Real-time Video Denoising for 2D Ultrasound Streaming Video on GPUs

B. Dolwithayakul, C. Chantrapornchai*
Department of Computing, Faculty of Science
Silpakorn University
Nakhon-Pathom, Thailand
*ctana@su.ac.th

N. Chumchob
Department of Mathematics, Faculty of Science
Silpakorn University Nakhon-Pathom, Thailand and
Centre of Excellence in Mathematics, CHE, Si Ayutthaya Rd., Bangkok 10400, Thailand

Abstract — The ultrasound videos are mainly contaminated by multiplicative noises but also contaminated with additive noises. As the past few decades, there are some studies to remove the noises from ultrasound images as in the JY model [1] and the variational model which removes both types of noises. However, denoising these noises from the ultrasound video is the time-consuming process. With the advancement of multi-core and many-core processors, it makes the denoising process much faster and it is possible to render while doing the real-time denoising. In this study, we propose the modified strategy from [2] to denoise the streaming ultrasound video in real-time. Our proposed model can retain the frame order, and get the satisfactory frame rate (about 14.98 fps). The proposed strategy boosts the speedup of the frame denoising to 3.79 times compared to the sequential computation.

Keywords: Real-time video denoising; Parallel computing; OpenMP; Graphic Processing Units (GPUs); multi-core; CUDA; Image processing; Ultrasound video.

I. INTRODUCTION

In the real-world, the usage of the image sensor such as video camera, sonar, and ultrasound usually incurs the noises to the media. The noises may cause degrading in image quality or video quality, losing some important information in the media. In the past decades, there are a lot of studies aimed to restore images or videos from noise.

In the mathematical area, image noises are categorized into two categories: additive and multiplicative noises. The additive noise can be written as

\[ z = u + \eta \]

where \( z \) is the noisy image, \( u \) is the noise-free image and \( \eta \) is the noise in the image. The multiplicative noise can be expressed as

\[ z = u \eta \]

The ultrasound image and image obtained from Synthetic Aperture Radars (SARs) such as radars and satellites [12] contains both additive and multiplicative noises as suggested by Hirakawa and Parks [4]. These noises can be expressed as (3):

\[ z = u + (k_0 + k_1 u)\eta \]

where \( k_0 \) and \( k_1 \) are parameters indicating how much additive and multiplicative noises in the image. These noises require more complicate model to remove them.

However, the ultrasound image has a bit difference noise model as (4)

\[ z = u + (k_0 + k_1 \sqrt{u})\eta \]

For this study, we assume that each frame has same level and ratio\((k_0/k_1)\) of additive and multiplicative noises and has no dependency or temporal noises on between frames.

The video denoising is more complicate than just still-image denoising. There are complications due to the following aspects.

1. Frame rate. Normally the human eyesight can process about 10-12 frames per second [5]. As suggested by [5], the real-time video frame rate should be normally higher than 15 frames per second so the latency will not be noticed.
2. Frame order. The denoised video frames order must be retained. In the process, the output frames need to be merged in the correct order.
3. Frame rate control. The output video should have stable frame rate for the entire video playback to guarantee the quality of service.

With these challenges, video denoising usually cannot be done in real-time due to the extensive computation. With the current multi-core and many-core technology such as multi-core processor and graphic processing units (GPU), it makes the video denoising possible.

In this paper, we extend the previous work from [2] which uses the ROF model [8] for denoising videos in real-time using GPUs. The work in [2] can remove only additive noise with GPUs in real-time. This study aims to remove noises from the
streaming ultrasound videos which contain both additive and multiplicative noises using both GPU and OpenMP technology.

II. Backgrounds

This section consists of two parts: First, we show the denoising model we used in this work. The next subsection shows the CUDA architecture for GPU computing [6, 7].

A. Variational Model Noise Removal Algorithm

The variation model for restoring an image contaminated with both additive and multiplicative noises can be modified from Equation (3) as described by Equation (5),

\[ z = u + k_2 \eta + k_1 \sqrt{u \eta}, \]

where \( \eta \) is the additive noise and multiplicative noise, respectively.

Due to the independence of additive and multiplicative noise, we can measure these noises using Equation (6)

\[ D[u] = \frac{\alpha_1}{2} \int_{\Omega} (u - z)^2 d\Omega + \frac{\alpha_2}{2} \int_{\Omega} (z - u)^2 d\Omega \]

Here \( \alpha_1 > 0 \) and \( \alpha_2 > 0 \) are the regularized fitting parameters for the additive and multiplicative noise removals, respectively. \( \Omega \) is the domain of the image. By using Euler-Lagrange equations, the denoising model is expressed as

The variation model and JY Model[1] for removing both additive and multiplicative noises is given by Equation (7)

\[ \min_{u, \Omega} \{ J_{\alpha_1, \alpha_2}(u) = D[u] + R[u] \} \]

where

\[ R[u] = \int_{\Omega} \nabla u |_{\beta} d\Omega = \int_{\Omega} u_x^2 + u_y^2 + \beta d\Omega, \beta > 0 \]

According to the calculus of variations the Euler-Lagrange equation from Equation (7) is given by:

\[ -\nabla |_{\beta} \nabla u + \alpha_1 (u - z) + \alpha_2 \left( 1 - \frac{z^2}{u} \right) = 0 \]

\[ K(u) = \nabla |_{\beta} \nabla u = \sqrt{|\nabla u|^2 + \beta} \]

where \( \alpha_1 \) and \( \alpha_2 \) are the fixed point iterative method for discretization \( \Omega \) to the discrete domain \( \Omega_x \), where \( h \) is the distance between each grid point, we discretize the domain into \( n_x \times n_y \) grid cells. Each cell has the size of \( 1 \times 1 \) (\( h_x = h_y = 1 \)). The discrete equation on \( \Omega_x \) on \( \Omega_x \) is obtained by Equation (10)

\[ -K^x(u^k)_{i,j} + \alpha_1 ((u^k)_{i,j} - (z^k)_{i,j}) + \alpha_2 \left( 1 - \frac{z^k}{u^k} \right) = (g^k)_{i,j}, \]

where

\[ K^x(u^k)_{i,j} = \frac{\delta_j \left( D(u^k)_{i,j}, \delta_i (u^k)_{i,j} \right)}{h_x} \frac{\delta_i \left( D(u^k)_{i,j}, \delta_j (u^k)_{i,j} \right)}{h_y} \]

From Equation (10), there are several methods for solving it. For example, Time Marching technique is the simple iterative technique using a synthetic time variable [13]. However, this method is slowly converged to the solution and not suitable for the parallel computing because of the data dependency in each iteration.

Alternatively, the fast and robust method for solving Equation (10), called Fixed-Point iterative method, is proposed by Vogel and Oman [9, 10]. This method works by freezing some coefficients and converting the problem into a system of linear equations, which can be solved by an iterative solver such as Gauss-Seidel, a modern solver technique such as multi-grid(MG), or a preconditioned conjugate gradient(PCG). However, our previous researches showed that Gauss-Seidel method is able to obtain a satisfactory convergence rate with an acceptable accuracy. Thus, in this work, the local fixed point iterative method is used because it is highly parallelizable and easy to implement on the both the multi-core CPU and the GPUs.

B. Compute Unified Device Architecture (CUDA)

Compute Unified Device Architecture (CUDA) is an architecture for Single Instruction Multiple Data(SIMD) from NVIDIA®. Despite of graphic processing, this render graphic card with CUDA can be used as a general-purposed processor, which is called the General-Purposed Graphic Processing Unit(GPGPU).

CUDA has 4 levels of memory. The first level is called "Global memory". It is the slowest memory accessed by the GPU. Hundreds of clock cycles are needed to access the global memory. The next level is called "shared memory," which is the fastest memory that a user can allocate and manage on the GPU device. Reading and writing through the shared memory uses approximately 40 clock cycles. Another two levels are local memory and texture memory. Both are large memories and can be allocated by users. To access them, more cycles are needed when compared to accessing the shared memory. However, this still uses number of cycles as same as the global memory.
memory. The CUDA memory model can be shown as in Figure 1.

Figure 1. CUDA memory model [6][7]

For programming on the CUDA platform, a developer has to specify the number of threads for computation. Threads that will be executed in a kernel must be managed as groups of threads with shared data, called thread blocks. A group of blocks form a grid. Creating, organizing, and destroying threads on the GPU consume only a few resources. This allows the developers to manage hundreds of threads very fast and effectively.

III. PROPOSED STRATEGY

The aim of our strategy is to make the denoising process can be done in real-time by utilizing all the computation resource both CPUs and GPUs efficiently.

On the CPU side, we create 4 types of threads as the following:

1. Fetching Thread is used for fetching and pre-fetching the frame and converting the image into 8-bit grayscale image, creating the label for each frame and storing in the main memory and labeling.

2. Compute Thread is used for computing gradient operation \(K(u)\) and transferring the computed frame to GPU’s memory.

3. Merging Thread is used for transferring denoised frame data back from GPUs and sorting the denoised frame and discarding expired frame.

4. Display Thread is an optional thread to display the denoised video in real-time.

Our strategy is illustrated as Figure 2.

Figure 2. Our propose strategy for denoising video in real-time.

Here the explanation of the denoising process step-by-step as following steps:

1. Fetching Thread fetches and uses the prefencing technique by fetching the frames ahead, stores them into the main memory and labeling them. Each frame’s label contains (i) frame sequence, (ii) frame expiration time.

2. Compute Thread keeps fetched frames from the main memory, computes \(K(u)\) and transfers them to the GPU’s memory, then invokes the kernel after the data transfer is finished.

3. Next, the GPU denoises the frame in Equation (7) by using the Fixed-point iterative method and Sliding Windows Gauss-Seidel [14].

4. After the GPU finishes denoising, the Merging Thread transfers the denoised frame back into main memory, reads the frame label and then rearranges the denoised frame in the buffer in the main memory and discards the expired frames.

5. Display Thread displays the denoised output in real-time.

IV. EXPERIMENT RESULTS

In this section, we divide into 3 parts. First, we validate our strategy and the noise removal algorithm. Next, we show the performance gain and the frame rate by using our strategy and finally, the results of some denoised frames are displayed.

The experiments were made on Intel® Core 2 Duo with 2.5 GHz of CPU and 4GB of main memory. The NVIDIA®
Quadro NVS 510M with 48 CUDA cores and 512MB of the graphic memory. Each core runs at 500MHz.

We use 64-bit Fedora 17 Linux with GNU C Compiler (GCC) 4.7 with GNU debugger (gdb) enabled, OpenCV 2.3 for image and video manipulation.

A. Denoising Model Validation

To ensure that the noise removal algorithm and our strategy can efficiently remove both additive and multiplicative noises, we use the sample video and synthesized both additive and multiplicative noises into it. The first frame of sample video is shown as Figure 3.

![Figure 3. A original sample picture for testing denoising algorithm (a) and sample picture with synthesized noises (b).](Image)

The synthesized noisy frame has Peak Signal-to-Noise Ratio (PSNR) value of 19.77dB.

After denoising the video with well-selected parameters, the first frame is denoised as shown in Figure 4.

![Figure 4. Denoised sample picture.](Image)

The PSNR value of image with the synthesized noise comparing with the original image is 85.64. This shows the denoising model and our strategy can remove both multiplicative noises efficiently.

In denoising the streaming video, the frame order is also checked to make sure that our strategy is working correctly.

B. Performance Results

First, we measure the average frame rate for the denoised video as Figure 5, while varying the number of GPU threads per frame.

![Figure 5. Time used per frame varying threads per frame.](Image)

We define the average speedup as:

$$s_{avg} = \frac{t_{seq}}{t_{avg}}$$  \hspace{1cm} (10)

Where $s_{avg}$ is the average speedup for each frame, $t_{avg}$ is the average time for each frame including the discarded frames. The $t_{seq}$ is the time used for denoising a frame sequentially. The average speedup varying the number of GPU threads per frame is shown as Figure 6.

![Figure 6. Speedup of propose strategy varying thread per frame.](Image)

We fix the frame rate of the output video at 15 frames per second. We measure the frame rate of the output varying number of threads per frame without discarded frames as in Figure 7. Please note that only the entire video average frame rates are shown in Figure 7.
Figure 7 shows that after we utilize both multi-core CPUs and GPUs asynchronously at the same time. The sequential data shows the single thread of CPU without any enhanced technique. The number of frames per second is boosted dramatically by launching multiple kernels. The increasing of the thread per frame results in increasing frame rate by allowing more GPU cores to work on the same frame in parallel using Sliding Window Gauss-Seidel.

The frame per second is satisfiable (~14-15 fps) when thread per frame ≥ 8 as we described in the first section that output video should has frame rate around 12-15 fps. The average frame drop is 1.1 frame per second on 32 threads per frame while the sequential computation has upto 14.8 frames drop per second. When the number of threads increase to 16-32 thread per frame, it will result in the almost same frame rate because we locked the output frame rate at 15fps. In the future, we will test our strategy with more powerful GPUs and lock the output frame rate at 24-30 fps.

C. Denoised Video Quality

We use sample ultrasound video from public domain video achieve [15]. The video resolution is 480×352 pixel and the frame rate is 29 frames per second. Each frame is converted into the 8-bit grayscale image in the denoising process. The 80th and 300th sample frames are shown as Figure 8.

The denoised 80th and 300th frames are shown as Figure 9.

The PSNR of denoised frame comparing with the noisy video frame is in the range of 78.97 dB – 86.43dB.

V. CONCLUSION AND FUTURE WORKS

We propose the new strategy for denoising the ultrasound streaming video which contains both additive and multiplicative noises. The GPUs and multi-core processor are used to accelerate the computation to ensure satisfy frame rate.

Our strategy uses the denoising model and improves video denoising strategy from [1]. The proposed strategy uses both multi-core advantages and reduces some overhead from the frame distribution while utilizing the GPUs efficiently.

Our results show that our strategy can achieve speedup per single frame computation up to 3.79 times comparing to the sequential computation. The output video frame rate is boosted 98.86 times comparing with the sequential computation. Moreover, the denoising video quality is visually satisfiable. This makes the real-time video denoising possible. However, fine denoising parameter tuning is still essential in practical to make sure that the denoised picture is smooth and retains all necessary information in the frame.

However, the experiments are tested on the small graphic card with only 48 CUDA cores. This strategy should be tested with the graphic card with the massive number of CUDA cores in the future.

REFERENCES


