PAPER

Window Memory Layout Scheme for Alternate Row-Wise/Column-Wise Matrix Access

Lei GUO[†], **Yuhua TANG**[†], **Yong DOU**[†], ***Nonmembers***, **Yuanwu LEI**[†a)], ***Student Member***, **Meng MA**[†], ***and*** **Jie ZHOU**[†], ***Nonmembers***

SUMMARY The effective bandwidth of the dynamic random-access memory (DRAM) for the alternate row-wise/column-wise matrix access (AR/CMA) mode, which is a basic characteristic in scientific and engineering applications, is very low. Therefore, we propose the window memory layout scheme (WMLS), which is a matrix layout scheme that does not require transposition, for AR/CMA applications. This scheme maps one row of a logical matrix into a rectangular memory window of the DRAM to balance the bandwidth of the row- and column-wise matrix access and to increase the DRAM IO bandwidth. The optimal window configuration is theoretically analyzed to minimize the total number of no-data-visit operations of the DRAM. Different WMLS implementationsare presented according to the memory structure of field-programmable gata array (FPGA), CPU, and GPU platforms. Experimental results show that the proposed WMLS can significantly improve DRAM bandwidth for AR/CMA applications. achieved speedup factors of 1.6× and 2.0× are achieved for the general-purpose CPU and GPU platforms, respectively. For the FPGA platform, the WMLS DRAM controller is custom. The maximum bandwidth for the AR/CMA mode reaches 5.94 GB/s, which is a 73.6% improvement compared with that of the traditional row-wise access mode. Finally, we apply WMLS scheme for Chirp Scaling SAR application, comparing with the traditional access approach, the maximum speedup factors of 4.73X, 1.33X and 1.56X can be achieved for FPGA, CPU and GPU platform, respectively.

key words: *window memory layout scheme (WMLS), alternate row-wise/column-wise matrix access, SDRAM, GPU, FPGA*

1. Introduction

Matrix is one of the most fundamental data representations in scientific and engineering applications, such as Linpack benchmark [1], synthetic aperture radar (SAR) [2] and two-dimension fast Fourier transform (2D FFT) [3]. A larger matrix size requires higher memory bandwidth demand. With the development of the very large scale integration technology, the speed gap between processor and memory is continuously growing. Lower memory bandwidth utilization further exacerbates this so-called "memory wall" [4] phenomenon, which increasingly becomes a bottleneck for matrix applications.

The dynamic random-access memory (DRAM) is a major storage media characterized by high density and low cost. DRAM is widely used in various computing platforms, such as CPU, field-programmable gate array (FPGA) and graphic processing unit (GPU) platforms. From SDRAM to DDR3, the theoretical peak bandwidth has improved significantly with the increase of the transmission frequency and the advent of double data transfer.

However, DRAM bandwidth utilization is quite different for different access modes. DRAM is organized as a large memory cell matrix and adopts a high-speed cache unit as row buffer between DRAM memory cells and processor. DRAM has large IO bandwidth for accessing data in the same physical row. When data in different rows are accessed, the data in the row buffer are written back into the physical memory matrix and then reloaded from a new physical row. This process takes a long time, thus reducing the practical bandwidth. The bandwidth to access a matrix in the row-wise order is one order of magnitude larger than that in column-wise order [5]. Therefore, the DRAM bandwidth utilization for alternate row-wise/column-wise matrix access (AR/CMA) mode is very low.

Many applications characteristically access matrixes under the AR/CMA mode. For example, the row-wise of data matrix represents the position direction in the SAR application, whereas the column-wise of data matrix represents the distance direction. In the process of SAR application, the data matrix calculations are performed both in the position and distance directions. High bandwidth utilization can be achieved in position direction when the matrix is arranged in traditional row-wise order. However, for the distance direction, the data in one column of the matrix are distributed in the different physical rows of DRAM. Therefore, the bandwidth utilization is very low, which is in contrast with the traditional row-wise order.

In this paper, we analyze the DRAM characteristics and the matrix access by using the AR/CMA mode. To match the memory continuity of DRAM and access continuity of the AR/CMA mode, we propose the window memory layout scheme (WMLS), which is a matrix layout scheme that does not require transposition, and identify the optimal window size of WMLS. WMLS reduces the total number of no-data-visit operations in the DRAM and balances the bandwidth of the row- and column-wise matrix access. Finally, we implement the WMLS and CS SAR applications on the basis of the memory structure features in the FPGA, CPU and GPU platforms and achieve significant improvement in the DRAM bandwidth for AR/CMA applications.

The remainder of this paper is organized as follows. Section 2 presents a brief background of AR/CMA and DRAM. Section 3 introduces the procedure of AR/CMA.

Manuscript received December 27, 2012.
Manuscript revised June 12, 2013.

[†]The authors are with the National Laboratory for Parallel and Distribution Processing, National University of Defense Technology, Changsha, 410073, P. R. China.
a) E-mail: yuanwulei@nudt.edu.cn

DOI: 10.1587/transinf.E96.D.2765

Section 4 presents several related works. Section 5 introduces the proposed WMLS and its optimal window size. Finally, Sect. 6 presents the implementation WMLS in FPGA, CPU, and GPU platforms.

2. Background

2.1 Characteristics of Matrix Access

The 2D matrix, which uses two indexes (i, j) to access a given element, is one of the most fundamental data representations. Figure 1 (A) shows that element $a_{i,j}$ in matrix A is identified by row index i and column index j. Matrix A can be regarded as M row vectors of size N or N column vectors of size M.

The successive data in the row or column vector are usually accessed as a pseudo code (sections R and C) in matrix computations, as shown in the Fig. 1 (B). For example, in matrix multiplication [6], each element in the resultant matrix is the inner product of the row and column vectors. The LU and QR decomposition [7] is a process that continuously updates the row and column data. In the Chirp Scaling (CS) SAR application [8], Azimuth FFT (or inverse FFT (IFFT)) performs FFT (or IFFT) operations for the row vector of the data matrix, whereas range FFT (or IFFT)) performs FFT (or IFFT) operations for the column vector of the data matrix.

2.2 DRAM Characteristics

The capacity and performance of DRAM families (e.g., SDRAM, DDR2, and DDR3) and the GDDR dedicated to the GPU platform have significantly improved in recent years. However, the physical architecture and access principle of DRAM and GDDR remain the same. As shown in Fig. 1 (C), DRAM is organized as a memory cell matrix and uses the row-buffer model to access data. Each memory cell stores one-bit of data and comprises one extremely small transistor and a capacitor. Therefore, billions of memory cells can be fitted in a single memory chip and high densities can be achieved. The row-buffer model uses two steps to access a given memory cell: reading the physical row into the row buffer according to the bank and row address and accessing the element according to the column address. Basic DRAM operations can be classified as ACTIVE, PRECHARGE, READ, WRITE, and REFRESH. In the current paper, we define ACTIVE, PRECHARGE, and REFRESH as no-data-visit operations.

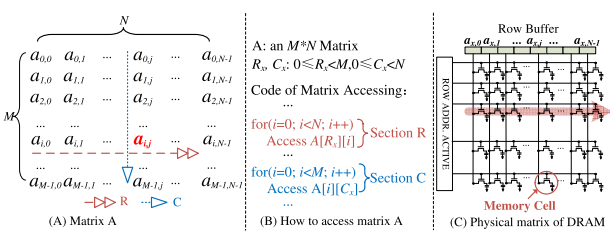


Fig. 1 Concept, access, and storage of a matrix.

DRAM has two key features in its structure and access mode. First, DRAM is internally configured as a multi-bank DRAM with a synchronous interface, where each bank is organized into R rows by C columns. Second, read and write DRAM are burst-oriented. Burst access begins with an ACTIVE command; then, a burst command (READ or WRITE) follows to access the data; finally, the PRECHARGE command is used to deactivate the opened physical row. The overhead of burst access can be attributed to three factors: the delay time between ACTIVE and burst commands (T_{RCD}); the delay between the registration of a READ command and the availability of the first piece of output data (T_{CL}); the minimum time interval between PRECHARGE command and a subsequent row access (T_{RP}). These overheads are specified by manufacturers and similar under different DRAM series. Several burst commands can be executed after one ACTIVE command in the row-buffer model. However, the PRECHARGE and ACTIVE commands should be performed to access two elements in different physical row. Therefore, several optimal methods are proposed to minimize the overheads of PRECHARGE and ACTIVE commands.

DRAM bandwidth utilization is determined by data distribution. When N data are accessed from one physical row (accessing the row vector of a matrix), one ACTIVE, one PRECHARGE and N burst commands are executed. The bandwidth utilization is expressed as

$$U_{row} = \frac{N/(2Freq_{DRAM})}{T_{RCD} + T_{CL} + N/(2Freq_{DRAM})} \quad (1)$$

where $Freq_{DRAM}$ is the frequency of DRAM interface.

When N data are read from N physical rows (accessing the column vector of a matrix), the N ACTIVE, N BURST, and $N-1$ PRECHARGE commands are executed. These physical rows can be distributed in a multi-bank, and the access operations can be executed in parallel. The maximum bandwidth utilization in this case is

$$U_{column} = \frac{B/Freq_{DRAM}}{T_{RCD} + T_{CL} + B/T_{CCD}} \quad (2)$$

where B is the number of banks in DRAM, and T_{CCD} ($T_{CCD} = 4 * T_{CK}$) is the minimal delay between two access commands, and T_{CK} is clock period of DRAM.

Taking the latest DDR3-1600, as example, T_{RCD}=11 cycles, T_{CL}=11 cycles, T_{CK}=3 cycles, and B=8. Given that N=1000, the latency for the row-vector access is 522 cycles, and the utilization is 96%. The latency for the column-vector access is 14750 cycles, and the availability is only 6.8%. Therefore, the DDR3 bandwidth utilization for row-vector access is theoretically 14 times more than that for the column-vector access.

3. Alternate Row-Wise/Column-Wise Matrix Access

3.1 Data Continuity

Data continuity is manifested in two aspects: storage and

access continuity. High bandwidth availability can be obtained only if the data access continuity of a program can be well integrated with its storage continuity.

Data storage continuity, which is a characteristic of a memory structure, implies that memory cells could be accessed successively and quickly. We define a memory cell set R to have a data storage continuity, if it can be read or written successively with high IO performance. If the data in one physical row of a DRAM can be accessed quickly due to the row-buffer and burst-oriented scheme, the the DRAM has data storage continuity in the physical rows.

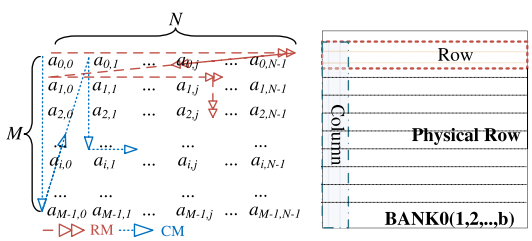
Data access continuity is a memory access characteristic of programs, in which data sets can always be successively accessed in a program. As analyzed in Sect. 2.1, the row or column vector is always accessed continuously in matrix application programs; thus, this program has data access continuity in row and column vector, which is defined as row and column continuity respectively

3.2 Alternate Row-Wise/Column-Wise Matrix Access

AR/CMA is the process of accessing all matrix elements in a row-wise order and then in a column-wise order, or vice versa. As shown in Fig. 2 (A), the double-arrow (RM) and single-arrow (CM), called RM access and CM access, refer to the matrix access with row and column vector, respectively. RM performs program section R in Fig. 1 (B) from R_x=0 to R_x=M-1 to access all elements in the matrix. Similarly, CM performs program section C in Fig. 1 (B) from C_x=0 to C_x=N-1.

AR/CMA is a basic characteristic of many matrix applications. In the SAR application, two AR/CMAs are employed to transform the raw data into a gray image. The 2D FFT also includes an AR/CMA procedure to compute the data in two dimensions.

The significantly different utilization of the memory bandwidth for RM and CM access seriously affects the performance of AR/CMA applications. In the traditional row-wise storage pattern, the matrix row is located in the continuous cell in a physical row, as shown in Fig. 2 (B). The distribution of the row data is parallel with the physical row, so one row exists in as least physical row as possible. The data access continuity of RM access matches well with its storage continuity; thus, the bandwidth utilization for RM access is very high, as shown in Sect. 2.2. Conversely, for the CM access, the column data are distributed in many physical rows, and the distribution direction crosses with the physical rows. In addition, many ACTIVE and PRECHARGE operations are executed. Therefore, the real bandwidth for the CM access is very small.



(A) Row and Column Matrix Access (B) Mapping from Matrix to DRAM

Fig. 2 Traditional row-wise mapping between logical matrix and DRAM.

4. Related Works

Several memory optimization schemes have been proposed to improve bandwidth utilization on the basis of the features of applications and platforms. First, methods for program transformation have been explored to improve the cache hit rate in uniprocessor systems, or to improve the data locality in distributed-memory parallel computers [9], [10]. Gottschling [11] proposed a representation-transparent matrix algorithm for multicore chip and developed matrix template library (MTL) for matrix application, such as matrix multiplication. Ruetsch [12] and Podlozhnyuk [13] combine features of the GPU warp access memory and the shared memory structure to enhance matrix transpose performance. However, their methods require additional memory space and lacks theoretical analysis and guidance on the optimal data layout.

Second, a special Corner Turn memory has been designed to shorten the latency for transposing matrix. However, this method introduced overhead, which consumed a considerable portion of the total running time. Tu [14], and Kim [15] presented a matrix-transpose memory controller in the FPGA platform to eliminate the transpose operation in the calculation process. However, their methods require many hardware resources and are only adaptable to the FPGA platform.

Third, special-purpose hardware designs have been proposed to implement AR/CMA applications without data transpose. Mo [16], Onoe [17], and Yu [18] proposed an SAR image system without data transpose. However, the memory access in their methods still span among many physical rows, and the DRAM IO bandwidth utilization for AR/CMA applications has not significantly improve. Their works are only suitable for the FPGA-based platform. The related work in [5] proposed an array-address translation algorithm, that maps a logic window matrix into a physical row of SDRAM. This approach is the reverse of our proposed scheme.

The proposed WMLS is presented in the succeeding sections. The optimal window configuration of the proposed approach is also analyzed theoretically and verified in the FPGA, CPU, and GPU platforms, respectively.

5. Window Memory Layout Scheme (WMLS)

A common data layout scheme called WMLS is presented to improve the bandwidth utilization for AR/CMA applications. This scheme maps one row of logical matrix into a rectangular memory window in the physical banks of DRAM in order to improve the match between the data access continuity of AR/CMA applications and data storage

continuity of DRAM. This approach enables us to access effectively both row and column vectors while eliminating the overhead of matrix transposition.

5.1 Memory Windows of Matrix

We consider an $M \times N$ logical matrix as an example to introduce mapping between logical matrix and DRAM in WMLS. As shown in Fig. 3, DRAM is organized as a rectangular memory space, and the physical rows from different banks are placed in the proper order. For description, we define the following notations:

- D[b][R][C]: physical memory space of DRAM. The number of banks is equal to b.
- LM[M][N]: $M \times N$ logical matrix with size smaller than that of DRAM, i.e. $b \times R \times C > M \times N$.
- $W(r, c)$: $A \times B$ rectangular area (i.e., window) in the physical bank. Here, $A * B * b = N$. DRAM contains M windows, organized as a $T \times S$ window matrix. $W(r, c)$ is the window with row index r and column index c ($0 \le r \le S$, and $0 \le c \le T$).
- $E[x][y]$: element in the window ($0 \le x < A$ and $0 \le y < B$).

The window data layout scheme organizes rows of logical matrix LM into consecutive memory windows in sequence, as shown in Fig. 3.

Step 1: We divide the elements $LM[i][j]$ ($i = 0, 0 \le j \le N - 1$) in the first row of logical matrix into $b \times B$ segments, and place these segments into the first memory windows in b physical banks in accordance with the bank order and column order.

Step 2: We repeat Step 1 S times to place the successively S row of logical matrix into form the row of memory windows.

Step 3: The other matrix rows are arranged according to Step 1 and 2 until all the matrix elements are mapped into DRAM physical row.

In this scheme, each physical row contains B elements from one logical matrix row and contains S elements from one logical matrix column. Each logical matrix row is mapped into b $A \times B$ continuous memory windows in b banks and each logical matrix column is mapped into a $T \times S$ discrete storage windows. This process balances the distribution of the row and column vectors of matrix in one physical row, thereby improving match between data access continuity of row and column in the ARM/CM application and the physical row of the DRAM.

The physical address of DRAM ($D[t][r][c]$) has a one-to-one correspondence with the logic address of matrix LM ($LM[i][j]$). The relationship between physical address and logic address is analyzed as follows, in which $W(m, n)$ is used as springboard.

- Window select function: $W(m, n) = W(i/S, i\%S)$, which shows that the window is determined by the row number of logical matrix;
- Window position function: $E[x][y] = E[j/(b * B)][j\%B]$, which shows that the position in the window is determined by the column number of the logical matrix. The bank of this window is $(j/B)\%b$, i.e. $t = (j/B)\%b$.

Therefore, the mapping relationship between the logical matrix and the DRAM is: $LM[i][j] \leftrightarrow D[(j/B)\%b][i/S * A + j/(b * B)][(i\%S) * B + j\%B]$.

5.2 Optimal Window Configuration of WMLS

WMLS improves the match between data access continuity and storage continuity of DRAM. The available DRAM bandwidth is improved by reducing the no-data-visit operations in AR/CMA applications. In the AR/CMA process, the amount of data is equal in both CM and RM access. Therefore, the available bandwidth is determined by the number of no-data-visit operations, such as ACTIVE and PRECHARGE commands.

Optimal window configuration aims to minimize the number of no-data-visit operations. As shown in Fig. 3, a row of logical matrix is distributed alternately in multiple physical rows from different banks. The ACTIVE and PRECHARGE commands for different physical rows can be executed in parallel and only the latency of one ACTIVE and PRECHARGE commands is needed. To access one row, the number of ACTIVE and PRECHARGE com-

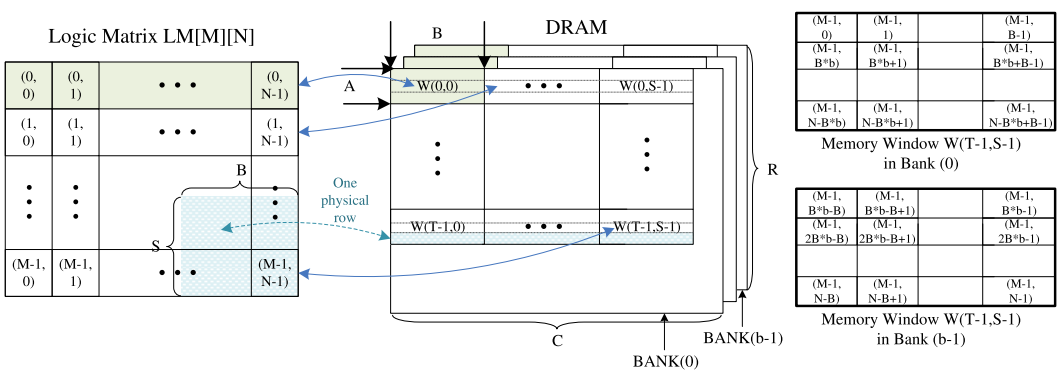


Fig. 3 Mapping relationship between the logical matrix and DRAM.

mands should be equal to A, i.e., the window height. To access one column, the data are distributed in every window in the same bank. Thus, the total number of no-data-visit operations is $2 * M * N/S = 2 * M * N * B/C$. The total number of no-data-visit operations in one AR/CMA access for an $M * N$ matrix is:

$$Z = 2 \times A \times M + \frac{2 \times M \times N \times B}{C} \tag{3}$$

Theorem 1: In the AR/CMA mode, the optimal window for WMLS is $A = N \times \sqrt{\frac{1}{b \times C}}$ and $B = \sqrt{\frac{C}{b}}$, where the logical matrix is $M \times N$, and the width of physical row is C, and the number of banks is b.

Proof. According to (1), the total number of ACTIVE and PRECHARGE operations is a function of the height A and length B of memory window, and $A \times B \times b = N$. We replace A with $N/(B \times b)$ in (1) and obtain:

$$Z = \frac{2 \times M \times N}{B \times b} + \frac{2 \times M \times N \times B}{C} \tag{4}$$

To fulfill the condition for minimal number of Z, we obtain the derivative of Z:

$$\frac{dZ}{dB} = -\frac{2 \times M \times N}{B^2 \times b} + \frac{2 \times M \times N}{C} \tag{5}$$

Given that $\frac{dZ}{dB} = 0$, only the positive value is valid, and the following condition holds:

$$B = \sqrt{C/b} \tag{6}$$

Therefore the optimal window is $A = N/\sqrt{b \cdot C}$ and $B = \sqrt{C/b}$, and the minimal number of Z is $4 \cdot M \cdot N/\sqrt{b \cdot C}$.

Taking a $4K \times 4K$ logical matrix as an example, the optimal window of the DRAM with 8 banks and 2048 elements in physical row is 32×16, according to Theorem 1.

For the traditional row-wise order storage scheme, each matrix row is stored in a DRAM physical row. M and $\frac{M \times N}{b}$ ACTIVE and PRECHARGE commands are needed to access the entire matrix for RM and CM access, respectively. The total number of no-data-visit commands is $2(M + \frac{M \times N}{b})$, which is larger than Z with an optimal window when $C > 2\sqrt{b}$.

In some applications, the number of the RM access and CM access is unequal. Therefore, we must adjust the optimal window to achieve high bandwidth utilization.

Deduction 1: The optimal window configuration for WMLS is $A = N \times \sqrt{\frac{y}{b \times C \times x}}$ and $B = \sqrt{\frac{C \times x}{b \times y}}$, where x and y are the number of RM and CM access, respectively. Therefore, the traditional row- and column-wise order storage schemes are simply the extreme cases of WMLS.

Proof. As analyzed above, for x times RM access and y times CM access, the total number of no-data-visit operations is

$$Z = 2 \times A \times M \times x + \frac{2 \times M \times N \times B \times y}{C} \tag{7}$$

By replacing A with $N/(b * B)$ in Eq. (5), we can obtain

$$Z = \frac{2 \times M \times N \times x}{B \times b} + \frac{2 \times M \times N \times B \times y}{C} \tag{8}$$

We obtain the derivative of Z to find the condition for the minimal number of Z:

$$\frac{dZ}{dB} = -\frac{2 \times M \times N \times x}{B^2 \times b} + \frac{2 \times M \times N \times y}{C} \tag{9}$$

When $\frac{dZ}{dB} = 0$, we obtain

$$A = N \times \sqrt{\frac{y}{b \times C \times x}}, B = \sqrt{\frac{C \times x}{b \times y}} \tag{10}$$

If a program contains only the RM access mode and it is said $y = 0$, then the optimal window is as small as possible for A, which is traditional row-wise order storage mode. Conversely, if a program contains only the CM access mode, then the optimal window is traditional column-wise order storage mode.

6. Implementation of WMLS

WMLS is a common optimization technique for AR/CMA applications and can be used to improve the DRAM bandwidth utilization in a variety of computing platforms, such as FPGA, CPU and GPU, as shown in Fig. 4. In this section, we combine the characteristics of various computing platforms to implement WMLS. Based on the reconfigurable features of the FPGA platform, the DRAM controller with an optimal window is customized. However, the memory controller is fixed in CPU and GPU general-purpose computing platforms. Therefore, a software optimization method is used for the effective mapping between logical matrix and memory window in DRAM.

6.1 WMLS Implementation in FPGA Platform

Recently, FPGAs have become an important platform in building application-specific hardware systems, particularly for matrix applications [6], [19]. The computational capability FPGAs has increased rapidly. FPGAs can be used to build a large number of custom processing element units. Given that high IO bandwidth is needed for FPGA, low bandwidth utilization significantly affects the performance of FPGA.

A custom WMLS DRAM controller is implemented

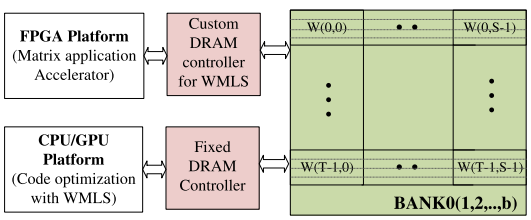


Fig. 4 Implementation of WMLS on different computing platforms.

on the FPGA platform to improve the match between matrix data and window memory. This controller provides a simple interface for users to access matrix data in a row- or column-wise mode. The matrix access is decomposed into a series of burst operations and these operations are further decomposed into basic commands for DRAM modules.

6.1.1 Structure of the WMLS DRAM Controller

Figure 5 shows that the WMLS DRAM controller is mainly composed of the Window Configure, Command Parsing, Burst Operation FIFO, DRAM Basic Control Signal Generate, Read Data FIFO and Write Data FIFO modules. Window Configure module stores the configured information of memory window, such as the width and length of the window and the size of physical page in DRAM. Command Parsing module decomposes the matrix access into a series of burst access operations and the burst operations are stored in the Burst Operation FIFO. Burst Operations FIFO, Read Data FIFO, and Write Data FIFO are used to buffer the burst operations, data of reading, and data of writing, respectively. Moreover, they play a role in separating the different clock domains to alleviate the computation logic from the constraints of DRAM clock, or vice versa.

Control Signal Generate module was developed from the Xilinx IP core, which is used to complete the initialization, refreshing, and control of state machine, as well as the issuing of access command of DRAM. According to the memory state and the current burst operation, this module generates corresponding commands (CS, RAS, CAS, WE, and addresses A0-A12), such as ACTVE, PRECHAREDE, READ and WRITE to DRAM modules.

The WMLS DRAM controller provides the matrix row and column command to access a continuous or discrete storage window in DRAM, to achieve high bandwidth utilization for AR/CMA applications. The command to access matrix data includes two parameters: the starting address (*addr*), row or column access (sel_RC). Here, *addr* is the starting physical row and column addresses in DRAM for the matrix access commands. The Command Parsing module decomposes the row or column access command into a series of short burst access operations according to window configuration. Figure 6 shows the decomposition principles. To access the matrix row, we use serial burst operations to access a continuous $A \times B$ rectangular window with *addr* for the upper left corner. To access the matrix column, we use the serial burst operations to access a discrete $T \times S$ window. The intervals for row and column are B and A, respectively.

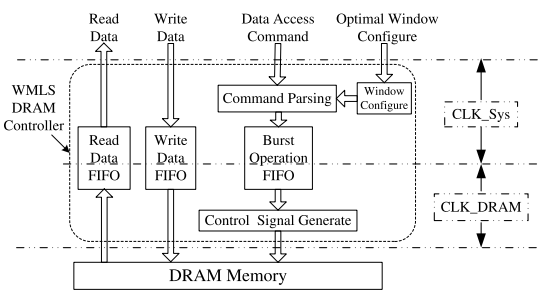


Fig. 5 Structure of the WMLS DRAM controller.

6.1.2 Performance

We implement the WMLS DRAM controller on Xilinx Virtex-5 FPGA platform based on the Memory Interface Generator for DDR2 SDRAM memory. The usage of hardware resources are shown in Table 1. WMLS controller uses more slice to support optimal window access mode. Six block RAMs are used to build read/write data FIFO. The final operating frequency of tradition and WMLS DRAM controller is 200 MHz.

We first test the effect of window configuration on the bandwidth of external DRAM and prove the optimal window theory. By using a 200 MHz clock, we determine that the peak bandwidth of DRAM module was 6.4 GB/s. As shown in Fig. 7, the available bandwidth of the DRAM is a function of A, which is the width of storage window. Based on Theorem 1, the bandwidth of AR/CMA reach the optimal value when $A = N \times \sqrt{\frac{1}{b \times C}}$, where $C = 1024$, and $b = 8$. For $1K \times 1K$ and $4K \times 4K$ logical matrices, the optimal windows are $A = 8$ and $A = 64$, respectively, as shown in Fig. 7. The vertex in each curve represents the practical peak bandwidth and the corresponding windows size, thus verifying

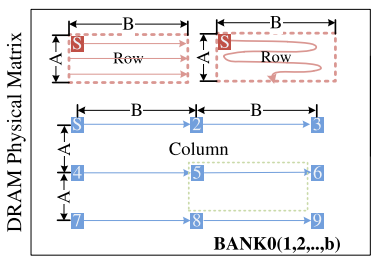


Fig. 6 Commands decomposition from logical matrix to burst operations.

Table 1 Resource usage of WMLS DRAM controller.

	Slice Register	Slice LUT	Block RAM	Frequency
Tradition	1888	2712	6	200 MHz
WMLS	2563	3733	6	200 MHz

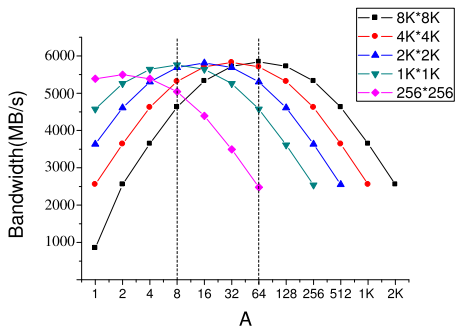


Fig. 7 Available bandwidth with different size window for ACM/RM.

Theorem 1.

Figure 8 shows the bandwidth comparisons of CM and RM access between WMLS and traditional row-wise order scheme for different matrix size. The optimal window configuration was used under different conditions. A sharp decrease in the bandwidth was observed for the col_access, along with the enlargement in the matrix. Because, for small matrices, such as 16×16 and 32×32, the whole matrix can be fitted into one physical row of DRAM, incurring no overheads for no-data-visit operations. However, for large matrices, the column data are stored in different physical row; thus, the bandwidth is only 500 MB/s (8% of the peak bandwidth). WMLS balances the performance between CM and RM access. The bandwidth utilization of CM and RM access is gradually stabilized at 94.7% and 91%, respectively. Compared with the col_access mode, the performance increased by a factor of 11.4. The average bandwidth of AR/CMA with WMLS improved by 73.6%, compared with the traditional row-wise order access mode.

6.2 WMLS Implementation in CPU Platform

The hardware structure of CPU platform is fixed. The computation unit achieves high operating frequency with deep pipeline organization and achieves high performance with superscalar, hyper-threading, out-of-order, and speculative techniques. The storage wall problem becomes increasingly significant with increasing difference between CPU performance and DRAM performance. For AR/CMA applications, the low DRAM bandwidth utilization further limits the play of CPU performance.

6.2.1 WMLS Layout Scheme

In general-purpose CPU platform, storage hierarchy based-on cache structure is employed, according to the temporal locality and spatial locality of data. This hierarchical storage structure is transparent to the programmer. When a data is accessed from DRAM, the corresponding block of data is read into a cache line first. The size of that block is the length of cache line. Then, data is read from cache. For the CM access, the continuous data of matrix column cannot be prefetched into a cache line, since the line-space of two continuous data in one column is very large. Therefore, the DRAM bandwidth utilization will be further reduced for AR/CMA access applications.

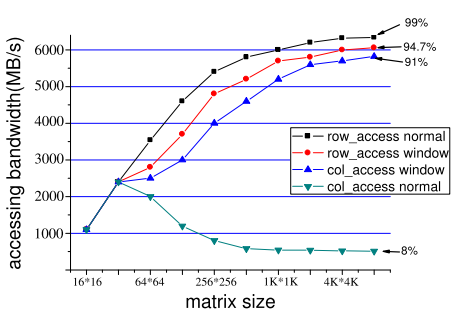


Fig. 8 Bandwidth comparison of the CM and RM access between traditional row-wise order and WMLS.

A similar WMLS can be applied in the CPU platform. Each program reserves a physical storage space as virtual space for data arrangement in WMLS. The operating system dynamically maps the virtual space into different DRAM physical spaces according to the current operating environment. The mapped unit is composed of multiple physical rows and is aligned with physical pages. Therefore, the virtual space can be regarded as a 2D storage structure, where width is equal to the size of DRAM physical page. In the CPU platform, the memory access mechanism is fixed, but the access behavior in high-level programs is flexible. We can modify the matrix program to realize mapping from logical matrix to DRAM space.

The implementation of WMLS in the CPU platform involves the modification of the data layout of matrix and the access algorithm. The unit of data in Cache is set to be the same as the size of a cache block according to the cache locality principle. The size of storage window is several times that of the cache block. Figure 9 shows the data layout of the matrix, where the storage window is $A \times B$, and the length of physical row is C. A row in the logical matrix is stored in a window and the window $W(0, S-1)$ corresponds to the S^{th} row. The first column of logical matrix corresponds to the first element in all windows, as shown by red dotted line.

The relationship between the starting address of virtual space (P_addr) and address of window data $W(t, s)[i][j](P\_w)$ is expressed as

$$P\_w = P\_addr + (t * A + i) * C + s * B + j \tag{11}$$

In application, a pointer is used to realize the algorithm that accesses the matrix in the storage window mode, such as the double loops Rc and Cc shown in Fig. 9. Code section Rc accesses a row of logical matrix, as well as the storage window. Code section Cc accesses the data in a fixed position of each storage window, as well as column vector.

6.2.2 Performance

We implement the WMLS layout scheme in a Linux server with Intel Core2 Quad Q8400 CPU, 32GB memory and the

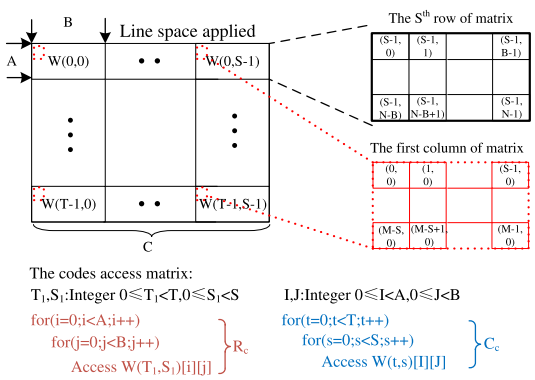


Fig. 9 WMLS map scheme in CPU platform.

Federo10 operating system. In the experiments, the matrix layout is optimized according to WMLS, and the matrix is read, modified, and written by the CM access mode and RM access mode, respectively. The $4K \times 4K$ matrix is divided in accordance with 64B, which is the size of cache block. Figure 10 shows the speedup of actual effective bandwidth for different window size. A significant performance increment is observed when the width of the storage window changed from 4 to 64. When the window width was 8, we have obtained the maximum speedup with a factor of 1.6.

6.3 WMLS Implementation in the GPU Platform

GPU includes many computing cores and has high computing performance. The ratio of computing performance to IO performance is very low. Therefore, the actual memory bandwidth utilization will decide the play of GPU and memory wall phenomenon is further exacerbated in GPU platform. The optimization of data layout scheme become main work in transplanting an algorithm into GPU platform.

6.3.1 WMLS for the Warp Mode

The GPU platform employs the warp mode to access memory. Each GPU kernel execution is a thread block and is mapped to a streaming multiprocessor (SM). The block is divided into multiple Warps, which contain 32 threads, as shown in Fig. 11. The program, which is executed in GPU, utilizes hundreds and thousands of threads, which are organized in a linear, 2D or 3D structures. Warp is the basic scheduling unit in SM and is the access unit in the process of accessing the memory. The access request of the 32 threads is issued together and data are returned from memory at the same time. If any thread of the warp is delayed during memory access, then the entire warp will suffer a delay.

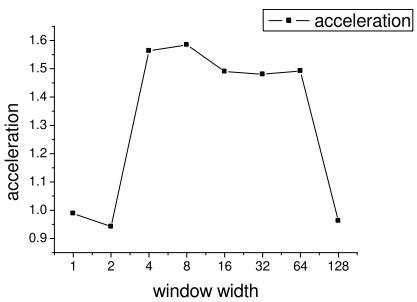


Fig. 10 Performance improvement with WMLS in CPU platform.

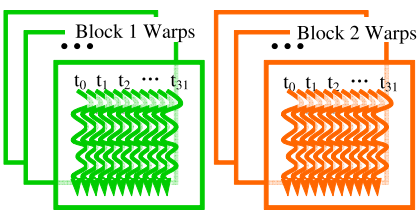


Fig. 11 GPU thread scheduling Wrap unit.

In traditional data layout scheme, the GDDR bandwidth utilization in the AR/CMA applications is very low. The global memory, which is composed of GDDR SDRAM, is the main storage structure of GPU. GDDR has the same basic characteristics as DDR2 or DDR3 does. Therefore, for the RM access, the data, which are accessed by using a warp with 32 threads, are stored contiguously in the same physical page, thus enabling high memory bandwidth utilization. However, for the CM access, the warp has to access the data in multiple physical rows, which results in significant decline in memory bandwidth utilization. Thus, the whole performance of AR/CMA will be greatly limited.

In GPU platform, WMLS is implemented with 2D dynamic memory, by using the cudaMallocPitch function. The 2D memory ensures continuous data in the row to enable quick access and to obtain a high memory bandwidth. Therefore, each row in the 2D memory can be treated as a physical row in DRAM. Figure 12 shows the 32×32 2D data space that stores the 32×32 matrix M, where the storage window is set as 8×4. The logical matrix is mapped into the 2D space according to WMLS, where $C = 32$, $A = 4$, $B = 8$, and $S = 4$. The matrix element $M[i][j]$ is mapped to the $GData[i/S * a + j/B][i\%S * B + j\%B]$. Similar to FPGA and CPU, the row data in WMLS in GPU platform are also distributed in the storage window, whereas the column data is distributed in a fixed position in each window.

In the CUDA programming model, block ID and thread ID are used to realize mapping between logical matrix and GDDR. Given that the basic access unit is warp with 32 threads, the aim of WMLS is mainly to balance the access of each warp and to improve the average performance of warp for AR/CMA applications, as evident in the example provided in Fig. 9. The GPU threads can also be organized into a 2D window, as shown in Fig. 12. The storage size is equal to or larger than that of the thread window. We can move the thread window in the storage window to access every element. The storage window of column can be processed in the same manner. The thread windows are discrete and are distanced from each other.

6.3.2 Performance

In our GPU platform experiments, the programming environment was the vs2008+CUDA, and the hardware was the

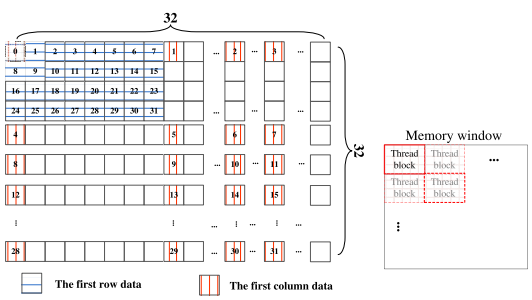


Fig. 12 Example of data arrangement and access with window width of 4.

Nvidia Geforce480. We implement four kernel functions, namely row_line_rw (RM access with traditional row-first model), col_line_rw (CM access with traditional row-first model), row_window_rw (RM access with WMLS scheme) and col_window_rw (CM access with WMLS). All matrix elements are read, modified, and written using these four kernel functions. The running time and memory performance are analyzed by the CUDA profile.

We use a the matrix size of $1K \times 1K$ to test the performance of WMLS. The width of window varies from 1 to 256 as a power of 2.Each kernel block has 1204 threads. As shown in Fig. 13, the performance of col_line_rw is the worst, among all kernel functions, whereas that of the row_line_rw is the best. When the window width is increased from 1 to 32, the performance of col_window_rw significantly declines and that of the row_window_rw significantly increases. The performance of row_window_rw improves gradually with window width, because the row data accessed by warp are more continuous. When the window width is more than 32, the performance of col_window_rw and row_window_rw changes slightly. Because the row data accessed by warp are completely continuous, and the column data were not continuous at all. More than one set of data can be accessed in a burst. The window width in the intersection between row_window_rw and col_window_rw is either 4 or 8. We have determined the balance of theAR/CMA process, and have obtained the window width in which the overall memory performance of the AR/CMA is the highest.

Figure 14 shows performance improvement of AR/CMA with different window width, when the matrix size is $1K \times 1K$ and $2K \times 2K$ respectively. When the window width is between 2 and 32, the storage performance acceleration is higher than 1. When window width is 4, we can achieve the highest speedup by a factor of 1.7X and 2.0X.

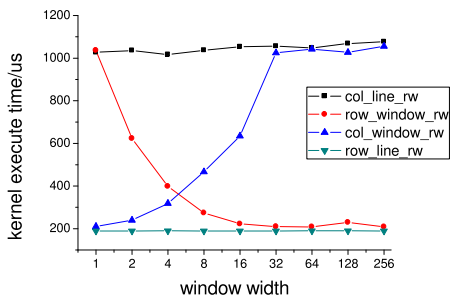


Fig. 13 Comparison of running time of four kernel functions with different window width.

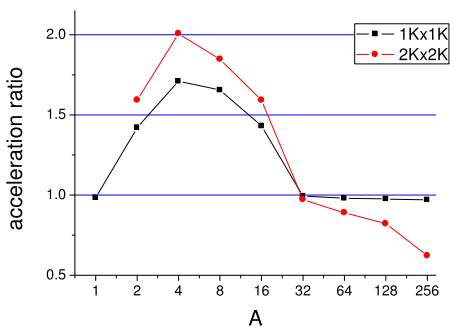


Fig. 14 Speedup of WMLS scheme with different window width.

7. Performance Evaluation for CS SAR Application

In this section, the Chirp Scaling (CS) application [18], which is a SAR radar image processing algorithm, is used to evaluation the performance improvement of WMLS scheme on FPGA, CPU, and GPU platforms, respectively.

7.1 CS Algorithm

As shown in Fig. 15 (A), the program flow diagram of CS algorithm consists of four 2D FFT (azimuth FFT, range FFT, range IFFT, and azimuth IFFT) and three factor compression computations, which is the complex element multiplication of two matrixes. The data access mode of CS application is shown in Fig. 15 (B). The first step is FFT operation in the azimuth direction, which carries the raw SAR data into the range domain. The access mode of step 4 (azimuth IFFT) is similar to step 1. The range operations are embedded in the middle of two azimuth transforms. The Chirp Scaling range compression operation (Range FFT) is followed with the first step. In this step, the raw SAR data matrix is read with column-wise and result matrix is written with column-wise. The access mode of step 3 (Range IFFT) is similar to step 2. Thus, the access mode of range operations (both range FFT and range IFFT) is to read data matrix and write result matrix with row-wise, and the access mode of azimuth operations (both azimuth FFT and azimuth IFFT) is to read data matrix and write result matrix with column-wise.

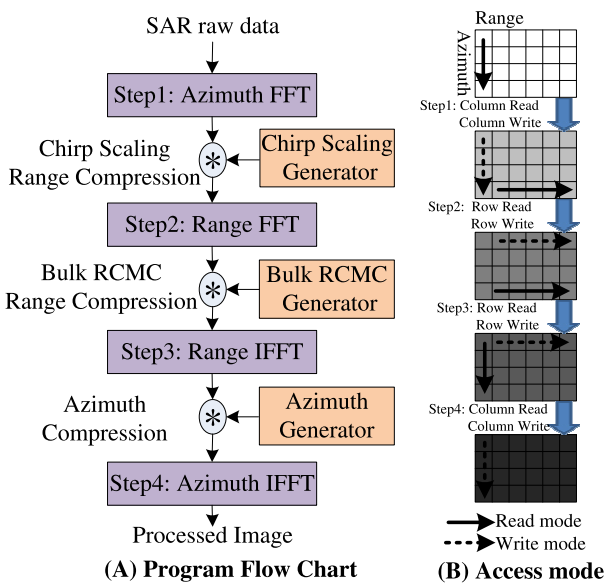


Fig. 15 Program flow chart and data access mode of CS application.

7.2 Performance Comparison for CS Application

Table 2 shows the performance comparison of CS application between normal access scheme and WMLS scheme for FPGA, CPU, and GPU platforms, respectively.

For FPGA platform, we implement a CS accelerator, equipped with a cluster of FFT-PEs, on Xilinx Virtex-5 XC5VLX330T FPGA chip, as shown in Fig. 16. The structure of FFT-PE is similar to that in [5], which is composed of four double-precision floating-point multipliers and six double-precision addition and only one RAM to buffer the intermediate data. The FFT-PE can perform the computation of 1D radix-2 FFT/IFFT and complex multiplication in compression operations with dynamic configurable pipeline scheme. Due to the limitation of DSP48E resource, only eight FFT-PE units could be integrated into CS accelerator. The achieve maximum frequency and run frequency are 213.9 MHz and 200 MHz, respectively.

The speedup of WMLS scheme is increased with the size of CS application, when the size is not greater than 1K. This is because the computation bandwidth is greater than DDR bandwidth and the computation is overlapped with data access, so the performance of WMLS scheme is proportional to the IO bandwidth utilization. When size is 4K, the maximum computation bandwidth for 8 FFT-PEs is about 4300 MB/s, and the IO bandwidth of WMLS for row access and column are 5978 MB/s and 5703 MB/s, respectively. So, the IO bandwidth was not made full use and the speedup of WMLS scheme is 4.29. However, more FFT-PE units can be integrated into new FPGA chips, such as Xilinx Virtex-6 or Virtex-7, and the IO bandwidth of WMLS scheme will make full use to achieve higher performance.

For CPU and GPU platforms, the optimal window configuration of WMLS scheme is employed in CS SAR application and the results are shown in Table 2. Due to the similar memory hierarchy structure in CPU and GPU platforms, the performance of WMLS scheme is improved with the increase of the size of CS application. Compare with the traditional access method, the maximum speedup factors are 1.33X and 1.56X for CPU and GPU platform, respectively. Each step in CS application, as shown in Fig. 15, is composed of three parts: read of matrix data, calculation of FFT/IFFT and factor compression, and write of results matrix. However, these parts can not be effectively overlapped and the run time for calculation is larger than that for access matrix, especially for CPU platform. Thus, the speedups of WMLS scheme are smaller than that in Fig. 10 and Fig. 14.

Table 2 Performance comparison of CS SAR application between normal access scheme and WMLS scheme for FPGA, CPU, and GPU.

Size of CS		128	256	512	1024	2048	4096
$FPGA^1$	Normal	0.63	3.46	17.96	78.45	327.4	1344.1
	WMLS	0.36	1.10	4.01	16.58	72.35	313.5
	Speedup	1.74	3.15	4.49	4.73	4.52	4.29
CPU^2	Normal	6.1	24.1	162.0	652.0	2667	16188
	WMLS	5.2	21.8	133.9	509.4	2136	12171
	Speedup	1.15	1.16	1.21	1.28	1.29	1.33
GPU^3	Normal	1.3	3.7	14.5	53.4	211.7	866.8
	WMLS	1.1	3.2	10.3	38.4	138.1	556.7
	Speedup	1.18	1.15	1.36	1.40	1.53	1.56

Note 1: Xilinx Virtex-5 XC5VLX330T FPGA, 8 FFT-PE units
Note 2: Intel Core 2 Quad Q8400 CPU, 128K L1 Cache, 4M L2 Cache
Note 3: Nvidia Tesla C2070 GPU, 448 thread processors

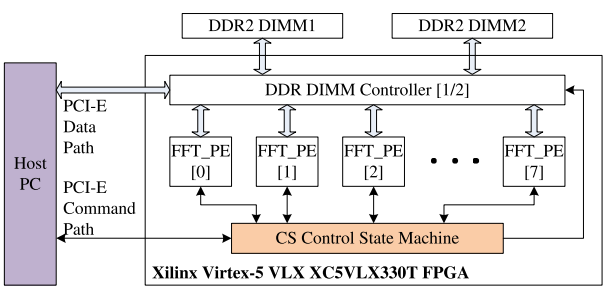


Fig. 16 FPGA accelerator structure for CS application.

8. Conclusions

This paper has presented the data layout scheme (WMLS) and the optimal window layout of WMLS to improve the practical DRAM bandwidth for AR/CMA applications. The proposed scheme combined the DRAM characteristics and the access feature of AR/CMA. This scheme matched the storage continuity of DRAM and access continuity of AR/CMA. The different implementations of WMLS were presented according to the memory structure of FPGA, CPU, and GPU platform, respectively. The experimental results show that the proposed WMLS can achieve a DRAM bandwidth improvement ranging from 1.6X to 2.0X.

Finally, we apply WMLS scheme for Chirp Scaling SAR application, comparing with the traditional access approach, the maximum speedup factors of 4.73X, 1.33X and 1.56X can be achieved for FPGA, CPU and GPU platform, respectively.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions. This work is partially supported by NSFC (61125201).

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK Users' Guide, SIAM, 1999.
[2] W.M. Brown, "Synthetic aperture radar," IEEE Trans., Aerosp. Electron. Syst., vol.AES-3, no.2, pp.217–229, 1967.
[3] M. Coskun, K. Donglok, and K. Yongmin, "Efficient 2D FFT implementation on mediaprocessors," Parallel Computing, vol.29, no.6, pp.691–709, 2003.
[4] A.W. Wm and A.M. Sally, "Hitting the memory wall: implications of the obvious," ACM SIGARCH Computer Architecture News, vol.23, no.1, pp.20–24, 1995

[5] Y. Dou, J. Zhou, Y. Lei, and X. Zhou, "FPGA SAR processor with window memory accesses," IEEE International Conf. on Application -specific Systems, Architectures and Processors (ASAP), pp.95–100, 2007.
[6] Y. Dou, S. Vassiliadis, G.K. Kuzmanov, and G.N. Gaydadjiev, "64-bit floating-point FPGA matrix multiplication," Proc. 2005ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays (FPGA), pp.86–95, 2005.
[7] Q. Yi, K. Kennedy, H. You, K. Seymour, and J. Dongarra, "Automatic blocking of QR and LU factorizations for locality," Proc. ACM SIGPLAN Workshop Memory System Performance, 2004.
[8] 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, 1994.
[9] J. Suh and S.P. Crago, "PIM- and stream processor-based processing for radar signal applications," Proc. 3rd Workshop on Media and Streaming Processors, pp.77–85, Dec. 2001.
[10] Y. Pi, H. Long, and S. Huang, "A SAR parallel processing algorithm and its implementation," Proc. FIEOS Conf., pp.211–214, 2002.
[11] P. Gottschling, D.S. Wise, and M.D. Adams, "Representation-transparent matrix algorithms with scalable performance," Proc. 21st Annual International Conference on Supercomputing (ICS '07), pp.116–125, 2007.
[12] G. Ruetsch and P. Micikevicius, "Nvidia optimizing matrix transpose in CUDA," pp.1–24, 2009. Available: http://www.cs.colostate.edu/~cs675/MatrixTranspose.pdf
[13] V. Podlozhnyuk, "FFT-based 2D convolution," NVIDIA White Paper, June 2007.
[14] B. Tu, D. Li, and C. Han, "Two-dimensional image processing without transpose," Proc. 7th International Conference on Signal Processing, vol.1, pp.523–526, 2004.
[15] H. Kim and I.C. Park, "Array address translation for SDRAM-based video processing applications," Electron. Lett., vol.35, no.22, pp.1929–1931, 1999.
[16] Z. Mo, J. Han, Z. Wang, and C. Han, Study and Implementation of Parallelized SAR Imaging Without Transposing Data Matrix, J. Computer Research and Development, vol.40, no.1, pp.19–25, 2003.
[17] M. Onoe, "A fast digital processing for synthetic aperture radar without transposing data matrix," 13th Int'l Symp on Space Technology & Science, 1982.
[18] C. Yu and C. Chakrabarti, "Transpose-free SAR imaging on FPGA platform," 2012 IEEE International Symposium on Circuits and Systems (ISCAS), pp.762–765, 2012.
[19] G. Wu, Y. Dou, J. Sun, and G.D. Peterson, "A high performance and memory efficient LU decomposer on FPGAs," IEEE Trans. Comput., vol.61, no.3, pp.366–378, 2012.

Lei Guo was born in 1987. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2010, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture.

Yuhua Tang was born in 1962, professor, Ph. D. supervisor in Computer Science and Technology at National University of Defense Technology. His research interests include high performance computer architecture, parallel computation.

Yong Dou was born in 1966, professor, Ph. D. supervisor, senior membership of China Computer Federation (E200009248). He received his BS, MS, and PhD degrees in Computer Science and Technology at National University of Defense Technology in 1995. His research interests include high performance computer architecture, high performance embedded microprocessor, reconfigurable computing, and bioinformatics.

Yuanwu Lei was born in 1982. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2007, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture.

Meng Ma was born in 1986. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2011, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture.

Jie Zhou was born in 1980, assistant researcher. He received his B.S., M.S. and Ph.D. degrees in Computer Science and Technology from the National University of Defense Technology in 2002, 2005, and 2011. His research interests include high-performance computer architecture, parallel computing, and reconfigurable computing.