Hindawi
Scientific Programming
Volume 2019, Article ID 8908950, 9 pages
https://doi.org/10.1155/2019/8908950

Research Article

Effective Parallelization Method for Object Recognition in 2D Sonar Images Based on Task Partitioning

Ok-Kyoon Ha,[1] **Keonpyo Lee**,[2] **Wan-Jin Kim**,[3] **and Kun Su Yoon**[4]

[1]*Department of Aeronautics & Software Engineering, Kyungwoon University, Gumi, Republic of Korea*
[2]*Department of Aerospace & Software Engineering, Gyeongsang National University, Jinju, Republic of Korea*
[3]*Agency for Defense Development, Jinhae, Republic of Korea*
[4]*Department of Aviation Control System, Aviation Campus of Korea Polytechnic, Sacheon, Republic of Korea*

Correspondence should be addressed to Ok-Kyoon Ha; okha@ikw.ac.kr

Received 28 September 2018; Revised 17 December 2018; Accepted 14 January 2019; Published 3 March 2019

Academic Editor: Cristian Mateos

Techniques for analyzing and avoiding hazardous objects and situations on the seabed are being developed to ensure the safety of ships and submersibles from various hazards. Improvements in accuracy and real-time response are critical for underwater object recognition, which rely on underwater sonar detection to remove noises and analyze the data. Therefore, parallel processing is being introduced for real-time processing of two-dimensional (2D) underwater sonar detector images for seabed monitoring. However, this requires optimized parallel processing between the modules for image processing and the data processing of a vast amount of data. This study proposes an effective parallel processing method, called Task Partitioning, based on central and graphical processing units for monitoring and identifying underwater objects in real time based on 2D-imaging sonar. The practicality of the proposed method is evaluated experimentally by comparing it to the sequential processing method. The experimental results show that the Task Partitioning method significantly improves the processing time for sonar images because it reduces the average execution time to 1% and 5% of the sequential processing method and general parallelization, respectively.

1. Introduction

During navigation, vessels face various threats, such as underwater mines and submarines/submersibles [1]. The safety of vessels can be assured by identifying hazards in the water via real-time monitoring and surveillance of the seabed environment. Most seabed environment monitoring systems use sound waves (i.e., sonar), which have a relatively innocuous effect on the underwater environment. Imaging sonar, an underwater image acquisition technology, generates images by transmitting sound waves, ranging from several tens of hertz to several megahertz, depending on the purpose, and analyzes signals reflected from the seabed or objects. To acquire the images, side-scan sonar [2], multibeam echo sounder [3], and synthetic aperture sonar [4] are used. Sonar-based seabed environment monitoring has great difficulty in detecting and identifying objects because of low resolution and disturbances in the underwater environment [5, 6]. Thus, the information collected using sonar contains a large amount of noise, making image processing expensive. Therefore, the usability of imaging sonar on unmanned platforms is low because of the time required to process images.

This study proposes an effective software method, called the Task Partitioning, to enable real-time parallel processing and the identification of objects on the seabed via sonar-image analysis. Our proposed method partitions images based on a multicore central processing unit (CPU), and it performs parallel processing in task units, detecting objects using a graphical processing unit (GPU). This method maximizes parallelism by piping jobs to CPUs and GPUs. The practicality of the proposed method is evaluated by applying it to a sonar image analysis simulator and by experimentally comparing it with the sequential and parallel processing methods [7]. The experimental results show that the proposed method significantly improves the processing

time for removing the noises of the sonar images; our method reduces the average execution time to 1% and 5% of that of the sequential processing method and general parallelization method, respectively. Moreover, our method showed an improvement of 164.5 times compared to sequential processing, and an improvement of 17.4 times compared to general parallelization in terms of the time consumed for identifying multiple undersea objects in 4K images having 4096×4096 resolution.

2. Background

2.1. Parallel Image Processing. The popularity of multiprocessors and multicore systems has increased the demand for improving the processing speed of image processing applications. However, hardware unit multiplexing (e.g., CPU) for simple processing has mostly produced disappointing results. Even so, it is well known that the image processing problem can be solved by software parallel processing. By parallel processing, the expected processing speed improvement value, $\text{SpeedUp}_{\parallel}$, can be obtained using Amdahl's law [8], as follows:

$$\text{SpeedUP}_{\parallel}(f, n) = \frac{1}{(1-f) + (f/n)}, \quad (1)$$

where f denotes the parallel processing ratio during the performance of a given application and n denotes the number of CPU or GPU cores where the application is executed. From Eq. (1), it can be seen that the processing speed improvement by parallelization increases in proportion to the ratio of code sections processed via multiple parallel cores. For example, in a system using eight cores, the performance improvement effects of the parallel-processed sections are 40% and 80%, with $\text{SpeedUp}_{\parallel}(0.4, 8) \fallingdotseq 1.6$ and $\text{SpeedUp}_{\parallel}(0.8, 8) \fallingdotseq 3.3$, respectively.

Effective parallelization is difficult with image processing because of the overhead, owing to the small dataset and short running time. These difficulties are faced by most image processing applications, which inherit the problem of load balancing because of limited parallelism [9]. Therefore, effective parallel processing methods for image processing applications are being researched. The key factor for effective parallel image processing is the selection of a software parallel processing method that is well matched to the CPU and GPU architectures. Thus, the target processor must be selected after understanding the characteristics of the algorithm, prior to the implementation. Furthermore, the algorithm must be designed per the characteristics of each processor [9–11].

2.2. Parallel Processing in Image Analysis. Parallel processing technologies are applied to various fields, including image processing and analysis and computer vision. First, Kim et al. [7] applied CPU and GPU parallel processing for an autonomous navigation robot's object recognition, proposing a parallel-processed keypoint detection method for feature extraction. They compared the running speeds of CPU and GPU parallel processing using the scale-invariant feature transform (SIFT) algorithm [12]. In each implementation, several optimized methods, such as the OpenMP [13], single-input multiple-data (SIMD) [14] structures, and streaming SIMD extensions (SSE) [15], were used for the CPU. CUDA (a parallel computing platform and programming model developed by NVIDIA) [16] was used for the GPU. The study suggested that the keypoint detection method could improve the performance 2.5 to 5 times, compared to the extant methods.

Park et al. [17] applied parallel processing to a three-dimensional (3D) visualization tool that simulated trawl fishing using complex internal calculations of each functional component of the gear and the underwater net. The 3D simulator developed in the study applied parallel processing to achieve an improved average processing performance of 40%, including real-time display of user inputs.

A representative example of the application of parallel processing in sonar image synthesis was SIGMAS+ [18] of the NATO Undersea Research Centre, which implemented each process using a structure optimized for GPU processing, enabling approximately 50 times improved execution and speed performance rendering simple scenes. However, when the results of [18] was used for object detection and tracking, additional overhead was generated because of the additional data conversion required owing to the characteristic of GPU architecture, whose double precision operation performance was lower than single precision.

Our research team developed an image analysis simulator that detects and identifies underwater objects from two-dimensional (2D) sonar images. For recognizing underwater objects, the developed analysis simulator leveraged a preprocessing unit that removes noise from sonar images, using Sonar Image Creator, similar to SIGMAS+ object detection. The tool detects objects in the preprocessed images and object identification units, which identify the detected objects. Figure 1 shows a schematic of the developed 2D Sonar Image Analysis Simulator (SIAS). For real-time processing via the improved performance of SIAS, parallelization, based on the results of [7], was applied to preprocessing, object detection, and object identification units. However, this method improved the average performance of object recognition by 45%, which was insufficient for achieving real-time capacity. Therefore, our study proposes an effective parallelization method for improving the real-time capacity of SIAS.

3. Optimized Parallelization Design for SIAS

3.1. Task Partitioning Method. In a seabed environment that is expressed using 2D sonar images, the image coordinates are determined by the distance from the sensor. Although parallelization is possible for an algorithm used during preprocessing and object detection for analysis, there is a section that parallelization cannot handle, owing to data dependencies of the image area and processing step. Thus, each algorithm performs parallel processing as a parallelization series. Shapes consisting of multiple forks and joins increase the frequent fork-join overhead and the sequential

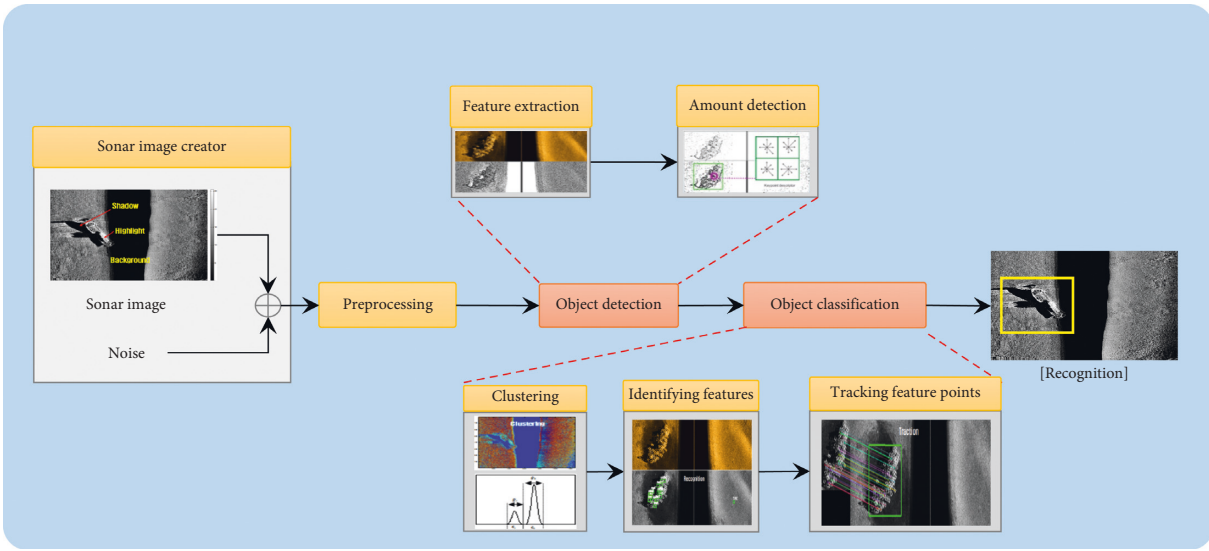


Figure 1: Overall architecture of the 2D sonar image analysis simulator.

processing section ratio. However, for the acquired side-scan sonar images, an empty image is output at the center, owing to the distance to the seabed below the sensor. The left and right sides of the image are fully divided and can be regarded as independent images. Task Partitioning performs parallel processing by dividing the left and right independent images into separate tasks during parallel processing for SIAS.

Task division by image portioning minimizes the sequence section processing of the algorithm because the left and right images are independently processed in parallel. For each task, to maximize the effects of parallelization, the images that can be processed in parallel are allocated to the threads corresponding to the maximum number of threads supported by the CPU. For general parallelization, the use of multiple threads decreases the parallelization efficiency because, after parallelization, the thread waits until other threads that branched simultaneously are completed.

However, during task portioning, as those of the left and right images are separated, the parallel threads in each task can occupy the CPU without waiting. Thus, performance improvement by parallelization can be expected. Furthermore, the performance improvement efficiency of parallelization can be maximized because 256-bit data can be processed simultaneously using the SIMD register, which performs the same command for multiple data in each thread's process. Figure 2 shows the Task Partitioning concept applied to improve the efficiency of parallel processing during SIAS preprocessing. In Figure 2, a tile is a standardized image area that not only enables equal task distribution to each thread but also limits the size of tasks processed by the CPU.

By Task Partitioning, parallelization entails the image division process for identifying variable vertical sections for the inputted sonar image and the task division processes, which allocate the segmented images by tile units to each thread. First, during the image division, the central section of the sonar image, based on the side-scan sonar, varies by the photographing situation and is identified and converted to a simplified 2D binary image. The image division method is shown in Algorithm 1, where it is divided into a port image (InL) and a starboard (InR) image using the size and position of the identified image sections.

During task division, the preprocessing algorithm that improves noise removal and accuracy of object identification for the left and right images are divided during image division, and the algorithm for detection is divided so that they both can operate as one task. Then, tasks are generated for each image, and they are divided for parallel processing based on the number of system cores. The task division process is shown in Algorithm 2, which applies Detection_Obj( ), a function for object detection that replaces the preprocessing algorithm. Therefore, the required object is identified in each divided image, and the search of each section is performed in parallel per the number of threads activated by the system or user.

3.2. Pipeline Method. Besides the processing speed improvement that can be obtained by the CPU task division, additional performance improvements are possible by allowing the GPU to process the algorithm. Single instruction multiple thread (SIMT) [19] parallel processing, which uses a many-core GPU, is more efficient than a CPU when the same command is repeatedly used for each pixel or feature point, such as with object detection and recognition. However, if the Task Partitioning method is applied using a GPU for object detection, it can decrease the gain from parallelization, owing to the overhead caused by frequent context changes.

Meanwhile, preprocessing for noise removal and object detection and recognition cannot use the two processing resources of CPU and GPU simultaneously because of a dependency in the processing sequence. Thus, the GPU must wait until sonar images are preprocessed by the CPU. Even if parallel processing is carried out using both the CPU and GPU, the time required for detecting the objects by using the two processing resources satisfies the following equation:

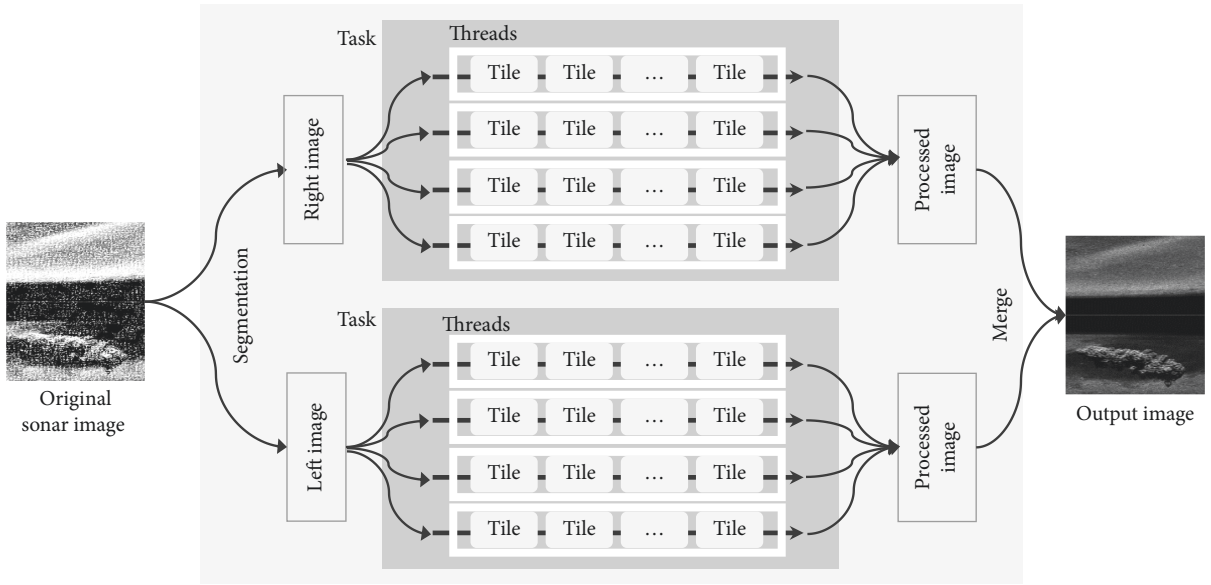


Figure 2: Design of the Task Partitioning approach.

```
FUNCTION Img_Partitioning(I, InL, InR)
    /*Color transform*/
    If an Image I is not a binary Image then
The Image I is converted to a new 2D Binary Image N with a threshold value
    else
      The Image I is just copied to a new Image N
    end if
    /*Labeling*/
    Find connected components in N
    Calculate position of each group
    Find the region of center in N
    /*Partitioning*/
    Divide N into InL and InR based on the center region
    if InL is empty or InR is empty then
      return FALSE
    else
      return Partitioned Images
    end if
End FUNCTION
```

Algorithm 1

```
FUNCTION Task_Partitioning(InL, InR, T)
    /*C_n is the number of cores*/
    Check the number of cores
    Create threads for detecting objects as C_n
    for each object t_i, i ⟵ 1 to n, t_i ∈ T do
      Create Task
        Parallel execution with Detection_Obj(InL, t_i)
      Create Task
        Parallel execution with Detection_Obj(InR, t_i)
End FUNCTION
```

Algorithm 2

$$\sum_{i=1}^{n} (Tc_i + Tg_i), \tag{2}$$

where Tc and Tg denote the time required for processing one frame using CPU and GPU, respectively. i denotes the frame sequence of the inputted images.

To minimize the GPU wait time of two jobs with a defined processing sequence, the pipeline method is simultaneously applied to the CPU and GPU tasks after allocating preprocessing to the CPU and object detection to the GPU. When sonar images are inputted, the preprocessing task is processed in parallel, based on the CPU, and the image outputted after preprocessing is uploaded to

the GPU memory and processed based on the GPU. When the upload to the GPU memory is complete, the CPU receives the next sonar image and performs preprocessing. Simultaneously, the GPU performs object detection for the first sonar image. Figure 3 illustrates this method, applying a pipeline between CPU and GPU. In the figure, the Waiting Queue is a FIFO (first-in-first-out) data structure and is employed by GPU to independently use image data processed by CPU. Therefore, it can maximize the performance improvement via parallelization by effectively reducing the GPU wait time, if simultaneous processing is carried out by the pipeline between CPU and GPU for all sonar images, for which object detection must be performed. The processing time required for object detection of a sonar image when this pipeline method is applied can be expressed as follows:

$$Tc_1 + \sum_{i=2}^{n} Tc_i + Tg_n. \tag{3}$$

The GPU waits only during preprocessing of the first inputted frame because there is no previous frame. Then, the CPU and GPU perform preprocessing and object detection simultaneously, and only the GPU processing time for the object detection of the last frame is added. Furthermore, the dormant time of processors is minimized by the proposed pipeline method. If the pipeline method is not applied, the total wait time of processors is $\sum_{i=0}^{n} Tc_i$. However, if the pipeline method is applied, it decreases to $\sum_{i=0}^{n-1} |Tc_{i+1} - Tg_i|$.

4. Implementation and Evaluation

The optimized SIAS parallel processing performance improvement method proposed in this study is evaluated by experimentally comparing it with the existing sequential processing methods and the parallel processing based on [7]. For this evaluation, a simulation system for sonar image analysis comprising multicore and many-core CPUs is constructed by applying the optimized parallel processing method for the SIAS developed by our research team.

4.1. Implementation. For the preprocessing step, we developed a parallel version of the wavelet method [20] using OpenMP APIs, and it was applied to the Task Partitioning method. Figure 4 illustrates the implementation of CPU-based parallel processing using the Task Partitioning method for the preprocessing step. In the image segmentation phase, two tasks for the left and right images were divided using the *#prama omp parallel* and *#prama omp section* directives with OpenMP. The *parallel* directive enables the operation of specified code sections in parallel, and the *section* directive allocates the code sections to be performed by each task. Then, the *nowait* clause is set in the section directive so that when a task is completed, the section finishes without waiting for the completion of other branched threads. The thread allocation for each tile in the partitioned task and the phases of wavelet method, such as Wavelet Transform, Ridgelet Transform, and Thresholding, were implemented by *#prama omp parallel for* directive. The *parallel for* directive makes the *for loop* perform the processing of each divided tile in parallel with a specified number of threads. For the specification of the number of threads to be operated in parallel, the *num_threads* clause was used in accordance with the number of logical threads of the CPU, as explained in Section 3. To enable the simultaneous execution of the instructions for eight float data types, the SIMD process for each divided tile was implemented using the *#pragma omp for simd* directive.

We employed the FAST method [21] and p-SIFT method [22] for the object detection step and implemented a pattern matching and a tracking feature-point technique to classify the detected objects. Figure 5 depicts the parallel processing phases based on GPU threads. To implement the object recognition, the tasks to be performed for each element to be parallelized by SIMT are divided into threads, per the number of total elements. The GPU processes the threads for the number of GPU cores simultaneously among the total allocated threads. We implemented the GPU-based parallelization, including SIMT, for the FAST method and p-SIFT method using CUDA APIs.

For feature-point extraction during object detection, threads for the number of pixels are allocated, and the feature amount detection and matching are divided into threads according to the number of feature points. In this case, the number of pixels and the number of feature points when a 4K-class image is inputted are larger than the number of GPU cores in the system. Thus, the usage efficiency reduction in the dormant GPU during parallelization is minimized. Data parallelization is impossible, and sequential sections for GPU thread allocation and kernel activation processed in the CPU are minimized via the use of an algorithm optimized to SIMT GPU data parallelization, having almost no effect on the processing speed of the preprocessing algorithm occupying the CPU. The object classification step is performed immediately after the coordinates and feature quantities of the detected candidate objects are identified. In addition, we implemented a template matching method in which multiple parallel threads are used in consideration of candidate points of objects to be compared to identify multiple objects simultaneously.

Figure 6 shows the process of the optimized parallelization implementation for SIAS. The CPU-based Task Partitioning method for the preprocessing step and the GPU parallel processing are applied. As shown in the figure, the pipeline method was implemented in which the object detection step shares the images generated by the preprocessing step using the Waiting Queue. We also used the *#pragma omp task* directive of OpenMP to enable the two tasks to operate independently for the pipeline method and activate the nested parallelism by using the *omp_set_nested* command to enable additional thread allocation within the task for image preprocessing.

The SIAS system uses Intel i7 quadcore CPUs, 16 GB RAM, and a Nvidia GEFORCE GTX graphic card for GPU. It operates on Windows 10 OS. Furthermore, the installed CPU operates in eight logical threads with four physical cores, and the GPU has 768 cores. The SIAS software was implemented using C++ and CUDA languages. OpenMP 5.0 was used for the CPU parallelization API, and CUDA

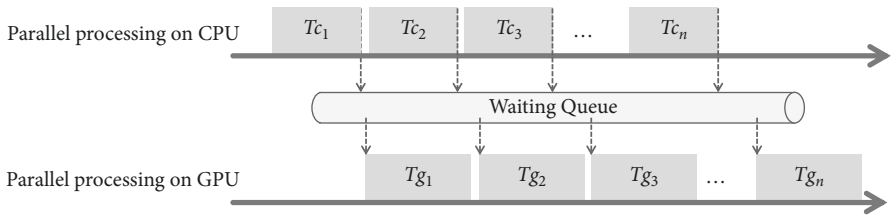


Figure 3: Process of the pipeline approach.

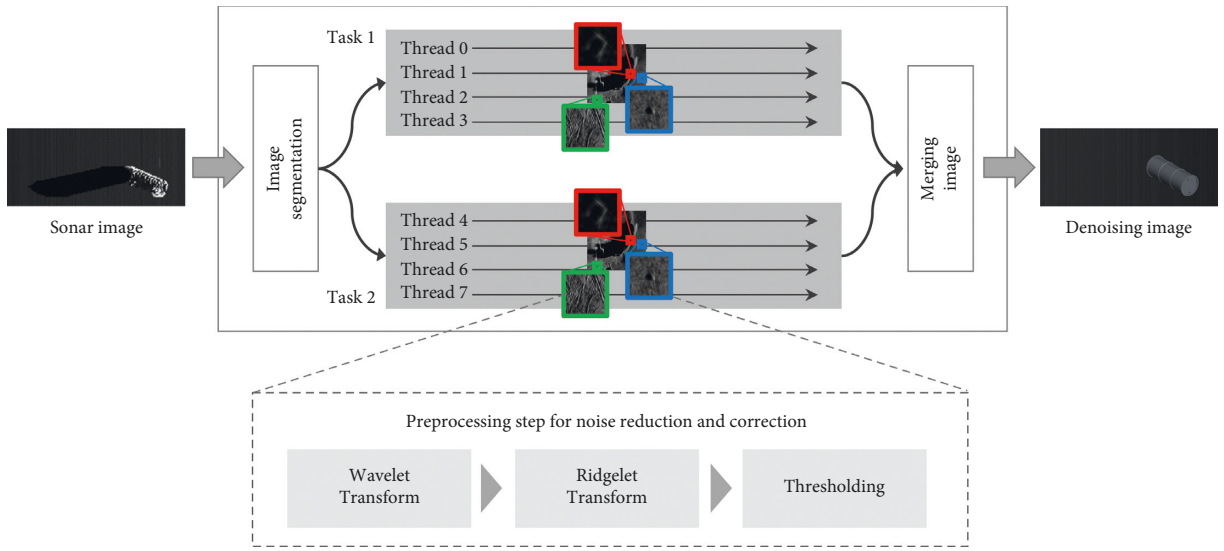


Figure 4: CPU-based parallel processing using the Task Partitioning method.

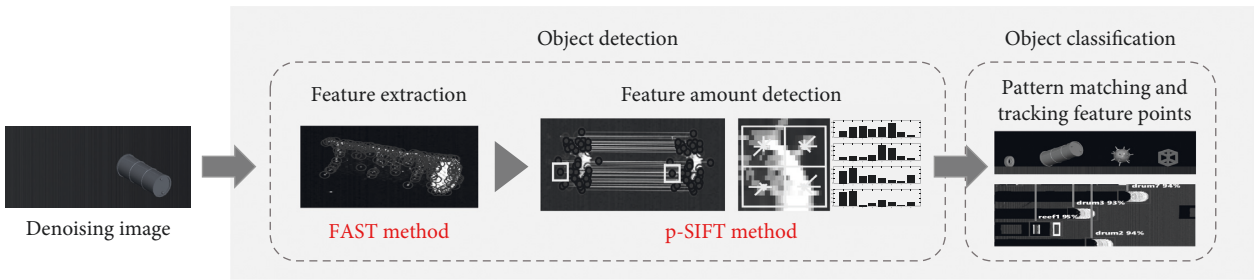


Figure 5: GPU-based parallel processing for object detection and object classification steps.

Toolkit 7.5 was used for the GPU parallelization. Finally, the programs for the SIAS were compiled using the LLVM 5.0 compiler [23].

4.2. Experiment and Analysis. The test images were side-scan sonar images with 4096×4096 resolution, created using Sonar Image Creator so that 20 types of objects, consisting of five units each of four types (i.e., mine, fishing reef, tire, and drum), could be inserted. We verified the accuracy of the new SIAS with parallel algorithms and measured the total time for recognizing 20 multiple targets in the inputted image.

To verify the accuracy of new SIAS, we compared to the original images and the prior version of SIAS which uses sequential algorithms. Figure 7 shows the result of object recognition with the SIAS. Figure 7(a) is a synthetic image created by the Sonar Image Creator of the SIAS, and Figure 7(b) is the results of identifying multiple underwater objects in Figure 7(a) through the SIAS with parallel processing. As resulted in Figure 7(b), the SIAS correctly identifies multiple underwater objects even if the parallel processing, including Task Partitioning and pipeline methods, is applied. From the figure, we see that the SIAS using parallel algorithms provides the accuracy of recognition because it identifies 20 multiobjects on the seabed with recognition rates of 80% or more.

To examine the parallelization performance improvement of the developed SIAS, the processing speeds were measured and experimentally compared with sequential

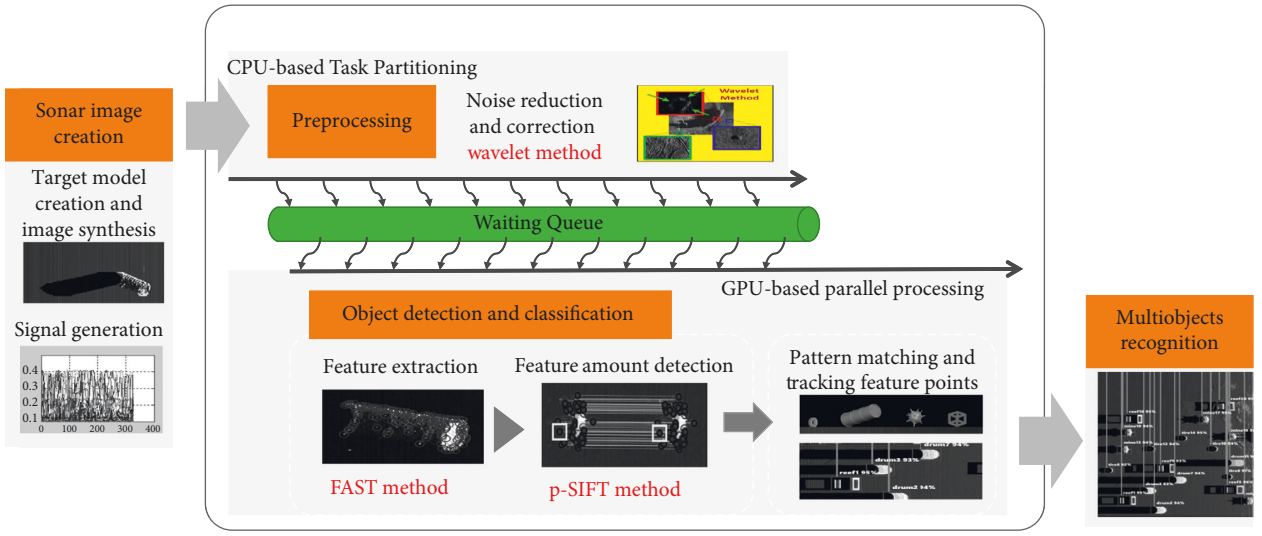
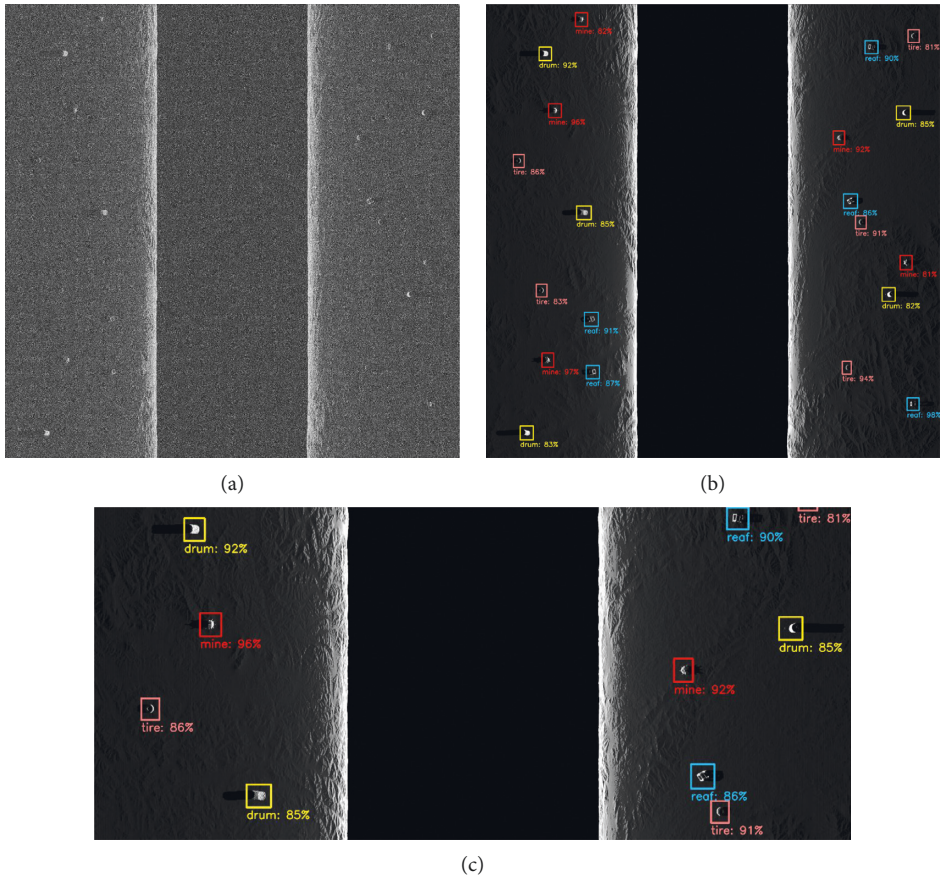


Figure 6: Optimized parallel processing with the pipeline method for SIAS.



(a) (b)

(c)

Figure 7: The results of multiple objects recognition under the parallel version of SIAS. (a) Original image generated by Sonar Image Creator of SIAS. (b) The recognized results by the parallel version of SIAS. (c) The recognized objects by the parallel version of SIAS.

processing (SQ) using no parallelization, CPU-based general parallel processing (PP), and Task Partitioning and GPU parallel methods (TG). The pipeline method was applied to Task Partitioning and GPU parallelization (TP).

The experiment results are summarized in Table 1 and Figure 8. The simple combination of Task Partitioning and GPU parallelization (TG) achieved a speedup of approximately 92.1 times, compared to sequential processing.

Table 1: Experimental results of SIAS under parallel processing methods.

	SQ (ms)	PP (ms)	TG (ms)	TP (ms)
Preprocessing	6,310	1,832	90	90
Object detection	12,100	109	107	109
Object classification	9	3	3	3
Total	18,419	1944	200	112

SQ: sequential process; PP: parallel process on CPU; TG: Task Partitioning on CPU + parallel processing on GPU; TP: Task Partitioning on CPU + parallel processing on GPU with the pipeline method.

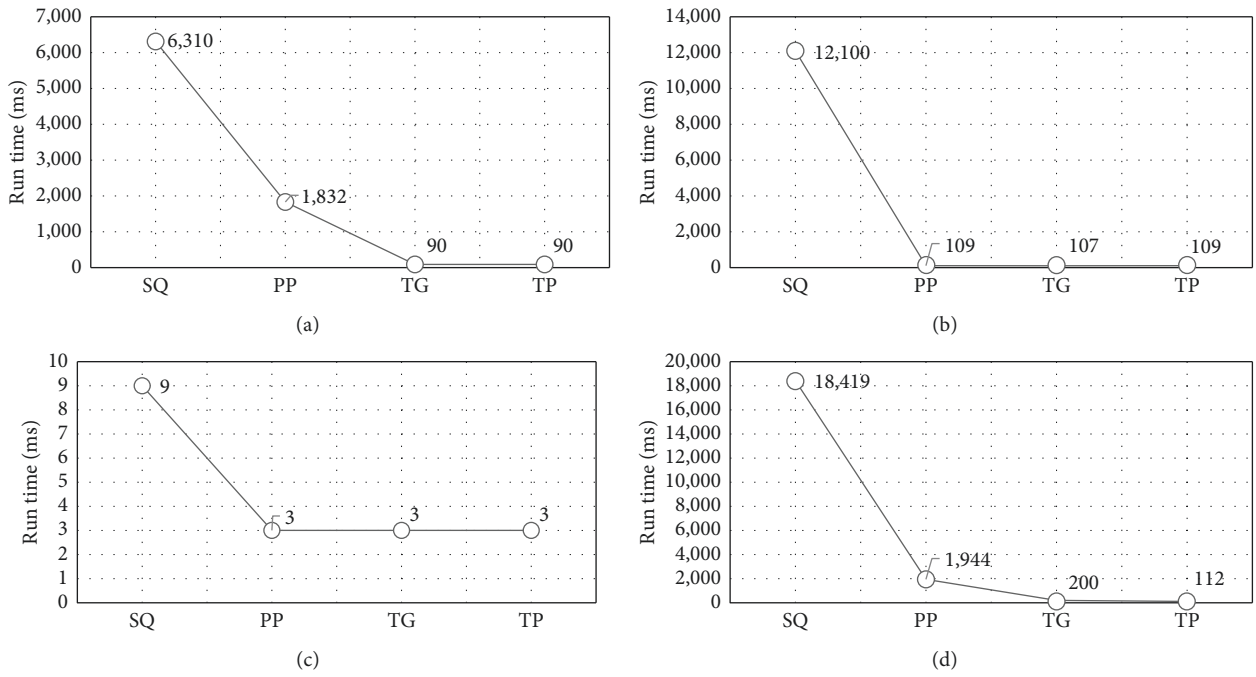


Figure 8: Measured run-time results of SIAS under parallel processing methods. (a) Preprocessing; (b) object detection; (c) object classification; (d) total run time for SIAS.

However, the pipeline application of Task Partitioning and GPU parallelization (TP) achieved a speedup of 164.5 times, compared to sequential processing (SQ). Furthermore, when the optimized parallel method was applied, TG and TP achieved speedups of 9.7 times and 17.4 times, respectively, compared to the parallel CPU processing (PP). When the parallelization effects were analyzed by steps, TG and TP showed the same results during the preprocessing step and improved by 70.1 times compared to SQ. It also improved 20.3 times compared to PP. From the results of the preprocessing step shown in Figure 8, we see that the Task Partitioning method is practical for sonar image processing because it dramatically reduces an average execution time to 5% and 1% that of PP and SQ, respectively.

In the object detection step, PP and TP showed 110 times faster processing compared to SQ, and TG improved by 113 times compared to SQ, showing the most effective results. Finally, during the object classification step, PP, TG, and TP all showed three times higher results than SQ. In the TG and TP results of Table 1, TP has a slightly time delay, about 2 ms, compared to TG in GPU parallelization for the pipeline method. Nevertheless, it confirms that SIAS is optimized by the pipelined approach because TP reduces the overall execution time of TG by about 90 ms through minimizing CPU and GPU latencies.

The test results from Table 1 and Figure 8 confirm that the CPU-based Task Partitioning method significantly improves the processing speed of parallelization and practical for sonar image processing because the Task Partitioning method reduces the average execution time to 1% and 5% that of the sequential processing method and the general parallelization, respectively. Furthermore, because of the application of GPU parallelization and the pipeline method, the total processing time approached 100 ms. Thus, it can be used for sonar imaging-based seabed monitoring systems by providing real-time performance.

5. Conclusion

Sonar technologies, which have relatively low effects on the underwater environment, have been widely used to secure the safety of vessels by identifying underwater hazards via real-time monitoring and surveillance of seabed environments. However, owing to the nature of the underwater

environment, the collected sonar signals contain large amounts of noise. Thus, a significant time delay occurs because of the process of removing noise to obtain calibrated underwater information. In this study, the time delay of underwater environment monitoring via sonar imaging analysis was significantly improved by applying the Task Partitioning method, a new parallelization method based on CPU- and GPU-based parallel processing methods. The CPU and GPU are used simultaneously through a pipeline. When the proposed method was experimentally compared to the existing methods using SIAS, a simulator system for identifying underwater objects based on sonar imaging, the proposed method showed an improvement of 164.5 times compared to sequential processing and 17.4 times compared to general parallelization in terms of the time consumed for identifying multiple objects in 4K images having 4096×4096 resolution. During image preprocessing, which removes unnecessary noises, and during calibration, CPU-based Task Partitioning drastically improved parallel processing speed by effectively supporting SIMD and multithreading. The empirical results showed that our method is practical for sonar image processing, reducing the average execution time to 1% and 5% of the sequential processing method and general parallel processing method, respectively. Finally, this improvement of optimized parallelization is significant because it can be used for the sonar imaging-based underwater real-time monitoring system and can be applied to ensure the safety of ships and submersibles from various hazards in the water.

Data Availability

The data source, including figures and table, used in this paper are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the Agency for Defense Development, Republic of Korea, under Grant UD160014DD.

References

[1] S. Reed, Y. Petillot, and J. Bell, "An automatic approach to the detection and extraction of mine features in sidescan sonar," *IEEE Journal of Oceanic Engineering*, vol. 28, no. 1, pp. 90–105, 2003.

[2] B. Philippe, *The Handbook of Sidescan Sonar*, Springer, Berlin, Germany, 2009.

[3] H. Medwin and C. S. Clay, *Fundamentals of Acoustical Oceanography*, Academic Press, Cambridge, MA, USA, 1997.

[4] R. E. Hansen, "Introduction to synthetic aperture sonar," in *Sonar Systems*, N. Kolev, Ed., pp. 3–28, InTech, Bolton, UK, 2011.

[5] G. J. Orris, B. E. McDonald, and W. A. Kuperman, "Matched phase noise reduction," *Journal of the Acoustical Society of America*, vol. 96, no. 6, pp. 3499–3503, 1994.

[6] A. Jarrot, C. Ioana, and A. Quinquis, "Denoising underwater signals propagating through multi-path channels," in *Proceedings of Oceans 2005 Europe*, vol. 1, pp. 501–506, Washington, DC, USA, September 2005.

[7] J. Kim, E. Park, X. Cui, H. Kim, and W. A. Gruver, "A fast feature extraction in object recognition using parallel processing on CPU and GPU," in *Proceedings IEEE International Conference on Systems, Man and Cybernetics*, pp. 3842–3847, San Antonio, TX, USA, October 2009.

[8] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[9] M. S. Rasmussen, M. B. Stuart, and S. Karlsson, "Parallelism and scalability in an image processing application," *International Journal of Parallel Programming*, vol. 37, no. 3, pp. 306–323, 2009.

[10] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 91–104, 2011.

[11] J. Kong, M. Dimitrov, Y. Yang et al., "Accelerating MATLAB image processing toolbox functions on GPUs," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pp. 75–85, ACM, Pittsburgh, PA, USA, March 2010.

[12] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the 7th International Conference on Computer Vision*, vol. 2, pp. 1150–01157, IEEE, York, UK, July 1999.

[13] R. Chandra, L. Dagum, D. Kohr, D. Maydan, R. Menon, and J. McDonald, *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers Inc., Burlington, MA, USA, 2001.

[14] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., Burlington, MA, USA, 2013.

[15] S. T. Thakkar and T. Huff, "Internet streaming SIMD extensions," *Computer*, vol. 32, no. 12, pp. 26–34, 1999.

[16] Nvidia, *CUDA C Programming Guide*, Nvidia, Santa Clara, CA, USA, 2018, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[17] M. C. Park, O. K. Ha, S. W. Ha, and Y. K. Jun, "Real-time 3D simulation for the trawl fishing gear based on parallel processing of sonar sensor data," *International Journal of Distributed Sensor Networks*, vol. 10, no. 7, 2014.

[18] E. Coiras, A. Ramirez-Montesinos, and J. Groen, "GPU-based simulation of side-looking sonar images," in *Proceedings of Oceans 2009-Europe*, pp. 1–6, Bremen, Germany, May 2009.

[19] C. Nvidia, "Nvidia's next generation CUDA compute architecture: Fermi," *Computer Systems*, vol. 26, pp. 63–72, 2009.

[20] C. K. Chui, *An Introduction to Wavelets*, Elsevier, New York, NY, USA, 2016.

[21] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proceedings of European Conference on Computer Vision*, pp. 430–443, Springer, Graz, Austria, May 2006.

[22] L. Seidenari, G. Serra, A. D. Bagdanov, and A. Del Bimbo, "Local pyramidal descriptors for image recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 1033–1040, 2014.

[23] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO 2004)*, pp. 75–86, San Jose, CA, USA, March 2004.