GPU-ACCELERATED NON-LOCAL MEANS SUPER-RESOLUTION RECONSTRUCTION

Kongzheng Sun，Jiade Li and Shuyan Xu

Abstract Super-resolution is a process via fusing low-resolution image frames to obtain high-resolution images that typically requires highly accurate image registration. The modern non-local means super-resolution algorithm has successfully evaded the problem of image registration. However, its great computational impact has encouraged us to seek an approach that is able to make it more feasible for practical use. Thus in this paper, we propose an optimized highly efficient GPU-accelerated non-local means super-resolution algorithm. The proposed algorithm sufficiently utilizes the on-chip shared memory of GPU that ensures a significant performance boost. Experimental results show that the proposed algorithm not only overwhelmingly outperforms its CPU counterparts, but also achieves approximately four times performance increase than a typically conventional straightforward GPU implementation.

Keywords GPU，super-resolution，nonlocal means

1 Introduction

Super-resolution(SR) reconstruction is the process of combining several low-resolution image frames (LRs) to generate high-resolution images(HRs). After it was first proposed by Tsai and Huang [1] decades ago, Elad, et al. [2] compared three of the typical theories – the maximum likelihood estimator(ML), the maximum a posteriori probability(MAP) estimator and the projection onto convex sets approach(POCS) – that since represent the cornerstone of the classical super-resolution solution within Bayesian framework over the last decade. We refer to [3] for the representative literature of massive study on super-resolution.

Kongzheng Sun (✉)
Changchun Institute of Optics, Fine Mechanics and Physics Chinese Academy of Sciences, Changchun, China
e-mail：skz3148@163.com

Jiade Li
Changchun Institute of Optics, Fine Mechanics and Physics Chinese Academy of Sciences, Changchun, China

Shuyan Xu
Changchun Institute of Optics, Fine Mechanics and Physics Chinese Academy of Sciences, Changchun, China

This work is supported by the National Nature Science Youth Foundation of China (No. 60902067).

Since super-resolution is deemed as an inverse problem, the traditional Bayesian framework usually requires image-registration, interpolation and deblurring. That is, the first step is to estimate the relative motion in the image sequence, followed by aligning the pixels on the HR grid and post-processing. While the classical super-resolution algorithms have gained widespread application in a variety of image processing fields, it is worth mentioning that only by making image registration accurate enough can the result of super-resolution reconstruction be good enough. In practical cases, one can hardly make such assumption that motion estimation is an easy task. The causes are twofold – first, one cannot expect the estimation parameter to be exact multiples of HR pixel coordinates due to the limitation of the camera's resolution; more importantly, the classical assumption of global motion will not stand as local motion exists in most actual scenes in nature. Park, et al. [4] proposed an algorithm that considers the inaccuracy of the motion information. It first analyses the motion error as modeled noise, which indicates the square of the misregistration error is proportional to the variance of the noise caused, and inversely proportional to the high-frequency energy of the image. However, it is a complicated task and the result might also be unstable. Encouraged by the recent development in denoising algorithms, Protter, et al. [5] proposed a non-local means (NLM) algorithm to super-resolution. NLM has been proved to be an effective algorithm to deal with sequences with local motion and most excitingly, the fuzzy nature of it does not require explicit motion estimation, exonerating the task from "manually" registering images. Nevertheless, the seemingly gorgeous features of NLM algorithm can barely overshadow its major drawback – time efficiency. Since the extensive neighborhood search scheme remains inevitable, NLM algorithm is virtually impractical in many demanding situations which, for example, require real-time processing of image sequences. Therefore, we have resort to parallel devices, most notably the graphics processing units (GPUs), to boost its overall performance.

GPUs are originally designed for graphics rendering that is computational intensive. They have outperformed traditional CPUs in many ways that include highly parallel architecture, multithreading, manycore and very high memory bandwidth. The discrepancies between the two lie under the fact that more transistors in GPUs are devoted to computing task rather than flow control and caching that are typical for CPUs. In recent years, many researchers have witnessed the development of GPUs and started to harness the horsepower. Zheng, et al. [6], for example, proposed algorithms for neighborhood denoising based on GPU. In light of this, we propose a GPU based parallel model for the time-consuming NLM super-resolution algorithm in this paper, which is structured as follows. Section 2 formulates the NLM super-resolution problem. Section 3 describes the algorithm we propose. Section 4 reveals and analyses the experimental results we have conducted over several popular parallel algorithms, and finally we conclude this paper.

2. NLM SUPER-RESOLUTION

Multiframe super-resolution problem with the *k*-th known LR frame y_k is usually formulated by minimizing an energy function which can be described as:

$$\hat{\mathbf{X}} = \arg\min_{\mathbf{X}} \left[\sum_{k=1}^{p} \rho(\mathbf{y}_k, D_k H_k F_k \mathbf{X}) \right]. \qquad (1)$$

Where ρ is certain kind of norm in which F_k is a matrix for the motion parameters of the *k*-th frame; H_k represents the blurring process to the *k*-th frame followed by the degradation matrix D_k. $\hat{X}$ is the super-resolution frame to be estimated from LR frames.

The idea of NLM super-resolution is originated from the prevailing NLM filter used in image denoising. It is based on the assumption that the repetitiveness of pixel patterns accounts for one of the characteristics of the image frames. Likewise, the energy function for NLM super-resolution is structured into the following form:

$$\hat{\mathbf{X}} = \arg\min_{\mathbf{X}} \left[\sum_{(k,l)\in\Omega} \sum_{t=1}^{p} \sum_{(i,j)^H \in N_{k,l}} \omega(k,l,i,j,t) \left\| \mathbf{D}_{k,l} \mathbf{E}_{k,l}^{H} \mathbf{H}\mathbf{X} - \mathbf{E}_{i,j}^{L} \mathbf{y}_t \right\|_2^2 \right]. \qquad (2)$$

where $\mathbf{E}_{k,l}^{H}$ and $\mathbf{E}_{i,j}^{L}$ denote the patch extractors around pixel (*k*,*l*) for high- and low-resolution images respectively. It should be noted that the reason $\mathbf{D}_{k,l}$ exists is that the high-resolution patch must be degraded to match the low resolution patch. They can be illustrated as in Fig. 1. The weight ω reflects the similarity between these two patches and is calculated in a similar fashion as in the NLM denoising filter [7],

$$\omega(k,l,i,j,t) = \exp\left(-\frac{\left\| \mathbf{P}_{k,l}\mathbf{y}_t - \mathbf{P}_{i,j}\mathbf{y}_t \right\|^2}{2\sigma^2} \right) \cdot f(k,l,i,j). \qquad (3)$$

with the only difference being the introduction of scale *t* – the index of *t*-th frame.

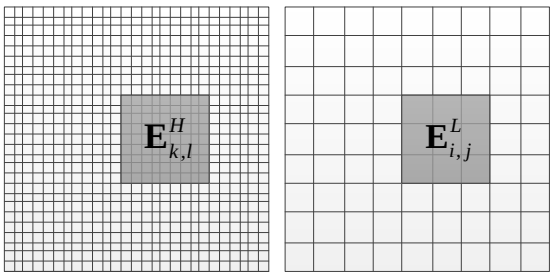

Fig.1. Description of the high-resolution patch and the low-resolution patch

Following the routine of solving minimization problems, eq. (2) can be solved by taking derivation and then equating to 0.The final results can be derived after several mathematical manipulations:

$$\hat{\mathbf{X}} = \left[\sum_{(k,l)\in\Omega} \sum_{t=1}^{p} \sum_{(i,j)^H \in N_{k,l}} \omega(k,l,i,j,t) \cdot \left(\mathbf{E}_{k,l}^H\right)^T \mathbf{D}_{k,l}^T \mathbf{D}_{k,l} \mathbf{E}_{k,l}^H \cdot \mathbf{1} \right]^{-1}$$

$$\cdot \left[\sum_{(k,l)\in\Omega} \sum_{t=1}^{p} \sum_{(i,j)^H \in N_{k,l}} \omega(k,l,i,j,t,\sigma) \cdot \left(\mathbf{E}_{k,l}^H\right)^T \mathbf{D}_{k,l}^T \mathbf{E}_{i,j}^L \mathbf{y}_t \right]. \qquad (4)$$

For a much simpler case where the low-resolution extractor only extracts one pixel, a further simplified form of (4) will be:

$$\hat{\mathbf{X}}(k,l) = \frac{\sum_{t=1}^{p} \sum_{(i,j)^H \in N_{k,l}} \omega(k,l,i,j,t) \mathbf{y}_t(i,j)}{\sum_{t=1}^{p} \sum_{(i,j)^H \in N_{k,l}} \omega(k,l,i,j,t)}. \qquad (5)$$

According to (5), we must further address two problems:1 – the different scale of low- and high-resolution patch; 2 –the use of unknown pixel (k,l) and its neighbors. The first problem has already been discussed in (2), as illustrated in Fig. 1. The second one can be solved by using an initial estimation of X being an interpolated version of one LR frame. This technique may lead to a consequence of iteration. However, in our experiments, we found the result good enough even with a single iteration.

3. GPU-ACCELERATEDNLM SUPER-RESOLUTION

3.1 Basics of GPU Programming

```
function SIMPLEKERNEL(dst, imageH, imageW)
    * Retrieve the thread index (ix, iy, t)
    if ix < imageW and iy < imageH then
        * Initialization
        for each pixel y_t(i, j) in NLM search window do
            * Initialize weight = 0
            * Calculate weight within the NLM block
            * Accumulate weight · y_t(i, j) to V
            * Accumulate weight to W
        end for
        * result = V/W
        * Write to destination frame dst ← result
    end if
end function
```

Fig. 2. A Simple Kernel

Modern GPUs are structured under an array of multithreaded Streaming Processors (SMs). The multiprocessors are built upon an architecture called Single-Instruction, Multiple-thread (SIMT) to manage a group of threads running on them. Therefore, each multiprocessor can execute multiple threads concurrently.

In compliance with this feature, there is a unique programming model named CUDA that is responsible for dispatching threads to multiprocessors. That is, each thread to be executed by GPU is called a kernel and threads are logically arranged by a Grid-Block model, under which each thread is assigned an unique 3-component vector comprised of one-, two- or three-dimensional index. Multiple threads form a block and multiple blocks form a grid. Blocks and grids are also identified by 3-component vectors, exact the same way as are the CUDA threads.

3.2. A Simple Approach

Since each thread is run by CUDA as a kernel and the NLM super-resolution algorithm keeps local in nature, an intuitive way of parallelization is to relate each of the threads to a single pixel and its neighbors on each image frame.

The 3-dimensional thread architecture perfectly satisfies the needs to correlate each thread to the pixel. For each pixel $y_t(i,j)$ located on the t-th frame of the LR sequence in(5), a GPU thread indexed (ix,iy,t) is assigned to compute weight $\omega(k, l, i, j, t)$ within an NLM block (which is searched for computing the weight and, of course, is a different matter from CUDA thread block). The weight is then accumulated to $y_t(i,j)$ and normalized so as to derive an estimation of $\hat{\mathbf{x}}$. This process can be illustrated as Fig. 2.

3.3. The Proposed Approach

Although the above algorithm (Fig. 2) sheds some light on how the NLM super-resolution should be implemented on CUDA, there are still some facts we cannot neglect.

CUDA thread may access three kinds of GPU memory. Each thread has its own private local memory. The threads bound in a thread block share the block-specific shared memory. All the threads in the GPU can also access the global memory directly. For all the three, shared memory is quite distinct from the other two. Since it is built on-chip, it can be accessed much faster than global memory and thus can be used as cache.

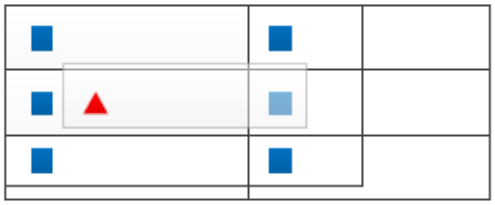

Fig. 3. Proposed scheme of mapping pixels to the thread

As for the simple and straightforward algorithm (Fig. 2), the image data are first loaded into global memory, then all the operations are done within global memory. However, we can further optimize the performance of the algorithm if we can sufficiently utilize the per-block shared memory. To achieve this, we need to cache a related region of the image data into shared memory before any further operations can take place. The size of the cached image region could be, for example, $(X_b+2R_w+2R_b) \times (Y_b+2R_w+2R_b) \times N$ which is determined by the CUDA block size (X_b, Y_b), NLM window radius R_w, NLM block radius R_b as well as the number of image frames N.

This cached size of shared memory should generally be set greater than the CUDA block size in order to maximize the performance. Therefore, it becomes a little tricky that each thread in a CUDA block should be responsible for loading multiple pixels into the shared memory. The scheme we propose to achieve this goal is to create a mapping between the thread and these pixels. As illustrated in Fig. 3, the area of the shared memory is divided up into several regions whose size is made equal to the size of a CUDA block except only for the regions on the boundary of the shared memory. For instance, the thread depicted by the red triangle will be obliged to load six pixels depicted by the blue squares if those pixels are within the boundary of the shared memory. This process continues until all the data required are loaded into the shared memory.

In general, the algorithm using shared memory as cache is outlined in Fig. 4. The main advantage of the proposed algorithm over the simple algorithm is that while the former has to synchronize threads twice, most of the calculations are done with respect to shared memory. Once loading the corresponding image region into shared memory, the global memory is freed from being read for the whole execution period in a CUDA block. This dramatically reduces the latency to access global memory and increases the memory throughput which ultimately leads to an overall performance boost.

4. EXPERIMENTAL RESULTS

```
function OPTIMIZEDKERNEL(dst, imageH, imageW)
    * Allocate shared memory for caching
    * Retrieve the thread index (ix, iy, t)
    if ix < imageW and iy < imageH then
        * Load image region into shared memory
        * Transform pixel coordinates
        * Allocate shared memory for V and W
        * Synchronize threads
        for each pixel y_t(i, j) in NLM search window do
            * Initialize weight = 0
            * Calculate weight within the NLM block
            * Accumulate weight · y_t(i, j) to V
            * Accumulate weight to W
        end for
        * Synchronize threads
        * result = V/W
        * Write to destination frame dst ← result
    end if
end function
```

Fig. 4. The Optimized Kernel

In this section, we will compare the performance of several distinct NLM super-resolution algorithm implementations and present the results. We conducted our

experiments on a machine with a GPU of NVIDIA GeForce GTX 460 (Fermi, 384 CUDA cores with compute capability 2.1), an Intel Xeon quad-core CPU of maximum clock 3.6GHz and an 8GB memory. All the programs were coded under Visual Studio 2010 with CUDA Toolkit 4.2 in 32-bit mode.

The source we used to perform our tests is an image sequence of LR Lena consisting of four frames. All the frames are of size 128×128, which are randomly shifted, rotated and then degraded from an HR image of 256×256. In order for the best of performance comparison, we set the parameters related to image quality identical. The NLM block for computing weight is set as a square of size 13×13, and the NLM search area 5×5. The parameter σ in (3) has been chosen to be a relatively large one – 13, due to the noises added to the images during the degradation process. As mentioned earlier, we only apply single iteration to each experiment since the results, shown in Fig. 5, are acceptable already.

For the sake of comparison, we first execute the NLM super-resolution algorithm (just as proposed by Protter, et al. [5]) on CPU with single thread, followed by running a revised multithreaded version which utilizes the power of quad-core processor by scheduling the computation task to four concurrent threads. Subsequently, we ran the straightforward simple NLM super-resolution algorithm on GPU, and finally the optimized approach proposed in this paper. For our Fermi GPU used in this experiment, we have tested out that the best performance is attained when setting the CUDA block size to 8×8×4. The experimental results for single threaded CPU, Multithreaded (MT) CPU, simple GPU and proposed GPU algorithms are shown in Table 1. It is obvious that the proposed GPU NLM su-per-resolution algorithm in this paper achieves more than 3.8 times performance gain than the simple algorithm and that its CPU counterparts are overwhelmingly inefficient.

5. CONCLUSION

(a) One of the LR frames (b) Super-resolved result

Fig. 5. Simulation results

Table 1. Performance Comparison

	CPU	CPU MT	GPU Simple	GPU Proposed
Time (ms)	1024.9	328.3	30.9	8.1

This paper introduces an optimized GPU-accelerated nonlocal means super-resolution algorithm that can significantly increase the performance of conventional NLM super-reso-lution algorithm. Compared to a simple and straightforward GPU implementation, the proposed algorithm is proper tuned to use the resources of CUDA shared memory so as to

maximize efficiency. The experimental results showed that the proposed algorithm is able to speed up to four times faster than the simple GPU algorithm.

References

[1] R. Y. Tsai and T. S. Huang, “Multiframe image restoration and registration,” Advances in Computer Vision and Image Processing, vol. 1, pp. 317–339, 1984.

[2] M. Elad and A. Feuer, “Restoration of a single supersolution image from several blurred, noisy, and undersampled measured images,” Image Processing, IEEE Transactions on, vol. 6, no. 12, pp. 1646–1658, 1997.

[3] Deepu Rajan and Subhasis Chaudhuri, “An mrf-based approach to generation of superresolution images from blurred observations,” 2002, 10.1023/A: 1013961817285.

[4] Min Kyu Park, Moon Gi Kang, and Aggelos K. Katsaggelos, “Regularized high-resolution image reconstruction considering inaccurate motion information,” Optical Engineering, vol. 46, no. 11, pp. 117004, 2007.

[5] M. Protter, M. Elad, H. Takeda, and P. Milanfar, “Generalizing the nonlocal-means to super-resolution reconstruction,” Image Processing, IEEE Transactions on, vol. 18, no. 1, pp. 36–51, 2009.

[6] Ziyi Zheng, Wei Xu, and Klaus Mueller, “Performance tuning for cuda-accelerated neighborhood denoising filters,” in Workshop on High Performance Image Reconstruction (HPIR), Potsdam, Germany, 2011, Workshop on High Performance Image Reconstruction (HPIR).

[7] A. Buades, B. Coll, and J. M. Morel, “A non-local algorithm for image denoising,” in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.