Manycore GPU Based High Performance Implementation of Ultrasound Color Doppler Imaging

Thi-Yen Phuong and Jeong-Gun Lee

Dept. of Computer Science, Hallym University, Chuncheon, South Korea

{ yanni, jeonggun.lee}@hallym.ac.kr

Abstract

The ability to detect and assess information of blood flow in color Doppler imaging (CDI) has played one of the most important roles in a modern ultrasound imaging system. However, since CDI requires large amount of data and computations, it has been mainly implemented on custom-designed hardware. Recent trend of programmable approach offers the advantages of flexibility and quick implementation. Specifically, general-purpose GPUs have been emerged as excellent accelerators across a wide range of applications. For best exploiting outstanding computational power, high memory bandwidth and SIMD architecture of a GPU, this paper explores the design space of CDI on GPU architecture platform and presents a high performance implementation of CDI on the GPU platform using CUDA API. The performance analysis shows our GPU-based CDI can achieve a frame rate of 152 for 800 range samples and 200 scan lines with an ensemble size of 12. Speedup of 19.8x can be obtained when compared with that on a CPU platform.

Keywords: *Ultrasound imaging; color Doppler imaging; GPU; parallel processing*

1. Introduction

Medical ultrasound imaging has been one of the most important and widely used technologies in medical diagnosis. It stands out from other medical imaging techniques such as magnetic resonance imaging (MRI), computed tomography (CT) and X-ray in terms of real-time applicability, portability and cost-effectiveness. In a diagnostic ultrasound system, three commonly used modes are B-mode for gray scale imaging, color Doppler for visualizing blood flow, spectral Doppler for tracking spectrum of the blood velocity. Even though B-mode is useful for observing and monitoring structures such as fetus or heart, it is impossible to visualize faster motion such as blood flow. The ability to detect and assess the velocity of moving blood cells is especially useful in detecting disorder in the cardio-vascular system. For example, the disturbance of flow in the arteries can be an indicator of stenosis. Therefore, CDI becomes one of the most important and commonly used technologies in ultrasound imaging.

Traditionally, most of commercial ultrasound systems are built on a dedicated hardware using ASICs and FPGA. High computing power and computational demands of ultrasound imaging makes its implementation on hardware complicated, time-consuming and costly. However, with a recent trend of programmable approach, software-based ultrasound imaging system can be implemented to offer the advantages of flexibility and adaptability. Specifically, with current successful researches in using GPUs as accelerations on various application domains, GPUs become a promising solution for moving from hardware-based to software-based ultrasound imaging system.

The main purpose of this study is to demonstrate the use of GPU on implementation of CDI under CUDA environment. Different strategies on how to optimize GPU framework is also presented. Performance analysis on different hardware platforms is discussed to give better understanding of GPU and CPU architectures.

This paper is organized as follows: Section 2 describes the algorithm of Doppler imaging system and briefly introduces GPU architecture and CUDA as a platform for parallel programming; Section 3 presents the implementation of CDI and performance analysis in detail and Section 4 concludes the paper.

2. Preliminaries

This section provides a general understanding on ultrasound imaging system and the algorithm of color Doppler Imaging.

2.1. Ultrasound Imaging

Figure 1 shows a simplified ultrasound system. A conventional medical ultrasound imaging system consists of several processing blocks: Beamforming process, Echo process (B-mode imaging), Color Doppler, Spectral Doppler and Scan conversion. The ultrasound acoustic signals are generated by converting 1 to 15 MHz electrical pulses from the transmitter into an acoustic wave using a piezoelectric transducer. As the acoustic wave pulse travels through tissue, a portion of the pulses is reflected at the interface of materials with different acoustical impedance. The reflected pulses are sensed by the transducer and converted into radio frequency (RF) electrical signals. The amplitude and time of returned pulses gives an image of the tissue structure. Each pulse is transmitted in slightly different directions to get all the scan lines needed to make a complete image. As the acoustic wave travels through the tissue, its amplitude is attenuated. Therefore, the receiver first amplifies the returned signal in proportion to its depth or the time required for the signal to return (*i.e.*, time-gain compensation, TGC). After the RF analogue signal is received and conditioned through time-gain compensation, it is typically sampled at a conservatively high rate. The demodulator then removes the carrier frequency by using techniques such as quadrature demodulation to recover the returned (echo) signal. The signal after demodulator is a complex baseband signal which contains both magnitude and phase information. Depending on how this information is processed, the image can be simply a gray-scale image of the tissue boundaries (known as echo imaging or B-mode) or a pseudocolor image where the color represents the speed and direction of blood flow (known as color Doppler) or the spectrum of the blood velocity at a single location over time (known as spectral Doppler) [1].

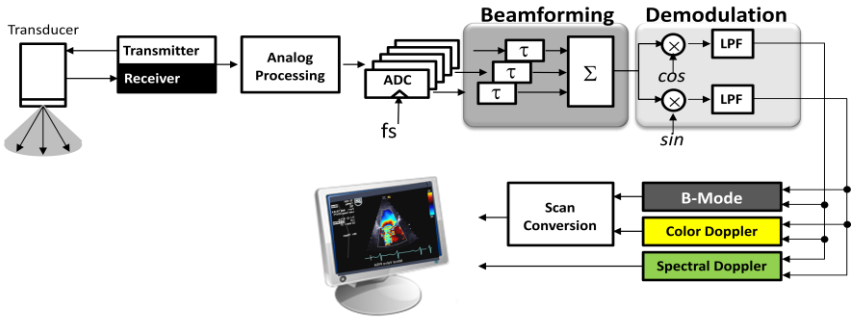


Figure 1. Block diagram of a typical diagnostic ultrasound machine

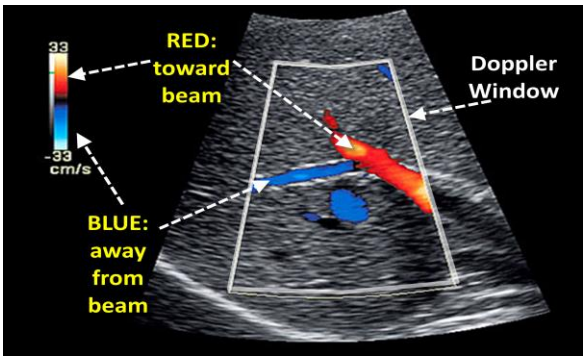


Figure 2. An example of color Doppler imaging

This paper only focuses on implementation of real-time color Doppler imaging which is one of the frequently used ultrasound modes by clinicians. CDI mode is used to estimate the velocity of moving blood cells by calculating phase shift or frequency change of echoes from the target. The velocity information (magnitude and direction) is presented as coded-color overlaid on top of the gray scale B-mode image. Typically red and blue colors are used to show flow towards and away from the transducer, respectively. The brightness of the color corresponds to the magnitude of the velocity. The main processing steps of CDI are wall filtering and velocity estimation. Figure 2 shows an example of color Doppler flow superimposed on a B-mode image.

2.2. Color Doppler Imaging

Color Doppler image system is divided into two main processing blocks: wall filtering and velocity estimation. Post-processing which includes image enhancement, color mapping and scan conversion is not included in our implementation.

2.2.1. Wall Filtering: In Doppler blood velocity estimation, signals consist of desired signals scattered from blood and clutter noise from the surrounding tissues and vessel walls. The frequency components due to wall motion are less than frequency of blood motion [1]. The clutter signals are generally much stronger (about 10dB to 40dB) than the scattered signals from the blood due to the smooth structures of the walls. Hence, a clutter high pass filter is necessary to remove these undesirable signals. Many high-pass filters including stationary echo cancelling, finite impulse response (FIR), and infinite impulse response (IIR), and regression filters have been developed to perform this task. They are designed to first estimate the clutter frequency and then use this estimated frequency to remove clutter signals effectively. The filtering can be expressed by a convolution operation of signal and filter function as shown in the equation 1:

$$(f * S)[n] = \sum_{m=0}^{N-1} f[m].S[n-m] , \quad (1)$$

where N is the length of the signal and M is the length of the filter. In this paper, a 4-tap- IIR high pass filter is used for wall filtering process.

2.2.2. Velocity Estimation: In a pulsed ultrasound system, the velocity of a sample volume is estimated from equation 2:

$$v \cong \frac{c.PRF.\Delta\varphi}{2\,.f_c.\cos\theta} = \frac{c.f_d}{2\,.f_c.\cos\theta}\,, \qquad (2)$$

Where c: Speed of sound in blood

PRF: Pulse repetition frequency

$\Delta\varphi$: Phase difference between two consecutive pulses

f_c : Transmitted frequency

θ (or insonation angle): Doppler angle between the direction of the motion and the ultrasound beam,

f_d : Doppler frequency shift

There are various techniques which have been proposed to estimate velocity of blood. One of the method is using Fast Fourier Transform (FFT). However, a high number of ensembles per scan line is required to obtain an accurate velocity resolution, which would considerably slow down the frame rate. Moreover, an unreliable velocity estimation will be achieved if only few ensembles are used. Therefore, other techniques are used and divided into two main categories: phase-shift and time-shift techniques. In the phase-shift or autocorrelation technique, the estimation is based on the phase shift from fixed-depth signals whereas in the time-shift or cross-correlation technique, a moving window is used to track blood movement over a changing depth to estimate the velocity [1]. In this study, we only focus on the design and implementation based on phase-shift (or autocorrelation) technique.

Autocorrelation-Based Velocity Estimation. Measurement the change in phase ($\Delta\varphi$) is the most common method to estimate the velocity. Multiple vectors (so-called ensembles) along a single scan line with the transducer stationary are acquired after demodulation. The vectors are divided into several range bins as shown in Figure 3. After that, the average change in phase at each range bin along the scan line is calculated through the equation 3:

$$\Delta\varphi(e,t) = \arctan\left(\frac{R_y(e,t)}{R_x(e,t)}\right), \qquad (3)$$

$$R(e,t) = R_x(e,t) + jR_y(e,t)\,, \qquad (4)$$

where $R_x(e,t)$ and $R_y(e,t)$ are the real and imaginary part of autocorrelation function R(e,t) of e^{th} ensemble at t^{th} range bin, respectively. $R_x(e,t)$ and $R_y(e,t)$ are calculated by equation 5 and 6:

$$R_x(e,t) = \frac{1}{E-1}\sum_{e=0}^{E-2}[I(e,t).I(e+1,t) + Q(e,t).Q(e+1,t)], \quad (5)$$

$$R_y(e,t) = \frac{1}{E-1}\sum_{e=0}^{E-2}[Q(e,t).I(e+1,t) - Q(e+1,t).I(e,t)], \quad (6)$$

where E is the ensemble size, varying from 4 to 12; I (in-phase) and Q (quadrature) are real and imaginary part of demodulated complex baseband signal S (S=I+jQ). The Doppler frequency shift and variance can be obtained from equation 6 and 7

$$f_d = \frac{PRF.\Delta\varphi}{2\pi}, \quad (7)$$

$$\sigma^2 = 2.(PRF)^2.\left(1 - \frac{|R(e,t)|}{R_x(0)}\right), \quad (8)$$

where $R_x(0)$ is the value at zeroth lag of autocorrelation function.

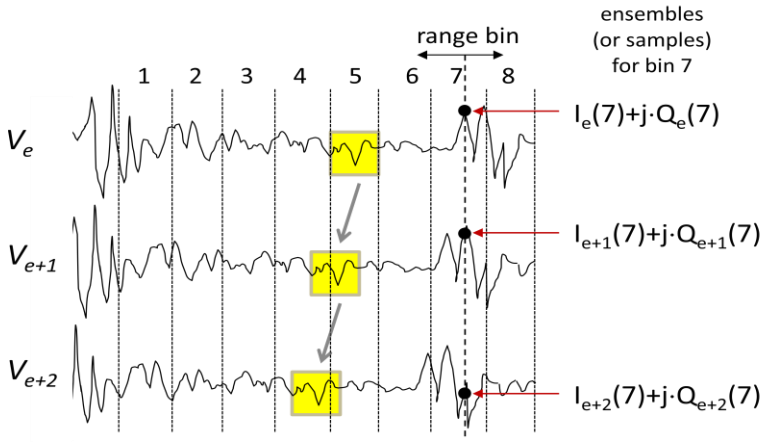


Figure 3. Example of autocorrelation velocity estimation

2.3. GPU Architecture and Programming Model

GPU architecture is made of a scalable number of streaming multiprocessors (SMs), each contains a number of parallel processing cores (SPs), warp schedulers, dispatch units, special function units (SFUs), local memory, shared memory, texture memory, L1 cache and constant cache. GPU has its own global device memory up to several gigabytes; it supports high memory bandwidth. Shared memory which is accessed as fast as register can be configured manually together with L1 cache. However, if there is a bank conflicts among threads within a warp, it slows down shared memory access. Local memory is used to store automatic variables in case of insufficient register space to hold variable. Access to local memory is as

expensive as access to global memory. Both constant memory and texture memory are cached. Reading data from constant cached is as fast as a register if all threads read the same address. However, accesses to different addresses from threads within a half warp will be serialized. The read-only texture cache is optimized for 2D spatial locality, so threads of the same warp reading texture addresses that are close to each other will achieve the best performance [4].

A group of 32 threads, so-called a *warp* is executed together, meaning that the same instruction is applied to all 32 threads (Single instruction multiple threads, SIMT). Even though accessing global device memory takes hundreds of clock cycles, GPU is designed to support zero-latency warp switching as long as there are enough active warps. As a result, if there is enough parallelism to make cores busy, they can run as productively and efficiently as they could.

Supported on NVIDIA GPUs, CUDA (Computed Unified Device Architecture) is a data parallel programming model. Host program launches a sequence of kernels with implicit barrier synchronization between kernels. Each kernel is organized as a hierarchy of light weight parallel threads which are grouped into blocks. A set of blocks are grouped into a grid that executes a kernel function. Each thread or block has a unique index in the grid. Threads within a block can be explicitly synchronized and share data using shared memory but blocks in a grid cannot be synchronized. The number of threads in a block is specified by programmers, however the numbers of blocks that are running on one SM are determined by the available resource of the SM and resource requirement of blocks. The challenge for programmers is to optimize GPU architecture features by writing efficient CUDA code so that it exposes enough parallelism to make full use of SIMD architecture, exploit full device memory bandwidth and maximize the utilization of shared memory and data cache efficiently.

3. GPU-based Color Doppler Imaging Implementation

This section presents the design and implementation method of CDI on GPU platform. Performance results and analysis are discussed to further understand the CDI algorithm and GPU hardware architecture.

3.1. Implementation Platform and parameters setting

In this paper, CDI algorithm is implemented under Windows 8 64-bit operating system and CUDA driver 5.0 running on the hardware platform of Intel Core i7-3770 CPU at 3.40GHz and 8GB of DDR3 RAM. Two GPUs are used: NVIDIA GeForce 660 OEM Kepler architecture (GK104) with computing capability of 3.0 at 0.89 GHz clock frequency with 1152 CUDA cores and 1.5GB global memory; NVIDIA GeForce 560 Ti Fermi architecture (GF114) with compute capability of 2.1 at 1.8GHz clock frequency with 384 CUDA cores and 1GB global memory. The experiment is done using parameters as described in Table 1.

Table 1. Experimental parameters

Parameters	Value
Speed of sound	1540 m/s
Number of ensembles, N_E	12
IIR Filter length, M	4
Pulse repetition frequency, PRF	1kHz

3.2. Implementation Method and Performance Analysis

Baseband I/Q data is stored as an array of $I/Q[N_S][N_L][N_E]$, where N_S is the number of range samples in a scan line, N_L is the number of scan lines and N_E is ensemble size. To process this data, it must be first transferred from CPU to GPU. Although memory copy to GPU takes a certain amount of time, the overall processing speed achieved by GPU still higher than that of CPU. As discussed before, the main processing steps of CDI are wall filtering and velocity estimation. They are implemented in different kernels.

3.2.1. Wall filtering: Since real and imaginary data processing are independent, the kernel takes advantage of *concurrent stream* so that two kernels can run concurrently. Therefore, clutter filtering is divided into two kernels: *filter_real()* and *filter_img()* for processing real and imaginary data, respectively. However, only small data size is benefited from streaming kernels since as bigger data size is, more resource needed, which leads to the fact that two kernels are unable to overlap each other. Each thread in the kernel produces filtered real or imaginary data after performing convolution operation with 4-tap IIR filter. After filtering, data length is reduced to N_E -M+1. Read-only filter coefficients can be kept in constant memory, global memory or texture memory. However, constant memory is expected to produce better results in convolution operation since all threads in a warp access the same memory address at a time. For these memory-bound kernels, it is extremely important to consider data array index arrangement to achieve highest memory bandwidth as possible. Therefore, the baseband I/Q array index in global memory is arranged in such a way that it exposes coalesced memory access pattern. Three-dimensional blocks are configured for best utilizing parallel SIMD architecture. After trials and testings, the best configuration is 512 threads/block ([32,8,2]). This configuration achieves 93.6% occupancy. Outputs of these kernels are signals without clutter noise, which are placed in global memory. These data are used for estimating Doppler frequency shift and velocity estimation later on.

3.2.2. Velocity Estimation: Velocity estimation is implemented in *veloc_est()* kernel. Each thread calculates flow parameters which are Doppler frequency shift, velocity and variance for each sample and each scan line. Unlike two previous kernels, in *veloc_est()* kernel, more computation is involved, which results in better latency hiding. To maximize instruction throughput, the kernel uses arithmetic instructions with high throughput, including trading precision for speed when it does not affect the end result (*-use_fast_math* functions), using single precision instead of double-precision. The best execution configuration in this kernel is two-dimensional blocks with 1024 threads/block ([64,16,1]), occupying **61.8%** hardware resource. While filter kernels above are optimized for higher occupancy to hide memory latency, increasing occupancy is not the best way to hide latency on *veloc_est()* kernel. Instead, large threads per block and small number of blocks per SM is preferable.

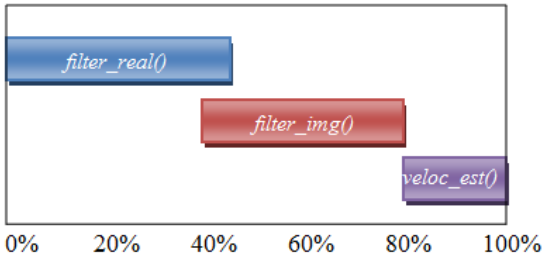


Figure 4. Execution timeline of kernels

Figure 4 shows the execution timeline of three kernels. Each filter kernel occupies about 40% of total execution time. That part of *filter_real()* kernel is overlapped by *filter_img()* kernel saves 8% of total execution time. It is true for as small as 200x50x12 data size. Overlapping time reduces as data size is bigger due to resource constraint. Total execution times of all kernels using different memory locations for filter coefficients are shown in Figure 5. As expected, filter coefficients are best stored in constant memory followed by global memory. However, there is very little difference on performance between constant memory and global memory in this case. Even though constant memory offers advantage of constant cache such as broadcast feature, the convolution algorithm does not need to use constant cache. All threads in a warp read the same address only one time, which costs one memory read from device memory. Therefore time taken for loading data from constant memory is almost equal to that from global memory. This also explains why the algorithm is not benefited from texture cache. Lower bandwidth of texture memory and overhead due to address computation for the textures produce execution time over 10x slower than using global memory or constant memory.

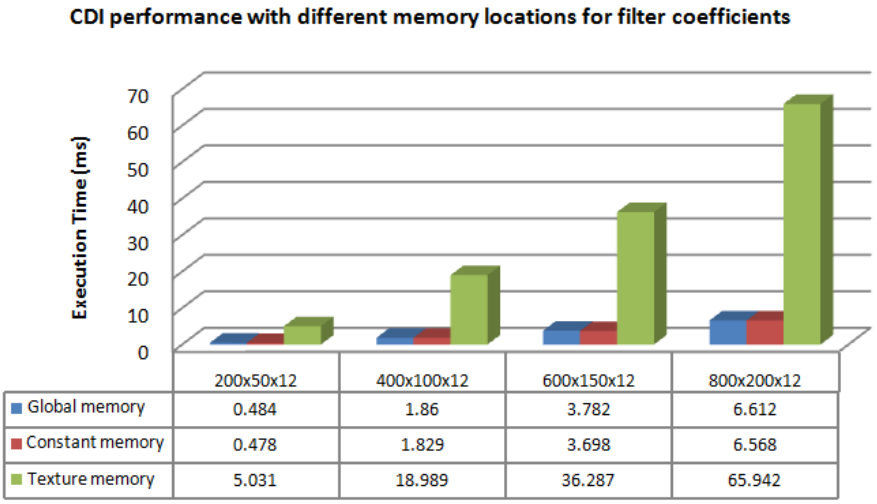


Figure 1. Total execution time (in ms on Fermi 560Ti platform) for different memory locations of filter coefficients

For all kernels, overall performance is improved up to 20% by setting ***cudaFuncSetCacheConfig****(kernel,cudaFuncCachePreferL1*) which configures L1 cache size to be bigger than shared memory size (48KB for L1 cache and 16KB for shared memory).

This is because L1 cache has beneficial impact on I/Q data access with high degree of locality.

Figure 6 shows the comparison between GPU-based and CPU-based implementation with various data sizes. It is interesting result that Fermi 560 Ti architecture outperforms newer generation, Kepler 660. Though Kepler 660 architecture has triple more number of cores than Fermi 560 Ti, the application actually runs faster on Fermi. One of the reason is that Fermi platform used in this implementation has GPU clock speed of 1.8GHz while Kepler 660 clock speed is only 0.89GHz (double slower). Moreover, all kernels in the application are ***memory-bound*** kernels which are directly affected by memory architecture such as memory bus width, L2 cache size and memory bandwidth as shown in Table 2. It indicates that a newer and more powerful with many-core GPU does not always produce better performance results in all cases.

For 800x200x12 volume size, speedup achieved by running the application on GPU is up to 19.8x compared to CPU in GTX 560 Ti case and 7.5x in GTX 660 case. For the same volume size, a frame rate of 152 can be achieved on Fermi 560 Ti architecture.

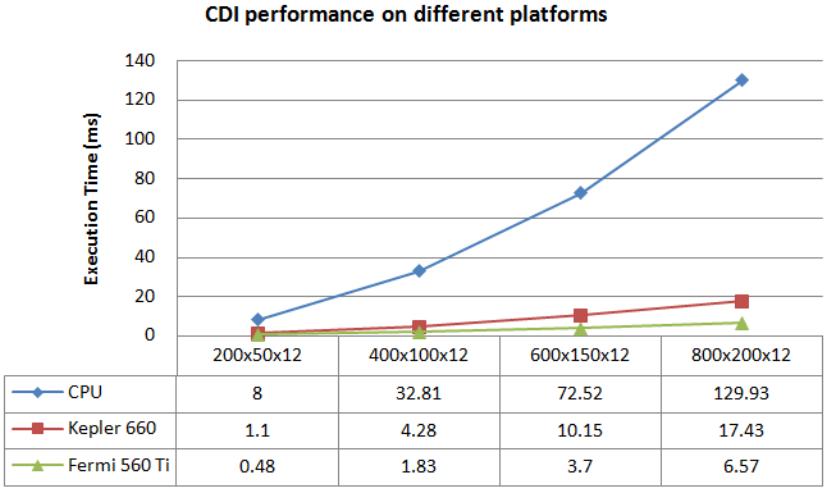


Figure 6. CDI performance on different platforms (in milisecond)

Table 2. Hardware specification comparison of Kepler 660 and Fermi 560 Ti

Specifications	Kepler 660	Fermi 560 Ti
GPU Clock speed	0.89GHz	1.8GHz
Memory bus width	192 bit	256 bit
L2 cache size	384kB	512kB
Device to device peak memory bandwidth	104GB/s	110GB/s

4. Conclusion

The paper presents an implementation of Color Doppler Imaging on GPU platform using CUDA parallel programming. A detailed analysis of the algorithm and how it is optimized are analyzed. Our experiments demonstrate that by employing various optimization strategies, the best speedup of GPU-based CDI system achieved is up to 19.8x over CPU for a volume data of 800x200x12. GPU framework provides advantages of flexibility, programmability and

easy-to-use design languages while meeting the real-time requirement of current as well as future expectation of CDI application.

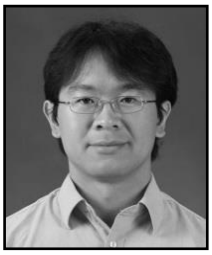
References

[1] G. York and Y. Kim, "Ultrasound processing and computing: review and future directions", Ann. Rev. Biomed. Eng., vol. 1, **(1999)**, pp. 559-588.
[2] H. K. So, J. Chen, B. Y. S. Yiu and A. C. H. Yu, "Medical Ultrasound Imaging: To GPU or Not to GPU?", Micro, IEEE, vol. 31, no. 5, **(2011)**, pp. 54-65.
[3] NVIDIA CUDA Best Practices Guide v.5.0, **(2012)**.
[4] NVIDIA CUDA Programming Guide v.5.0, (**2012**).
[5] C. Xia, A. Zhao and D. C. Liu, "Optimized GPU Framework for Ultrasound B-mode Imaging", Bioinf., Biomed., Engin., **(2010)**, pp. 1-4.
[6] G. Pratx and L. Xing, "GPU computing in medical physics: A review", Medical Physics, vol. 38, no. 5, **(2011)**.
[7] L. -W. Chang, K. -H. Hsu and P. -C. Li, "Graphics Processing Unit-Based High-Frame-Rate Color Doppler Ultrasound Processing", Ultrasonics, Ferroelectrics, and Frequency Control, IEEE Transactions on, vol. 56, Issue 9, **(2009)**, pp. 1856-1860.
[8] L. -W. Chang, K. -H. Hsu and P. -C. Li, "GPU-Based Color Doppler Ultrasound Processing", IEEE Ultrason., Symp., **(2009)**, pp. 1836-1839.
[9] T. Fukuoka, F. K. Schneider, Y. M. Yoo, A. Agarwal and Y. Kim, "Ultrasound color Doppler imaging on a fully programmable architecture", Proc. IEEE Ultrason. Symp., **(2006)**, pp. 1639-1642.
[10] C. Kasai, K. Namekawa, A. Koyano and R. Omoto, "Real-time two dimensional blood flow imaging using an autocorrelation technique", IEEE Trans. on Sonics and Ultrason, vol. 32, **(1985)**, pp. 458-464.

Authors

Thi-Yen Phuong

She is currently a master student in Computer Engineering at Hallym University in South Korea. She received her bachelor's degree of Electrical and Electronics Engineering in University Tenaga National, Malaysia in 2011. Her research interests are many-core microprocessor, system-on chip design and GPU-based parallel processing.



Jeong-Gun Lee

He received the B.S. degree in computer engineering from Hallym University in 1996, and M.S. and Ph.D degree from Gwangju Institute of Science and Technology (GIST), Korea, in 1998 and 2005. He is currently an assistant professor in the Computer Engineering department at Hallym university. Prior to joining the faculty of Hallym University in 2008, he was a postdoctoral researcher of the Computer Lab. at the University of Cambridge, UK.