KMI International Journal of Maritime Affairs and Fisheries • Volume 12 Issue 2 December 2020 pp. 001-020 •

Application of Parallel Processing Techniques to Satellite Ocean Color Data Processing

Jae-Moo Heo* · Hyun Yang** · Young-Je Park*** · Hee-Jeong Han****

ABSTRACT

Recent advances demand that remote-sensing satellites efficiently process massive amounts of ocean color data. This paper compares the open multi-processing (OpenMP), the open computing language (OpenCL), the Message Passing Interface (MPI), the hybrid MPI/OpenMP, and the hybrid MPI/OpenCL in the parallel implementation of ocean color processing algorithms using data from the Geostationary Ocean Color Imager (GOCI), which is the first ocean color remote sensor operated in geostationary orbit. Since 2010, GOCI has observed ocean color around the Korean Peninsula and has generated hundreds of terabytes of big data. When any of the data-processing algorithms are updated, all preexisting data is required to be reprocessed, which can take hundreds of days because GOCI data are currently processed sequentially. Therefore, we attempted to develop an efficient parallel processing methodology for GOCI data. We tested well-known GOCI data-processing algorithms, like the chlorophyll (CHL) and total suspended solid (TSS) concentration estimation algorithms, using a cluster system. This cluster uses the Red Hat Linux operating system with two Intel Xeon 8-core processors (CPU), an AMD Radeon HD 7970 (GPU), and InfiniBand 4x QDR (network). As a result of this study we were able to improve the GOCI ocean color algorithms' processing speeds for OpenMP, OpenCL, MPI, hybrid MPI/OpenMP, and hybrid MPI/OpenCL by 3.92, 2.56, 2.51 3.27, and 2.05 times, respectively, than that of when we run the data sequentially. Moreover, we confirmed that the OpenMP programming model is the most useful for real-time processing GOCI data, which involves large amounts of input data and relatively simple formulas. Also, the vast number of computational nodes helps reduce the time taken to reprocess all data.

Key words: OpenMP, OpenCL, MPI, GOCI, parallel programming, ocean color data processing, satellite, big data

* First author, Researcher at National Institute of Environmental Research, Korea, jaemoo@korea.kr
** Senior Research Scientist at Korea Ocean Satellite Center, Korea Institute of Ocean Science and Technology, Korea, yanghyun@kiost.ac.kr
*** Principal Research Scientist at Korea Ocean Satellite Center, Korea Institute of Ocean Science and Technology, Korea, youngjepark@kiost.ac.kr
**** Corresponding author, Principal research specialist at Korea Ocean Satellite Center, Korea Institute of Ocean Science and Technology, Korea, han77@kiost.ac.kr

Received 30 Oct. 2020; Received in revised form 21 Nov. 2020; Accepted 20 Dec. 2020

1. Introduction

The amount of data generated increases with the development of spatial information acquisition technology, while data analyses are becoming more complicated with the increasing demand for high-quality spatial information. The performance of spatial information processing and parallel processing constitutes an essential solution to this problem.

In almost all data processing systems, execution time is one of the most important factors to be considered. Several algorithms and techniques to speed up processing time have been studied in many fields (Chen and Chen, 2011; Ehsan and Malehmir, 2012; Molnar et al., 2020). Especially in data processing, the two most important factors to be considered to speed up data processing are the processor's performance and the algorithm that can efficiently process data (Chen and Chen, 2011). Multi-core processor is commonly used recently, and it is easy to parallelize the process of data processing algorithm because of repeatedly using the same command for the sequential structure. Therefore, it is possible to take advantage of multi-core processor to parallelize the data processing algorithm.

The Geostationary Ocean Color Imager (GOCI) is mounted on a Communication Ocean and Meteorological Satellite (COMS) that was launched from French Guiana in June 2010. It is the world's first geostationary ocean color satellite and operates at an altitude of 35,789 km. Its primary purpose is monitoring the marine environment around the Korean peninsula (Ryu et al., 2012). The GOCI provides ocean color data with a spatial resolution of 500 m at hourly intervals up to 8 times a day, allowing observations of short-term changes in the Northeast Asian region. The GOCI Data-Processing System (GDPS) (Ryu et al., 2012), which provides specialized data-processing software for GOCI, was developed for real-time generation of various products.

The Korea Ocean Satellite Center (KOSC) of the Korea Institute of Ocean Science and Technology (KIOST), which manages GOCI, plays a role in the reception, processing, storage, and distribution of GOCI ocean color data (Yang et al., 2014). Since 2010, hundreds of terabytes of ocean color big data have been acquired from around the Korean Peninsula. When a data-processing algorithm is updated, all the data needs to be reprocessed by the updated algorithm. This takes hundreds of days because GOCI data are currently processed in sequence. Therefore, introducing a parallel processing system for efficient processing and reprocessing of satellite ocean color data is required.

This paper evaluated the most suitable parallel processing methodology for GOCI data. An analysis of parallel processing techniques proposes an efficient GOCI data-processing approach and substantially reduces GOCI data processing execution time. Open multi-processing (OpenMP) is one of the best-known parallelism methods. This technique takes advantage of multi-core

processors with a shared memory system and enables parallel processes in C/C++ and FORTRAN. Sequential process can be parallelized through the "#pragma omp" pre-processing directive (Shen and Zhang, 2013). Open computing language (OpenCL) and compute unified device architecture (CUDA) are recently proposed methods. These methods use graphic processing unit (GPU) cores as the processing elements (PE) in an Advanced Micro Devices (AMD) GPU with tens to hundreds of independent floating point unit (FPU) and arithmetic logic unit (ALU) coprocessors. They are applied in medical image processing (Chen and Chen, 2011), geology (Ehsan and Malehmir, 2012), and meteorology (Molnar et al., 2010). The most popular GPU programming model is CUDA, a parallel programming system using GPUs manufactured by NVIDIA. However, CUDA[1] was not designed for a heterogeneous system made with components from multiple vendors. In comparison, the OpenCL[2] programming model, a cross-platform developed by the Khronos Group (Beaverton, OR, USA), enables parallel processing in a heterogeneous system. The Message Passing Interface (MPI), a message-passing model for distributed memory systems, is the de facto standard for vendors, developers, and user committees and was developed to support parallel programming libraries. MPI[3] defines the functions required for communication between the nodes used by most clusters. Each node accesses solely its local memory and nodes are connected to other nodes via the interconnection network. Recently, multi-core CPUs with from two to as many as eight cores have become predominant. A method that uses a hybrid MPI/OpenMP model has become common in multi-core CPU-equipped clusters and has been the subject of several studies (Gorobets, Trias and Oliva, 2013; Wan and Liu, 2013).

This paper describes the parallelization methods used for GOCI data processing with the parallel programming models OpenMP, OpenCL, MPI, Hybrid MPI/OpenMP, and Hybrid MPI/OpenCL. It analyzes and compares the performance of two GOCI ocean color algorithms using parallelization technique. Section 2 introduces the GOCI ocean color algorithms used to analyze the parallel processing performance, and Section 3 presents algorithm I/O data and describes the algorithms' parallelization methods. Section 4 presents the experimental results derived using the proposed methods and analyzes each technique-specific algorithm's performance time and improvement. Finally, we conclude our research in Section 5.

1 NVIDIA CUDA. http://developer.nvidia.com/object/cuda.html/
2 OpenCL. http://www.khronous.org/opencl/
3 OpenMPI. http://www.open-mpi.org/

2. GOCI Data Processing Algorithms

The GOCI acquires ocean color image data at hourly intervals up to 8 times a day, daily from 09:15 until 16:45, to monitor the marine environment in near real-time. The GOCI analyzes eight bands centered on 412, 443, 490, 555, 660, and 680 nm in the visible light range and 745 and 865 nm in the near-infrared range (Ryu et al., 2012). Figure 1(a) shows a color image made by combining the red, green, and blue images centered on 680, 555, and 412 nm, respectively. At 130°E, 36°N, it has a spatial resolution (GSD) of less than 500 m. The GOCI observation area of 2,500 × 2,500 km covers Korea, Japan, the eastern coast of China, and parts of Taiwan's northern coast.

The raw data obtained from GOCI is digitized via 12-bit depth quantization. The digital data are converted into two-dimensional radiance data consisting of 31,648,395 pixels (5567 × 5685 pixels) through geometric correction (Han et al., 2010). Since this is two-dimensional radiance data, the data for each of the eight bands constitute one set. This set is called level-1 data, and Figure 1(b) shows an example.

In general, more radiation is detected from areas covered by land and cloud, while less is detected from sea areas (Ahn et al., 2012). In processing the GOCI data, the radiance data of each band included in level-1 are used as input data for processing algorithms such as the chlorophyll concentration (CHL) and total suspended solid concentration (TSS) estimation algorithms after the atmospheric correction (Ahn et al., 2012) thus these are called level-2 data. In addition, combinations of level-2 data obtained eight times a day are used to produce level-3 data, such as the ocean primary production (PP) and water current vectors (Yang et al., 2014). We choose the algorithms which are well-known in the ocean remote sensing community and used in the GOCI data processing system. These data are essential for monitoring ocean environments.

In this paper, we tested the chlorophyll concentration (CHL) and total suspended solid concentration (TSS) estimation algorithms, using the ocean chlorophyll 2-band (OC2) and the Yellow Sea Large Marine Ecosystem (YSLME) schemes (Siswanto et al., 2011). The CHL algorithm calculates the total amount of chlorophyll pigment contained in the plankton present in seawater. To compare the computational complexity and difference in the amount of input data between algorithms, the OC2 algorithm, which inputs two remote-sensing reflectances (Rrs) data, was selected as below

$$CHL_{OC2} = e_0 + 10^{e_1+e_2\times R+e_3\times R^2+e_4\times R^3} \quad (1)$$

$$R = log_{10}\left(\frac{R_{rs}(490)}{R_{rs}(555)}\right) \quad (2)$$

where e is the coefficient value, and Rrs(490) and Rrs(555) are Rrs data at 490 nm and 555 nm wavelengths, respectively.

Likewise, the KOSC-TSS algorithm which inputs a single Rrs data and determines the weight of total suspended solid per unit volume of seawater was selected as below

$$TSS = c_0 \times R_{rs}(555)^{c_1} \quad (3)$$

where c is the coefficient value and Figure 2 shows an example of the level-2 data in (a) CHL and (b) TSS data.

Figure 1. (a) A colored image and (b) level-1 radiance data obtained with GOCI on 2 September 2015

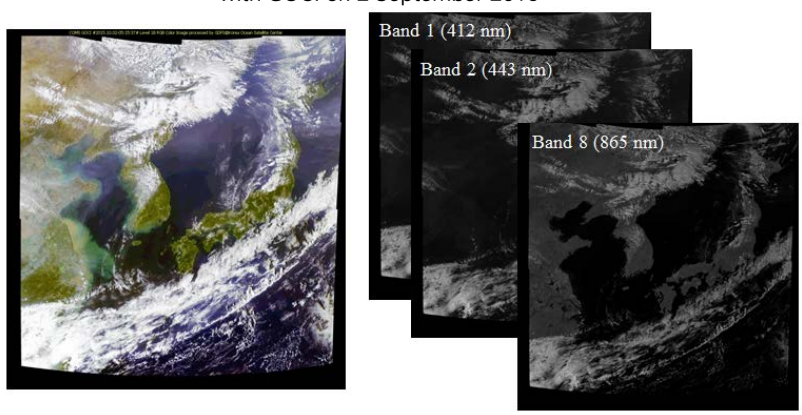
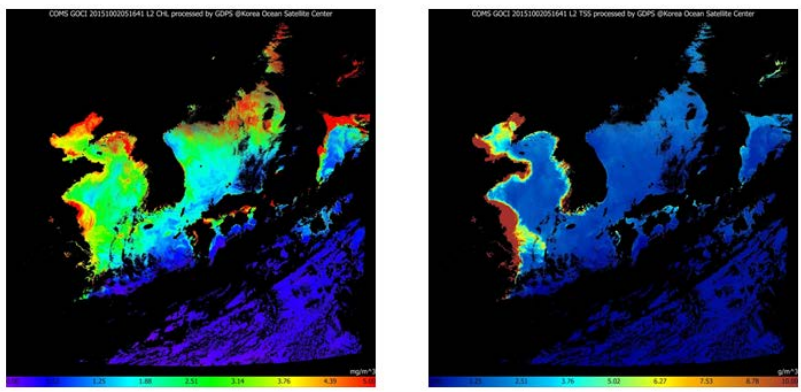


Figure 2. (a) Chlorophyll concentration (CHL) and (b) total suspended solid (TSS) level-2 data obtained with GOCI on 2 September 2015



3. Experimental Results

This paper checked the processing time required to execute each parallel processing model by recording the start and the end times for File I/O, data transmission, and algorithm calculation. We used the "gettimeofday" library function to measure the processing time. The processing time included the data transmission time before calling the "clEnqueueNDRangeKernel" OpenCL API function for the OpenCL model. We also used the "clGetEventProfilingInfo" function to measure the time that a kernel spent in the command queue and included it in the time for actual algorithm calculation. For the MPI model, we defined the transmission time as the total time for data transmission from the master node to the last slave node.

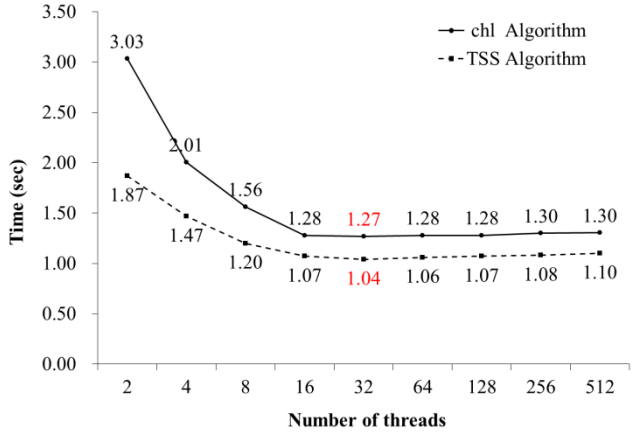
The CHL algorithm uses the Rrs data of Band 3 (443 nm) and Band 4 (555 nm) as input data and the TSS algorithm uses the Rrs data of Band 4 (Siswanto et al., 2011; Min et al., 2013). Each of the binary Rrs files is the same size as the GOCI level-1 data, i.e., about 130 MB (width × height × size of float type or 5567 × 5685×4). The experimental results included the total processing time for each parallel processing model, including the algorithm calculations, file I/O, and data transmission.

The results showing in the figures in section 3 show the total execution time of each parallel processing model in seconds' level. These figures provide the optimal condition information of each parallel processing model for CHL, TSS processing. It is important to find the optimal number of threads, the optimal number of work groups, the optimal number of nodes for OpenMP, OpenCL, MPI, and the hybrid MPI model.

3.1 OpenMP

In the performance of OpenMP model, it is important to find the optimal number of threads. There is a correlation between core and thread (Shen and Zhang, 2013). The ocean color data must be divided so that the data processed by each core do not overlap. In this study, one computing node is equipped with two 8-core processors, for a total of 16 physical cores and 32 logical cores. When the process was divided into 32 threads using OpenMP, the best result was 1.27 seconds for the CHL algorithm and 1.04 seconds for the TSS algorithm (Figure 3). A single core must process approximately 4 MB, i.e., (total amount of GOCI ocean color data) / (number of cores).

Figure 3. Total execution time of the OpenMP model

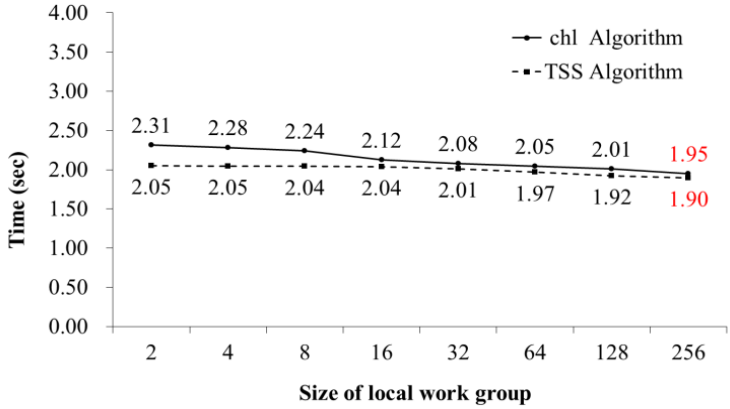


3.2 OpenCL

Using the OpenCL model, it is essential to find the optimal number of work groups based on one local work group's size and the size of one work item. Let's give more work to one work item and reduce the number of total work items. We can reduce the wavefront occupancy because one work item requires a fixed number of registers, while the advantage of reducing the overhead is related to running the kernel (Pennycook et al., 2013). However, the processing time of the GOCI algorithm using the GPU was short. Consequently, it was less useful to change the number of work items. Therefore, we set the number of work items to equal the amount of GOCI data size, i.e., the total number of work items = 5567 × 5685.

We also need to consider the size of the local GPU memory. If one work group has too many work items because the number of work groups is small, not all the data for the work items will go into the local memory at once and occupancy will be reduced. Conversely, if one work group has too few work items, it is impossible to take advantage of the local memory and it generates a local memory area that is not used. Therefore, performance is substantially based on the size of the work groups and local memory (Pennycook et al., 2013). Generally, the size of the registers and local memory to be used by threads is limited, because of the influence of the hardware resources. In our experimental environment, the maximum size for a single work group is 256 in GPU specifications. When we increased the size of the local work group to 256, the total processing time decreased. When there were 256 work groups in the OpenCL model, the best result was 1.95 seconds for the CHL algorithm and 1.90 seconds for the TSS algorithm (Figure 4). The amount of data that a single work group had to process was approximately 0.5 MB, i.e., (the number of total work items) / (the number of work groups).

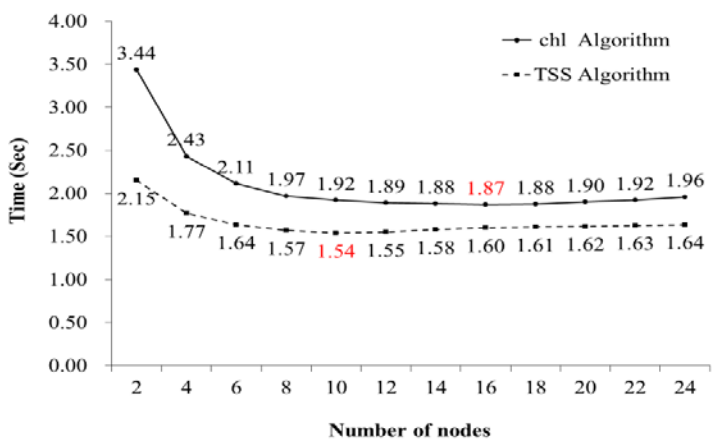
Figure 4. Total execution time of the OpenCL model



3.3 MPI

The ocean color data should be divided so that there is no overlap in the data processed by each node. The amount of data that a single core has to process is (amount of GOCI ocean color data) / (number of nodes). In the MPI cluster, a network bottleneck may arise when the nodes increase (Wolf and Mohr, 2003). For the CHL algorithm, the best result was 1.87 seconds when MPI divided the process into 16 nodes; for the TSS algorithm, the best result was 1.54 seconds when MPI divided the process into 10 nodes (Figure 5). For the two algorithms, each node processed approximately 13 MB and 8 MB, respectively.

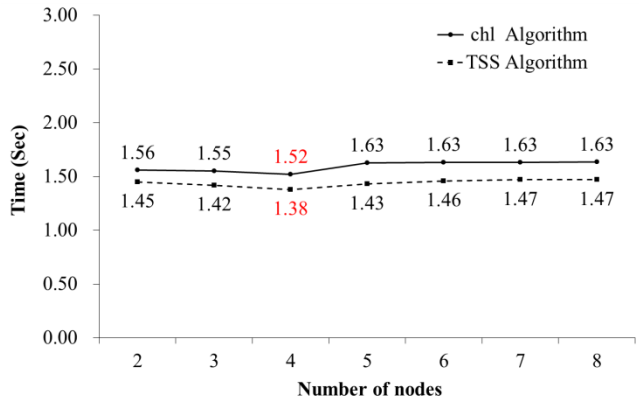
Figure 5. Total execution time of the MPI model



3.4 Hybrid MPI/OpenMP

Like the MPI model, the ocean color data have to be divided so that there is no overlap in the data processed by each node, and the data assigned to each node have to be divided among the core processors. With the OpenMP model, 32 threads were allocated to each node. Experimentally, on adjusting the number of nodes, if there were more than five nodes, a network bottleneck caused by the use of shared memory occurred with MPI and OpenMP (Shen and Zhang, 2013; Wolf and Mohr, 2003). It showed a marginal improvement in the performance compared to few nodes. When the process was divided into 128 (4 nodes through MPI and 32 threads through OpenMP), the best result was 1.97 seconds for the CHL algorithm and 1.57 seconds for the TSS algorithm (Figure 6). A single core had to process approximately 0.1 MB, i.e., (amount of GOCI ocean color data) / (number of cores × number of nodes).

Figure 6. Total execution time of Hybrid MPI/OpenMP model



3.5 Hybrid MPI/OpenCL

Like the MPI model, the ocean color data should be divided so that a GPU processed no overlap in the data processed by each node and the data assigned to each node. Based on the OpenMP and OpenCL models results, each node was assigned 32 threads and 256 work groups. Experimentally, on adjusting the number of nodes, the products were similar to those with the Hybrid MPI/OpenMP model. If there were more than five nodes, the exchange of memory between the CPU and GPU resulted in a network bottleneck caused using memory shared by MPI and OpenCL (Gorobets, Trias and Oliva, 2013; Pennycook et al., 2013). It showed a marginal performance improvement. When the Hybrid MPI/OpenCL model had 256 work groups and four nodes, the best result was 2.43 seconds for the CHL algorithm and 2.35 seconds for the TSS

algorithm (Figure 7). The amount of data that a single work group had to process was approximately 0.13 MB, i.e., (total number of work items) / (number of work groups × number of nodes).

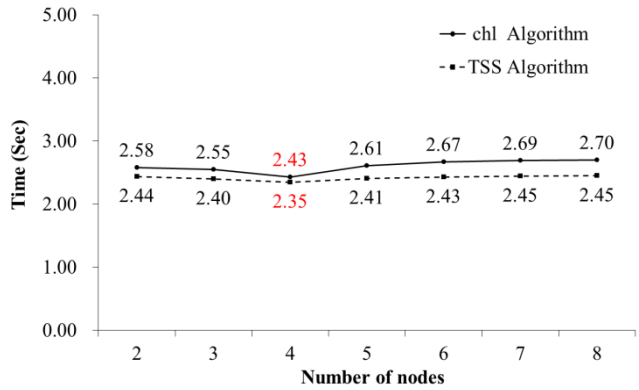


Figure 7. Total execution time of Hybrid MPI/OpenCL model

4. Discussion

When comparing the CHL and TSS algorithms' execution times using parallel processing, both algorithms showed similar patterns for each parallel programming model. The TSS algorithm has only one binary file for input data and consistently took less time than the CHL algorithm, which has two binary files for file I/O and data transmission. However, the two algorithms' differences were not significant because the speed of a network with InfiniBand is fast. Based on the sequential model, the CHL and TSS algorithm's calculation time was 4.05 and 1.81 seconds, respectively (Table 1). The CHL algorithm's execution time, which has greater computational complexity, was approximately 2.24 times that of the TSS algorithm.

We compared the performance of each parallel programming model with sequential processing. OpenMP was carried out with 32 threads, but we did not see the ideal 32-fold speedup in performance because the 32 logical cores are represented by 16 physical cores. The actual speedup was close to a 16-fold increase. With the OpenCL model, the CHL and TSS algorithms' speedup was 45.01 and 22.58 times, respectively (Table 2). There were several reasons for these results. The GPU manufactured by AMD contains 2,048 processing elements (PE) and 64 wavefronts. There were also different specifications for the CPU and GPU cores used in the experiment.

When we measured the time for algorithm calculation separately in the MPI model's distributed memory environment, the performance of the parallelism model is expected to equal the number of nodes. However, the MPI model's performance was 15.89 times that for the CHL algorithm and 9.92 times that for the TSS algorithm when the process involved 16 and 10 nodes, respectively (see Figure 5). The reason for the difference from the ideal is the overhead of calling MPI functions. For the Hybrid MPI/OpenMP and Hybrid MPI/OpenCL models, the two algorithms' performance was approximately four times compared with the OpenMP and OpenCL models, respectively. Therefore, the experiment showed that the parallelism of the algorithm worked successfully in a heterogeneous system.

Note that the speedup of the OpenMP model was the most considerable overall in this study. Nevertheless, the Hybrid MPI/OpenCL model showed a 130-fold speedup in the algorithm calculation time. The OpenMP model improved 3.92 times for the CHL algorithm and 2.56 times for the TSS algorithm. In the Hybrid MPI/OpenCL model, the overhead took about 1.5 seconds for file I/O and data transmission. This process can only be performed sequentially. The time for algorithm calculation was an almost negligible proportion of the entire process. As a result, the Hybrid MPI/OpenCL model's total speedup was small despite the considerable speedup in the algorithm calculation time. Likewise, the overhead was relatively more extensive than the reduced execution time due to the parallelization of algorithm calculation in the OpenCL and MPI models. If there is a complex algorithm to consume more than 1 minute, the Hybrid MPI/OpenCL model is the best parallel processing choice.

Table 1. The performance time of each parallel programming model

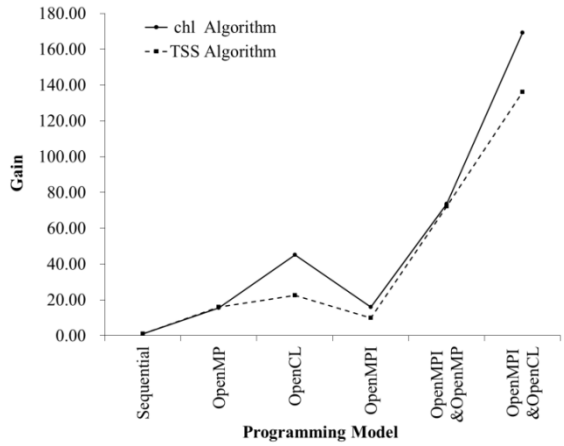
Programming Model		Sequential	OpenMP (32 threads)	OpenCL	MPI	MPI/OpenMP (4 nodes /32 threads)	MPI/OpenCL (4 nodes 32 threads)
File I/O & Data Transmission	CHL	0.94	1.01	1.86	1.61 (16 nodes)	1.47	*2.41*
	TSS	0.86	0.93	1.82	1.36 (10 nodes)	1.36	*2.33*
Algorithm Calculation	CHL	4.05	0.26	0.09	0.26	*0.05*	*0.02*
	TSS	1.81	0.11	0.08	0.18	*0.02*	*0.01*
Total (Sec)	CHL	4.99	*1.27*	1.95	1.87	1.52	2.43
	TSS	2.67	*1.04*	1.90	1.54	1.38	2.35

Table 2. The speedup of each parallel programming model

Programming Model		Sequential	OpenMP (32 threads)	OpenCL	MPI	MPI/OpenMP (4 nodes /32 threads)	MPI/OpenCL (4 nodes /32 threads)
File I/O & Data Transmission	CHL	1.00	0.93	0.50	0.58 (16 nodes)	0.64	0.39
	TSS	1.00	0.93	0.47	0.63 (10 nodes)	0.63	0.37
Algorithm Calculation	CHL	1.00	*15.58*	*45.01*	*15.89*	*73.67*	*169.09*
	TSS	1.00	*15.92*	*22.58*	*9.92*	*72.26*	*136.01*
Total (Sec)	CHL	1.00	*3.92*	2.56	2.67	3.27	2.05
	TSS	1.00	*2.56*	1.40	1.73	1.93	1.14

Figure 8 outlines another aspect of the two algorithms, showing the speedup in algorithm calculation with the OpenCL and Hybrid MPI/OpenCL models. The OpenCL model increased the algorithm calculation times of the CHL and TSS algorithms 45.01 and 22.58 times, respectively. In comparison, the Hybrid MPI/OpenCL model increased this 169.01 times for the CHL algorithm and 136.01 times for the TSS algorithm. The large difference in the two algorithms' results is due to the two algorithms' computational complexity. The GPU core is less advantageous than the CPU core for higher-order calculations. However, neither of the algorithms involves higher-order calculations and the formula of the TSS algorithm is relatively simple. In summary, the TSS algorithm does not take full advantage of the PEs in the GPU compared with the CHL algorithm.

Figure 8. The speedup in algorithm calculation



5. Conclusion

This paper introduced several parallel processing models and compared how parallel processing techniques reduce the processing time of GOCI data, the world's first ocean color remote sensor operated in a geostationary orbit. We examined the performance of the CHL and TSS concentration estimation algorithms. With the OpenMP model, the best improvement was 3.92 times for the CHL algorithm and 2.56 times for the TSS algorithm. The performance of the parallel programming models from the most improved followed in the order of: OpenMP > Hybrid MPI/OpenMP > MPI > OpenCL > Hybrid MPI/OpenCL.

This study examined basic parallelism using each of the parallel processing models. In the future, the amount and allocation of CPU and memory must be analyzed precisely to determine whether it can improve performance with all the parallelism techniques. It is also necessary to optimize and vectorize the algorithms in the OpenMP model, which performed the best. In addition, the results of this study were limited to level-2 data. In future research, it is desirable to establish parallel processing methods for level-1 and level-3 data to reduce the execution time for GOCI data processing markedly.

Recently, GOCI-II (Han et al., 2017, Han et al., 2019), the successor of GOCI, was launched, and has begun servicing data. We expect this study's results to help in real-time processing of GOCI-II data, which have superior spatiotemporal resolution than GOCI data. Furthermore, we believe that the results of this study will contribute to the efficient processing of ocean satellite big data.

Acknowledgments

This research was supported by the "Development of the integrated data processing system for GOCI-II" and the "Technology development for Practical Applications of Multi-Satellite data to maritime issues" funded by the Ministry of Oceans and Fisheries, Korea. This research was also supported by the "Operation in Korea Ocean Satellite Center" funded by the Korea Institute of Ocean Science and Technology (KIOST).

Appendix. Algorithm of Parallel Processing Techniques

This study considers three well-known parallel processing techniques - OpenMP, OpenCL, MPI - and its hybrids to accelerate data processing and reduce satellite data processing time. This Appendix explains the main concepts and advantages of each technique shortly.

1. OpenMP

OpenMP refers to a parallel processing method using a CPU with multiple cores and a shared memory model in a node computer. OpenMP is a multi-threaded application for parallel programming. It provides a simple and easy-to-use API. It is easier to implement and debug than other techniques by using a simple API. We can implement it in a way that divides data and processes loops in parallel. In the OpenMP programming model (Figure A-1), we considered an interrelation between core and thread. We can simultaneously execute the threads as much as the number of cores. Generally, thread number had better generate as multiple of the number of cores. OpenMP can support several computing languages like Fortran, C, C++.

Figure A-1. OpenMP Programming Model

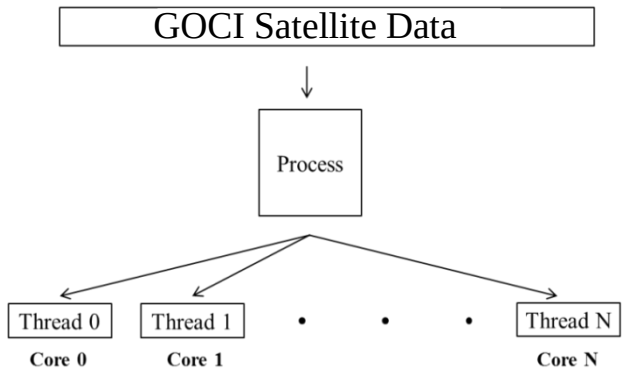
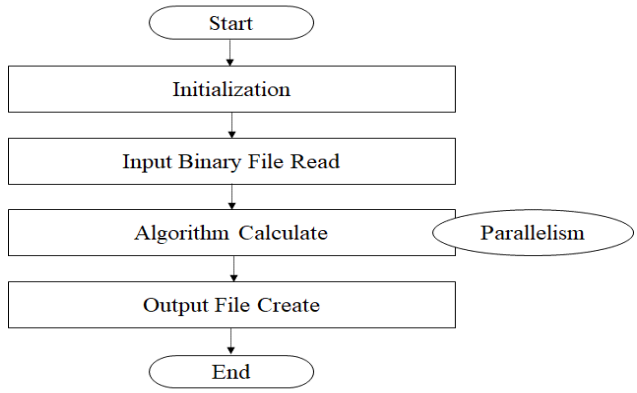


Figure A-2. OpenMP Processing flow chart



2. *OpenCL*

OpenCL is a standard parallel programming application and is a cross-platform model to support both homogeneous systems and heterogeneous systems. We can program it for various devices such as CPU, GPU, and FPGA. In the OpenCL programming model (Figure A-3), we generated two programs, the host program for CPU and the OpenCL program for GPU. We should consider an interrelation between the size of a local work group and run-time. GPU's compute unit is similar in meaning to CPU. A local work group is a task unit inside of 1 computing unit. If we allocate a maximum local work group size not exceeding the local memory size, the algorithm will perform efficiently.

Figure A-3. OpenCL programming model

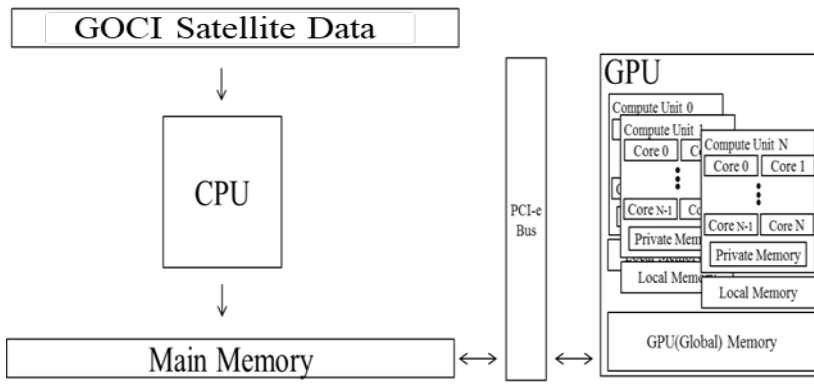
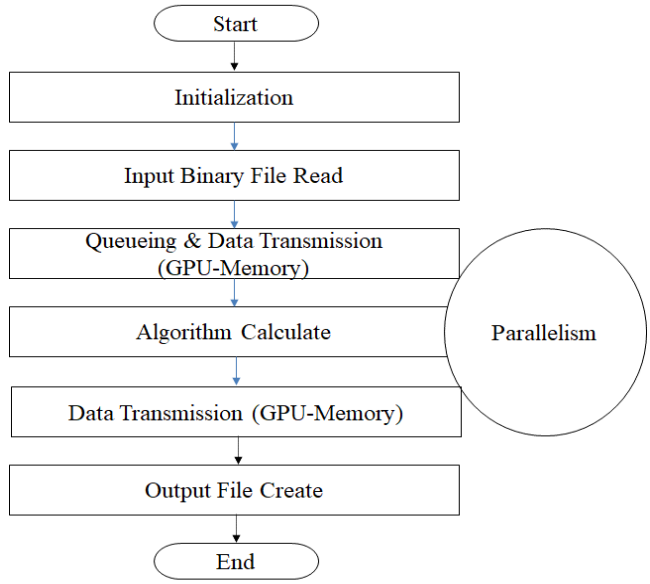


Figure A-4. OpenCL Processing flow chart



3. Message Passing Interface (MPI)

Message Passing Interface (MPI) is a standard that describes the exchange of information in distributed and parallel processing. Making a node that has many CPUs and GPUs is difficult. This is because the mainboard capable of supporting multi CPUs and GPUs is so expensive with a serious heating problem. So, we considered the MPI model on a cluster system that has many nodes. In the MPI programming model (figure A-5), we gave a role to a master node and slave nodes. The master node manages the whole, and slave nodes perform each divided task. Then, we considered an interrelation between the number of nodes and run-time. It has merits and faults. For example, if we increase the node number, each slave node has a small task, and algorithm calculation takes short. But, each slave node has to receive data for processing. So, data transmission takes longer.

Figure A-5. MPI Programming Model

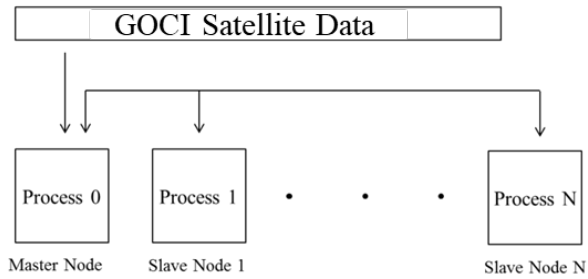
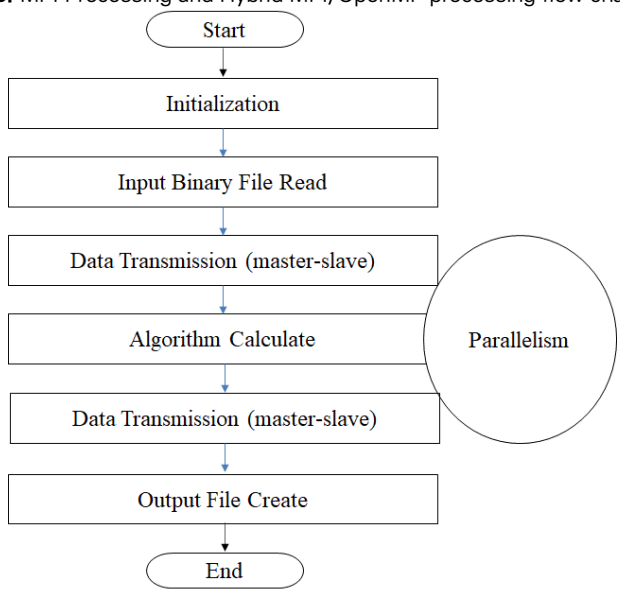


Figure A-6. MPI Processing and Hybrid MPI/OpenMP processing flow chart



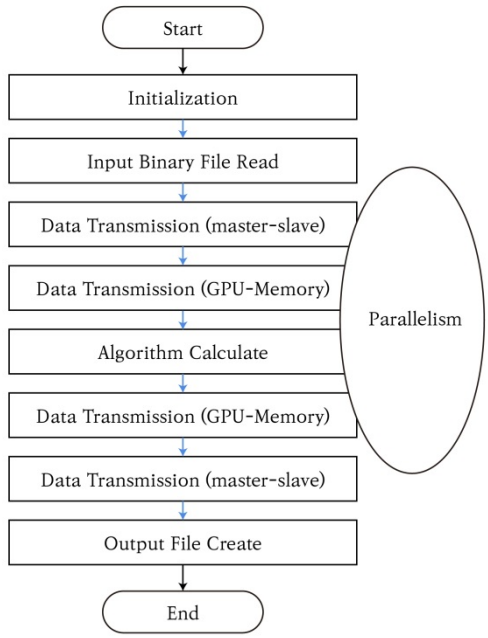
4. *Hybrid MPI/OpenMP model*

The Hybrid MPI/OpenMP model is a combination of MPI and OpenMP. This model's flow chart is the same as the MPI model because the OpenMP has no data transmission part.

5. *Hybrid MPI/OpenCL model*

The Hybrid MPI/OpenCL model is combined with MPI and OpenCL. This model will be decisive to massive data batch processing using a giant cluster.

Figure A-7. Hybrid MPI/OpenCL Processing flow chart



References

Ahn, J.H., Park, Y.J., Ryu, J.H., Lee, B., and Oh, I.S. (2012) Development of Atmospheric Correction Algorithm for Geostationary Ocean Color Imager (GOCI). *Ocean Science Journal* 47:247-259.

Chen, Y. and Chen, Y. (2011) Matching of a Huge Set of MR Images with a Parallel Processing Model. *Journal of Medical Systems* 35(5):795-800.

Ehsan, S.A. and Malehmir, A. (2012) Re-processing and interpretation of 2D seismic data from the Kristineberg mining area, northern Sweden. *Journal of Applied Geophysics* 80:43-55.

Gorobets, A.V., Trias, F.X., and Oliva, A. (2013) A parallel MPI + OpenMP + OpenCL algorithm for hybrid supercomputations of incompressible flows. *Computer and Fluids* 88:764-772.

Han, H.J., Ryu, J.H., and Ahn, Y.H. (2010) Development the Geostationary Ocean Color Imager (GOCI) Data Processing System (GDPS*). Korean Journal of Remote Sensing* 26(2):239-249. (in Korean with English abstract).

Han, H.J., Yang, H., Heo, J.M., and Park, Y.J. (2017) Systemic Ground-Segment Development for the Geostationary Ocean Color Imager II, GOCI-II. *KIISE Transactions on Computing Practices*, 23(3):171-176.

Han, H.J., Yang, H., Heo, J.M., and Park, Y.J. (2019) Systemic Design and Development of the Second Geostationary Ocean Color Satellite Ground Segment. *KIISE Transactions on Computing Practices*, 25(10):477-484.

Min, J.E., Choi, J.K., Park, Y.J., and Ryu, J.H. (2013) Retrieval of suspended sediment concentration in the coastal waters of yellow sea from Geostationary Ocean Color Imager (GOCI). In *Proceedings of International Symposium of Remote Sensing* (Tokyo, Japan, May 15-17, 2013), Korean Society of Remote Sensing, pp.809-812.

Molnar, F., Szakaly, T., Meszaros, R., and Lagzi, L. (2010) Air pollution modelling using a Graphics Processing Unit with CUDA. *Computer Physics Communications* 181(1):105-112.

Pennycook, S.J., Hammond, S.D., Wright, S.A., Herdman J.A., Miller, I., and Jarvis, S.A. (2013) An investigation of the performance portability of OpenCL. *Journal of Parallel and Distributed Computing* 73:1439–1450.

Ryu, J.H., Han, H.J., Cho, S., Park, Y.J., and Ahn, Y.H. (2012) Overview of geostationary ocean color imager (GOCI) and GOCI data processing system (GDPS). *Ocean Science Journal* 47:223-233.

Shen, H. and Zhang, Y. (2013) Comparison and Analysis of Parallel Computing Performance Using OpenMP and MPI. *The Open Automation and Control Systems Journal* 5:38-44.

Siswanto, E., Tang, J., Yamaguchi, H., Ahn, Y.-H., Ishizaka, J., Yoo, S., Kim, S.-W., Kiyomoto, Y., Yamada, K., Chiang, C., and Kawamura, H. (2011) Empirical

ocean-color algorithms to retrieve chlorophyll-a, total suspended matter, and colored dissolved organic matter absorption coefficient in the Yeloow and East China Seas. *Journal of Oceanography* 67(5):627-650.

Wan, J. and Liu, Y. (2013) Hybrid MPI-OpenMP Parallelization of Image Reconstruction. *Journal of Software* 8(3):687-693.

Wolf, F. and Mohr, B. (2003) Automatic performance analysis of hybrid MPI/OpenMP applications. *Journal of System Architecture* 49:421–439.

Yang, H., Choi, J.K., Park, Y.J, Han, H.J., and Ryu, J.H. (2014) Application of the Geostationary Ocean Color Imager (GOCI) to estimates of ocean surface currents. *Journal of Geophysical Research: Oceans* 119(6):3988-4000.

Yang, H., Oh, E., Han, T.H., Han. H.J., and Choi. J.K. (2014) An Efficient Data Processing Method to Improve the Geostationary Ocean Color Imager (GOCI) Data Service. *Korean Journal of Remote Sensing* 30(1):137-147. (in Korean with English abstract).