applied sciences

Article

An Asynchronous Parallel I/O Framework for Mass Conservation Ocean Model

Renbo Pang [1,2], Fujiang Yu [1,2,*], Yu Zhang [1,2,*] and Ye Yuan [1,2]

[1] National Marine Environmental Forecasting Center, 8 Dahuisi Road, Beijing 100080, China; pangrb@nmefc.cn (R.P.); yuanye@nmefc.cn (Y.Y.)
[2] Key Laboratory of Research on Marine Hazards Forecasting, Ministry of Natural Resources of China, 8 Dahuisi Road, Beijing 100080, China
* Correspondence: yvfujiang_2022@163.com (F.Y.); zhangy@nmefc.cn (Y.Z.)

Abstract: I/O is often a performance bottleneck in global ocean circulation models with fine spatial resolution. In this paper, we present an asynchronous parallel I/O framework and demonstrate its efficacy in the Mass Conservation Ocean Model (MaCOM) as a case study. By largely reducing I/O operations in computing processes and overlapping output in I/O processes with computation in computing processes, this framework significantly improves the performance of the MaCOM. Through both reordering output data for maintaining data continuity and combining file access for reducing file operations, the I/O optimizing algorithms are provided to improve output bandwidth. In the case study of the MaCOM, the cost of output in I/O processes can be overlapped by up to 99% with computation in computing processes as decreasing output frequency. The 1D data output bandwidth with these I/O optimizing algorithms is 3.1 times faster than before optimization at 16 I/O worker processes. Compared to the synchronous parallel I/O framework, the overall performance of MaCOM is improved by 38.8% at 1024 computing processes for a 7-day global ocean forecast with 1 output every 2 h through the asynchronous parallel I/O framework presented in this paper.

Keywords: parallel I/O; asynchronous I/O; overlapping output; combining file access; NetCDF

Citation: Pang, R.; Yu, F.; Zhang, Y.; Yuan, Y. An Asynchronous Parallel I/O Framework for Mass Conservation Ocean Model. *Appl. Sci.* **2023**, *13*, 13230. https://doi.org/10.3390/app132413230

Academic Editor: Jae Soo Yoo

Received: 9 November 2023
Revised: 30 November 2023
Accepted: 5 December 2023
Published: 13 December 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Mass Conservation Ocean Model (MaCOM) is a numerical ocean model developed by the National Marine Environmental Forecasting Center of China based on the non-Boussinesq momentum equations in the pressure coordinate system. The MaCOM, along with its data assimilation system, is capable of providing short-term forecasts of temperature, salinity, flow velocity, and sea surface height from global basins to regional areas with a valid forecasting period of 7 days.

The MaCOM utilizes cubed sphere grids and the pressure vertical coordinate system. The physical solving equations of the MaCOM are based on the Navier–Stokes equations, which are simplified with methods including the spherical approximation of the Earth, the thin layer approximation of seawater, and the hydrostatic balance approximation. The simplified equations for solving the horizontal velocity component **u** in the Cartesian coordinate system are shown in Equations (1) and (2), Φ represents geopotential height, ζ_a represents absolute vorticity, KE ($KE = \frac{1}{2}(u^2 + v^2)$) represents the kinetic energy of the fluid element, G^{hdiss} represents the parameterization of small-scale physics for horizontal momentum, G^{vdiss} represents the parameterization of small-scale physics for vertical momentum, G^{force} represents the parameterization of surface turbulent fluxes between the atmosphere and ocean, and G^{drag} represents the parameterization of drag between water and land/ice. The equation for solving the vertical velocity ω in the p coordinate system is shown in Equation (3), and ∇_p represents the generalized derivative vector operator in the pressure vertical coordinate. The equations for solving the active tracers (temperature T and salinity S) are shown in Equations (4) and (5), where G^{hdiff} represents the parameterization of small-scale physics for horizontal tracer diffusion, G^{vdiff} represents the parameterization of small-scale physics for vertical tracer diffusion.

$$\frac{\partial u}{\partial t} = -\frac{\partial \Phi}{\partial x} + \zeta_a v - \frac{\partial KE}{\partial x} - \omega \frac{\partial u}{\partial p} + G_u^{hdiss} + G_u^{vdiss} + G_u^{force} + G_u^{drag} \quad (1)$$

$$\frac{\partial v}{\partial t} = -\frac{\partial \Phi}{\partial y} - \zeta_a u - \frac{\partial KE}{\partial y} - \omega \frac{\partial v}{\partial p} + G_v^{hdiss} + G_v^{vdiss} + G_v^{force} + G_v^{drag} \quad (2)$$

$$\frac{\partial \omega}{\partial p} = -\nabla_p \cdot \mathbf{u} \quad (3)$$

$$\frac{\partial T}{\partial t} = -\nabla_p \cdot (TU) + G_T^{hdiff} + G_T^{vdiff} + G_T^{force} \quad (4)$$

$$\frac{\partial S}{\partial t} = -\nabla_p \cdot (SU) + G_S^{hdiff} + G_S^{vdiff} + G_S^{force} \quad (5)$$

NetCDF (Network Common Data Form) is the data format of I/O (Input/Output) in MaCOM. It is a machine-independent format that is widely used for storing and retrieving multidimensional data in large-scale scientific applications such as climate, oceanic, and atmospheric models [1]. It is also user-friendly to read and write data in files through a variety of interface libraries of NetCDF.

In the performance test of MaCOM, I/O becomes a bottleneck that impedes efficiency, particularly when a large scale of parallel processes is involved. As the number of computing processes increases, the computation time decreases proportionally, which shows strong scalability. However, the time of I/O decreases disproportionately with computation. For instance, for a global 1/12-degree modeling of ocean circulation with an output interval of 2 h that is used in forecast result comparison and artificial intelligence (AI) models, I/O takes up over 40% of the entire execution time when MaCOM runs on 1024 processors. The sea surface temperature and salinity, generated by numerical ocean models, are commonly employed to provide rapid forecast results with AI methods such as long-short term memory [2], convolutional neural network [3], gate recurrent unit [4], etc.

To enhance the efficiency of I/O, we provide an asynchronous parallel I/O framework that can significantly reduce I/O costs in computing processes. The contribution of this paper includes: (1) providing a highly efficient asynchronous parallel I/O framework, which implements independent I/O processes for I/O operations, thus significantly reducing I/O costs by overlapping I/O operations in I/O processes with computation in computing processes; (2) presenting optimizing algorithms to improve output bandwidth by reordering I/O data for continuous data access and decreasing the number of file accessing times by combining file open and closing operations; (3) verifying the efficiency of the asynchronous parallel I/O framework and data output optimization through a case study of MaCOM.

The remainder of this paper is organized as follows. Section 2 summarizes the related work, including libraries of asynchronous parallel NetCDF I/O and their optimizing algorithms. Section 3 explains the motivation behind developing an asynchronous parallel I/O in MaCOM. The methods used to design and optimize the asynchronous parallel I/O are discussed in Section 4. Section 5 presents experimental results and analysis. Finally, the paper concludes in Section 6.

2. Related Work

There are several I/O methods based on NetCDF that are widely used in hydrodynamic models, as shown in Figure 1. The basic NetCDF libraries include NetCDF and PnetCDF (Parallel NetCDF). In contrast, higher-level I/O libraries such as XIOS (XML IO Server), PIO (Parallel I/O Libraries), and CFIO (Climate Fast I/O) call NetCDF or PnetCDF APIs to implement new features, including asynchronous I/O. NetCDF typically calls HDF (Hierarchical Data Format) to store and retrieve data, but it can also call PnetCDF for reading and writing data with the classic NetCDF format. PnetCDF depends on MPI-IO (Message Passing Interface Input Output) to store and retrieve data by default. However, it can also call NetCDF for I/O to support NetCDF4 format.

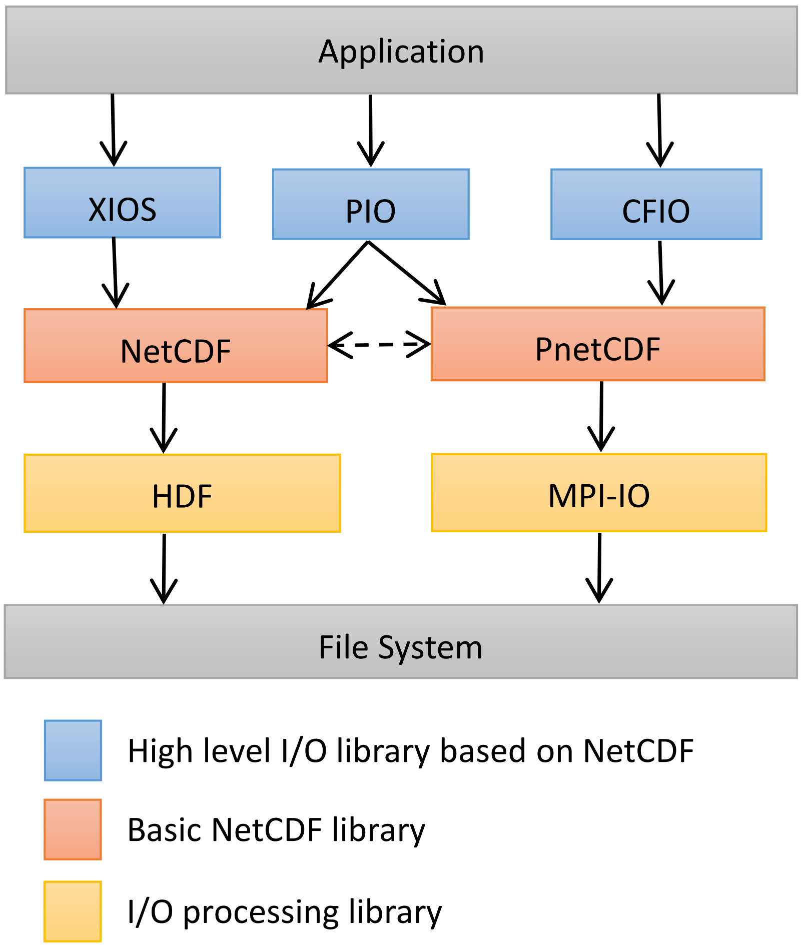


Figure 1. I/O methods based on NetCDF.

NetCDF is a general basic library that defines a set of I/O functions and a machine-independent file format to support the creation, access, and sharing of array-oriented scientific data [5,6]. It is widely used by many physical simulation models, such as POP (Parallel Ocean Program) [7], CLM (Community Land Model) [8], CICE (Community Ice CodE) [9], and WRF (Weather Research and Forecasting) [10]. NetCDF provides a way to encapsulate structured scientific data of a common variable data type to support high-level data access and shell-level application programming [6]. NetCDF performs I/O to a single NetCDF file via one process with the serial netCDF API through collecting and distributing data from and to other processes [11]. However, this approach has the drawback of causing an I/O performance bottleneck and may overwhelm its memory capacity [11]. Additionally, NetCDF does not support irregular I/O data or asynchronous data reading and writing.

PnetCDF uses collective I/O operations through MPI-IO to optimize data reading and writing in the file system [11,12], which is used in CAM (Community Atmosphere Model) [13] and GEOS-5 (Goddard Earth Observing System, version 5) [14]. The collective method in PnetCDF can improve parallel I/O performance by significantly reducing many small and non-contiguous I/O requests [11]. PnetCDF supports irregular data distribution [5], MPI derived data types, and non-continuous data accessing from different processes on a single NetCDF file [11]. PnetCDF can collect multiple I/O requests over a record variable and optimize the I/O over a large pool of data transfers by producing more contiguous and larger transfers [15]. For access to large regions of single array variables, PnetCDF can achieve high performance [15]. However, PnetCDF does not have asynchronous I/O APIs to improve I/O performance.

PIO supports several back-end I/O libraries including NetCDF and PnetCDF [16], which is used in CESM (Community Earth Systems Model) [12,16] and MPAS-Ocean (Model for Prediction Across Scales—Ocean) [17]. PIO can redistribute data in all the I/O processes and rearrange data in memory into a more I/O-friendly decomposition [18]. PIO facilitates continuous output for NetCDF format by using I/O decomposition methods that relocate each data element in the multi-dimensional array according to its physical layout, compared with the PnetCDF library [12]. PIO can perform data output operation through dedicated I/O processes asynchronously [19]. These methods can improve the I/O performance and minimize the memory consumption of PIO efficiently [18].

CFIO is a parallel I/O library that is based on PnetCDF and has been specifically developed for climate models, with the goal of providing automatic computation and I/O overlap [18]. A CFIO server is needed to be an independent program that runs on an I/O process to receive I/O requests from computing processes. It then calls the PnetCDF API to output the data into the underlying parallel file system [18]. However, the output operation of the CFIO server can fail due to the failure of I/O forwarding or other errors, which can result in data loss [18]. When using CFIO with a POP ocean model at a 0.1 resolution degree, it was found to provide a 7.9 times speedup compared to using NetCDF [18].

The XML Input/Output Server (XIOS) is an asynchronous MPI parallel I/O server [20] based on NetCDF that is used in NEMO (Nucleus for European Modelling of the Ocean) [21] for I/O operations. The XIOS server is controlled by an XML file, which defines the characteristics of input and output data, including model fields, domains, grids, I/O frequencies, time averaging for outputs, etc. [21]. The computing processes send I/O requests to the XIOS server, which buffer data in memory with a decisive advantage of not interrupting computing processes as reading and writing in the file system [21].

3. Motivation

At the beginning, MaCOM used PnetCDF to implement I/O functions as MOM (Modular Ocean Model) [22] and sbPOM (Stony Brook Parallel Ocean Model) [23]. Because PnetCDF does not support asynchronous I/O, computation and I/O are executed in the same processes. For the I/O settings, the output frequency is once every 2 h, and the output variables include 6 1D variables (with reduced dimension from horizontal 2D after data optimization), including sea surface height and 5 2D variables (with reduced dimension from horizontal and vertical 3D after data optimization), including sea temperature, salinity, and others. The data volume of each 1D variable is 43.9 MB and 3.3 GB for each 2D variable.

The breakdown of the MaCOM wall time with the global 1/12-degree grid resolution for a 7-day forecast with a 2-h output frequency is shown as Figure 2. As the parallelism increases, the performance of the MaCOM is dominated by the I/O efficiency. For example, I/O takes 10.6% of the whole runtime with 128 processes, but it reaches 40.6% with 1024 processes. With an increasing number of computing processes, the computing cost decreases proportionally, while the I/O cost declines slowly compared to computation. Due to the I/O speed limit of a file system and competition to access the same file from all processes, writing data into one file will be a bottleneck for synchronous I/O with massive processes. That is a general performance problem for synchronous parallel I/O. To solve this problem, asynchronous I/O with independent I/O processes implemented in PIO, XIOS, or CFIO is an effective method [24–26].

In oceanic and meteorological models, the prognostic variables, including temperature, salinity, and three velocity components (zonal, meridional, and vertical), are numerically integrated forward in time by solving the governing equations. This allows for overlapping the storage of computing results from the preceding time step with the dynamical integration of the subsequent time step. This feature enables MaCOM to utilize asynchronous I/O, thereby improving its performance. Meanwhile, to facilitate post-processing of output of MaCOM, the output of all variables from I/O processes in the same period should be written into a single NetCDF file.

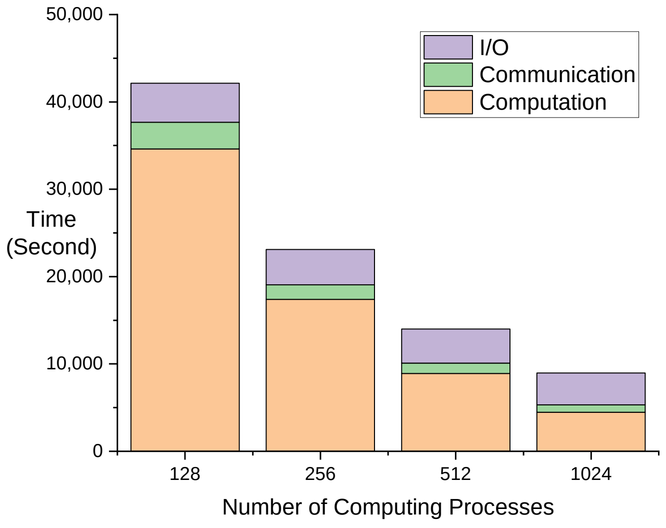


Figure 2. Time of computation, communication, and I/O with PnetCDF in MaCOM.

MaCOM is an operational model that runs daily on a high-performance computer to provide global operational service. To improve the timeliness of this model and save computing resources, maximum improvement in I/O performance is required since it is a bottleneck during the execution of the model. Non-blocking communication in MPI is an effective way to save communication time when sending output data [27–29]. However, to work with non-blocking communication in MPI, there should be a method to avoid the buffer of sending output data in the current I/O step from being written by the next I/O step before completing communication in the current I/O step. Unfortunately, there is no flexible library including NetCDF, PnetCDF, PIO, and CFIO to meet the requirement of sending I/O data with non-blocking communication and protecting the buffer of sending output data from being overwritten incorrectly by the next step of sending output data. Therefore, it is necessary to develop an effective communication method to meet the above requirements of non-blocking communication for higher I/O performance.

Due to the unstructured nature of grid decomposition in MaCOM, I/O data is scattered in different processes. Therefore, it can improve output performance after reordering irregular and discontinuous output data into continuous data. It is also convenient for data post-processing. However, there are no related functions to reorder output data according to the user-defined order in the NetCDF, PnetCDF, PIO, XIOS, and CFIO libraries.

Combining I/O operations for multiple array variables has been proven to be a way to improve I/O performance [15]. Unfortunately, this optimizing method is not currently implemented in current NetCDF libraries. To best improve I/O performance, we need to develop a new asynchronous output framework with all aforementioned optimizing methods in the MaCOM.

Compared with the NetCDF, PnetCDF, PIO, XIOS, and CFIO libraries, the new features of the asynchronous parallel I/O framework proposed in this paper are as follows: (1) sending I/O data with non-blocking communication and protecting the buffer of sending output data from being overwritten incorrectly by the next step of sending output data; (2) reordering output data according to a user-defined order to improve output bandwidth; (3) combining I/O operations for outputting multiple array variables to reduce file operations.

4. Design and Algorithm

4.1. Workflow of the Asynchronous Parallel Output Framework

In our new design, the I/O operation has been largely moved from computing processes to dedicated I/O processes. The workflow for the asynchronous parallel output framework in computing processes and I/O processes is shown in Figure 3. The computing processes (total number is m) send output data to the I/O processes through MPI non-blocking communication after completing a fixed number of user-defined integral calculations. The computing processes continue to run without waiting for completion of communication. To prevent the sending buffer of I/O data from being overwritten by the next step of non-blocking communication, a method is employed to check communication status and protect the sending buffer until the communication in the previous step is completed. This is a trade-off that increases the memory usage of output variables in the preceding computing step in order to achieve asynchronous communication in computing processes.

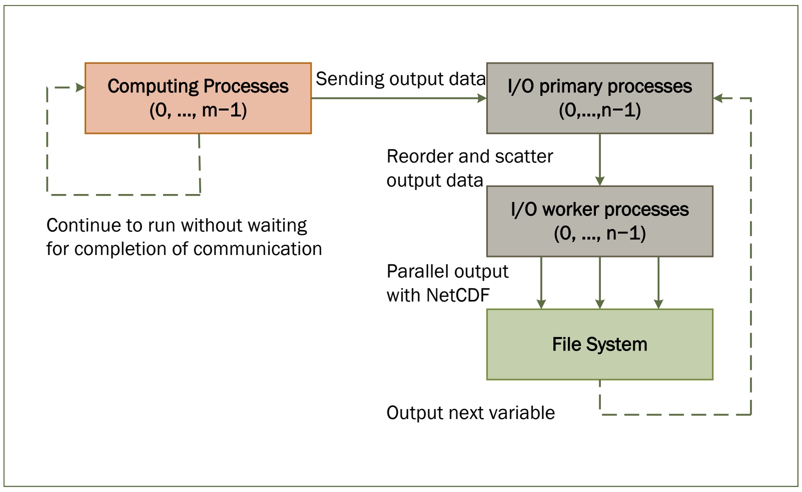


Figure 3. Workflow of the asynchronous parallel output framework.

In the I/O processes (total number is n), all processes are I/O workers responsible for outputting data into the same NetCDF file. Some of the I/O processes (total number is p, where $1 \leq p \leq$ n) are selected as primary processes. Each primary process collects data from the corresponding computing processes through non-blocking communication. Then, each primary process sorts the scattered data into a continuous order. Next, each primary process partitions the adjusted data equally and distributes them to the corresponding I/O worker processes. Finally, all I/O worker processes perform parallel output of the data into the same NetCDF file. After completing the above steps, the I/O primary processes resume receiving I/O data from computing processes for outputting next variables.

4.2. Data Communication and Protection in Computing Processes

The workflow of data communication and protection in computing processes is shown in Figure 4. In step 1, the mpi_wait function is used to check the status of communication in step 3 and wait until it is finished. Initially, communication check and protection in step 1 is not executed until I/O data communication in step 3 has begun. In step 2, the assemble function collects output data and saves them in the sending buffer of send_buf. The send_buf can be accessed by a block of continuous elements instead of one element for better memory accessing performance. In step 3, the mpi_igatherv function of the MPI APIs is used to send data to the I/O primary processes through collective non-blocking communication.

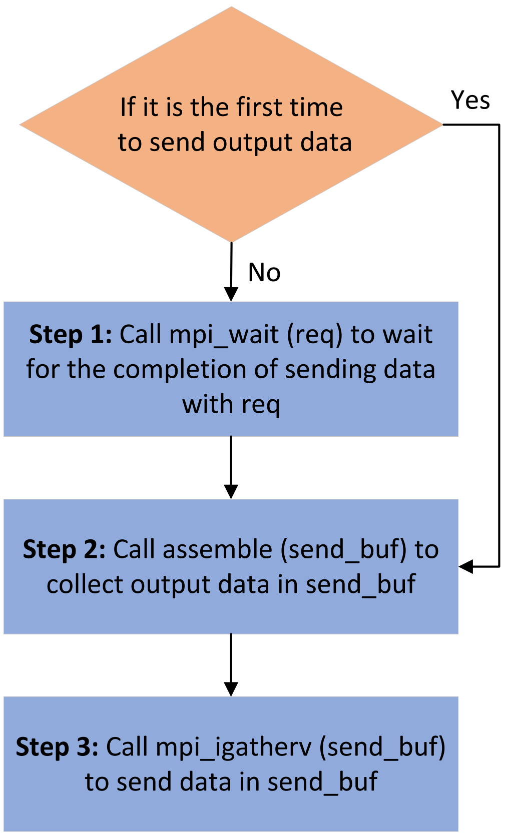

If it is the first time to send output data
Yes
No
Step 1: Call mpi_wait (req) to wait for the completion of sending data with req
Step 2: Call assemble (send_buf) to collect output data in send_buf
Step 3: Call mpi_igatherv (send_buf) to send data in send_buf

Figure 4. Workflow of data communication and protection.

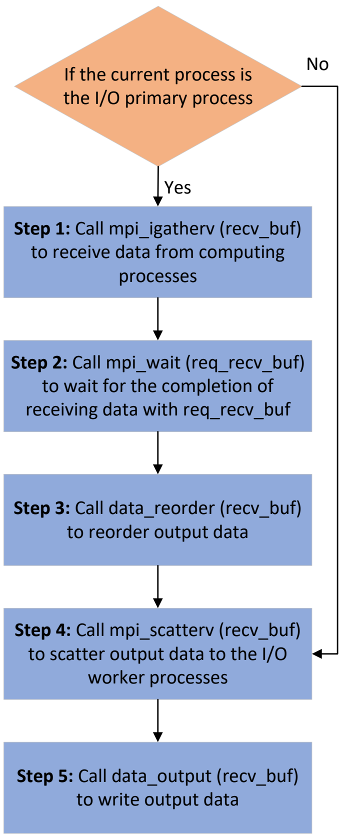
4.3. Reordering and Parallel Outputting Data in I/O Processes

The workflow for reordering and parallel outputting data in I/O processes is shown in Figure 5. First, the I/O primary process receives data from the corresponding computing processes through the mpi_igatherv function in step 1. Second, the I/O primary process waits for communication to be completed in step 1 by calling mpi_wait of the MPI APIs. Third, the I/O primary process calls a self-defined function of data_reorder to adjust the data into a continuous order for output. The method of data reordering is shown in the following section of data optimization. Next, the I/O primary process partitions the output data equally and continuously based on the number of I/O worker processes, and scatters these data to the I/O worker processes in step 4. Finally, all I/O worker processes use data_output in step 5 to parallel write the output data into a NetCDF file by calling nf90_put_var of the NetCDF APIs or nf90mpi_put_var_all of the PnetCDF APIs according to the user's preference.

To output data into a file in step 5, there are two methods: independent file access and combined file access. The independent file access method executes the functions of opening and closing a file for outputting each variable output. In the combined file access method, the function of opening a file is only executed once at the beginning of outputting all variables, and the function of closing a file is also only executed once at the end of outputting all variables. With the combined file access method, the cost of frequently opening and closing a file can be avoided.

4.4. Data Optimization

Redundant data on land grids that is not used in ocean models is removed, as shown in Figure 6. The gray grids represent land, and the blue grids represent ocean. First, the pre-processing program of MaCOM deletes all land grids that are not neighbors of ocean grids. Second, the remaining ocean grids and their neighboring land grids are reordered from 2D to 1D. This can save nearly 28% in memory usage and unnecessary computation in global ocean models. It can further improve load balance among computing processes due to the different computational load between an ocean grid and a land grid.



If the current process is the I/O primary process
No
Yes
Step 1: Call mpi_igatherv (recv_buf) to receive data from computing processes
Step 2: Call mpi_wait (req_recv_buf) to wait for the completion of receiving data with req_recv_buf
Step 3: Call data_reorder (recv_buf) to reorder output data
Step 4: Call mpi_scatterv (recv_buf) to scatter output data to the I/O worker processes
Step 5: Call data_output (recv_buf) to write output data

Figure 5. Workflow for reordering and parallel outputting data.

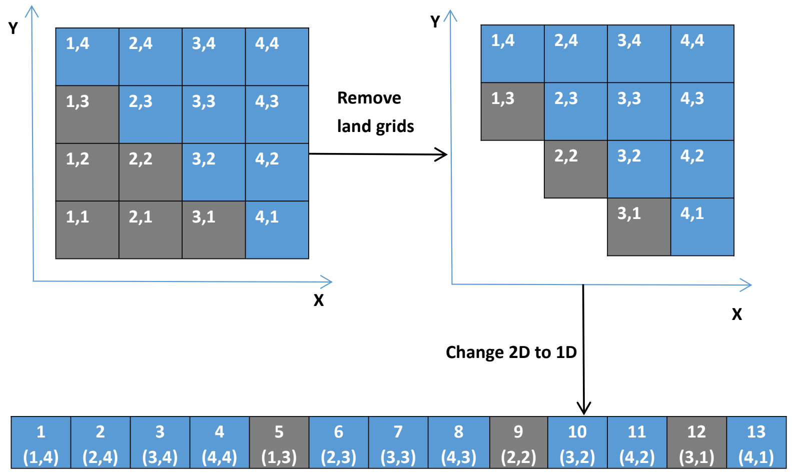


Figure 6. Removing redundant data on land grids.

Data partition in computing processes is completed by the multilevel graph partitioning tool of METIS [30]. Grid neighboring relations are the input for METIS to complete data partition. After reordering the output data, the data is contiguous and sorted in I/O primary processes. The data can be sequentially and continuously divided according to the number of I/O worker processes. Both the data partitioning method in the computing processes and I/O processes are based on horizontal 2D grids.

The order of grids in computing processes depends on the neighboring relation of grids. The data is arranged in the result file in the order of a fixed grid index. The data received from computing processes is disordered and non-contiguous for output. To improve output performance, the data is reordered in I/O primary processes, as shown in Figure 7. First, we combine continuous elements in a block that are continuous in both the I/O primary process and the output file, and operate on them together by block. For example, elements 5 and 6, which are continuous in both the receiving buffer and the output file, will be copied at one time to the adjusted array. Next, we reorder and store these blocks in the same order as the output file.

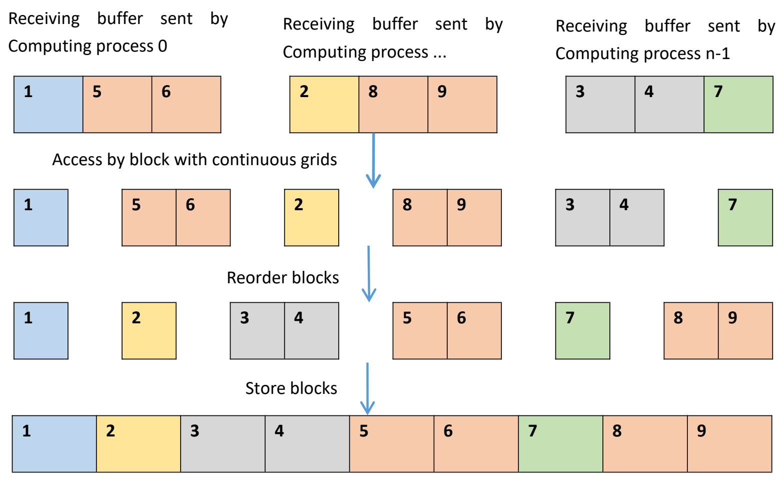

Receiving buffer sent by Computing process 0
Receiving buffer sent by Computing process ...
Receiving buffer sent by Computing process n-1
1 5 6
2 8 9
3 4 7
Access by block with continuous grids
1 5 6 2 8 9 3 4 7
Reorder blocks
1 2 3 4 5 6 7 8 9
Store blocks
1 2 3 4 5 6 7 8 9

Figure 7. Data reordering method.

5. Experiment

5.1. Experiment Environment and Configuration

Our experiments were conducted on the cluster of the National Marine Environmental Forecasting Center in Beijing of China. The software and hardware environment used in the test is shown in Table 1.

Table 1. Software and hardware environment.

Name	Version
CPU	Intel(R) Xeon(R) CPU E5-2680 v4 2.40 GHz
Node	2 Intel CPU (14 cores each CPU)
Memory	128 GB
Hardware Architecture	X86_64
Operating System	Linux 3.10.0
Compiler	Mpiifort version 2021.4.0
Compiling Option	-O3
MPI	Intel(R) MPI Library 2021u4
NetCDF	NetCDF 4.7.4
PNetCDF	PnetCDF 1.12.3
File System	LUSTRE 2.12.6

For the MaCOM configuration, the grid resolution is global 1/12-degree, the number of horizontal grids after removing land grids is 5,749,313, and the vertical layer is 75. The execution time was calculated by calling the system_clock of Fortran intrinsic functions. The parameter of system_clock is defined as a double precision integer, and its time counting frequency is 10^6 per second.

We performed a test of MaCOM with a fixed number I/O processes and different I/O output frequency to evaluate asynchronous output efficiency through overlapping output with computation. We also measured parallel output bandwidth with different I/O processes using several I/O optimizing algorithms. Finally, we compared the execution performance of MaCOM after using the asynchronous parallel output framework provided in this paper.

5.2. Asynchronous Parallel Output Performance

We used 1024 computing processes and 4 I/O processes (1 primary process and 4 I/O worker processes) to test the output time in computing processes with different output frequencies. The total times of output was 5, while the output frequency increased from 1 output per 1 step of computation to 1 output per 10 steps of computation. One 1D double precision real variable (43.86 MB) and one 2D double precision real variable (3.21 GB) were outputted into a NetCDF file each time. The time statistics of the output operations for 5 times of output in computing processes are shown in Figure 8. Due to the fixed number of computing processes and I/O processes, it takes a fixed and small amount of time to assemble and send data with non-blocking communication. The time for one step of computation is much slower than the time for one time of output. To prevent overwriting the reusable sending buffer, computing processes have to wait until sending data to I/O primary processes is completed. The waiting time gradually reduces with decreasing output frequency. It becomes an extremely small and stable amount of time (0.2 s for 5 waiting operations) for 1 output per 9 or more computing steps.

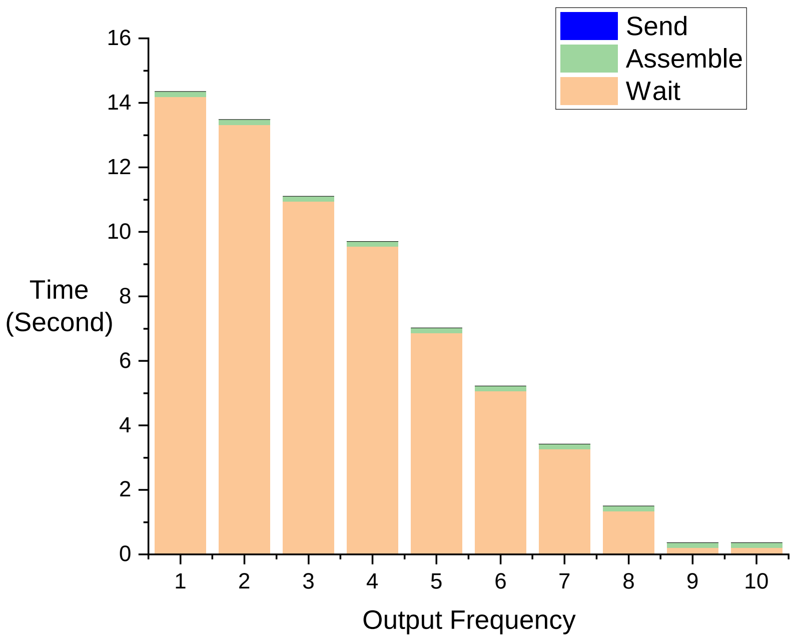


Figure 8. Output time in computing processes.

The output time with four I/O processes to write five times of one 1D variable and one 2D variable (total volume of 16.28 GB) is 21.88 s. Based on the output time in the computing processes shown in Figure 6, we calculated the percentage of overlapping output with computation, as shown in Figure 9. The cost of outputting one 1D variable and one 2D variable can be overlapped by up to 99.09% with one output per nine steps of computation. It can significantly improve the I/O performance with the asynchronous parallel output framework proposed in this paper by overlapping I/O operation with computation.

5.3. Parallel Output Performance

We measured the time of outputting one 1D variable and one 2D variable in I/O processes with different numbers of I/O processes and the same 1024 computing processes. Because each node has 28 cores, the last computing processes utilize 16 CPU cores, while the initial I/O processes utilize 12 CPU cores within the same node. The subsequent I/O processes exceeding 12 occupy an entire node. The total times of output for each variable is 5, and the output frequency is 1 output per 10 steps of computation. The number of I/O primary process was the same as one. The bandwidth for outputting 5 times of one 1D variable (reduced from horizontal 2D in data optimization, 43.86 MB each time of output) into a file is shown in Figure 10. The parallel output bandwidth through PnetCDF reached the peak of 2 GB/s at 16 I/O processes. Due to competition for I/O bandwidth on the same file, the output bandwidth decreases when there are more than 16 I/O processes to output 1D variable.

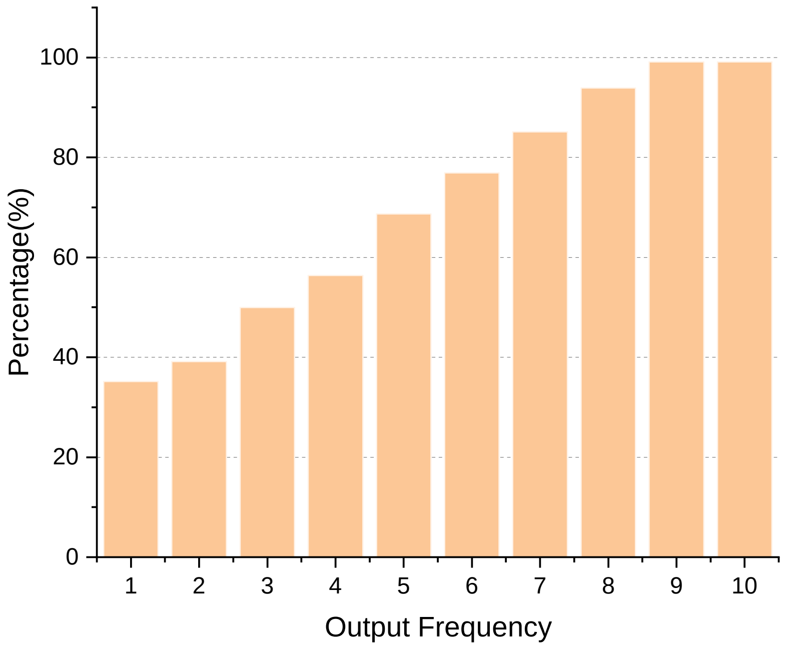


Figure 9. Percentage of overlapping output with computation.

Data reordering optimization to merge a large number of non-continuous blocks into one continuous block can significantly increase output bandwidth. File accessing optimization through reducing the number of times of opening and closing file for outputting variable can also improve output performance. For example, the output bandwidth after file accessing optimization and data reordering optimization is 3.1 times faster than before optimization at 16 I/O worker processes.

The runtime for outputting the 1D variable five times at the same I/O primary and worker process is shown in Figure 11. File opening and closing operations are only performed once by all worker processes. The runtime of file operations slightly increases as the number of worker processes increases, mainly due to I/O competition among them. Data reordering is only executed by the primary I/O process, and its runtime remains almost constant because of the consistent total data volume. The runtime for scattering data increases gradually as the number of worker processes increases. The minimum runtime for outputting data is achieved with 16 I/O workers.

The bandwidth for outputting five times of one 2D variable (3.21 GB for each time of output) into a file is shown in Figure 12. Similar to outputting the 1D variable, reordering the data can significantly improve the performance of outputting the 2D variable. The size of the first dimension of the 2D variable is the same as that of the 1D variable, and the second dimension of the 2D variable is 75. The reduced time through file accessing optimization is fixed and only frequency-related. Due to the fact that file accessing operations consume a decreasing percentage of output time as the output data volume rises, it has minor impact on optimizing efficiency while outputting the 2D variable.

Data continuity is impacted by how I/O data is partitioned. I/O data along the first dimension (5,749,313 horizontal grids) increases parallel scalability but breaks data continuity. I/O data along the second dimension (75 vertical layers) reduces parallel scalability but keeps all I/O data continuous. The output performance with data partition along the second dimension has much better performance than that with data partition along the first dimension. For example, output bandwidth after file accessing optimization and data reordering optimization along the second dimension is 2.55 times faster than that along the first dimension, and 2.94 times faster than before optimization at 32 I/O worker processes. The parallel output bandwidth through PnetCDF reached the peak of 1.79 GB/s at 32 I/O processes, which is more than 16 I/O worker processes for outputting the 2D variable. This shows that the amount of data can improve the parallel scalability of output. However, since the size of second dimension is small, as being 75, it causes more load imbalance compared to data partition along the first dimension. This could be the cause of the peak output performance for the 2D variable being lower than for the 1D variable.

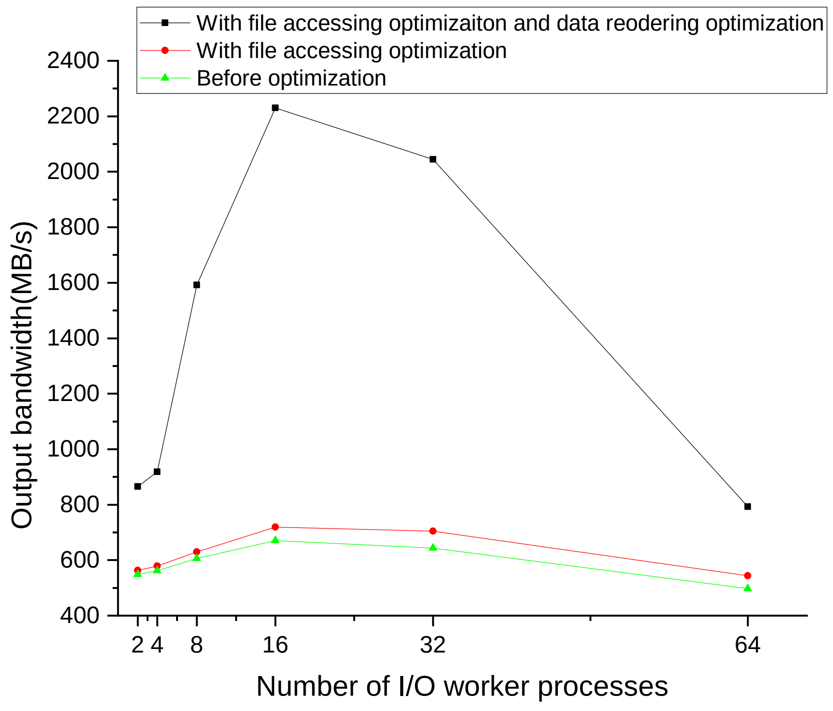


Figure 10. Bandwidth of outputting a 1D variable.

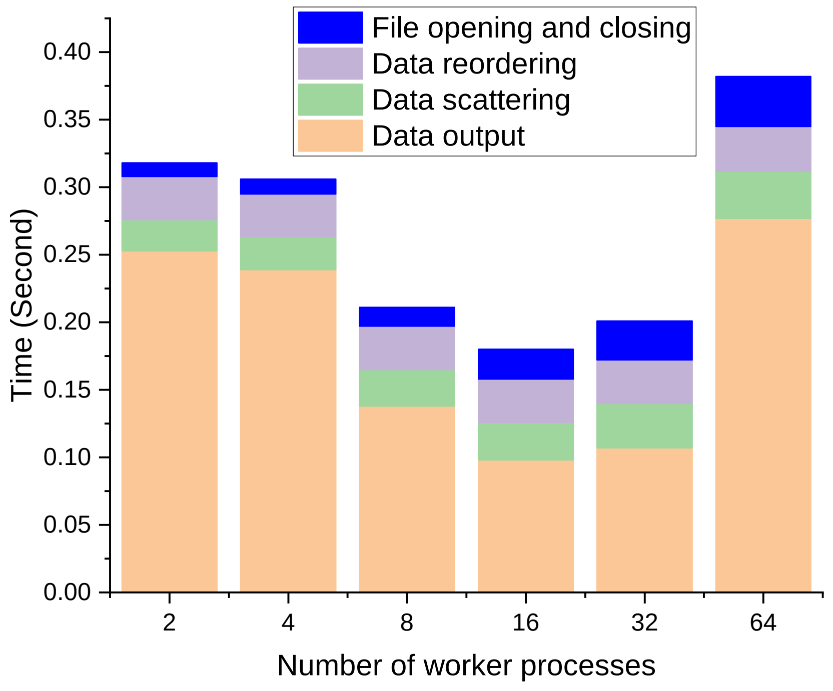


Figure 11. Runtime of outputting a 1D variable at the same I/O primary and worker process.

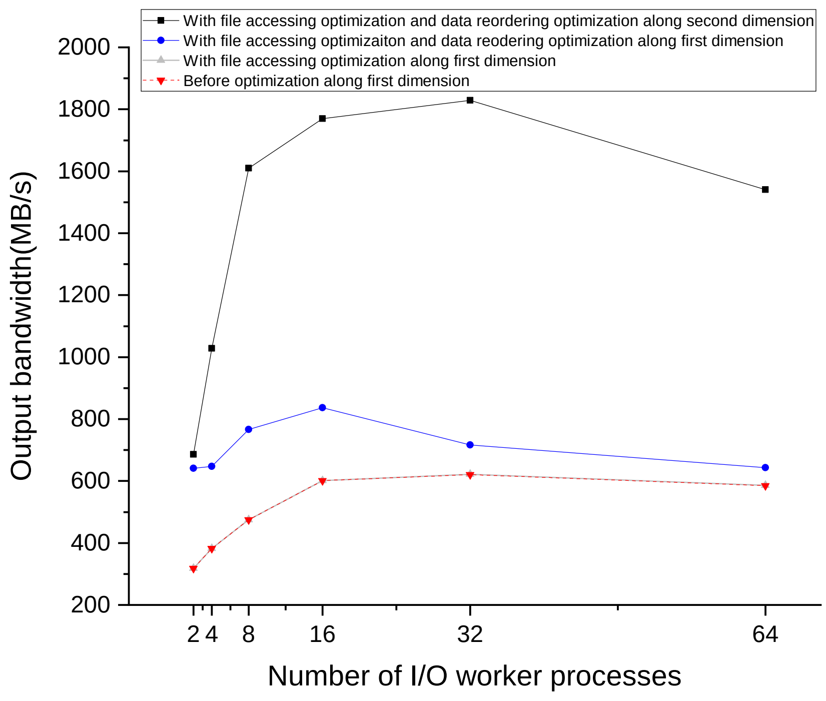


Figure 12. Bandwidth of outputting a 2D variable.

The runtime for outputting the 2D variable along the second dimension five times at the same I/O primary and worker process is shown in Figure 13. The runtime for file opening and closing operations has no connection to the output data volume. They have minor effect on the performance optimization of outputting the 2D variable. The runtime of data reordering, scattering, and output for outputting the 2D variable scales up compared to outputting the 1D variable. The runtime of data scattering and output is also affected by the number of worker processes.

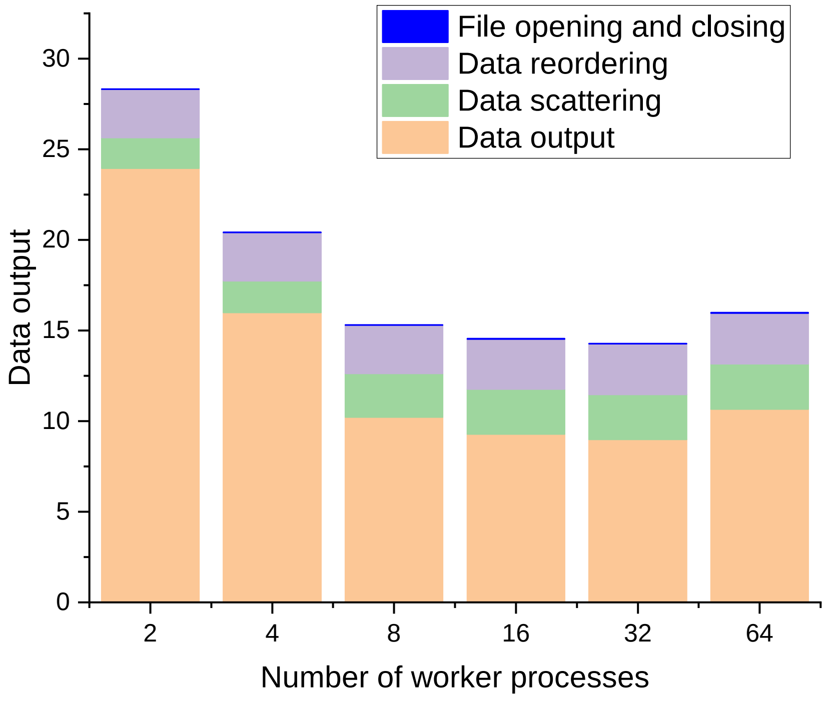


Figure 13. Runtime of outputting a 2D variable along the second dimension at the same I/O primary and worker process.

5.4. Output Evaluation in MaCOM Model

We conducted a runtime test of MaCOM with different numbers of computing processes, from 128 to 1024. The synchronous parallel I/O framework using PnetCDF utilizes all computing processes for data output. This framework is also widely employed in other oceanic and atmospheric models, such as the Modular Ocean Model 5 (MOM5) [22] and the Weather Research and Forecasting model (WRF) [31]. The asynchronous parallel I/O framework proposed in this paper employs 4 I/O processes exclusively and consistently (1 primary process and 4 I/O worker processes) for data output. The forecast period is 7 days, which is required by operational forecasting service. The total computing steps are 15,120. The output frequency is once per 2 h, and the total number of outputs is 84. The output variables consist of six 1D variables and five 2D variables, and the total output data volume is 1.37 TB.

The runtime of the MaCOM with both synchronous I/O and asynchronous parallel I/O is shown in Figure 14. The asynchronous parallel I/O framework in this paper can significantly improve performance of MaCOM. For example, it reduced the runtime by 3474.4 s at 1024 computing processes after I/O optimization, and improved the performance of MaCOM by 38.8%. This means that with the asynchronous I/O framework the runtime of MaCOM at 1024 processes can meet the requirement for operational service, as the runtime to provide 7-day forecasting products is less than 2 h.

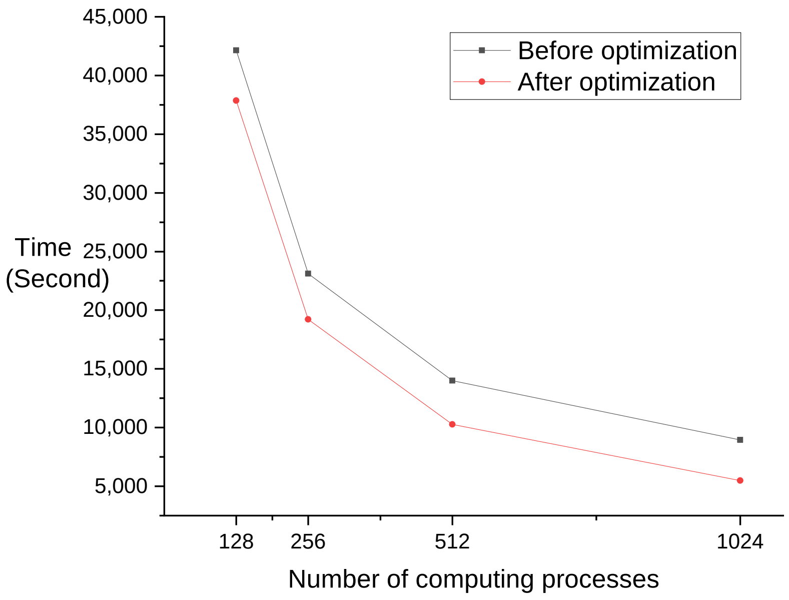


Figure 14. Runtime of the MaCOM model with different I/O methods.

6. Conclusions and Future Work

In this work, we presented a highly efficient asynchronous parallel I/O framework that supports both PnetCDF and NetCDF. This framework can significantly reduce I/O time in computing processes by overlapping I/O with computation in the case study of MaCOM. Moreover, we improved parallel output bandwidth through reordering the output data for data continuity and reducing the I/O file accessing operations by combining them. The test results showed that data continuity is a key factor for improving output performance.

In the future, we plan to conduct further evaluations on different machines with varying file systems. We also plan to improve communication performance among I/O processes with non-blocking communication and investigate output bandwidth optimization with other methods.

Author Contributions: Conceptualization, F.Y. and Y.Y.; methodology, R.P. and Y.Z.; software, R.P. and Y.Z.; resources, F.Y.; writing—original draft preparation, R.P.; writing—review and editing, F.Y., Y.Z. and Y.Y.; supervision, F.Y.; project administration, F.Y.; funding acquisition, Y.Z. and Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China under contract No. 41506031 and the National Key Research and Development Program of China under contract No. 2023YFC3107800.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rew, R.; Davis, G. NetCDF: An interface for scientific data access. *IEEE Comput. Graph. Appl.* **1990**, *10*, 76–82. [CrossRef]
2. Xie, J.; Zhang, J.; Yu, J.; Xu, L. An adaptive scale sea surface temperature predicting method based on deep learning with attention mechanism. *IEEE Geosci. Remote Sens. Lett.* **2019**, *17*, 740–744. [CrossRef]
3. Zhang, X.; Zhao, N.; Han, Z. A Modified U-Net Model for Predicting the Sea Surface Salinity over the Western Pacific Ocean. *Remote Sens.* **2023**, *15*, 1684. [CrossRef]
4. Chen, G.; Huang, B.; Chen, X.; Ge, L.; Radenkovic, M.; Ma, Y. Deep blue AI: A new bridge from data to knowledge for the ocean science. *Deep Sea Res. Part I Oceanogr. Res.* **2022**, *190*, 103886. [CrossRef]
5. Gao, K.; Jin, C.; Choudhary, A.; Liao, W.K. Supporting computational data model representation with high-performance I/O in parallel netCDF. In Proceedings of the 2011 18th International Conference on High Performance Computing, Bengaluru, India, 18–21 December 2011; pp. 1–10.
6. Galiano, V.; Migallón, H.; Migallón, V.; Penadés, J. PyPnetCDF: A high level framework for parallel access to netCDF files. *Adv. Eng. Softw.* **2010**, *41*, 92–98. [CrossRef]
7. Jones, P.W.; Worley, P.H.; Yoshida, Y.; White, J.B., III; Levesque, J. Practical performance portability in the Parallel Ocean Program (POP). *Concurr. Comput. Pract. Exp.* **2005**, *17*, 1317–1327. [CrossRef]
8. Hoffman, F.M.; Vertenstein, M.; Kitabata, H.; White, J.B., III. Vectorizing the community land model. *Int. J. High Perform. Comput. Appl.* **2005**, *19*, 247–260. [CrossRef]
9. Rae, J.; Hewitt, H.; Keen, A.; Ridley, J.; West, A.; Harris, C.; Hunke, E.; Walters, D. Development of the global sea ice 6.0 CICE configuration for the met office global coupled model. *Geosci. Model Dev.* **2015**, *8*, 2221–2230. [CrossRef]
10. Shantharam, M.; Tatineni, M.; Choi, D.; Majumdar, A. Understanding I/O bottlenecks and tuning for high performance I/O on large HPC Systems: A case study. In Proceedings of the Practice and Experience on Advanced Research Computing, Pittsburgh, PA, USA, 22–26 July 2018; pp. 1–6.
11. Li, J.; Liao, W.K.; Choudhary, A.; Ross, R.; Thakur, R.; Gropp, W.; Latham, R.; Siegel, A.; Gallagher, B.; Zingale, M. Parallel netCDF: A high-performance scientific I/O interface. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, Phoenix, AZ, USA, 15–21 November 2003; p. 39.
12. Zou, Y.; Xue, W.; Liu, S. A case study of large-scale parallel I/O analysis and optimization for numerical weather prediction system. *Future Gener. Comput. Syst.* **2014**, *37*, 378–389. [CrossRef]
13. Tseng, Y.H.; Ding, C. Efficient parallel I/O in Community Atmosphere Model (CAM). *Int. J. High Perform. Comput. Appl.* **2008**, *22*, 206–218. [CrossRef]
14. Liu, Z.; Wang, B.; Wang, T.; Tian, Y.; Xu, C.; Wang, Y.; Yu, W.; Cruz, C.A.; Zhou, S.; Clune, T.; et al. Profiling and improving I/O performance of a large-scale climate scientific application. In Proceedings of the 2013 22nd International Conference on Computer Communication and Networks (ICCCN), Nassau, Bahamas, 30 July–2 August 2013; pp. 1–7.
15. Gao, K.; Liao, W.K.; Choudhary, A.; Ross, R.; Latham, R. Combining I/O operations for multiple array variables in parallel netCDF. In Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–10.
16. Dennis, J.M.; Edwards, J.; Loy, R.; Jacob, R.; Mirin, A.A.; Craig, A.P.; Vertenstein, M. An application-level parallel I/O library for Earth system models. *Int. J. High Perform. Comput. Appl.* **2012**, *26*, 43–53. [CrossRef]
17. Woodring, J.; Petersen, M.; Schmeißer, A.; Patchett, J.; Ahrens, J.; Hagen, H. In Situ eddy analysis in a high-resolution ocean climate model. *IEEE Trans. Vis. Comput. Graph.* **2015**, *22*, 857–866. [CrossRef] [PubMed]
18. Wang, W.; Huang, X.; Fu, H.; Hu, Y.; Xu, S.; Yang, G. CFIO: A fast I/O library for climate models. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, Australia, 16–18 July 2013; pp. 911–918.

19. Hartnett, E.; Edwards, J. The parallelio (PIO) C/FORTRAN libraries for scalable HPC performance. In Proceedings of the 37th Conference on Environmental Information Processing Technologies, American Meteorological Society Annual Meeting, Virtual, 12–15 January 2021; pp. 10–15.
20. Yepes-Arbos, X.; Acosta, M.; van den Oord, G.; Carver, G. I/O scalability boost for the next generation of Earth system models: IFS-XIOS integration as a case study. In Proceedings of the AGU Fall Meeting 2018, Washington, DC, USA, 10–14 December 2018.
21. Boussetta, S.; Simarro, C.; Lucas, D. *Exploring EC-Earth 3.2-Beta Performance on the New ECMWF Cray-Broadwell*; European Centre for Medium Range Weather Forecasts: Reading, UK, 2016.
22. Yang, R.; Ward, M.; Evans, B. Parallel I/O in Flexible Modelling System (FMS) and Modular Ocean Model 5 (MOM5). *Geosci. Model Dev.* **2020**, *13*, 1885–1902. [CrossRef]
23. Jordi, A.; Wang, D.P. sbPOM: A parallel implementation of Princenton Ocean Model. *Environ. Model. Softw.* **2012**, *38*, 59–61. [CrossRef]
24. Balle, T.; Johnsen, P. *Improving I/O Performance of the Weather Research and Forecast (WRF) Model*; Cray User Group: Bloomington, IN, USA, 2016; p. 123.
25. Kougkas, A.; Devarajan, H.; Sun, X.H. Bridging Storage Semantics Using Data Labels and Asynchronous I/O. *ACM Trans. Storage* **2020**, *16*, 1–34. [CrossRef]
26. Byna, S.; Breitenfeld, M.S.; Dong, B.; Koziol, Q.; Pourmal, E.; Robinson, D.; Soumagne, J.; Tang, H.; Vishwanath, V.; Warren, R. ExaHDF5: Delivering efficient parallel I/O on exascale computing systems. *J. Comput. Sci. Technol.* **2020**, *35*, 145–160. [CrossRef]
27. Al-Tawil, K.; Moritz, C.A. Performance modeling and evaluation of MPI. *J. Parallel Distrib. Comput.* **2001**, *61*, 202–223. [CrossRef]
28. Hatanaka, M.; Hori, A.; Ishikawa, Y. Optimization of MPI persistent communication. In Proceedings of the 20th European MPI Users' Group Meeting, Madrid, Spain, 15–18 September 2013; pp. 79–84.
29. Zheng, Y.; Marguinaud, P. Simulation of the performance and scalability of message passing interface (MPI) communications of atmospheric models running on exascale supercomputers. *Geosci. Model Dev.* **2018**, *11*, 3409–3426. [CrossRef]
30. Karypis, G.; Kumar, V. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*; Computer Science & Engineering Technical Reports: Minneapolis, MN, USA, 1997.
31. Christidis, Z. Performance and scaling of WRF on three different parallel supercomputers. In *High Performance Computing, Proceedings of the 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, 12–16 July 2015*; Proceedings 30; Springer: Cham, Switzerland, 2015; pp. 514–528.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.