Parallel 2D FFT implementation on FPGA suitable for real-time MR image processing

Limin Li[1,a)] and Alice M. Wyrwicz[1,2]
[1]*Center for Basic MR Research, NorthShore University HealthSystem Research Institute, 1033 University Place Suite 100, Evanston, Illinois 60201, USA*
[2]*Department of Biomedical Engineering, Northwestern University, Evanston, Illinois 60208, USA*

(Received 17 December 2017; accepted 4 September 2018; published online 21 September 2018)

We report the design and implementation of a parallel two-dimensional fast Fourier transform (2D FFT) algorithm on a Field Programmable Gate Array (FPGA) for real-time MR image processing. Although a number of architectures of 2D FFT hardware processors have been reported, these generic processors or IP cores are not always effective for processing MRI data. The key feature of our design is that our processors are customized solely for real-time MRI applications. We demonstrate that by considering the unique features of real-time MRI data streams, we were able to develop and implement the 2D FFT processors that are resource-efficient and flexible enough to handle both regular and irregular data. Using a data-driven approach, we were able to simplify the inter-processor data communication while maintaining data synchronization without a synchronous clock signal bus and complex interconnection network. We experimentally verified our designs by processing multi-slice image data sets with 128 × 128 and 256 × 256 in-plane resolution. The results demonstrate the effectiveness of our 2D FFT processors and show that image reconstruction can be accelerated in proportion to the parallel processing factor. We achieved image-reconstruction processing rates up to 3000 and 800 slices per second for images with 128 × 128 and 256 × 256 in-plane resolution, respectively. The results also indicate that the image-reconstruction acceleration is primarily limited by the speed of the data transfer between the FPGA device and external sensors. *Published by AIP Publishing.* https://doi.org/10.1063/1.5019846

I. INTRODUCTION

Magnetic resonance imaging (MRI) applications, such as functional and connectivity imaging of the brain, typically involve acquiring a large number of images with high temporal and spatial resolution. Fast imaging sequences such as single-shot echo-planar imaging (EPI)[1] and its variants are used for data acquisition but require fast gradient switching that induce a variety of effects, including eddy currents,[2] mechanical vibrations,[3] and uncomfortable levels of acoustic noise,[4] and over long periods can cause frequency drifts[5,6] of the MR scanner, RF heat accumulation, and the involuntary motions[7,8] of imaged subjects. Such instabilities lead to magnetic field fluctuations which adversely impact the data quality, resulting in intensity and geometric distortions in a single image and inconsistency of pixel locations in repeated images. It is therefore essential to actively monitor both the system hardware and imaged subject to stabilize and correct for these effects as much as possible. A wide variety of methods, both passive and active, have been developed for this purpose, including corrections for head motions,[8] dynamic B_0 and gradient shimming,[5,6] resonant frequency monitoring and adjustments,[5] and eddy current compensation.[9,10] In passive approaches, field errors are only identified and corrected prior to data acquisition (i.e., pre-scan calibration),[11,12] and thus compensations for the field imperfection remain fixed until the data acquisition is completed. For active approaches, fields are not only monitored but also adjusted in real time via a feedback loop whenever fluctuations exceed preset criteria.[13,14] Although in principle active methods can offer superior elimination of field fluctuations, they present considerable challenges in practice because the feedback latencies must be negligible on the time scale of the data-acquisition. For example, a recent report[14] shows that a minimum feedback latency of 20 ms, which is long on the data-acquisition time scale, was achieved on a 7 T whole-body MR system with a large portion of the latency attributed to data processing on a CPU. It is possible to achieve real-time system shimming and real-time data correction using a feedback-based approach, however, if a hardware accelerator can dramatically reduce computational latencies within the feedback loop.

Graphics processing units (GPUs) are also increasingly used as data processing accelerators for MRI and other medical imaging techniques.[15] The GPU processing power capabilities are achieved by using thousands of small cores. Both GPUs and FPGAs have advantages and disadvantages for real-time data processing. GPUs can run at clock rates ten times as high as FPGAs, and they are optimized for floating point operations, which offer designers great flexibility in designing and implementing algorithms. FPGAs are optimized for fixed-point data operations, which often require the algorithms to be redesigned to improve operational efficiencies. In terms of data processing rates alone, currently the performance of GPUs is better than that of FPGAs.[16] However, there is a fundamental difference in the operation of GPUs and FPGAs: the former are

[a)]Author to whom correspondence should be addressed: lli@northshore.org. Tel.: (224) 364-1405. FAX: (847)492-0731.

software-based processors, whereas the latter are hardware-based processors. In particular, GPUs operate through fetching and decoding operation instructions, a process that occupies many clock cycles. FPGAs implement algorithms on the logic circuits without the need for such processes. For this reason, currently the timing jitter of GPUs is ten times higher than that of FPGAs.[16] Thus, GPUs are not the best candidate for some applications such as inline data correction, especially in high-speed MRI, where time-deterministic processing with a very low timing jitter is required.

In a wide variety of MRI applications, the data processing in the feedback loop may involve the field map and image reconstruction, raw data or image filtering, image correlation and convolution calculation, and other processing functions. The most common core processing function is a 2D FFT, and thus it is crucial to accelerate the 2D FFT computations in order to reduce the total processing time. Hardware processors including dedicated FFT processor chips and DSP chips such as FPGAs or IP cores[17–24] have been reported to dramatically improve processing speeds for many applications, and further gains can be achieved by using parallel processing. However, generic hardware processors are not always effective for accelerating image processing of MRI data streams due to the unique structures of the raw data. It is necessary to customize an algorithm and redesign the architectures in order to implement a parallel 2D FFT specifically for MRI data streams.

In this paper, we first describe a new algorithm for parallel implementation of a 2D FFT which considers the real-time flow patterns of MRI data. We then present the design and testing of a hardware processor for MRI image reconstruction, capable of supporting both regular and irregular MRI data, using the 2D FFT algorithm implemented on an FPGA. We demonstrate the effectiveness of our design through real-time reconstruction of MR images. Our results show that the computational acceleration was proportional to the parallelism factor and we were able to achieve processing rates up to 3000 and 800 frames/second for 128 × 128 and 256 × 256 images, respectively. Finally, the limiting factors for further acceleration of the 2D FFT computations are discussed.

II. DESIGN FUNDAMENTALS

A. Review of a generic parallel 2D FFT algorithm

To illustrate the problems with using generic 2D FFT processors to process MRI data in real time, it is helpful to review the conventional implementation of a parallel 2D FFT on a hardware device. A sequential 2D DFT on a $N_1 \times N_2$ data matrix, omitting normalization factors, can be defined by the following equation:

$$X(m,n) = \sum_{k=0}^{N_2-1} \left\{ \sum_{l=0}^{N_1-1} S(k,l) W_{N_1}^{nl} \right\} W_{N_2}^{mk}, \tag{1}$$

where $0 \leq n, l \leq N_1 - 1$ and $0 \leq m, k \leq N_2 - 1$, and $W_N = \exp(j2\pi/N)$. $S(k,l)$ represents time-domain signal samples, and $X(m,n)$ represents its DFT, the frequency components, or calculated image pixels. The part inside the curved parentheses represents a N_1-point 1D DFT. The computation of this 2D DFT can usually be carried out through row-column (RC) decomposition: N_2 N_1-point 1D DFTs on the rows of the data matrix $S(k,l)$ followed by N_1 N_2-point 1D DFTs on the columns of the resulting data matrix. Fast computations of the 1D DFT are usually executed by means of a 1D FFT. Throughout the remainder of this paper, we assume the values of $N_1 = N_2 = N$ but will maintain different subscripts to distinguish the row-wise operations from the column-wise operations for the purpose of clearer discussion. To parallelize 2D FFT computations, typically the 2D-matrix data set is uniformly partitioned and distributed to P processing elements (PEs) that process subsets of the data simultaneously. Here a PE consists mainly of a 1D FFT module with supporting circuits and P is the parallelism factor. For example, parallel operations of the RC algorithm on a $N_1 \times N_2$ matrix of data with P PEs can be executed in three main steps. In step 1, each PE is allocated m_2 consecutive rows of data in which $m_2 = N_2/P$ and the 1D FFTs are performed on the rows. In step 2, a matrix transposition is performed. In step 3, the resulting data are redistributed such that the data can be consecutively accessed when 1D FFTs are performed on the m_1 rows with each PE in which $m_1 = N_1/P$.

When this parallel algorithm of the 2D FFT is applied to the real-time MRI data streams, we face two main issues. First, the signal comes in a sequence of blocks, and thus a complete 2D-matric data set is not always available at any moment. The uniform data partition and allocation described above would make some of PEs idle much longer than others because no data are available for processing, which would increase overall processing latencies. Therefore, real-time processing constrains how the 2D data can efficiently be partitioned and allocated to each PE. Second, the algorithm is not always effective due to the unique structures of MRI data. The 2D DFT definition given in Eq. (1) requires that in order to correctly reconstruct images the raw signals must be spatially encoded such that their phase modulation is linear in any dimension. The data that satisfy this requirement are defined to be regular in structure; otherwise, the data are defined to be irregular.[25] Although the MRI raw data acquired under some basic imaging sequences are regular, modern MRI experiments generate many more irregular data sets. The irregularity of the data not only prevents us from a direct application of the above algorithm but also complicates the data partition and allocation during parallel computations of a 2D FFT. Therefore, generic hardware processors or IP cores of a parallel 2D FFT are not always effective for processing MRI data streams in real time. The irregular data could be re-regulated by data reordering prior to the 2D FFT computations. However, while this approach could be effective with a software processor running on a central process unit (CPU), the data reordering approach would lead to unacceptable processing latencies and inefficient resource utilization on a hardware processor. Thus, it is necessary to customize an algorithm to the real-time dataflow.

B. Proposed 2D FFT algorithm

To effectively solve the two main issues described above, we propose a new algorithm for parallel implementation of

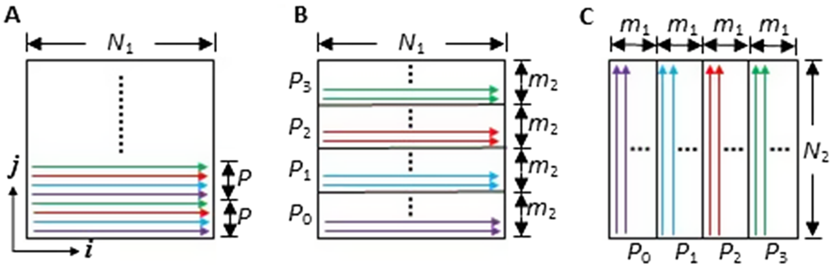

FIG. 1. Practical real-time data flow and allocation to four PEs. Data blocks are initially delivered to the PEs in an interleaving manner (a). All of the PEs begin computations almost simultaneously, and intermediate data are stored in a temporary storage (b). The stored data are read in column by column and delivered to the four PEs in an interleaving manner to complete the second FFTs (c). The data blocks are color coded for clearer illustration of the data distribution during FFT computation operations.

the 2D FFT. The proposed algorithm takes into consideration the real-time flow of the raw data and re-regulation of the data structure during processing. With four PEs as an example, the main steps and dataflow for a complete 2D FFT are illustrated in Fig. 1. In practice, assuming that an MR scanner is equipped with only one receiving channel, the input data streams into the processing system block by block. In step 1, the entry data stream is divided into blocks with N_1 length and delivered to the four PEs in an interleaving manner. The data structure can be represented as a virtual 2D matrix [Fig. 1(a)]. All of the PEs have almost equal idle time, and thus they each have their own shares of the data available and perform row-wise FFT operations simultaneously. When the row-wise FFTs are completed, the resulting data are placed into storage [Fig. 1(b)]. To complete the second FFTs on the columns of the stored data, the data elements are read in such a way that the columns are delivered to the four PEs in an interleaving manner [Fig. 1(c)].

In order to solve the second issue regarding the unique structures of MRI data, we have to consider a common feature of MRI data streams. We observed that MRI signals are spatially encoded and are uniformly sampled so that in a sequence of digitized data blocks with N_1 elements, the phase modulation within one block is always linear. With reference to Fig. 1(a), specifically, the phase modulation is always linear with index i. If a set of data is irregular in structure, the phase modulation is not linear from block to block, which suggests that the data re-regulation performed after the row-wise FFTs can be equally effective. In addition, the resulting data following the first FFTs are usually stored temporarily in memory before the second (column-wise) FFTs are performed. We can re-regulate the data during writing to the storage with an appropriate pattern [Fig. 1(b)]. By this method, the data re-regulation can be performed concurrently with data accesses, which does not introduce any additional processing latency, as can be seen from a specific example later.

III. DESIGN DESCRIPTION

A. Overview of 2D FFT architecture

Figure 2 shows a block diagram of a 2D FFT processor to be built on an FPGA device. The processor architecture can be seen as a hardware layout translated functionally from the proposed algorithm. It consists mainly of three types of modules: PE, on-chip storage, and Data Management Unit (DMU). A PE (indicated by a dashed box) is a basic building unit which consists of a basic 1D FFT core with connections to input and output FIFO (first in first out) buffers. The input buffer is used for reading the data from on-chip memory or an external sensor such as an MR detection coil. The output buffer is used for writing the data to on-chip memory. At the start of data processing, a sequence of data blocks of N_1 length streams into the 2D FFT processor and is immediately interleaved to multiple PE modules via the demultiplexer. All of the PEs perform 1D FFTs on their shares of the data, and the intermediate results

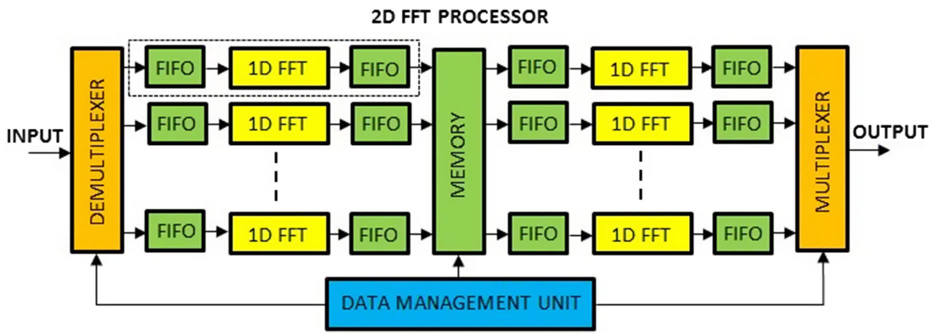

FIG. 2. Block diagram of a proposed 2D FFT processor. The processor consists primarily of three types of modules: PE, on-chip storage, and Data Management Unit (DMU). The PE, indicated with a dashed box, is a 1D FFT core combined with input and output FIFO buffers, and thus a core module to carry out the FFT computations. The on-chip memory is the module for storing the intermediate results between the first and second FFTs. The demultiplexer and multiplexer facilitates the data partition prior to the first FFTs and the data combination after the second FFTs, respectively. The dataflow is controlled by the DMU.

are temporarily stored into on-chip memory. The second FFTs begin when the first FFTs are complete. The stored data are then ready to be fed to the PE modules on the right side to complete the 2D FFT computations; at the same time, the PE modules on the left side are ready to receive the data from the next 2D slice. The processed data after a complete 2D FFT are multiplexed to become a sequence of data to be transferred to a host computer. When the 2D FFT is implemented on an FPGA, we not only focus on the processing speeds but also must consider the practical issues such as reduction in processing latencies and efficient utilization of on-chip resources, as explained in greater detail below.

B. PE core module

Typically the PEs responsible for the first FFTs (on the left side of the memory module in Fig. 2) do not physically occupy the same logic as those responsible for the second FFTs (on the right side of the memory module). During the row-wise operations of a 2D FFT process, all data elements pass through the physical logic of the left-side PEs once; during the column-wise operations, all data elements pass through the physical logic of the right-side PEs once. A PE structured in this way is illustrated with a dashed box on Fig. 2 and is defined as a single-pass PE. For extremely fast data streams, single-pass PEs may be required so that the left-side PEs can start receiving data from the next 2D slice without waiting until the 2D FFT computations on the current 2D slice are completed. However, in many other MRI applications, input data streams may be slower or may only be slice-wise (i.e., a short time interval is present between adjacent 2D slices during data acquisition). Such situations are typical even in high-speed MRI applications where a series of 2D slices of raw data are generated by using EPI-based imaging sequences.[1] The short time interval is required for spin system relaxation or switching different spin excitation locations in space. In these cases, fast processing allows the data to make double passes through a single PE module, the first pass for row-wise operations and the second pass for column-wise operations, without a significant decrease in processing rates. The PE modules on both sides of the memory (Fig. 2) can physically occupy the same logic as a single PE, which is defined as a double-pass PE. A PE structured in this way can significantly save hardware resources. The resource savings are especially desirable for parallel implementation of a 2D FFT on an FPGA device because more PEs can be structured and thus higher degrees of parallel processing can be achieved for the fixed capacity of a given device.

Figure 3 shows a block diagram of the architecture of a double-pass PE. For row-wise operations, initial input data pass through the PE along a pathway labeled "1" and are routed to on-chip memory SRAM for temporary storage. After the FFT computation operations on the rows are completed, the column-wise FFT operations begin and the data flow along a pathway labeled "2." The PE circuit logic is utilized twice in a single 2D FFT process which saves resource hardware.

In theory, the highest processing throughput could be obtained by using a large number of PE cores such that each core handles only one block of N_1 data elements. However,

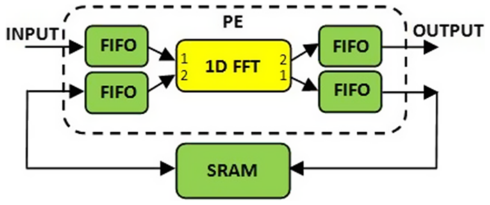

FIG. 3. Block diagram of a double-pass PE with connection to on-chip SRAM. Numbers 1 and 2 indicate the dataflow pathways of row-wise and column-wise operations, respectively, during the 2D FFT computations.

the throughput realized with such a high degree of parallelism comes at a high cost of the hardware resources. In practice, the achievable maximum throughput depends not only on the number of PEs but also on how fast the data are fed to the PE cores and the rate of data transfer during the inter-processor data communication. If the data access speed does not match the processing speed, portions of the logic will remain idle for many clock cycles, which not only reduces the hardware utilization of the FPGA but also places a limit on the achievable maximum throughput of the 2D FFT processor. This is especially true for real-time processing of MRI data. MR time-domain baseband signal samples usually can be delivered to FPGA-based FFT processors at rates not exceeding 400 kHz, equivalent to one sample per 2.5 μs under the current gradient technology. An FPGA device is capable of operating at rates of one sample per 25 ns or faster. Therefore, the choice of a parallelism factor P must take into consideration the throughput requirement, data availability, and data access bandwidth. Any extra savings on the hardware resources in a single 2D FFT processor can be used for other image-processing computations.

C. Data management unit

The major challenge in implementing parallel processing is accurately and efficiently controlling data flow. The operations of multiple PEs require multiple accesses to the same memory. It is crucial to prevent memory cells from updating before the values stored in these memory cells are utilized. At the same time, specific blocks of data in the memory must be updated and ready to be read by the multiple PEs in a specific time so that the latencies and idle clock cycles can be minimized. Many methods for multiple accesses to memory for parallel and distributed operations of an algorithm have been reported.[26–37] In our work, a data management unit (DMU) is used to schedule all requests for transferring data between the FIFO buffers and on-chip memory. The DMU consists mainly of multiple address generation units (AGUs) used to control reading and writing access to on-chip memory. Here we utilized a distributed scheme to design our DMUs. With the distributed scheme, multiple local data access sequences are generated simultaneously by the multiple AGUs, each of which supports the operations of only one PE core. There are two advantages of employing a distributed scheme. First, the interconnection network can be substantially simplified or even eliminated entirely, potentially reducing hardware costs and allowing for short data access latencies. Second, since one AGU is responsible for the operation of one PE core, these units

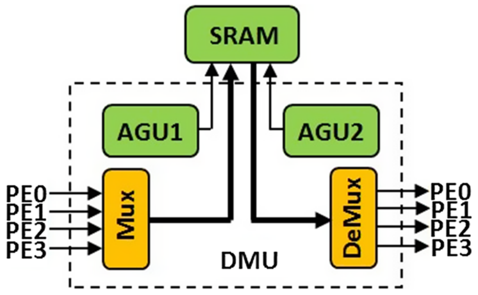

FIG. 4. Block diagram of the DMU architecture with connections to SRAM and four PEs. The thick lines represent data paths; and the thin lines represent address lines.

operate relatively independently which simplifies the design of the DMU.

In the parallel operations, the multiple AGUs must be synchronized to avoid conflicts among the multiple data-access requests. Conventionally, a single clock signal for the synchronization would be provided to all the AGUs, but this approach would require additional hardware resources and clock cycles. In our design, we greatly simplified the data synchronization by using a data-driven approach.[27,28] In data-driven processes, no operation instructions are required because the execution of an algorithm is predefined. All non-arithmetic operations are moved forward as long as their input data elements (operands) are available. This approach saves synchronization signal lines and avoids the need for a relatively complex network. To successfully implement the data-driven approach to manage the data flow, a handshaking scheme must be set up to avoid data losses. The major advantage of using the data-driven approach is that the address sequences can be generated recursively and the clock signal for data synchronization is no longer required, which greatly simplifies the architectures of our DMUs.

Figure 4 shows a block diagram of a DMU with $P = 4$. The data blocks from multiple input PEs are converted to a series of blocks that are written to the memory at the cell locations specified by the first AGU, AGU1, after the first (row-wise) FFT computations. Similarly, the stored data are read from the memory and transferred to the output PEs. The read access to the memory is specified by the second AGU, AGU2, prior to the second (column-wise) FFT computations.

D. Address generation unit

In this work, only a single memory module is used in a 2D FFT processor due to its programming and implementation simplicity. The sole function of an AGU is to map a 2D address space to a 1D address space. The design of an AGU is essentially a two-step process. First, a mapping equation is established, and then the AGU architecture is designed based on the equation.

AGU1 is designed to generate addresses for writing access to the on-chip memory after the first (row-wise) FFT computations. With reference to Fig. 1, the dataflow pattern can be described in the following equation:

$$l = i + (j \bmod P)N_1 + [j/P]N_1P, \tag{2}$$

where l is the output address and i and j are the indices of the 2D matrix. The flow pattern of the data elements determines which index runs faster. Recall that first the data elements are multiplexed over P PEs and followed by advancing i. When $i = N_1$, the same pattern repeats, but at the address offsets $l = N_1P, 2N_1P, \ldots, ([N_2/P] - 1)\, N_1P$. The function of the nested loops in Eq. (2) can be realized relatively easily in a logic circuit by using a modularized approach.[23] The main advantage of using the approach is that the modulo, multiplication, and division operations are completely avoided, which reduces clock cycles and hardware overhead. With this approach, an AGU can be constructed from one or more address generation cores (AGCs). Its behavior is determined by setting stride (**S**), base (**B**), and enable ($\mathbf{C_{in}}$). Here the AGU1 can be considered as a composite AGU since it is constructed from three cascaded AGCs [Fig. 5(a)]. The numerical labels of the AGCs correspond to the terms on the right side of Eq. (2). The connections and parameter settings of the AGCs are chosen to follow the data flow pattern described above. Specifically, AGC 2 advances automatically because $\mathbf{C_{in}}$ is set to a constant logic high (**T**), while the advances of AGC 1

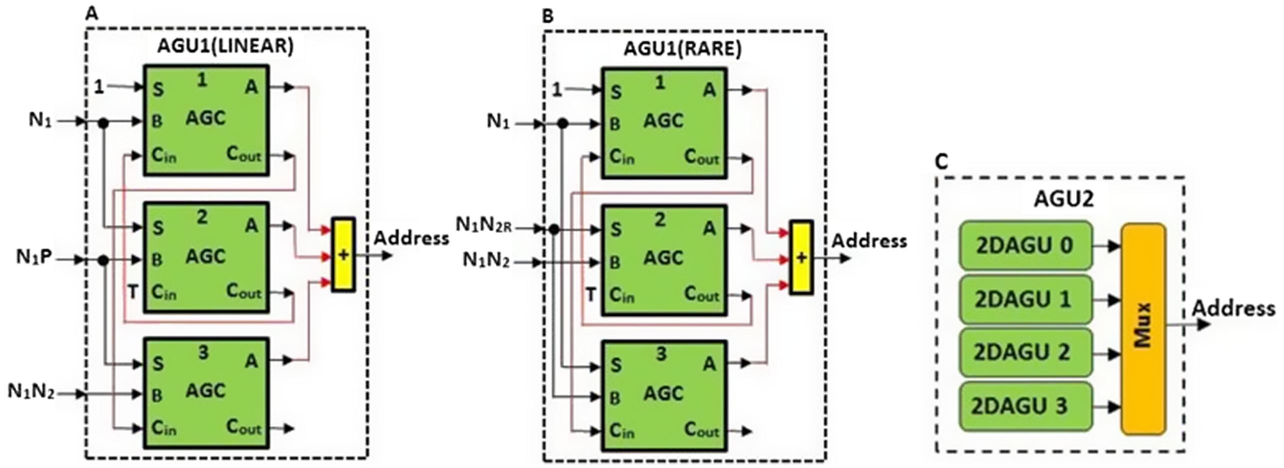

FIG. 5. Block diagrams of the AGUs. AGU1s are used to support regular MRI data (a) and RARE data (b), respectively, during the writing access, while AGU2 are used during the reading access (c).

are controlled by the carry signal $\mathbf{C_{out}}$ of AGC 2 and AGC 3 is controlled by AGC 1 in the same way. The final address output is simply a sum of the outputs of the three individual AGCs. Such an AGU is only effective for supporting regular data, such as spin-echo imaging data,[38] and thus is named AGU1(LINEAR).

In general, it is difficult to design an AGU that is flexible enough to support the irregular data acquired using different imaging sequences. The irregularities of these types of data complicate data partition and distribution. In addition to accommodating the data flow pattern mentioned above, this AGU must be able to re-regulate the data to be stored in the on-chip memory at the same time. Here for a detailed study, we target only the data acquired under RARE-based imaging sequences[39] which are used routinely in MRI experiments. In the case of RARE data, we have an address mapping equation as below,

$$l = i + (j \bmod R) \cdot N_1 \cdot N_{2R} + [j/R] \cdot N_1, \tag{3}$$

where R is the RARE factor and $N_{2R} = N_2/R$. To further facilitate the implementation of the address mapping, we set R equal to the parallelism factor P. Since Eq. (3) is apparently the same as Eq. (2), the AGU, named AGU1(RARE), can be constructed in the same way as AGU1(LINEAR), as can be seen from Fig. 5(b). The internal difference between AGU1(LINEAR) and AGU1(RARE) lies in the individual parameter settings of the AGCs.

AGU2 is designed to generate addresses for reading access to the on-chip memory such that the read-in data elements are uniformly distributed to the PEs prior to the second (column-wise) FFT computations. It is composed of P 2DAGUs. A 2DAGU was previously developed to support a sequential 2D FFT.[23] Each 2DAGU runs with the address offset equal to kN_1/P, where $k = 0, 1, \ldots, P-1$. AGU2 is able to perform 2D matrix transposition of the data to be de-multiplexed to the four PEs.

IV. HARDWARE IMPLEMENTATION

We chose National Instruments' 1D FFT IP core as a building block to develop the architectures of the 2D FFT processors using National Instruments LabView FPGA 2014 (National Instruments, Inc., Austin, TX, USA). We prototyped the processors on an FPGA chip XC7K410T (Xilinx Inc., San Jose, CA, USA.) which sits on a NI USRP-2940R board (National Instruments, Inc., Austin, TX, USA). The chip contains 406 720 logic cells, 795 36-Kbit block RAMS (BRAMs), and 1540 DSP slices. The FPGA device operates at a basic clock frequency of 40 MHz, but multiple clock domains are supported. Some portions of the same FPGA chip can run as high as 240 MHz depending on the utilization efficiency of the hardware resources. The board also provides on-board DRAM as an external memory. Since our processors are intended for real-time processing, we use only on-chip memory (BRAM) to avoid the bottleneck of the data transfer bandwidth between the FPGA and the DRAM.

The processors were designed to operate on fixed-point (FXP) numbers. Real/imaginary (I/Q) components of a digital signal were represented with 16-bit signed FXP-type numbers. The I/Q components of a data point were packed to a single 32-bit unsigned integer when the data flowed between processing modules or between the FPGA chip and the host computer. In order to reduce the impact of the resolution of the numerical representations on the precision of the 2D FFT, at different stages of a 2D FFT process, the word lengths and the ratios of the integer portion to the decimal portion were empirically adjusted based on the likelihood of data overflows or underflows. Specifically, these adjustments were performed to ensure that the signal-to-noise ratios of the images reconstructed with our 2D FFT processors matched those of the images reconstructed on our 9.4 T MRI scanner's host computer.

V. EXPERIMENTAL RESULTS AND DISCUSSION

In our experiments, the 2D FFT processors were implemented and tested for two data sizes, 128×128 and 256×256, with parallel processing factors P of 1, 2, 4, and 8. We restricted our experiments to utilizing on-chip memory (BRAM) even though on-board DRAM was available. All of our processors were designed for real-time image reconstruction. Ideally, all of the experimental tests would have been performed with the FPGA device integrated with the multiple-channel receiver of a MR scanner, in which the raw data received from multiple coils would be directly fed to the FPGA. Because such an MR scanner was not available for testing, all the tests were performed on static data which were already acquired and stored in a host computer. A graphical user interface (GUI) program was developed and operated on the PC to initialize, start, and interact with the FPGA. However, in order to simulate real-time processing, the GUI was also designed to manage the data transfer between the FPGA board and the PC in such a way that the data flowing into the FPGA appeared as a real-time data stream that would be received during an MRI experiment,

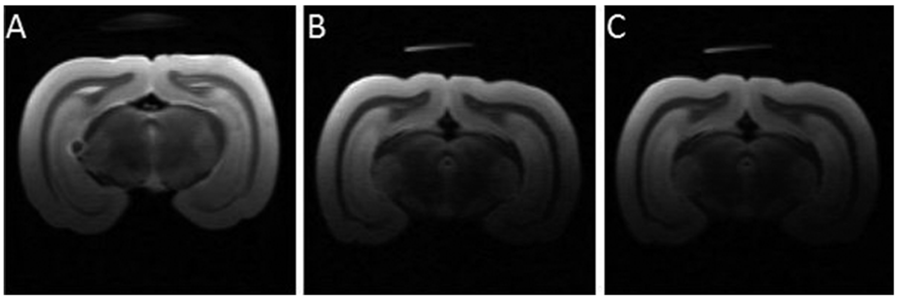

FIG. 6. Images with 128 × 128 plane resolution selected from a multi-slice RARE image set of a rabbit brain that were reconstructed with the 2D FFT processors with various settings. Images shown in (a) and (b) were processed with $P = 4$ and 8, respectively; image (c) was processed with conventional software running on the PC.

TABLE I. Comparison of the computation time (ms) at 40 MHz with various *P* values.

P	1	2	4	8
128 × 128	3.80	2.00	1.00	0.46
256 × 256	16.5	8.32	4.26	3.48

TABLE II. Comparison of the computation time (ms) at 80 MHz with various *P* values.

P	1	2	4	8
128 × 128	0.96	0.50	0.31	0.29
256 × 256	4.14	2.10	1.25	1.24

as described previously.[24] The PC is equipped with an 8-core i7 CPU and 12 GB DRAM. The FPGA board and the PC were linked by means of an MXI-Express 4 module installed in an NI PXIe-1073 express chassis (National Instruments, Inc., Austin, TX, USA). The data sets were transferred directly from the PC to the FPGA board at a rate up to 320 MB/s according to the specifications of the MXI-Express 4 module.

Figure 6 shows single-slice images selected from multi-slice RARE image sets of a rabbit brain. These images were reconstructed from raw RARE data using the 2D FFT processors with $P = 4$ and 8 [Figs. 6(a) and 6(b)] and also using a conventional FFT software running on the PC [Fig. 6(c)]. Images B and C appear almost identical, which verifies the effectiveness of our designs of parallelized 2D FFT and implementation on an FPGA.

In order to test how parallel processing accelerates the FFT computations, we measured the computation time of a single slice image reconstruction when the 2D FFT processors were used with variable *P* values and operated at a 40 MHz or 80 MHz clock frequency. The results are summarized in Tables I and II. Note that the computation time is reduced proportionally with increasing *P*. The time is reduced almost by a factor of 9 with $P = 8$ compared to the sequential computations ($P = 1$) for a 128 × 128 image at a 40 MHz clock. The computation acceleration is also roughly increased by the square of the clock frequency. For example, up to $P = 4$, the computations are accelerated by a factor of 4 by increasing the clock frequency from 40 MHz to 80 MHz. The results summarized in Tables I and II verify that the image reconstruction is effectively accelerated with the parallelized 2D FFT algorithm. However they also show that there is no further significant reduction in computation time when *P* is greater than 4. In fact, the computation time is approximately the same with $P = 4$ and $P = 8$ at a clock frequency of 80 MHz. With a higher parallelism, we may reach a point where the data transfer between the FPGA device and the host PC becomes a bottleneck for processing throughput, as will be further explained later. The results in these tables were obtained from spin-echo image[38] data with regular structure. We found that repeating the same measurements for RARE images[39] gave similar results, which confirms that the data re-regulation does not require additional clock cycles.

Since multiple PEs are employed in a parallel 2D FFT processor, it is interesting to compare the 2D FFT architectures built on single-pass PEs with those built on double-pass PEs. The comparison (Table III) demonstrates that employing double-pass PEs leads to a 30% reduction in DSP48 slices and a 10% reduction in BRAMs. The reduction is linearly scaled with the factor *P*, which implies that the use of double-pass PEs enables significant improvement in the utilization of the resources with large *P* values and thus enhances the processing capacities of a single-chip FPGA.

Table IV summarizes the hardware utilization for two typical 2D FFT processors with $P = 8$ for data sizes 128 × 128

TABLE III. The resource usage for the architectures of 2D FFT processors built on single- or double-pass PEs versus parallelism factor *P*.

P	1	1	2	2	4	4
Resource type	Single pass	Double pass	Single pass	Double pass	Single pass	Double pass
Slice registers	10 978	10 083	13 866	12 076	18 271	14 572
Slice LUTs	8 191	6 706	12 373	9 232	19 988	13 328
Block RAMs	27	24	52	46	99	87
DSP48s	26	18	52	36	104	72

TABLE IV. The resource utilization of the architectures of 2D FFT processors with 8 PEs built for different data sizes.

Data size	128 × 128		256 × 256		
Resource utilization	Used	Percent	Used	Percent	Total
Slice registers	21 714	4.3	21 828	4.3	508 400
Slice LUTs	25 498	10.0	25 872	10.2	254 200
Block RAMs	84	10.6	127	16.0	795
DSP48s	144	9.4	144	9.4	1 540

and 256 × 256. For both data sizes, the percentages of the slice registers, LUTs, and DSP change a little, while the block RAMs are dramatically increased with the data size. Since slices of block RAMs and DSP are rare resources, their utilization actually affects the performance of processors and also determines the scalability of the computation capabilities, given an FPGA chip. For example, in the case of data size 128 × 128, assuming that 80% is the acceptable maximal percentage, eight 2D FFT modules can be built on the FPGA we used. In theory, such an FPGA is potentially capable of processing 8 slice images simultaneously, accelerating the image processing by a factor of 8. Similarly, in the case of data size 256 × 256, the FPGA can accommodate 5 2D FFT modules that can potentially accelerate the image processing by a factor of 5.

It is important to note why the image reconstruction is no longer accelerated in proportion to increasing P when P exceeds 4 (Tables I and II). This behavior is due primarily to the design method with which all non-arithmetic operations inside a processor are driven by available data. In these situations, the limited speeds of data transfer to the processors result in the processors remaining idle for many clock cycles. Therefore, in applications where the input data are transferred from the host computer, achievable maximum throughput of the processors is primarily limited by the speed of data transfer between the host computer and the FPGA device. If the processors are used for real-time processing of parallel MRI, however, then the input streams will come directly from multiple detection coils and will be directly fed to the multiple input FIFO buffers on the FPGA without the data demultiplexing. In these applications, the data-acquisition speed may become a limiting factor for further image-reconstruction acceleration.

VI. CONCLUSION

In this paper, we present a method for designing FPGA processors capable of parallelizing a 2D FFT algorithm. The design of the processor architectures not only takes into consideration the organization of the distributed computations and simplified data communication but also is customized to the characteristics of MRI data streams. The design features simpler internal data communications, synchronization, and more efficient usage of hardware resources. It is expected that, compared to commercially available FFT hardware processors, our FPGA processors will be more suitable and flexible for MRI applications.

ACKNOWLEDGMENTS

This work was supported by NIH Grant No. R21EB 024852.

[1]P. Mansfield, "Multi-planar imaging formation using NMR spin-echo," J. Phys. C: Solid State Phys. **10**, L55–L58 (1977).
[2]P. Mansfield and B. Chapman, "Active magnetic screening of coils for static and time-dependent magnetic field generation in NMR imaging," J. Phys. E: Sci. Instrum. **19**, 540 (1986).
[3]D. Gallichan, A. Scholz, T. E. Behrens, M. D. Robson, and K. L. Miller, "Addressing a systematic vibration artifact in diffusion-weighted MRI," Hum. Brain Mapp. **31**, 193–202 (2010).
[4]D. L. Price, J. P. De Wilde, A. M. Papadaki, J. S. Curran, and R. I. Kitney, "Investigation of acoustic noise on 15 MRI scanners from 0.2 T to 3 T," J. Magn. Reson. Imaging **16**(5), 497–510 (2001).
[5]R. J. Ordidge and I. D. Cresshull, "The correction of transient B0 field shifts following the application of pulsed gradients by phase correction in the time domain," J. Magn. Reson. **69**, 151–155 (1986).
[6]S. Crozier, C. D. Eccles, F. A. Beckey, J. Field, and D. M. Doddrell, "Correction of eddy-current-induced B0 shifts by receiver reference-phase modulation," J. Magn. Reson. **97**, 661–665 (1992).
[7]E. M. Haacke and J. L. Patrick, "Reducing motion artifacts in two-dimensional Fourier transform imaging," Magn. Reson. Imaging **4**, 359–376 (1986).
[8]J. Maclaren, M. Herbst, O. Speck, and M. Zaitsev, "Prospective motion correction in brain imaging: A review," Magn. Reson. Med. **69**(3), 621–636 (2013).
[9]D. J. Jensen, W. W. Brey, J. L. Delayre, and P. A. Narayana, "Reduction of pulsed gradient setting time in the superconducting magnet of a magnetic resonant instrument," Med. Phys. **14**, 859–862 (1987).
[10]N. G. Papadakis, K. M. Martin, J. D. Pickard, L. D. Hall, T. A. Carpenter, and C. L. Huang, "Gradient preemphasis calibration in diffusion-weighted echo planar imaging," Magn. Reson. Med. **44**, 616–624 (2000).
[11]T. Onodera, S. Matsui, K. Sekihara, and H. Kohno, "A method of measuring field-gradient modulation shapes. Application to high-speed NMR spectroscopic imaging," J. Phys. E: Sci. Instrum. **20**, 416–419 (1987).
[12]O. Josephs, R. Deichmann, and R. Turner, "Trajectory measurement and generalized reconstruction in rectilinear EPI," NeuroImage **11**(5, Suppl. 1), 543 (2000).
[13]C. Barmet, N. D. Zanche, and K. P. Pruessmann, "Spatiotemporal magnetic field monitoring for MR," Magn. Reson. Med. **60**, 187–197 (2008).
[14]Y. Duert, B. J. Wilm, B. E. Dietrich, S. J. Vannesjo, C. Barmet, T. Schmid, D. O. Brunner, and K. P. Pruessmann, "Real-time feedback for spatiotemporal field stabilization in MR systems," Magn. Reson. Med. **73**, 884–893 (2015).
[15]P. Guillem and X. Lei, "GPU computing in medical physics: A review," Med. Phys. **38**(5), 2685 (2011).
[16]See http://www.bertendsp.com for Berten Digital Signal Processing: "GPU vs FPGA performance comparison," BWP001 v1.0.
[17]D. E. Dudgeon and R. M. Mersereau, *Multidimensional Signal Processing* (Prentice-Hall, 1983).
[18]H. Kee, N. Petersen, J. Kornerup, and S. S. Bhattacharyya, "System generation of FPGA-based FFT implementation," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (IEEE, 2008).
[19]H. Kee, N. Petersen, J. Kornerup, and S. S. Bhattacharyya, "Resource-efficient acceleration of 2-Dimensional fast Fourier transform computations on FPGAs," in *Proceedings of the International Conference on Distributed Smart Cameras* (IEEE, 2009).
[20]S. A. Christe, M. Vignesh, and Kandaswamy, "An efficient FPGA implementation of MRI image filtering and tumor characterization using Xiline system generator," Int. J. VLSI Des. Commun. Syst. **2**(4), 95 (2011).
[21]S. Hasan, S. Boussaka, and A. Yakovle, "FPGA-based architecture for a generalized parallel 2-D MRI filtering algorithm," Am. J. Eng. Appl. Sci. **4**(4), 566 (2011).
[22]C. L. Yu, J. S. Kim, L. Deng, S. Kestur, V. Narayanan, and C. Chakrabarti, "FPGA architecture for 2D Fourier transform based on 2D decomposition for large-size data," J. Signal Process. Syst. **64**, 109 (2011).
[23]See http://www.xilinx.com/support/documentation/ip_documentation/ for Xilinx, LogiCORE IP fast Fourier transform v9.0: Product guide for Vivado design suite.
[24]L. Li and A. M. Wyrwicz, "Design of an MR image processing module on an FPGA," J. Magn. Reson. **255**, 51–58 (2015).
[25]L. Li and A. M. Wyrwicz, "Modularized architecture of address generation units suitable for real-time processing MR data on an FPGA," Rev. Sci. Instrum. **87**(6), 063705 (2016).
[26]S. D. Kaushik, C. H. Huang, J. R. Johnson, R. W. Johnson, and P. Sadayappan, "Efficient transposition algorithms for large matrices," in *Proceedings of Supercomputing* (IEEE, 1993), pp. 656–665.
[27]C. Galuzza, C. Gou, H. Calderon, G. N. Gaydadjiev, and S. Vassiliadis, "High-bandwidth address generator unit," J. Signal Process. Syst. **57**(1), 33 (2009).
[28]K. E. Vistnes and O. Sorasen, "Reconfigurable address generators for stream-based computation implemented on FPGAs," in *Parallel and Distributed Processing Symposium* (IEEE, 2005).

[29]R. W. Hartenstein and H. Reinig, "Novel sequencer hardware for high-speed signal processing," in Workshop on Design Methodologies for Microelectronics, 1995.
[30]R. Espasa, M. Valero, and J. E. Smith, "Vector architectures: Past, present and future," in *Proceedings of the 12th International Conference on Supercomputing* (ICS, 1998), pp. 425–432.
[31]J. Corbal, R. Espasa, and M. Valero, "Command vector memory systems: High performance at low cost," in *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques* (IEEE, 1998), p. 68.
[32]B. K. Mathew, S. A. KcKee, J. B. Carter, and A. Davis, "Design of a parallel vector access unit for SDRAM memory systems," in *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture* (IEEE, 2000), pp. 39–48.
[33]B. K. Mathew, S. A. KcKee, J. B. Carter, and A. Davis, "Algorithmic foundations for a parallel vector access memory systems," in *Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures* (IEEE, 2000), pp. 156–165.
[34]D. H. Baily, "Vector computer memory bank contention," IEEE Trans. Comput. **36**(3), 293 (1987).
[35]G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "Multimedia rectangularly addressable memory," IEEE Trans. Multimedia **8**(2), 315 (2006).
[36]D. H. Lawie and C. R. Vora, "The prime memory system for array access," IEEE Trans. Comput. **31**(5), 435 (1982).
[37]P. J. Won, "Multiaccess memory system for attached SIMD computer," IEEE Trans. Comput. **53**(4), 439 (2004).
[38]W. A. Edelstein, J. M. S. Hutchison, G. Johnson, and T. Redpath, "Spin-warp NMR imaging and applications to human whole-body imaging," Phys. Med. Biol. **25**, 751 (1980).
[39]J. Hennieg, J. A. Nauerth, and H. Friedburg, "RARE imaging: A fast imaging method for clinical MR," Magn. Reson. Med. **3**, 823 (1986).