Strategies for Fast I/O Throughput in Large-scale Climate Modeling Applications

Koushik Sen
Qualcomm
Hyderabad, India
koushiksen@alum.iisc.ac.in

Sathish Vadhiyar
Department of Computational and Data Sciences
Indian Institute of Science
Bangalore, India
vss@iisc.ac.in

PN Vinayachandran
Centre for Atmospheric and Oceanic Sciences
Indian Institute of Science
Bangalore, India
vinay@iisc.ac.in

***Abstract*—Large-scale HPC applications are highly data-intensive with significant times spent in I/O operations. Many large-scale scientific applications do not adequately optimize the I/O operations, leading to overall poor performance. In this work, we have developed two main strategies for providing fast I/O throughput for an important climate modeling application, namely, Regional Ocean Modeling System (ROMS) that uses NetCDF for I/O operations. The strategies include load balancing the I/O operations and selective writing of data. We have also implemented file striping to improve I/O performance. Our experiments with up to 1440 processor cores and 5 days of simulations showed that our load balancing strategy resulted in about 27% decrease in execution times over the default executions, our selective writing strategy resulted in a further decrease of about 30% and the optimized file striping resulted in a further decrease of about 12% in execution times. All the strategies combined together improved the overall performance of the application by about 70%.**

***Index Terms*—Parallel I/O, ROMS climate model, Collective I/O, Lustre striping, netCDF.**

I. Introduction

ROMS [1] is an ocean model widely used by the scientific community for a wide range of climate applications. It can work standalone as well as it can be coupled to atmospheric and/or wave models. It is built based on the Earth System Modeling Framework (ESMF) [2] which provides high performance and flexibility for coupling climate and related other scientific applications. The format of the input and output data of the model are Network Common Data Form (NetCDF) [3] which helps to interchange the data in a user-friendly way.

The ROMS model equations are discretized both in horizontal and vertical directions. It can run serially as well as in parallel using OpenMP and MPI. It has the ability to mask the land points which is done while preparing the input file. We have used MPI for all our experiments. It also supports nested grids to define fine and coarse grids at the same time. In distributed-memory, the I/O can be done serially on the master process or in parallel by all the processes. The parallel I/O requires the NETCDF-4 parallel version which uses parallel HDF-5 library. The HDF-5 library internally calls MPI library functions to interact with the parallel Lustre file system. The I/O stack in ROMS is shown in Figure 1.

This work is funded by Science and Engineering Research Board (SERB), Department of Science and Technology (DST), India, Core Research Grant, Sanction Order No. CRG/2021/005278.

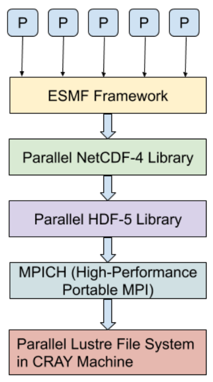

P
P
P
P
P
ESMF Framework
Parallel NetCDF-4 Library
Parallel HDF-5 Library
MPICH (High-Performance Portable MPI)
Parallel Lustre File System in CRAY Machine

Fig. 1: Parallel I/O Framework in ROMS

The ROMS model reads the input data from NetCDF data files, performs computations and then writes different types of output data files including quick, average, history, diagnostic, and restart files depending on user requirements. The quick output files store a snapshot of the present step into the file. The history files are also snapshots but the writes of the history files occur at a higher precision compared to quick files. The average files store the temporal average records over a period of steps as specified by the user. The restart files store the data that is required to restart the model from some specific point of time. The diagnostic files are saved for analyzing various aspects of the ROMS model. The parameters to write and the frequency of writing each such file can be set during execution. The reading of input data takes about 2-3% of the total execution time whereas the writing of output data varies from 50 to 90% of the total execution time depending on the number of variables written, number of output types, their frequencies and number of processes used. Hence, our work is primarily focused on writing the output files.

In this paper, we have performed detailed analyses of the I/O times in the ROMS model and have developed different techniques to improve the performance of the application with the I/O. Different types of analysis like scalability studies, load imbalance studies, bottleneck function studies are performed on the ROMS model to determine the I/O bottlenecks. Different performance techniques are proposed for the different sources of bottlenecks.

The primary objective of this work is to minimize the large I/O times in the ROMs and in general climate models, where the I/O consumes 70% of the total execution times. Towards this, we have developed and explored two main strategies. We have proposed a method for reduction in load imbalance in the write times. Our strategy has resulted in at least 27% reduction in the overall execution time for 1440 cores. We also observed that some of the high-dimensional data in ROMS are slowly varying across time steps. We have proposed a second strategy that selective writes only some data in a time step within the constraints of acceptable accuracy loss. This strategy gave around a 52% reduction in I/O write time and a 31.6% reduction in the overall execution time for 1440 cores. We also explored different configurations for striping Lustre data. The optimized Lustre striping improved the I/O write performance by another 41% in the ROMS model. All the strategies combined together improved the parallel NetCDF performance by about 70%. Thus, significant improvements are achieved in the Parallel I/O performance as well as in the overall performance of the ROMS model using our I/O improvement strategies.

The paper is organized as follows. Section II present related work in the area of I/O optimizations for large-scale applications. Section III gives background on the I/O in the ROMS model, and the model setup used for this work. Section IV gives detailed analysis on the scalability of the ROMS model with I/O and profiles the I/O bottlenecks in the model. Sections V, VI and VII explain our optimization strategies related to load balancing, selective writing and Lustre striping, respectively. These sections also present experiments and results related to performance improvement obtained due to the strategies. Section VIII gives conclusions and future work.

II. Related Work

Liu et al. [4] have implemented implemented I/O backends in GEOS, based on PnetCDF and Adaptable IO System (ADIOS) for alternative I/O solutions to the performance issues in large scale applications. Adaptable IO System is quite flexible in terms of I/O routine selection. The existing implementation of the Goddard Earth Observing System Model (GEOS) [5] collects data from multiple variables with one or more planes and then writes them to a bundle file. The data of each plane is collected by a plane root process. Then, the data is sent to the bundle root from the different plane roots. For different planes and bundles, different processes are selected so that they can work in parallel. However, this approach does not scale well due to resource contention, memory overhead and network bandwidth saturation.

ADIOS by Lofstead et al. [6] performs asynchronous I/O and offers multiple customizable I/O mechanisms. ADIOS can write data from multiple bundles in the same file in parallel. The data is stored in buffer memory and written to disk if the buffer is full. This helps to reduce the total number of disk access. Also, there is no inter-process communication overhead. For improving the read performance, Tang et al. [7] proposed methods to develop an online analyzer with low computational and low memory overhead maintaining high accuracy. The main objective is to detect patterns in reading and use these patterns to reduce file read times. The work supports both structured and unstructured reading patterns. The rule-based model performs access pattern analysis during run-time and utilizes the pattern detected to perform prefetching using prefetch cache memory.

Rew et al. [8] proposed NetCDF-4 [9] that combines the benefits of NetCDF, which provides an easy user interface for analysis and visualization of array-oriented scientific data with the HDF5 library, which supports irregular data, big datasets and parallel I/O. In the rest of this paper, parallel NetCDF refers to the NetCDF-4 parallel library. Optimization techniques like data buffering [10], auto-tuning [11], combining multiple I/O operations to reduce disk access [12], sub-filing [13], topology-aware I/O [14], data staging [14], data sieving and collective I/O [15] have been explored in previous works. Performance characterization of HPC systems is performed on various aspects including inter-process communication, interconnect technologies, parallel file systems, reading patterns, etc. Different I/O characterization tools including Darshan [16], output bottleneck characterization [17], continuous characterization tools [18] have been developed. Our work proposes to employ some of these techniques along with domain-specific I/O optimization strategy for improvement of I/O performance in the ROMS model.

III. Background

A. I/O Libraries in the ROMS Model

NetCDF [3] is a machine-independent I/O library that is useful to store and access high dimensional scientific data. It stores the metadata making it easy to share. It can scale for large datasets and we can read a subset of the whole data efficiently. Moreover, it provides diskless support by writing it to disk optionally and in-memory support. Multiple parameters like units, coordinates, dimensions can be stored as required by climate scientists. Ncview [19], a NetCDF visual browser provides a quick visualization of the NetCDF format data along different dimensions making the post processing process much easier. All these advantages make the NetCDF format well-suited for climate science applications.

NetCDF-4 is built on top of the HDF5 library [20]. The HDF5 library supports parallel I/O on NetCDF-4 files while PnetCDF supports parallel I/O on classic NetCDF files. In ROMS model, 2D and 3d variables are primarily written to output files in NetCDF format. Data can be added in the unlimited dimension settings as the simulation progresses. The variables have ocean-time as an unlimited dimension. It gets incremented when a new record is appended in the variables.

The Hierarchical Data Format v5 (HDF5) [20] I/O library supports high-dimensional complex scientific datasets, provides easy shareable data, supports multiple languages, provides quick access of subset of data and optimizes the storage space with different levels of compression. Any number of data objects can be written making it useful for the climate community when the number of records to be written is not known a-priori.

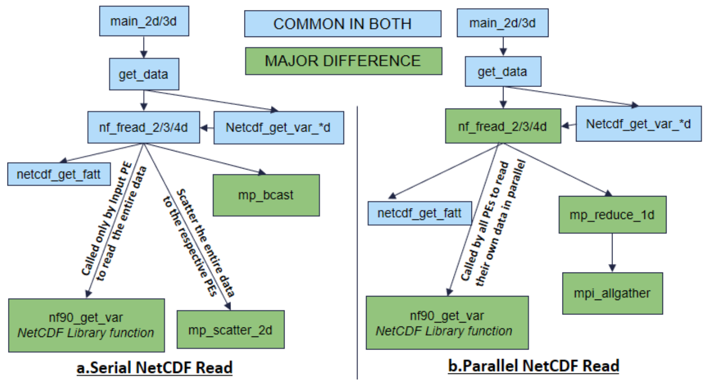
B. *Serial and Parallel I/O in ROMS*

In the ROMS model, both input and output data are in NetCDF format. The input data is read only once before the model starts to do computation whereas the output file is generated multiple times as per write frequencies set by the user. There are different types of output files in ROMS model. The output variables are mainly 2D and 3d variables that are written to these files. The 2D variable varies along latitude and longitude whereas the 3D variables vary along latitude, longitude, and depth. Both these types of variable have an extra dimension for record number while storing them into the NetCDF file. The primary variables that are written to the output files are velocity, temperature, and salinity. The output variables, the type, frequency and the number of records to be written in each file can be specified as per user requirements.

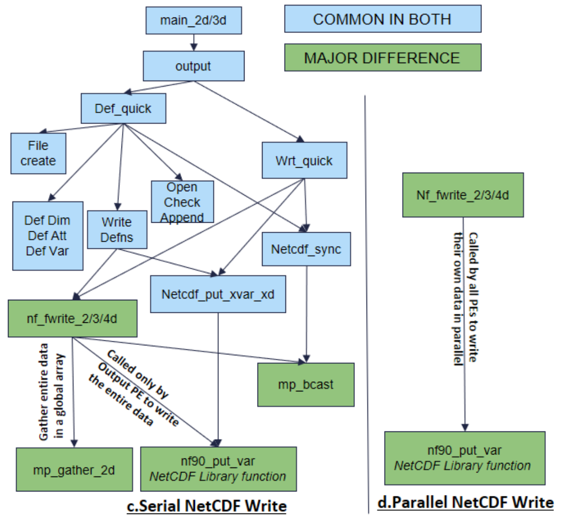
The function calls involved in the serial and parallel reading of NetCDF data is shown in Figure 2a. The similarities and differences are highlighted in blue and green, respectively. In serial NetCDF read, the requested data by all the processes are read only by the input thread and then distributed to the various processes as per their respective tiles using MPI scatter call. In parallel NetCDF read, all parallel threads read their own tile data, and then a global reduction is called using MPI all gather for the part of the data that is needed by all the processes.

The function calls involved in serial and parallel writing of NetCDF data is shown in Figure 2b. In serial NetCDF write, the complete data is packed into a global 1D array using MPI gather call from all the processes into the output thread, and then the data is written to the NetCDF file only by the output thread. The IO error flag is sent to all threads in case of serial NetCDF using MPI Broadcast call as all the processes will proceed for computation only after receiving the information that the output thread has written the complete data successfully in the output file.

In Parallel NetCDF write, all parallel threads write their own packed tile data. The output process involves defining output files (define phase) and then writing actual GRID data into those files (write phase). In the file define phase, all metadata operations like defining dimensions, attributes, and variables are performed. The variable units and the coordinates are also saved for each of them making the data self describing. It also involves writing the definition of time-independent variables. The define phase is done only for those steps in which new files need to be created. Otherwise, the previous files are opened, the variables are checked and the new record is appended at the end of the previous records. Thereafter, the write phase involves the writing of time-dependent variables at user-specified time-steps in different types of output files.



(a) Reading of Input files using serial and parallel NetCDF



(b) Writing of Output files using serial and parallel NetCDF

Fig. 2: Workflow Related to Reading and Writing in NetCDF

TABLE I: ROMS MODEL Setup

Grid Size	899*629*40
Time Step size	240 secs
Number of Time Steps	500 steps – 33.33 hours of simulation (500*240(step-size)/3600)

C. *Model and Experimental Setup*

We executed the ROMS model for the region of Ganga-Brahmaputra Plume in Bay of Bengal (BoB) [21]. The other parameters of the model are shown in Table I.

Most of our experiments are with 5-day simulations. The predictability of ocean state decreases with time. The accuracy of the predicted variables tend to be low after a period of about 5 days. The ocean model is forced by surface boundary conditions obtained from atmospheric forecast models. The accuracy of the ocean simulation depends on the accuracy of atmospheric forecasts. In general the accuracy of the atmospheric forecasts are good up to 3-days and reasonably good up to about 5 days. Therefore ocean model forecasts are in general useful for application for about 5 days.

All our experiments were conducted on our Institute's supercomputer cluster called SahasraT, a Cray-XC40 machine

TABLE II: Output file types, writing frequency and storage space (30 days simulation. Writing frequency refers to the interval between the timesteps when the corresponding file is written. e.g., Quick file is written every time step.)

File Types	Writing Frequency	Total Files	Per File Size in GB	Total File Size in GB
Quick	1	30	0.93	27.9
Average	30	1	1.1	1.1
History	30	1	1.1	1.1
Diagnostic	30	1	1.1	1.1

which has 1376 compute nodes. Each node has two CPU sockets with 12 cores each, 128GB RAM and connected using Cray Aries interconnect with Dragonfly topology. We used CrayPat [22] performance analysis tool to analyze the performance bottlenecks in ROMS. The results were obtained by averaging across 5 executions for each experiment.

As mentioned earlier, the quick output files store a snapshot of the present step into the file. In the ROMS model, it is found that out of all the output files, the quick files are written most frequently accounting for most of the I/O time as shown in Table II. Hence, the quick files are written in output for the experiments. Our results are also valid for other types of output files as the data format is the same for all files. Also, the same defining phase and writing phase functions are used in all types of output files. The intial dataset that is required to run the ROMS model is generated using Matlab scripts. Input data consists of 18 NetCDF files of a total of 18 GB.

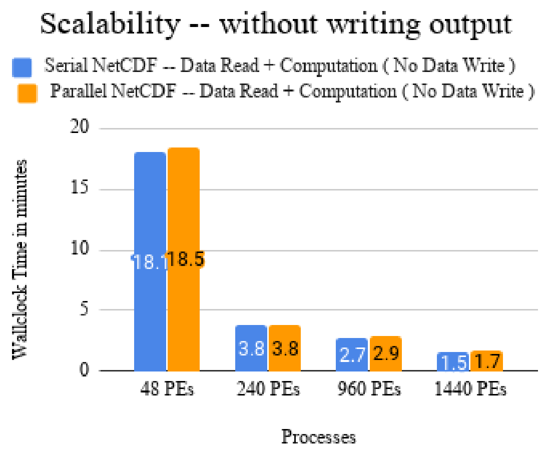
IV. I/O Analysis of the ROMS Model

A. Scalability and Performance of I/O

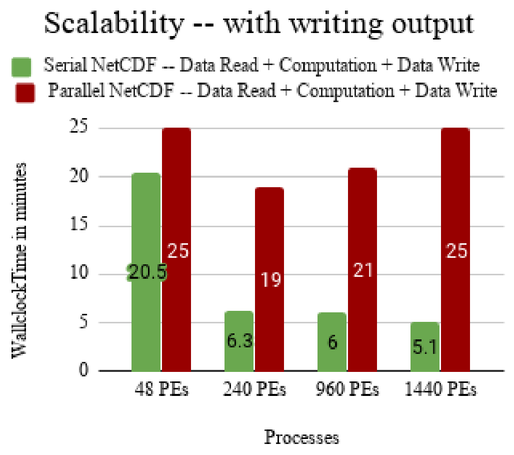
We first performed an experiment in which serial and parallel NetCDF are used for only reading the NetCDF data files and performing the computations. No output NetCDF file is generated in this experiment. It is found that both serial and parallel NetCDF scale up to around 1440 processes as shown in Figure 3a. Also, it is seen that parallel NetCDF read is taking more time compared to serial NetCDF. In both cases, the computation is performed in parallel and is scaling well.

Next, both serial and parallel NetCDF are used for both reading and writing the output files. 50 snapshots of data records are written over 500 time-steps to check the I/O performance. The variables in each record are written into a separate file. Hence, a total of 5 output files are generated in this experiment. It is found that the parallel model is scaling only up to 240 processes as shown in Figure 3b. Moreover, the parallel NetCDF times are found to be more than the serial NetCDF times.

From these two results, the times taken only for the writes are obtained and plotted in Figure 4a. As shown, the I/O time for writing the output files increases with the increase of the number of processes for both serial and parallel NetCDF as shown in Figure 4a. The serial NetCDF write is not scaling with the increasing number of processes as the data is transferred from a large number of processes to the master process that writes the data. However, contrary to expectations,



(a) Scalability of Application Involving Only Input Data Reading and Computations

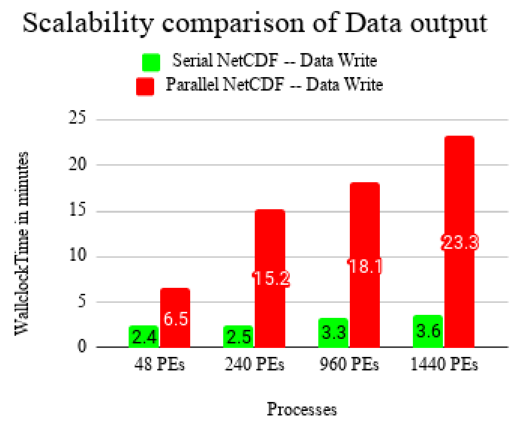


(b) Scalability of Application Involving Input Data Reading, Computations and Output Data Writing

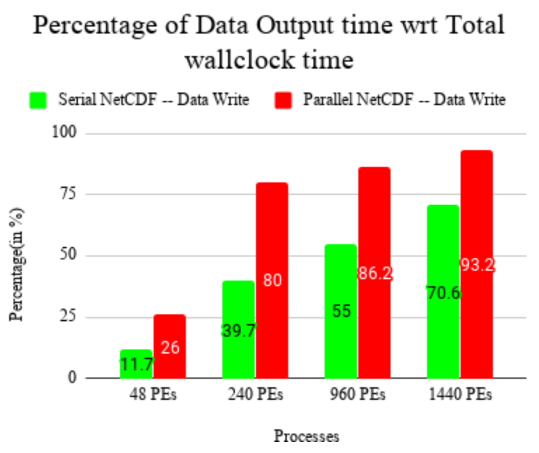
Fig. 3: Scalability of ROMS Model Without and With Output Writes

the results show that the NetCDF parallel is not scaling and taking more time than the NetCDF serial.

We also obtained results for percentage of data reading and data writing times with respect to the total execution time. We found that the data reading using serial NetCDF and parallel NetCDF takes only 0.5% and 2.4% of the total execution time. This is because the data is read only once at the beginning. The percentage of data writing times is shown in Figure 4b. As can be seen, data writes take around 25% of the total execution time for 48 processes and around 95% of the total execution time for 1440 processes in case of parallel NetCDF. The higher percentage of data write with respect to data read is because data is written many times as specified by the user. The increase in the percentage of data write with increasing number of processes is because of the good scalability of computations and the poor scalability of the data writes as shown in the previous figures. Hence, our first objective is to focus on the writing of output files as they consume a large fraction of the total execution time.



(a) Scalability of Times for Output Data writing for Different Number of Cores



(b) Percentage of Data Writing Times wrt Total Execution Times for Different Number of Cores

Fig. 4: Scalability and Percentage Time Consumption for Output Data Writing

B. Load Imbalance in Data Writes

The minimum and maximum CPU times among all processes and the average of the execution times are measured to check the load imbalance. We found that for serial NetCDF data reading as well as data writing, and parallel NetCDF data reading the execution time of all CPUs are almost similar. So the average execution time per CPU is close to the wall-clock time for such cases. However, when parallel NetCDF writing is enabled, there is a large execution time imbalance among different processes as shown in Figure 5. The figure also shows that the imbalance increased with the increase in the number of processes i.e. the difference between the minimum and maximum CPU timings increased with increasing processes. Hence, our next objective is to find the reason for this huge imbalance in the case of Parallel NetCDF.

V. Balancing I/O Loads

Load imbalance in I/O in many parallel applications arise due to non-uniform data distributions among processors. Hence, we first investigate the data distribution among the processors. Initially, the complete grid is partitioned into tiles among all the participating processes for parallel computations. It is observed that the data distribution is balanced among the processes initially as well as while writing data of both land and water data points. Also, in case only water data points are written, the land data points are removed (Blue shade represents the water points after removal of land points) and the water data points are re-distributed among the processes again before writing to re-balance the data (Green shade represents the re-distributed water points). Hence, it is confirmed that the imbalance is not due to data distribution.

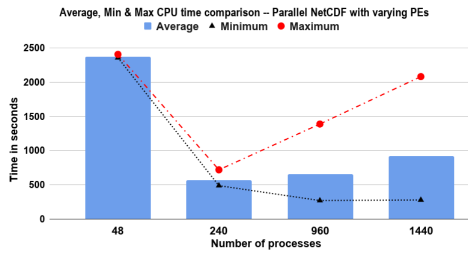


Fig. 5: Execution Times for Different Number of Processors when Parallel NetCDF is Enabled. The large difference between the minimum and maximum times among the processors for a particular Execution demonstrates the load imbalance among processors.

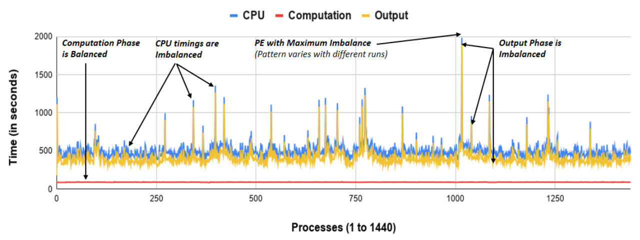


Fig. 6: Execution Times of Different Processors for a 1440-core Execution. The graph shows the Computation, CPU, Output Times. We find that the CPU and Output Times are Imbalanced.

A detailed analysis of the function calls and the per-process timings for such functions is performed to investigate the reason for load imbalance. Figure 6 shows the execution for a 1440-process execution. The total CPU time (in blue) is the sum of computation including I/O read time (in red) and Output time (in yellow) as shown in the figure. It is observed that the computation including I/O read time is balanced while the output times are imbalanced. The processors responsible for the imbalance can be seen as the spikes in the plots.

The output phase further consists of two phases: define file and data write phase. In the define phase, the NetCDF file is created and all the variables are defined. In the write phase, the data for the time varying variables that are defined during define phase is written into the file. It is found that the write phase is load-balanced whereas the define file times are load imbalanced.

The define phases involves either $Define\_Info$, that involves creating a NetCDF file, defining dimensions, attributes, and variables, and $Write\_Info$, that involves writing time-recordless, information variables. It is found that the $Write\_Info$ phase is load imbalanced.

The non-tiled variable function calls 1 to 66 in the define phase are defined as independent (default setup for parallel NetCDF calls) and the tiled-variable function calls 67 to 83 in define phase and all function calls in write phase are defined as collective. The independent I/O calls lead to file locking for the portion where a process sends a write request in a file. In case of non-tiled variables, as all processes involved in parallel NetCDF call are sending an independent write request to the same portion of the file, there is I/O contention due to file locking during multiple such write requests leading to imbalance. As a result, the imbalance was found to increase with an increasing number of processes. But for tiled-variables, all processes involved in parallel NetCDF call are sending I/O write requests collectively to different portions of the variable in the file. This leads to a single collective call request to the file, merging separate requests as the entire grid data is written by all the processes making the I/O write efficient for all tiled-variable collective calls.

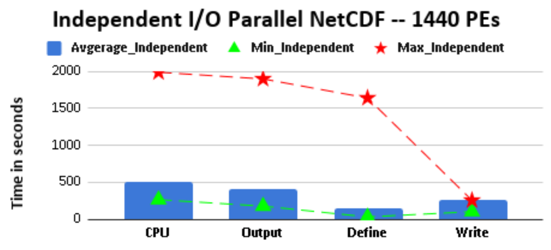
Further analysis of the amount of imbalance coming from each of the 66 function calls (which were defined independently by default) are studied to check the contribution by different functions. It is found that 7 different types of functions are involved. Barrier is added at the start and end of those functions to measure their imbalance contributions. It is observed that each of the functions contributes imbalance proportional to the number of times they are called.

Hence, the primary source of imbalance is in the writes of the non-tiled variables ($Write\_Info$) during the define phase. We removed the imbalance by converting the independent writes to collective writes. The reduction of imbalance is shown in Figure 7.

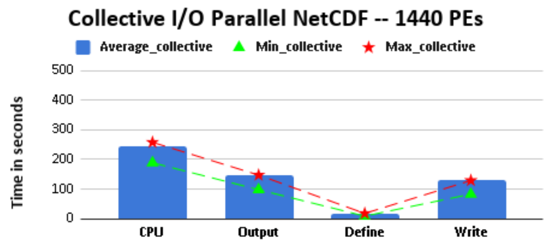
A. Performance and Scalability Results

Significant improvement is achieved for 1440 processes after changing the independent calls to collective calls of parallel NetCDF. The improvement in timings of parallel NetCDF (collective calls) wrt parallel NetCDF (independent calls) and serial NetCDF are shown in Figure 8. Each of the times shown in the figure are obtained as averages across 5 runs.

It is observed that parallel NetCDF with collective calls (our method) gave 85% performance improvement when compared to parallel NetCDF with independent calls (default Setup) and 27.3% improvement when compared to serial NetCDF in the ROMS model. The maximum output times wrt the maximum CPU times has come down from 91% in default parallel NetCDF and 70.5% in serial NetCDF to 62.4% in our method in the ROMS model. The bottleneck was because the define phase itself was taking 9.75 minutes which is even more than write phase of 1.84 minutes in the case of default setup of parallel NetCDF. With the new setup, the define phase for parallel NetCDF has come down to 0.34 minutes.



(a) Execution Times of Different Phases with Independent I/O Calls in Parallel NetCDF for a 1440-core Execution. The figure shows the average, minimum and maximum times across processors. The large difference between the maximum and minimum times shows the load imbalance when using independent I/Os.



(b) Execution Times of Different Phases with Collective I/O Calls in Parallel NetCDF for a 1440-core Execution. The small difference between the maximum and minimum times shows the load imbalance when using independent I/Os.

Fig. 7: Reduction in Load Balance by Making the Independent I/O Calls as Collective

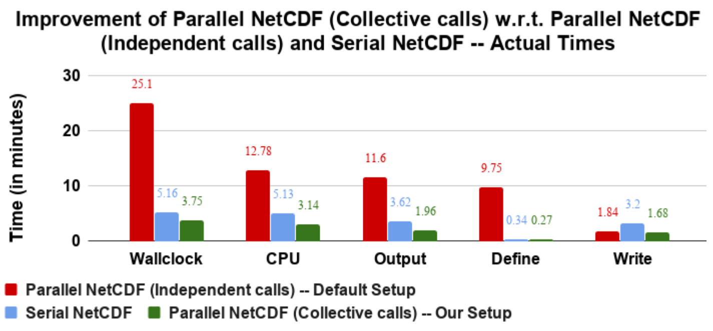


Fig. 8: Execution Times of the ROMS Model when using Serial NetCDF, and Parallel NetCDF with Independent and Collective I/O Calls for a 1440-core Execution. We find that the execution times significantly decrease when using collective I/O calls.

The scalability results achieved with parallel NetCDF collective calls (mentioned as Par_COL) are compared with that of parallel NetCDF independent calls i.e. default setup (mentioned as Par_IND) and serial NetCDF (mentioned as Serial). The comparisons are shown in Figure 9. It is seen that the execution, output and write times for Par_COL scale well unlike Par_IND. Also, Par_COL gives smaller execution times than Serial for 960 and 1440 processes. The define phase imbalance increases for Par_IND with more number

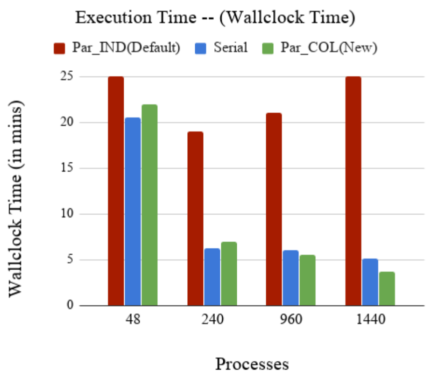


Fig. 9: Scalability in Execution Times for Different Number of Processors when using Independent and Collective I/O Calls

of processes as there is an increase in I/O contention with increasing independent I/O requests. The serial output does not scale as data needs to be gathered from more processes. It is seen that the benefits of parallel NetCDF are achieved in case of a large number of processes.

VI. Selective Writing Strategy

The output data in climate modeling have spatial and temporal variability. An example of spatial variability is eddies where the variation in data is most important. Temporal variability can be hourly, daily, monthly, intraseasonal (15 to 60 days), seasonal(180 days), annual(1 year), or interannual(multiple years). For a given writing frequency for some parameters, there are some time steps where the variation in data from the last written step and the present time-step is small compared to other time-steps. In that case, we can avoid writing the present step data again and use the previously written data for analysis to be performed on the present time-step. This strategy can be implemented for those simulations where some compromise on the accuracy loss is acceptable. Thus, the objective of selective writing strategy is to keep the accuracy loss within the tolerable limits to improve the overall I/O performance of climate models.

In the ROMS model, the time-consuming variables are the 3D variables. In our model setup, each variable size is 40 times of the 2D variables. The main parameters written to output files are velocity, temperature and, salinity. Hence, these parameters are used for the selective writing strategy. The RMS (Root Mean Square) difference for different parameters between consecutive records is used for determining the parameters for selective writing in a time step.

A. Implementation

Before writing the data, RMS difference between the present time-step and the last written time-step is compared for each variable separately. If the RMS error is above a certain threshold set for a variable, then the data corresponding to that variable is written or else skipped for that time step. The skipped time-step values for each variable are approximated with the last time-step values written. A separate array is maintained to store the final processed data of the last time-step that was written to disk, using the SAVE option in FORTRAN. The array is updated in a time-step if the data is written to the disk in the time-step. The strategy of selective writing of parameters in the time steps based on RMS error is illustrated in Figure 10.

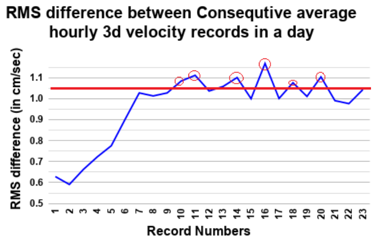


Fig. 10: Illustration of Selective Writing Strategy. Output files are written for only the time steps or record numbers corresponding to the marked circles in the figure.

TABLE III: Selective Writing Thresholds used for different parameters

Variable	Thresholds
Velocity	1, 1.5, 2, 2.5, 5 cm/sec
Temperature	0.02 , 0.025 and 0.05 Celsius
Salinity	0.0005 , 0.00075 and 0.001 ppt

B. Evaluation of Accuracy

The selective writing strategy will generate different output from the original executions since the data of the previous time steps are used for variables that are skipped for writing. We evaluate the acceptable accuracy of data generated by the selective writing strategy using *temporal mean and standard deviation* (S.D.) of the 3D variables. The RMS difference between temporal mean (similarly standard deviation) with selective writing and original executions is calculated to check the accuracy loss due to the selective writing strategy. The quality of data is also evaluated from the *surface contour plots* of the difference between the temporal means (similarly Standard Deviation). Smaller differences indicate better accuracy.

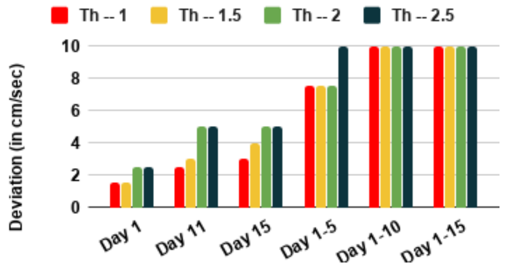
C. Experimental Setup

We executed the ROMS model on 1440 cores for 15 simulated days. In the original executions, write outputs (records) are generated at the end of every simulated hour. The accuracy loss and write reduction analyses are performed both on a daily basis (Day 1, Day 11 and Day 15) and on multiple days (Day 1 to 5, Day 1 to 10, and Day 1 to 15). Sensistivity studies were performed for different threshold values for the Velocity, Temperature and Salinity variables as shown in Table III.

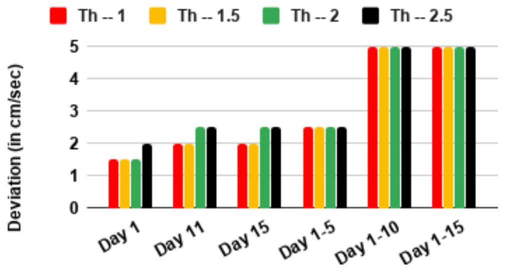
Based on the domain knowledge related to ocean simulations with the ROMS model, the range of variation of each parameter and the corresponding range of acceptable loss in accuracy were fixed as mentioned in Table IV.

TABLE IV: Range of variation and acceptable accuracy loss for different parameters for Selective Writing

Variable	Range	Acceptable Deviation
Velocity	+/- 100 cm/sec	+/- 7.5 cm/sec
Temperature	20 to 35 Celsius	0.05 Celsius
Salinity	25 to 35 ppt	0.005 ppt



(a) Accuracy Loss in Mean



(b) Accuracy Loss in S.D.

Fig. 11: Accuracy Loss for Velocity Values due to Selective Writing

D. Results and Observations

1) Sensitivity Studies with Different Thresholds: Figure 11 shows the sensitivity studies conducted with different thresholds for the Velocity variable to determine thresholds with acceptable accuracy loss.

We find that the threshold of 1.5 cm/sec for velocity limits the difference in temporal mean and S.D. to less than 5 cm/sec for single day analysis (Day 1, 11 and 15) and 7.5cm/sec for 5 days analysis. However, a steep jump is observed in accuracy loss for Day 11 analysis for changing the threshold from 1.5 to 2 cm/sec. Hence, 1.5 cm/sec is selected as the threshold for velocity.

We performed similar sensitivity studies to determine the threshold parameters for Temperature and Salinity. For Temperature, we observed that all thresholds meet the difference in temporal means constraint of 0.05 Celsius. However, only threshold of 0.02 Celsius meets the constraint of difference in temporal S.D. of 0.05 Celsius for up to 15 days. Hence, 0.02 Celsius is selected as the threshold for temperature. For Salinity, For Salinity, we observed that all thresholds meet the difference in temporal means constraint of 0.005 ppt. However, up to a threshold of 0.00075 ppt meets the constraint of difference in temporal S.D. of 0.005 ppt for up to 10 days. Hence, 0.00075 ppt is selected as the threshold for salinity.

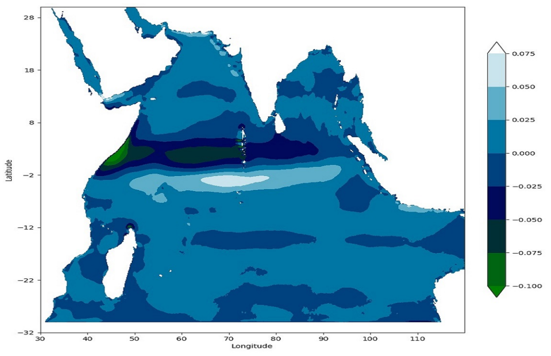


Fig. 12: Difference in Temporal Mean in Velocity

2) Contour Plots: The surface contour plots of differences in temporal mean of data without and with selective writing for velocity is shown in Figure 12. Similar plots were also obtained for temperature and salinity.

From the velocity plot, it was observed that some specific regions of the temporal means have maximum velocities. The difference in temporal means is also found maximum in those regions. These regions are primarily constrained within the acceptable deviation of 7.5 cm/sec using a threshold of 1.5 cm/sec. The difference in temporal S.D. is within 2.5 cm/sec for velocity. For temperature and salinity, the variation is not as high as velocity. Both temporal mean and S.D. for temperature are within the acceptable deviation of 0.05 Celsius for a threshold of 0.02 Celsius. Similarly, both temporal mean and S.D. for salinity are within the acceptable deviation of 0.005 ppt for a threshold of 0.00075 ppt. Hence, temperature and salinity can be written selectively for 15 days and 10 days respectively due to lesser variation in data whereas velocity can be written for only up to 5 days.

It is concluded that velocity can be written selectively for hourly data for up to 5 days whereas salinity and temperature can be written selectively for hourly data for up to 10 days and 15 days respectively. This is because both spatial and temporal variation of temperature and salinity is found to be less compared to velocity. Also, maximum variation is present at the surface (2-D). Hence, all the 2D variables are chosen for writing.

3) I/O Performance Improvement: Experiments are performed for 5 days of simulation as it is the common acceptable length of executions with selective writing in all variables. The experiments are done with the load balance ROMS Model obtained by replacing independent calls by collective calls. The improvements obtained with these thresholds for a run of 5 days are mentioned in Table V.

Around 60% improvement is achieved in the write time of 3d variables using this method. It contributed to almost 52% improvement in the output time and 31.6% improvement in the execution time.

E. Summary and Observations on Selective Writing

Typically, ocean prediction systems yield reasonable forecasts for up to 5 days. Hence, based on the acceptable range of

TABLE V: Selective Writing Time Reduction Results for 5 days of run

Type	Execution Time	Output Time	$nf\_fwrite\_3dfunction$
Non-Selective i.e. Default	24.36 mins	14.2 mins	12.98 mins
Selective (Reduction in %)	16.66 mins (31.6%)	6.83 mins (51.87%)	5.26 mins (59.43%)

accuracy loss and the results, it is concluded that our selective writing strategy is suitable for ocean forecast systems. Also, it is concluded that the daily means and S.D. are acceptable for Day 1, 11, and 15 daily basis results. So, this method can be used for experiments where analysis is done based on daily data.

The advantage of our selective writing strategy is that it can help accelerate ocean simulations, as shown in our results. However, the accuracy loss can increase for an increase in the number of days as the variability in the data increases. Also, it is found that the method is applicable for experiments which store hourly data. But for daily data (data saved once in 24 hours), the performance degrades as the variability in the data is more. In general, the appropriate threshold for each variable needs to be selected based on the experimental accuracy required and the length of the run.

VII. Lustre Stripe Size and Stripe Count

The Lustre file system consists of I/O servers called Object Storage Servers(OSSs) for managing the I/O requests and disks called Object Storage Targets(OSTs) for storing the data. File striping [23] is a technique to distribute the data of a single file across multiple OSTs. The striping helps in increasing the bandwidth to read or write to a file and increases the upper limit of file size also. But it also leads to extra overhead due to increased operations in network and contention in I/O servers. So there is a trade-off involved. Hence, optimizing the Lustre parameters helps in the improvement of the I/O performance. The I/O performance is improved when processes access multiple OSTs in parallel. Also, the number of OSTs, which each process needs to access should be minimized by stripe alignment.

We performed experiments on different sets of processes to find the best lustre striping for different number of processes. The Lustre system in our experiments has capacity 2 PB and its version is 2.12.0.2. It consists of 16 OSTs with the default stripe size of 4MB and stripe count of -1, with all OSTs used. The stripe count is varied from 1 to the maximum available OSTs and the stripe size is varied from 1 to 32 Mb. 3D variables are used as they consume most of the write times. A total of 64 experiments were performed, as shown in the experimental setup in Table VI.

The write timings obtained for default setup (stripe count of 4 and stripe size of 1Mb) and for the best combinations (Table VII) are shown in Figure 13. About 40-60% performance improvement is seen with the best combinations compared to

TABLE VI: Lustre Striping Setup

Processes	48, 240, 960, 1440
Stripe Count	1 OST, 4 OSTs , 16 OSTs, 96 OSTs(set as -1 for max)
Stripe Size	1Mb, 4Mb, 16Mb, 32Mb
Number of records	20 records each with 5 3D variables (100 3D write requests)

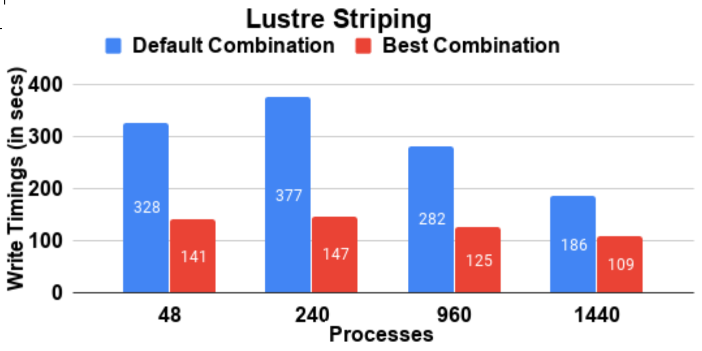


Fig. 13: Write Timing improvement results using Lustre Striping [Default Combination is Stripe Size 1 Mb , Stripe Count 4 OSTs and for Best Combination refer Table VII]

the default for all different combinations of processes used. It is observed that for most of the processes, the max stripe count with a stripe size of 1Mb performed th best.

TABLE VII: Lustre Striping – Best Combinations of Stripe Size and Stripe Count

Processes	Stripe Size	Stripe Count
48	32Mb	1
240	1 Mb	96
960	1 Mb	96
1440	1 Mb	96

A. Observations

Serial I/O does not scale with an increase in the number of OSTs as a single process writes in all of them in sequence. For Parallel I/O with a small number of processes, the stripe count should be 1 with a large stripe size. For large number of processes, the creation of large files with low stripe count can lead to I/O bottleneck. For a single file shared by all processes, the stripe count should be the same as the number of processes. If the number of processes is greater then the maximum OSTs available (96 in our Lustre file system), then it should be set to -1 (maximum 96) and the stripe size should be set to achieve maximum stripe alignment distributing the data uniformly across all OSTs.

B. Putting It All Together

The different I/O improvement strategies are combined. The experimental setup is the same as the selective writing strategy result for 5 days of simulation present in Table V. As mentioned earlier, the non-selective data writing and the selective data writing experiments shown previously are

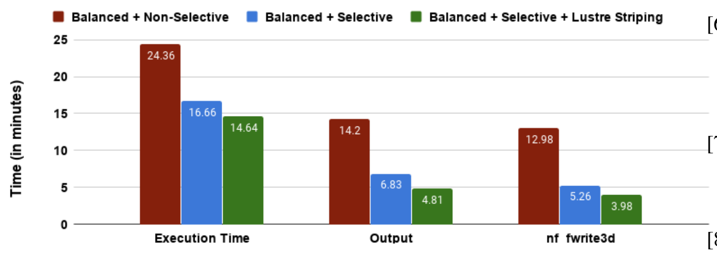


Fig. 14: Combined improvement results – Actual Times

already implemented on top of the balanced ROMS model obtained by the load balancing strategy using collective calls for the imbalanced functions. Here, the third strategy of lustre striping is combined with the load balanced model with selective writing strategy. Figure 14 shows the results. It is observed that the lustre striping improved the writing phase by 10% and output phase by 15%.

VIII. Conclusions and Future Work

An efficient parallel I/O strategy can help us improve the overall performance of climate models as I/O consumes a significant part of the total execution time for these models. In this work, we had developed two main I/O optimization strategies, namely, load balanced I/O writes and selective writing of slow-varying parameters. We also explored I/O striping. All the strategies together improved parallel NetCDF performance by about 70%. A similar approach can be adopted in case load imbalance is detected from the initial analysis for other climate models that deal with large number of large multi-dimensional arrays. The selective writing can be implemented in other models where a large volume of I/O output is involved and performance-accuracy tradeoffs can be made. The striping technique can also be implemented for other file systems for achieving similar performance improvements. In future, we plan to develop methods to perform selective writing strategy on a region basis, where selective writing can be applied to regions where change is minimal. Implementation of collective I/O in both space and time can also be explored. We also plan to study topology-aware I/O strategies. Other I/O optimizations including data layout and asynchronous I/O are possible and are part of our future efforts. While asynchronous I/O is generally considered a good strategy, its benefits will have to be evaluated in the ROMS code wrt the dependencies between the computations and communications of large number of large multi-dimensional arrays.

References

[1] Wikiroms. [Online]. Available: https://www.myroms.org/wiki/Documentation_Portal

[2] Esmf. [Online]. Available: https://www.earthsystemcog.org/projects/esmf/

[3] Netcdf. [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/

[4] Z. e. a. Liu, "Profiling and Improving I/O Performance of a Large-scale Climate Scientific Application," in *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, 2013, pp. 1–7.

[5] Geos. [Online]. Available: https://gmao.gsfc.nasa.gov/systems/geos5/

[6] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible I/O and Integration for Scientific Codes through the Adaptable I/O System (ADIOS)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, 2008, pp. 15–24.

[7] H. Tang, X. Zou, J. Jenkins, D. A. Boyuka, S. Ranshous, D. Kimpe, S. Klasky, and N. F. Samatova, "Improving Read Performance with Online Access Pattern Analysis and Prefetching," in *European Conference on Parallel Processing*, 2014, pp. 246–257.

[8] R. K. Rew, B. Ucar, and E. Hartnett, "Merging NetCDF and HDF5," in *20th Int. Conf. on Interactive Information and Processing Systems*, 2004.

[9] M. J. Folk, R. Rew, M. Yang, E. Hartnett, R. E. McGrath, and Q. Koziol, "NetCDF-4: Combining NetCDF and HDF5 Data," in *AGU Fall Meeting Abstracts*, 2003.

[10] W. W. Dai, A. J. Scannapieco, F. L. Cochran, C. Chang, P. M. Weber, E. L. Sandford, and B. A. Girard, "Buffering I/O for Data Management in Multi-Physics Simulations," in *Proceedings of the High Performance Computing Symposium*, 2013, pp. 1–8.

[11] B. Behzad, J. Huchette, H. Luu, R. Aydt, Q. Koziol, M. Prabhat, S. Byna, M. Chaarawi, and Y. Yao, "Auto-tuning of Parallel I/O Parameters for HDF5 Applications," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 1430–1430.

[12] K. Gao, W.-k. Liao, A. Choudhary, R. Ross, and R. Latham, "Combining I/O Operations for Multiple Array Variables in Parallel NetCDF," in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–10.

[13] K. Gao, W.-k. Liao, A. Nisar, A. Choudhary, R. Ross, and R. Latham, "Using Subfiling to Improve Programming Flexibility and Performance of Parallel Shared-file I/O," in *2009 International Conference on Parallel Processing*. IEEE, 2009, pp. 470–477.

[14] Y. Tsujita, A. Hori, T. Kameyama, A. Uno, F. Shoji, and Y. Ishikawa, "Improving Collective MPI-IO Using Topology-aware Stepwise Data Aggregation with I/O Throttling," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2018, pp. 12–23.

[15] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *Proceedings. Frontiers' 99. Seventh Symposium on the Frontiers of Massively Parallel Computation*, 1999, pp. 182–189.

[16] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, 2016, pp. 9–17.

[17] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing Output Bottlenecks in a Supercomputer," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

[18] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and Improving Computational Science Storage Access through Continuous Characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, pp. 1–26, 2011.

[19] Netcdf. [Online]. Available: \url{http://meteora.ucsd.edu/~pierce/ncview_home_page.html}

[20] Hdf5. [Online]. Available: https://www.hdfgroup.org/

[21] S. M. Pargaonkar and P. N. Vinayachandran, "Wind Forcing of the Ganga-Brahmaputra River Plume," *Ocean Dynamics*, vol. 71, no. 2, pp. 125–156, Feb. 2021.

[22] Craypat. [Online]. Available: https://docs.nersc.gov/programming/performance-debugging-tools/craypat/

[23] Lustre. [Online]. Available: https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips