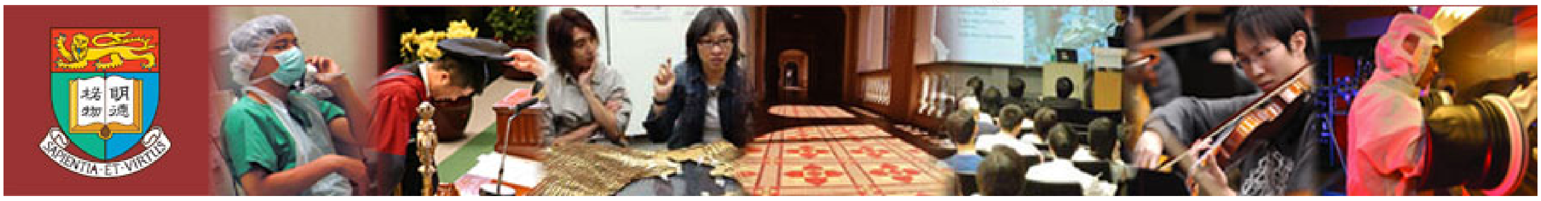

10.1109/ULTSYM.2010.0485

Real-Time GPU-Based Software Beamformer Designed for Advanced Imaging Methods Research

Billy Y. S. Yiu, Ivan K. H. Tsang, and Alfred C. H. Yu
Medical Engineering Program,
The University of Hong Kong, Pokfulam, Hong Kong SAR

Corresponding Email: alfred.yu@hku.hk

Abstract—**High computational demand is known to be a technical hurdle for real-time implementation of advanced methods like synthetic aperture imaging (SAI) and plane wave imaging (PWI) that work with the pre-beamform data of each array element. In this paper, we present the development of a software beamformer for SAI and PWI with real-time parallel processing capacity. Our beamformer design comprises a pipelined group of graphics processing units (GPU) that are hosted within the same computer workstation. During operation, each available GPU is assigned to perform demodulation and beamforming for one frame of pre-beamform data acquired from one transmit firing (e.g. point firing for SAI). To facilitate parallel computation, the GPUs have been programmed to treat the calculation of depth pixels from the same image scanline as a block of processing threads that can be executed concurrently, and it would repeat this process for all scanlines to obtain the entire frame of image data - i.e. low-resolution image (LRI). To reduce processing latency due to repeated access of each GPU's global memory, we have made use of each thread block's fast-shared memory (to store an entire line of pre-beamform data during demodulation), created texture memory pointers, and utilized global memory caches (to stream repeatedly used data samples during beamforming). Based on this beamformer architecture, a prototype platform has been implemented for SAI and PWI, and its LRI processing throughput has been measured for test datasets with 40 MHz sampling rate, 32 receive channels, and imaging depths between 5-15 cm. When using two Fermi-class GPUs (GTX-470), our beamformer can compute LRIs of 512-by-255 pixels at over 3200 fps and 1300 fps respectively for imaging depths of 5 cm and 15 cm. This processing throughput is roughly 3.2 times higher than a Tesla-class GPU (GTX-275).**

Keywords—software beamformer, parallel processing, graphics processing units, synthetic aperture imaging, plane wave imaging.

I. Introduction

Ultrasound imaging is conventionally based upon a pulse-echo sensing mechanism that sequentially acquires image data over a group of beam lines. This imaging paradigm can typically achieve a frame rate of about 30-40 Hz in existing scanners [1]. To increase the imaging frame rate substantially without concomitantly reducing the imaging view or image quality, it is necessary to make use of alternative imaging paradigms. One approach is to use plane wave imaging (PWI) principles to simultaneously transmit all the beam-lines and then perform parallel receive beamforming on receive [2]. Another fast imaging method that has been proposed is the use of the synthetic aperture imaging (SAI) technique that employs unfocused point-source firings on transmit and detect echoes at all transducer channels on receive [3]. Since focused beam-lines are not formed physically in SAI, it is possible to form an image from each firing; also, high-quality images may be obtained computationally via recursive summation of a series of low-resolution synthetic aperture images.

Although PWI and SAI have shown potential in raising the frame rate limit, their real-time realization is inherently not a trivial implementation task. One technical challenge that concerns many system developers is the massive computational demand of these fast imaging methods as compared to conventional ultrasound image formation [4]. Existing ultrasound scanners are usually equipped with field programmable gate arrays (FPGA) and digital signal processors (DSP) to handle computations related to image formation [1]. [5]. Nevertheless, they are merely intended to work with the conventional ultrasound imaging paradigm, so their computational capacity is not sufficient to facilitate all the computation processes required for advanced ultrasound imaging methods. Thus, it is necessary to develop another real-time computing platform in order to address such a computation bottleneck.

Recently, the emergence of graphics processing units (GPU) has spurred the pace of development in ultrasound imaging systems. These computing devices, which can be readily converted into parallel processors through the use of application programming interfaces provided by the vendors, have been used to compute color Doppler images [6], render three-dimensional images in real-time [7], derive motion vector estimates from echocardiography images [8], and implement elasticity imaging algorithms [9]. Initial attempts have also been made to convert these units into parallel computing engines for SAI [10].

In this paper, we present the development of a GPU-based software beamformer architecture that is intended for use in advanced ultrasound imaging. It has been our intent to develop a high-speed, programmable beamformer module that can form ultrasound images from radiofrequency (RF) data samples acquired at the pre-beamform level (i.e. the channel domain). This can enable us to realize advanced imaging paradigms that work with pre-beamform data, such as SAI and PWI.

II. Beamformer Architecture

A. Hardware Setup

The developed beamformer is implemented by employing a group of GPUs as the computational platform. As illustrated in Fig. 1, the GPUs are housed inside a personal computer (PC) workstation as expansion boards that are connected through

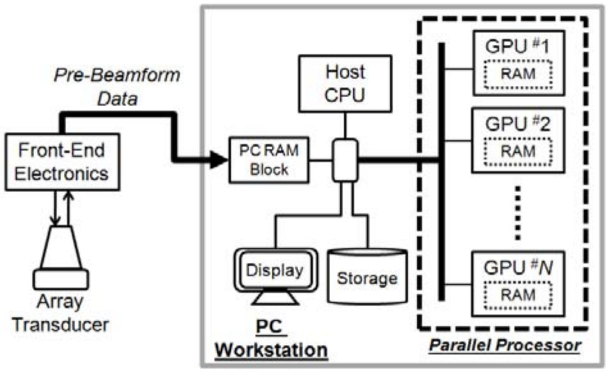

Fig. 1. Hardware setup for the GPU-based software beamformer. During operation, data is streamed into the RAM block inside a PC workstation. The host CPU then distributes each frame of pre-beamform data into a GPU to perform beamforming.

PCI-Express buses with real-time data-transfer bandwidth (maximum of 8 GB/s for ×16 buses). Their parallel processing resources are managed through a software-based application programming interface known as CUDA (compute unified device architecture; NVIDIA, Santa Clara, USA) [11]. As will be shown in Sec. III, our beamformer architecture is compatible with both Tesla-class and Fermi-class GPUs (NVIDIA), and a combination of GPUs in these two classes may be used as the computational hardware for this beamformer.

The aim of our GPU-based beamformer is to perform real-time processing using advanced imaging schemes like SAI and PWI. During operation, it takes in raw RF data from each array element and calculates beamformed images in real-time. The pre-beamform data is presumed to be first stored in the PC's random access memory (RAM). They are streamed into each GPU on a frame-by-frame basis (controlled by a master central processing unit (CPU)) to facilitate beamforming of multiple image frames in parallel. The beamformed image results are then transferred to the PC display and the storage device for archival.

The advantages of using GPUs for this work are two-folded. First, their hardware architecture, comprising hundreds of processor cores, has been specifically developed to facilitate single-instruction, multiple-thread (SIMT) computations. This parallelism can help accelerate the beamforming operation of multiple image pixels without running into power wall issues faced by conventional microprocessors. Second, the software-based programmability of GPUs (via the C++ language) may represent a more accessible alternative over FPGAs that need to be configured using low-level hardware description languages like VHDL or modified high-level languages like SystemC (both have limited syntax functionality) [12].

B. *Software Architecture*

Our beamformer has been programmed on the basis of a frame-based pipelining approach where each available GPU in the group is assigned to process one frame of pre-beamform data at any given time. For fast imaging paradigms like SAI and PWI, this approach is essentially equivalent to assigning each GPU to compute a low-resolution image (LRI) from the pre-beamform pulse-echoes of one firing. It is worth noting that the beamformer's frame processing capacity (i.e. the number of LRIs that can be computed in parallel) directly scales with the number of GPUs in the group.

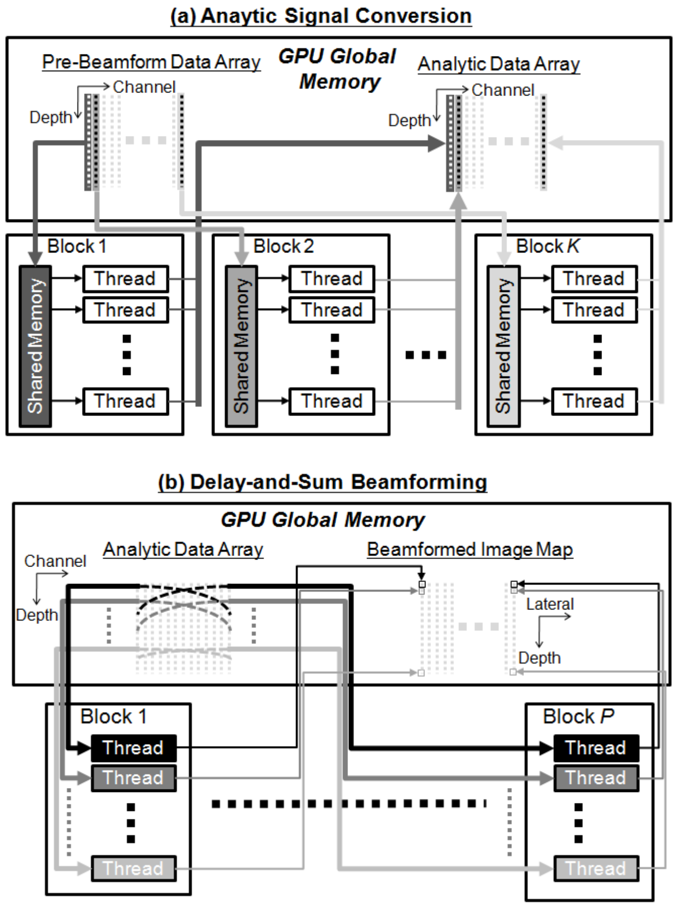

Fig. 2. Multi-thread processing architecture of the GPU-based beamformer. During analytic signal conversion (a), latency is reduced by copying an entire channel of pre-beamform data to the thread block's shared memory. For the delay-and-sum stage (b), the position of sample in each channel to be beam-summed in a thread is denoted by a dashed curve in the analytic data array.

For the processing of each frame, a two-stage approach has been adopted by the beamformer to derive the LRI value at every pixel position. In the first stage, the analytic form of pre-beamform data is computed for each array channel through the use of Hilbert transform. Subsequently, a delay-and-sum procedure is carried out to add the relevant analytic samples of different channels and thereby obtain the beamformed value of every LRI pixel. The beamforming delays of all channels are dynamically calculated for each image location based on the nominal pulse-echo flight-time for that location. The SIMT processing architecture for each stage are described as follows.

1) Analytic Signal Conversion: Fig. 2a gives a conceptual illustration of how this stage is implemented in the GPU-based beamformer. As can be seen, one block of threads in the GPU is assigned to compute the analytic signal for one channel of pre-beamform RF data. This thread-block allocation scheme is intended to facilitate efficient use of the GPU's shared memory (high-speed on-chip memory allocated for each thread block) and reduce the processing latency (an issue to be discussed in further details in Sec. II-C). Each thread in the block is instructed to carry out a Hilbert transform operation and in turn derive the analytic signal sample at one index position in the channel's analytic data array. Note that the Hilbert transform is implemented as a many-tap finite-impulse-response (FIR) filter that has an impulse response equal to the definition of the discrete Hilbert transform [13]. As such, each analytic data sample is essentially equivalent to the FIR filter output for a finite window of pre-beamform data in the corresponding channel.

2) Delay-and-Sum Operation: Fig. 2b illustrates how delay-and-sum beamforming is performed in the GPU-based beamformer to obtain the LRI pixel values. In this processing stage, each block of threads is allocated to compute one scanline of LRI pixel values. For each individual thread, it is instructed to calculate a single LRI pixel value via the following three-step procedure:

a) Estimate the beamforming delays for all channels with respect to the pixel position for that thread;

b) Retrieve the corresponding data sample in each channel of the pre-beamform analytic signal array based on the delays;

c) Obtain the LRI value by multiplying an apodization weight to the retrieved set of analytic data samples and summing the apodized values.

It should be noted that the beamforming delays for this operation may be calculated differently depending on the imaging scheme. For PWI and SAI, the delays correspond to two-way propagation between the transmit source and the receive element position [2], [3]. Another point worth noting is that although pre-defined apodization weights are used in the current version of the beamformer, it is possible to extend the software architecture to adaptively compute these weights depending on the pre-beamform signal statistics. As such, our beamformer may potentially be used to investigate various adaptive beamforming algorithms [14].

C. *Processing Speed Optimization*

To reduce latency overheads in GPU-based parallel processing, efficient management of memory access is known to be crucial given that each memory read-write operation may require up to several hundred clock cycles [11]. In general, this task can be facilitated through agile use of GPU's two-tier memory structure that comprises: 1) shared memory for each thread block (small in size, but with fast access speed); 2) texture and global memory residing in the GPU's device core (slower access speed, but may improve if cached).

In the first stage of our beamformer (Fig. 2a), processing latency is lowered by using the shared memory to store an entire channel of pre-beamform data and thereby facilitating fast data access by each thread in a block. Note that the Hilbert transform filter coefficients are also stored in the shared memory to accelerate the analytic signal computation process. For the delay-and-sum stage (Fig. 2b), latency is kept low by either: 1) creating texture memory pointers to cache data samples that are repeatedly fetched to different threads, or 2) simply exploiting the global memory cache (only available in Fermi-class GPUs). The shared memory is used in this stage to store the apodization weights. It should be noted that the shared memory size in currently-available GPUs is not large enough to store all the data samples needed for beam-summing in each thread block.

III. Prototype Implementation

A. *PC Backbone*

Based on our beamformer architecture, we have assembled a prototype PC platform that can support different GPU group sizes. This prototype operates on a motherboard with three PCI-Express×16 expansion slots (P6T Deluxe; ASUSTek, Taipei, Taiwan), and it uses a quad-core, 2.66 GHz CPU as the host controller (i7-920; Intel Corporation, Santa Clara, USA). 6 GB of DDR3 RAM is included in the prototype PC to store pre-beamform data prior to processing.

B. *GPU Computational Platform*

In this work, we have used the prototype PC to experiment with four different GPU groupings: 1) a single Tesla-class GTX-275 GPU; 2) a single Fermi-class GTX-470 GPU; 3) two GTX-470 GPUs; 4) one GTX-275 alongside one GTX-470. Specifications for these two GPU models are readily available from the manufacturer, so they will not be repeated here. Nevertheless, three important differences should be noted between them. First, GTX-470 has more processor cores (448 vs. 240), but it runs at a slower clock rate than GTX-275 (1.215 GHz vs. 1.404 GHz). Second, GTX-470 includes a level-two global memory cache (768 kB) that is not found in GTX-275, but its texture memory filling rate is slower (34 billion/sec vs. 50.6 billion/sec). Third, GTX-470 allocates 48 kB of shared memory for every 32 cores, and this should be contrasted against GTX-275 that assigns 16 kB for every 8 cores.

C. *Beamformer Software*

The GPU-based beamformer is coded in C++ via a functional programming approach, and various CUDA syntaxes and functions (ver. 3.0) are invoked to realize multi-thread processing on the GPUs. In the analytic signal conversion stage of the beamfomer, a 51-tap FIR filter is implemented for the Hilbert transform, and its coefficients are computed during beamformer initialization. This filter order is empirically chosen to achieve accurate Hilbert transform results. In the delay-and-sum stage, SAI or PWI is assumed to be the imaging paradigm, so the beamforming delays are computed as the two-way pulse-echo propagation times (for a nominal acoustic speed of 1540 m/s). Also, a Hanning window is used as the apodization weight.

IV. Performance Assessment

A. *Overview of Methodology*

To evaluate the computational performance of our GPU-based beamformer prototype, a series of processing trials has been carried out. These trial runs are conducted on SAI and PWI test data acquired using a pre-beamform data acquisition system [15] that is connected to a reconfigured research scanner (Sonix-RP; Ultrasonix, Richmond, Canada). The RF sampling rate for the test datasets is 40 MHz, and each frame of pre-beamform data (i.e. for each transmit firing) comprises 32 receive channels. With these data, we have used our beamformer to compute a series of LRIs for imaging depths between 5-15 cm (260 additional RF samples acquired per

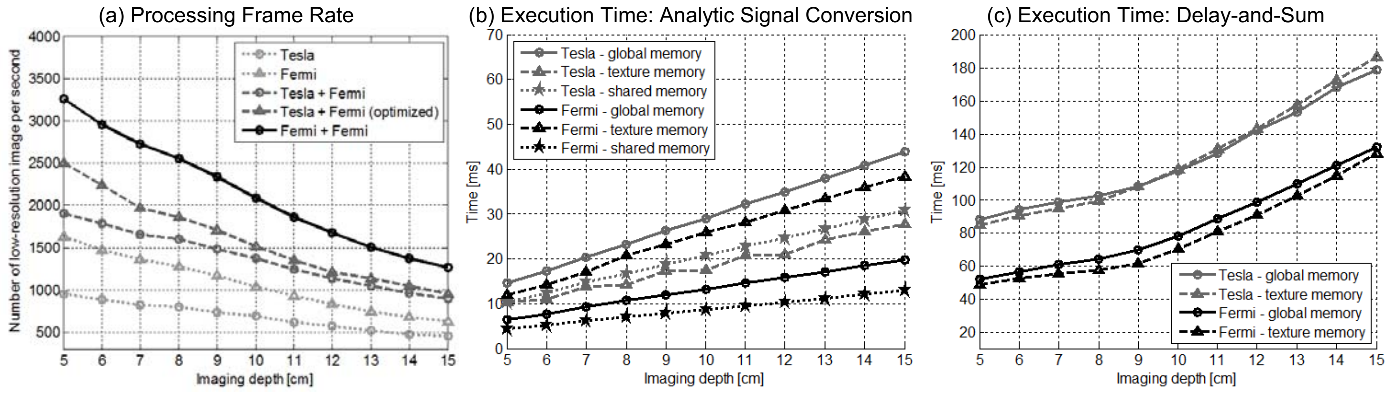

Fig. 3. Performance results of our GPU-based beamformer as a function of imaging depth: (a) processing frame rate, shown for single- and dual-GPU setups; (b) execution time of analytic signal conversion; (c) execution time of delay-and-sum. "Tesla" and "Fermi" respectively denote GTX-275 and GTX-470 GPUs.

channel for every 1 cm increase in imaging depth). Each LRI has a pixel dimension of 512-by-255 (for all imaging depths examined). The execution time of each processing stage is recorded using CUDA's built-in timing functions, and the measurements are averaged over 30 trials. The processing frame rate is then calculated based on the total execution time of the beamformer.

B. Results and Discussion

For the GPU groupings considered in our beamformer prototype, Fig. 3a plots their processing frame rate as a function of imaging depth. As can be seen, the dual GTX-470 configuration (black curve) has achieved the highest processing frame rate. It is capable of computing over 3200 and 1300 LRIs per second respectively for imaging depths of 5 cm (used in carotid studies) and 15 cm (needed for cardiac studies). This processing throughput is roughly 3.2 times higher than the single GTX-275 setup (light-gray curve with circle markers). Another point worth noting is that for the hybrid configuration (GTX-275 + GTX-470), its processing frame rate may be improved after an optimization procedure that reduces the GPU idle time due to pipelining synchronization between the two GPU models (see dark-gray curve with triangle markers).

For Figs. 3b and 3c, GTX-470 has expectedly yielded a shorter execution time for the two beamforming stages (see dark curves) as compared to GTX-275 because more parallel processing cores are available in the new Fermi-class GPUs (448 vs. 240). Also, the advantage of using shared memory in the analytic signal conversion stage (to store the entire channel of pre-beamform data for a thread block) can be observed by noting the few-fold reduction of execution time for both GPU models (see asterisk-marked curves in Fig. 3b). For the delay-and-sum stage (Fig. 3c), the performance difference between using texture memory pointers and global memory for data caching is found to be insignificant.

V. Conclusion

Our GPU-based beamformer has demonstrated a processing frame rate of over 1000 fps when two Fermi-class GPUs are used. Its processing capacity can expectedly increase further if additional GPUs are included in the system hardware. As such, with the advent of GPUs, it becomes possible to use standalone PC workstation as a real-time processor for advanced imaging methods that work with pre-beamform data.

Acknolwedgements

We wish to thank Prof. Paul Cheung and Mr. Harry Chiu at HKU for their enthusiastic support on the project. This work is supported in part by the Hong Kong Innovation and Technology Fund (ITS/492/09, InP/210/10, InP/211/10).

References

[1] G. York and Y. Kim, "Ultrasound processing and computing: review and future directions", *Ann. Rev. Biomed. Eng.*, vol. 1, pp. 559-588, 1999.

[2] G. Montaldo, M. Tanter, J. Bercoff, N. Benech, and M. Fink, "Coherent plane-wave compounding for very high frame rate ultrasonography and transient elastography", *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 56, pp. 489-506, 2009.

[3] J. A. Jensen, S. I. Nikolov, and K. L. Gammelmark, "Synthetic aperture ultrasound imaging," *Ultrasonics,* vol. 44, pp. e5-e15, 2006.

[4] S. I. Nikolov, B. G. Tomov, and J. A. Jensen, "Real-time synthetic aperture imaging: opportunities and challenges", *Proc. Asilomar Conf. Signals Syst. Comp.*, pp. 1548-1552, 2006.

[5] C. Basoglu, R. Managuli, G. York, and Y. Kim, "Computing requirements of modern medical diagnostic ultrasound machines", *Parallel Comput.*, vol. 24, pp. 1407-1431, 1998.

[6] L. W. Chang, K. H. Hsu, and P. C. Li, "Graphics processing unit-based high-frame-rate color Doppler ultrasound processing", *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 56, pp. 1856-1860, 2009.

[7] A. F. Elnokrashy, A. A. Elmalky, T. M. Hosny, M. A. Ellah, A. Megawar, A. Elsebai, A. B. M. Youssef, and Y. M. Kadah, "GPU-based reconstruction and display for 4D ultrasound data", *Proc. IEEE Ultrason. Symp.*, pp. 189-192, 2009.

[8] G. Kiss, E. Nielsen, F. Orderud, and H. G. Torp, "Performance optimization of block matching in 3D echocardiography", *Proc. IEEE Ultrason. Symp.*, pp. 1403-1406, 2009.

[9] N. Deshmukh, H. Rivaz, and E. Boctor, "GPU-based elasticity imaging algorithms", *Proc. Int. Conf. Med. Imag. Comp. & Comp. Assist. Interven.*, 2009.

[10] D. Romero, O. Martinez, C. J. Martin, R. T. Higuti, and A. Octavio, "Using GPUs for beamforming acceleration in SAFT imaging", *Proc. IEEE Ultrason. Symp.*, pp. 1334-1337, 2009.

[11] NVIDIA Co. *CUDA Programming Guide*. Santa Clara, USA: 2010.

[12] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs", *Proc. IEEE Symp. App. Spec. Process.*, pp. 101-107, 2008.

[13] T.K. Moon and W.C. Stirling, *Mathematical Methods and Algorithms for Signal Processing.* Upper Saddle River, USA: Prentice-Hall Inc., 2000.

[14] J. F. Synnevag, A. Austeng, and S. Holm, "Adaptive beamforming applied to medical ultrasound imaging", *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 54, pp. 1606-1613, 2007.

[15] I. K. H. Tsang, B. Y. S. Yiu, D. K. H. Cheung, H. C. T. Chiu, C. C. P. Cheung, and A. C. H. Yu, "Design of a multi-channel pre-beamform data acquisition system for an ultrasound research scanner", *Proc. IEEE Ultrason. Symp.*, pp. 1840-1843, 2009.