High-speed optical coherence tomography signal processing on GPU

To cite this article: Xiqi Li et al 2011 J. Phys.: Conf. Ser. 277 012019

View the article online for updates and enhancements.

Related content
- GPUs for event reconstruction in the FairRoot framework
  M Al-Turany, F Uhlig and R Karabowicz
- GPU-accelerated 3-D model-based tracking
  J Anthony Brown and David W Capson
- Graphics processing unit aided highly stable real-time spectral-domain optical coherence tomography at 1375 nm based on dual-coupled-line subtraction
  Ji-Hyun Kim, Jae-Ho Han and Jichai Jeong

Recent citations
- Dual-band Fourier domain optical coherence tomography with depth-related compensations
  Miao Zhang et al
High-speed optical coherence tomography signal processing on GPU

Xiqi Li\textsuperscript{1,2,3}, Guohua Shi\textsuperscript{1,2,3}, and Yudong Zhang\textsuperscript{1,2}
\textsuperscript{1}The Laboratory on Adaptive Optics, Institute of Optics and Electronics, Chinese Academy of Sciences, Chengdu 610209, China
\textsuperscript{2}The Key Laboratory on Adaptive Optics, Chinese Academy of Sciences, Chengdu 610209, China
\textsuperscript{3}Graduate School of the Chinese Academy of Sciences, Beijing 100039, China

E-mail: lixiqi@yahoo.cn

Abstract. The signal processing speed of spectral domain optical coherence tomography (SD-OCT) has become a bottleneck in many medical applications. Recently, a time-domain interpolation method was proposed. This method not only gets a better signal-to-noise ratio (SNR) but also gets a faster signal processing time for the SD-OCT than the widely used zero-padding interpolation method. Furthermore, the re-sampled data is obtained by convoluting the acquired data and the coefficients in time domain. Thus, a lot of interpolations can be performed concurrently. So, this interpolation method is suitable for parallel computing. An ultra-high optical coherence tomography signal processing can be realized by using graphics processing unit (GPU) with computer unified device architecture (CUDA). This paper will introduce the signal processing steps of SD-OCT on GPU. An experiment is performed to acquire a frame SD-OCT data (400A-lines×2048 pixel per A-line) and real-time processed the data on GPU. The results show that it can be finished in 6.208 milliseconds, which is 37 times faster than that on Central Processing Unit (CPU).

1. Introduction
Optical coherence tomography (OCT) is a non-invasive, cross-sectional optical imaging technique that allows for high-sensitivity, high-resolution imaging of backscatter samples [1,2,3]. Because of system simplicity and economical design, spectral domain optical coherence tomography (SD-OCT) has undergone rapid development. SD-OCT is based on spectral interferometer and detected by a line scan charge coupled device (CCD). The interference signal I(\(k\)) can be expressed as:

\[
I(k) = S(k)(1 + 2\Omega_2 a(z)\cos(2knz)dz + \Omega_2 \Omega_2 a(z)a(z)e^{-2kn(z - z')}dzdz')
\]

Where I(\(k\)) denotes the detected spectrum; S(\(k\)) denotes the spectral intensity distribution of light source; a(\(z\)) is the scattering amplitude. The a(\(z\)) can be obtained by Fourier Transformation of the detected spectrum I(\(k\)). It is obvious that the intensity information is coded in the wave number \(k\). Unfortunately, the CCD detects the signal evenly in wavelength. It need resample the data from wavelength space to wave number space, or else, it lead to coherence envelop broadening and fall-off degradation in SD-OCT systems [4]. Fig.1 shows
the signal processing procedure. The intensity data \( S_{\text{out}}(x) \) of the spectrometer is acquired by the linescan CCD. The interference spectrum \( S_{\text{int}}(x) \) is obtained after reducing the dc terms and the low-frequency noises by the background subtraction procedure. The re-sampled data \( S(k) \) is obtained by a non-uniformly sampled, which distributed linearly in wave number spaced.

**Fig.1.** Flow chart of SD-OCT structure imaging procedure.

Re-sampling is an important part in the signal processing step. Because \( k \) and \( \lambda \) has a relationship of \( k = \frac{2\pi}{\lambda} \) and \( \Delta k = \frac{2\pi \Delta \lambda}{\lambda^2} \), and the frequency oscillation period is as small as two pixels in the spectrometer, resulting in the low processing speed. Thus, a lot of SD-OCT systems are post processing. Real-time optical coherence tomography processing by using Digital Signal Processing (DSP) or field-programmable-gate-array-based (FPGA) was proposed [5-8], which increase the system complexity and cost because additional hardware inclusions are needed. A real-time swept source polarization-sensitive OCT system by using OpenMP model has been reported [9], but that needs expensive multi-core central processing unit (CPU).

In the last ten years, graphics processing unit (GPU) becomes more and more powerful in parallel computing [10,11]. Its special architecture is well suitable for the problems that can be parallel executed. Commodity GPU like NVIDIA’s GTX 295 has 480 processing cores and can achieve 1784 G float point operation per second (FLOPS) of computational horsepower, which is much faster than that of the current CPU. But traditional tools for computing on GPU were too complex to program, limiting the general purpose use of GPU. Recently, NVIDIA Corporation develops a tool for computing on GPU, which is named as the compute unified device architecture (CUDA) programming model [10,11]. With the help of CUDA model, a general purpose computing application can be developed easily. A real-time 4D signal processing and visualization using GPU on a regular nonlinear-k Fourier-domain OCT system is reported [12]. But the linear spline interpolation (LSI) they used is barely improve the signal-to-noise ratio (SNR) of the SD-OCT system. A high speed and high accuracy interpolation method is needed.

A time domain interpolation for Fourier domain optical coherence tomography was reported [1]. This method not only obtains a better SNR but also gets a faster computing speed than commonly used zero-padding interpolation [1]. Furthermore, by this interpolation method, the linear k space data can be obtained by convoluting the original data detected in CCD with interpolation coefficients [1].Thus, only the data and coefficients in cut-off window are processed. So, this interpolation method is suitable for parallel computing.

This paper will briefly introduce the time-domain interpolation. A CUDA program is developed to utilize the real-time parallel computing capability of the GPU. This program follows the steps as Fig.1 shows. These steps are processed step by step, but every step is processed by a lot of threads concurrently, which can accelerate the processing speed of SD-OCT significantly.

2. Time domain interpolation

In SD-OCT system, interference signal \( x_n(n) \) is acquired by the pixels of the linescan CCD, whose corresponding wavelengths are linearly distributed in wavelength space. We assumed that there has a sequence of wavelengths linearly distribute in \( k \) space, whose interference data are \( x(s) \). The virtual position index \( s \) is obtained by mapping this sequence to CCD pixel index \( n \). The interpolated data can be obtained by using the follow equation [2]:

\[
D_{\text{ata acquisition}} \rightarrow S_{\text{int}}(x) \rightarrow \text{Background subtraction} \rightarrow S_{\text{out}}(x) \rightarrow \text{Re-sampling} \rightarrow S(k) \rightarrow \text{FFT} \rightarrow \text{Image creation}
\]
The range of index $n$ and $s$ is determined by the pixels of the CCD $N$ and the resolution of the OCT system. The coefficients are defined as:

$$x_2(s) = \frac{1}{N+1}\sum_{j=0}^{N-1} x_i(n) + 2\frac{\hat{a}}{N} \sum_{j=1}^{N} \cos \left( \frac{2\pi}{N} j(s-n) \right)$$ \hspace{1cm} (2)

$$C(D) = \frac{1}{N+1}\sum_{j=0}^{N-1} + 2\frac{\hat{a}}{N} \cos \left( \frac{2\pi}{N} jD \right)$$ \hspace{1cm} (3)

$s,n$. Eq. 3 shows that the coefficients do not relate to the data. It can be computed separately if $s$ and $n$ known. The virtual CCD position of the linear $k$ space can be obtained by calibrating the spectrometer of the SD-OCT system.

It has reported that only a small number ranges of $C(D)$ contribute to the interpolation. Therefore, a narrow window can be used to cut off the $C(D)$ with an acceptable accuracy to reduce the computation time, whose cut off width is $L[2]$. The larger $L$ can get a better accuracy, but it would consume more computing time. Thus, the cut off width $L$ is determined by a trade-off between the accuracy and the computation time [2]. The interpolation can be finally expressed as:

$$x_2(s) = \max_{n=\text{Min}}^{\text{Max}} x_n H_{N,N'}(n,s_n) W(n)$$ \hspace{1cm} (4)

Where Min and Max are determined by the cut-off length and the position $s$. $H_{N,N'}$ is the coefficients matrix. $W(n)$ is the cut-off windows. A rectangle window is suitable. Eq.4 shows that $x_2(s)$ are only relate to several original data and the coefficients range in the cut-off window. So the sequence $x_2(s)$ can be computed out by a lot of parallel streams.

3. Signal Processing on GPU

Figure 2 shows the setup of SD-OCT. The light source is a SLD (Superlum, Russia, SLD-371-HP1) with a bandwidth of 45 nm FWHM at 840 nm. The spectrometer consists of a line scan camera (AVVIVA, SM2 CL2015, 28 KHz at 2048 pixels), a transmission grating [Wasatch Photonics, 1200 (lines $\times$ pairs)/mm at 830 nm], and an achromatic lens ($f=150$ mm). Two galvo mirrors are driven by a waveform generator (PCI 6221, National Instrument) to control the scan light. The interference is detected by the CCD and transmits to computer after it was grabbed by the image acquisition device (IMAQ) (PCIe 1430, National Instrument). A computer with a signal-core CPU AMD sempron 3600+ at 2.0GHz and a GPU NVIDIA GTX295 is used as the data acquisition and processing computer. The operating system was Microsoft Windows XP pro SP2 and the software developing platform was Microsoft Visual Studio 2005 SP1.
In order to utilize capability of the GPU and speed up the imaging processing time, a NVIDIA GTX 295 GPU is used to do the data processing instead of CPU. In CUDA model, GPU is treated as a dedicated coprocessor of the host CPU. CUDA allows the same code to be simultaneously run on different cores of GPU as threads [10,11]. All threads are organized into thread blocks, which are executed concurrently on one Single-Instruction Multiple-Data (SIMD) stream multiprocessor (SM). A GPU has two types of memory: global memory and SM on-chip memory. Each SM has four different types of on-chip memory, namely, constant cache, texture cache, registers, and shared memory [10,11]. Different type’s memory has different properties. In order to gain optimal performance when utilizing CUDA, the user must determined the thread blocks, the shared memory, registers, and the global memory usage.

Figure 3 shows data flow of GPU computing for SD-OCT. Data is acquired by IMAQ and transmitted to a bulk page-locked memory, in which system can get a high transmission performance. And then, the data is transmitted to the global memory of the GPU for parallel computing. After the data is processed by GPU, it is transmitted to page-locked memory again. Finally, it is showed by the display screen. All the operations are scheduled by CPU. In order to utilize the capability of the GPU, the parallel computing on GPU follows the steps as Fig.4 shows.
3.1. Background subtraction

\[ x' = x - DC \]

Symbol \( x' \) stands for the interference data of the SD-OCT system. The symbol \( DC \) stands for the spectral intensity distribution of the light source. It is the sum of the all lines’ acquired data and then divided by the line number. This step is not suitable for parallel computing.

3.2. Re-sampling

In CUDA architecture, global memory read function would consume a lot of time, usually 400 to 600 clock cycles [10,11]. As discuss above, one point’s interpolation only relates to the acquired data and coefficients in the cut-off window. A lot of interpolations use the same acquired data and a lot of lines use the same coefficients. If all the data stored in global memory, it would consume a lot of read memory time. Additionally, if a lot of blocks read the same memory, it would result in block conflicts [10,11]. So, it’s better to share these data and coefficients by reading them into every block’s shared memory before interpolation take place. The read speed of the shared memory can reach the read of the registers, which is the least communication latency in threads. It can be realized like this:

\[
\text{__shared__ float data [block size][begin to end];}
\]
\[
\text{__shared__ float coefficient [block size][ cut-off length];}
\]

By using the shared memory, every block read the acquired data and the coefficient once, reducing the read latency and block conflict. After this step, interpolation is operated by multiply-add functions as Fig. 3 shows. A point can be operated by a thread. So a lot of points can be operated concurrently, reducing the computing time.

3.3. FFT, FFT shift and get modulus

Depth information can be retrieved by performing an inverse FFT algorithm. The NVIDIA CUDA fast Fourier transform library (CUFFT) provides a simple interface for computing parallel FFT on an NVIDIA GPU, which allows users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-Base FFT implementation [13]. By setting the types, transform directions and size of the application programming interface (API), FFT can be operated automatically. The data obtained from FFT operations are complex quantities and the sequence is not distributed well. So, it needs the FFT shift and get modulus operations to do the pre-processing for the image creation.

3.4. Image creation

The demodulated data from FFT is complex, so it needs to map to structure image by Eq.5 [2]. A lot of intensity also can be operated concurrently in this step. A pixel of the image is obtained by combining two pixels of the acquired data and restored in 8 bit in step.

\[
\text{Intensity} = \text{Contrast'} (10^{- \log_{10}(x'(s'))} + \text{Brightness}) + 255 \tag{5} \]

The resolution of GPU computing supports 32 bit and 64 bit [10,11]. The 64 bit’s computing will consume much more time than that of the 32 bit’s. Furthermore, the error between the accuracy of 64 bit and 32 bit is acceptable for OCT applications [9]. Thus, 32 bit’s computing is adopted in this program.

4. Result and discussion

In order to compare the processing capability of the GPU and CPU, the computing time of the steps is measured by CUDA event function. A CUDA event is started and stopped at the beginning and ending, respectively. The computing time of a program is obtained by recording the time between the two events, whose resolution can be reach to microseconds (us). A speed
up ratio (R) is defined as R = \frac{t_c}{t_g} - 1, where t_c and t_g is the computing time on CPU and on GPU, respectively.

An Experiment is performed to obtain the computing time. One frame data contained 400 (A-lines per frame) \times 2048 (pixels per lines) \times 12 (bits per pixel) is acquired by the image acquisition device in SD-OCT. Table 1 shows every step’s computing time on GPU and that of on CPU. A single core CPU is adopted to compare the parallel capability of GPU. The difference between the time for copy data to GPU and the time for copy data to host is that the image data is compressed and stored in 8 bit per pixel. So, the time for copy image data to host is less than the time for copy acquired data to GPU. The R of the background subtraction is less than others is because it is The Computing time on GPU is much faster than that of on CPU, which has folded more than 37 times computing time. It should be noted that the R depends the capability of the CPU and GPU. Different CPU and GPU result in different R. It’s easy to realize real-time video rate SD-OCT imaging by this software program. The computing capability can be achieved 60,000 A-lines per second, which is much faster than the acquisition speed of the linescan camera. Thus, the acquisition speed becomes the imaging limitation of SD-OCT system.

| Method                  | GPU time (ms) | CPU time (ms) | R   |
|-------------------------|---------------|---------------|-----|
| Copy acquired to GPU    | 0.463         |               |     |
| Background subtraction  | 2.047         | 31.42         | 14.35 |
| Interpolation\(^a\)     | 2.060         | 87.18         | 42.32 |
| FFT                     | 0.854         | 52.63         | 60.63 |
| Image creation          | 0.689         | 67.09         | 96.37 |
| Copy image data to Host | 0.095         |               |     |
| Total time              | 6.208         | 238.32        | 37.90 |

\(^a\)Note: the computer platform: CPU, AMD sempron 3600+ 2.0 GHz; Memory: 2G DDR2 800; GPU, NVIDIA GTX 295.

In conclusion, we have described a new way to accelerate the signal processing speed of SD-OCT. The computing time of time-domain interpolation on GPU can be 42-folded times less than that on CPU. The total imaging processing time of SD-OCT has been improved by more than 37 times. The total signal processing of SD-OCT can be finished in 6.208. So, with the help of the GPU, real-time SD-OCT data processing is realized. Furthermore, CUDA model supports multi GPUs. The user can get a higher computing capability by using several GPUs, which can process more A-lines per frame in real-time.

5. References
[1] Yudong Zhang, Xiqi Li, Ling Wei, Kai Wang, Zhihua Ding, and Guohua Shi, Opt. Lett. 34 1849
[2] James G. Fujimoto, in Handbook of Optical Coherence Tomography, Brett E. Bouma and Guillermo J. Tearney, eds, (Marcel Dekker, Inc. 2002) Chap.1.
[3] R. Leitgeb, C. Hitzenberger, and Adolf Fercher, Opt. Express. 11 889
[4] Christophe Dorrer, Nadia Belabas, Jean-Pierre Likforman, and Manuel Joffre, J. Opt. Soc. Am. B. 17 179
[5] A. W. Schaefer, J. Joshua Reynolds, D. L. Marks, and S. A. Boppart, IEEE Trans. Biomed. Eng. 51 186
[6] S. Yan, D. Piao, Y. Chen, and Q. Zhu, J. Biomed. Opt. 9 454
[7] J. Su, J. Zhang, L. Yu, H. G. Colt, M. Brenner, and Z. Chen, J. Biomed Opt. 13 030506
[8] T. E. Ustun, N. V. Iftimia, R. D. Ferguson, and D. X. Hammer, Rev. Sci. Instrum. 79, 114301.
[9] Gangjun Liu, Jun Zhang, Lingfeng Yu, Tuqiang Xie, and Zhongping Chen, Appl. Opt. 48, 6365.
[10] NVIDIA CUDA. http://developer.nvidia.com/object/cuda.html.
[11] NVIDIA CUDA computer Unified Device Architecture: Programming Guide, Version 2.0beta2, Jun. 2008.
[12] Kang Zhang and Jin U. Kang, Opt. Express. 18, 11772.
[13] NVIDIA CUDA CUFFT Library, Version 1.1, Oct, 2007.