PORTABLE PARALLEL KERNELS FOR HIGH-SPEED BEAMFORMING IN SYNTHETIC APERTURE ULTRASOUND IMAGING

Joao Amaro, Gabriel Falcao*, Billy Y. S. Yiu‡, and Alfred C. H. Yu‡*

*Instituto de Telecomunicações, University of Coimbra, Portugal
‡Medical Engineering Program, University of Hong Kong, Hong Kong SAR

ABSTRACT

In medical ultrasound, synthetic aperture (SA) imaging is well-considered as a novel image formation technique for achieving superior resolution than that offered by existing scanners. However, its intensive processing load is known to be a challenging factor. To address such a computational demand, this paper proposes a new parallel approach based on the design of OpenCL signal processing kernels that can compute SA image formation in real-time. We demonstrate how these kernels can be ported onto different classes of parallel processors, namely multi-core CPUs and GPUs, whose multi-thread computing resources are able to process more than 250 fps. Moreover, they have strong potential to support the development of more complex algorithms, thus increasing the depth range of the inspected human volume and the final image resolution observed by the medical practitioner.

Index Terms— Synthetic aperture, Ultrasound medical imaging, Beamformer, OpenCL, GPU

1. INTRODUCTION

Ultrasound medical imaging systems are nowadays a fundamental tool for helping medical practitioners performing a reliable non-invasive diagnosis procedure. Based on the processing and analysis of pulse-echo signals, current ultrasound imaging systems are complex from a hardware perspective due to the use of array transducers that inherently involve multi-channel processing. They are supported by extensive microelectronics systems such as filed-programmable gate arrays (FPGA) and digital signal processors (DSP) [1].

As the theoretical principles of advanced ultrasound image formation paradigms have become more mature in recent years, there is a growing level of interest in realizing them in practice. Of particular interest is the real-time execution of those algorithms, which represents a key factor in ultrasound imaging regarding its bedside clinical role.

One such advanced ultrasound technique is synthetic aperture beamforming, which transmits unfocused pulses form distinct lateral positions [2]. Each pulse generates echoes that are received by all channels in the sensor to form a low-resolution image (LRI) per each instance of pulse-echo sensing, which is accomplished by performing delay-and-sum beamforming at each pixel position. Then the sum of a predefined set of LRIs can be used to form high-resolution images (HRI) [3]. Naturally, these operations are computationally demanding. Additionally, unlike previous ultrasound techniques that use the same set of focusing delays, synthetic aperture beamforms each pixel based on different sets of varying focus delays [2], which is more complex to do and demands higher processing capabilities.

Although originally dedicated to image rendering, graphics processing units (GPU) have been recently introduced as powerful parallel accelerators for general-purpose computing [4, 5]. They have been shown to be well-suited to the real-time realization of synthetic aperture imaging algorithms [6]. Unlike conventional approaches that exploit the compute unified device architecture (CUDA) interface [7] which is limited to execute only in NVIDIA GPUs, in this article we propose using OpenCL [8, 9], a more generic programming model, and show that it supports the execution of these parallel kernels on a wide variety of multi- and many-core systems [10]. Depending on the specificities of the system (e.g., number of array transducers, image resolution, etc.), OpenCL allows targeting synthetic aperture kernels to the most appropriate heterogeneous computational environment [11] that is capable of supplying the necessary processing power. In this article we report experimental results obtained by running the same kernel on Intel CPUs and ATI or NVIDIA GPUs. We also show how these OpenCL-based signal processing kernels were developed in order to extract parallelism from the architecture.

2. SYNTHETIC APERTURE IMAGING

We first discuss the theory associated with synthetic aperture imaging. We start by identifying and elaborating on the different phases of the algorithm.

This work is funded in part by the Portuguese Foundation for Science and Technology (FCT) project PEst-OE/EEI/LA0008/2011, as well as the Hong Kong Innovation and Technology Fund (ITS/292/11).

2.1. Signal transmission characteristics

The transmitting elements used in current transducers emit a linear signal. The synthetic aperture algorithm is based on the emission of a spherical wave from each transmission source. To minimize changes in the hardware, we can use the current transducers, but the signal is emitted as in figure 1 to emulate a spherical wave. The point behind the element array represents the epicenter of the spherical wave, or the virtual source point.

2.2. Signal reception

As previously mentioned, the signal propagates in the form of a spherical wave both in transmission and in reception (after reflection in the scattering medium). So, we must take into account the relative delay of the signal received from a reflection in a pixel in relation to the neighboring receiving elements.

2.3. LRI calculation

To compute each pixel of a LRI, we now have to account for the contribution of each receiving element, with a delay-and-sum procedure. This is basically a linear interpolation of the pixels' position neighboring analytic data samples $\alpha_{n,m}$:

$$\alpha_{n,m}(P_0) = \lambda a_{n,m}(k) + [1-\lambda] a_{n,m}(k+1), \qquad (1)$$

where n corresponds to the $n-th$ receive channel and m represents the $m-th$ transmitting virtual source point. Each of the receiving elements contribution is passed through a window function ω_n. To find the depth sample number k, we must first consider the focusing delay $\tau_{n,m}(P_0)$ and the interpolation weight λ:

$$k = \lfloor f_s \tau_{n,m}(P_0) \rfloor, \qquad (2)$$

$$\lambda = 1 + k - f_s \tau_{n,m}(P_0). \qquad (3)$$

The focusing delay in SA imaging is calculated in the following way:

$$\tau_{n,m}(P_0) = \frac{d_T(P_0;m) + d_R(P_0;n)}{c_0}, \qquad (4)$$

where $d_T(P_0;m)$ represents the distance between the transmitting position and the position of pixel P_0, and $d_R(P_0;n)$ the distance between this position and the nth receiving element, while c_0 is the speed in the scattering medium.

Finally, the value for pixel P_0 of the mth LRI can be obtained by:

$$L_m(P_0) = \sum_{n=1}^{N} \omega_n \alpha_{n,m}(P_0), \qquad (5)$$

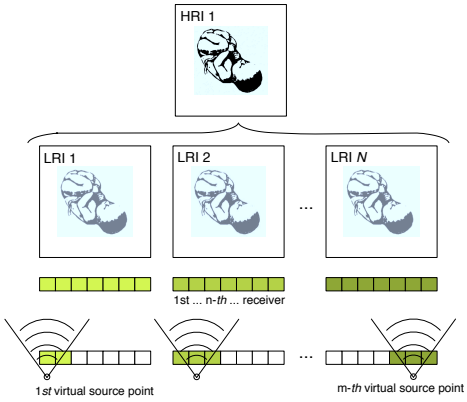


Fig. 1. Pulse-echo system based on synthetic aperture and the multi-channel generation of LRI and HRI.

2.4. HRI compounding

After computing a new LRI, we replace the oldest LRI in the compounding frame, and after recursive summation of the entire frame (with size N), we are able to form a new HRI. This can be modeled as a relation between the new HRI and the previous one:

$$H_i(P_0) = H_{i-1}(P_0) + L_i(P_0) - L_{i-M}(P_0). \qquad (6)$$

3. PARALLEL PORTABLE KERNELS FOR SA

Historically, developing an algorithm for running on a multicore system was considered nontrivial. Recently, parallel programming models were developed to allow programming some specific architectures in particular [7]. Later, the introduction of a broadly accepted programming model compatible with different multi- and many-core architectures was made possible with the arrival of OpenCL [8]. OpenCL provides a framework that allows signal processing programmers to develop code once and execute it on a variety of multicore systems such as CPUs or GPUs. Using a C/C++ environment, the programmer instructs the compiler how a code section should be parallelized. Parallelization is organized by the programmer in work-groups, where each work-group dispatches a certain predefined number of work-items. At runtime, the program inspects the compute resources available on the platform, compiles the source code according to it and launches execution. At the end, processed data is sent back to the host system that orchestrates execution [9].

The parallel algorithm developed exploits thread-level parallelism to perform the calculation of new LRIs. Processing is performed on a pixel-per-pixel basis for all channels of the system. The processing unity with smaller granularity-level is the work-item.

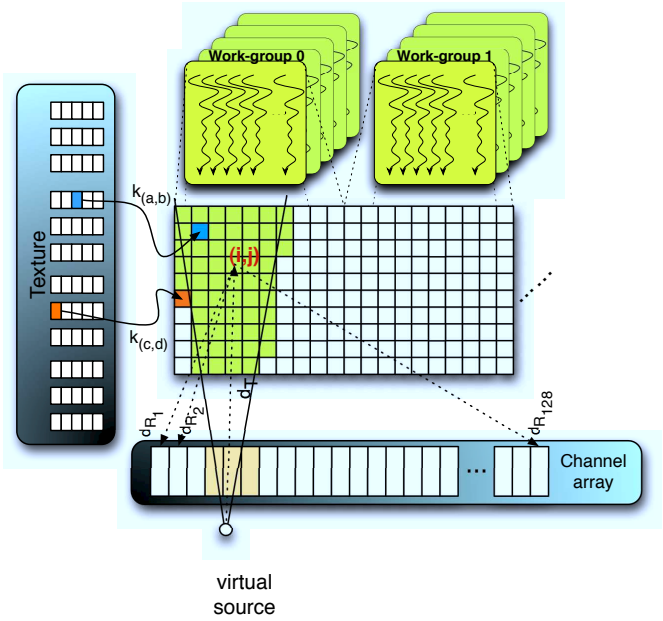


Fig. 2. Calculation of pixel (i, j) for the nth LRI. Illustration of the influence of a virtual source point in the calculation of the pixels of an LRI. Pixel with index (i, j) is processed by a work-item in parallel with the processing of other pixels in the same work-group.

3.1. Parallel calculation of LRIs

Figure 2 describes how pixels are influenced by each virtual source. The processing is performed in parallel by all work-items and one of two situations occur: either the pixel is under the influence of a virtual source or it isn't. This verification implies the use of divergent (conditional) instructions, which penalizes performance. However, parallelization is achieved since each work-item processes in parallel one of the pixels that are under the influence of a same virtual point (for those who are outside, the work-item returns execution). Every two work-groups are able to perform the parallel processing of a complete LRI.

Pulse-echo distances are represented by d_T and d_R, which are used to calculate the delay $\tau_{n,m}$ as shown in (4) and finally obtain the depth sample number k indicated in (2).

The level of parallelism achieved increases (until a certain limit) with the number of compute resources available. In the case of a GPU, the processing of hundreds of pixels in parallel is possible.

3.2. Using texture memory to accelerate computation

On the GPU, texture memory has latency times similar to those of global memory. The main advantage of using this type of memory lies on the level 1 cache capabilities associated with textures. Not only data used is maintained for reuse, but also do its immediate neighboring elements, which is useful in this particular algorithm. Therefore, analytical data is loaded into textures before the kernel is launched. Then, data is accessed by each work-item at element in the position given by depth sample k previously calculated, in order to produce $\alpha_{n,m}$ [6]. This procedure basically consists of a weighted summation of interpolated channel-domain samples for all N array channels as described in (1).

In the case the OpenCL kernel is running on a CPU, where texture memory does not exist, this functionality is obtained using software emulation. However, it is worth noting that the use of textures in this scenario is not recommended mainly due to efficiency reasons.

4. EXPERIMENTAL RESULTS

4.1. Apparatus

The experimental results were obtained using the OpenCL C API, interfaced with Matlab's MEX-function, and the C console compiled with Visual Studio 2010. The computer has a quad-core Intel Core i7 950 @3.07 GHz, 3GB of RAM memory, running Windows 7 Ultimate x86. The OpenCL devices are an ATI Radeon HD6970 (Cayman) with 1536 shaders, a NVIDIA Tesla C1060 with 240 cores. We must note that the relation between performance vs. number of compute units is not the same for both vendors, as NVIDIA indicates fewer units. As an additional test, the algorithm was also run on the CPU, to further demonstrate the computing potential of the GPUs.

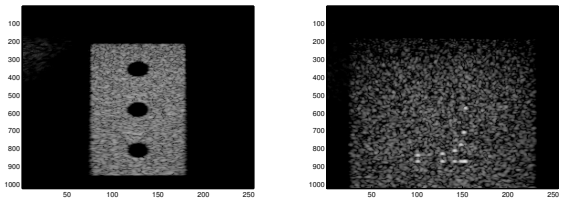
A Sonix-RP research scanner equipped with a pre-beamformed data acquisition tool was used to collect the dataset processed in the host. Ultrasound parameters are: frequency – 10 MHz; transmit pulse shape – 2-cycle sinusoid; pulse repetition frequency – 5 kHz. The synthetic aperture implementation is based on a scanner front-end that was reprogrammed to fire according to a virtual point source configuration. It uses 97 point sources in total (0.3 mm laterally spaced apart, 20 mm axially behind field of view), each formed from a 64-channel aperture. It performs one firing from each virtual point source, swept from left to right side. The data acquisition is based on 128 channels received in parallel using the pre-beamformed data acquisition tool, with 40 MHz sampling and 12-bit resolution.

4.2. High frame-per-second LRI throughput calculation

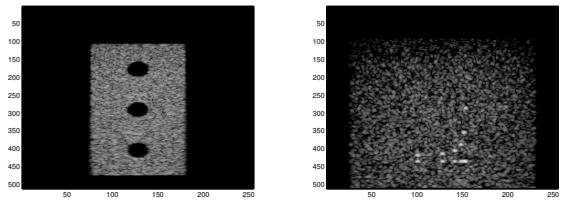
To perform the experimental results we have run the same OpenCL kernel on 3 different platforms: one multi-core CPU and two many-core GPUs. We used 2 datasets which were obtained using synthetic aperture beamforming: dataset 1 (DS1) consists of a Perforated Plate, while dataset 2 (DS2) shows random Dots. The selected aperture is 256 for all experiments. Table 1 indicates the total computation times for gen-

Table 1. Computation times for two datasets: Perforated Plate (DS1) and Dots (DS2) generating images with resolution 512×255 pixels.

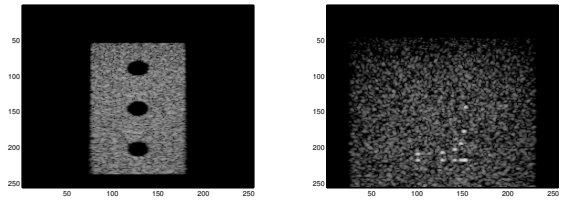
Platform	Radeon HD6970 GPU		Tesla C1060 GPU		Intel i7 950 CPU	
Dataset 512×255 pixels	DS1	DS2	DS1	DS2	DS1	DS2
Total memory operations time	695 ms	695 ms	1.068 s	1.058 s	187 ms	182 ms
Total kernel execution time	403 ms	382 ms	1.067 s	670 ms	24.349 s	20.221 s



(a) 1024×255 pxl. images gen. on ATI Radeon HD 6970



(b) 512×255 pxl. images gen. on Intel i7 950



(c) 256×255 pxl. images gen. on NVIDIA Tesla C1060

Fig. 3. Perforated Plate (left) and Dots (right) images generated from datasets obtained using synthetic aperture beamforming. The images were generated by running the same OpenCL kernel on three different multi-core platforms.

erating 97 LRIs. They include data transfers between host and device, and kernel execution times on device. It can be seen that the CPU platform presents lower transfer times, which is natural since it is not limited by the PCIe bus, but it also shows considerably higher kernel processing times due to a lower number of processing cores available. The speedup obtained from CPU to ATI GPU execution ranges from 22 to 30, while against the NVIDIA GPU it approximates $52 \sim 60$ times. The Radeon GPU is capable of processing more than 253 frames per second (fps), as the inspection of table 1 indicates, achieving 1.35 GFLOPS for DS1 with 512×255 pixels.

Figure 3 demonstrates the code portability concept by showing that all generated images are equivalent and that the main difference is computation time. Many-core systems with superior capabilities, such as number of cores, higher clock frequencies or memory bandwidth would run the OpenCL kernel even faster, thus processing more fps.

5. RELATION TO PRIOR WORK

To our knowledge, this work is perhaps the first attempt to investigate the feasibility of adopting a hardware-flexible parallel processing approach to execute synthetic aperture beamforming operations at real-time throughput. Based on devising portable software kernels using the OpenCL framework, our approach inherently differs from previous CUDA-based synthetic aperture beamformers that can only work on vendor-specific hardware (NVIDIA) [6]. Also, it is not the same as other solutions that attempt to use FPGAs [12], computer clusters [13], and DSP platforms [14] for synthetic aperture image computing purposes. From an ultrasound system design standpoint, our OpenCL-based solution should be more favorable than others as its code portability allows the beamforming kernel to be hosted on a wide variety of computing hardware, from multi-core CPUs to many-core GPUs and even FPGAs. This would provide ultrasound system designers with more flexibility in designing novel hardware architecture for synthetic aperture ultrasound imaging.

6. CONCLUDING REMARKS

With the availability of code-portable parallel processing kernels to handle beamforming operations, it becomes more feasible to pursue practical realization of synthetic aperture ultrasound imaging whose image formation principles are known to be computationally demanding. This work is therefore expected to contribute to the latest developments in advanced ultrasound system design. It is worth noting that, besides synthetic aperture beamforming, our OpenCL-based parallel processing approach may be extended to other computing operations in synthetic aperture imaging. For instance, in the delay-and-sum process, it may be of interest to include an adaptive apodization module that applies signal-dependent channel weighting. Such an adaptive beamforming strategy is well considered to be computationally demanding as well. Parallel processing, especially portable ones that can be executed on a variety of computing hardware, may provide an answer to this technical hurdle.

7. REFERENCES

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

[2] J. A. Jensen, S. I. Nikolov, K. L. Gammelmark, and M. H. Pedersen, "Synthetic aperture ultrasound imaging," *Ultrasonics*, vol. 44, Supplement, pp. e5–e15, 2006.

[3] S. I. Nikolov, K. L. Gammelmark, and J. A. Jensen, "Recursive ultrasound imaging," in *Proc. IEEE Ultrasonics Symp.* IEEE, 1999, pp. 1621–1625.

[4] T. P. Chen and Yen-Kuang Chen, "Challenges and opportunities of obtaining performance from multi-core CPUs and many-core GPUs," in *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'09)*, April 2009, pp. 613–616.

[5] H.K.-H. So, Junying Chen, B.Y.S. Yiu, and A.C.H. Yu, "Medical ultrasound imaging: To GPU or not to GPU?," *IEEE Micro*, vol. 31, no. 5, pp. 54–65, 2011.

[6] B.Y.S. Yiu, I.K.H. Tsang, and A.C.H. Yu, "GPU-based beamformer: Fast realization of plane wave compounding and synthetic aperture imaging," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 58, no. 8, pp. 1698–17052, August 2011.

[7] CUDA Developer NVIDIA, "CUDA 5.0," Nov. 2012.

[8] Khronos Group, "OpenCL 1.2," Nov. 2011.

[9] B. R. Gaster, H. Lee, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL*, Morgan Kaufmann, 2012.

[10] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC Decoding on Multicores Using OpenCL," *IEEE Signal Processing Magazine*, vol. 29, no. 4, pp. 81–109, July 2012.

[11] S. Singh, "Computing without processors," *Communications of the ACM*, vol. 54, no. 8, pp. 46–54, August 2011.

[12] J. A. Jensen, M. Hansen, B. G. Tomov, S. I. Nikolov, and H. Holten-Lund, "System architecture of an experimental synthetic aperture real-time ultrasound system," in *Proc. IEEE Ultrason. Symp.*, 2007, pp. 636–640.

[13] F. Zhang, A. Bilas, A. Dhanantwari, K. N. Plataniotis, R. Abiprojo, and S. Steriopoulos, "Parallelization and performance of 3D ultrasound imaging beamforming algorithms on modern clusters," in *Proc. ACM Int. Conf. Supercomput.*, 2002, pp. 294–304.

[14] C. R. Hazard and G. R. Lockwood, "Theoretical assessment of a synthetic aperture beamformer for real-time 3-d imaging," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 46, pp. 972–980, 1999.