of matrix transposition using load/store units, modern DSPs integrate the support for matrix transposition into DMA controllers [5]–[7]. The DMA controller accomplishes the matrix transposition during data movements. Since the DMA controller works independently of the computation units, this design allows the parallel execution of matrix transposition and computation tasks. As shown in Fig. 3, the matrix transposition is realized by moving rows of the source matrix into columns of the destination matrix. The data movement is composed of multiple transfers, and each transfer moves one row of the source matrix. Each transfer is generally triggered with one event. Some sophisticated DMA controllers leverage the chaining scheme to allow all transfers to be triggered by only one event [5], [6]. In either case, there are startup and cleanup overheads for each transfer, such as copying the transfer parameters and cleaning the finite-state machine (FSM) status registers.

The main limitation of these designs is the low bandwidth utilization for the on-chip bus. Generally, the matrix is stored in the memory in the row-major order; consecutive elements in the same row are stored in continuous memory addresses, while elements in the same column are separated by large address gaps. For the elements moved by one transfer, although their source addresses are continuous, their destination addresses are non-continuous. Non-continuous destination addresses result in that only one element can be moved into the destination memory per clock cycle. This is acceptable when the width of on-chip bus is similar to the width of one element [6]. However, with a wide on-chip bus, there is significant bandwidth loss. Take the X-DSP as an example. Its on-chip bus width is 512 bits, while a matrix element is generally 32 bits or 64 bits. When only moving one matrix element per clock cycle, the bandwidth utilization of on-chip bus can at most be 6.25% or 12.5% for 32-bit or 64-bit elements, respectively. To address this limitation, we propose a novel design, MT-DMA, to improve the on-chip bandwidth utilization.

III. MATRIX-TRANSPOSITION DMA (MT-DMA)

Here, we delve into the design of the proposed Matrix-Transposition DMA (MT-DMA). We first introduce its overall structure. Then, we focus on the transposition of basic matrices. Finally, we describe the transposition of large matrices.

A. OVERALL STRUCTURE

The MT-DMA controller is modified based on a traditional DMA controller used in current DSPs [5]–[8]. Fig. 4 shows its structure. The DMA controller receives the transfer parameters from the SPU, and stores them into the parameter RAM.

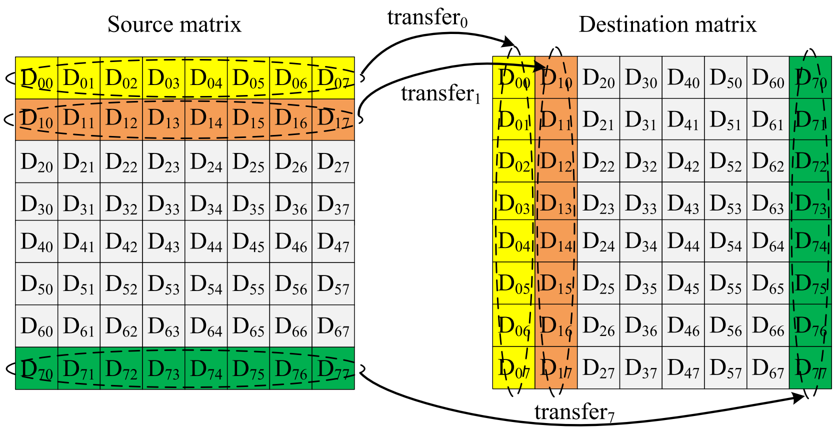


FIGURE 3. The matrix transposition using multiple DMA transfers. We use an 8 × 8 matrix as an example.

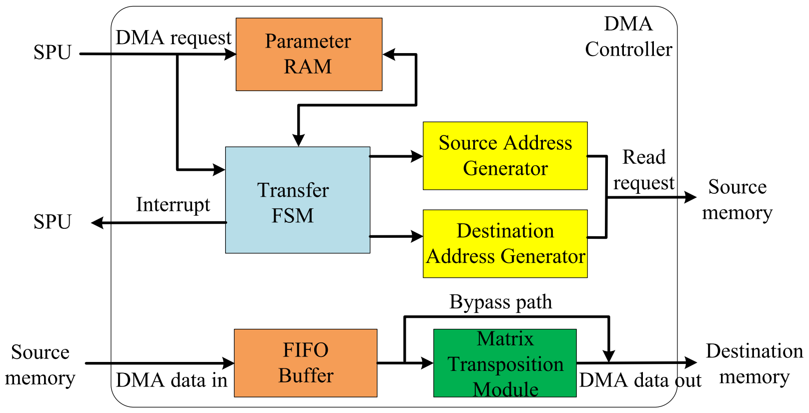


FIGURE 4. The structure of the Matrix-Transposition DMA (MT-DMA).

These parameters generally include the initial source address, the initial destination address, the count of the data array, and other transfer features.

The DMA controller is mainly composed of two parts. The first part sends read requests. It includes the transfer FSM, source address generator, and destination address generator. The transfer FSM controls the overall status. Once a DMA transfer finishes, it sends out an interrupt to the SPU. Based on the configured transfer parameters, the source and destination address generators calculate the current source and destination addresses, which are used to access the source and destination memories, respectively. To allow out-of-order arrival of data replies, the read request carries both the current source address and destination address. The data replies fetches the destination address from its corresponding read request. The second part of the DMA controller processes data replies. It configures a first-in, first-out (FIFO) buffer to synchronize the transfer between the source and destination memories. The received data reply is first stored into the FIFO buffer, and then sent out to the destination memory.

To support the matrix transposition, as shown in Fig. 4, an additional matrix transposition module (MT module) is added. For regular transfers that do not require matrix transpose operations, the MT module is bypassed. For matrix transposition transfers, the data out of the FIFO buffer is sent into the MT module to accomplish the matrix transposition. We later describe the matrix transposition procedure.

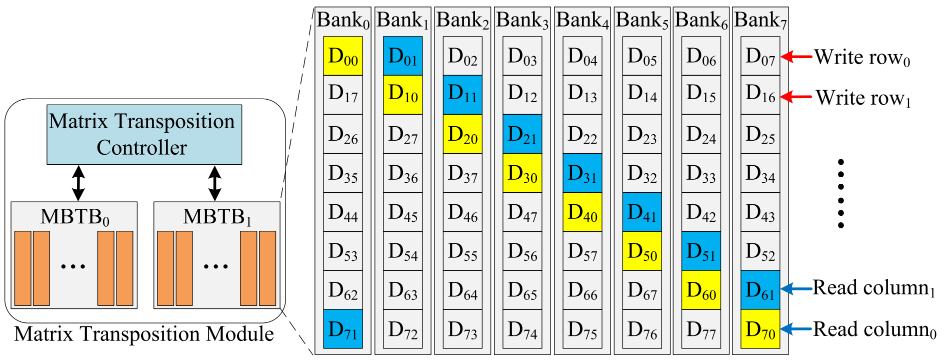
B. THE TRANSPOSITION OF BASIC MATRICES

We use the block matrix transposition algorithm. The cornerstone of this algorithm is the transposition of basic matrices. Here, we focus on this issue.

C. THE TRANSPOSITION OF A BASIC MATRIX

The transposition of a basic matrix is accomplished by the MT module. Fig. 5 shows its structure, which mainly consists of the matrix transposition controller and two multi-bank transposition buffers (MBTBs). The capacity of an MBTB is decided based on the tradeoff between the performance and hardware overheads. We configure the MBTB with 8 banks, and each bank has 8 slots. Each slot is 64 bit wide. As shown in Section IV, this capacity is enough to offer high performance, while only induces low hardware overheads. The MBTB supports multiple transposition granularities; it allows the matrix element to be 32 bits, 64 bits or 128 bits.

Here, we use 64-bit elements as a study case. This element size is widely used in the FFT computation for complex numbers. Each complex number contains a real part and an



Matrix Transposition Controller
MBTB0
MBTB1
...
Matrix Transposition Module
Bank0 Bank1 Bank2 Bank3 Bank4 Bank5 Bank6 Bank7
D00 D01 D02 D03 D04 D05 D06 D07
D17 D10 D11 D12 D13 D14 D15 D16
D26 D27 D20 D21 D22 D23 D24 D25
D35 D36 D37 D30 D31 D32 D33 D34
D44 D45 D46 D47 D40 D41 D42 D43
D53 D54 D55 D56 D57 D50 D51 D52
D62 D63 D64 D65 D66 D67 D60 D61
D71 D72 D73 D74 D75 D76 D77 D70
Write row0
Write row1
Read column1
Read column0

FIGURE 5. The structure of the MT module.

imaginary part, and both are 32-bit single-precision floating-point numbers. With 64-bit elements, the size of a basic matrix is 8×8. As shown in Fig. 5, a basic matrix is transposed with two steps: row-wise write operations and column-wise read operations.

1) Step 1: Row-wise write operations. All rows of a basic matrix are sequentially written into the MBTB. As shown in Fig. 5, the 8 elements of the same row are distributed to different banks; there is no bank conflict. Therefore, these elements can be written into the MBTB in one clock cycle. For example, the write operations of Row_0 and Row_1 only takes two clock cycles.
2) Step 2: Column-wise read operations. All columns of a basic matrix are sequentially read out of the MBTB. To read out elements of the same column in one clock cycle, a row shift write strategy is applied in Step 1 to distribute elements of the same column into different banks; the starting elements of adjacent rows are written into neighboring banks. For example, the starting element of Row_0 (D_{00}) is written into $bank_0$, and the starting element of Row_1 (D_{10}) is written into $bank_1$. This eliminates the bank conflict when reading elements of the same column out of the MBTB.

After the elements of the same column are read out by the column-wise read operations, they are sent into the destination memory to compose a row of the destination matrix. Therefore, the columns of the source matrix are transposed into rows of the destination matrix; this accomplishes the matrix transposition for a basic matrix.

D. PING-PONG SCHEME

To further improve the performance, we apply the ping-pong scheme to overlap the write operations and read operations of MBTBs. Fig. 6 shows the execution procedure. Two MBTBs are leveraged. At first, the received data replies are written into $MBTB_0$ with row-wise write operations. After all rows are written, $MBTB_0$ begins to perform column-wise read operations to read out all columns. At the same time, the newly received data replies are written into $MBTB_1$. The remaining process is similar; when one MBTB performs write operations, the other MBTB performs read operations.

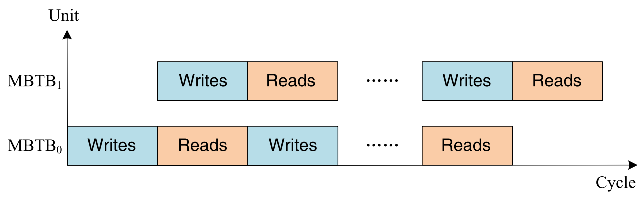


FIGURE 6. The space-time graph of ping-pong scheme.

The ping-pong scheme is managed with two finite state machines (FSMs) implemented in the matrix transposition controller; one FSM is deployed for each MBTB. Fig. 7 shows their state transition diagrams. The state transition

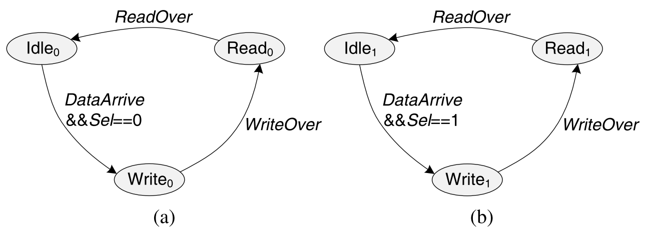


FIGURE 7. The state transition diagram of MBTBs. (a) MBTB0. (b) MBTB1.

diagrams of the two MBTBs are similar. Both have three states. The *Idle* is the initial state. Once a data reply arrives (the *DataArrive* signal is valid), the MBTB goes into the *Write* state to perform row-wise write operations. The *Sel* signal selects the MBTB for arrived data replies. If the *Sel* signal is '0', the arrived data reply is written into $MBTB_0$. Otherwise, the arrived data reply is written into $MBTB_1$. After all rows of a basic matrix are written into an MBTB (the *WriteOver* signal is valid), the current MBTB goes into the *Read* state, and the *Sel* signal flips to select the other MBTB for arrived data replies. The *Read* state performs the column-wise read operations. Once all columns of a basic matrix are read out, the MBTB goes back into the *Idle* state.

E. OUT-OF-ORDER ARRIVAL OF DATA REPLIES

Depended on the characteristics of the memory, there are two types of arrival order for replies: in-order arrival and out-of-order arrival. With in-order arrival, the replies arrive in the same order that the requests were sent out. In this case, the above-mentioned ping-pong buffer scheme can work at a full speed: it can receive a row of a basic matrix at each clock cycle, and send out a column of a basic matrix at each clock cycle. This makes full utilization of the on-chip bandwidth.

When the replies arrive out-of-order, there is a subtle but critical issue. The reply of a later request may arrives earlier, and blocks the write operations for replies of earlier requests, and finally causes the system to enter a deadlock. For example, assume Row_8 arrives earlier than Row_0. Since the depth of each MBTB is 8, Row_0 will be written into $MBTB_0$, and Row_8 will be written into $MBTB_1$. According to the management of ping-pong buffer scheme, only after all expected rows are written into $MBTB_0$, the *Sel* signal flips to allow newly received rows to be written into $MBTB_1$. Since Row_0 is blocked by Row_8, $MBTB_0$ does not received all expected rows, and the *Sel* signal maintains as '0'. Therefore, Row_8 cannot be written into $MBTB_1$, and continues to block Row_0. There is a cyclic dependency. The write operation of Row_8 depends on the write operation of Row_0, while Row_0 is blocked by Row_8; the system enters a deadlock.

Modifying the FSMs of ping-pong scheme cannot address the deadlock issue. Instead, we put some restrictions on the request sending to avoid the deadlock. Once requests for all rows of an MBTB have been sent out, we stop sending requests until all replies corresponding to previous requests arrive. Then, the requests for rows of the other MBTB can be sent out. This restriction guarantees that the replies of the other MBTB will not block the replies of the current MBTB; this avoids the deadlock.

The restriction may degrade the performance. Two factors mitigate the negative effect. First, not all memories in a system are out-of-order memories. The monolithic memories are generally in-order memories. For example, in the X-DSP, although the distributed global memory is an out-of-order memory, the scalar memory and the vector memory are in-order memories. Performing matrix transposition on these in-order memories can make full utilization of the on-chip bandwidth. Second, the proposed MT module can transpose all elements of a column at each clock cycle; this improves the bandwidth utilization. As shown in Section IV, our current design meets the performance requirements of several important applications, including FFT calculation and SAR image processing. For other applications which need much higher performance, we can improve the capacity of MBTB to mitigate the negative effect of the restriction on request sending.

F. SUPPORTING MULTIPLE TRANSPOSITION GRANULARITIES

In addition to the 64-bit elements, the MT module allows the matrix element to be 32 bits and 128 bits wide as well. The transposition with these two element sizes is similar the case of 64-bit elements. The difference is the size of the basic matrix. For 32-bit elements, the size of a basic matrix is 8×16. For 128-bit elements, the size of a basic matrix is 8×4.

We first analyze the case of 32-bit elements. The total width of 16 elements in the same row of a 8×16 basic matrix is 512 bits; these elements can be written into the MBTB in one clock cycle. Every two adjacent elements are located to a different bank. Similar to the case of 64-bit elements, a row shift write strategy is applied during the row-wise write operations. It guarantees that the 8 elements of the same column can be read out in the same clock cycle.

Then, we discuss the case of 128-bit elements. The total width of 4 elements in the same row of a 8×4 basic matrix is also 512 bits; they can also be written into the MBTB in one clock cycle. Every element occupies two adjacent banks. The row shift write strategy distributes every 4 consecutive elements in the same column to different banks. These elements can be read out in the same clock cycle. Since the total width of 4 elements is 512 bits, reading out these elements per clock cycle makes full utilization of the on-chip bandwidth. Reading out the total 8 elements of a column takes two clock cycles.

G. THE TRANSPOSITION OF LARGE MATRICES

The transposition of large matrices is fulfilled by the block matrix transposition algorithm, which consists of three steps. First, the large matrix is partitioned into several basic matrices. Assume the size of a basic matrix is $BN_1 \times BN_2$ (For example, $BN_1 \times BN_2$ is 8×8 for 64-bit elements). An $N_1 \times N_2$ matrix is partitioned into $BR \times BC$ basic matrices, where

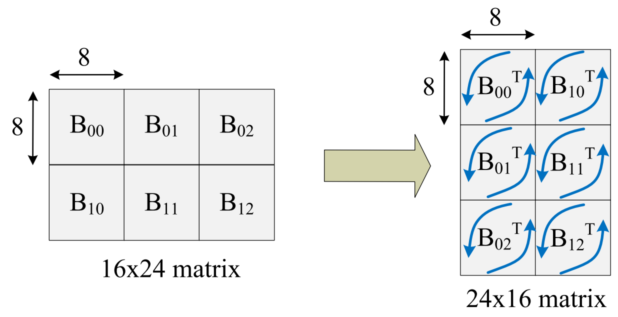


FIGURE 8. **The transposition of a 16 × 24 matrix containing 2 × 3 basic matrices.**

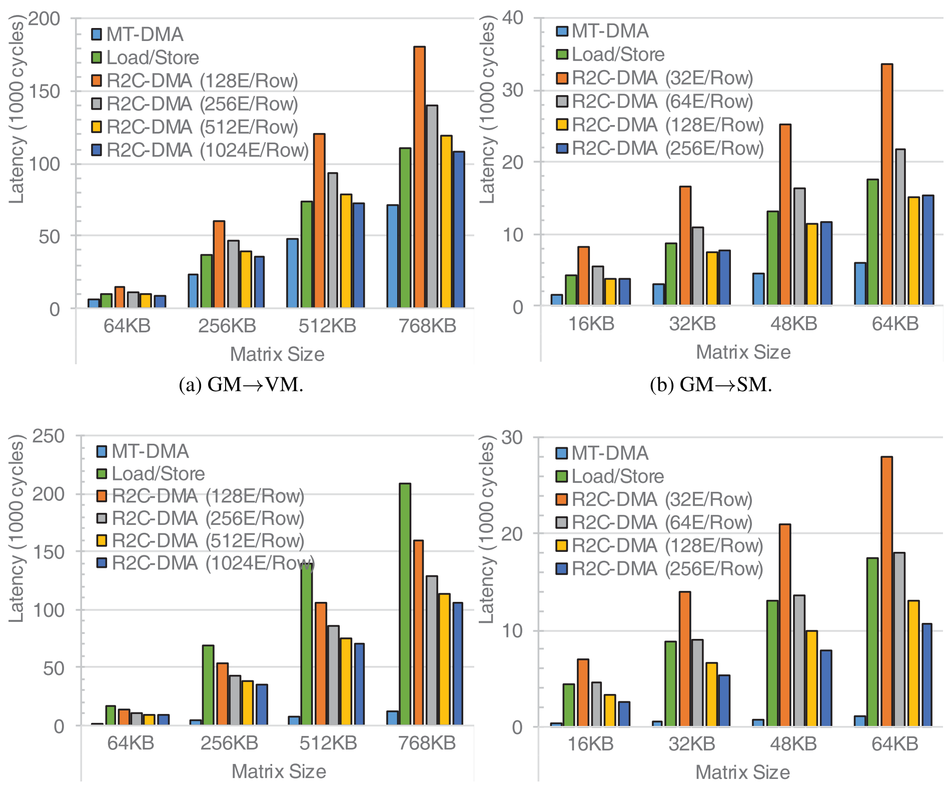
$BR = N_1/BN_1$ and $BC = N_2/BN_2$. Second, each basic matrix is viewed as an element of a $BR \times BC$ matrix. And this $BR \times BC$ matrix is transposed. This transposition only needs to calculate the starting destination address for each basic matrix. The basic matrix at the position (i, j) is transposed into the position (j, i). And its starting destination address is calculated as $initDest + j \times BN_2 \times N_1 \times 8 + i \times BN_1 \times 8$, where $initDest$ is the initial destination address of the transposed matrix. Third, each basic matrix is transposed with the MT module described in Section III-B by setting the starting destination address of this basic matrix as the one calculated in the second step. Fig. 8 shows the transposition procedure of a 16×24 matrix. This matrix is partitioned into 2×3 basic matrices. The 2×3 matrix is transposed by viewing each basic matrix as an element. Finally, each basic matrix is transposed.

IV. EVALUATION

A. METHODOLOGY

We implement the proposed MT-DMA with synthesizable RTL Verilog and integrate it into the X-DSP. We conduct the evaluation with the Cadence NC-Verilog simulator on the RTL simulation environment of the X-DSP. We compare the MT-DMA with two other designs. The first one uses load and store instructions to transpose a matrix (See Section II-B). For brevity, we label it as 'load/store' in this evaluation section. The second design accomplishes matrix transposition with traditional DMA controllers (See Section II-C). It consists of multiple DMA transfers, and each transfer moves a row of the source matrix into a column of the destination matrix. We label this design as 'R2C-DMA'. We evaluate these designs with both micro-benchmarks and real applications.

The micro-benchmarks perform matrix transposition between the memory inside the core and the memory outside the core. The memory inside the core includes the scalar memory (SM) and the vector memory (VM). The memory outside the core is the global memory (GM). The load/store design first moves the data from the source memory into the destination memory with a regular DMA transfer, and then uses load and store instructions to perform matrix transposition in the destination memory. Both the R2C-DMA design and MT-DMA design fulfill the matrix transposition during data movements. The real applications include two important



(a) GM→VM. (b) GM→SM.

(c) VM→GM. (d) SM→GM.

FIGURE 9. **The performance of matrix transposition. (a) GM→VM. (b) GM→SM. (c) VM→GM. (d) SM→GM.**

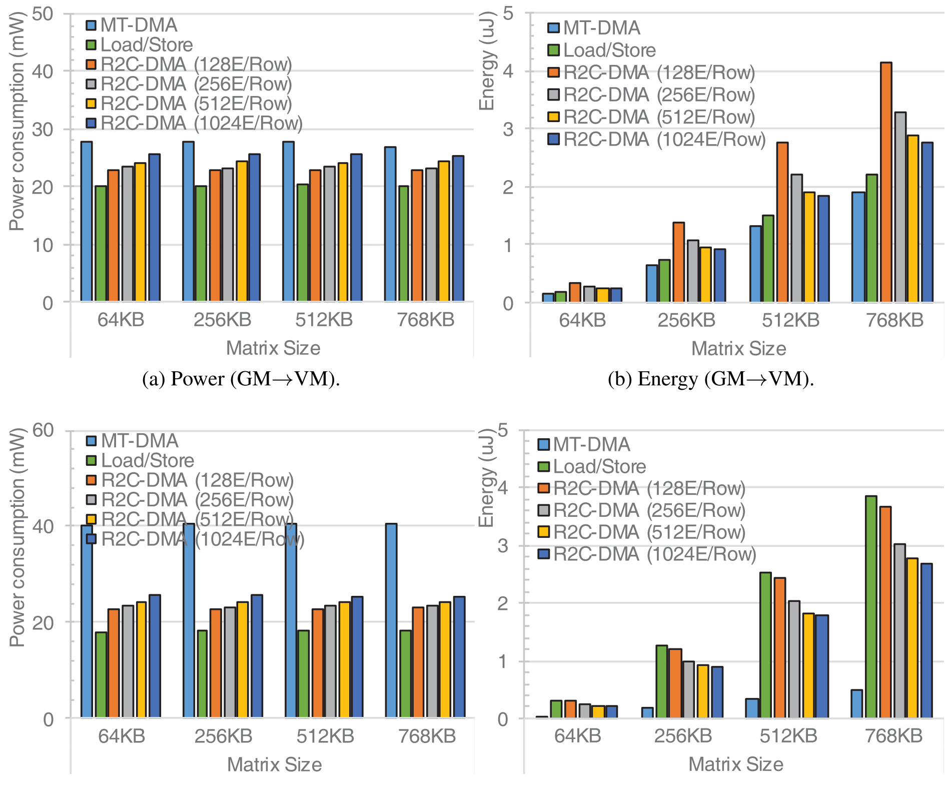
kernels for digital signal processing: the FFT calculation and the SAR image processing. We evaluate both applications with several workload sizes.

To evaluate the hardware overhead, we synthesize the designs by Synopsys Design Compiler with a commercial 40 nm standard cell library. After conducting the back-end physical design, the proposed MT-DMA can work at a 1 GHz frequency. The processor can also work at a 1 GHz frequency. The power consumption is estimated with the Cadence Encounter tool based on the value change dump (VCD) file under the typical corner (25 °C, 0.9 V).

B. THE PERFORMANCE OF MICRO-BENCHMARKS

Fig. 9 shows the latency of matrix transposition for micro-benchmarks. The latency is measured as the time interval between the start of a DMA transfer and the receipt of the last reply. We evaluate matrix transposition for four directions, including transposing the matrix from the GM to the VM (‘GM→VM’ in Fig. 9a), from the GM to the SM (‘GM→SM’ in Fig. 9b), from the VM to the GM (‘VM→GM’ in Fig. 9c), and from the SM to the GM (‘SM→GM’ in Fig. 9d). For each direction, we experiment several matrix sizes. The capacity of the VM is 768 KB in the X-DSP; the evaluated matrix sizes for the directions ‘GM→VM’ and ‘VM→GM’ are 64 KB, 256 KB, 512 KB and 768 KB. Similarly, the evaluated matrix sizes for the directions ‘GM→SM’ and ‘SM→GM’ include 16 KB, 32 KB, 48 KB and 64 KB, since the capacity of SM is 64 KB. The R2C-DMA induces one transfer for each row; for the same matrix size, the performance of R2C-DMA varies with the row sizes. We evaluate the R2C-DMA with several different row sizes. For example, ‘128E/Row’ and ‘256E/Row’ in Fig. 9a mean that each row have 128 matrix elements and 256 matrix elements, respectively.

The proposed MT-DMA achieves the lowest latencies and performs best in all evaluations. Its performance advantage comes from the high bandwidth utilization. MT-DMA can transpose multiple matrix elements per clock cycle, while the load/store and R2C-DMA designs can only transpose up to one matrix element per clock cycle. MT-DMA’s performance is 1.4-5.0 times of the other two designs when transposing the matrix from the GM (‘GM→VM’ and ‘GM→SM’). By comparison, MT-DMA’s performance is 8.9-23.9 times of the other two designs when transposing the matrix from the VM and SM (‘VM→GM’ and ‘SM→GM’). This performance gain difference is due to the restriction on request sending. The data replies from the GM arrive out-of-order; to avoid deadlock, the MT-DMA stops sending new requests until all replies to the current MBTB arrive (See Section III-B). This restriction limits the performance of MT-DMA when transposing the matrix from the GM. Yet, the data replies from the VM and SM arrive in-order. There is no restriction



(a) Power (GM→VM). (b) Energy (GM→VM).

(c) Power (VM→GM). (d) Energy (VM→GM).

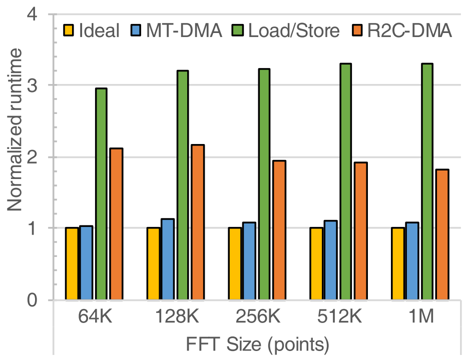
FIGURE 10. The power and energy of matrix transposition. (a) Power (GM→VM). (b) Energy (GM→VM). (c) Power (VM→GM). (d) Energy (VM→GM).

on the request sending, and the MT-DMA can make full utilization of the on-chip bandwidth. The MT-DMA takes 12481 cycles when transposing a 768 KB matrix from the VM; the achieved bandwidth is 504 Gbps since the MT-DMA works at a 1 GHz frequency. This bandwidth is very close to the theoretical maximum bandwidth of 512 Gbps. The bit width of the bus is 512 bits, so the theoretical maximum bandwidth is 512 Gbps. The minor bandwidth loss comes from the startup and cleanup overhead for DMA transfers.

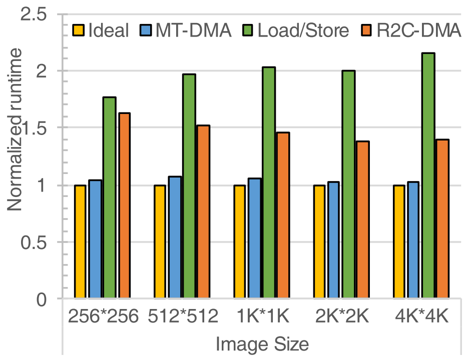
The R2C-DMA performs better with more matrix elements per row. For the same matrix size, the more elements per row, the less the row count. The R2C-DMA uses one transfer for each row, and there are some startup and cleanup overheads for each transfer. Therefore, for the same matrix size, the latency of R2C-DMA decreases with more elements per row. The load/store design is superior to the R2C-DMA when transposing the matrix from the GM to the VM ('GM→VM'). The VPU has two load/store units, and can transpose one matrix element per clock cycle from the VM (See Section II-B). The R2C-DMA can transpose one matrix element per clock cycle as well. Yet, the startup and cleanup overhead for the transfers of R2C-DMA deteriorate its performance. For the other three matrix transposition directions, the performance of load/store is poorer than the R2C-DMA, except for the cases where the R2C-DMA has very few matrix elements per row. For these matrix transposition directions, the load/store operations are performed by the SPU, which has only one load/store unit. It can only transpose one matrix element every two clock cycles. This significantly limits the performance of load/store design.

C. POWER AND ENERGY REQUIREMENTS

We evaluate the power and energy consumption of matrix transposition. Fig. 10 shows the results for two matrix transposition directions: 'GM→VM' and 'VM→GM'. The trends for the other two directions are similar; they are omitted for brevity. The MT-DMA consumes the most power for all evaluations since it makes the highest utilization of on-chip bandwidth. Its power consumption of the 'VM→GM' direction is higher than the 'GM→VM' direction, since the bandwidth utilization of the 'GM→VM' direction is limited by the restriction on request sending. Although the load/store design provides higher performance than the R2C-DMA design for some cases ('GM→VM' in Fig. 9a), it consumes the least power in all evaluations due to the simple structure of the load/store units. The energy consumption is the product of the power consumption and latency. It highlights the advantages of MT-DMA since MT-DMA optimizes the latency. MT-DMA consumes the least energy as its latency is significantly lower than other designs. On average, the energy consumption of the load/store design is 1.15 times and 7.64 times of MT-DMA for the 'GM→VM' and 'VM→GM' directions, respectively. The energy



(a) FFT calculation.



(b) SAR image processing.

FIGURE 11. The performance of real applications. (a) FFT calculation. (b) SAR image processing.

consumption of R2C-DMA is 1.46 times and 5.6 times of MT-DMA for these two directions.

D. PERFORMANCE IN APPLICATIONS

We evaluate the performance of the designs with two important digital signal processing applications: the FFT calculation and the SAR image processing. The FFT calculation leverages the widely used Cooley-Tukey algorithm. This algorithm consists of six steps, and three of them involve matrix transposition, including matrix transposition before column FFT calculation, matrix transposition after the column FFT calculation, and matrix transposition after the row FFT calculation. In the X-DSP, the data set is initially located in the GM, and all calculations are performed in the VM using vector processing units. Therefore, the matrix transposition direction before the column FFT calculation is ‘GM→VM’, and the matrix transposition direction after the column FFT and row FFT calculation is ‘VM→GM’. The SAR image processing applies the chirp scaling algorithm, which includes four batches of FFT calculation: azimuth FFT, range FFT, range IFFT, and azimuth IFFT. All these FFT calculations apply the Cooley-Tukey algorithm. We evaluation both applications with several workload sizes. The sizes for FFT calculation include 64K, 128K, 256K, 512K and 1M points. The image sizes for SAR processing include 256×256, 512×512, $1K \times 1K$, $2K \times 2K$, and $4K \times 4K$.

Since the DMA controller works independently of the computation units, we divide the VM into two parts to apply the ping-pong scheme for both the MT-DMA and R2C-DMA. When one part performs matrix transposition, the other one performs data calculations. This can overlap matrix transposition and data calculations. We also evaluate an ideal design, assuming that it can completely hide the latency of matrix transposition behind data calculations.

Fig. 11 shows the runtime normalized to the ideal design for the applications. The load/store design performs poorest for all evaluations, since it only allows serial execution of matrix transposition and data calculations, and totally exposes the latency of matrix transposition. In contrast, the R2C-DMA and MT-DMA designs can hide some of the latency of matrix transposition behind the calculation. For both the FFT calculation and SAR image processing, R2C-DMA performs better with larger workload sizes. Larger workload sizes can more efficiently amortize the startup and cleanup overheads for R2C-DMA. The load/store design performs worse with large workload sizes. Larger workload sizes increases the latency of matrix transposition, and reduces the performance of load/store design.

The MT-DMA’s performance advantage is more significant in the FFT calculation than the SAR image processing. It average performance gains against R2C-DMA are 85.2% and 43.4% for the FFT calculation and SAR image processing, respectively. Similarly, its average performance gains against the load/store design are 215.2% and 94.4% for these two applications. The matrix transposition plays a more important role in the FFT calculation, since the SAR image processing has several factor compensation steps that does not involve matrix transposition. For both applications, MT-DMA performs very closely to the ideal design. On average, its performance is only 5.2% and 3.8% lower than the ideal design for the FFT calculation and SAR image processing, respectively. This demonstrates the efficiency of MT-DMA.

E. HARDWARE OVERHEAD

Here, we discuss the hardware overhead of MT-DMA. First, we consider the implementation of MBTB. We evaluate two choices. One is implemented by registers synthesized from the RTL Verilog description, and the other one is implemented with SRAMs generated by the Synopsys Embed-It! Integrator. Since the capacity of a MBTB is small, the register implementation is more area efficient than the SRAM implementation. The area of the former is 30964 μm^2, and the area of the latter is 65472 μm^2. Therefore, we implement the MBTB with registers. Then, we synthesize a regular DMA controller and an MT-DMA controller. The area of a regular DMA controller is 976806 μm^2, and the area of an MT-DMA controller is 1064718 μm^2. Therefore, the area of the additional hardware for supporting the MT-DMA is 87912 μm^2, which is 9.1% of the area of a regular DMA controller. The majority of the additional area is induced by the two MBTBs; they occupy 63.2% of the additional area, and the remaining area is the control logics. Considering the performance improved by the MT-DMA, these minor hardware overheads are acceptable.

V. RELATED WORK

The matrix transposition plays a critical role in several important fields, including digital signal processing and scientific computing. There are numerous work on the efficient implementation of matrix transposition on different computing platforms. Chatterjee and Sen propose cache efficient matrix transposition for general purpose processors [16], and Ruetsch and Micikevicius research high performance matrix transposition for GPGPUs [17]. The matrix transposition on both the general purpose processors and GPGPUs are implemented with load and store instructions. They do not allow the parallel execution of matrix transposition and calculations. To address this issue, modern DSPs, such as TI TMS320C6678 [5], TMS320C54x [6], TMS320C6000 [7], FT-Matrix [8], integrate the support of matrix transposition into the DMA controller; a matrix is transposed during the data movement. This approach allows the parallel execution of matrix transposition and calculation. Several work leverage this feature to achieve high performance for several important workloads, including SAR image processing [18], signal processing kernels [19] and BLAS routines [20]. A limitation of these DMA controllers [5]–[10], [21] is that they can only transfer one matrix element per clock cycle, which significantly degrades the bandwidth utilization. In contrast, our proposed MT-DMA can transpose multiple matrix elements per clock cycle to improve the bandwidth utilization.

VI. CONCLUSION

This paper delves into the design of efficient matrix transposition for digital signal processors. By leveraging the MBTB, the proposed MT-DMA can transpose multiple matrix elements per clock cycle to improve the bandwidth utilization. We also apply the ping-pong scheme to overlap write and read operations for the MBTB to further improve the performance. The evaluation results show that MT-DMA performs much better than existing designs. Meanwhile, it is more energy efficient. MT-DMA can efficiently hide the latency of matrix transposition behind data calculations, and performs very closely to an ideal design for real applications. Our future work include performing further optimizations of the implementation of MT-DMA, such as conducting physical custom designs for the MBTB. We will also work on integrating support for other critical kernels of digital signal processing, such as matrix multiplication, in DMA controllers.

REFERENCES

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[2] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform.* Philadelphia, PA, USA: SIAM, 1992.

[3] B. Spinean and G. Gaydadjiev, "Implementation study of FFT on multilane vector processors," in *Proc. 15th Euromicro Conf. Digit. Syst. Design*, Sep. 2012, pp. 815–822.

[4] R. K. Raney, H. Runge, R. Bamler, I. G. Cumming, and F. H. Wong, "Precision SAR processing using chirp scaling," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 4, pp. 786–799, Jul. 1994.

[5] "KeyStone architecture enhanced direct memory access (EDMA3) controller: User's guide," Texas Instrum. Corp., Dallas, TX, USA, Tech. Rep. SPRUGS5B, May 2015. [Online]. Available: http://www.ti.com/lit/ug/sprugs5b/sprugs5b.pdf

[6] "TMS320C54x DSP reference set volume 5: Enhanced peripherals," Texas Instrum. Corp., Dallas, TX, USA, Tech. Rep. SPRU302B, Mar. 2007. [Online]. Available: http://www.ti.com.cn/cn/lit/ug/spru302b/spru302b.pdf

[7] "TMS320C6000 DSP enhanced direct memory access (EDMA) controller: Reference guide," Texas Instrum. Corp., Dallas, TX, USA, Tech. Rep. SPRU234C, Nov. 2006. [Online]. Available: http://www.ti.com/lit/ug/spru234c/spru234c.pdf

[8] S. Chen *et al.*, "FT-matrix: A coordination-aware architecture for signal processing," *IEEE Micro*, vol. 34, no. 6, pp. 64–73, Nov./Dec. 2014.

[9] W. Hu, J. Wang, X. Gao, Y. Chen, Q. Liu, and G. Li, "Godson-3: A scalable multicore RISC processor with x86 emulation," *IEEE Micro*, vol. 29, no. 2, pp. 17–29, Mar./Apr. 2009.

[10] D.-R. Fan *et al.*, "Godson-T: An efficient many-core architecture for parallel program executions," *J. Comput. Sci. Technol.*, vol. 24, no. 6, p. 1061, Nov. 2009.

[11] C. Park *et al.*, "A 1.2 TB/s on-chip ring interconnect for 45 nm 8-core enterprise Xeon processor," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2010, pp. 180–181.

[12] L. Seiler *et al.*, "Larrabee: A many-core x86 architecture for visual computing," *IEEE Micro*, vol. 29, no. 1, pp. 10–21, Jan./Feb. 2009.

[13] Intel Corporation, "Intel Xeon Phi coprocessor system software developers guide," Intel Corporation, Santa Clara, CA, USA, Tech. Rep. SKU# 328207-003EN, Mar. 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system-software-developers-guide.pdf

[14] Texas Instruments Corporation, "TMS320C6678 multicore fixed and floating-point digital signal processor," Texas Instrum., Intel Corp., Dallas, TX, USA, Tech. Rep. SPRS691E, Mar. 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf

[15] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime anywhere anyway signal processing," *IEEE Micro*, vol. 30, no. 1, pp. 81–91, Jan./Feb. 2010.

[16] S. Chatterjee and S. Sen, "Cache-efficient matrix transposition," in *Proc. 6th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Jan. 2000, pp. 195–205.

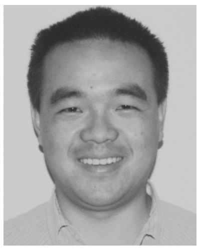
[17] G. Ruetsch and P. Micikevicius, "Optimizing matrix transpose in CUDA," NVIDIA, Santa Clara, CA, USA, Tech. Rep., Mar. 2009. [Online]. Available: http://developer.download.nvidia.com/compute/DevZone/C/html_x64/6_Advanced/transpose/doc/MatrixTranspose.pdf

[18] D. Wang and M. Ali, "Synthetic aperture radar on low power multi-core digital signal processor," in *Proc. IEEE Conf. High Perform. Extreme Comput.*, Sep. 2012, pp. 1–6.

[19] B. Ramesh, A. Bhardwaj, J. Richardson, A. D. George, and H. Lam, "Optimization and evaluation of image- and signal-processing kernels on the TI C6678 multi-core DSP," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2014, pp. 1–6.

[20] M. Ali, E. Stotzer, F. D. Igual, and R. A. van de Geijn, "Level-3 BLAS on the TI C6678 multi-core DSP," in *Proc. IEEE 24th Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2012, pp. 179–186.

[21] G. Jiang, "Design and implementation of a DMA controller for digital signal processor," M.S. thesis, Dept. Syst. Eng., Linköping Univ., Linköping, Sweden, Aug. 2010.



SHENG MA received the B.S. and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), in 2007 and 2012, respectively. He visited the University of Toronto, from 2010 to 2012. From 2012 to 2017, he was an Assistant Professor with the College of Computer, NUDT, where he is currently an Associate Professor with the College of Computer. His research interests include microprocessor architecture, on-chip networks, SIMD architectures, and arithmetic unit designs. He was a recipient of the Young Talent Development Program of the China Computer Federation, in 2016.

YUANWU LEI was born in 1982. He received the B.S. degree in computer science and technology from North China Electric Power University, Baoding, China, in 2005, and the M.S. and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), Changsha, China, in 2007 and 2012, respectively. He is currently an Assistant Researcher with NUDT. His current research interests include computer architecture, microprocessor and digital signal processing design, high-precision computation, parallel computing, and reconfigurable computing.

LIBO HUANG received the B.S. and Ph.D. degrees in computer engineering from the National University of Defense Technology, China, in 2005 and 2010, respectively, where he is currently an Associate Professor with the College of Computer. He has authored over 50 papers in internationally recognized journals and conferences. His research interests include computer architecture, hardware/software co-design, VLSI design, and on-chip communication.

ZHIYING WANG received the Ph.D. degree in electrical engineering from the National University of Defense Technology (NUDT), in 1988 where he is currently a Professor with the School of Computer. He has contributed over 10 invited chapters to book volumes, has published 240 papers in archival journals and refereed conference proceedings, and has delivered over 30 keynotes. His main research fields include computer architecture, computer security, VLSI design, reliable architecture, multi-core memory system, and asynchronous circuit. He is a member of the IEEE and ACM.