ChinaRxiv

AI translation · View original & related papers at
chinarxiv.org/items/chinaxiv-201805.00284

Multi-Subarray Synthetic Aperture Sonar Range-Doppler Imaging Algorithm in Heterogeneous Environments (Postprint)

Authors: Zhong Heping, Tang Jinsong, Huang Pan

Date: 2018-05-20T00:00:00+00:00

Abstract

To address the low imaging efficiency of multi-subarray synthetic aperture sonar, a fast imaging method for multi-subarray synthetic aperture sonar in heterogeneous environments is proposed. Based on the characteristics of the multi-subarray synthetic aperture sonar range-Doppler imaging algorithm and the respective computational features of CPU and GPU, the intensive operations in the algorithm—including range compression, fixed phase compensation, range cell migration correction, and azimuth compression—are performed using GPU computation, which significantly improves the imaging efficiency of multi-subarray synthetic aperture sonar. Finally, the correctness and efficiency of the proposed algorithm are verified through imaging experiments with measured data, achieving a speedup of up to 14.45 compared with the serial computation method.

Full Text

Preamble

Range Doppler Algorithm for Multi-array Synthetic Aperture Sonar in Heterogeneous Environment

Zhong Heping, Tang Jinsong, Huang Pan
(Naval Institute of Underwater Acoustic Technology, Naval University of Engineering, Wuhan 430033, China)

Abstract: To address the low imaging efficiency of multi-array synthetic aperture sonar, this paper proposes a fast imaging method for multi-array synthetic aperture sonar in heterogeneous environments. By considering the characteristics of the range Doppler algorithm for multi-array synthetic aperture sonar and the distinct computational features of CPU and GPU, the computationally

intensive operations—including range compression, fixed phase compensation, range cell migration correction, and azimuth compression—are implemented on GPU, which greatly enhances the imaging efficiency for multi-array synthetic aperture sonar. Finally, the validity and efficiency of the proposed method are verified through imaging experiments on real data, achieving a speedup of up to 14.45 times compared to the serial algorithm.

Key Words: synthetic aperture sonar; range Doppler algorithm; parallel computing; GPU

0 Introduction

Synthetic aperture sonar (SAS) is a high-resolution imaging sonar that synthesizes a large virtual aperture by moving a small acoustic array uniformly along a straight line. During this motion, signals are transmitted and received at a constant pulse repetition interval. Echo signals from different positions are coherently superimposed based on their spatial positions and relative relationships to achieve high resolution in the azimuth direction [1-4]. High resolution in the range direction is obtained by transmitting wideband signals and applying pulse compression techniques. Due to these high-resolution capabilities, SAS has found extensive applications in both military and civilian domains in recent years [5,6]. Military applications primarily include detection of bottom/buried mines, unexploded ordnance detection, and high-resolution reconnaissance of critical sea areas. Civilian applications mainly involve underwater topographic mapping, seabed geological exploration, archaeological search and rescue, submarine cable and pipeline laying and positioning, and waterway dredging project volume assessment.

As SAS technology continues to evolve, imaging resolution must constantly improve to meet small target detection requirements, while the swath width must increase to satisfy high-speed mapping demands. These combined factors lead to continuous growth in the number of range samples in the raw data. Simultaneously, as the maximum mapping distance increases with swath width, the synthetic aperture length grows, resulting in an increasing number of azimuth samples. This expansion of raw data volume significantly extends SAS imaging time, reducing efficiency and making it difficult to meet real-time system requirements.

Hardware selection for SAS imaging primarily falls into two categories: dedicated hardware solutions [7] and general-purpose hardware solutions [8-10]. In the early stages of SAS development, general-purpose computers lacked sufficient performance, necessitating dedicated hardware approaches despite their long development cycles, high costs, and limited scalability. With advances in computer technology, researchers improved computational capability by constructing computer clusters, achieving parallelization of SAR and SAS imaging algorithms with notable efficiency gains. However, cluster computers suffer from large size, high power consumption, and lack of mobility required for SAS signal

processing platforms.

The emergence of the Graphics Processing Unit (GPU) provides a new processing platform for SAS signal processing, offering powerful signal processing capabilities, high cost-effectiveness, and broad applicability [11,13]. This paper proposes a GPU-based range Doppler imaging method for multi-array SAS in heterogeneous environments, effectively addressing real-time imaging challenges and providing a fundamental framework for software development of multi-array SAS in heterogeneous environments. We first present the basic principles of the SAS range Doppler imaging algorithm and analyze the signal processing characteristics of each step. We then design parallel implementations for range compression, fixed phase compensation, range cell migration correction, and azimuth compression in the GPU environment. Finally, we validate the performance of the proposed algorithm through imaging experiments on real SAS data.

1 Multi-array SAS Imaging Model

[Figure 1: see original paper] illustrates the multi-array SAS imaging model. In the Cartesian coordinate system, the SAS platform moves along the azimuth direction, with a target located at position (y', R_0). The transmitting array and N receiving sub-arrays are arranged along the azimuth direction, with the transmitting array positioned at the trailing end at location $(0, vt)$. For simplified processing, this paper considers only one receiving sub-array (the i-th sub-array), where the equivalent phase center spacing between this receiving sub-array and the transmitting array in azimuth is $h_i + \Delta h$, assuming the platform moves a distance vt during signal propagation. The echo signal received by the i-th sub-array can be expressed as:

$$s_i(t,\tau;r_c) = A\cdot\mathrm{rect}\left(\frac{t}{T}\right)\cdot\mathrm{rect}\left(\frac{\tau-2R_i(t,r_c)/c}{T_p}\right)\cdot\exp\left(j2\pi f_0\left(\tau-\frac{2R_i(t,r_c)}{c}\right)\right)\cdot\exp\left(j\pi k_r\left(\tau-\frac{2R_i(t,r_c)}{c}\right)^2\right)$$

where $\mathrm{rect}(\cdot)$ is the rectangular window function defined as:

$$\mathrm{rect}(x) = \begin{cases} 1, & |x| \le 0.5 \\ 0, & |x| > 0.5 \end{cases}$$

$R_i(t, r_c) = R_1 + R_2 + R_3$ represents the actual signal propagation path, with the forward path $R_1 = \sqrt{R_0^2 + (vt)^2}$ and return path $R_2 = \sqrt{R_0^2 + (vt + h_i + \Delta h)^2}$. $s(\tau)$ is the transmitted signal envelope, c is the sound speed, f_0 is the signal center frequency, and k_r is the signal chirp rate.

2 Multi-array SAS Range Doppler Imaging Algorithm

The range Doppler algorithm is a classical SAS imaging approach that exploits the property that scatterers at the same range but different azimuths share identical energy trajectories in the range-Doppler domain. Through interpolation to resolve two-dimensional coupling, the 2D imaging processing is converted into a cascade of two 1D processes. The most common method for multi-array SAS imaging is to convert multi-array echo signals into single-array echo signals, then invoke the single-array range Doppler imaging algorithm.

[Figure 2: see original paper] shows the RD algorithm flowchart, where FFT and IFFT denote Fourier transform and inverse Fourier transform, respectively. The flowchart reveals that range and azimuth processing are performed separately. When echo signals cannot be decoupled in range and azimuth, sinc interpolation is required to implement range cell migration correction to complete the decoupling.

From a signal processing perspective, SAS imaging using the range Doppler algorithm mainly comprises four steps: raw data preprocessing, range compression, fixed phase compensation, and azimuth compression. Raw data preprocessing primarily includes effective data truncation, data type conversion, and sub-array data stacking. Effective data truncation determines the number of effective sub-arrays M for imaging based on the relationship between pulse repetition interval PRI, platform velocity V, and sub-array length D: $M = PRI \cdot V/D$, then extracts the valid raw data. Since this preprocessing step involves minimal computation, it is performed directly on CPU. Range compression consists mainly of FFT, pointwise multiplication, and IFFT operations, all processed in the range direction, with completely independent processing across different range lines. Fixed phase compensation converts multi-array received signals into single-array received signals for subsequent single-array imaging algorithm processing. Azimuth compression includes FFT, IFFT, range cell migration correction, and pointwise multiplication, where the range cell migration correction process is a point-by-point interpolation that severely impacts SAS imaging efficiency.

3 GPU Environment Parallel Algorithm

To address the real-time performance issues of the SAS range Doppler imaging algorithm, we redesigned the computational methods for the time-consuming steps—range compression, fixed phase compensation, and azimuth compression —to satisfy GPU parallel processing conditions and improve computational efficiency.

3.1 Range Compression

Range compression first applies FFT to each range line of echo data. Since each range line of echo data is stored contiguously in memory, this step can be directly implemented by calling the efficient FFT transformation function `cufftExecC2C`

in CUDA. Before calling `cufftExecC2C`, the FFT transform handle must be constructed using the `cufftPlan1d` function to specify the transform data type, number of consecutive transform points (range samples), number of consecutive transforms (azimuth samples), and transform type. The transform handle advantageously pre-configures memory and computational resources based on input parameters, enabling the processor to achieve optimal performance during operations.

After completing the range FFT, the data becomes frequency-domain in range and is multiplied by the range reference function $H_r(f_r) = \exp(j\pi f_r^2/k_r)$, where f_r represents range frequency and k_r is the signal chirp rate. The range reference function is essentially a 1D array computed and stored in shared memory before imaging, with the same number of points as the range samples. Since the reference function multiplies each range line and the same data is multiplied at the same range position, to reduce the time overhead of repeated reference function reads, thread blocks are partitioned in 1D along the range direction, as shown in [Figure 3: see original paper]. Assuming the current thread block index is BN with TN threads, this thread block completes pointwise multiplication for range columns $BN \times TN$ to $(BN+1) \times TN - 1$. Each thread in the block sequentially completes pointwise multiplication for all data along the azimuth direction at that range position. After completing the multiplication, the inverse FFT function is called again to finish range compression. Since the inverse FFT handle is identical to the forward transform, only the transform type parameter needs to be changed in the `cufftExecC2C` function call.

3.2 Fixed Phase Compensation

Fixed phase compensation transforms the original multi-array echo format into a single-array echo format for subsequent single-array imaging algorithm processing. The key to SAS imaging lies in the precision of fixed phase compensation, with different scholars investigating specific compensation methods for various environments. Generally, phase compensation precision is proportional to computational complexity. The fixed phase compensation formula considering non-stop-and-hop conditions is:

$$H_i(h_i, \Delta h, r) = \exp\left(j\frac{2\pi f_0}{c}\left(\frac{h_i + \Delta h}{v}\right)^2 \cdot \frac{c}{r}\right)$$

where f_0 is the center frequency, $h_i + \Delta h$ is the equivalent phase center spacing between the i-th receiving sub-array and transmitting array in azimuth, c is the sound speed, v is the platform velocity, and r is the slant range. The compensation value depends only on the receiving sub-array position and slant range, with identical and independent compensation processes across different pulses.

Based on these characteristics, we again employ 1D thread blocks for fixed

phase compensation, as illustrated in [Figure 4: see original paper]. During compensation, the raw data is treated as a 3D structure where each pulse' s raw data forms a 2D array, with different pulses stacked in layers. The 1D threads are partitioned along the range direction. Assuming thread block index BN contains TN threads, the block completes fixed phase compensation for range columns $BN \times TN$ to $(BN+1) \times TN - 1$. Each thread sequentially performs compensation for all data at the same range position, first across pulses then across sub-arrays. To save computation and fully exploit the compensation characteristics, the compensation factor is constant when sub-array and range are fixed. Therefore, the compensation order processes all pulse data for the first sub-array first, then sequentially processes each subsequent sub-array, effectively avoiding redundant calculations.

3.3 Azimuth Compression

Azimuth compression is performed in range-time/azimuth-frequency domain, with range cell migration correction at its core. [Figure 5: see original paper] shows the azimuth compression flow. After fixed phase compensation, FFT in azimuth is required. To satisfy CUDA FFT function requirements, a data transpose operation is necessary to ensure memory contiguity. The flowchart shows that two matrix transpose operations are needed before and after FFT and IFFT transforms. The matrix transpose process utilizes shared memory and coalesced read/write patterns to improve transpose speed.

After transforming data to range-time/azimuth-frequency domain, the migration amount is calculated using the range migration formula:

$$\Delta R(f_a, r) = \frac{\lambda^2 r f_a^2}{8v^2}$$

where r is slant range, f_a is azimuth frequency, and $f_{a_{max}}$ is maximum azimuth frequency. The migration amount is a binary function of slant range and azimuth frequency. During range cell migration correction, data correction processes at different azimuth frequencies are completely independent. We employ 1D threads where each thread block completes migration correction for all azimuth frequency data at the current range position. Since adjacent threads require overlapping data for interpolation at the same azimuth frequency, to reduce data access latency, the required data is first loaded into shared memory by the current thread block before correction, then directly used from shared memory during correction.

Assuming interpolation points N_{int}, the spatial range migration amount for thread i in the current 1D thread block is ΔR_i, with corresponding range migration cell number $\Delta N_i = \lfloor \Delta R_i / \Delta r \rfloor$. The original data indices used for interpolation range from $N_{ri} + \Delta N_i - N_{int}/2$ to $N_{ri} + \Delta N_i + N_{int}/2$. After completing range cell migration correction, the result is multiplied by the azimuth reference function $H_a(f_a, r) = \exp(j\pi f_a^2 / k_a(r))$, where f_a is Doppler frequency

and k_a is Doppler chirp rate as a function of slant range r. Finally, azimuth IFFT is performed to complete azimuth compression and obtain the final imaging result.

4 Experimental Results

To verify the efficiency of the proposed heterogeneous imaging algorithm, tests were conducted on the following platform: Intel(R) Core(TM) i5-4200H CPU @ 2.80 GHz (4 cores), NVIDIA GeForce GTX 950M GPU, with VS2008+CUDA5.0 development environment. To test algorithm efficiency, we compared CPU serial computation, OpenMP-based shared-memory parallel computation, and CPU+GPU heterogeneous parallel computation methods. The interpolation points for range cell migration correction were set to 6. In the heterogeneous parallel algorithm, the 1D thread block size was set to 256.

The experiment used real data acquired in July 2010 during an interferometric SAS sea trial of a prototype system in an inland lake. System parameters are listed in . The raw echo amplitude information is shown in [FIGURE:6(a)], with 9600 range samples and 3456 azimuth samples. Data acquisition started at 51m and ended at 231m, showing good amplitude consistency in range. The azimuth data consisted of 72 pulses across 4 synthetic aperture lengths, with total acquisition time of 23.04s. Due to one synthetic aperture length overlap between adjacent raw data blocks in azimuth, the actual usable signal processing time was only 17.28s. All three imaging methods produced identical results, clearly reconstructing the scene contour as shown in [FIGURE:6(b)], validating the effectiveness of the range Doppler algorithm.

To fully demonstrate the efficiency advantages of the heterogeneous range Doppler imaging algorithm, we compared it with serial and shared-memory parallel methods, with results summarized in . The SAS imaging preprocessing steps—including raw data truncation, data type conversion, and sub-array data stacking—involve minimal computation and are negligible in efficiency comparisons. The results show that CPU serial computation required the longest time at 14,653 ms. Using OpenMP shared-memory parallelism significantly reduced computation time for all components, decreasing total time to 5,706 ms with a speedup of 2.58. GPU computation dramatically improved efficiency for range compression, azimuth FFT, and azimuth compression—reducing times from 3,736 ms, 2,783 ms, and 3,895 ms to 82 ms, 109 ms, and 143 ms respectively, by leveraging CUDA' s efficient FFT library. Although GPU computation added overhead for memory allocation, data upload, and download, the total time decreased substantially to only 1,013 ms, achieving an impressive speedup of 14.45 and meeting real-time imaging requirements.

5 Conclusion

This paper proposes a range Doppler imaging method for multi-array synthetic aperture sonar in heterogeneous environments. By parallelizing the compu-

tationally intensive steps of the imaging algorithm on multi-core GPU, we achieve significant computational efficiency improvements, with a speedup of up to 14.45 compared to the serial algorithm, effectively solving the real-time imaging problem for SAS. Experimental results indicate that the heterogeneous range Doppler imaging algorithm's time consumption is dominated by the range cell migration correction step. Future work will focus on parallel optimization methods for this step to further enhance algorithm efficiency.

References

[1] Hayes M P, Gough P T. Synthetic aperture sonar: a review of current status [J]. IEEE Journal of Oceanic Engineering, 2009, 34 (3): 207-224.

[2] Zhang X B, Tang J S, Zhong H P. Multireceiver correction for the chirp scaling algorithm in synthetic aperture sonar [J]. IEEE Journal of Oceanic Engineering, 2014, 39 (3): 472-481.

[3] Tian Z, Tang J S, Zhong H P, et al. Extended Range Doppler Algorithm for Multiple-Receiver Synthetic Aperture Sonar Based on Exact Analytical Two-Dimensional Spectrum [J]. IEEE Journal of Oceanic Engineering, 2016, 41 (1): 164-174.

[4] Zhang Xuebo, Huang Haining, Ying Wenwei, et al. An indirect range-doppler algorithm for multireceiver synthetic aperture sonar based on lagrange inversion theorem [J]. IEEE Trans on Geoscience and Remote Sensing. 2017, 55 (6): 3572-3587.

[5] Carballini J, Viana F. Using synthetic aperture sonar as an effective tool for pipeline inspection survey projectsg [C]// Proc of Acoustics in Underwater Geosciences Symposium. 2015: 1-5.

[6] Hoggarth A, Kenny K. Using synthetic aperture sonar as an effective hydrographic survey tool [J]. Oceans, 2014, 107 (3): 1-12.

[7] Cholewa F, Pfitzner M, Fahnemann C, et al. Synthetic aperture radar with backprojection: A scalable, platform independent architecture for exhaustive FPGA resource utilization. [C]// Proc of Radar Conference. 2014: 1-5.

[8] 徐永健, 王贞松, 罗晓广. 星载 SAR 数据成像的并行和实时处理研究-曙光机的应用实例 [J]. 电子与信息学报, 2001, 23 (8): 736-742.

[9] 龙卉, 皮亦鸣, 黄顺吉. 合成孔径雷达并行成像算法研究及在曙光 3000 并行计算机上的实现 [J]. 电子与信息学报, 2002, 24 (12): 1990-1995.

[10] 江泽林, 刘维, 李保利, 等. 基于集群的高频合成孔径声呐并行处理方法 [J]. 应用声学, 2011, 30 (3): 167-176.

[11] Fasih A, Hartley T. GPU-accelerated synthetic aperture radar backprojection in CUDA [C]// Proc of Radar Conference. 2010: 1408-1413.

[12] Ning Xia, Yeh C, Zhou B, et al. Multiple-GPU accelerated range-doppler algorithm for synthetic aperture radar imaging [C]// Proc of Radar Conference. 2011: 698-701.

[13] 吴胜强, 彭建平, 郭建强. 基于 GPU 的合成孔径超声成像算法并行处理 [J]. 信息技术, 2017, 30 (2): 126-129.

Funding: National Natural Science Foundation of China (61671461, 41304015); China Postdoctoral Science Foundation (2015M582813)

Authors: Zhong Heping (born 1983), male, from Zhongxiang, Hubei, lecturer, Ph.D., research interests: interferometric signal processing and parallel computing (zheping525@sohu.com); Tang Jinsong (born 1964), male, researcher, Ph.D., research interests: synthetic aperture sonar and underwater acoustic communication; Huang Pan (born 1986), male, Ph.D. candidate, research interests: interferometric synthetic aperture sonar image processing.

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv —Machine translation. Verify with original.