

Analysis and Improvement of I/O Performance of large-scale Climate Scientific Application

A PROJECT REPORT
SUBMITTED FOR
THE DEGREE OF
Master of Technology
IN
Computational and Data Sciences

BY
Koushik Sen



Computational and Data Sciences
Indian Institute of Science
Bangalore – 560 012 (INDIA)

July, 2020


Declaration of Originality

I, **Koushik Sen**, with SR No. **06-18-01-10-51-18-1-15804** hereby declare that the material presented in the thesis titled "**Analysis and Improvement of I/O Performance of large-scale Climate Scientific Application**" represents original work carried out by me in the **Department of Computational and Data Sciences** at **Indian Institute of Science** during the years **2018-2020**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: July,2020


Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Dr. Sathish Vadhiyar

Advisor Signature

© Koushik Sen
July, 2020
All rights reserved

DEDICATED TO

The Student Community

who can use and reuse this template to glory

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Sathish S. Vadhiyar, for his continuous guidance and immense support throughout the project. He always had one-one meetings almost every week to discuss the ideas and the progress despite his busy schedule. It helped me a lot to keep myself motivated and constantly engaged with the research work. I am thankful to the members of MARS Lab for their useful advice, support, and valuable discussions.

I am grateful to Prof.P.N.Vinayachandran and his student Shrikant from Centre for Atmospheric and Oceanic Sciences (CAOS) department for their valuable suggestions and continuous support. They played an important role in the completion of my project. I am thankful to Hernan G. Arango from Rutgers University, administrator of the ROMS model for his valuable insights. I am also thankful to Prasanthi from the Cray team of the SERC department for her quick support in case of any issues throughout the project.

I am thankful to the Department of Computational and Data Sciences, SERC team, and MARS Lab for providing me the high-quality research facilities required for my project. I am thankful to my batch mates for their unconditional support. I am thankful to everyone who helped me directly or indirectly for this project.

Last but not least, I am thankful to my parents, my sister, my teachers, and my dear friends for their continuous encouragement and moral support.

My two years of IISc journey made me realize that there is much more to learn ahead. I will cherish my IISc life forever.

Abstract

There is a huge gap between computational power and I/O performance of HPC systems. The computational power is improving at a much faster rate as compared to the overall I/O performance of HPC systems. Also, the HPC applications are becoming more data-intensive primarily dominated by I/O writes. Many large-scale scientific applications do not adequately optimize the I/O operations. Hence, it leads to overall poor performance for the models in which a lot of I/O is involved. So, there is a need to analyze the I/O operations in large scale scientific applications to achieve better scalability and to improve the overall I/O performance provided by the underlying parallel HPC file systems. A detailed study is required to identify and understand the I/O bottlenecks present in serial and parallel reading and writing of scientific data. They need to be addressed properly to improve the I/O which will help to improve a significant amount of the total execution time. Also, there is a need for efficient technique to manage complex access patterns which can be high-dimensional strided or composition based unstructured patterns for reading the data efficiently from the underlying file systems.

Regional Ocean Modeling System (ROMS) model is used for analyzing the I/O performance issues for all experiments. Based on the analysis results, the improvement strategies are implemented in the ROMS model for improving the I/O performance. Performance improvement of around 85% on execution time is achieved by solving the load imbalance that was detected in the ROMS model for parallel I/O from the imbalance studies. The second strategy of selective data writing strategy implemented on top of load balanced ROMS model helped to achieve around 31% reduction in execution time. The third strategy of Lustre striping gave around 41% reduction in the I/O write time. The load balanced ROMS model with selective writing strategy and lustre striping together gave 40% reduction in execution time. All three strategies combined together improved the parallel NetCDF performance by about 60% with respect to serial NetCDF setup. So, all the above three strategies contributed significantly for improving the I/O performance. Hence, the overall execution time of the ROMS model is reduced.

Contents

Acknowledgements	i
Abstract	ii
Contents	iii
List of Figures	v
List of Tables	viii
1 Introduction	1
1.1 An Introduction	1
1.2 Organization	4
2 Related Work	5
3 Work Background	7
3.1 The Parallel Platform	7
3.2 ROMS Model – Overview	7
3.3 ROMS Model – I/O Strategy	8
3.3.1 Serial and Parallel NetCDF Data Reading mechanism	8
3.3.2 Serial and Parallel NetCDF Data Writing mechanism	9
3.4 Experiment Setup	10
3.4.1 Input Dataset Used	12
3.5 Overall Methodology for I/O analysis	13
3.6 Scalability Studies	14
3.6.1 Strong Scaling Results - file read and computation only	14
3.6.2 Strong Scaling Results - file read, computation and file write	14

CONTENTS

3.6.3	Strong Scaling Results - file write only	15
3.6.4	Input and Output percentages wrt total execution time	16
3.7	Load Imbalance Studies	17
3.8	Bottleneck Study	18
3.8.1	Using CrayPAT Tool	18
3.8.2	MPI communications	18
4	I/O improvement	20
4.1	Overall Methodology for I/O improvement	20
4.2	Load Imbalance Studies	21
4.2.1	Experimental Setup	21
4.2.2	Data Distribution	21
4.2.3	Imbalance Flow	22
4.2.4	Bottleneck Function Studies	25
4.2.5	Performance Improvement Results	27
4.2.6	New Scalability Results	29
4.3	Selective Writing Approach	31
4.3.1	Motivation	31
4.3.2	Initial Analysis – RMS error comparison for different output variables . .	31
4.3.3	Our Approach	33
4.3.4	Accuracy Evaluation Strategy	33
4.3.5	Experimental Setup	34
4.3.6	Results and Observations	35
4.3.7	Conclusion	44
4.4	Lustre Stripe Size and Stripe Count	44
4.4.1	Experimental Setup	45
4.4.2	Results and Observations	45
4.4.3	Conclusion	46
4.5	Combining the different I/O optimization techniques	47
5	Conclusions and Future Works	48
5.1	Conclusions	48
5.2	Future Works	49
	Bibliography	50

List of Figures

1.1	ROMS model	1
1.2	ROMS Model I/O Framework and Partitioning	2
1.3	NetCDF Output Variables Note: <i>ocean</i> time is defined as UNLIMITED	3
3.1	Reading of Input files using serial and parallel NetCDF	8
3.2	Writing of Output files using serial and parallel NetCDF	9
3.3	Initial Input Data using MATLAB	12
3.4	Intermediate steps	12
3.5	Final Input Data	13
3.6	Scalability – Input Data Reading + Computation	14
3.7	Scalability – Input Data Reading + Computation + Output Data writing	15
3.8	Scalability – Output Data writing [Fig 3.8 = Fig 3.7 - Fig 3.6]	15
3.9	Percentage of Data Read and Write times wrt the total execution time	16
3.10	Load Imbalance Study	17
3.11	MPI Communications – Serial Read + Computation	18
3.12	MPI Communications – Serial Read + Computation + Serial Write	18
3.13	MPI Communications – Parallel Read + Computation	19
3.14	MPI Communications – Parallel Read + Computation + Parallel Write	19
4.1	Data Distribution among 1440 processes(Only 750 to 1000 processes are shown for better clarity of the data points in each process)	21
4.2	CPU (<i>Imbalanced</i>) = Computation (<i>Balanced</i>)+ Output (<i>Imbalanced</i>)	22
4.3	Output (<i>Imbalanced</i>) = Define Files (<i>Imbalanced</i>) + Write into files (<i>Balanced</i>)	22
4.4	Imbalance Flow Diagram	23
4.5	Min, Max and Averages times comparison for 1440 Processes CPU = Computation + Output and Output = Define + Write	24
4.6	Independent functions	25

LIST OF FIGURES

4.7	Collective functions	26
4.8	Bottleneck in the 7 different functions	26
4.9	Imbalance Contribution by different NetCDF Functions	27
4.10	Improvement Results after solving imbalance for 1440 processes (Actual Times)	27
4.11	Non-overlapping CPU time imbalance contributed by different processes	28
4.12	Improvement Results after solving imbalance for 1440 processes(in percentage) .	28
4.13	Scalability in Execution	29
4.14	New Scalability Results for our parallel NetCDF new setup (with collective calls) and comparison wrt parallel NetCDF default setup (with independent calls) and serial NetCDF – for 48,240,960 and 1440 processes	30
4.15	RMS difference comparison of different parameters	32
4.16	RMS Thresholding	32
4.17	Selective writing approach	33
4.18	Temporal Mean and S.D. calculation and Surface Plotting Illustration	34
4.19	Selective Writing – Reduction Gained versus Accuracy Lost – VELOCITY . . .	36
4.20	Selective Writing – Reduction Gained versus Accuracy Lost – TEMPERATURE	37
4.21	Selective Writing – Reduction Gained versus Accuracy Lost – SALINITY	38
4.22	Results – Accuracy Loss in VELOCITY with the selected Threshold of 1.5cm/sec for 5 days of run	40
4.23	Results – Accuracy Loss in TEMPERATURE with the selected Threshold of 0.02 Celsius for 5 days of run	41
4.24	Results – Accuracy Loss in SALINITY with the selected Threshold of 0.00075 ppt for 5 days of run	42
4.25	Selective Writing Accuracy Evaluation Illustration LEFT – Plot a (similarly c) is the surface contour plot of differences in Temporal Mean (similarly Temporal S.D.) of data without selective writing i.e. Th 0cm/sec and with selective writing of Th 2 cm/sec, RIGHT – Plot b (similarly d) is the surface contour plot of differences in Temporal Mean (similarly Temporal S.D.) of data without selective writing i.e. Th 0cm/sec and with selective writing of Th 5 cm/sec. [Difference should be close to zero for better accuracy. As the threshold increases, accuracy drops]	43
4.26	Physical and Logical views of each write request of 3d variables (86.65 Mb) for stripe size of 1 Mb and Stripe count of -1 (i.e. maximum 96 in CRAY Lustre file system)	45

LIST OF FIGURES

4.27	Write Timing improvement results using Lustre Striping [Default Combination is Stripe Size 1 Mb , Stripe Count 4 OSTs and for Best Combination refer Table 4.8]	46
4.28	Combined improvement results – Actual Times	47
4.29	Combined improvement results – In percentage	47

List of Tables

3.1	The Cray XC40 system	7
3.2	Experimental Setup table	11
3.3	Domain Partitioning table	11
3.4	Output file types, writing frequency and storage space (30 days simulation) . .	11
3.5	ROMS MODEL Setup table	11
4.1	Experimental Setup for I/O Imbalance Improvement	21
4.2	Experimental Setup for Selective Writing	34
4.3	Thresholds used for different parameters	35
4.4	Range of variation and acceptable accuracy loss	35
4.5	Threshold selected and allowed length of run using selective approach	39
4.6	Selective Writing Time Reduction Results for 5 days of run	39
4.7	Lustre Striping Setup	45
4.8	Lustre Striping – Best Combinations of Stripe Size and Stripe Count	45

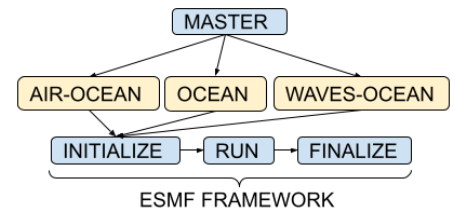
Chapter 1

Introduction

This chapter introduces the technical terms and models used in this work and the organization of subsequent chapters.

1.1 An Introduction

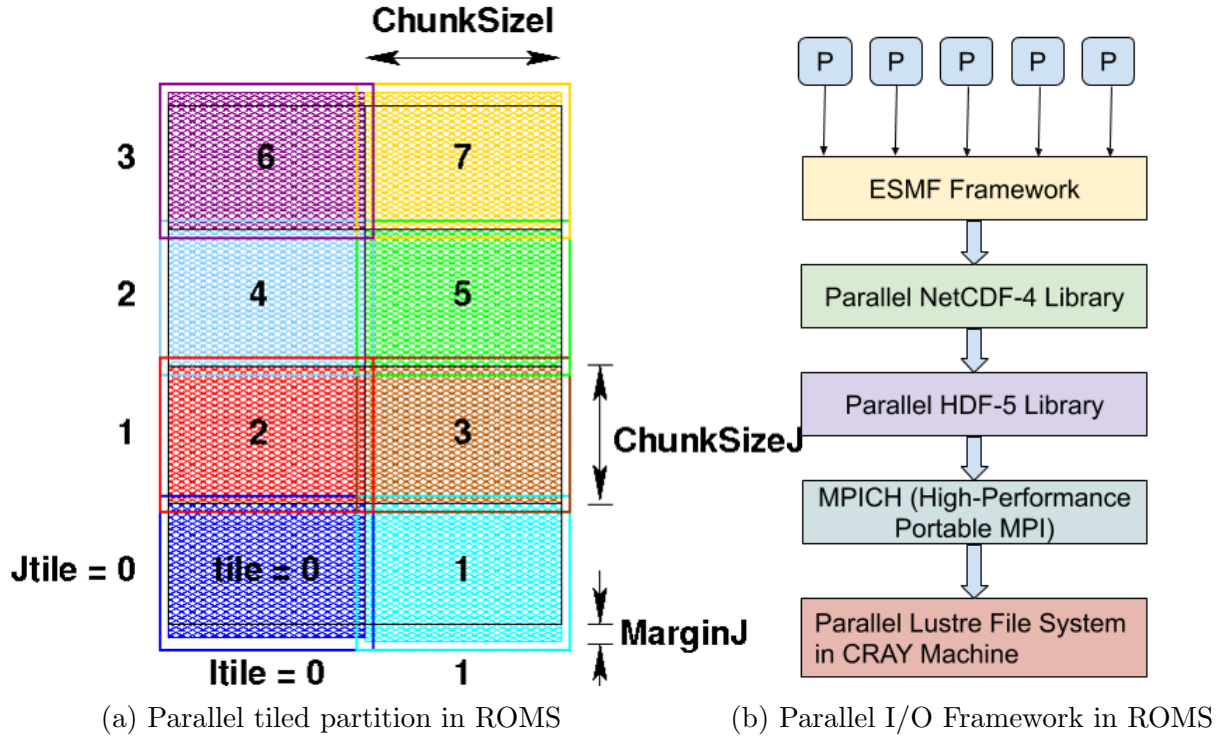
For many climate scientific applications, there is a lot of data that needs to be read and written to the underlying file systems. ROMS [8] is an ocean model developed in Rutgers University and UCLA and is widely used by the scientific community for a wide range of climate applications. It can work standalone as well as it can be coupled to atmospheric and/or wave models as shown in Figure 1.1. 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. In ROMS, the entire input and output data format of the model are via Network Common Data Form (NetCDF) which helps to interchange the data in a user-friendly way. We have used the ROMS framework for our experiments.



Source: <https://www.myroms.org>
Figure 1.1: ROMS model

ROMS model code is written in FORTRAN language. The ROMS model equations are discretized both in horizontal and vertical directions. It has the ability to mask the land points which is done while preparing the input file. It can run serially as well as it supports OpenMP and MPI for the computations. We have used MPI for all our experiments. It also supports nested grids to define fine and coarse grids at the same time. For using MPI, the neighboring points between tiles need to exchange information for independent computations as shown in (figure 1.2a). In distributed-memory, the I/O can be done serially on the master process or in

parallel by all the processes. The parallel IO requires the NETCDF4 parallel which works on top of parallel HDF-5 library which internally calls MPI library functions to interact with the parallel Lustre file system as shown in (figure 1.2b).



Source: <https://www.myroms.org/wiki/>

Figure 1.2: ROMS Model I/O Framework and Partitioning

From the I/O perspective of the ROMS model, it reads the input data from NetCDF data files before the computation stage starts. Then, it writes different types of output data files like 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 write data 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 in case the model fails for some reason. The diagnostic files are saved for analyzing various issues in the ROMS model. Also, the parameters to write and the frequency of writing each such file can be set during execution. It is checked that the reading of input data takes about 2 to 3 percent of the total execution time whereas the writing of output data varies from 50 to 90 percent of the total execution time depending on the number of variables written, number of output types, and their frequencies and number of processes used. So, our work is primarily focused

on writing the output files.

NetCDF [6] 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 stores as required by climate scientists. Ncview [7], 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 makes the NetCDF format well-suited for climate science applications.

The NetCDF-4 is built on top of the HDF5 library. 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 in output files in NetCDF format. The examples of 2d(ubar) and 3d(u) variable is shown in figure 1.3. 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. Also, all the related information about the variables is stored making them self-explanatory.

<pre>float ubar(ocean_time, eta_u, xi_u) ; ubar:long_name = "vertically integrate"; ubar:units = "meter second-1" ; ubar:time = "ocean_time" ; ubar:grid = "grid" ; ubar:location = "edge1" ; ubar:coordinates = "lon_u lat_u ocean_"; ubar:field = "ubar-velocity, scalar, s"; ubar:_FillValue = 1.e+37f ;</pre>	<pre>float u(ocean_time, s_rho, eta_u, xi_u) ; u:long_name = "u-momentum component" ; u:units = "meter second-1" ; u:time = "ocean_time" ; u:grid = "grid" ; u:location = "edge1" ; u:coordinates = "lon_u lat_u s_rho ocean_time" ; u:field = "u-velocity, scalar, series" ; u:_FillValue = 1.e+37f ;</pre>
(a) 2d-variable – "ubar"	(b) 3d-variable – "u"

Figure 1.3: NetCDF Output Variables Note: *ocean_time* is defined as UNLIMITED

The Hierarchical Data Format v5 (HDF5) [4] I/O library supports high-dimensional complex scientific datasets, easily shareable as the metadata is present along with the data in the same file making them self-describing, supports multiple languages, provides quick access of subset of data and optimizes the storage space with different levels of compression. Also, 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 prior.

Our focus is to analyze the I/O performance issues by profiling the ROMS model and then develop an I/O framework to improve the overall performance. CrayPat [1] is a performance analysis tool offered by Cray for the XC platform. We have used it to find the major bottlenecks

of the ROMS. We have used primarily own profiling strategies to analyze the bottlenecks and used this tool for overall analysis.

Different types of analysis like scalability studies, load imbalance studies, bottleneck function studies are performed on the ROMS model to find the reason behind large I/O times in the ROMS model. Based on the analysis results, the improvement techniques are decided for the ROMS model. The load imbalance found in the ROMS model in case of parallel I/O is solved after tracing the source of such imbalance and then finding the reason for such imbalance. The load imbalance reduction of Parallel I/O in the ROMS model gave 85% wallclock time reduction with respect to the present setup for Parallel I/O and 27% wallclock time reduction with respect to Serial I/O in the ROMS model for 1440 processes. There are many time-steps where the data variation is much less as compared to other time-steps in climate data. Based on this aspect, the data is written selectively in the output files with some acceptable accuracy loss constraints for time-consuming variables to reduce the total I/O involved in the ROMS model. The selective data writing technique is implemented on top of the load-balanced Parallel I/O setup of the ROMS model. This strategy gave around a 52% reduction in I/O write time and a 31.6% reduction in wallclock time within the accuracy loss constraints for 1440 processes. Also, the best combination of lustre striping improved the I/O write performance by another 41% in the ROMS model for 1440 processes. The selective writing combined with lustre striping gave around 40% improvement in wallclock time and 66% improvement in output time for parallel NetCDF. So, all three strategies combined together improved the parallel NetCDF performance by about 60% with respect to serial NetCDF setup. Hence, a significant improvement is achieved in the Parallel I/O performance as well as in the overall performance of the ROMS model using the above three I/O improvement strategies.

1.2 Organization

In Chapter 2, works related to I/O optimizations are discussed. The methods to improve the read and write performance of ROMS are discussed.

In Chapter 3, the working of the ROMS model and the I/O strategy used in this model are discussed. Also, the experimental set-up along with the data sets used for the ROMS model is mentioned. The overall approach to the problem is mentioned. The different techniques used for analysis and the results obtained are mentioned in detail.

In Chapter 4, the improvement strategies and corresponding results obtained are discussed. The methodologies for I/O improvement are based on observations and the conclusions obtained from I/O analysis results discussed in the previous chapter.

In Chapter 5, the conclusions obtained from the project and the future works are discussed.

Chapter 2

Related Work

This chapter includes the related works related to I/O in HPC systems.

For improving the write performance, the paper by Liu et al.[16] has implemented PnetCDF and Adaptable IO System (ADIOS) for alternative I/O solutions to the performance issues detected. Adaptable IO System is quite flexible in terms of I/O routine selection.

The existing implementation of the Goddard Earth Observing System Model (GEOS) [3] 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. But it does not scale well due to resource contention. Even if larger data is allocated to each plane to reduce the number of planes, it results in memory overhead and network bandwidth saturation. Also, the issue is that other processes need to wait when the bundle root writes the data to the file system. Now if the data is written in separate files by different processes, then it becomes difficult to manage the metadata and it is difficult to analyze and visualize the data. So the proposed idea is to write in parallel but in the same file system for performance improvement similar to what is done in PnetCDF.

ADIOS by Lofstead et al.[17] perform asynchronous I/O and offer multiple I/O mechanisms that can be changed as per user need. It can be converted into NetCDF and HDF5 format too. ADIOS can write data from multiple bundles in the same file in parallel. The data is stored in buffer memory and then it is written to disk if the buffer is full. It 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.[20] have proposed methods to develop an Online Analyzer with low computational overhead and low memory overhead maintaining high accuracy. The main aim is to detect patterns in reading and reducing the file read time. The data reading and analysis are done alternatively and the two phases are overlapped using the prefetching technique. There can be structured as well as unstructured reading patterns. Structured access patterns are either continuous or Kd-strided patterns. Unstructured access patterns can be computation based or correlation-based patterns. The rule-based model performs access pattern analysis during run-time and utilizes the pattern detected to do prefetching using prefetch cache memory. Tracer traces the read requests, analyzer searches in the pattern records, and then decides to either activate previous pattern and prefetch or use the data for further analysis. The requested data is copied directly from user buffer if found in prefetch cache or else it is directly read from the input files.

In Serial NetCDF, all the processes send their data to the master process and then the master process performs the I/O operation. Li et al. [15] proposed an approach where all the processes will be able to perform I/O collectively or independently on a single file through Parallel NetCDF(PnetCDF) library. But there are some advantages and disadvantages of PnetCDF over HDF5 format. A local copy of header information is kept in all processes to reduce inter-process communication in PnetCDF. But it does not support irregular layout patterns whereas HDF5 does. So, Rew et al. [18] decided to combine the benefits from NetCDF which provides an easy user interface for array-oriented scientific data, easier to analyze and visualize data with the HDF5 library which supports irregular data, big datasets and parallel I/O and NetCDF-4 data format was developed [12]. In the rest of this paper, parallel NetCDF refers to the NetCDF4 parallel library.

Optimization Techniques like data buffering [11], auto-tuning [9], combining multiple I/O operations to reduce disk access [13], sub-filing [14], topology-aware I/O [22], data staging [22], data sieving and collective I/O [21] are explored. Performance characterization of HPC systems is performed on various aspects like inter-process communication, interconnect technologies, parallel file systems, reading patterns, etc. Different I/O characterization tools like Darshan [19], output bottleneck characterization [23], continuous characterization tools [10] 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.

Chapter 3

Work Background

This chapter describes the Input/Output workflow of the ROMS model and the experimental setup of the ROMS model in the Cray XC40 supercomputer used in our experiments.

3.1 The Parallel Platform

All the experiments of the ROMS model are performed in the Cray XC40 system using the CPU only clusters. The detailed specifications of the SERC's Cray CX40 HPC system are mentioned in table 3.1.

Table 3.1: The Cray XC40 system

System	Cray XC40: Intel Haswell 2.5 GHz based CPU cluster
Number of Nodes	1376 nodes
Cores	24 cores per node (Total 33024 cores)
Main Memory	128 GB per node (Total RAM 172 TB)
Network	Cray Aries Interconnect with Dragonfly topology
Storage	Cray's Parallel Lustre File System (2 PB usable space)

3.2 ROMS Model – Overview

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 dumped into 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 need to be written to output files are velocity, temperature, and salinity. The output variables, the type of records to be written, the frequency of records to be written and the number of records to be written in each file can be specified as per user requirements.

Distributed memory in Cray machine with varying number of processes is used to run the model. The computations are always done in parallel for all experiments whereas the I/O is either done serially by the master process or in parallel by all the processes. In the next section, the serial and parallel I/O technique used in the ROMS model is discussed in detail.

3.3 ROMS Model – I/O Strategy

As the computation is always done in parallel, the input data is required in distributed form across all the threads. Similarly, the output data is present in distributed form across all the threads.

3.3.1 Serial and Parallel NetCDF Data Reading mechanism

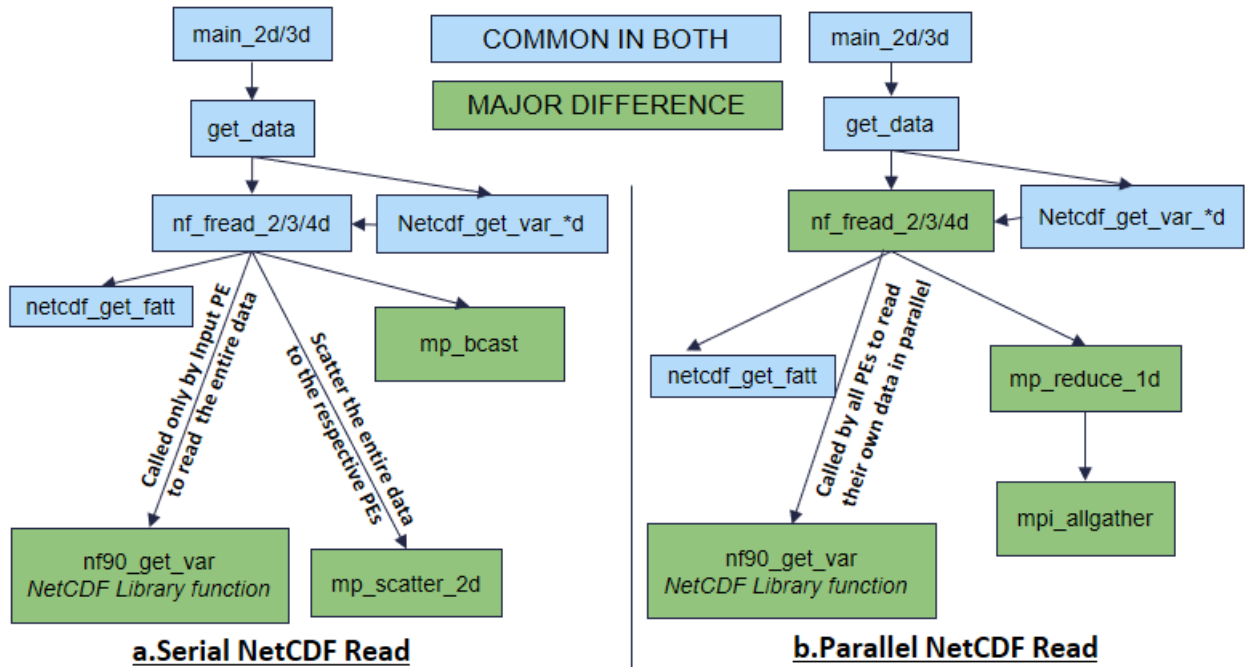


Figure 3.1: Reading of Input files using serial and parallel NetCDF

The function calls involved in the serial and parallel reading of NetCDF data is shown in figure 3.1. 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 `nf90_get_var` is the NetCDF library function for data reading.

3.3.2 Serial and Parallel NetCDF Data Writing mechanism

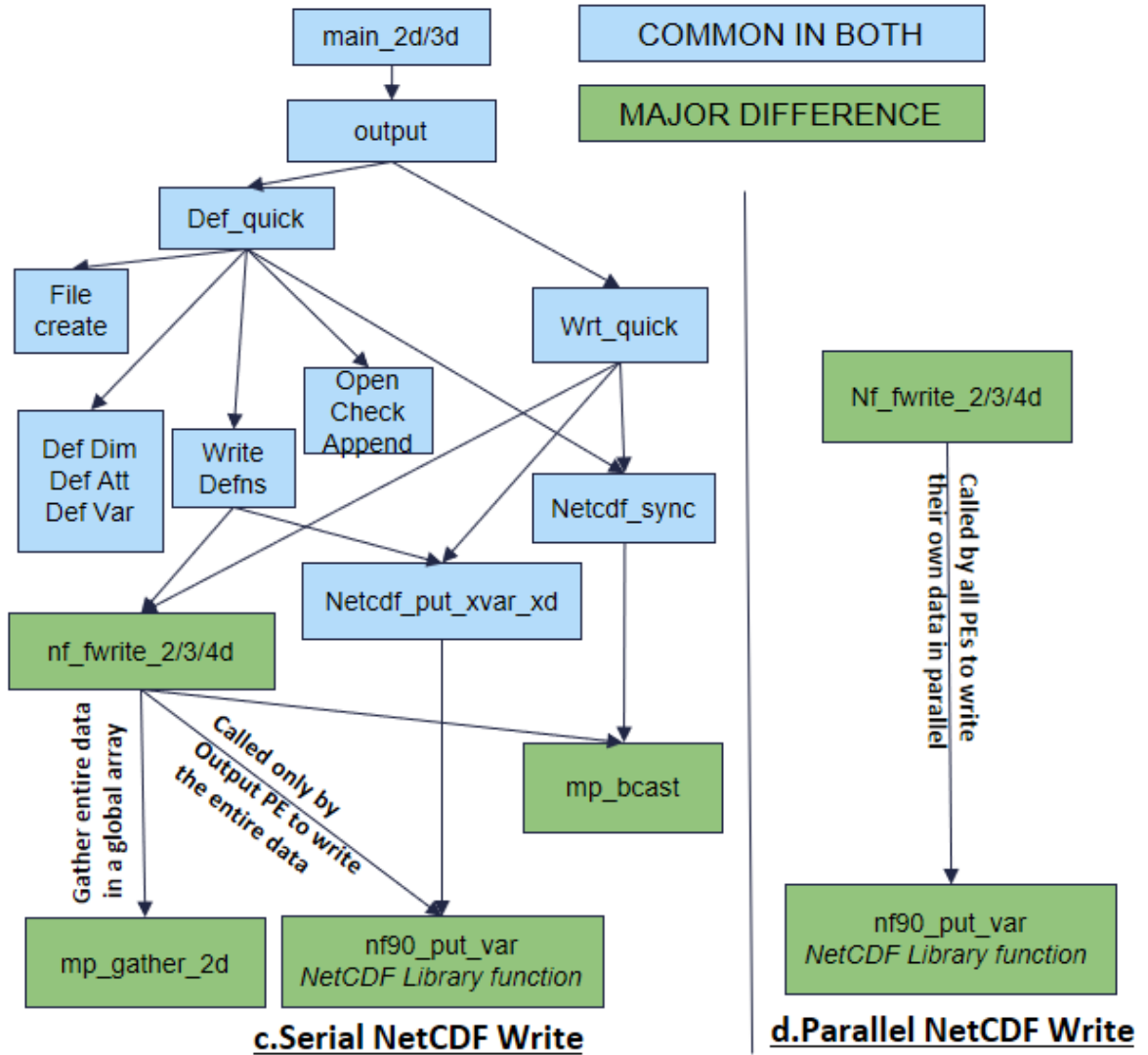


Figure 3.2: Writing of Output files using serial and parallel NetCDF

The function calls involved in serial and parallel writing of NetCDF data is shown in figure 3.2. 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(i.e. define phase) and then writing actual GRID data into those files(i.e. 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. The `nf90_put_var` is the NetCDF library function for data writing.

3.4 Experiment Setup

All the below experiments are done in the CRAY-XC40 supercomputing system. Cray's parallel lustre file system is used to read and write the I/O files. There is a minimum as well as maximum limit on the number of processes that can be allocated along latitude and longitude depending on the grid size used for ROMS setup as mentioned in (Table 3.2). We have used 48 processes to up to 1440 processes with fixed domain partitioning for a given number of processes as shown in (Table 3.3) for all our experiments.

It is found that in the ROMS model 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 3.4). Hence, the quick files are written in output for the experiments. The results also are valid for other types of output files as the data format is the same in all. Also, the same defining phase and writing phase functions are used in all types of output files.

For the analysis part, the ROMS model is run for 500 time steps as mentioned in (Table 3.5) with 10 records are written in each quick file and 5 quick files are written in that period.

Table 3.2: Experimental Setup table

Objective	To analyze the I/O characteristics of ROMS model
Model	ROMS – Ganga-Brahmaputra Plume in BOB
System	Cray XC40: Intel Haswell 2.5 GHz based CPU cluster
Cores (24 cores in each node)	48 to 1440 PEs (2 to 60 nodes)
Compiler	INTEL
Profiler	Cray PAT

Table 3.3: Domain Partitioning table

Domain partitions-X*Y	Total Cores	Nodes in the CRAY system
8*6	48	2
16*15	240	10
32*30	960	40
40*36	1440	60

Table 3.4: Output file types, writing frequency and storage space (30 days simulation)

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
Diagnostics	30	1	1.1	1.1

Table 3.5: ROMS MODEL Setup table

Application	Ganga-Brahmaputra Plume in BOB
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)
I/O	Serial NetCDF and Parallel NetCDF-4 (works on top of HDF-5)
File Types	Quick Files every 100 steps (As they are most time consuming)

3.4.1 Input Dataset Used

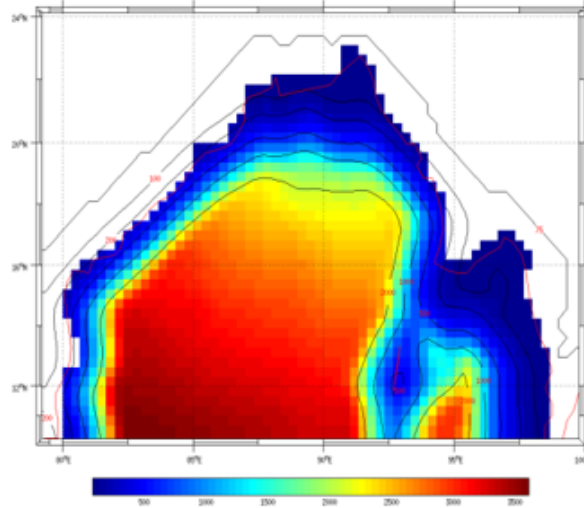
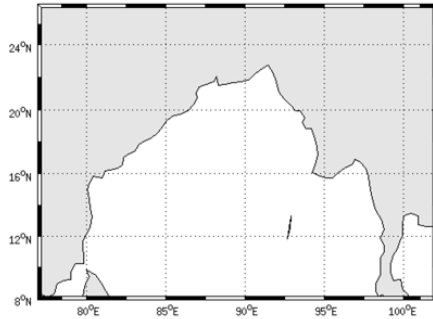
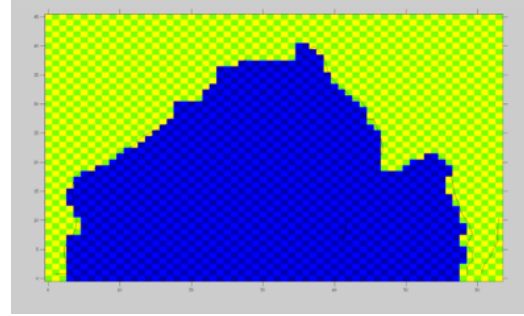


Figure 3.3: Initial Input Data using MATLAB



(a) Before manual editing step



(b) Land parts edited manually

Figure 3.4: Intermediate steps

The initial dataset that is required to run the ROMS model is generated using Matlab scripts as shown in figure 3.3. Input data consists of 18 NetCDF files of a total of 18 GB is obtained. The pre-processing steps involve the preparation of the grid file as in figure 3.4. The land areas that need to be masked are defined during the grid preparation. Manual editing is required before generating the final data as shown in figure 3.5.

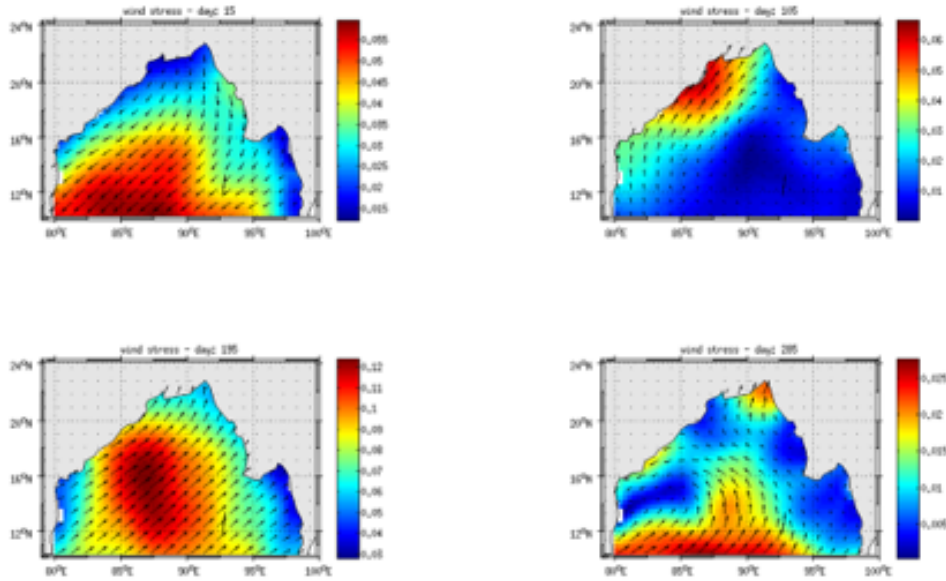


Figure 3.5: Final Input Data

3.5 Overall Methodology for I/O analysis

The overall methodology comprises of two steps - first, we need to perform a detailed analysis of the I/O performance of the present climate models (mentioned in this chapter) and then we need to improve the overall I/O performance (next chapter) of such models. We have worked on the ROMS model for the I/O analysis and I/O improvement.

For doing I/O analysis, the below methodologies are followed:

1. Scalability Studies – It is done with and without I/O for both NetCDF serial and NetCDF parallel to find the right set of processes for both read and write performance optimization in the current ROMS model.
2. Load Imbalance Studies – The load imbalance is studied for both serial or parallel NetCDF as there is a large time difference found in the wallclock time and the average time taken per CPU from profiling.
3. Bottleneck Studies – CrayPAT analysis tool is used to trace the time-consuming functions. Message Passing Interface (MPI) calls are traced to find the reason for the difference in execution times of NetCDF serial and NetCDF parallel.

3.6 Scalability Studies

The scalability studies are performed on the ROMS model using serial as well as parallel NetCDF for comparison. The processes are varied from 48 to 1440. The data reading is mandatory for the model to do computations whereas the data writing is optional. So, two separate experiments were performed with and without data write to find the scalability in the writing.

3.6.1 Strong Scaling Results - file read and computation only

In this experiment, serial and parallel NetCDF is 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 scales up to around 1440 processes as in figure 3.6. Also, it is seen that parallel NetCDF read is taking a bit more time compared to serial NetCDF. In both cases, the computation is performed in parallel and is scaling well.

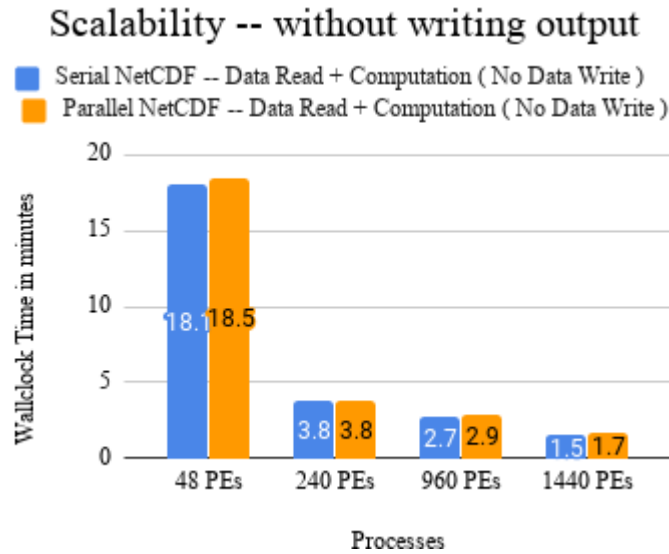


Figure 3.6: Scalability – Input Data Reading + Computation

3.6.2 Strong Scaling Results - file read, computation and file write

In this experiment, serial and parallel NetCDF is 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 shown in figure 3.7. Moreover, the parallel NetCDF timings are found to be more than the serial NetCDF timings.

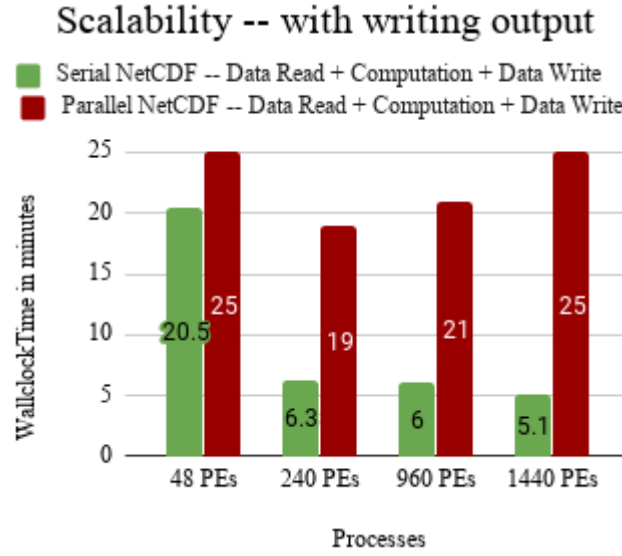


Figure 3.7: Scalability – Input Data Reading + Computation + Output Data writing

3.6.3 Strong Scaling Results - file write only

For this, the difference in timings between figure 3.7 and figure 3.6 is plotted in figure 3.8. The I/O time for writing the output files is increasing with the increase of the number of processes for both serial and parallel NetCDF as shown in figure 3.8. The serial NetCDF write is not scaling with the increasing number of processes as the data is transferred from a more number of processes to the master process writing the data. Our primary goal is to find the reason for which the NetCDF parallel is not scaling and taking more time than the NetCDF serial.

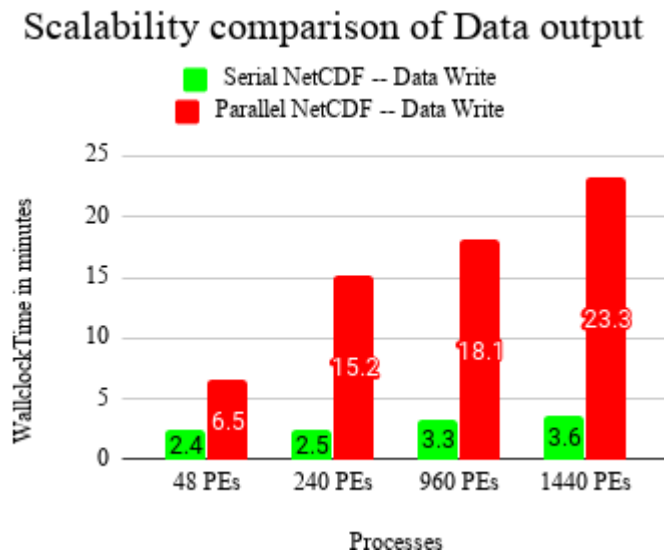


Figure 3.8: Scalability – Output Data writing [Fig 3.8 = Fig 3.7 - Fig 3.6]

3.6.4 Input and Output percentages wrt total execution time

The percentage of data reading and data writing times with respect to the total execution time for the varying number of processes obtained for I/O using serial NetCDF and parallel NetCDF in the ROMS model is shown in figure 3.9. It is observed that the data reading using serial NetCDF and parallel NetCDF is taking only 0.5% and 2.4% of the total execution time. This is because the data is read only once at the beginning. The difference between Parallel and Serial data read cases found to be insignificant. Hence, we have not investigated the reason for the slight increase in the overall execution time of parallel NetCDF over serial NetCDF reading in the ROMS model.

But the data write is taking 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 an increasing number of processes is because computation is scaling whereas I/O write is not scaling. Hence, our primary goal is to focus on the writing of output files as they are consuming a large fraction of the total execution time.

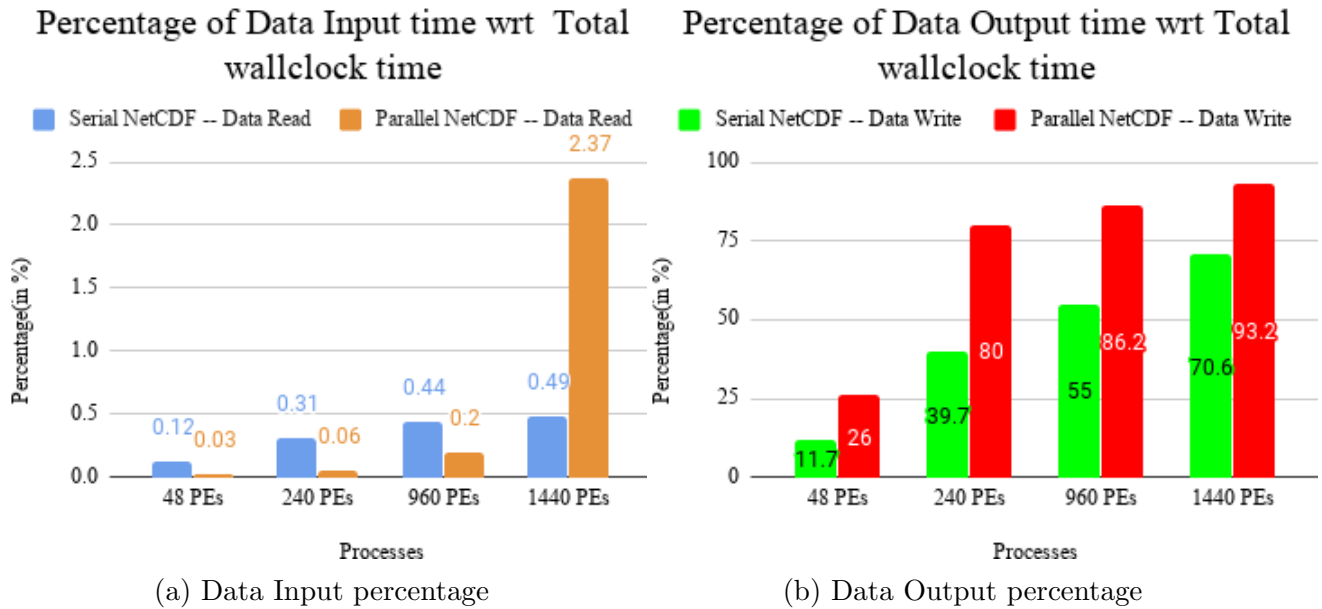


Figure 3.9: Percentage of Data Read and Write times wrt the total execution time

3.7 Load Imbalance Studies

The minimum and maximum CPU times among all processes and the average of the execution times are measured to check the load imbalance. From this experiment, it is found that for serial NetCDF data reading as well as data writing, and parallel NetCDF data reading the execution time all CPUs are almost similar. So the average execution time per CPU is close to the wall-clock time for such cases. The corresponding plots are not shown here. But when parallel NetCDF writing is enabled, there is a lot of execution time imbalance among different processes as shown in figure 3.10.

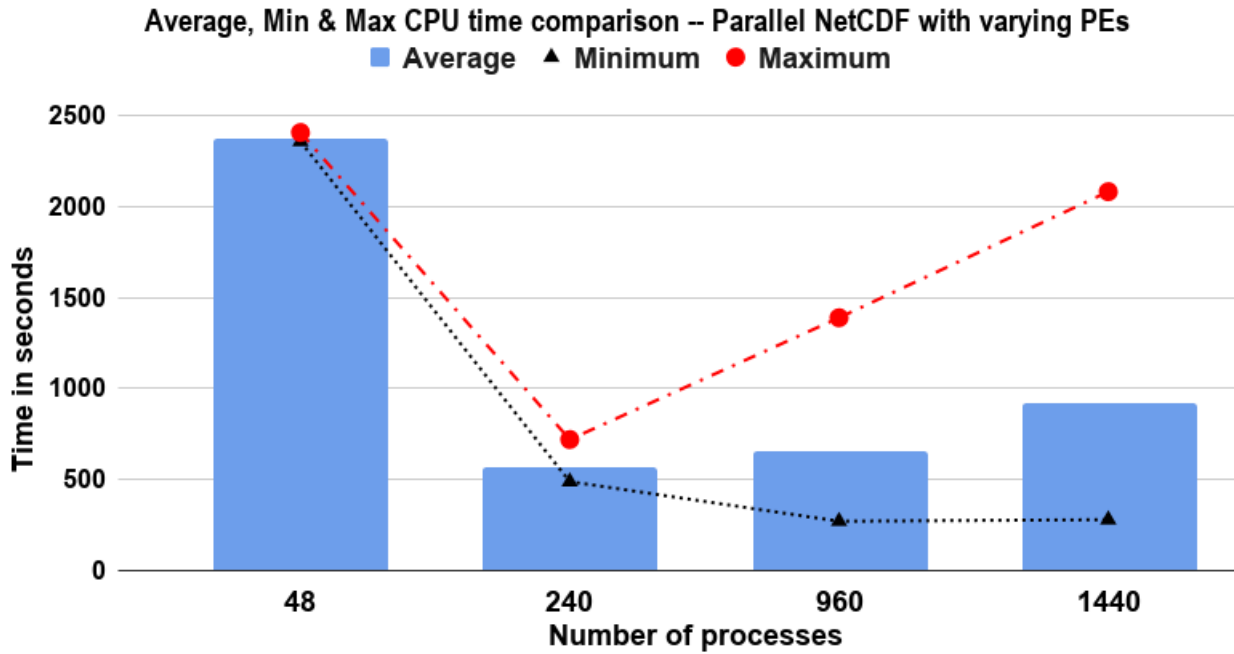


Figure 3.10: Load Imbalance Study

To analyze the variation of imbalance with increasing processes, the number of processes used for Parallel NetCDF data read and write are varied. It is found 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. As the wall-clock execution time is always going to be greater than the maximum CPU timings, the scalability curve for Parallel NetCDF in figure 3.7 is justified by the max CPU timings curve in figure 3.10. Next, our goal is to find the reason for this huge imbalance in the case of Parallel NetCDF. The parallel NetCDF timings can come down if the minimum and maximum can be made close to the average timings. Our primary focus is for 1440 processes as computation is scaling well till 1440 processes. So, we did further imbalance analysis experiments on 1440 processes.

3.8 Bottleneck Study

3.8.1 Using CrayPAT Tool

To trace the bottlenecks in serial and parallel NetCDF writing, we have used the CrayPAT performance analysis tool. For 48 processes, 2d and 3d tiling steps related to computation took almost 55 percent and 7 percent respectively of the total execution time in both serial and parallel NetCDF. In the case of 1440 processes, the bottleneck functions are found to be related to I/O. For serial NetCDF, the MPI broadcast is almost consuming 87 percent of the total execution time whereas, for Parallel NetCDF, the H5FD_mpio_write is consuming 60 percent nc4_put_vara is consuming 20 percent of the total execution time.

3.8.2 MPI communications

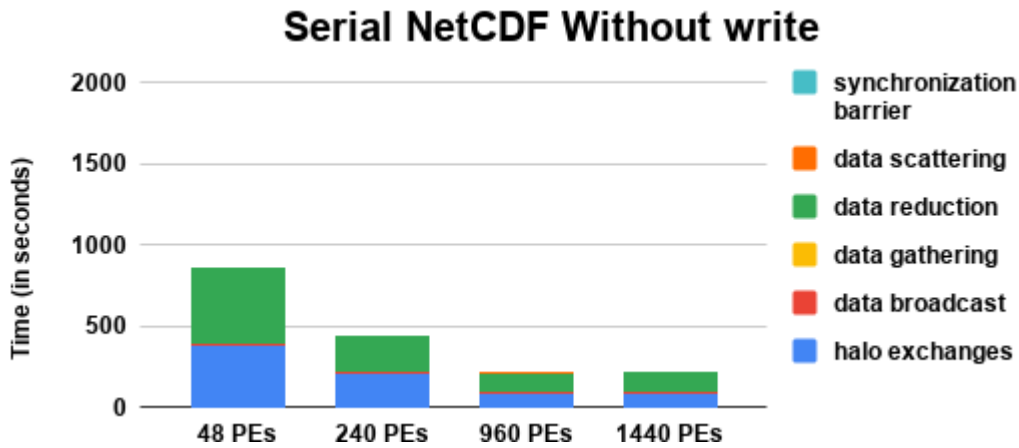


Figure 3.11: MPI Communications – Serial Read + Computation

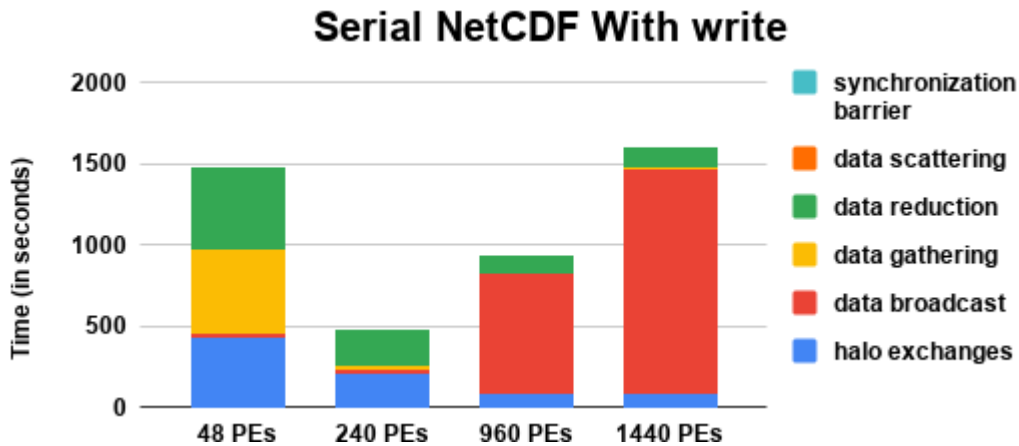


Figure 3.12: MPI Communications – Serial Read + Computation + Serial Write

All the MPI functions are compared to find bottlenecks. In both cases of serial NetCDF without (figure 3.11) and with write (figure 3.12), the halo exchanges are scaling well which is a part of the computation. Data gathering is extra in serial NetCDF with write as it is required for collecting data from all processes to master for writing. For serial NetCDF with write, the broadcast time is increasing for serial NetCDF with an increasing number of processes as more processes need to send their data and then wait for the master process to complete the data write. They proceed for the next computations once "No Error" is broadcast by the master to all leading to an implicit barrier.

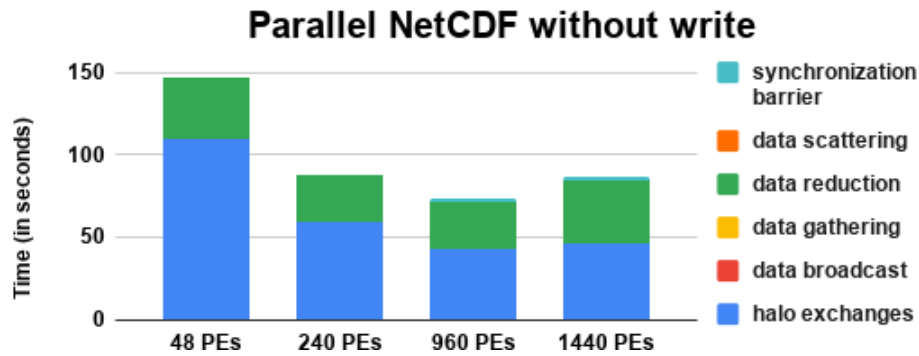


Figure 3.13: MPI Communications – Parallel Read + Computation

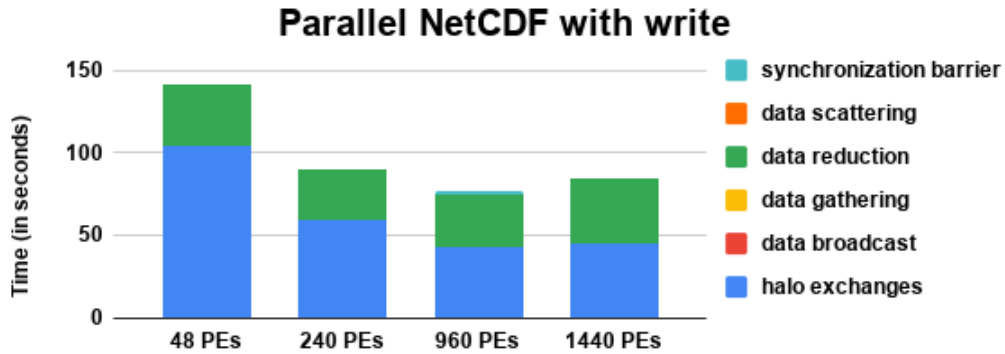


Figure 3.14: MPI Communications – Parallel Read + Computation + Parallel Write

For parallel netCDF without write (figure 3.13) and with write (figure 3.14), scatter and gather are absent as all the processes read and write data of their tile. Also, the broadcast time is negligible as all the processes receive the write error messages separately. There is no extra MPI communication overhead for Parallel NetCDF with write as compared to Parallel NetCDF without write. Also, the total MPI communication times in parallel I/O is almost 10 times lesser as compared to serial I/O. Hence, it is confirmed that the MPI communications are not responsible for the large I/O time in parallel NetCDF.

Chapter 4

I/O improvement

This chapter describes the different methods and techniques used to solve the issues that are found from the I/O analysis as mentioned in the last chapter. Our goal is to improve the I/O performance which will lead to the improvement of the overall performance of climate models.

4.1 Overall Methodology for I/O improvement

For I/O improvement, the below methodologies need to be performed:

1. Imbalance time reduction – It is found that there is a huge time difference between average execution time and maximum CPU times in the case of Parallel NetCDF. So, we can get better parallel I/O performance if the loads can be balanced properly. The source of such imbalance is traced throughout the code and the reason is found. After solving the imbalance, the improvement results in parallel I/O are compared with default setup of parallel I/O and serial I/O in the ROMS model.
2. Reducing output file writes when possible – Based on the variability in the data and then by using a evaluation strategy to check the accuracy loss, the data is written selectively maintaining the feasible accuracy loss constraints for practical applications.
3. Lustre Striping – Instead of setting the lustre striping to default, the best combinations of stripe size and stripe count (it depends on the number of processes used, size of each write request and the way they are distributed among the processes) are found and the results are compared with the default striping.

4.2 Load Imbalance Studies

The experiments are performed for 1440 processes as the computation is found to scale till 1440 processes whereas I/O using the NetCDF parallel is taking maximum time for 1440 processes.

4.2.1 Experimental Setup

Table 4.1: Experimental Setup for I/O Imbalance Improvement

Processes	1440 processes
Variables written	u(2D),v(2D),u(3D),v(3D)
Output Files	Quick Files(Data Snapshots)
Length of Run	500 time-steps
Write Frequency	10 time-steps(50 records)
File creation Frequency	100 time-steps(5 output files)

4.2.2 Data Distribution

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 (WRITE_WATER flag is OFF) as shown in figure 4.1. Also, in case only water data points are written (WRITE_WATER flag is ON), 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).

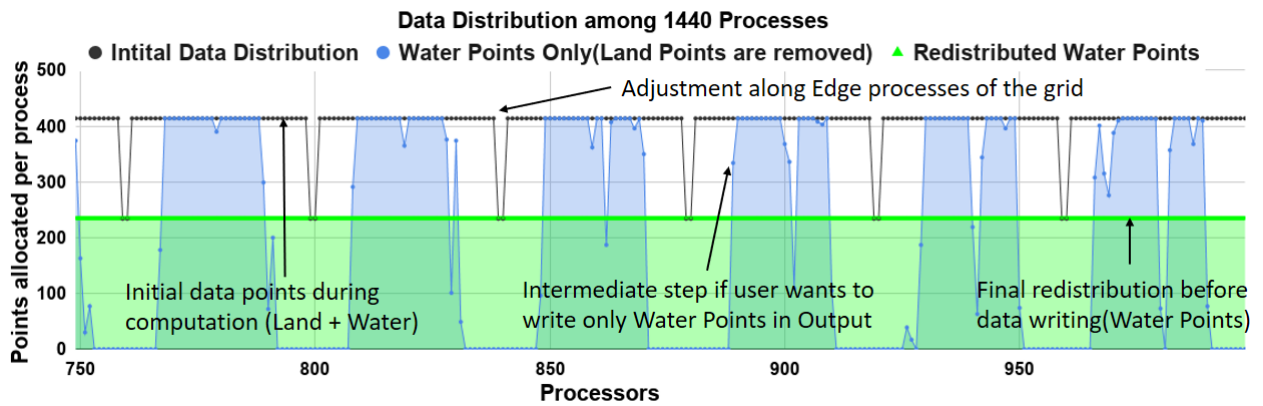


Figure 4.1: Data Distribution among 1440 processes(Only 750 to 1000 processes are shown for better clarity of the data points in each process)

Only the processors at the boundary have lesser points for adjustments along both latitude and longitude. It is found that the imbalance pattern seen is not related to the edge processors as shown in figure 4.2. Hence, it is confirmed that the imbalance is not due to data distribution.

4.2.3 Imbalance Flow

So a detailed analysis of the function calls and the per-process timings for such functions is performed to investigate the reason for such imbalance. 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 figure 4.2. It is observed that the computation including I/O read time is balanced and the imbalance is coming from Output part.

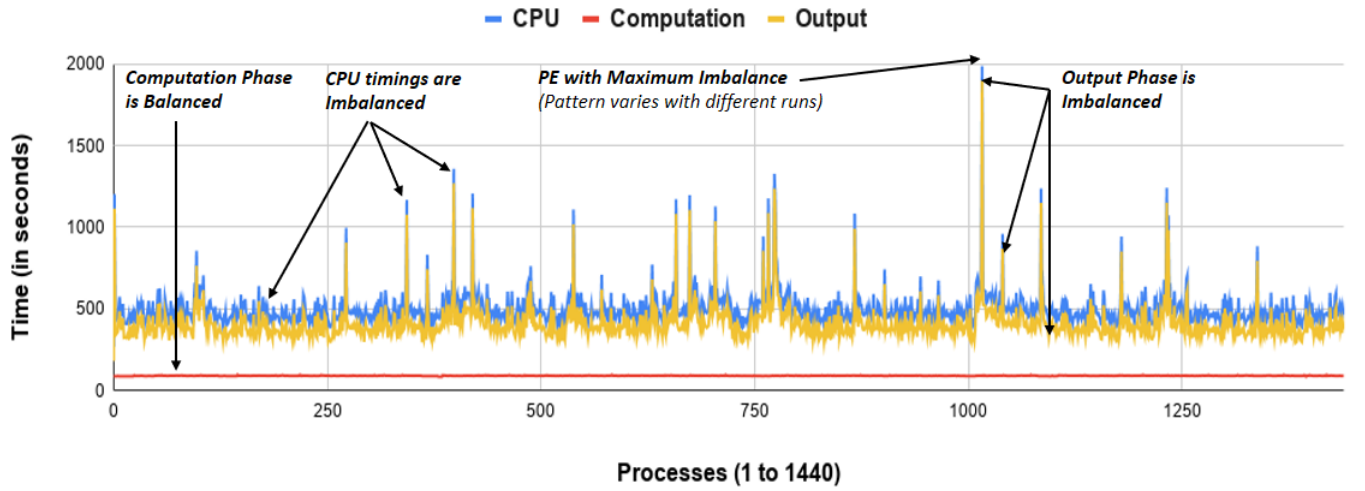


Figure 4.2: $\text{CPU}(\text{Imbalanced}) = \text{Computation}(\text{Balanced}) + \text{Output}(\text{Imbalanced})$

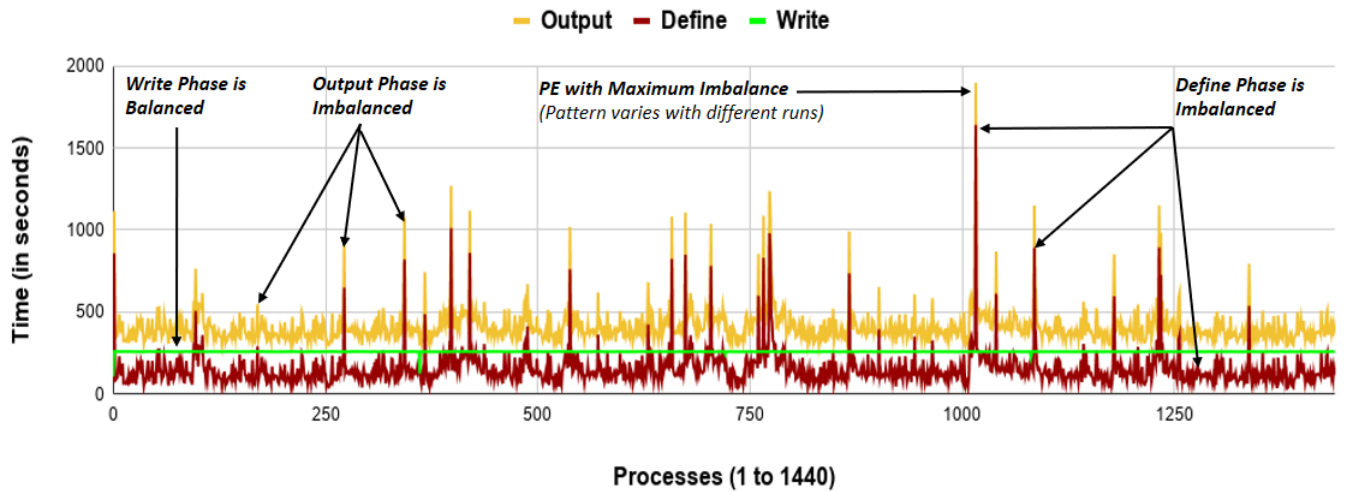


Figure 4.3: $\text{Output}(\text{Imbalanced}) = \text{Define Files}(\text{Imbalanced}) + \text{Write into files}(\text{Balanced})$

The processors responsible for the imbalance can be seen as the spikes in the plots. It is verified that the imbalance pattern is not the same in different runs. It means different processors are responsible for such imbalance peaks in different runs ensuring further that it is not because of data distribution as data distribution is always the same in each processor.

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 is load-balanced whereas the define file times are load imbalanced as shown in figure 4.3. It is also seen that the imbalance is increasing if we define more files keeping the total amount of data to be written constant i.e. lesser records in each file.

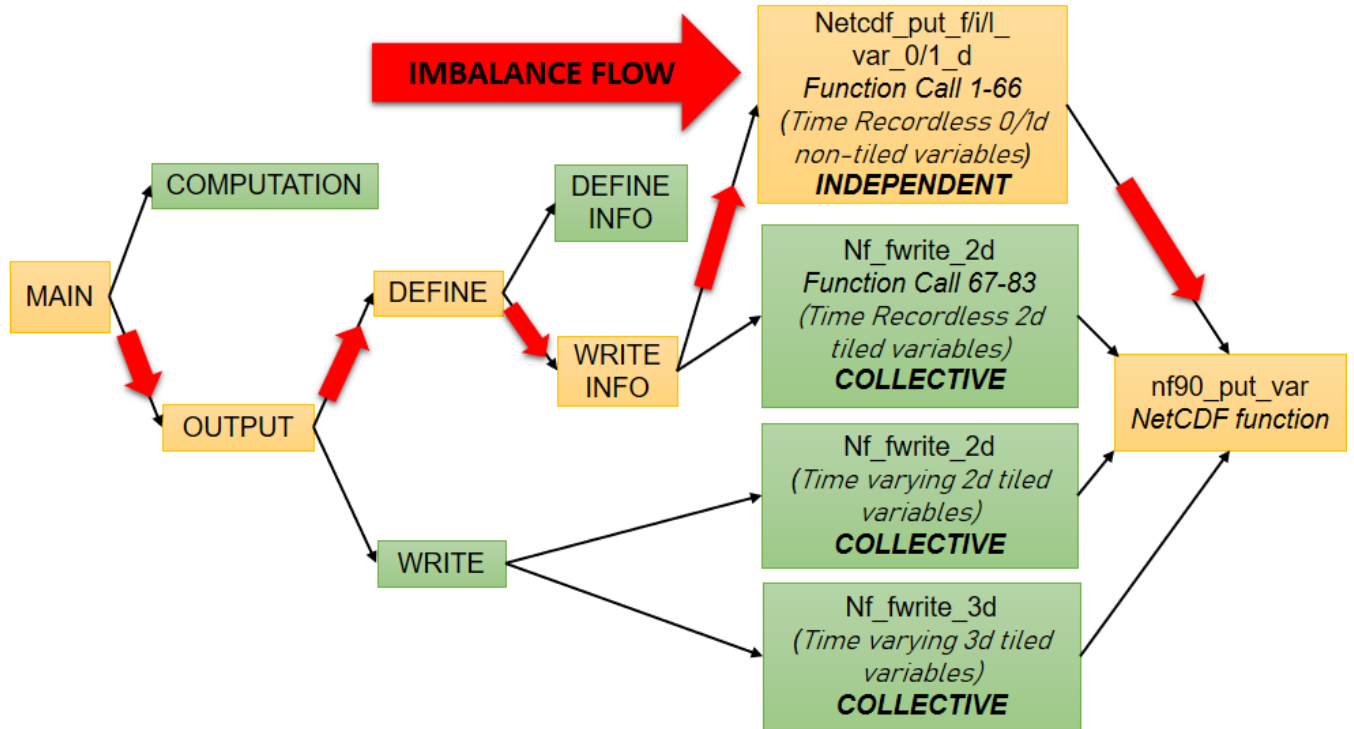


Figure 4.4: Imbalance Flow Diagram

So, a further breakup of defining file timings is obtained. The defining phases involve either Define_Info (It involves creating a NetCDF file, defining dimensions, attributes, and variables) and Write_Info (It involves writing time-recordless, information variables) or appending into an existing NetCDF file. Among these, Write_Info consumes significant time and other define call timings are found insignificant. It is found that the Write_Info phase is load imbalanced as shown in figure 4.4. Write_Info is consuming a major portion of Define time as in figure 4.5.

During Write_Info within define file phase(mentioned as WRITE INFO), `nf_fwrite2d` function are called and there is no `nf_fwrite3d` call inside `wrt.info.F` whereas during writing data into file phase (mentioned as WRITE), both `nf_fwrite2d` and `nf_fwrite3d` are called. It is found that the `nf_fwrite2d` consume almost insignificantly small time (hence not shown in plot) as compared to `nf_fwrite3d` calls (`nf_fwrite3d` is consuming major portion of Write time as in figure 4.5). So, it can be concluded that the load imbalance is not coming from `nf_fwrite2d` calls during write info. Also, writing 3d variables consume most of the time during write phase.

It is found that `Netcdf_put_f/i/l_var_0/1d` functions 1 to 66 are called by the non-tiled variables (The non-tiled variable data is not distributed among processes as they are either single-valued variable or small vectors) whereas `nf_fwrite2d` functions 67 to 83 is called by the tiled variables (The tiled variable data is distributed in tiles among the processes typically the 2d or 3d variables) within Write_Info as shown in figure 4.4. Both of them calls `nf90_put_var`, a NetCDF Fortran 90 library function. The imbalance flow is highlighted with red arrows.

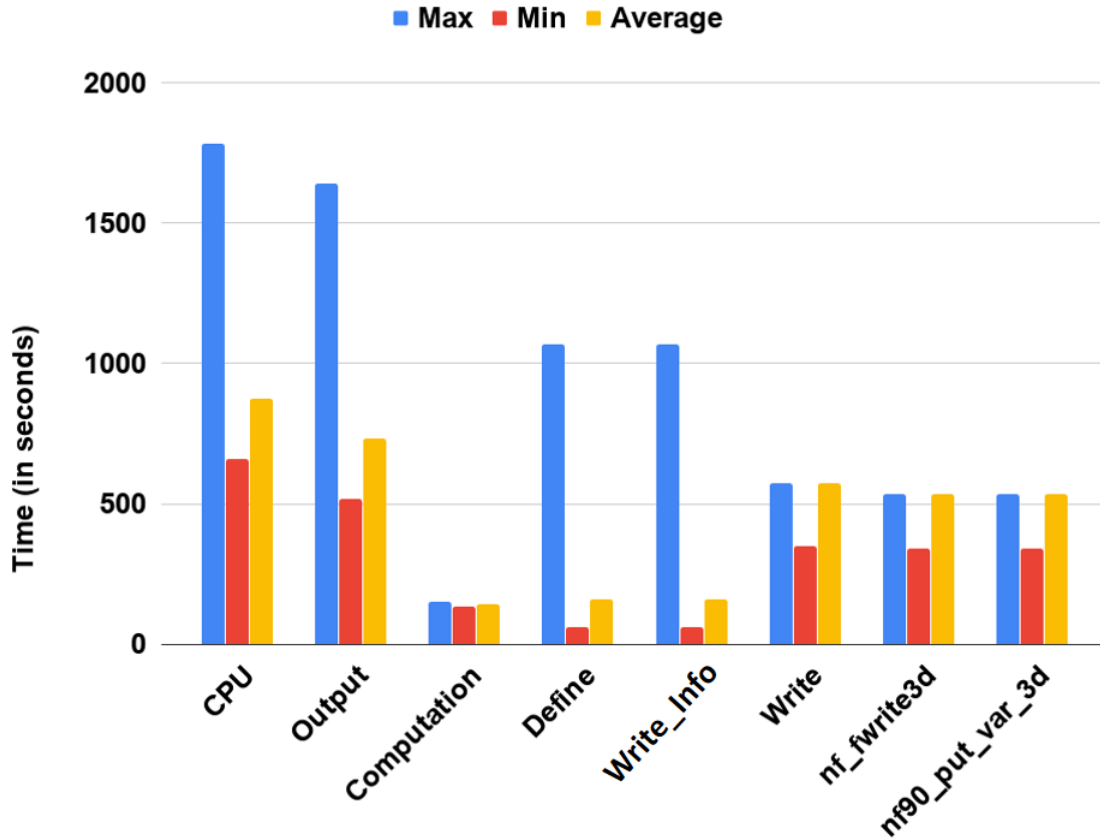


Figure 4.5: Min, Max and Averages times comparison for 1440 Processes
CPU = Computation + Output and Output = Define + Write

4.2.4 Bottleneck Function Studies

The NetCDF operations in the Define_Info phase which involves metadata writing, i.e creation of variables dimensions and attributes are always collective for parallel NetCDF. But the parallel `nf90_put_var` write function calls used during Write_Info calls of define phase or during write phase can be independent (all the processes write their part of the data independently) or collective (all processes write the data collectively).

It was found that 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. The collective I/O calls lead to optimization depending on I/O patterns as the file allows to write the processes collectively at the same time. 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 (though independent calls help to minimize global synchronization overhead). 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 as shown in figure 4.4.

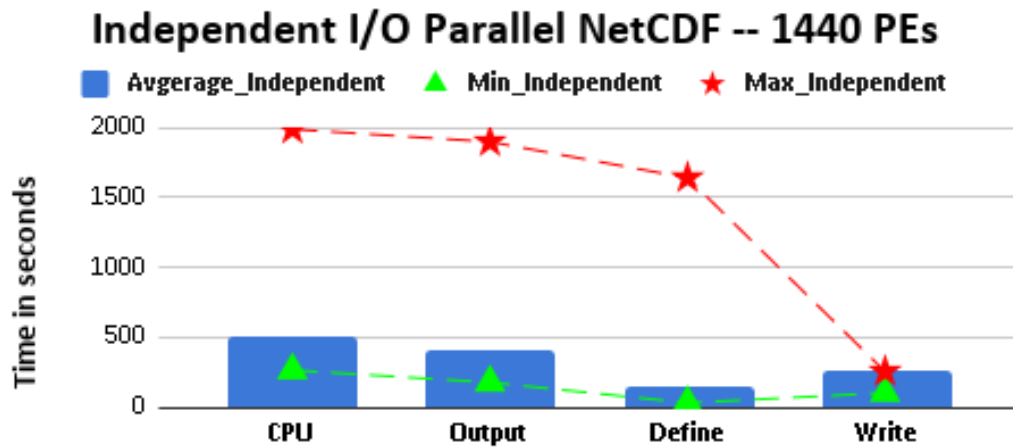


Figure 4.6: Independent functions

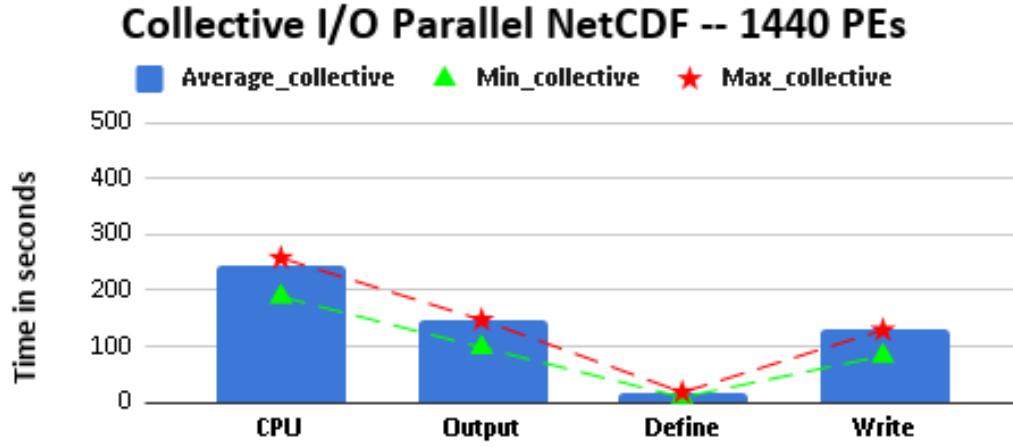


Figure 4.7: Collective functions

So, it is concluded that the imbalance is coming from the independent `nf90_put_var` function calls inside `Netcdf_put_f/i/l_var_0/1d` function for non-tiled variables (`Write_Info`) during define phase. The imbalance is solved after those function calls were changed from independent (figure 4.6) to collective (figure 4.7). Now, the non-tiled variables make a single collective write request instead of independent calls for the same data by all the processes.

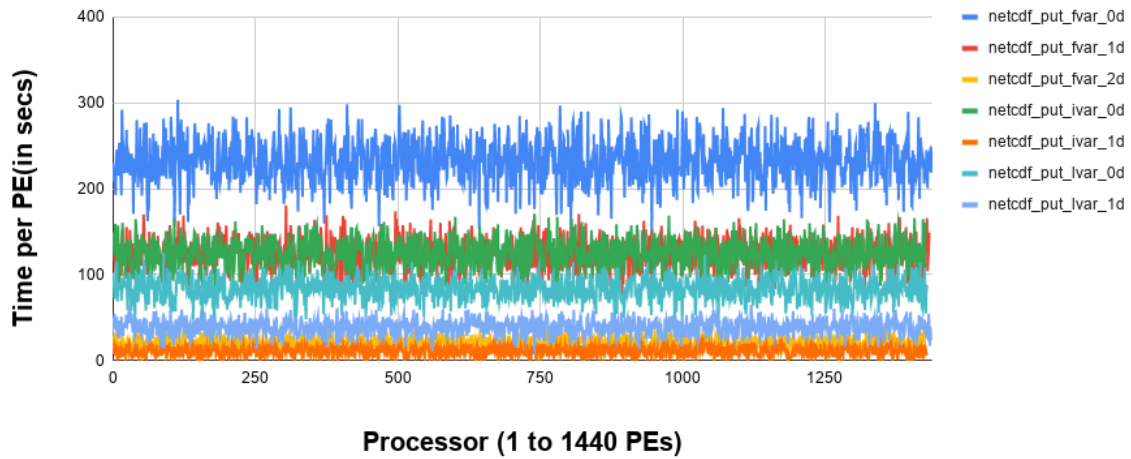


Figure 4.8: Bottleneck in the 7 different functions

Further analysis of the amount of imbalance coming from each of those 66 function calls (which were defined independently by default) are studied to check the contribution by different functions. It is found that there are 7 different types of functions involved. The barrier is added at the start and end of those functions to measure their imbalance contribution as shown in figure 4.8. It is observed that each of those functions is contributing imbalance proportional to

the number of times they are called as shown in figure 4.9.

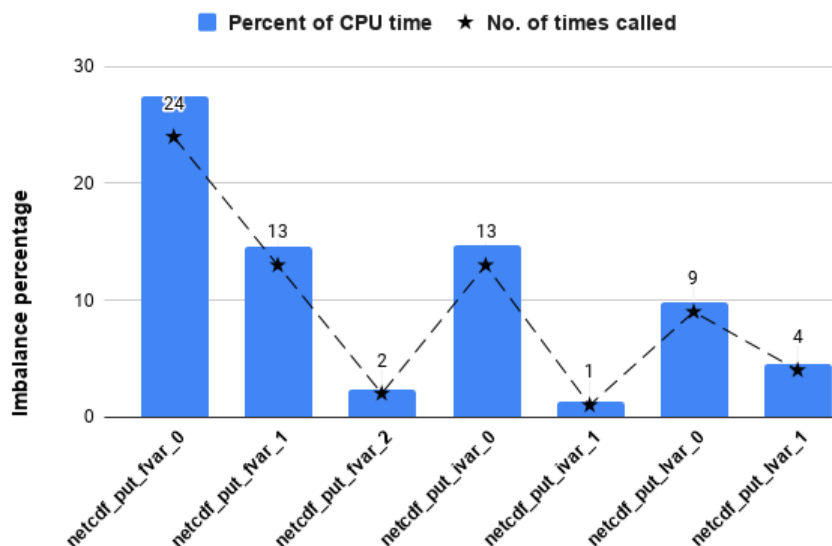


Figure 4.9: Imbalance Contribution by different NetCDF Functions

4.2.5 Performance Improvement 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 4.10.

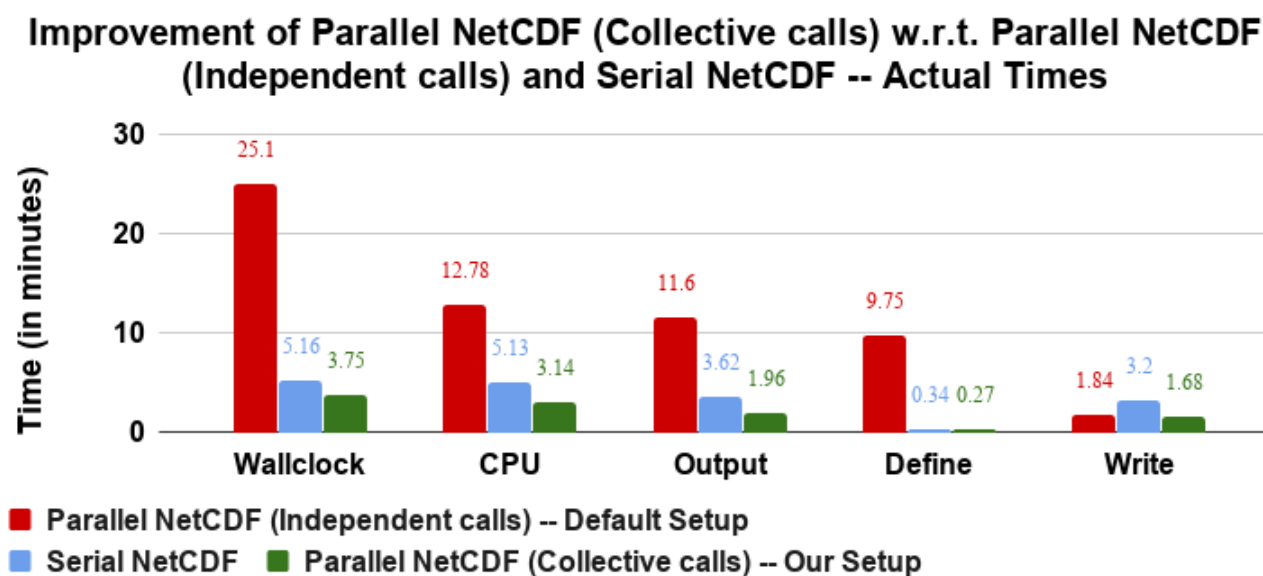


Figure 4.10: Improvement Results after solving imbalance for 1440 processes (Actual Times)

The maximum out of all the processes is considered for comparing CPU, Output, Define, and Write timings. It is observed that the execution time has come down from 25.1 minutes to 3.75 minutes (i.e. saving of 21.35 minutes wrt default setup of parallel NetCDF). Also, it is performing better than serial NetCDF which took 5.16 minutes (i.e. saving of 1.41 minutes wrt serial NetCDF). 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 the new parallel NetCDF setup 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 case of default setup of parallel NetCDF. With the new setup, the define phase itself for parallel NetCDF has come down to 0.34 minutes. It is also observed that there is a huge difference in the wallclock time(almost double) and maximum CPU time out of all the processes in case of default parallel NetCDF setup. Whereas for the new parallel NetCDF setup and serial NetCDF the difference in the wallclock time and maximum CPU time out of all the processes is quite small. It shows that the imbalance present because of the independent calls in parallel NetCDF is non-overlapping as shown in figure 4.11.

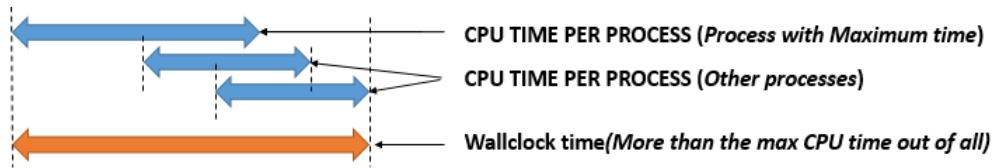


Figure 4.11: Non-overlapping CPU time imbalance contributed by different processes

Improvement of Parallel NetCDF (Collective calls) w.r.t. Parallel NetCDF (Independent calls) and Serial NetCDF -- In percentage

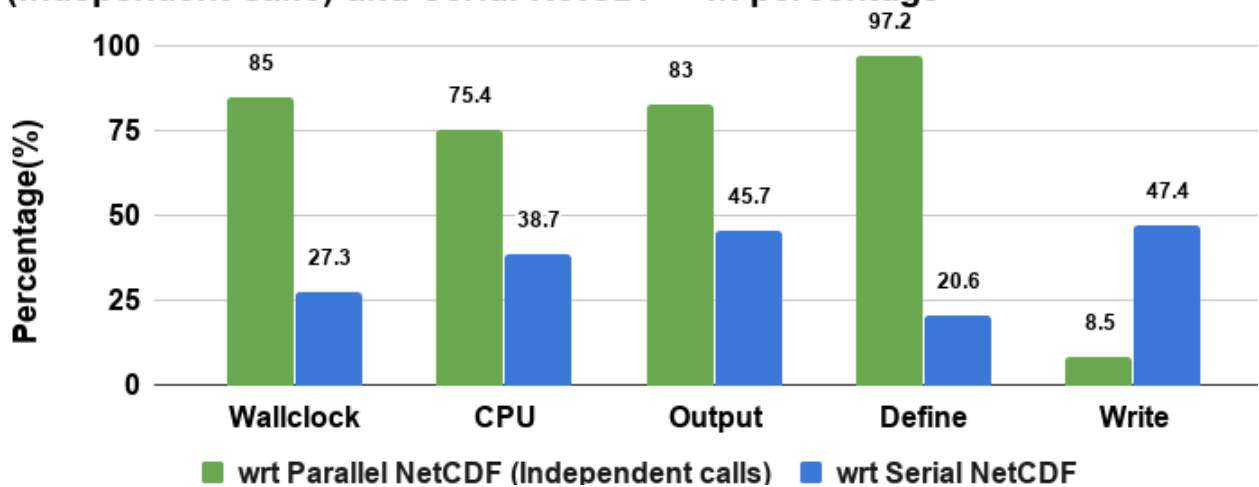


Figure 4.12: Improvement Results after solving imbalance for 1440 processes(in percentage)

The corresponding improvements (in percentage) is shown in figure 4.12. Average from 5 experimental runs are taken for these results. It is observed that parallel NetCDF with collective calls (Our new Setup) gave 85% improvement as compared to parallel NetCDF with independent calls (Default Setup) and 27.3% improvement as compared to serial NetCDF in the ROMS model. Primarily, the reduction is coming from the define phase of the parallel NetCDF with collective calls (97.2% wrt parallel NetCDF with independent calls). The improvement achieved from the define phase is reflected in the output phase, CPU times, and wall-clock times. In the write phase, the data write timings are similar as in the default setup of parallel NetCDF as the tiled variable function calls were already in a collective mode.

4.2.6 New Scalability Results

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) is shown in figure 4.13 and 4.14. The execution time, output and write time for Par_COL is scaling now unlike Par_IND and it is performing better than Serial for 960 and 1440 processes. The define phase imbalance is increasing for Par_IND with more number of processes as there is an increase in I/O contention with increasing independent I/O request. The serial output is not scaling as data need to be gathered from more processes. The execution time for Serial is scaling though the output of Serial is not scaling because of dominating scalability in computation which is always parallel. It is seen that the benefits of parallel NetCDF are achieved in case of a large number of processes.

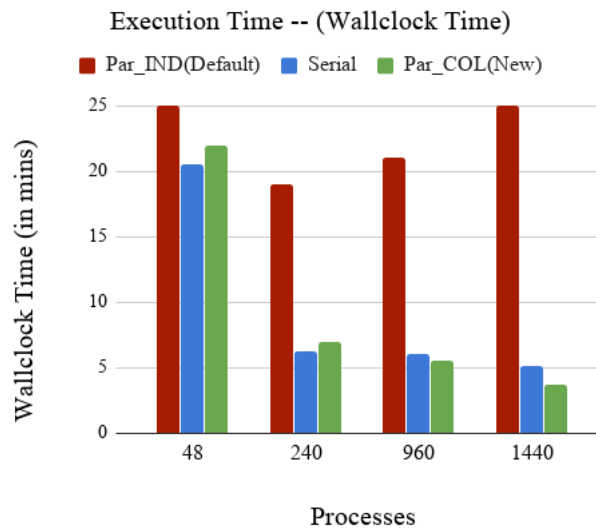
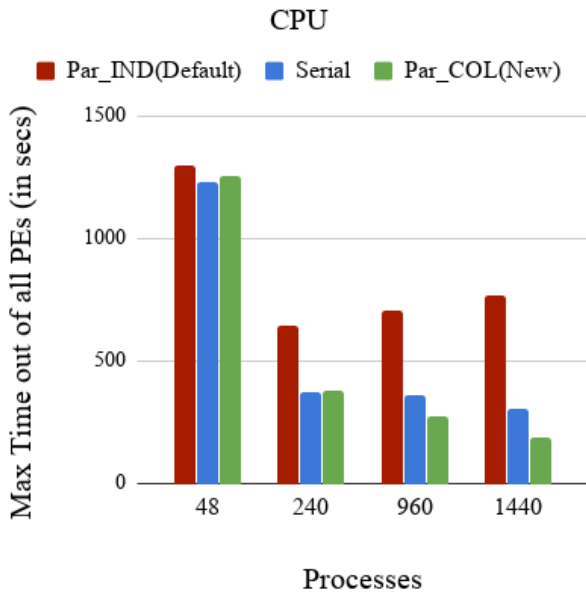
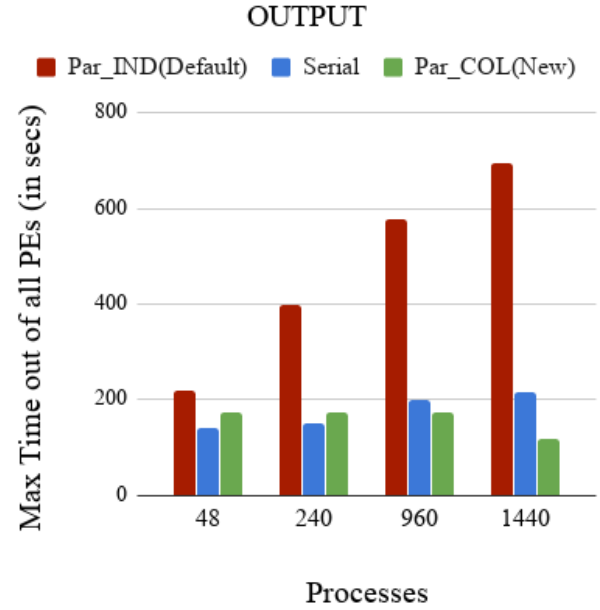


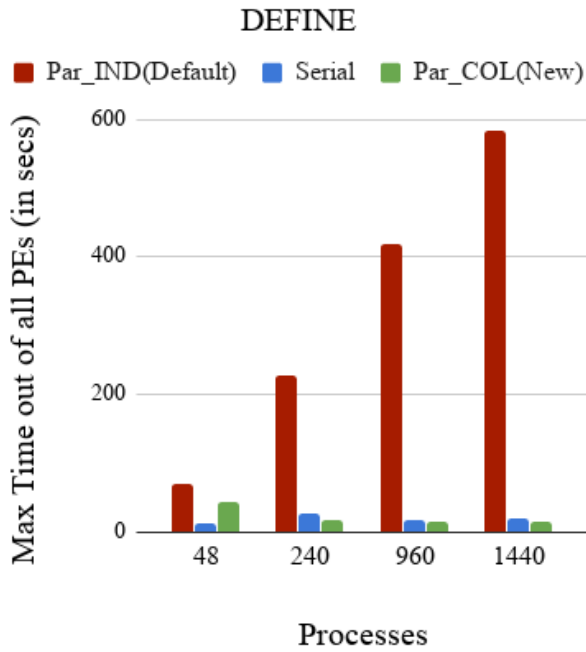
Figure 4.13: Scalability in Execution



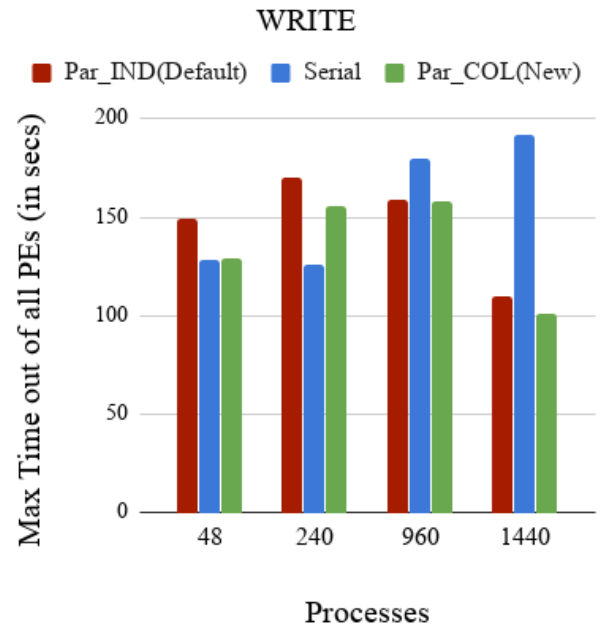
(a) Scalability in CPU time



(b) Scalability in Output time



(c) Scalability in Define time



(d) Scalability in Write time

Figure 4.14: New Scalability Results for our parallel NetCDF new setup (with collective calls) and comparison wrt parallel NetCDF default setup (with independent calls) and serial NetCDF – for 48,240,960 and 1440 processes

4.3 Selective Writing Approach

4.3.1 Motivation

There are different types of experiments performed on climate models. It can be done for simulation purposes to learn about the dynamics of nature or it can be used for weather forecasting. The output data in such experiments have spatial and temporal variability. Spatial variability is the variation of a parameter across space for any given time instant. An example of spatial variability is eddies where the variation in data is most important. Temporal variability is the variation of a parameter over time for any given spatial location. Temporal variability can be hourly, daily, monthly, intraseasonal(15 to 60 days), seasonal(180 days), annual(1 year), or interannual(multiple years). So, depending on the length of run and the experimental purpose, the frequency of writing the parameter changes.

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. Hence, this method will lead to the accuracy loss of the data. But, at the same time, it can reduce the total time consumed by the data write process. It can be implemented for those experiments where some compromise on the accuracy loss is acceptable. The variation captured by the selectively written data and the normally written data needs to be compared to examine the accuracy loss. Hence, there is a scope for selective writing strategy by keeping the accuracy loss within the tolerable limits to improve the overall I/O performance of climate models. We have implemented this strategy in the ROMS model within the acceptable accuracy loss constraints set separately for each parameter.

4.3.2 Initial Analysis – RMS error comparison for different output variables

The RMS difference for different parameters between consecutive records is studied. The main parameters written to output files are velocity, temperature and, salinity. It is checked that the time-consuming variables are the 3D variables(Each variable size is 40 times of the 2D variables as 40 is the depth). So, they are opted for selective strategy. It is found that the scale of variation in magnitudes of RMS difference is different for different parameters as shown in figure 4.15. So, different thresholds are required for different parameters.

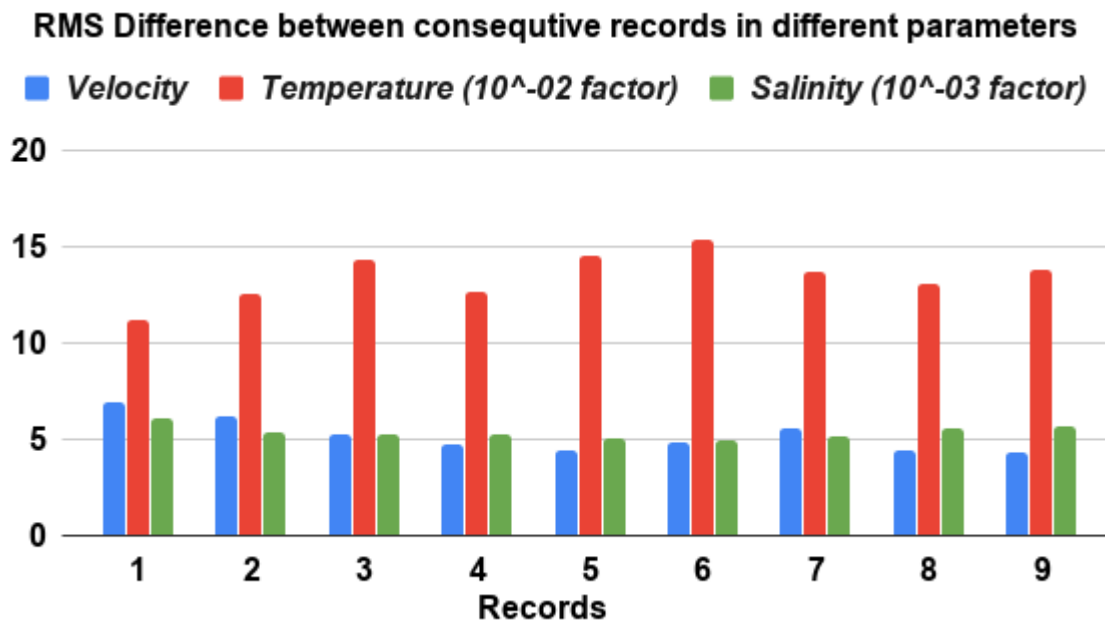


Figure 4.15: RMS difference comparison of different parameters

We can skip writing such parameters to reduce I/O where the RMS difference between two consecutive time steps is less than a certain threshold. The records written are encircled with a circle in figure 4.16. For the next writing step, it is compared with the last written record.

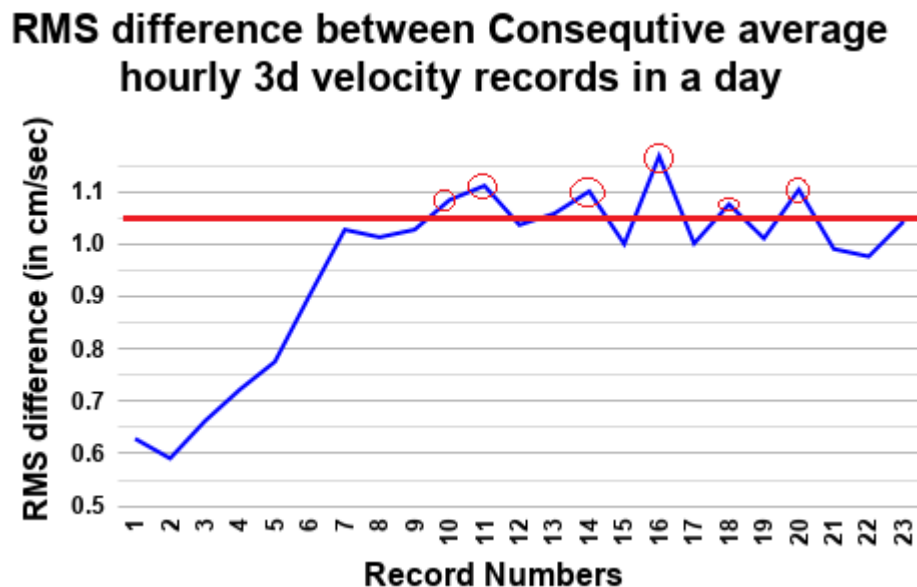


Figure 4.16: RMS Thresholding

4.3.3 Our Approach

Before writing the data, RMS difference between the present time-step and the last written time-step (Square root of the square of differences within all data points of a variable) is compared for each variable separately as shown in figure 4.17. If the Root Mean Square(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 value written. For each parameter, the NetCDF variable ID to be written in the output file is traced. A separate array is maintained to store the final processed data of the last time-step written to disk using the SAVE option in FORTRAN. If the data is written, the separate array is updated else it is kept unchanged.

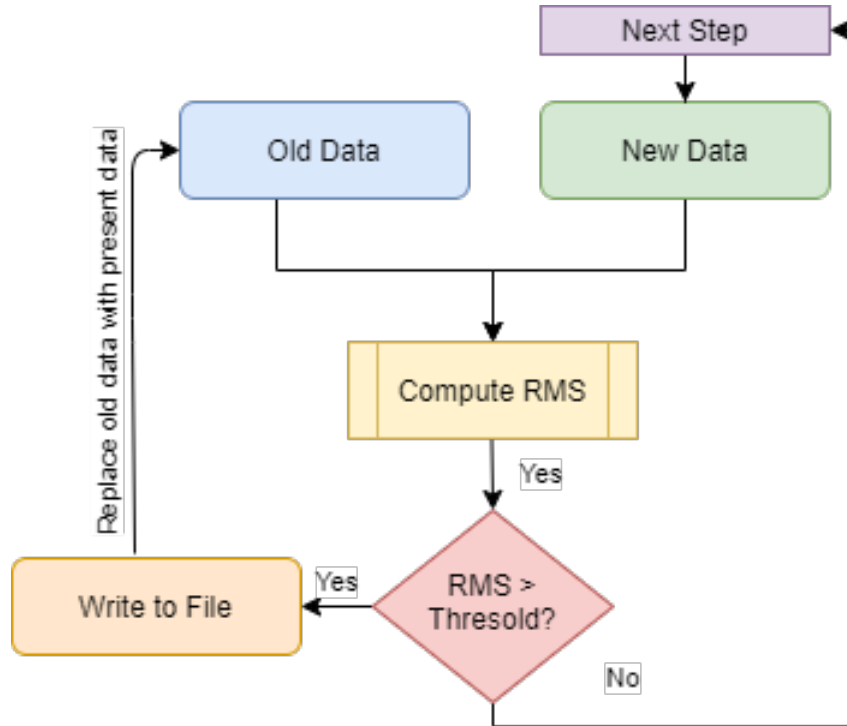


Figure 4.17: Selective writing approach

4.3.4 Accuracy Evaluation Strategy

Temporal Mean and Standard Deviation (S.D.) of each 3D variables are calculated to evaluate the data accuracy as shown in figure 4.18. Then, RMS difference between Temporal Mean (similarly Standard Deviation) with selective writing and without selective writing is calculated to check the accuracy loss due to selective writing strategy. The quality of data is also checked from the Surface Contour Plots of the difference between Temporal Means (similarly Standard

Deviation). The more the difference is closer to zero signifies better accuracy. The post-analysis of the data and the surface contour plots are done in PYTHON.

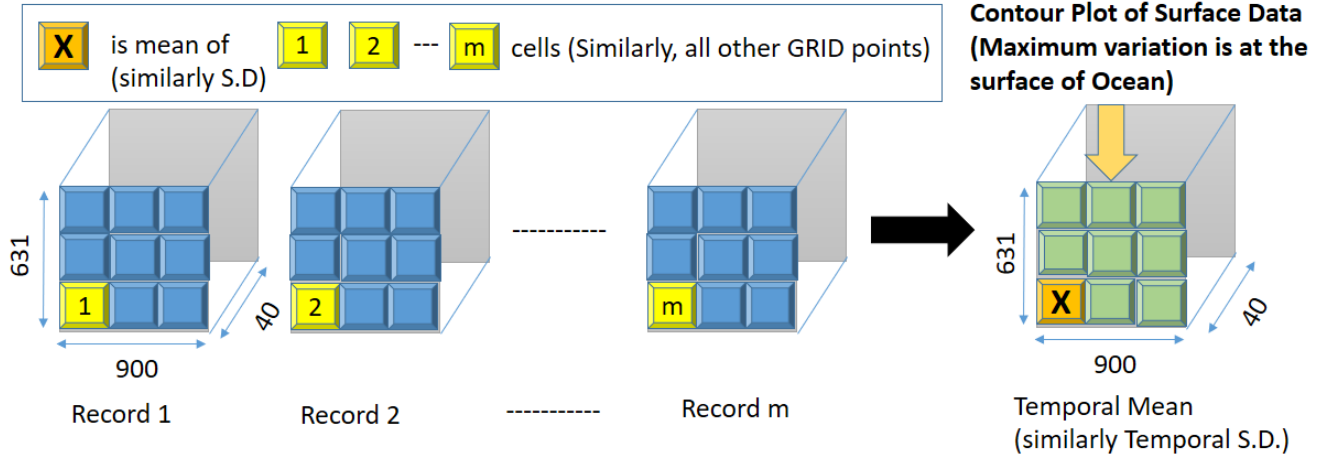


Figure 4.18: Temporal Mean and S.D. calculation and Surface Plotting Illustration

4.3.5 Experimental Setup

The experimental details of selective writing are given in Table 4.2. As 2D variables consume much lesser time compared to 3D variables, so selective writing is applied only on 3D variables.

Table 4.2: Experimental Setup for Selective Writing

Processes	1440 processes
Variable type	3D variables(time consuming variables)
Variables written	velocity (u,v,w), temperature, salinity
Output Files	Average file (Time-averaged data)
Threshold Used	Multiple Thresholds used for each variable
Write Frequency	Hourly records(15time-steps*240(step-size)/3600=1 hour)
Length of Run	15 days(5400 time-steps*240(step-size)/3600/24=15 days)

There are 24 records per day (hourly records) without selective writing. Daily data for all variables are saved in a single file for post-analysis. To check the feasibility of the approach, the accuracy loss and write reduction analysis are done on a daily basis (Day 1, Day 11 and Day 15). Similarly, it is done on multiple days (Day 1 to 5, Day 1 to 10 and Day 1 to 15). Velocity threshold used in the experiments for the reduction gained versus accuracy loss analysis are mentioned in Table 4.3.

Table 4.3: 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

The experiments were also performed on daily records (1 record per day). It is found that the deviation is more than the tolerable limits for daily records even with the lowest possible thresholds. It is because daily record data have more variance compared to hourly records.

4.3.6 Results and Observations

It is observed that a separate threshold is required for each variable depending on the scale of variation and level of accuracy required for each of them. For each variable, it is observed that the number of writes have reduced with increasing thresholds with a reduction in accuracy. For a given threshold, the accuracy loss is increasing for the velocity variable with the increase in the number of days. Whereas for temperature and salinity, the accuracy loss and write time reduction have almost remained the same for up to 10 days of run. The detailed analysis about the acceptable deviations and the corresponding thresholds selected are given next.

After discussing with the users of the ROMS model at Centre for Atmospheric and Oceanic Sciences(CAOS) department of IISc, the range of variation of each parameter and the corresponding range of acceptable loss in accuracy were decided as mentioned in Table 4.4.

Table 4.4: Range of variation and acceptable accuracy loss

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

The thresholds for different variables under the constraint of acceptable accuracy loss for selective writing of hourly data of velocity, temperature and salinity are decided from figures 4.19, 4.20, and 4.21 respectively. The maximum length of the run that meets the constraints of acceptable accuracy loss with the selected thresholds are also decided. The variation in the difference in Temporal Mean and S.D. with different thresholds for each variables is shown next.

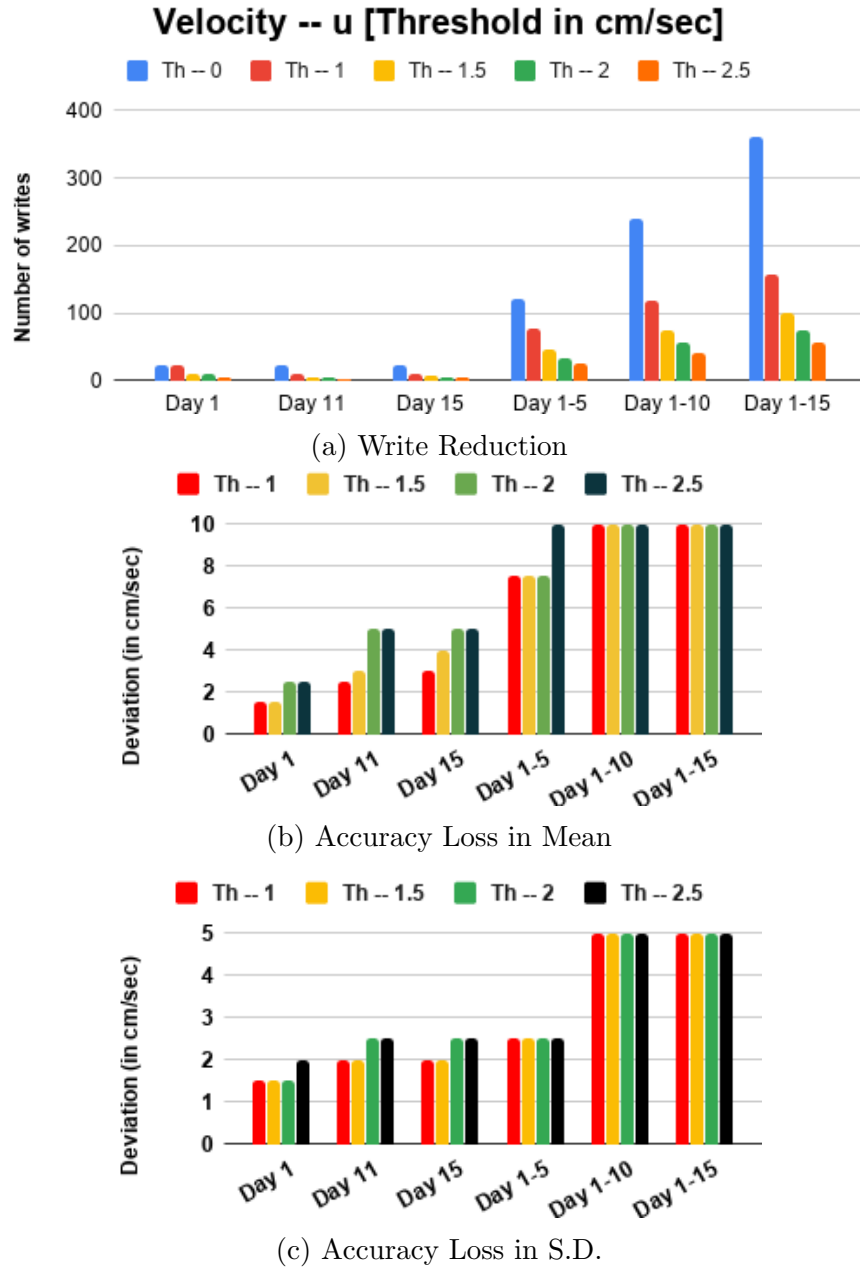


Figure 4.19: Selective Writing – Reduction Gained versus Accuracy Lost – VELOCITY

Observations – Threshold of 1.5 cm/sec 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 7.5 cm/sec for 5 days analysis. However, there is a steep jump 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. This threshold is giving around 60% reduction in number of writes for 5 days of run.

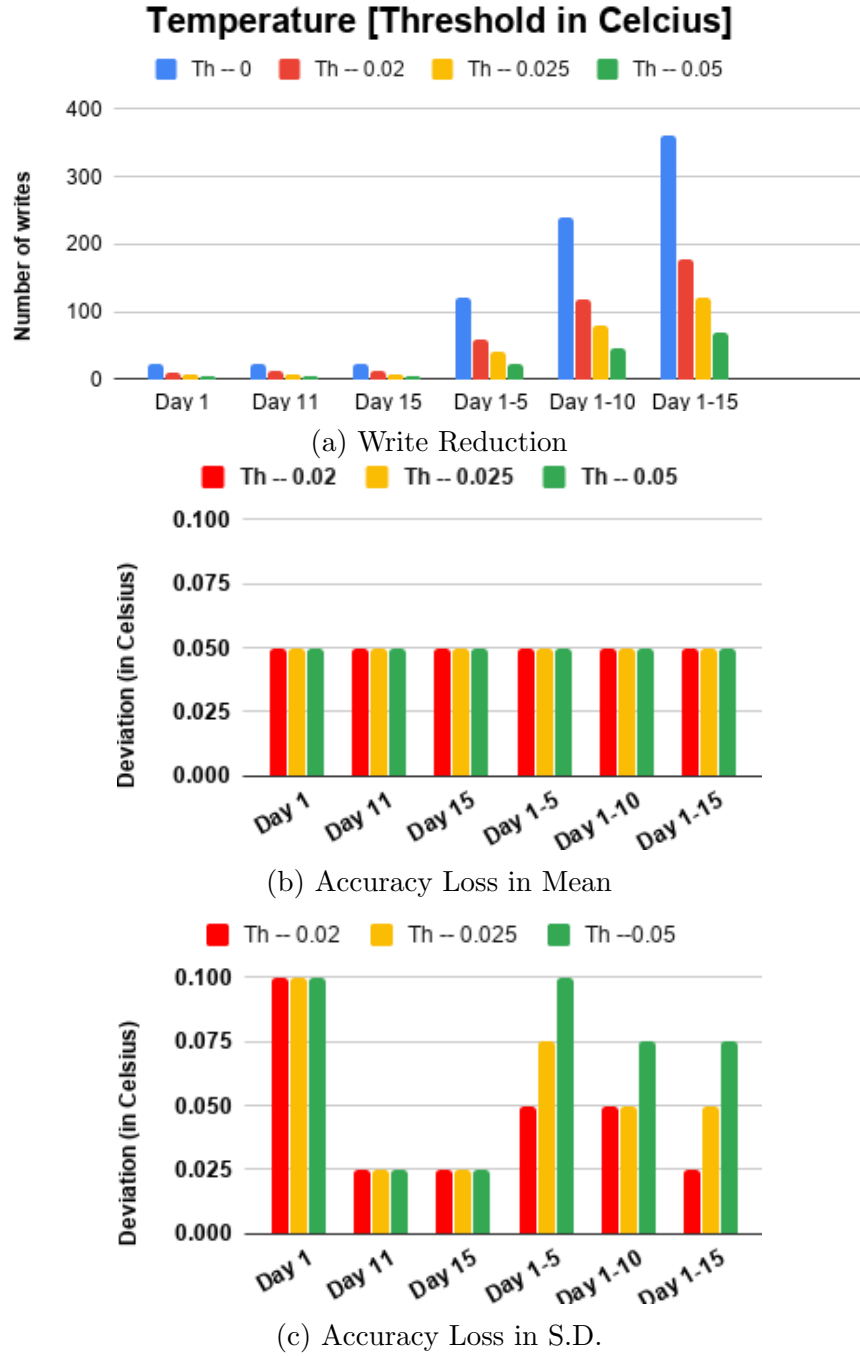


Figure 4.20: Selective Writing – Reduction Gained versus Accuracy Lost – TEMPERATURE

Observations – All thresholds are meeting the difference in temporal means constraint of 0.05 Celsius. However, only threshold of 0.02 Celsius is meeting 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. It is giving around 50% reduction in number of writes for 5 days of run.

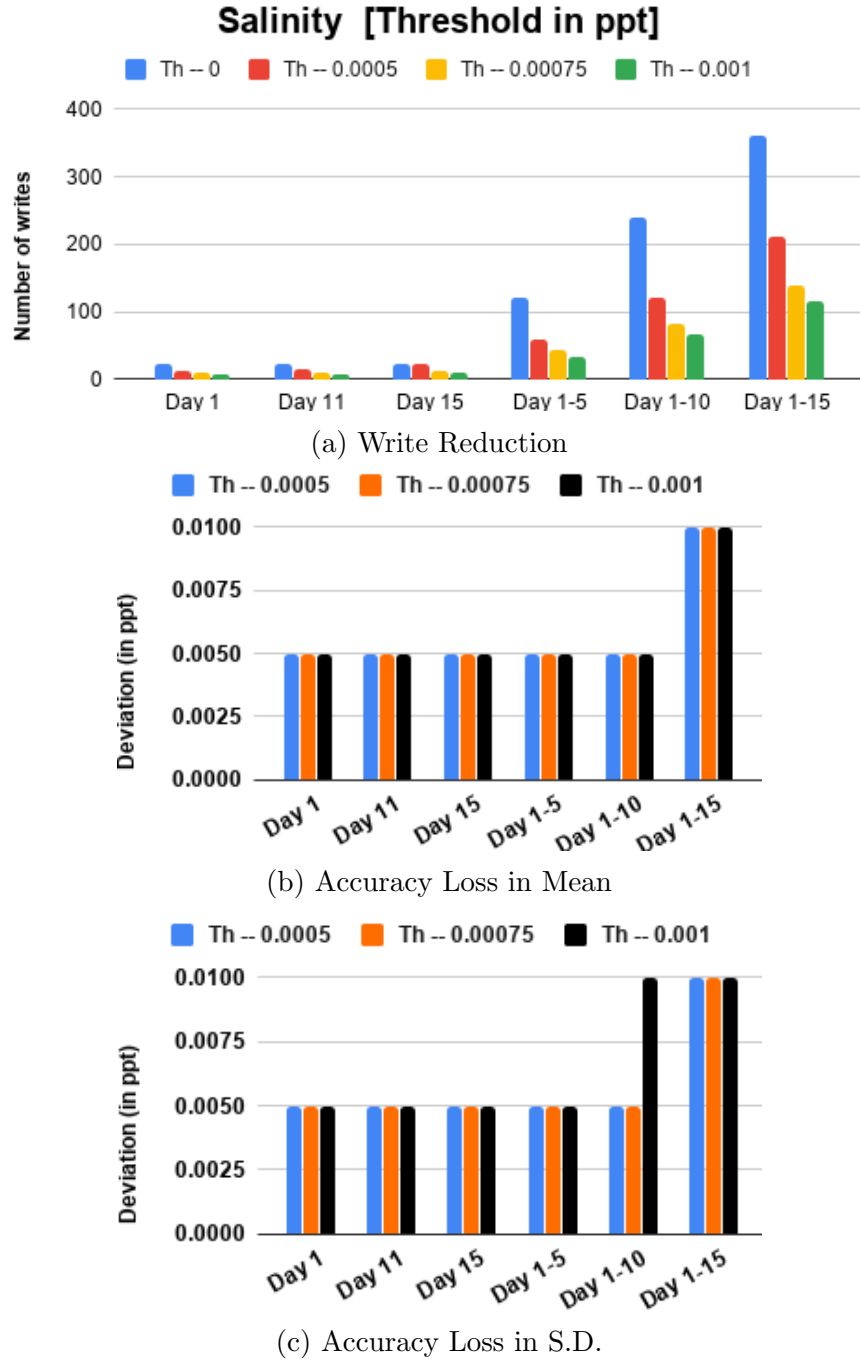


Figure 4.21: Selective Writing – Reduction Gained versus Accuracy Lost – SALINITY

Observations – All thresholds are meeting the difference in temporal means constraint of 0.005 ppt. However, up to a threshold of 0.00075 ppt is meeting 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. It is giving around 65% reduction in the number of writes for 5 days of run.

Table 4.5: Threshold selected and allowed length of run using selective approach

Variable	Threshold selected	Acceptable Length of Run
Velocity	1.5 cm/sec	upto 5 days analysis
Temperature	0.02 Celsius	upto 15 days analysis
Salinity	0.00075 ppt	upto 10 days analysis

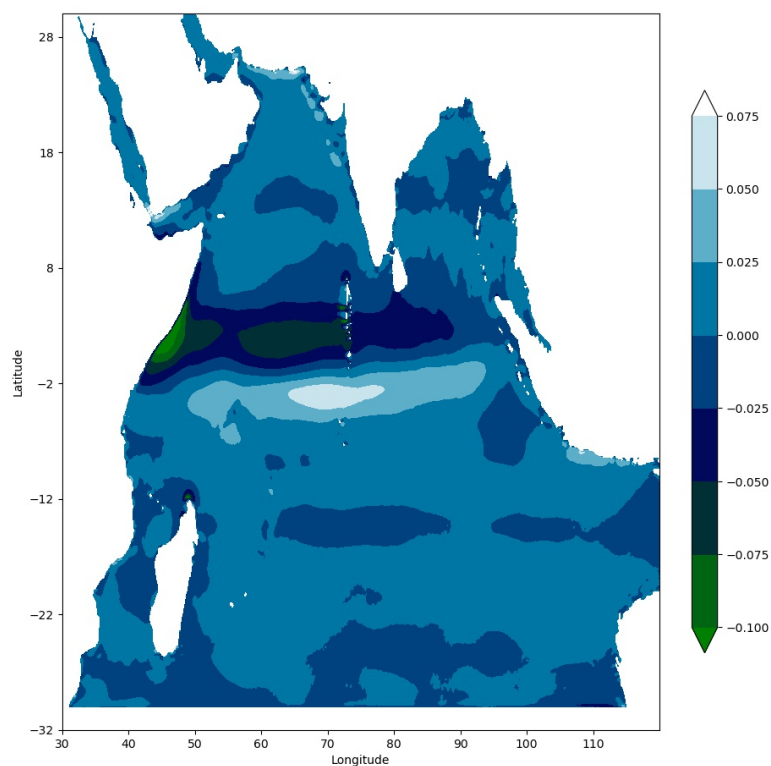
The thresholds selected for each parameter and the corresponding allowed length of runs decided based on the accuracy loss constraints are shown in Table 4.5. The selective writing performance improvement experiment is performed on 5 days as it is the common acceptable length of run in all variables. The experiments are done on top of 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 4.6.

Table 4.6: Selective Writing Time Reduction Results for 5 days of run

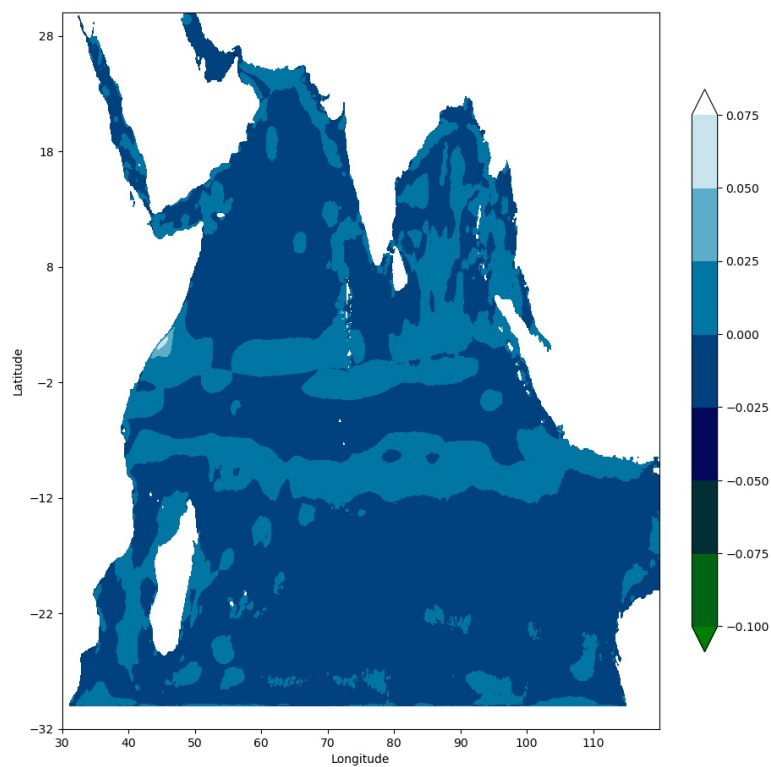
Type	Execution Time	Output	nf_fwrit3d function
Non-Selective i.e. Default	24.36 mins	14.2 mins	12.98 mins
Selective(Reduction in %)	16.66 mins (31.6%)	6.83 mins (51.87%)	5.26 mins (59.43%)

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. 53% of the execution time is consumed by the nf_fwrit3d function in case of non-selective writing. Hence, this strategy is giving significant reduction in execution time.

The surface contour plots of differences in Temporal Mean(similarly Temporal S.D.) of data without and with selective writing for velocity, temperature, and salinity are shown in figure 4.22, figure 4.23, and figure 4.24 respectively. Some specific regions of the temporal means are found to have maximum velocities. The difference in temporal means is also found maximum in those regions. So, those regions are primarily constrained within the allowed threshold 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 0.05 Celsius for a threshold of 0.02 Celsius. Similarly, both temporal mean and S.D. for salinity are within 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 only up to 5 days.

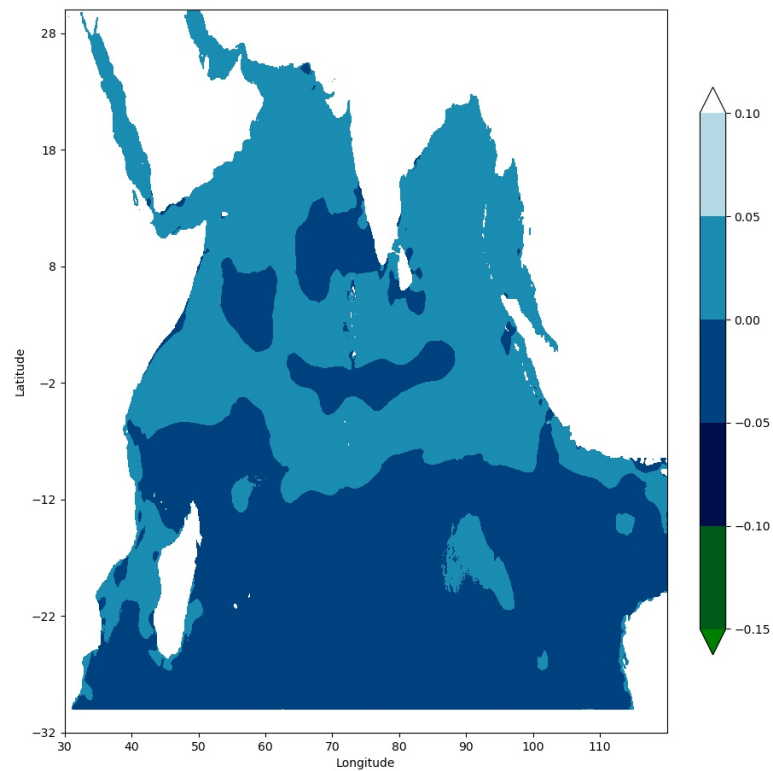


(a) Difference in Temporal Mean

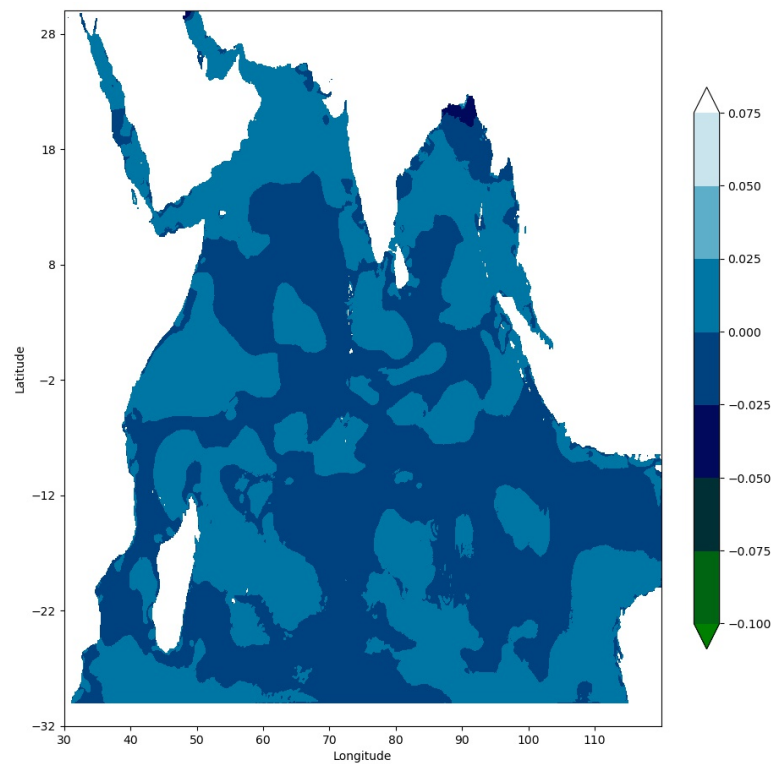


(b) Difference in S.D.

Figure 4.22: Results – Accuracy Loss in VELOCITY with the selected Threshold of 1.5cm/sec for 5 days of run

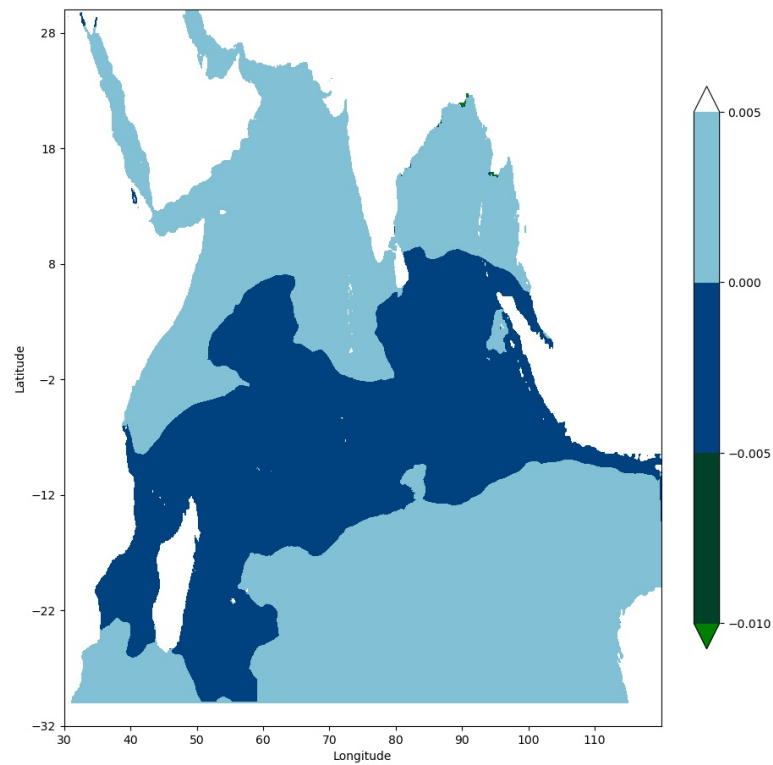


(a) Difference in Temporal Mean

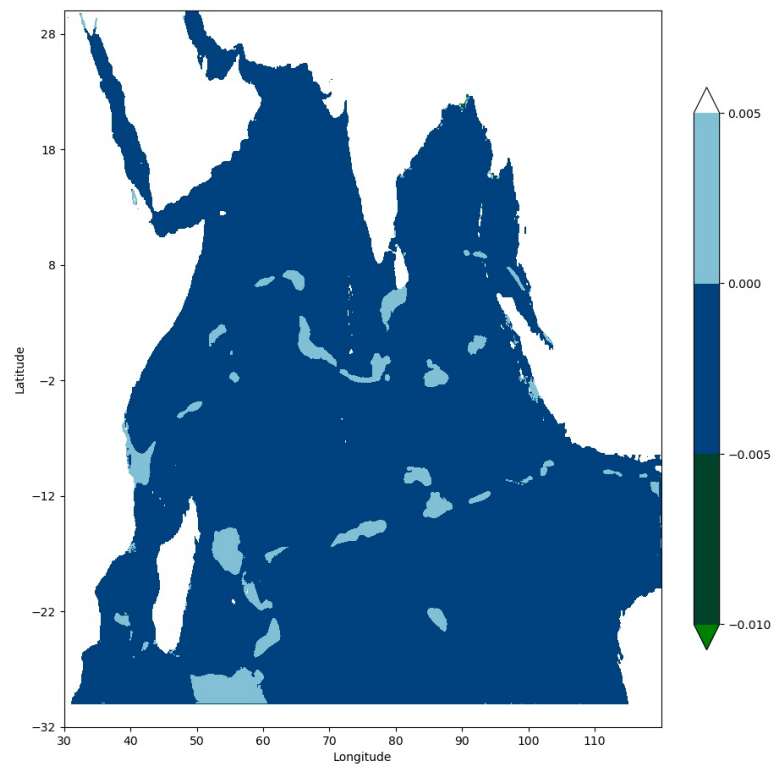


(b) Difference in S.D.

Figure 4.23: Results – Accuracy Loss in TEMPERATURE with the selected Threshold of 0.02 Celsius for 5 days of run



(a) Difference in Temporal Mean



(b) Difference in S.D.

Figure 4.24: Results – Accuracy Loss in SALINITY with the selected Threshold of 0.00075 ppt for 5 days of run

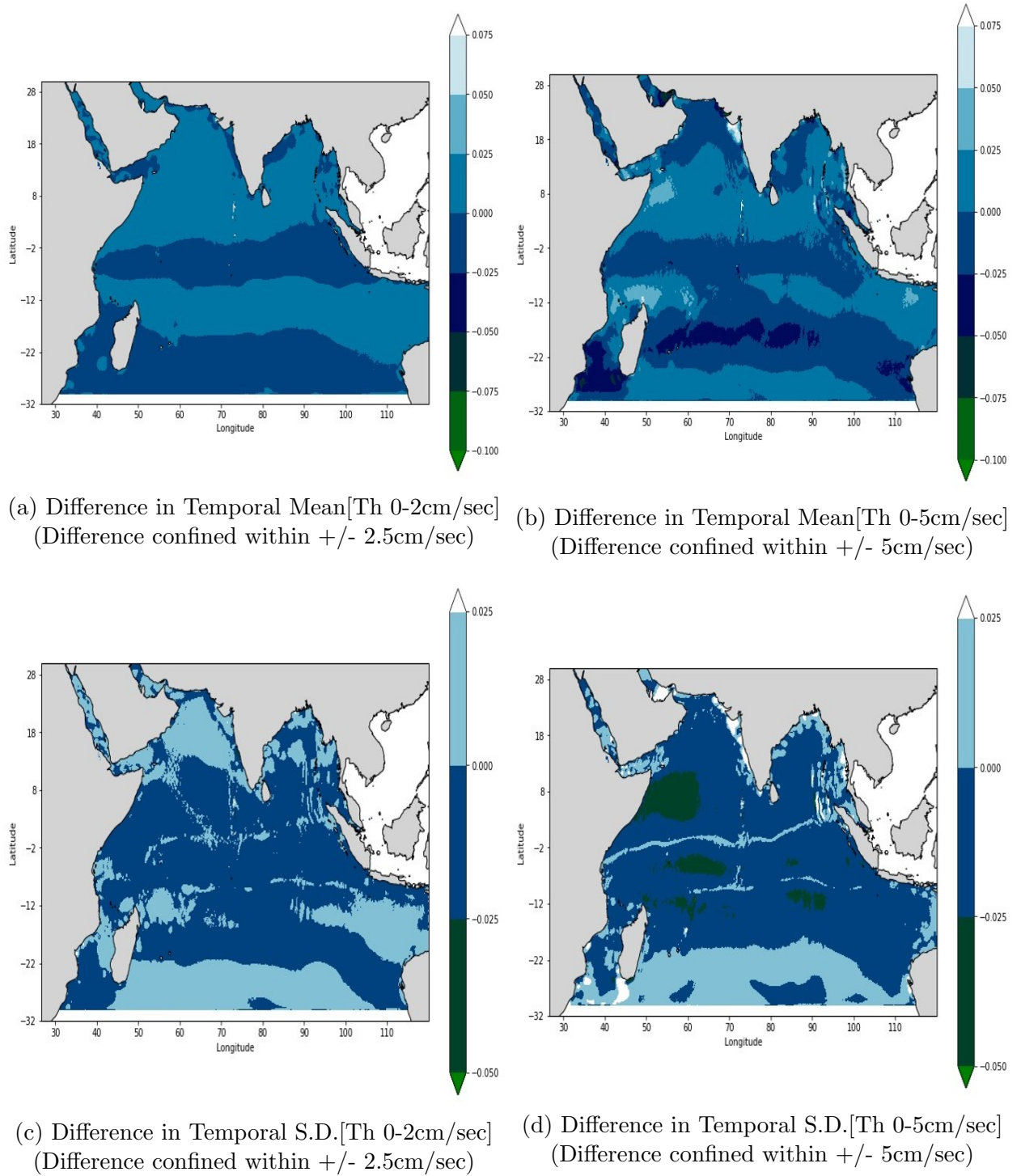


Figure 4.25: Selective Writing Accuracy Evaluation Illustration

LEFT – Plot a (similarly c) is the surface contour plot of differences in Temporal Mean (similarly Temporal S.D.) of data without selective writing i.e. Th 0cm/sec and with selective writing of Th 2 cm/sec, RIGHT – Plot b (similarly d) is the surface contour plot of differences in Temporal Mean (similarly Temporal S.D.) of data without selective writing i.e. Th 0cm/sec and with selective writing of Th 5 cm/sec.

[Difference should be close to zero for better accuracy. As the threshold increases, accuracy drops]

4.3.7 Conclusion

So from the contour plots, it is concluded that the variation in the velocity variable is more as compared to variation in temperature and salinity. The high-velocity regions are leading to the responsible for the accuracy loss as the difference in the temporal mean is maximum in those regions. The large increase in accuracy loss for increasing the velocity threshold from 2cm/sec to 5cm/sec is shown in figure 4.25. 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. It 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. So, it is better to write all the 2D variables.

Typically, ocean prediction systems are good for 5 days. The quality of forecast degrades as a function of time and beyond 5 days the quality becomes poor. Hence, based on the acceptable range of accuracy loss and the results, it is concluded that this method is suitable for ocean forecast systems. Also, it is concluded that the daily means and S.D. are acceptable throughout run from 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 this approach is that it can help us to do weather forecasting faster. But the disadvantage is the accuracy loss increases 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.

4.4 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 [5] 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 help in the improvement of the I/O performance. The I/O performance is improved when processes access multiple OSTs in parallel as in figure 4.26. Also, the number of OSTs, which each process needs to access should be minimized by stripe alignment.

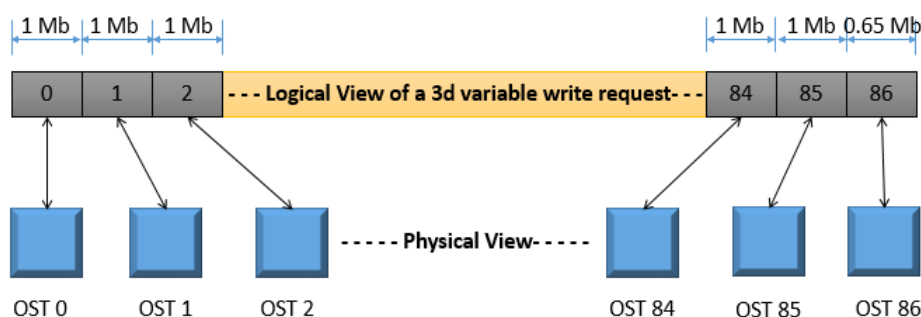


Figure 4.26: Physical and Logical views of each write request of 3d variables (86.65 Mb) for stripe size of 1 Mb and Stripe count of -1 (i.e. maximum 96 in CRAY Lustre file system)

4.4.1 Experimental Setup

The experiment is done on different sets of processes to find the best lustre striping that can be achieved with different number of processes as compared to the default stripe setting. The stripe count is varied from 1 to the maximum available OSTs and different stripe sizes from 1 to 32 Mb is used for the experiments. 3D variables are used as they consume most of the write times as mentioned earlier. Total 64 experiments were performed.

Table 4.7: 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)

4.4.2 Results and Observations

Table 4.8: 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

The write timings obtained for default setup (stripe count of 4 and stripe size of 1Mb) and for the best combinations (Table 4.8) are shown in figure 4.27. Significant improvement is seen with the best combinations compared to the default for all different combinations of processes used. 41 % improvement in data writing is obtained for 1440 processes. The improvements for 48, 240 and 960 are 57%, 61% and 55% respectively. So, this method is giving around 40% to 60% performance improvement in writing phase depending on number of processes used. The best timings found are reducing with increasing number of processes.

It is observed that for 240, 960, and 1440 processes, the max stripe count with a stripe size of 1Mb performed best whereas 48 processes worked best with stripe count of 1 and stripe size of 32Mb. The 3D variables are of size 86.65 Mb. The stripe size of 1 Mb distributes the data to the first 87 OSTs minimizing the number of processes accessed per OST for a large number of processes as shown in figure 4.26. For a small number of processes, the benefit of using multiple OSTs is lost because of dominating I/O contention for using multiple OSTs.

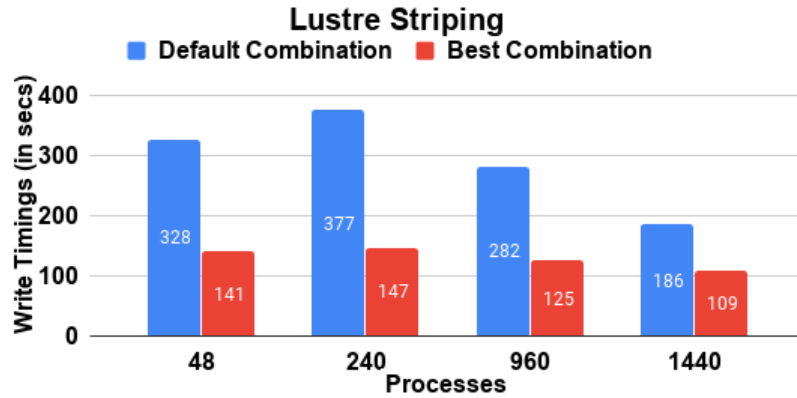


Figure 4.27: Write Timing improvement results using Lustre Striping [Default Combination is Stripe Size 1 Mb , Stripe Count 4 OSTs and for Best Combination refer Table 4.8]

4.4.3 Conclusion

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 in parallel, the stripe count should be 1 with a large stripe size. For Parallel I/O, the creation of large files with low stripe count can lead to I/O bottleneck for a large number of processes. 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 than the maximum OSTs available (96 in CRAY 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.

4.5 Combining the different I/O optimization techniques

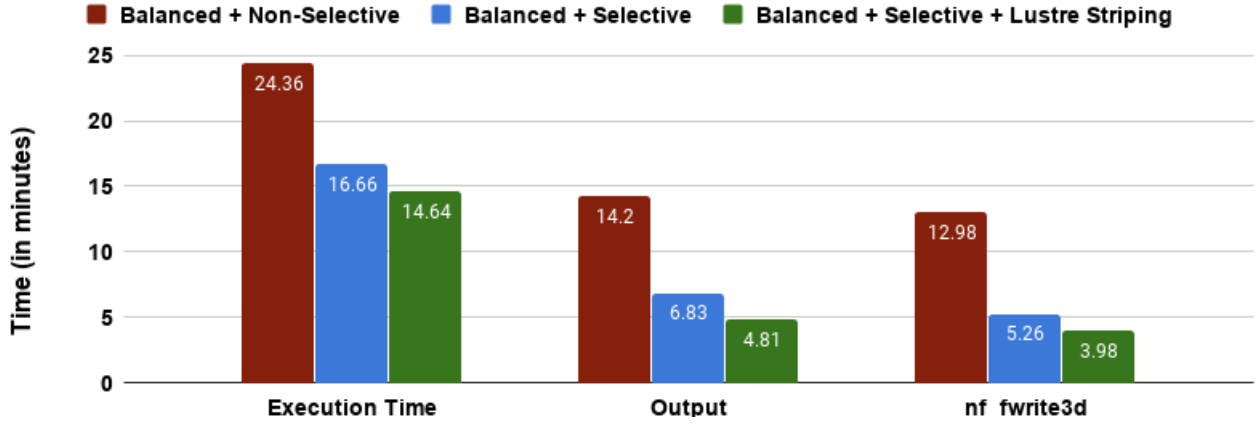


Figure 4.28: Combined improvement results – Actual Times

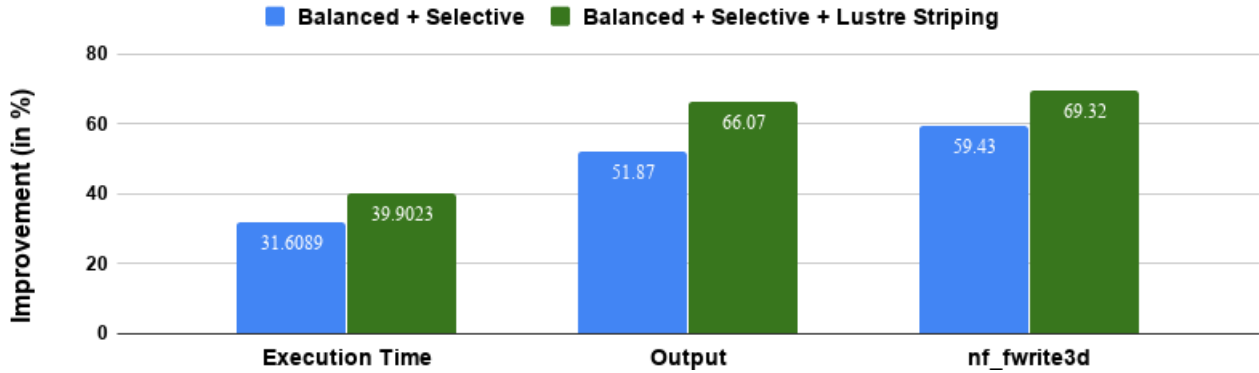


Figure 4.29: Combined improvement results – In percentage

The three different I/O improvement strategies are combined here. The experimental setup is the same as the selective writing strategy result for 5 days of simulation present in Table 4.6. As mentioned earlier, the non-selective data writing and the selective data writing experiments shown previously are 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. It is observed that the lustre striping improved the writing phase by 10%, output phase by 15%, and `nf_fwrite_3d` function by 8% with respect to the improvement gained due to the selective writing strategy. As the data is compared with respect to balanced non-selective writing timings, the performance gains due to imbalance solving are not reflected in the results.

Chapter 5

Conclusions and Future Works

This chapter details the conclusions obtained from this project and the further works that need to be done in this area.

5.1 Conclusions

The detailed analysis of the ROMS model illustrated the different possible reasons that can be possible behind load imbalance and its impact on execution time. An efficient parallel I/O strategy can help us to improve the overall performance as I/O consumes a significant part of the total execution time for climate models. The primary challenge was to get complete control of the I/O strategy used in the ROMS model as it has large codebase and a large number of files are interlinked to perform the I/O.

The computation was found to scale but the issue was in the I/O for which the parallel NetCDF was not scaling. We used 1440 processes for our experiments as the computation was scaling well till 1440 processes. Load balancing strategy helped in improving the define phase performance (it improved the execution time of load balanced parallel NetCDF by 85% wrt default setup for parallel NetCDF and by 27% wrt serial NetCDF for 1440 processes) whereas the selective writing strategy implemented on top of the load-balanced model setup (it improved the execution time of parallel NetCDF by 31% within allowed accuracy loss constraints) and the lustre striping technique (it improved the writing phase of I/O output time by 41% for parallel NetCDF with 1440 processes) also helped in improving the write phase performance. All the three strategies together improved parallel NetCDF performance by about 60% with respect to the serial NetCDF setup. Hence, the I/O performance and the overall wallclock times are improved.

A similar approach can be examined in case load imbalance is detected from the initial analysis for other climate models. The selective writing can be implemented in other models where a lot of I/O output is involved. The striping technique can also be implemented for other file systems for achieving similar performance improvements.

5.2 Future Works

The next goal is to check other aspects to improve the I/O performance. Methods to further improve the parallel I/O over serial I/O need to be designed. For climate data, it is seen that the data varies majorly in some specific regions. Methods to perform selective writing strategy on a region basis need to be examined. The sub-filing technique can be used for implementing this region basis selective writing. Selective writing may be applied to regions where change is minimal and writing all records for regions where changes are maximum. Implementation of collective I/O with space as well as time can be implemented so that the total file access can be reduced further as more data can be read as well as written in a single chunk together rather than multiple reading and writing of smaller chunks. Topology aware I/O also needs to be studied as resource contention may happen from network topology in the CRAY machine.

Bibliography

- [1] Craypat. URL <https://docs.nersc.gov/programming/performance-debugging-tools/craypat/>. 3
- [2] Esmf. URL <https://www.earthsystemcog.org/projects/esmf/>. 1
- [3] Geos. URL <https://gmao.gsfc.nasa.gov/systems/geos5/>. 5
- [4] Hdf5. URL <https://www.hdfgroup.org/>. 3
- [5] Lustre. URL <https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips>. 44
- [6] Netcdf, . URL <https://www.unidata.ucar.edu/software/netcdf/>. 3
- [7] Netcdf, . URL http://meteora.ucsd.edu/~pierce/ncview_home_page.html. 3
- [8] Wikiroms. URL https://www.myroms.org/wiki/Documentation_Portal. 1
- [9] Babak Behzad, Joey Huchette, Huong Luu, Ruth Aydt, Quincey Koziol, Mr Prabhat, Suren Byna, Mohamad Chaarawi, and Yushu Yao. Auto-tuning of parallel io parameters for hdf5 applications. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 1430–1430. IEEE, 2012. 6
- [10] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)*, 7(3):1–26, 2011. 6
- [11] William W Dai, Anthony J Scannapieco, Frederick L Cochran, Chong Chang, Paul M Weber, Elsie L Sandford, and Britton A Girard. Buffering io for data management in multi-physics simulations. In *Proceedings of the High Performance Computing Symposium*, pages 1–8, 2013. 6

-
- [12] 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*, volume 2003, pages U41B–0009, Dec 2003. 6
- [13] Kui Gao, Wei-keng Liao, Alok Choudhary, Robert Ross, and Robert Latham. Combining i/o operations for multiple array variables in parallel netcdf. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009. 6
- [14] Kui Gao, Wei-keng Liao, Arifa Nisar, Alok Choudhary, Robert Ross, and Robert Latham. Using subfilng to improve programming flexibility and performance of parallel shared-file i/o. In *2009 International Conference on Parallel Processing*, pages 470–477. IEEE, 2009. 6
- [15] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Robert Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netcdf: A high-performance scientific i/o interface. In *SC’03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 39–39. IEEE, 2003. 6
- [16] Zhuo Liu, Bin Wang, Teng Wang, Yuan Tian, Cong Xu, Yandong Wang, Weikuan Yu, Carlos A Cruz, Shujia Zhou, Tom Clune, et al. Profiling and improving i/o performance of a large-scale climate scientific application. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7. IEEE, 2013. 5
- [17] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008. 5
- [18] Russell K Rew, B Ucar, and EJ Hartnett. Merging netcdf and hdf5. In *20th Int. Conf. on Interactive Information and Processing Systems*, 2004. 6
- [19] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. Modular hpc i/o characterization with darshan. In *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, pages 9–17. IEEE, 2016. 6
- [20] Houjun Tang, Xiaocheng Zou, John Jenkins, David A Boyuka, Stephen Ranshous, Dries Kimpe, Scott Klasky, and Nagiza F Samatova. Improving read performance with online access pattern analysis and prefetching. In *European Conference on Parallel Processing*, pages 246–257. Springer, 2014. 6

-
- [21] Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective i/o in romio. In *Proceedings. Frontiers' 99. Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE, 1999. [6](#)
 - [22] Yuichi Tsujita, Atsushi Hori, Toyohisa Kameyama, Atsuya Uno, Fumiyoshi Shoji, and Yutaka 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*, pages 12–23, 2018. [6](#)
 - [23] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. Characterizing output bottlenecks in a supercomputer. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2012. [6](#)