SOUND SPEED OPTIMIZATION USING IMAGE TEXTURE ON CUDA

[1] XINGWU HE, [2]HUAGUO YIN, [3]HONGLIN ZHOU AND [4]XIA ZHANG
[1] Department of Electronics and Information Chengdu Vocational College of Agricultural Science and Technology Chengdu, China
E-mail:[1] hexw981@126.com, [2]xingwuHe@126.com

ABSTRACT

The Compute Unified Device Architecture (CUDA) is a brand new parallel processing platform making use of the unified shader design of the most current Graphics Processing Units (GPUs) from NVIDIA. In this paper, we apply this revolutionary new technology to implement the sound speed optimization (SSO) with image texture analysis for medical ultrasound imaging. The sum and difference histogram of parallel texture mask production is also presented. This SSO method achieves 77ms for a group of alternative sound velocities that are from 1400 to 1700m/s every 10m/s a sample image with the size of 256 × 512, about 45 times faster than the CPU implementation. Testing results from GPU and CPU are compared in terms of decision results and program runtime with different image size.

Keywords: *Phase Aberration; Sound Speed Correction; Image Texture; Parallel Processing; GPU*

1. INTRODUCTION

Ultrasound imaging has been a valuable tool in medical diagnosis for a long time. In ultrasound imaging systems, the sound speed is a very important parameter for beamforming procedure and usually considered as a constant, typically 1540 m/s in human tissue [1]. However, the actual speed of sound in human body covers 1.42 mm/s (in breast fatty tissue) to about 3.70 mm/s (in bone); furthermore, the same tissue of different patients might have different speeds of sound. If the system pre-set sound velocity is away from the true one in the tissue too much, the shift of structure position and loss of image contrast will appear up, which causes small mis-registrations of image location. Therefore, sound speed optimization in ultrasound imaging system is of great importance. Though researchers have proposed many algorithms in phase aberration correction, most of these algorithms require very high complexities both in process time and memory storage which make a real-time application impractically on the general CPU of a PC.

Many researchers are interested in phase aberration correction produced by mis-matching local tissue velocity with pre-set one. Jochen F. Krücker et al. [2] have proposed a way to estimate the sound speed in a region of interest (ROI) by obtaining views of that area from different views and quantifying the relative geometric distortions between these different directions using image registration. And one method to correct unknown phase aberration in the coherent imaging systems is present in [3], which uses the area-wise average speckle brightness as a quality factor. It is known that image texture is an important characteristic for the image analysis. Haralick (1973) defined a set of texture parameters to quantify the image texture properties that can be qualitatively evaluated as having one or more of the properties of fineness, coarseness, smoothness, and so on [4]. With this analysis tool, Du and Liu give out a method to find the best sound speed by the image texture parameters which is a function of trial sound speeds [5]. Useful as these methods, there share a same withdraw, the time consumed in computation is too high to satisfy the need of the real time ultrasound imaging system.

Recently, the revolutionary of computing platform has come up with faster parallel processors developed. Especially, the general purpose GPU overwhelms Multi-core CPUs for image processing due to its unique benefits, such as lighter thread, lower synchronization time consumption, more powerful computing capability, and so on. The CUDA technique released by NVIDIA provides efficient methods to solve complex computation problems on the brand new GPUs with the capability of general purpose computing [6].

The main purpose of this study focuses on the design and implementation of parallel algorithms for sound speed optimization using image texture, which is mainly, based on CUDA performance features, such as the memory selection strategy, applicable thread structure, parallel histogram produced and so on. These parallel algorithms are suitable for CUDA platform and implement in GPU hardware that meet the requirements of real time system.

The organization of the paper is as follows: Part A of Section II introduces the analysis of the sound speed optimization with image texture used in the ultrasound imaging system. In part B of Section II, we represent a GPU-based way for this method. Testing results are show in Section III. Conclusions and future directions are represented in Section IV.

2. METHOD

2.1 Sound Speed Optimization On Ultrasound *imaging system with image texture.*

In ultrasound imaging system, there could be a defocusing on image that leads to degradation in image quality, if the system pre-set speed of sound is away from the actual one in the analysis area. There is an idea to find the best speed of sound by creating a function of trial sound speeds with the image qualities. Therefore, we analyze the impact of mismatched sound speed first and then give out the sound speed optimization method with image textures.

2.1.1 The impacts of mismatched speed of sound on imaging qualities

In the ultrasound imaging system, the time delays (both TX and RX) is essential for the electronic focusing. They are used to make the pulse signals of each element arrive at a designated position in phase. These delay curves can be set to each transducer element in a format of time-of-flight computed by using a pre-setting speed of sound, e.g., 1540 m/s. For a linear or phased array, the time delay needed for the element i to focus at (x0, z0) can be represented as:

$$\tau_{\mathrm{i}} = \frac{1}{c_0}\left(\sqrt{(x_0 - x_i)^2 + z_0^2} - z_0\right) \tag{1}$$

Where c0 is the pre-setting speed of sound, xi is the position of the element i in the active aperture. From (1), one can see the delay curves are depend on the assumed speed of sound, which may cause the focus point shifted, when the actual speed of sound doesn't equal to the assumed one. Moreover, when the speed of sound of the actual area is smaller than the assumed speed of sound, the actual focal depth will be deeper than the designated position. Therefore, when the assumed time delay could not always be fit for the transducer elements, the mismatched acoustic velocity will cause a very imperfect focus.

More specifically, the focus length zfl on the displayed image in ultrasound imaging system can be computed as follows:

$$z_{\mathrm{fl}} = \frac{c_0 \tau}{2} \tag{2}$$

From the above equation, a shift for the focus length of the displayed image will be present when the pre-setting speed of sound is not close to the actual one in the medium. The actual focus length z_{true} should locate at

$$z_{\mathrm{true}} = \frac{z_{\mathrm{fl}} c_1}{c_0} \tag{3}$$

Eqn. (2) and (3) demonstrate that detail resolution in the ultrasound image is reduced using the mismatched speed of sound.

Based on the above discussion, we can see mismatched speed of sound would produce a shift in axial direction which will involve confusion regions in ultrasound image; especially the decimation of pulse-echo signal for display would exaggerate the degradation of detail resolution. The edge of tissue structure in ultrasound image will be blurred and the contrast will be also reduced.

2.1.2 Sound speed optimization with textures analysis

Image texture is a term that refers to properties that represents the surface of an object, so they can be used to describe the focus quality with some sound velocities. In statistical texture analysis, Unser's sum and difference histograms approach is an approximate the two-dimensional texture parameters for texture discrimination [7].

In this paper, we choose the image texture energy, mean and contrast as the focus quality factors. Texture Mean represents the intensity of local region. The better the focus quality is, the greater it is; Texture Energy can measure image homogeneity. The more homogeneous the image, the larger the value is. The other texture contrast is a measure of local image variations. High contrast textures are characterized by large edge magnitudes. It means the clearer boundaries and the higher resolution images, the greater value. The texture features with sum and difference histogram method are defined as follow:

Energy: $\sum_i P_s(i)^2 \sum_j P_d(j)^2$

Contrast: $\sum_j j^2 P_d(j)$

Mean: $\frac{1}{2}\sum_i iP_s(i)$

Where Ps and Pd are defined by

$$P_s(i;d_x,d_y) = Card\{(k,l) \mid y_{k,l} + y_{k+d_x,l+d_y} = i\}$$

and

$$P_d(i;d_x,d_y) = Card\{(k,l) \mid y_{k,l} - y_{k+d_x,l+d_y} = i\}$$

respectively. And Card refers to the number of elements of a set and yk, l = i means that the intensity at the position (k, l) is equal to the grey level of i.

After giving out the definition of image texture, the procedure of sound speed optimization is as follows:

- Preset a group of alternative sound velocities, and select a range in the displayed image. The selected rows should cover enough bins located at different resolution cells axially to increase the signal-to-noise ratio (SNR) for further averaging. Generally, the range should include the focus area.
- Obtain the ultrasound images produced by the group of sound velocities.
- Compute the textures in the selected range. In this paper, we use multiple analysis windows for texture calculations and then average them for high precision.
- Then the image quality factor is now defined as the triple of image textures.
- After normalization these image quality factors, we find the best one by weighting the triple of image textures for the corresponding trial speed of sound.

2.2 CUDA -Based Algorithm

There are four parts in our sound speed optimization method, embracing one necessary data transformation and three main GPU procedures: data acquisition, texture computation based on the sum and difference histograms, quality factors normalization and the optimized sound speed make-decision. The flowchart of this algorithm is shown in Figure.1.

2.2.1 Data acquisition

In our sound speed optimization method, we set a group of candidates firstly. And then use the digital ultrasound scanner built by Saset Healthcare Inc to obtain the corresponding images. Unlike the traditional CPU processing, the image data must be transferred from main memory to graphics memory in order to utilize the GPU platform. It's really extra time consumption for our GPU-based algorithm. To abate this extra cost and to achieve the high memory bandwidth, the choice of memory type for both host and device end is considered.

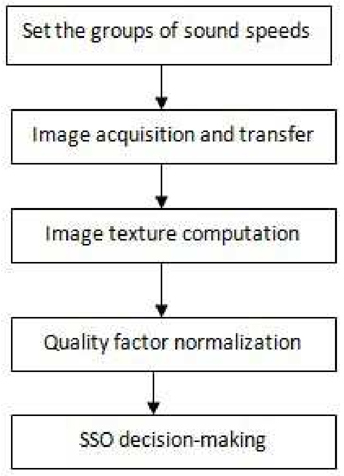

Set the groups of sound speeds
Image acquisition and transfer
Image texture computation
Quality factor normalization
SSO decision-making

Figure 1 Flowchart of sound speed optimization processing

In the host end, there are two types of main memories in host end, that's page memory and page-locked memory. The page-locked memory bandwidth is nearly 2 times higher than the page memory due to our actual tests. CUDA 2.3 or higher provide four kinds of page-locked memories with some special features. The write-combing memory frees up L1 and L2 cache resources, making more cache available to the rest of the application. In addition, write-combining memory is not snooped during transfers across the PCI Express bus, so it is suitable for the memory that the host only writes to; the default one is without the proprieties mentioned above. We use the write-combing memory to storage our image data.

In the device end, we put the data into the global memory, as the brand new GPU architecture Fermi provides the L1 and L2 cache for the graphics memory that improves the bandwidth much more. Note that if the GPU device is below CUDA2.0, the image data had better be put into texture memory for high bandwidth.

2.2.2 Texture computation

In the texture analysis, the key computation is to build the sum and the difference histograms. It is data dependence for producing one global histogram, as each bin can be set by multiple different position indexes with the same image pixels. Obvious, it is possible to distribute the computation process between multiple execution threads by subdividing the input data to create the sub-histograms and then merging them into a single global histogram. This naive and direct idea should be adapted following several constraints of GPU processing.

In the procedure of histogram producing, it is sequential to access to the data, but it is random (data-dependence) to modify the histogram bins. In order to achieve the high memory bandwidth, shared memory will be the most optimal storage for the sub-histogram. In the GPU device below 2.0, it is not possible to create the sum and different histogram for image data, for they need 512 bins while there is not enough shared memory for them. In the Fermi device (above 2.0), it has four times shared memory than the previous device. Thus it is possible to implement the sum and different histogram creation.

The warp (it is the smallest execution unit for GPU threads, 32 threads) shares the same memory range. We give out one method to produce the sub-histograms in thread warps and resolve the thread collisions. This method mainly includes the three steps:

- Each thread reads the previous value of the histogram bin. The most significant bits of the count are masked and replaced with the tag of the current thread. Then each thread writes the incremented bin back to the sub-histogram in the shared memory. Besides, the tag is stored in the 5 most significant bits of the histogram counters and only 5 bits are required for thread warp.
- When each thread in the warp receives unique data values, there are no collisions at all, and no extra actions are needed. But, when two or more threads colliding to write to the same location, the hardware performs shared memory write combining the results in the acceptance of the tagged bin from any one thread, and the rejection from the other pending threads. After the write attempt, each thread reads from the same shared memory location. The threads were able to write their bin, exit the loop and stay idle waiting for remaining threads in the warp. The warp will stop until all the threads exit the loop
- Each warp uses its own sub-histogram to produce the final histogram by merging it into the global memory.

After producing the sum and difference histogram, we can compute the three image textures in parallel.

2.2.3 Normalization

The key step for normalization is to obtain the maximum of parameters (energy, contrast and mean). To reduce the runtime of finding the maximum, one parallel method using Fan-in method on CUDA is proposed, and also utilizes the share memory, including the follow steps.

First, set the number of times the threads need to be synchronized. Here, we use 3 rounds to carry this on, twice thread-synchronization for each block and once kernel-synchronization for all blocks.

Second, let each thread find the first local maximum from the subdivision of data, then put it in the share memory and wait other threads in the same block.

Third, let any thread in every block finds the second local maximum from data of each thread block which have existed in share memory, and write it back to the global memory. Since it cannot be synchronized between different blocks before kernel stopped, another kernel must start to finish the last round.

Finally, a single thread is assigned to find the final maximum, in order to facilitate later normalization using many thread blocks, actually, not a single champion for all threads but one for a thread block, which is placed in the share memory. As GPU spends its transistors on ALUs, redundant computation is more efficient than accessing global memory repeatedly.

Two GPU passes do the first and second round of the maximum of the three textures, respectively. The maximum of all the three texture can be obtained in another kernel, which also finishes the normalization using the three maximum of all the textures.

2.2.4 Decision-making

To finish the decision-making, one can weigh the three factors using the preset coefficients from experiments, and then use the maximum of the weighting results to be the final output. On CUDA, we can use one GPU pass to weigh the factors and the function cublasIsamax to get the maximum of the non-zero vector [8].

3. RESULTS AND DISCUSSION

The software environment used Windows 7 and NVDIA CUDA v.4.0. The hardware environment comprised an AMD Althlon (tm) IIX2 240 CPU running at 2.81GHz and coupled to 2 GB of RAM, and an NVIDIA GTX 560 GPU running at 1.645 GHz with 14 multiprocessors and 1 GB memory. The results produced by CPU platform were calculated with general C code, without using any instruction extensions.

The test images are phantom images obtained from a digital ultrasound scanner built in our lab. Figure2(a) shows the speckle image obtained from 1540 m/s and (b) from 1700 m/s. Use a matched system sound speed, the speckle image of Figure2(a) is better than Figure2(b) in terms of speckle size around the focal region. Figure3(a) shows the point target image obtained from 1540 m/s and (b) from 1700 m/s. It's clear that a right system sound speed of 1540 m/s has better image quality of Figure. 3(a) compared with the right one of 1700m/s, Figure3(b). And the factors show the consistence with the image qualities.

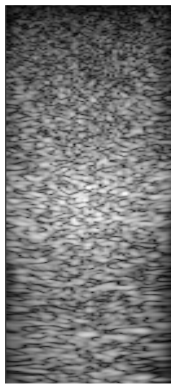
The performances of the CPU-based and GPU-based implementations are compared in Table I. The program runtime of the whole sound speed optimization processing in milliseconds from different image sizes and demonstrates that our CUDA-based implement will speed up this algorithm 45 times compared with that on the CPU platform in the same situation. Besides, the time delay is hidden and the GPU occupancy increase, as the image data size growing up. So the bigger data size is, the higher speedup is, before GPU resource is fully used.

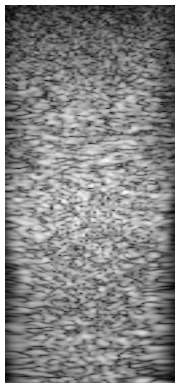
Table.1 Performance Comparison

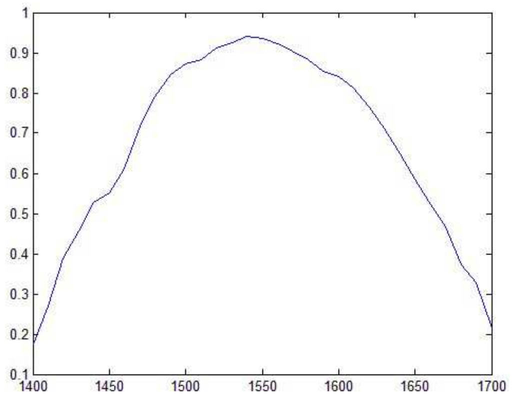
Image Size (pixel)	CPU(ms)	GPU(ms)	Speedup Rate
128 × 128×30	628.64	15.57	40.37
128 × 256×30	1254.32	28.80	43.55
256 ×256×30	2506.44	55.44	45.21

4. CONCLUSION

In the GPU implement, the sound speed optimization with image texture can be used for the real time ultrasound imaging system. Our tests demonstrate that the results of the decision-making by using GPU processing is the same as that on CPU; while the GPU implementation offers increased by more than 45 times. Above all, CUDA-enabled GPU could well suit for delivering high-speed image processing algorithms, like the computational intensive image texture analysis.


(a) 1540m/s(true)


(b)1700m/s(false)


(c) factors from CPU

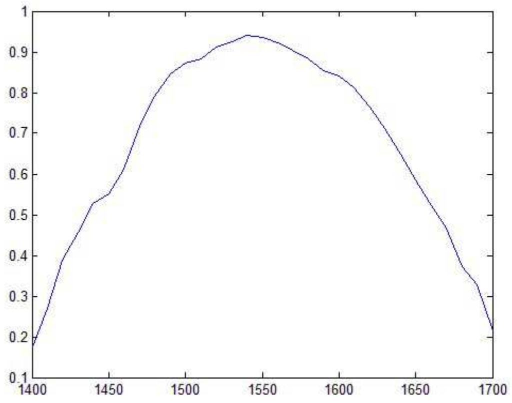
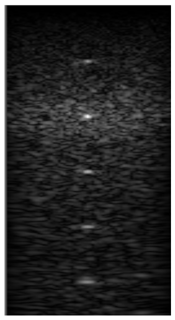
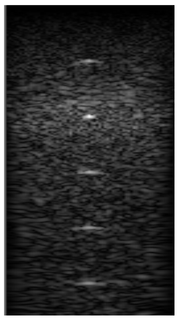
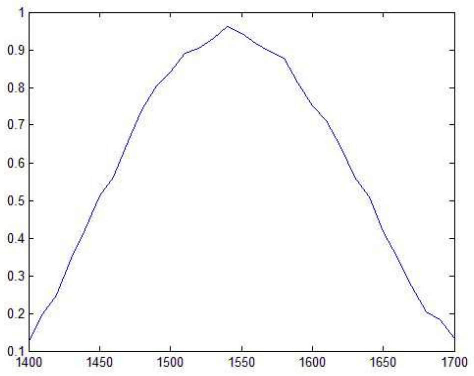

(d) factors from GPU

Figure 2 Speckle Image And Quality Factors


(a) 1540m/s(true)


(b)1700m/s(false)


(c) factors from CPU

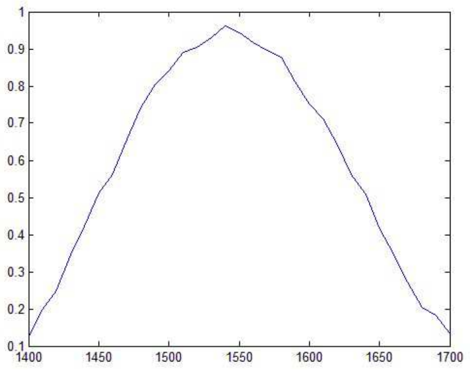

(d) factors from GPU

Figure 3 Point target image and quality factors

REFERENCES:

[1] G. D. Ludwig, "The velocity of sound through tissues and the acoustic impedance of tissues", *J. Acoust. Soc. Am*, Vol. 22, 1950, pp. 862–866.

[2] J. F. Krucker, J. B. Fowlkes and P. L. Carson, "Sound speed estimation using automatic ultrasound image registration", *IEEE Transl. on ultrasonics, ferroelectric, and frequency control*, Vol. 51, No. 9, September 2004.

[3] R.M. Haralick, K. Shanmugan, and I. Dinstein, "Texture features for image classification," *IEEE Trans. Syst. Man, Cybern*., Vol.SMC-8, Nov, 1973, pp:610-621.

[4] R.M. Haralick, K. Shanmugan, and I. Dinstein, "Texture features for image classification," *IEEE Trans. Syst. Man, Cybern*, Vol.SMC-8, Nov, 1973, pp.610-621.

[5] H. Du and D. C. Liu, "Simulation of sound speed optimization in the ultrasound system," *CBME2007 Chinese Biomedical Engineering Conference*(Chinese), 2007.

[6] CUDA Programming Guide, version 4.0, NVIDIA Co., Santa Clara, CA, 2011.

[7] M. Unser, "Sum and difference histograms for texture classification", *IEEE Trans. Pattern Anal. Mach. Intell*., Vol. PAMI-8, No.1, January, 1986, pp.118-125.

[8] NVIDIA: CUBLAS Library. version 4.0, NVIDIA Co., Santa Clara, CA, 2011