Postprint of an OpenMP-Based Efficient Multi-Subarray Synthetic Aperture Sonar Range-Doppler Imaging Algorithm

Authors: Zhong Heping, Huang Pan, Tang Jinsong

Date: 2018-05-02T00:00:00+00:00

Abstract

To fully utilize multi-core CPU computational resources and address the low imaging efficiency of multi-subarray synthetic aperture sonar, a parallel solution for the Range-Doppler imaging algorithm in a shared-memory environment is proposed. Based on an analysis of the parallelism in the multi-subarray synthetic aperture sonar Range-Doppler imaging algorithm, OpenMP parallelization design was implemented for preprocessing, range compression, fixed-phase compensation, range cell migration correction, and azimuth compression within the algorithm, thereby fully leveraging multi-core CPU computational resources to achieve rapid reconstruction of large-data-volume synthetic aperture sonar images. Imaging experimental results using measured data demonstrate that the parallel imaging algorithm achieves a speedup ratio as high as 19.86, meeting the imaging requirements of real-time synthetic aperture sonar systems.

Full Text

Preamble

Title: Efficient Range-Doppler Imaging Algorithm for Multi-Array Synthetic Aperture Sonar Based on OpenMP

Authors: Zhong Heping, Huang Pan, Tang Jinsong
Affiliation: Naval Institute of Underwater Acoustic Technology, Naval University of Engineering, Wuhan 430033, China

Abstract: To fully utilize multi-core CPU computing resources and address the low efficiency problem in multi-array synthetic aperture sonar (SAS) imaging, this paper proposes a parallel solution for the range-Doppler imaging algorithm in a shared-memory environment. Based on an analysis of the parallelism inherent in the multi-array SAS range-Doppler imaging algorithm, we designed

OpenMP parallelization for the algorithm' s key stages: preprocessing, range compression, fixed-phase compensation, range cell migration correction, and azimuth compression. This approach leverages multi-core CPU resources to achieve rapid reconstruction of large-scale SAS images. Imaging experiments on real measured data demonstrate that the parallel imaging algorithm achieves a speedup of up to 19.86, meeting the real-time imaging requirements of SAS systems.

Keywords: synthetic aperture sonar; range-Doppler imaging algorithm; parallel computing; shared memory; OpenMP

0 Introduction

Multi-array synthetic aperture sonar (SAS) is a high-resolution imaging sonar developed in recent years, characterized by azimuth resolution independent of range and one to two orders of magnitude higher than conventional side-scan sonar, making it valuable for underwater small target detection applications. A critical prerequisite for multi-array SAS application is the rapid reconstruction of SAS images. The range-Doppler algorithm (RDA) is a classical synthetic aperture imaging algorithm that combines modular processing capabilities with one-dimensional operational simplicity, and remains widely used in various synthetic aperture imaging scenarios. However, as SAS system resolution and swath width continue to increase, the volume of raw data for imaging grows substantially, severely impacting SAS system imaging efficiency.

Traditional real-time SAS imaging has relied on dedicated signal processors, but these suffer from long development cycles, high costs, and poor scalability. While cluster systems have been applied to SAS imaging to achieve real-time processing results, they exhibit large size and high power consumption that fail to meet the compact size, low power consumption, and transportability requirements of underwater acoustic equipment. With the maturation of multi-core technology, shared-memory architectures have become standard across various computing platforms. Shared-memory parallel programs offer excellent scalability, with performance proportional to single-core computing speed and core count, enabling direct execution across different platforms. Building upon this foundation, this paper proposes a shared-memory-based fast range-Doppler imaging algorithm for multi-array SAS that effectively improves imaging efficiency and satisfies real-time SAS imaging requirements.

1 Shared Memory Parallelism

1.1 OpenMP Parallel Model

OpenMP is a parallel programming standard for shared-memory models, providing an application programming interface for writing parallel programs in shared-memory environments. Currently supporting Fortran and C/C++, the standard includes a set of compiler directives and a supporting function library. OpenMP employs a "Fork/Join" execution model, illustrated in [Figure 1: see original paper]. The program begins with a single master thread that executes all serial regions. Parallel sections spawn additional threads to work collaboratively. As shown in the figure, OpenMP programs cannot execute serial portions until all parallel sections complete, representing the standard Fork/Join parallel paradigm.

1.2 Parallel Computing Efficiency

Speedup is a crucial metric for parallel algorithm performance, quantitatively describing performance gains from reduced execution time through parallelization. Assuming an application' s serial execution time is T_1 and its parallel execution time on p processors is T_p, the speedup on that computer is defined as $S_p = T_1/T_p$. Theoretically, larger S_p values indicate higher parallel system efficiency, though speedup typically remains below the total CPU core count. Therefore, speedup should ideally approach the CPU core number.

High parallel algorithm efficiency does not necessarily imply high processor utilization, necessitating the concept of efficiency E_p. Efficiency measures the ratio of each processor' s effective usage time to total execution time, typically defined as $E_p = S_p/p$. The ideal parallel algorithm efficiency is 1, but practical values are generally lower due to insufficient parallelization and overhead from communication and synchronization.

1.3 Test Platform

To evaluate parallel imaging algorithm performance, this study utilized both a laptop and a blade computing node with the following configurations:

Laptop: Intel(R) Core(TM) i5-4200H CPU @ 2.80 GHz (4 cores), NVIDIA GeForce GTX 950M graphics, 4 GB RAM, Windows 7 Ultimate 64-bit OS, Visual Studio 2008 development environment.

Blade Computing Node: Dual Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.0 GHz (40 cores total), 32 GB RAM, Windows Server 2008 64-bit OS, Visual Studio 2008 development environment.

2 Algorithm Description

2.1 Multi-Array SAS Range-Doppler Imaging Algorithm

The range-Doppler imaging algorithm exploits the property that scatterers at the same range but different azimuths share identical energy trajectories in the range-Doppler domain. By interpolating to resolve two-dimensional coupling, the algorithm transforms two-dimensional processing into a cascade of two one-dimensional operations, achieving separation between range and azimuth processing. The common imaging approach for multi-array SAS converts multi-array echo signals into a single-array format, then applies the single-array range-Doppler imaging algorithm for image reconstruction. [Figure 2: see original paper] illustrates the basic flow of the multi-array SAS range-Doppler imaging algorithm, where FFT and IFFT denote Fourier transform and inverse Fourier transform, respectively.

From a signal processing perspective, multi-array SAS range-Doppler imaging primarily comprises four steps: preprocessing, range compression, fixed-phase compensation, and azimuth compression. During azimuth compression, echo signals transform into the range-Doppler domain where range and azimuth signals remain coupled, requiring computationally expensive interpolation operations to achieve decoupling.

2.2 Algorithm Parallel Performance Analysis

2.2.1 Preprocessing SAS raw data is stored sequentially by acquisition pulse order, with channels ordered within each pulse. For a given SAS raw echo dataset, let N_R represent range points and N_A represent azimuth points, corresponding to pulse count and receiver array count, respectively. Preprocessing mainly involves data type conversion and valid sub-array data extraction. Data type conversion transforms sample points from short to float type, with parallelism of $N_R \cdot N_A$. Valid sub-array extraction determines the number of effective sub-arrays M for imaging based on the relationship between pulse repetition interval, platform velocity, and sub-array length, then truncates the first M sub-array data from each pulse to form a new data block of size $M \cdot N_A \cdot N_R$. Since truncation operates on individual pulse data as data movement operations, its parallelism is N_{Pulse}.

2.2.2 Range Compression Range compression of preprocessed data includes three steps: range FFT, multiplication with the range reference function, and range IFFT. Since preprocessed data is stored by pulse order, different azimuth range lines are independent during range FFT transformation (i.e., image rows are independent), enabling simultaneous FFT operations with parallelism of N_A. The range reference function is $H_r(f_r) = \exp(j\pi f_r^2/\mu)$, where f_r represents range baseband frequency and μ is the signal chirp rate. Multiplication with each range line is independent, and multiplication across different range lines is also independent, yielding parallelism of N_A. Similar to FFT processing, range

IFFT transformation also achieves parallelism of N_A.

2.2.3 Fixed-Phase Compensation Fixed-phase compensation transforms multi-array echoes into single-array format after range compression, enabling processing with the single-array SAS range-Doppler imaging algorithm. Compensation accuracy directly affects final imaging resolution, with various scholars investigating phase compensation methods for specific imaging conditions. The fixed-phase compensation formula incorporating stop-and-go approximation corrections, considering only range-variant scenarios, is:

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

where f_0 is the center frequency, Δh_i is the azimuthal equivalent phase center spacing between the i-th receiving sub-array and transmitting array, c is sound speed, v is platform velocity, and r is slant range. The compensation factor is a function of two variables (Δh_i and Δr), with each point' s compensation process being independent, resulting in parallelism of $N_R \cdot N_A$.

2.2.4 Azimuth Compression Azimuth compression comprises azimuth FFT, range cell migration correction (RCMC), multiplication with the azimuth reference function, and azimuth IFFT. Since sonar data is arranged in range order, matrix transposition is required before azimuth processing, with another transposition returning to the original arrangement after completion. During azimuth FFT transformation, azimuth lines across different ranges are independent, enabling simultaneous FFT operations with parallelism of N_R.

RCMC uniformly corrects energy curves for point targets at the same nearest range but different azimuths through interpolation in the range-Doppler domain. Its efficiency depends on the interpolation kernel, with 8-point sinc interpolation typically sufficient. After transforming raw data to the range-Doppler domain, the time offset requiring correction is a two-dimensional space-variant function:

$$\Delta\tau_a(f_a,r)=\frac{1}{4}\left(\frac{\lambda^2 r f_a^2}{c^2 v^2}-1\right)$$

where c is sound speed, v is platform velocity, r is slant range, f_0 is center frequency, and f_a is azimuth frequency. Each point' s correction process in the range-Doppler domain is independent, yielding parallelism of $N_R \cdot N_A$.

In the range-Doppler domain, the azimuth reference function is:

$$H_a(f_a,r)=\exp\left\{-j\frac{4\pi r}{\lambda}\sqrt{1-\left(\frac{\lambda f_a}{2v}\right)^2}\right\}$$

This represents a two-dimensional space-variant function of f_a and r. Multiplication with each point in the range-Doppler domain is independent, resulting in parallelism of $N_R \cdot N_A$. The final IFFT transformation in azimuth compression is similar to FFT, with parallelism of N_R.

summarizes the parallelism of each processing step in the imaging algorithm.

2.3 Implementation Details

2.3.1 Preprocessing Preprocessing includes data type conversion and valid sub-array extraction. To avoid address calculations, the former uses a single for-loop while the latter employs nested for-loops, with the outer loop iterating over pulse count. Both are parallelized using parallel for directives. The valid sub-array extraction step parallelizes the outer pulse loop. The pseudocode is:

```
Preprocessing(A, B, NumOfPulse, Nc)
Input:
  A: raw data in complex<short> type
  B: data in complex<float> type
  NumOfPulse: number of pulses
  Nc: number of valid sub-arrays

parallel for i = 1 to length[A]
  B[i] ← A[i] / 32768.0

parallel for i = 1 to NumOfPulse
  for j = 1 to Nc
    Extract j-th sub-array data of i-th pulse from B to form new data block
    Discard excess sub-array data
```

2.3.2 Range Compression Before range compression, the range reference function must be calculated. To enable direct multiplication with FFT results, the frequency range is set to $[-F_s/2, F_s/2]$ to avoid additional operations from spectral shifting, where F_s represents range sampling frequency. Range compression parallelizes the outer row loop using parallel for directives, with each row performing FFT, pointwise multiplication with the range reference function, and IFFT to complete range compression in parallel.

2.3.3 Fixed-Phase Compensation The compensation factor is identical across different pulse data, and the compensation process is uniform. To avoid repeated calculations, this study first allocates temporary space equivalent to a single pulse' s data size to store phase compensation factors, computes the compensation factor array, then multiplies it with sonar data to complete multi-to-single array conversion. During initialization, the range sample count loop serves as the outer loop with sub-array count as the inner loop. When multiplying with sonar data, the pulse count loop becomes the outer loop, parallelized via parallel for directives.

2.3.4 Azimuth Compression After fixed-phase compensation, data is arranged range-first then azimuth-second, requiring data reorganization before azimuth FFT. Considering the independence of FFT transformations across different range lines, this study allocates temporary data space of length N_A within each thread to relocate discretely stored azimuth data into contiguous memory, performs FFT, then moves data back and releases temporary space.

RCMC follows, with correction processes independent across different rows (representing azimuth frequencies). The outer row loop is directly parallelized using parallel for directives. Since interpolation requires adjacent data at fixed offset positions, strict index validation is necessary during array access within threads.

The final step involves multiplication with the azimuth reference function and IFFT transformation. While processing is independent across different ranges, the azimuth reference function varies by range. To enable simultaneous operations, each thread allocates temporary storage of length N_A, relocates discretely stored azimuth data into contiguous memory, computes the azimuth reference function for multiplication, performs IFFT, then restores data to complete azimuth compression and final SAS image reconstruction.

---

3 Experimental Results

To verify the efficiency of the proposed multi-array SAS range-Doppler imaging algorithm in shared-memory environments, this study tested both CPU serial and OpenMP-based shared-memory parallel computing methods on the two platforms described in Section 1.3. The same program was used on both platforms, automatically detecting core count at startup and configuring it as the computational resource before imaging processing. Range cell migration correction employed 8-point sinc interpolation. Preprocessing time was negligible in efficiency analysis.

The test data was acquired during July 2010 field trials of an interferometric SAS prototype in an inland lake. System parameters are detailed in . The raw echo signal amplitude is shown in Figure 3: see original paper. The data block contains 72 pulses across 4 synthetic aperture lengths, with $N_R = 9,600$ range points and $N_A = 3,456$ azimuth points, acquired over 23.04 seconds. Due to one synthetic aperture length overlap between adjacent data blocks, the effective imaging time window is only 17.28 seconds. The number of valid sub-arrays used in processing was 40, reducing the final image azimuth points to 2,880.

Serial and parallel imaging experiments were conducted on both platforms. Figure 3: see original paper and (c) show imaging results from serial and parallel execution, respectively. The identical results validate the parallel implementation' s correctness.

Performance comparisons are presented in . On Platform A (laptop), serial imaging required 16,731 ms versus 6,534 ms for parallel execution, achieving

a speedup of 2.56 with 0.64 efficiency. Individual step speedups were: range compression 2.39, fixed-phase compensation 2.40, azimuth FFT 2.57, RCMC 2.53, and azimuth IFFT 2.76.

On Platform B (blade node), serial imaging took 19,244 ms compared to only 969 ms for parallel execution, yielding a speedup of 19.86 and efficiency of 0.50. Step-wise speedups were: range compression 13.55, fixed-phase compensation 9.41, azimuth FFT 18.01, RCMC 23.47, and azimuth IFFT 27.83. The higher speedup on Platform B results from increased core count, though efficiency is lower compared to Platform A. The parallel imaging time of 969 ms satisfies real-time SAS system requirements when compared to the raw data acquisition time of 17.28 seconds.

---

4 Conclusion

This paper analyzed the parallel characteristics of multi-array SAS range-Doppler imaging algorithms and proposed a fast SAS range-Doppler imaging algorithm using OpenMP parallel tools in a shared-memory environment. Parallelization designs for key processing steps—including preprocessing, range compression, fixed-phase compensation, and azimuth compression—were completed. Performance and efficiency were evaluated on different parallel computing platforms using measured data. Experimental results demonstrate that the proposed parallel imaging algorithm' s speedup increases with core count, meeting real-time imaging requirements when sufficient cores are available.

References

[1] Sæbø T O, Synnes S A V, Hansen R E. Wideband interferometry in synthetic aperture sonar [J]. IEEE Trans on Geoscience and Remote Sensing, 2013, 51(8): 4450-4459.

[2] 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.

[3] 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.

[4] 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.

[5] Piper J E, Commander K W, Thorsos E I, et al. Detection of buried targets

using a synthetic aperture sonar [J]. IEEE Journal of oceanic engineering, 2002, 27(3): 495-504.

[6] Hansen R E, Sæbø T O, Callow H J, et al. Interferometric synthetic aperture sonar in pipeline inspection [C]// Proc of OCEANS 2010 IEEE-Sydney. [S.l.]: IEEE Press, 2010: 1-10.

[7] 杨敏, 宋士林, 徐栋, 等. 合成孔径声纳技术以及在海底探测中的应用研究 [J]. 海洋技术学报, 2016, 35(2): 51-55.

[8] Riyait V S, Lawlor M A, Adams A E, et al. Real-time synthetic aperture sonar imaging using a parallel architecture [J]. IEEE Trans on Image Processing, 1995, 4(7): 1010-1019.

[9] 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.

[10] 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.

[11] 陈东升, 刘维, 刘纪元, 等. 基于 PowerPC 的合成孔径声纳实时信号处理系统 [J]. 微计算机应用, 2008, 29(7): 6-10.

[12] Li B, Liu W, Liu J, et al. Real-time implementation of synthetic aperture sonar imaging on high performance clusters [C]// Proc of the 11th ACIS International Conference on Software Engineering Artificial Intelligence Networking and Parallel//Distributed Computing. 2010: 89-92.

[13] Dagum L, Menon R. OpenMP: An industry standard API for shared-memory programming [J]. IEEE Computational Science and Engineering, 1998, 5(1): 46-55.

[14] 杨海亮, 张森, 唐劲松, 等. 一种精确的多接收阵合成孔径声纳距离多普勒成像算法 [J]. 数据采集与处理, 2010, 25(3): 313-317.

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