<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Design considerations of real-time adaptive beamformer for medical ultrasound research using FPGA and GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Chen, J; Yu, ACH; So, HKH</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>The 2012 International Conference on Field-Programmable Technology (FPT 2012), Seoul, South Korea, 10-12 December 2012. In Conference Proceedings, 2012, p. 198-205</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2012</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/189830">http://hdl.handle.net/10722/189830</a></td>
</tr>
</tbody>
</table>

**Rights**: IEEE International Conference on Field-Programmable Technology Proceedings. Copyright © IEEE Computer Society.; ©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
Design Considerations of Real-time Adaptive Beamformer for Medical Ultrasound Research using FPGA and GPU

Junying Chen 1, Alfred C. H. Yu 2, Hayden K.-H. So 1

1 Department of Electrical and Electronic Engineering, The University of Hong Kong
2 Medical Engineering Program, The University of Hong Kong

{jychen, hso}@eee.hku.hk, alfred.yu@hku.hk

Abstract—Adaptive beamforming has been well considered as a potential solution for improving the imaging quality of medical ultrasound systems. Despite the promised improvement in lateral resolution, image contrast and imaging penetration, the use of adaptive beamforming is substantially more computationally demanding than conventional delay-and-sum beamformers. While a dedicated hardware solution may be able to address the computational demand of one particular design, the need for an efficient algorithm exploration framework demands a platform solution that is high-performance and easily reprogrammable. To that end, the use of FPGA and GPU for implementing real-time adaptive beamforming on such platform has been explored. The results are evaluated quantitatively in terms of performance and image quality, and qualitatively with respect to ease of system integration and ease of use. In our test cases, both FPGA- and GPU-based solutions achieved real-time throughput exceeding 80 frames-per-second, and over 38x improvement when compared to our baseline CPU implementation. While the development time on GPU platform remains much lower than its FPGA counterpart, the FPGA solution is effective in providing the necessary I/O bandwidth to enable an end-to-end real-time reconfigurable image formation system.

I. INTRODUCTION

The use of adaptive beamforming (ABF) techniques has been proposed by researchers to improve image quality of medical ultrasound imaging systems recently. When compared to a conventional delay-and-sum (DAS) beamformer that utilizes predefined apodization weights, an adaptive beamformer computes the apodization weights during run-time in response to the input data. As a result, better image contrast and resolution, as well as better imaging penetration depth have been observed.

One particular approach of adaptive beamforming is the minimum-variance (MV) technique [1], [2], [3], [4] that works by minimizing the variance among the apodization weights. Figure 1 illustrates the benefits of MV adaptive beamforming. As shown in the figure, for instance, at 30 mm imaging depth, DAS beamforming with Hamming weighting fails to resolve the two very close point targets, making the two points look like a short line instead. But on the other hand, the two point targets are clearly distinguishable at the same depth using MV beamforming.

Despite the promising results, the real-time implementation of MV adaptive beamforming techniques remains a challenge due to the much-increased computational requirements. Since medical ultrasound imaging is often used to provide real-time diagnoses by displaying the images along the way the patients are being scanned, the real-time definition in medical ultrasound imaging is the human perception of a continuous image video with an image frame rate in a range approximately from 20 frames-per-second (fps) to 100 fps.

Furthermore, in order to experiment with different complex algorithms, such as MV adaptive beamforming, on real-world diagnostic targets, a reconfigurable/reprogrammable research platform must be employed, ruling out the use of application-specific integrated circuit (ASIC) implementations. While a simple processor-based solution may allow convenient algorithm exploration via software development environment, real-time performance is usually sacrificed. As a result, the merits of such advanced algorithms are often overshadowed by the lack of real-time full system experimental validations.

In the past few years, specific-purpose real-time medical ultrasound research machines using FPGAs have been presented, for example, a real-time synthetic aperture ultrasound scanner [5] and a breast ultrasound computed tomography...
system [6]. Besides, a general-purpose medical ultrasound reconfigurable system [7] was built for basic ultrasound imaging algorithms like DAS.

In this project, we explored the implementation of MV adaptive beamforming algorithm using the CPU, FPGA and GPU on the medical ultrasound research platform that we have been constructing in-house (Figure 2). Apart from the basic requirement of meeting real-time performance, we also evaluated the image quality, ease of use, and actual system integration involved when all parts of the system, from the ultrasound transducer to the user display, were considered.

As such, we consider the main contribution of this work is in the following aspects:

- The designs and implementations of MV adaptive beamforming for medical ultrasound using FPGA and GPU that achieve real-time performance requirement are presented and compared;
- A tradeoff study of the two implementations from a system perspective, considering image quality, design productivity, and end-to-end I/O requirements, showing the unique capability of FPGA serving both as a system integration component and as a computing device at the same time.

In the next section, background information about MV adaptive beamforming will be first introduced. The target research platform and the implementations of the beamforming algorithm using the FPGA and GPU on this platform will follow in Section III. The implementation results and evaluation of the tradeoffs will be presented in Section IV. We conclude and discuss future work in Section V.

II. MV ADAPTIVE BEAMFORMING ALGORITHM

In medical ultrasound, the images consist of image pixels organized in rows and columns. The determination of the value of one image pixel in DAS and MV beamforming is shown in Figure 3, which demonstrates that the key difference between an adaptive beamforming system and a traditional delay-and-sum (DAS) beamforming system rests on the applied apodization weights on the delayed channel data.

Traditionally, scanline-based imaging system utilizes DAS beamforming (Figure 3(a)) with fixed apodization weights [8]. In such systems, digitized samples of the received echoes are first applied with appropriate delays, then multiplied by fixed apodization weights, and finally summed up to form an amplitude estimate for a particular image pixel. On the other hand, in an adaptive beamforming system shown in Figure 3(b), the apodization weights for delayed digitized samples are calculated every time when a new set of delayed samples arrive. In an MV adaptive beamforming system, the weights are computed such that the variance among the channels is minimized. Here, only a summary on the MV adaptive beamforming algorithm is described. Please refer to [1], [9] for more details.

A. Regular MV algorithm

As shown in Figure 4, when applying sub-aperture averaging in MV beamforming, a receive aperture is formed by $M$ consecutive channels, and it is segmented into a set of lag-one overlapping sub-apertures, each of which is constructed by $L$ continuous channels. As a result, $(M - L + 1)$ sub-apertures are formed. With sub-aperture averaging, the covariance matrix for pixel $p$ can then be estimated as:

$$
R(p) = \frac{1}{M - L + 1} \sum_{k=1}^{M-L+1} x_k(p)x_k^H(p),
$$

\[1\]
where $x_k(p)$ is a $L \times 1$ vector of delayed echo samples in the $k^{th}$ sub-aperture, which starts from $k^{th}$ channel to $(k+L-1)^{th}$ channel inside the receive aperture. In other words, $x_k(p)$ is the assembly of $k^{th}$ to $(k+L-1)^{th}$ elements in $x(p)$, while $x(p)$ is a $M \times 1$ vector, which refers to the set of delayed pre-beamform samples used to form one pixel. After $R(p)$ is calculated, adaptive apodization weights are calculated based on MV optimization criteria [1], [9]:

$$w(p) = \frac{R^{-1}(p) a}{a^HR^{-1}(p)a},$$  \hfill (2)

where $a$ is the steering vector with simply all ones, because the channel data being processed are already delayed. Finally, the amplitude estimate of the image pixel $p$ is obtained by averaging the weighted sum of focus-delayed echo signals from all sub-aperture channels.

$$z(p) = \frac{1}{M-L+1}\sum_{k=1}^{M-L+1} w^H(p)x_k(p).$$ \hfill (3)

The correct estimation of the covariance matrix is imperative as it is a key factor affecting the resulting image quality. Sub-aperture averaging is demonstrated as the main measure to handle coherent ultrasound echoes [2], so that high image quality is acquired and the MV beamformer is also robust. When doing sub-aperture averaging, $L = M/4$ is usually used to fulfill the high image quality and robust beamforming requirements.

### B. Toeplitz structured MV algorithm

When the approximation of spatial stationariness of the imaging field signals is made, the estimated covariance matrix can be assumed to have a Toeplitz structure (A matrix which has constant left-to-right descending diagonals). Built upon this concept, the covariance matrix becomes [10]:

$$\hat{R}(p) = \begin{bmatrix}
\hat{R}_0 & \hat{R}_1 & \cdots & \hat{R}_{L-1} & \hat{R}_L \\
\hat{R}_1 & \hat{R}_0 & \cdots & \hat{R}_{L-2} & \hat{R}_{L-1} \\
\hat{R}_2 & \hat{R}_1 & \cdots & \hat{R}_{L-3} & \hat{R}_{L-2} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\hat{R}_{L-2} & \hat{R}_{L-3} & \cdots & \hat{R}_1 & \hat{R}_2 \\
\hat{R}_{L-1} & \hat{R}_{L-2} & \cdots & \hat{R}_0 & \hat{R}_1 \\
\hat{R}_L & \hat{R}_{L-1} & \cdots & \hat{R}_1 & \hat{R}_0
\end{bmatrix}.$$ \hfill (4)

Therefore, only $L$ elements of $\hat{R}(p)$, $\{\hat{R}_0, \hat{R}_1, \hat{R}_2, \ldots, \hat{R}_L\}$, need to be calculated. The estimated elements along the $m^{th}$ descending diagonal of the Toeplitz structured $\hat{R}(p)$ are calculated by averaging the components along relevant descending diagonal of the regular $R(p)$ [10]:

$$\hat{R}_m = \frac{1}{L-m}\sum_{l=1}^{L-m} R_{l+1+m},$$ \hfill (5)

where $m = 0, 1, \ldots, L-1$, which represents the value difference between row and column indices of elements in $R(p)$.

![A cyst imaging phantom](image1.png)

![MV beamforming output image](image2.png)

Fig. 5. Simulated cyst imaging phantom and MV beamforming imaging result. (a) presents the simulated cyst phantom with 4 cysts. They are placed at 15 mm, 25 mm, 35 mm, 50 mm. (b) demonstrates the output image using MV beamforming.

The benefit of forming the covariance matrix using a Toeplitz formation is to save computation time for matrix inverse, which is proven to reduce the computation complexity as compared to the matrix inverse calculation in regular MV algorithm, especially in cases with large $M$ and $L$.

### III. IMPLEMENTATION

#### A. Target Platform

The goal of this project is to develop a reconfigurable solution for real-time complex medical ultrasound imaging research. As such, we have implemented the MV adaptive beamforming algorithm on the FPGA and GPU in the target system, and compared their performance against a baseline implementation using the onboard CPU.

Figure 2 depicts a high-level block diagram of our target platform. An ultrasonic transducer with 128 linear array elements is used for both transmitting and receiving echoes. The received echoes are digitized into 12-bit data at 40 MHz. In our current real-time implementation, 16 channels of the digitized data are streamed to the ML605 FPGA board via two FMC connections. The processed data is passed to the host PC system via its PCIe connection. A GTX 480 GPU is plugged into the host PC, next to an Intel Core 2 Quad CPU Q6600.

To verify and evaluate the implementations on FPGA, GPU and CPU, synthesized ultrasound pulse-echo samples were generated using the Field II simulator [11], [12]. The simulated scenario was a scanline-based scanning over a cyst phantom with 4 cysts of different diameters at different imaging depths, as illustrated in Figure 5(a). The point scatterers in the imaging region were of Gaussian distributed amplitudes and of 100 scatterers/mm$^2$ average density, so that the speckle pattern of the imaging view could be regarded as fully developed [8]. Besides, on the simulated transmitter end, a 2-cycle ultrasound pulse centering at 5 MHz that propagated at a speed of 1540 m/s in ultrasound imaging field was simulated, and
such transmitted pulse was focused at 25 mm depth using Hamming apodization and fired 127 times resulting in 127 image scanlines. Furthermore, on the simulated receiver end, digitized echo samples were received using up to all 128 channels, so as to explore computing features of GPU and CPU implementations whose number of data input channels was not constrained to 16 channels as FPGA’s (Section III-B).

B. Design Overview

Figure 6 depicts the high-level block diagram of the target MV adaptive beamformer that forms the basis for both the FPGA and GPU designs. In both cases, the beamformer took \( M \) receive channels as its inputs to generate an amplitude estimation of 1 image pixel.

In the case of FPGA real-time streaming implementation, the entire MV adaptive beamforming algorithm was implemented within the FPGA, outputting only the amplitude estimation to the PC for final image display. Due to limited available connection on the current FPGA board, only 16 channels of data were streamed in real-time. Hence, \( M \) was constrained to 16 in current design. Moreover, only one datapath was implemented on the FPGA.

In the case of GPU implementation, the digitized data was buffered and sent to the GPU for beamforming and display. Hence, GPU design could take up to the full 128 channels of simulated data as input. Furthermore, depending on the GPU capability, more than 1 pixel datapaths may be executed in parallel.

Each receive channel streamed 12-bit digitized echo samples to the delay calculation block, forming an \( M \times 1 \) vector of delayed echo samples as output. The purpose of the this block was to equalize the delay among the receive channels due to the differences in receive path lengths.

The delayed sample vector must subsequently be multiplied by the adaptive apodization weights. In the FPGA design, the sample vector was delayed by \( T \) time units to match the latency of the adaptive weight calculation block. The value of \( T \) was fixed, since the adaptive weight calculation block was operated in fixed clock cycles. No such delay was needed in the GPU implementation, but the delayed sample vector needs to be stored in GPU shared memory to wait until adaptive weight calculation block finished.

Finally, the pixel amplitude estimation block output a 32-bit pixel value at a time, because single precision floating point representation was employed. It first multiplied \((M – L + 1)\) segmented delayed sample vectors \( x(k) \), where \( k = 1, 2, \ldots, (M – L + 1) \), and their adaptive weights. The results of these \((M – L + 1)\) pixel value estimates were subsequently averaged to obtain the final pixel amplitude value output.

C. Adaptive Weight Calculation

Although the implementation of the MV algorithm for adaptive weighting could be carried out straightforward without computation considerations, probability theories and linear algebra theories can be made use of to optimize the detailed implementation and reduce the computation operations. The integration of the mathematical theories into the implementation will be described in three parts in the following.

The adaptive weight calculation block in Figure 6 contained the core computation of the MV adaptive beamformer. It was consist of three major units: covariance matrix calculation, linear equation solver, and final weight calculation step. Here, we elaborate on the inner working of these blocks.

1) Covariance matrix calculation: Derived from (1), the covariance matrix element \( R_{ij} \) is calculated as:

\[
R_{ij} = \frac{1}{M - L + 1} \sum_{n=1}^{M-L+1} x(i+n)x(j+n).
\]  

As the input digital sample data from ADCs are real numbers, the covariance matrix is a symmetric matrix [13], which has the following property:

\[
R^T(p) = R(p). 
\]  

As a result, only the diagonal elements and the lower (L) or upper (U) triangular matrix elements of the covariance matrix need to be calculated. Therefore, \( L \times (L + 1)/2 \) calculations are needed in stead of \( L \times L \) calculations. Taking the advantage of the symmetry makes the covariance matrix implementation nearly twice faster.

While \( R(p) \) is calculated for regular MV algorithm, an extra step needs to perform to obtain the approximated \( \tilde{R}(p) \) array for Toeplitz structured MV algorithm, which is described in Section II-B.

2) Linear equation solver: As shown in the weight calculation (2), \( R^{-1}(p) \) has to be calculated. But in (2), every \( R^{-1}(p) \) is multiplied by a vector \( a \). As a result, a linear equation solver which outputs \( R^{-1}(p)a \) can take over the places of the matrix inverse unit and the matrix multiplication unit. The solver is used to solve a system of linear equations like:

\[
R^{-1}(p)y = a. 
\]  

Using a system solver reduces extra operation time and storage resources.

The covariance matrix has positive-semidefinite and symmetric properties [13], hence, Cholesky decomposition [14] is applicable to this weight solver in regular MV algorithm. Cholesky decomposition is derived from Gaussian Elimination, but halves the decomposition operations and is more stable than \( LU \) decomposition which is the matrix form of Gaussian Elimination. \( LDL^T \) decomposition form of the Cholesky decomposition was adopted. Besides, for Toeplitz structured MV algorithm, a Toeplitz system solver was utilized [14].

Since the weight solver was iterative, the iterations cannot be parallelized. But inside the iterations, parallelization was achievable. For example, the \( L \) matrix was formed column by column and only one column in one iteration, but the element calculations within each column of \( L \) could be parallelized.
3) Final weight calculation step: The final step of the weight calculation is to calculate:
\[
    w(p) = \frac{y}{a^H y}, \quad (9)
\]
As \(a\) is a vector of ones, \(a^H y\) can be calculated as:
\[
    a^H y = \sum_{n=1}^{L} y_n. \quad (10)
\]
Therefore,
\[
    w(p) = \frac{y}{\sum_{n=1}^{L} y_n}. \quad (11)
\]
This step was the same for regular MV algorithm and Toeplitz structured MV algorithm.

D. FPGA Design

As the target of this work is a research platform, the ease of implementing new image forming algorithms is one of the important metrics of success of the work. For that, we have chosen to implement the entire adaptive beamforming design using Simulink and Xilinx System Generator for DSP (v13.4) [15], [16]. The top level block diagram of the design is shown in Figure 7.

Single-precision floating point operation units were used in the FPGA design when the calculations involve fraction numbers. It was to match as closely as possible in terms of precision against the GPU and CPU implementations. The floating point operator blocks were made fully pipelined by setting the latency values of the operator blocks to their maximum values [17], so as to achieve fast FPGA processing clock frequency.

Apart from streaming data processing, some blocks were handled simultaneously in FPGA design. For example, the delay information calculation block and data load-in block were running at the same time. While in GPU implementation and CPU implementation, these two blocks were executed one after one.

E. GPU Design

We present the GPU design implemented using CUDA C programming language here, because in our experiments, processing speed of OpenCL implementation was a little slower than that of CUDA C implementation. The CUDA toolkit used in the implementation was v4.2.

In our current GPU implementation, the entire MV adaptive beamforming algorithm was implemented as one GPU compute kernel which was divided into s set of compute blocks. Each compute block was therefore responsible for producing the amplitude estimation of one image pixel. Multiple compute blocks were then executed in parallel as long as there were available processing resources on the target GPU. Most experimental test cases in our design indicated that a compute block size having 32 threads produced the optimal performance. Additional performance improvements were made referring to [18].

Unlike the FPGA implementation that receives streaming data from each receive channels, the GPU implementation takes as input a matrix of simulated all-channel digitized ultrasound echo samples. The use of simulated data is needed as there is currently inadequate I/O bandwidth for streaming data to GPU, which also highlights one of the disadvantage of the GPU implementation. This is because in our current design, 3000 samples from each receive channel are used to accommodate the delay difference among the 128 receive channels, generating an image scanline output of 1000 pixels.

IV. RESULTS & DISCUSSION

The beamforming algorithm implementations on FPGA and GPU are evaluated from four aspects – speed, image quality, ease of use, and integration with the rest of the system.

A. Performance

As the final goal of the project is to integrate any new imaging algorithm with the rest of the research ultrasound system, it is imperative for the system to perform under
real-time performance constraints. There are a number of constraints that must be met to ensure real-time processing.

First, at the output of the beamformer, in order to provide smooth video output of the images, a throughput between 20 – 100 fps is targeted. The number of image scanlines in each image frame equals to the number of transmit firing times for one image frame. The number of transmit firing times per image frame is defined in the transducer transmission sequence. Hence in this work, we target an image frame with 127 scanlines, each with a depth of 1000 pixels. Therefore, the beamforming algorithm must be able to perform at least at a speed of $20 \times 127 \times 1000 = 2.54$ mega pixels-per-second. Furthermore, the FPGA must be able to capture each scanline from the ADCs at a rate higher than the pulse repetition frequency of the ultrasound scanner, which is usually set to 5 KHz. Finally, the FPGA must also be able to sink the data from the ADCs fast enough for continuous streaming of data. In our current implementation, the ADCs run at 40 MHz, generating a 12-bit data for each of the channels.

Table I shows the performance of the FPGA implementations with and without the use of Toeplitz approximation. Note that, in the table, the beamformer data output rate is always 1/10 of the FPGA clock frequency. This is because there are 10 clock cycles interval between every two beamformation pixel value outputs in current FPGA designs. Also, the scanline data input rate is 1/10000 of the clock frequency, because the time interval between the data stream-in timestamps of every two scanline data has been arbitrarily chosen to be 10000 clock cycles. As shown in the table, both FPGA implementations fulfill the requirements of a real-time ultrasound imaging system.

Table II shows a summary of FPGA and GPU implementation performance and compares them to the baseline CPU implementation, which was implemented on an Ubuntu 10.04 operating system with gcc compiler v4.4.3. In the case of CPU and GPU, an additional set of data with $M = 128$, $L = 32$ is included, which represents the final real-time beamformer we are targeting. Unfortunately, the current setup limits us to feeding only 16 channels of data to the FPGA. From the table, it is clear that both FPGA and GPU perform equally well.

Both FPGA and GPU implementations outperform the CPU implementation by over $33 \times$. Note that for cases with small design size ($M = 16$), contrary to the prediction from theoretical study, the performance of the regular MV algorithm implementation in fact outperforms that of the Toeplitz MV algorithm. It is because the overhead of implementing the Toeplitz MV algorithm can only be amortized when the problem size increases.

Figure 8 shows the advantage of Toeplitz MV implementation when the design is large. As shown in the figure, given a larger value of $M$, the performance advantage of the Toeplitz MV algorithm implementation over regular MV implementation increases as the value of $L$ increases.

In the case of GPU implementation, we assume that the data from the ADCs are pre-captured and are processed offline. This is due to the lack of I/O bandwidth to transfer raw data from the ADCs to the desktop computer in our current setup (see Section IV-D). The performance requirement for GPU implementation is therefore limited to the 20 fps video frame rate.
TABLE II
PERFORMANCE OF FPGA AND GPU IMPLEMENTATIONS COMPARED TO CPU

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CPU (fps)</th>
<th>FPGA (fps)</th>
<th>GPU (fps)</th>
<th>FPGA speedup</th>
<th>GPU speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular MV</td>
<td>2.215</td>
<td>86.977</td>
<td>84.538</td>
<td>39x</td>
<td>38x</td>
</tr>
<tr>
<td>Toeplitz MV</td>
<td>2.184</td>
<td>71.213</td>
<td>77.888</td>
<td>33x</td>
<td>36x</td>
</tr>
</tbody>
</table>

\[ M = 16, L = 4 \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CPU (fps)</th>
<th>FPGA (fps)</th>
<th>GPU (fps)</th>
<th>FPGA speedup</th>
<th>GPU speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular MV</td>
<td>0.017</td>
<td>–</td>
<td>2.681</td>
<td>158x</td>
<td></td>
</tr>
<tr>
<td>Toeplitz MV</td>
<td>0.020</td>
<td>–</td>
<td>3.126</td>
<td>156x</td>
<td></td>
</tr>
</tbody>
</table>

\[ M = 128, L = 32 \]

754 standardized single-precision floating-point numbers were used in all cases in FPGA [19], GPU [20] and CPU, no major difference in image quality was expected among the implementations. The difference in final image quality was mainly due to numerical error arose from different sequences of floating point operations in parallelized processes. In our experiments, the image quality performance on different computing devices were similar, with very little error.

Furthermore, the difference between regular MV output image and Toeplitz MV output image is shown in Figure 9. Despite of the approximation performed in the Toeplitz MV algorithm, as indicated in the figure, the quality of the resulting image remains acceptable, which has also been illustrated in [10]. We therefore anticipate that Toeplitz MV will be an important approximation technique for our future MV systems.

C. Ease of Use

As a research platform, it is important that users with little prior knowledge in programming accelerators may be able to make maximum use of the system. The design environment should allow users to focus on algorithm development and be able to explore a large design space easily. Comparing the GPU and FPGA as accelerators, the GPU has a clear advantage in terms of ease of use. In this particular project, the CUDA C developing environment was used to develop our GPU implementations. About 2 months of development time with one PhD student was spent to achieve real-time performance on GPU from scratch. Moreover, the GPU implementation allows a wide design space to be explored by simple changing of parameters.

On the FPGA front, we have opted to rely on Simulink and Xilinx System Generator for development. Our experience has shown that its integration with Matlab has indeed been useful in enabling incremental algorithm development on FPGA. Started with the original Matlab beamforming algorithm, the FPGA design was developed incrementally by converting blocks using System Generator blockset. At the end, about 5 months were spent by one PhD student to have the FPGA design fully functioning at real-time. Furthermore, despite the flexibility offered by the System Generator blockset, it remains difficult to explore a large design space as many of the hardware structure are specifically designed for certain design parameters.
D. System Integration

A major drawback with GPU as an accelerator rests with its limited I/O bandwidth. Although commercial GPUs are connected to the PC host system via high-speed PCIe connections, they possess no other external I/O connections by design. This inherent design choice has made it very difficult to fully take advantage of the computing power on board. In our case, although the GPU may perform beamforming tasks well over the real-time requirement for video display, it remains challenging to sustain a real-time streaming of data from the transducer front-end to the GPU. If all ADC output data is streamed to the GPU in its entirety, with 128 channels sampling at 40 MHz with 12-bit precision, almost 8 GB/s bandwidth is needed. This is right at the limit of the common x16 PCIe v2 line rate, making it impractical for actual implementation. Although it is possible to further reduce the bandwidth requirement if enough buffering is available to buffer the ADC output, it remains unclear if that can be easily addressed by novel algorithm designers.

On the other hand, the FPGA is already placed on the signal path with adequate I/O connections. On one end, it operates at line rate with the ADCs. On the other end, it is connected to the PC and the rest of the system via high-speed PCIe connection. As a result, it is capable of serving both as a low-level data manipulating hardware and a computational accelerator at the same time. This unique characteristics has made the FPGA much better positioned to integrate with the rest of the system. Our experience has shown that shifting some of the acceleration task to the FPGA may significantly reduce the I/O requirement to/from the host PC. In addition, processing on FPGA frees up compute resources on the GPU which may be used for later processing stages.

V. CONCLUSIONS

In this paper, we have presented the accelerations of regular MV beamforming and Toeplitz structured MV beamforming implementations using FPGA and GPU on our medical ultrasound research platform. The merits of the two implementations are compared with respect to speed, image quality, ease of use, and system integration. Both FPGA and GPU beamformers are capable of performing real-time ultrasound imaging using adaptive MV beamforming techniques with similar image qualities. Compared to the CPU baseline design, 30x to 158x speedup has been achieved. Qualitatively, the development effort associated with the GPU implementation remains much lower than that for FPGA implementation. However, the use of Simulink and Xilinx System Generator has demonstrated its merit for algorithm designers. Also, the FPGA has an advantage over GPU implementation as the FPGA is well connected to the signal path, thereby allowing it to more easily sustain real-time data streaming. Another point worth noting is that, as the FPGA and GPU accelerators are built within our reconfigurable medical ultrasound imaging research platform, they can be readily leveraged to investigate the real-time performance of other advanced ultrasound imaging algorithms in addition to the adaptive beamforming algorithm demonstrated in this work.

Currently, our design is connected to 16 channel inputs only as limited by the FPGA board. Furthermore, in the future, we intend to overcome the hardware limitation by using custom-made FPGA connection modules. Furthermore, we will be investigating the tradeoff of hybrid computing using both the GPU and FPGA, which we anticipate as design size increases.

ACKNOWLEDGMENT

This work is supported in part by Hong Kong Innovation and Technology Fund (ITS/292/11), and in part by the Research Grants Council of Hong Kong, project GRF 716510.

REFERENCES


205