Parallel Implementation of Compressive Sensing Based SAR Imaging with GPU

1Tian Jihua, 2Sun Jinping, 3Zhang Yuxi, 4Najeeb Ahmad, 5Zhang Bingchen
1School of Electronic and Information Engineering, Beihang University
tianjihua203@gmail.com
2,3,4School of Electronic and Information Engineering, Beihang University
5Nat. Key Lab of MW Imaging Tech. Institute of Electronics, CAS

Abstract

The paper proposed a new scheme for parallel implementation of compressive sensing based SAR imaging on GPU with Iterative Shrinkage/Thresholding algorithm. To get a faster recovery speed, we modified the existed IST algorithm structure, and realized the fast implementation on GPU. The experiment result shows that parallel computing capabilities of GPU have a significant speedup in comparison with computing capability of CPU.

Keywords: Compressive Sensing, Synthetic Aperture Radar, Graphics Processing Unit, CUDA

1. Introduction

As a major remote sensing sensor, Synthetic aperture radar (SAR) can produce high resolution images from a moving platform, such as an airplane or a satellite. A SAR system produces 2D (range and azimuth) terrain reflectivity images by emitting a sequence of closely spaced radio frequency pulses and by sampling the echoes scattered from the ground targets [1]. The main advantage of SAR is that images of the illuminated area can be obtained independent of time-of-day or weather conditions (e.g., fog, cloud level, rain, and snow). Modern airborne and spaceborne SAR systems can produce very high resolution images and are being widely used in many civilian and military applications [1, 2].

Compressive sensing (CS), proposed by Donoho [3], Emmanuel Cand`es [4] and Micheal Elad [5] et al. is a new developing novel theory that enables perfect recovery of signals and data from what appear to be highly sub-Nyquist-rate samples. CS proclaims that an unknown sparse (or sparse under certain basis) signal can be exactly recovered with high probability from very limited number of measurements by solving a convex l₁ optimization problem. Based on rigid mathematics, CS has attracted many attentions in image processing, data fusion of multiple sensors, radar applications and so on. Up to now, a few literatures have addressed adopting CS in some radar applications including SAR and Inverse Synthetic Aperture Radar (ISAR) [6-9].

However, the reconstruction of sparse signal requires numerous matrix-vector multiplications, which imposes a heavy burden on the numerical computation, especially when the sensing matrix is a large dense one. Meanwhile, the computation of compressive sensing based SAR imaging technique becomes larger and larger along with the increasing demand on high resolution SAR images. As a result, it takes quite a long time to reconstruct a SAR image on CPU, which can not be implemented in real-time. Recently, the fast processing performance of graphics processing unit (GPU) offers an alternative for fast reconstruction of sparse signal. This paper realized the fast reconstruction of compressive sensing based SAR images, taking advantage of the efficient parallel computing capabilities of GPU.

2. GPU Architecture and Software Framework

Recently, driven by the insatiable market demand for real-time, high-definition 3D graphics, the programmable GPU has evolved into a highly parallel, multithreaded many core processor with tremendous computational horsepower and very high memory bandwidth [10].

NVIDIA released computed unified device architecture (CUDA) in November 2006, the first developed environment and software framework for GPU, which is an extension to the standard C
language that allows users to manage the GPU as computational device without the help of graphic API.

In CUDA architecture, tasks are split into a grid of multiple thread blocks each of which consists of series of threads. The thread blocks are arranged into different stream multiprocessors of GPU. CUDA adopts the single instruction multiple thread (SIMT) model, which means all the threads in one block share the same instruction code with different data, and possibly run at different states. In CUDA, every thread has its own dedicated registers, and communication between blocks is realized through shared memory and synchronous mechanism. This design helps minimize the costs of context switching on GPU.

The design aim of GPU is to realize the parallel computation through numerous threads, fixing it suitable for large scale parallel computing tasks that are intensive in computation and simple in logic.

3. Theory of Compressive Sensing

The theory of CS reveals that exact recovery of an unknown sparse signal is possible from very limited samples by solving an inverse problem through either a linear program or a greedy pursuit. Suppose that signal $s \in \mathbb{R}^N$ is K-sparse on an overcomplete dictionary $\Psi$, i.e.

$$s = \Psi x$$  \hspace{1cm} (1)

where $\Psi = \{ \psi_1, \psi_2, \ldots, \psi_N \}$ is an $N \times N$ matrix constructed by a sparse basis $\{ \psi_j \}$, and $x$ is a vector with all except $K$ of its entries are zeros. Various expansions, including wavelets, the DCT, and Gabor frames, are widely used for the representation and compression of natural signals, images, and other data. The matrix $\Psi$ is constructed according to the selected expansion. In order to reconstruct signal $s$, a set of $M$ measurements is acquired $(M < N)$, which are linear combinations of the points within $s$.

More precisely

$$y = \Phi s = \Phi \Psi x$$  \hspace{1cm} (2)

where $\Phi$ is a $M \times N$ matrix, hereinafter called measurement matrix. Since $M < N$, recovery of the signal $s$ from the measurements $y$ is ill-posed in general. The CS theory reveals that when the matrix $A = \Phi \Psi$ has the Restricted Isometry Property (RIP) [3-5], the $x$ or the signal $s$ can be recovered from a similarly sized set of $M = O(K \log(N/K))$ measurements $y$ with high probability. The RIP closely relates to an incoherency property between $\Phi$ and $\Psi$, which means the rows of $\Phi$ can not provide a sparse representation of the columns of $\Psi$ and vice versa. The RIP and incoherency holds for many pairs of basis, such as delta spikes and Fourier sinusoids, or sinusoids and wavelets. It can be proved that (pseudo) random noise-like matrix performs excellently as $\Phi$, such as the matrix constructed by Bernoulli or Gaussian random variables. Another choice for the measurement matrix $\Phi$ that offers good performance in many cases is a causal, quasi-Toeplitz matrix [3-6]. When the RIP holds, the $x$ or the signal $s$ can be recovered from the solution of a convex optimization problem. Formally, with high probability, $x$ is the unique solution to

$$\min \| x \|_1 \text{ s.t. } y = \Phi \Psi x$$  \hspace{1cm} (3)

which can be solved efficiently with linear programming techniques. Current reconstruction methods include iterative greedy algorithms such as Matching Pursuit (MP), Orthogonal Matching Pursuit (OMP) and convex relaxation algorithms such as Basis Pursuit (BP) and Iterative Shrinkage/Threshold (IST) and so on.

4. Iterative Shrinkage/Thresholding Algorithm and GPU Realization

4.1 Iterative Shrinkage/Thresholding Algorithm
In practice, the measurements are always disturbed by noise or other interference. So it is no more suitable to enforce $y = \Phi \Psi x$ according to the constraint in Equation 3. Generally we solve the problem by transforming the constraint convex optimization problem into the following unconstraint convex optimization problem

$$\min_x \frac{1}{2} \|y - Ax\|^2 + \tau \|x\|$$  \hspace{1cm} (4)

where $\|\cdot\|$ denotes the Euclidean norm and $\tau$ is the regularization parameter which provides a tradeoff between fidelity to the measurements and the noise sensitivity. Iterative Shrinkage/Thresholding (IST) [11, 12] is a state-of-the-art algorithm in solving the unconstraint convex optimization problem, with the following iterative scheme

$$x_{k+1} = \text{soft}(x_k + A^H (y - Ax_k), \tau)$$  \hspace{1cm} (5)

where $\text{soft}(x, \tau) = \text{sgn}(x) \max(|x| - \tau, 0)$ is the shrinkage operator.

IST algorithm has already been applied extensively to handle the unconstraint convex optimization problem arising in recovery of sparse signal, image restoration and other linear inverse problems. Each iteration step of IST algorithm only requires matrix-vector multiplications and addition computation, which is suited to utilize the efficient parallel computing capabilities of GPU to realize fast recovery. So this paper realized the fast reconstruction of compressive sensing based SAR images using IST algorithm on GPU. The detailed procedure of IST algorithm is described as follows

1. Initialize $x_0 = 0$, residual vector $r_0 = y$, and set iteration step $k = 1$;
2. Compute the correlation of $A$ with the current residual, and the next estimate according to the current one

$$x_{k+1} = x_k + A^H r_k$$  \hspace{1cm} (6)

3. Process the new estimate with shrinkage operator

$$x_{k+1} = \text{soft}(x_{k+1}, \tau)$$  \hspace{1cm} (7)

4. Update the residual vector

$$r_{k+1} = y - Ax_{k+1}$$  \hspace{1cm} (8)

5. Compute the objective function value $f_{k+1} = 0.5 \|r_{k+1}\|^2 + \tau \|x_{k+1}\|$, and get the relative change $\Delta f = \left|\frac{f_{k+1} - f_k}{f_k}\right|$. If $\Delta f$ is smaller than the stopping threshold then terminate the iteration and output the estimate $\hat{x}$. Otherwise, go to step 2 for the next iteration.

In IST algorithm, the most computation prohibitive portion is the matrix-vector multiplication involving $A$ and $A^H$, with computation complexity of $O(MN)$. Besides, each iteration step requires two such multiplications, so the computation of the whole recovery process is very large, especially when the matrix is a large dense one. However, if we convert Equation 5 to Equation 9, we can find that, $A^H y$ participates in the computation as a constant vector, and the two multiplications reduce to one multiplication only involving $A^H A$ in each iteration. Although the computation complexity of $A^H Ax_k$ is $O(N^2)$, larger than the matrix-vector multiplication involving $A$ and $A^H$ on CPU, it really can reduce the whole computation when implemented in parallel.

$$x_{k+1} = \text{soft}(x_k + A^H y - A^H Ax_k, \tau)$$  \hspace{1cm} (9)

So we proposed a new scheme based on the above analysis, precompute the $A^H A$ and $A^H y$ before the iteration, then only one matrix-vector multiplication is required in each iteration step. In this way, it not only reduces the cost of matrix-vector multiplication, but also reduces the time cost by data transmission between two multiplications. In addition, we noticed that the residual vector should be available when calculate the objective function value. However, the calculation of residual vector involves matrix $A$, which is against the proposed method requiring only $A^H A$ and $A^H y$. So we need
some changes to the calculation of residual vector to meet with the proposal. Fortunately, we find that if we replace the computation formula of objective function value with Equation 10, the two different methods show the same effect when judging whether the termination criterion is satisfied based on the relative change of two contiguous objective function values, although \( f \) is different from \( f \). In addition, as we know, the SAR images are not sparse over all range gate, so we need add some constraint to escape from recovering the unsparse ones. If the objective function value in one step is no less than the one got in the former step, then we can say the scene is not sparse and exit the recovery.

\[
\tilde{f} = 0.5 \| A^H y - A^H A x \|_2^2 + \tau \| x \|_1
\]  

(10)

The basic procedure of the modified version of IST algorithm is

1. Initialize \( x_0 = 0 \), compute \( A^H y \) and \( A^H A \), and set iteration step \( k = 1 \);
2. Compute the correlation of \( A \) with the current residual 
\[
x_{\text{temp}} = A^H y - A^H A x_k
\]  

(11)
3. Compute the new estimate 
\[
x_{k+1} = \text{soft}(x_k + x_{\text{temp}}, \tau)
\]  

(12)
4. Compute the objective function value 
\[
\tilde{f}_{k+1} = 0.5 \| x_{\text{temp}} \|_1 + \tau \| x_{k+1} \|_1,
\]  

if \( \tilde{f}_{k+1} > \tilde{f}_k, k > 2 \), then terminate the iteration. Otherwise, get the relative change \( \Delta \tilde{f} = \tilde{f}_{k+1} - \tilde{f}_k / \tilde{f}_k \). If \( \Delta \tilde{f} \) is smaller than the stopping threshold then terminate the iteration and output the estimate \( \hat{x} \). Otherwise, go to step 2 for the next iteration.

4.2 GPU implementation of IST

As we know, in CUDA framework, communication between host CPU and GPU device often costs lots of time, so we should use such communication as few as possible[13,14]. In this paper, data communication between host CPU and GPU device only occurs at the start and end of implementation of the algorithm. At the start phase, the precomputed \( A^H y \) and \( A^H A \), regularization parameter \( \tau \) and other necessary parameters are transmitted to GPU, while the reconstructed results are transmitted back to CPU at the end of the recovery. Where \( A^H A \) is stored in the global memory, and \( A^H y \) is stored in the constant memory as it will not be changed during the recovery. In order to save the communication time further, we transmit back the nonzero elements and the corresponding indices instead of the whole elements of the reconstructed signal [15]. Besides, when numerous data need to be reconstructed, a series of streams can be built, each of which is responsible for transmission and execution of different data. Processing with two streams allows for the memory copies of one stream to overlap with the kernel execution of the other stream. Then the time cost by memory copies between CPU and GPU can be efficiently hidden, and get the performance improved.

GPU device begins to execute the iterative recovery once it receives the data from CPU. As mentioned above, the dominant computation during the recovery is the matrix-vector multiplication. Note that in the matrix-vector multiplication, column vectors of the matrix are mutually independent, which is fit to be implemented in parallel. The matrix-vector multiplication is realized with coarse-grained parallelism blocks that can not communication with each other, together with the fine-grained parallelism threads. In detail, multiplications between column vectors of \( A^H A \) with \( x \) are realized in coarse-grained parallelism, while elements and elements products inside the vector multiplication are realized in fine-grained parallelism. We stored the matrix \( A^H A \) in global memory, so we have to access the global memory to fetch it when it is needed. To limit the memory latency, we utilized the shared memory that is as fast as register. For instance, column vectors of \( A^H A \) and \( x \) are all stored in the shared memory that lies in each thread block.

During the IST recovery, we have to transform some multidimensional data to one dimension, such as the calculation of Euclidean norm and \( l_1 \) norm. Take the calculation of \( l_1 \) norm for instance, normally
we add all the elements step by step. But on GPU, there are only 512 threads in each block which is smaller than dimension of the vector, so we split such task into parallel accumulation involving multiple thread blocks, where each block is responsible for addition of part data and the partial sum got by each thread are summed up at last. To make the most efficient use of the compute power of GPU, we should utilize enough thread blocks, keeping the maximum active thread blocks per multiprocessor. However, communication between thread blocks works only through global memory that is limited in GPU and has long access latency, which means it will cost lots of time in memory access if too many blocks were used. So we should make a tradeoff to select the suitable thread blocks.

During the vector multiplication realization in fine-grained parallelism and computation of multidimensional data to one dimension, each thread block will complete summation of many data. This paper adopts the parallel summation reduction method to make most efficient of the parallel performance of GPU. Figure 1 shows the procedure of parallel summation reduction with 8 elements. The traditional serial summation method requires $n$ steps to sum up $n$ elements, while the parallel summation reduction method only requires $\log_2 n$ steps. Meanwhile, the parallel summation reduction works with sequential addressing which is bank conflict free, avoiding the reduction in efficient access bandwidth. In addition, the threads in each warp will either execute the summation or not, which will avoid the performance degradation caused by divergence.

Figure 1. Parallel summation reduction with 8 elements

5. Experiment

To validate the speeding up of parallel realization of compressive sensing based SAR imaging on GPU, we separately reconstructed the same SAR image using IST algorithm on CPU and GPU. The configuration of the CPU used in this paper is Intel Core2 Quad 8400, 2.66GHz, and the GPU is Tesla C1060. And the data used in the experiment are real airborne SAR data which have been collected by an X-band SAR with the resolution of 2m. For the detailed compressive sensing based SAR imaging technique, please refer to literature [16]. We implemented the CPU and GPU code in single precision float and computed the average processing time over 100 repeated executions on CPU and GPU separately. Figure 2.a shows the conventional SAR imaging result with full samples, while Figure 2.b shows the compressive sensing base SAR image reconstructed from 50% samples implemented on GPU. The time cost by CPU and GPU are shown in Table 1. From Table 1, we can see that GPU speeds up 35 times than CPU.
Figure 2: (a) Conventional SAR imaging result with full samples. (b) Compressive sensing based SAR imaging result with 50% samples implemented on GPU.

Table 1: The average execution times on CPU and GPU

<table>
<thead>
<tr>
<th></th>
<th>CPU (s)</th>
<th>GPU (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8.995</td>
<td>0.258</td>
<td>35</td>
</tr>
</tbody>
</table>

6. Conclusion

The paper realized the parallel implementation of compressive sensing based SAR imaging on GPU, and Iterative Shrinkage/Thresholding algorithm is adopted to reconstruct the SAR image. To make the most efficient use of parallelism characteristic of GPU, we modified the existed IST algorithm structure, and realized the fast implementation on CUDA platform. The experiment result shows that parallel computing capabilities of GPU have a significant speedup in comparison with computing capability of CPU.

7. Acknowledgement

This work was supported by the “973” Program of China under Grant 2010CB731903, the National Natural Science Foundation of China (Grant No. 60901056).

8. References