Manuscript prepared for Geosci. Model Dev.
with version 5.0 of the LaTeX class copernicus.cls.
Date: 22 August 2013

A fast I/O library for high resolution climate models

Xiaomeng Huang, Wencan Wang, Haohuan Fu, Guangwen Yang, Bin Wang, and Cheng Zhang

Ministry of Education Key Laboratory for Earth System Modeling,
and Center for Earth System Science, Tsinghua University,
Beijing, 100084, China

Correspondence to: Xiaomeng Huang
(hxm@tsinghua.edu.cn)

Abstract. We describe the design and implementation of Climate Fast I/O(CFIO) library for high resolution climate models in this paper. CFIO provides a simple method for modelers to overlap the I/O phase with computing phase automatically, so as to shorten the execution time of numerical simulation. To minimize the required code modifications, CFIO provides the similar interfaces and features to the PnetCDF, which is a frequently used I/O library in climate models. We deployed CFIO in three high resolution climate models , including two ocean models(POP and LICOM) and one sea ice model(CICE). The experimental results show CFIO improves the performance of climate models significantly versus PnetCDF and the original serial I/O approach. The maximum speedup for POP, CICE, and LICOM, running in 0.1 degree resolution and with about 1,000 CPU cores, are 7.9 times, 4.6 times and 2.0 times respectively.

1 Introduction

Scientific computing for climate modeling has undergone radical changes over the past decade. One major trend is to increase the resolution of the models, so as to provide finer simulation of the physics processes of the atmosphere, ocean, land, and sea ice. This trend is motivated by the availability of supercomputers with core counts in the range of tens to hundreds of thousands.

With a higher resolution, the amount of data generated by climate models will be significantly larger than before. Modelers who provided scientific data for the Fifth Assessment Report of the United Nations Intergovernmental Panel on Climate Change (IPCC AR5) must run coupled climate models for a long period of time, to simulate various types of climate change scenarios and to address hundreds of terabytes of resulting data. The output of such a large amount of data will result in severe performance degradation for numerical simulation experiments.

A lot of attention has been drawn to parallel I/O for scientific computing. Corbetty et al. (1996) proposed an MPI-IO library to process file streaming in low level. MPI-IO is currently the most widely used parallel I/O library in scientific computing. For modern scientific applications, high-level I/O libraries that are based on multidimensional arrays and provide self-describing information are more convenient for data analysis and post-processing. The netCDF format designed by Rew and Davis (1990) is the de facto standard for climate models. Since netCDF library only provides serial accessing interfaces, Li et al. (2003) developed PnetCDF library to improve the performance of netCDF library. PnetCDF is widely used by the climate community and different climate models. PIO designed by Dennis et al. (2012) is an application level parallel I/O library that supports multiple back-end I/O libraries. PIO was developed for the Community Earth System Model(CESM) and mainly adopts a data rearrangement method in the memory to improve the disk I/O performance. Another library, ADIOS designed by Lofstead et al. (Lofstead et al. (2009), Lofstead et al. (2008)) uses a novel BP format to store multidimensional arrays, which can eliminate the overhead of consistency checking. ADIOS makes it possible to independently select the I/O methods that are used for each grouping of data, so that users can use I/O methods that exhibit the best performance based on both I/O patterns and the underlying hardware. However, it also involves a porting cost from netCDF format to BP format.

Most of the above libraries attempts to improve the I/O throughput through parallelization techniques. For real applications, the overall execution time mainly consists of two aspects: computing time and I/O time. The various proposals are helpful for shortening the I/O time of large-scale data. However, the computing phase need to wait for the imple-

mentation of the last I/O phase in iterative simulations. In a sense, the I/O phase and the computing phase are still serial. There is a big space to improve I/O efficiency through overlapping of I/O phase and computing phase.

With these issues in mind, we design and implement Climate Fast I/O (CFIO), a parallel I/O library that has been specifically developed for climate models. The main idea of CFIO is to apply an I/O forwarding technique to provide automatic overlapping of I/O with computing. To minimize the required code modifications for switching to the CFIO, CFIO provides similar interfaces and features to PnetCDF. The experimental results show that, although the throughput of CFIO achieves only 90% of PnetCDF, the I/O time of the MPI test application with CFIO achieves a 4.17x speedup compared with PnetCDF. We also test CFIO on three real climate model: Parallel Ocean Program (POP,Smith et al. (2010)), The Community Ice Code (CICE,Hunke and Lipscomb (2010)) and LASG/IAP Climate systems Ocean Model(LICOM,Yu et al. (2012)). The maximum speedups for POP, CICE, and LICOM, running in 0.1 degree resolution and over 1,000 CPU cores, are 7 .9 times, 4.6 times and 2.0 times respectively. The open source code and documents can be downloaded from Github website(https://github.com/chengzhang/cfio).

The remainder of this paper is organized as follows. Section 2 discusses the motivation and the main idea of CFIO. The design and architecture of CFIO is presented in section 3 in detail. Section 4 evaluates and analyses the performance of CFIO. Section 5 introduces the related work. Conclusions and possible future work are discussed in section 6.

2 Motivation

In traditional climate models, the computing phase and the I/O phase run alternately. The computing phase performs simulation for a specific time, and then the I/O phase outputs results following each computing phase. Namely, the computing phase and the I/O phase for traditional climate models are serial.

In fact, for most of current climate models, the initial conditions or datasets for processing will be loaded at the starting phase. Then the restart files, which contain all of the initial condition information that is necessary to restart from a previous simulation, will be written to file system at a fixed frequency. Finally, the historic files, which include all of the diagnostic variables, will also be written to file system at certain frequency. In general, there are no random seeks and no read-after-write throughout the file, and writing operations are usually append-only for all of the appropriate parts of the initial files, restart files and historic files.

Because of the append-only data accessing patterns with no random seeks and no read-after-write, a specific climate I/O library does not need to keep the integrity of the transactional writing. In other words, the computing step does not need to wait for the completion of the last I/O step.

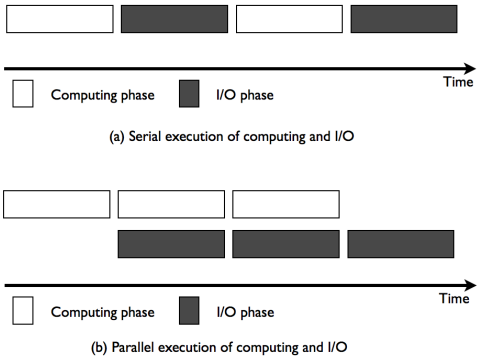

Fig. 1. Overlapping I/O with Computing

Motivate by this observation, we consider the possibility of overlapping the I/O phase with the computing phase. As shown in Fig. 1, we hope the computing and the I/O phases can be performed in parallel. Compared with the serial computing and I/O method, the I/O time will be hidden by the computing time in parallel computing and I/O method. It will be useful to shorten the execution time of climate models.

Another advantage of overlapping the computing phase with the I/O phase is that the efficiency of computing and storage resource can be improved. For the serial method, the computing resource is idle when the I/O phase is executing. In the contrary, the storage resource is idle when the computing phase is executing. For the parallel method, the computing phase and the I/O phase are both pipelined. The efficiency of computing and storage resource are always fully utilized.

3 Design of CFIO

This section will describe how to achieve overlapping of computing phase and I/O phase through I/O forwarding technique. The CFIO interface and a simple example will be given.

3.1 I/O Forwarding System Architecture

The overlapping technique is an established method for improving the performance of a parallel program. CFIO takes advantage of the computing pattern to reduce the I/O impact, and uses I/O forwarding to facilitate automatic overlapping of I/O with computing.

The diagram of I/O forwarding technique is show in Fig. 2. When the climate model uses CFIO as its I/O method, extra I/O processes are launched in addition to the original compute processes. The compute processes are in charge of numerical computing. All of the I/O requests generated in the compute processes will be forwarded to the I/O processes;

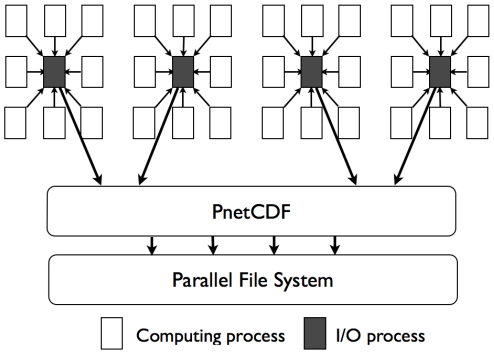

Fig. 2. The schematic diagram of I/O forwarding technique

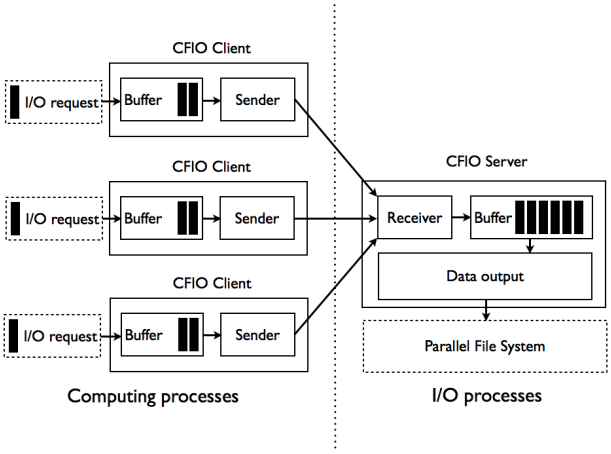

Fig. 3. The system architecture of CFIO

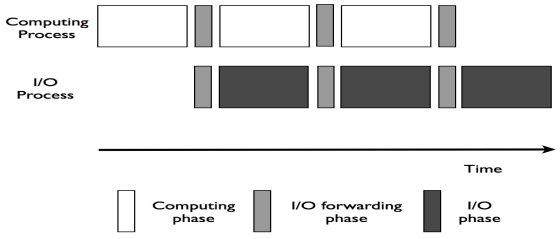

Fig. 4. Overlapping I/O with computing through I/O forwarding

the compute processes can compute continuously without I/O blocking. The I/O processes are responsible for the data output through calling parallel I/O library. The I/O processes acts as a very large buffer pool and can be used to exchange data efficiently.

The advantages of dividing the processes into compute processes and I/O processes individually are obvious. For one compute node, forwarding I/O requests to other nodes is useful for decreasing the local resource competition with respect to CPU and memory. In addition, the independent I/O processes have a very large buffer, to make certain optimized operations possible, such as data aggregation and rearrangement. The non-continuous writing of small data blocks can be transformed into continuous writing of large data blocks, to make it easy to improve the accessing performance of the parallel file system.

The I/O forwarding system architecture of CFIO is shown in Fig. 3. The CFIO client is co-located with the compute process and provides the climate model with a series of interfaces for accessing the model data. When an I/O request is generated in a compute process, the client will pack the request into a buffer and then send it to the server via MPI communication.

The CFIO server running as a daemon program in the I/O process is used to receive messages and perform I/O requests. It can receive I/O requests from multiple clients. After receiving an I/O request, the CFIO server places the request in an I/O queue. Because all of the CFIO functions are collective I/O operations, I/O requests of each client arrive and enqueue in the same order. When performing I/O requests, the server dequeues one I/O request from queue and unpacks it. Next, it calls the corresponding PnetCDF function to perform the actual I/O operation. For data write operations, data aggregation will be performed to gather subarray data from each client into a large array of data. Because PnetCDF is based on MPI-IO, CFIO can benefit from the use of collective I/O in MPI-IO. MPI-IO also supports portability and optimization with different file systems.

As shown in Fig. 4, the data will be forwarded from computing process to I/O process. Thus the whole execution time of simulation consists of computing time, I/O forwarding time and I/O time. One possible scenario that we must consider is when the I/O time is greater than the computing time, in which case the I/O latency cannot be hidden by the computing phase. In this scenario, we can increase the number of I/O processes that are used to solve the problem. More I/O processes provide a larger buffer pool, which can accommodate more data. At the same time, we can clear the buffer pool faster through more I/O processes compared with the previous approach. This method is useful to shorten the I/O time and to make the I/O phase overlap with the computing phase perfectly.

3.2 Speedup Analysis

We can analyze the speedup of program that we can obtain by using CFIO in climate model through simple calculation. We denote the execution time of a model with CFIO and its default I/O approach as T_{cfio} and T_{origin}. As shown in Fig. 4, T_{cfio} and T_{origin} can be calculated as follows :

$$T_{origin} = T_{compute} + T_{io} \quad (1)$$

$$T_{cfio} = max\{T_{compute} + T_{send}, T_{recv} + T_{io}\} \quad (2)$$

where $T_{compute}$ and T_{io} are the corresponding computing time and I/O time in one simulation step, T_{send} and T_{recv} are the time of sending I/O requests in the client and the time of receiving I/O requests in the server.

If T_{cfio} is assigned with the value of $(T_{recv} + T_{io})$, it means that the I/O latency cannot be hidden by the computing phase. As mentioned above, this scenario can be avoided by increasing the number of I/O processes. So the ideal situation for T_{cfio} is $T_{cfio} = T_{compute} + T_{send}$. The speedup S of using CFIO can be calculated as :

$$S = \frac{T_{origin}}{T_{cfio}} = \frac{T_{compute} + T_{io}}{T_{compute} + T_{send}} \quad (3)$$

An upper bound on speedup can be derived as :

$$S < \frac{T_{compute} + T_{io}}{T_{compute}} = 1 + \frac{T_{io}}{T_{compute}} \quad (4)$$

The equation (4) means that the upper bound of speedup with CFIO is decided by the proportion of the I/O time and the compute time of the origin model. The greater proportion of I/O in whole climate model, the greater speedup CFIO can achieve.

3.3 Communication Method for I/O Forwarding

A natural communication choice for I/O Forwarding is the asynchronous communication approach. Using this approach, all of the I/O requests are packed into a client buffer; then, forwarding is performed by a separate sending thread during the computing phases. This approach permits I/O forwarding to overlap with computing, which implies that the major overhead of calling an asynchronous CFIO function is memory copy.

However, when performing many experiments, we observed that the asynchronous communication approach brings about network resource competition between the computing phase and the I/O forwarding phase. The communication performance of the computing phase will be influenced by the concurrent I/O forwarding. The impact is small when the climate model only runs over a small number of cores. When the program scales to more cores, the influence will grow more.

Therefore, the synchronous communication approach is a better choice for larger scale computing. Because the intercommunication that is needed by the computing phase will not occur before the completion of I/O forwarding, a network resource conflict between them is avoided. Although the extra overhead of waiting for the completion of I/O forwarding is introduced into the I/O processes, the overall execution time of the climate model is reduced because of the decrease in the impact on the intercommunication. The effects of asynchronous communication and synchronous communication used in CFIO are compared in Section 4.1.

We must note that synchronous communication can lead to buffer exhaustion in I/O processes because of the bursty I/O behavior in climate models. In this case, the client cannot finish I/O forwarding until the server completes parts of the buffered I/O requests, to clear sufficient buffer space. Because the I/O pattern of a climate model is known, the buffer exhaustion can be avoided by launching enough servers, which is controlled by users.

Table 1. Additional Functions of CFIO

Function	Description
cfio_init	Initialization of CFIO
cfio_final	Finalization of CFIO
cfio_proc_type	Indicating whether the process is an I/O process or a compute process
cfio_io_end	Indicating the completion of an I/O phase

3.4 CFIO Interface

CFIO inherits the classical netCDF format to minimize the non-meaningful efforts in terms of code updates and data post-processing for climate models. In netCDF, writing a new dataset contains a sequence of operations, which creates the dataset, defines the dimensions, variables, and attributes, ends define mode, writes variable data, and closes the dataset file. CFIO supports all of the functions that are required to perform the series of operations. These functions can be classified into three categories:

1. **Dataset Functions:** create/close a dataset, set the dataset to define/data mode.
2. **Define Mode Functions:** define dataset dimensions, variables and attributes in define mode.
3. **Data Access Functions:** read/write variable data in data mode.

There are four additional functions that involve initialization and finalization of the library and an operation that concerns I/O forwarding. All of the additional functions are shown in Table 1.

For the requirement for consistency across all compute processes, all CFIO functions are defined as collective I/O operations. When a climate model intends to write a new dataset, all of the computer processes should call the CFIO functions in the same sequence, and the same arguments should be passed into the functions.

Listing 1. A simple example with CFIO

```
!*******************************
! Initialize
!*******************************
ierr = cfio_init(LAT_PROC, LON_PROC, ratio)
```

```
!********************
! Computing and Outputing
!********************
IF (cfio_proc_type() == CFIO_TYPE_CLIENT) THEN
    !********Computing phase************
    ......
    !********I/O phase**********************
    !output data
    ierr = cfio_create(filename, 0, ncid)
    ierr = cfio_def_dim(ncid,"lat",lat,dim(1))
    ierr = cfio_def_dim(ncid,"lon",lon,dim(2))
    ierr = cfio_def_var(ncid,"sst",NF_FLOAT,2,dim,0)
    ierr = cfio_enddef(ncid)
    ierr = cfio_put_vara_real(ncid,0,2,start,count,sst)
    ierr = cfio_close(ncid)
    ierr = cfio_end_io()
END IF

!******************************
! Finalize
!******************************
ierr = cfio_finalize()
!******************************
```

Listing 1 shows a simple example of outputting data with CFIO. This example output Sea Surface Temperature(SST) data with latitude and longitude dimensions . The *cfio_init* function is used to decribe the size of the output array and the number of CFIO server. The function accepts tree arguments : *LAT_PROC*, *LON_PROC* and *ratio*. *LAT_PROC* and *LON_PROC* is used to describe the two-dimensional data decomposition of the horizontal domain among compute processes. *ratio* stands for the proportion of compute processes and I/O processes. If we run the example application with N MPI processes, there will be $N/ratio$ processes acted as I/O processes. The *cfio_proc_type* function is called to indicate the type of the local process. The computer processes should run the computing code, and call the CFIO data access functions to output data. The I/O process will run the CFIO server automatically, and no more other code need to be added. The *cfio_put_vara_real* function is called to output the variable data and the *cfio_end_io* function is called to send a signal that an I/O phase is finished. This signal is used to control the buffer management and the process of sender and receiver. This example shows that the I/O forwarding is automatically implement and the complicated data transfer is transparent to modelers.

4 Experiments

We conducted our experiments on the $Explore100$ cluster at Tsinghua University. Each node of the $Explore100$ cluster has two 2.93GHz Intel Xeon X5670 6-core processors, which share 32 GB memory. The cluster is connected by an Infiniband network, which enables a maximum bandwidth of 40 Gb/s. The file system is Lustre, with 1 Meta-Data Servers (MDS) and 40 Object Storage Targets (OST). The peak writing performance of this file system is 4 GB/s. The node operating system is RedHat Enterprise Linux 5.5 x86_64. All of the applications were compiled with Intel compiler v11.1, and the MPI environment is Intel MPI library v4.0.2.

We will evaluate the performance of CFIO three practical climate models and compare CFIO with PnetCDF in the following section.

4.1 POP Case

The Parallel Ocean Program (POP) is an ocean circulation model that was developed at Los Alamos National Laboratory and was written in Fortran with MPI support. The POP partitions the data arrays equally across all of the compute processes by using a two-dimensional data decomposition of the horizontal domain. In its default I/O approach, data are gathered in one process and then are written by calling a serial netCDF interface. National Center for Atmospheric Research(NCAR) officially adopted POP as the ocean component of the Community Climate System Model(CCSM) and Community Earth System Model (CESM) and added various features to meet the needs of the CCSM and CESM coupled model. The PIO library is also used to improve the I/O performance in CESM. For simplicity, we only use the standalone POP version 2.0.1 in this experiment.

The POP output files comprise restart files, history files and movie files. In this experiment, the POP with 0.1 degree resolution ran for 440 iterations to simulate 2 days. The restart file is generated periodically in days. History and movie files are generated periodically in hours. The variables included in output netCDF files are two dimensional arrays of size $3,600 \times 2,400$, which stand for the horizontal domain, and three dimensional arrays of size $3,600 \times 2,400 \times 40$, in which the third dimension stands for the level of the sea depth. The size of the final output files is 315 GB in total.

We measured the overall POP execution time with CFIO and compared the results with the default I/O approach and the NO-I/O approach in POP. The overall execution time with the NO-I/O approach stands for the maximum performance that can be achieved by complete I/O overlapping with the computing phase. Fig. 5 shows the experimental result. As expected, CFIO outperforms the default I/O approach in POP, with both 32 CFIO servers and 64 CFIO servers. When running with 1,280 clients and 32 CFIO servers, the overall execution time of POP decreases from 3,246 seconds to 471 seconds, which means that we obtained a 6.9x speedup for POP using CFIO. This speedup is very close to the NO-I/O approach. The performance of running with 64 servers is better than that of running with 32 servers. When the client number is 1,280, it is observed that the overall execution time of running with 64 CFIO servers has a reduction of 12%, compared to that of running with 32 CFIO servers. The execution time decreases from 471 seconds to 413 seconds, which means a speedup of 7.9x for origin POP.

We compared the POP execution time for each of the two different communication approaches discussed in Section 3.3. Because there is intercommunication that occurs in the compute phases of POP, the impact on computing from different communication methods is shown in the test result.

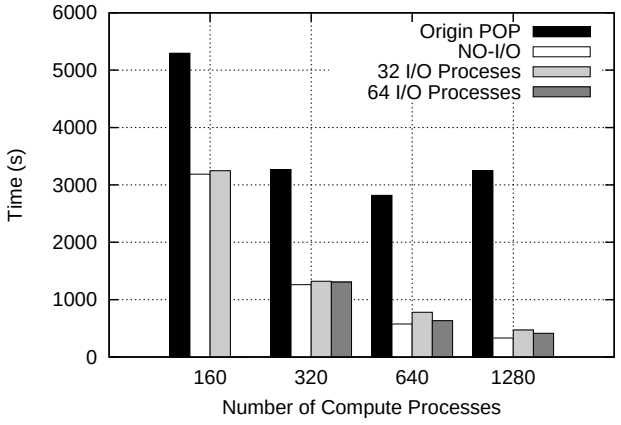

Fig. 5. The overall execution time of POP with different I/O approaches

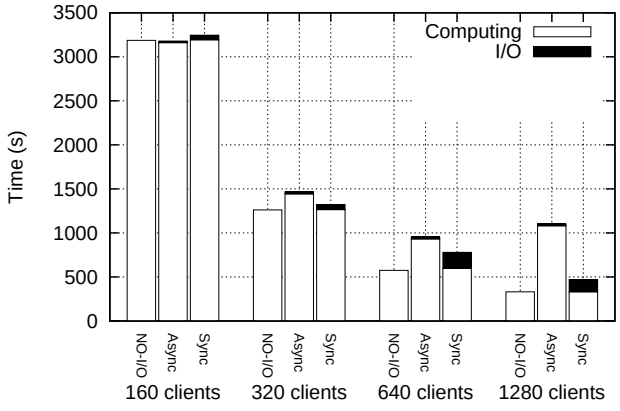

Fig. 6. The execution time for computing and I/O with different communication methods when running with 32 CFIO servers

To obtain an accurate understanding of the impact from I/O forwarding, we measured the execution time for computing and I/O separately in the POP. We also evaluated POP with NO-I/O, for measuring the computing execution time that is not impacted by any I/O behaviors.

Fig. 6 shows the test result of running with 32 CFIO servers. It is observed that the computing time of synchronous communication is always close to that of NO-IO while the asynchronous communication becomes more time-consuming than NO-IO when the POP scales to more cores. When the client number is 160, we observe that the overall runtime of using asynchronous communication is less than that of using synchronous communication. The computing time of asynchronous communication is almost the same with that of NO-IO approach since the impact is small when the POP runs over a small number of cores. When running with 320 and more clients, we observe that the impact on computing time from asynchronous communication becomes more than that from synchronous communication and the performance of synchronous communication becomes better. When running with 1,280 clients, the execution time decreases from 1132 seconds, which includes a computing time of 1101 seconds and an I/O time of 31 seconds, to 471 seconds, which includes a computing time of 331 seconds and an I/O time of 140 seconds. Based on these results, we use synchronous communication as our default communication method.

4.2 CICE Case

The Los Alamos sea ice model(CICE) is a sea ice model which was also developed at developed at Los Alamos National Laboratory. It is the sea ice component model of the CESM. In general, the CICE uses the same horizontal grid resolution with the POP. The CICE partitions the data arrays equally across all of the compute processes by using the same two-dimensional data decomposition of the horizontal domain with POP. CICE uses netCDF as the output file format for history files. The restart files are generated as binary file format. When outputting history data in CICE, data are gathered in one process and then are written by calling a serial netCDF interface.

Because CFIO only supports netCDF format, we only used CFIO to output history files in CICE, and the output of restart files is disabled. we used the CICE version 4.1 with 0.1 degree resolution in this experiment. The CICE ran for 960 iterations to simulate 40 days. History files are generated periodically in days. The variables included in output netCDF files are 2D arrays of size $3,600 \times 2,400$. The output files only have a fixed size of 80 GB in total.

We measured the overall CICE execution time with CFIO and compared the results with the default I/O approach and the NO-I/O approach in CICE. Fig. 5 shows the experimental result. The number of computer processes varies from 160 to 1,280. Although the execution time of CICE with the NO-IO approach decreases with the increment of the number of computer processes, the execution time of CICE with the default I/O approach stops decreasing due to the poor I/O scalability. CFIO outperforms the default I/O approach in CICE, with both 32 CFIO servers and 64 CFIO servers. When running with 1,280 computer processes, the execution time of CICE with 64 CFIO servers is 204 seconds, which is less than the 226 seconds execution time of CFIO with 32 CFIO servers. The scalability of CFIO is confirmed by using more servers. Compared to the origin CICE execution time of 928 seconds, the speedup that we can obtain by using 64 CFIO servers is 4.6 times. The speedup is a little lower than that of the POP. The main reason is the I/O load of CICE in less than that of POP. Since the execution time with the NO-I/O approach, which stands for the computing time of the origin CICE, is 178 seconds, the proportion of the I/O time and the computing time is 4.2. We can derive from the upper bound equation of speedup that the maximum speedup is 5.2 times.

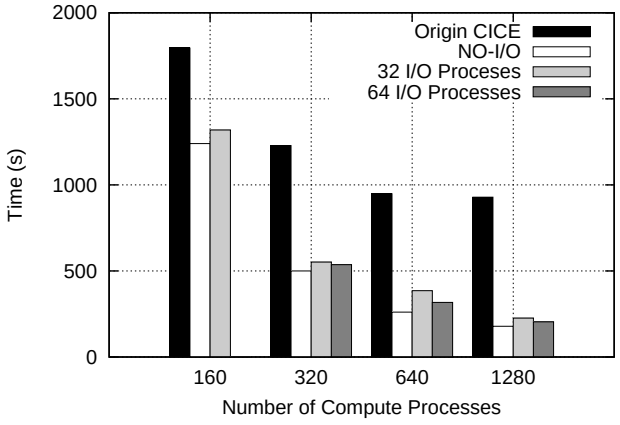

Fig. 7. The overall execution time of CICE with different I/O approaches

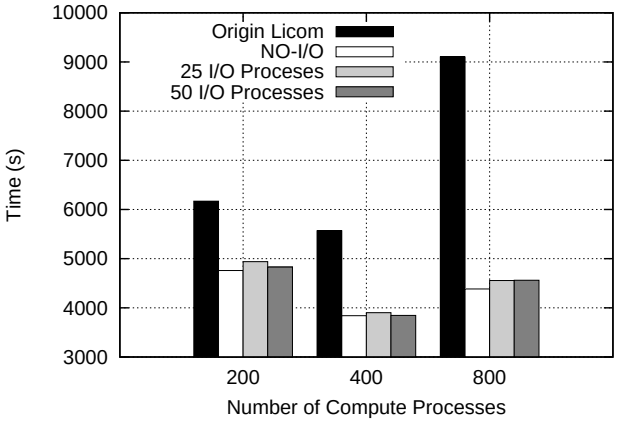

Fig. 8. The overall execution time of Licom with different I/O approaches

4.3 LICOM Case

The LASG/IAP Climate Ocean Model (LICOM) is an ocean model developed by the State Key Laboratory of Numerical Modeling for Atmospheric Sciences and Geophysical Fluid Dynamics in China. It is the sea component model of LASG/IAP Earth System Model FGOALS_g2. LICOM also partitions the data arrays equally across all of the compute processes by using a two-dimensional data decomposition of the horizontal domain. LICOM uses netCDF as the output file format. The output data, including restart files and history files, are gathered in one process and then are written by calling a serial netCDF interface.

In this experiment, we used LICOM version 2 with 0.1 degree to simulate 10 days. Restart files are generated periodically in days. Only one restart file is generated at the end of the program. The variables of output files are two dimensional arrays of size $3,602 \times 1683$, which stand for the horizontal domain, and three dimensional arrays of size $3,602 \times 1,683 \times 55$, in which the third dimension stands for the level of the sea depth. The output files have a fixed size of 144 GB in total.

Fig. 8 shows the test result of the LICOM. In this experiment, the number of computer processes varies from 200 to 800. The scalability of LICOM performs not very well. When scaled to 800 computer processes, the computing time of LICOM begins to decrease. So we did not use more computer processes in this experiment. The LICOM runing with 800 clients and 50 servers has a execution time of 4,561 seconds. Compared to the original execution time of 9,101 seconds, The LICOM obtained a 2.0x speedup by using CFIO. The execution time with the NO-I/O method, which stands for the computing time of the origin LICOM, is 4,383 seconds, the proportion of I/O time and the computing time is 1.07, which means the maximum speedup for the LICOM with CFIO is 2.07.

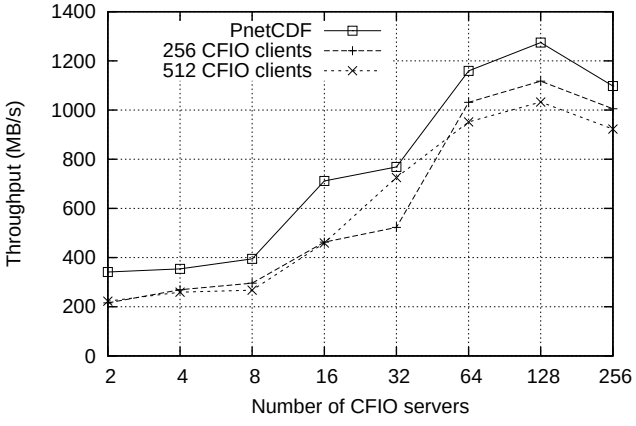

Fig. 9. Throughput comparison of CFIO and PnetCDF

4.4 CFIO v.s. PnetCDF

We performed two MPI test applications to compare the performance of CFIO with PnetCDF in this experiment. The first MPI test application outputs a 32 GB dataset with 500 variables in one huge netCDF file format to evaluate the throughput of CFIO and PnetCDF. The second MPI test application simulates the common I/O pattern of climate models to show the advantage of I/O forwarding technology.

In the first application, every variable is an two dimensional array of $4,096 \times 2,048$ double-precision floating-point number. Data arrays are partitioned equally across all of the client processes, using two-dimensional data decomposition. The size of the output data per client process decreases as the number of client processes increases. We also measured the writing performance with an MPI test application that simulates the I/O pattern of climate models.

Fig. 9 shows the throughput of CFIO as a function of the number of CFIO servers. The throughput of PnetCDF is shown as a contrast. The horizontal axis of PnetCDF stands

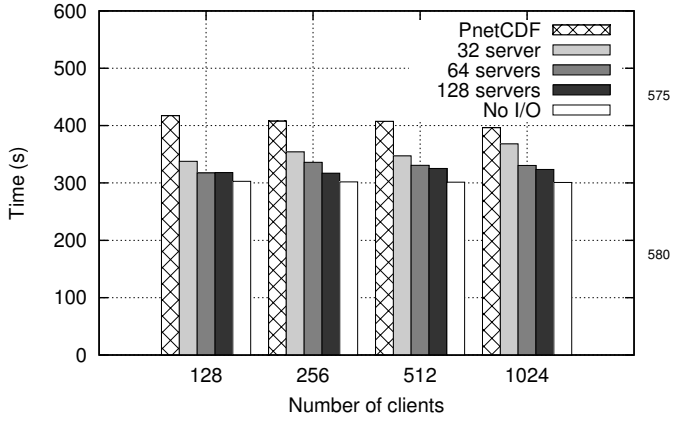

Fig. 10. The overall execution time of the MPI test application with CFIO and PnetCDF

for the number of clients that call PnetCDF functions. Fig. 9 shows that the throughput of CFIO increases with the growth in the number of CFIO servers but then stops increasing when the number of CFIO servers is increased to 128. This result occurs because the maximum throughput is limited by the number of storage devices in the Lustre file system. Then, the writing throughput of CFIO can achieve approximately 1 GB/s when using 128 servers and 512 clients. The same pattern is observed with PnetCDF. The write throughput of PnetCDF can achieve approximately 1.24 GB/s when using 128 clients.

The results show that the throughput of CFIO is approximately 10% less than that of PnetCDF because of the overhead that is associated with I/O forwarding. Although the CFIO performs worse than PnetCDF with respect to the throughput, we will show that the practical performance is better than PnetCDF in the second application.

In the second application, we deploy a similar computing and I/O pattern to the common climate models. There are a total of 40 loops in the application, and each loop will spend 7.5 seconds in performing floating point computation and will produce 3.2 GB of data in each loop. There are no intercommunication operations during the computing phases. For comparison purposes, we evaluate the PnetCDF method and NO-I/O method. NO-I/O means that all I/O operations is disabled in the test application.

Fig. 10 shows the overall execution time of the test application. Without any I/O operations, the whole execution time is 300 seconds. It is pure computing time. When running with 128 clients, the whole execution time for using PnetCDF is 417 seconds, while the corresponding time for using CFIO with 128 servers and 128 clients is only 323 seconds. Fig. 10 also shows that the performance of running with more servers is better. The overhead of I/O when using CFIO is the cost of I/O in the last loop, which cannot be overlapped, plus the cost of I/O forwarding. When the throughput of CFIO grows with an increment in the number of servers, the cost of I/O in the last loop is reduced. The cost of I/O forwarding is also naturally reduced as the number of CFIO servers increases.

The reason for the better performance of CFIO in the first case is that the computing part is not included in the first case. Thus there is no chance to put into full play of I/O forwarding. When we consider more practical computing and I/O pattern in the second case, the I/O time will be hidden by the computing time. This experiment shows that CFIO will perform better performance in real climate models.

5 Related Work

5.1 Low-level parallel I/O libraries

MPI-IO, a parallel low-level I/O standard, is the I/O part of the MPI-2 standard. MPI-IO allows users to collectively specify the I/O requests of a group of processes and to use a stream of bytes in a file as its data model. ROMIO(Thakur et al. (1999)), which is one of the implementations of the MPI-IO standard, uses data sieving and collective I/O techniques to improve the I/O performance. Data sieving is used to process noncontiguous requests from one process. Collective I/O is coordinated access to storage by a group of processes.

5.2 High-level parallel I/O libraries

NetCDF is a scientific I/O library that is widely used in climate models. NetCDF provides a self-describing format for presenting a multidimensional data model. PNetCDF was developed to support parallel I/O for NetCDF. It was built on MPI-IO, to take the advantages of collective I/O optimizations. Starting with version 4, NetCDF allows the interoperability with HDF5, which is another high-level I/O library for storing and accessing multidimensional datasets.

PIO is an application-level parallel I/O library that was developed for the Community Earth System Model. PIO supports several back-end I/O libraries, including MPI-IO, netCDF, and PnetCDF. PIO can redistribute data in all of the processes that participate in I/O and can rearrange data in memory into a more I/O-friendly decomposition. These methods can improve the I/O performance and minimize the memory consumption of PIO efficiently.

ADIOS provides a simple API and an external XML file to configure the data structure and I/O methods. By switching parameters in the XML file, users can choose an optimal I/O method for their application according to the runtime environment. In ADIOS, a novel BP file format is designed to decrease the overhead of maintaining metadata consistency.

5.3 Overlapping I/O and I/O forwarding

Overlapping I/O has been shown as a useful technique to dramatically improve I/O performance. Dickens and Thakur

(1999) implemented split-collective I/O in MPI 2.0 to provide a collective I/O that overlaps with computing by threads. The research found that simply spawning a thread to perform the collective I/O operation in the background is worse than the sequential approach. The best approach is to only perform write to disk in the background, and to perform the copying and intercommunication that is required by the collective I/O in the main thread. More et al. (1997),Tsujita (2004) and Caglar et al. (2003) also used multi-threaded mechanisms to increase the performance of MPI applications(). Patrick et al. (2008) presents a comparative study of different strategies of overlapping I/O, communication and computation. The performance results show an obvious benefit from overlapping I/O with other procedures.

I/O forwarding has been used in Prost et al. (2001),Oldfield et al. (2006),Nisar et al. (2008),Fu et al. (2010),Docan et al. (2010) and May (2001) to reduce the I/O impact on computing. The IBM Blue Gene series of supercomputers(Yu et al. (2006)) uses independent I/O nodes in their system to handle I/O requests, which are generated in computer nodes and forwarded to I/O nodes. Datastager designed by Abbasi et al. (2009) is a data staging services that provide asynchronous data extraction for ADIOS. This service takes the approach that is the mostly closely related to CFIO among the studies that inspire CFIO's design. Datastager uses server-directed I/O to manage asynchronous communication for data transfer. The research about Datastager found that the asynchronous method for data transfer can significantly impact the performance of tightly coupled parallel applications. Datastager implements two schedulers to reduce the impact. The phase-aware scheduler prevents background data transfer in the communication phase, which is predicted by Datastager or marked by the application developers. The rate limiting scheduler manages the number of concurrent requests that are made to compute nodes to control the data transfer rate.

Palmer et al. (2011) proposed a specialized parallel data I/O method for Global Cloud Resolving Model. This method can avoid the creation of very large numbers of files. The output data layout which linearizes the data in a consistent way that is independent of the number of processors used to run the simulation and provides a convenient format for subsequent analyses of the data.

Compared with all of the above work, CFIO provides automatic overlapping of I/O with computing. CFIO uses I/O forwarding to perform overlapping I/O on remote processors so that the overhead for managing multiple threads is avoided. In addition, CFIO provides synchronous APIs that performs I/O overlapping automatically. Modifications to the existing climate modeling code for asynchronous APIs are not necessary.

6 Conclusions

We presented a parallel I/O library, CFIO, which provides automatic overlapping of I/O with computing. The interface of CFIO is based on netCDF and is designed to be easily adopted in existing climate models. The experimental results show that CFIO outperforms PnetCDF when using it in real applications. We indicated that the computing performance can be influenced by I/O forwarding when using an asynchronous communication method. The performance of using different communication methods is compared, and we noted that I/O synchronous forwarding performs better when a program is running on a larger scale.

For our future work, we will conduct more evaluation on different machines that have different file systems. We will also adopt CFIO in more climate models. The MPI communication method that we used for I/O forwarding still requires further optimization. We will attempt to use the RDMA technique in CFIO to obtain better performance.

Acknowledgements. This work is supported in part by a grant from the National Grand Fundamental Research 973 Program of China (No. 2014CB347800) and the National High Technology Development Program of China (2011AA01A203).

References

Abbasi, H., Wolf, M., Eisenhauer, G., Klasky, S., Schwan, K., and Zheng, F.: DataStager: scalable data staging services for petascale applications, in: Proceedings of the 18th ACM international symposium on High performance distributed computing, pp. 39–48, New York, NY, USA, 2009.

Caglar, S., Benson, G., Huang, Q., and Chu, C.: USFMPI: A multithreaded implementation of MPI for Linux clusters, in: Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems, 2003.

Corbetty, P., Feitelsony, D., Fineberg, S., Hsuy, Y., Nitzberg, B., Prosty, J., Sniry, M., Traversat, B., and Wong, P.: Overview of the MPI-IO parallel I/O interface, Input/output in parallel and distributed computer systems, 362, 127–146, 1996.

Dennis, J., Edwards, J., Loy, R., Jacob, R., Mirin, A., Craig, A., and Vertenstein, M.: An application-level parallel I/O library for Earth system models, International Journal of High Performance Computing Applications, 26, 43–53, 2012.

Dickens, P. and Thakur, R.: Improving collective I/O performance using threads, in: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing, pp. 38–45, Washington, DC, USA, 1999.

Docan, C., Parashar, M., and Klasky, S.: Enabling high-speed asynchronous data extraction and transfer using DART, Concurrency and Computation: Practice and Experience, 22, 1181–1204, 2010.

Fu, J., Liu, N., Sahni, O., Jansen, K. E., Shephard, M. S., and Carothers, C. D.: Scalable parallel I/O alternatives for massively parallel partitioned solver systems, in: Proceedings of the Workshop on Large-Scale Parallel Processing in conjunction with the

IEEE International Parallel and Distributed Processing Symposium, pp. 1–8, Atlanta, Georgia, USA, 2010.

Hunke, E. and Lipscomb, W.: CICE: the Los Alamos Sea Ice Model documentation and software user's manual version 4.1, Rep. LA-CC-06, 12, 2010.

Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M.: Parallel netCDF: A High-Performance Scientific I/O Interface, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, pp. 39–39, New York, NY, USA, 2003.

Lofstead, J., Zheng, F., Klasky, S., and Schwan, K.: Input/output apis and data organization for high performance scientific computing, in: Proceedings of Petascale Data Storage Workshop 2008 at Supercomputing 2008, pp. 1–6, Austin, Texas, USA, 2008.

Lofstead, J., Zheng, F., Klasky, S., and Schwan, K.: Adaptable, metadata rich IO methods for portable high performance IO, in: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, pp. 1–10, Washington, DC, USA, 2009.

May, J. M.: Parallel I/O for high performance computing, Morgan Kaufmann, 2001.

More, S., Choudhary, A., Foster, I., and Xu, M.: MTIO - A Multi-Threaded Parallel I/O System, in: Proceedings of the 11th International Symposium on Parallel Processing, pp. 368–373, Washington, DC, USA, 1997.

Nisar, A., Liao, W.-k., and Choudhary, A.: Scaling parallel I/O performance through I/O delegate and caching system, in: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pp. 1–12, Piscataway, NJ, USA, 2008.

Oldfield, R., Ward, L., Riesen, R., Maccabe, A., Widener, P., and Kordenbrock, T.: Lightweight I/O for scientific applications, in: Proceedings of the IEEE International Conference on Cluster Computing, pp. 1–11, Barcelona, Spain, 2006.

Palmer, B., Koontz, A., Schuchardt, K., Heikes, R., and Randall, D.: Efficient data IO for a parallel global cloud resolving model, Environmental Modelling & Software, 26, 1725–1735, 2011.

Patrick, C., Son, S., and Kandemir, M.: Comparative evaluation of overlap strategies with study of I/O overlap in MPI-IO, ACM SIGOPS Operating Systems Review, 42, 43–49, 2008.

Prost, J.-P., Treumann, R., Hedges, R., Jia, B., and Koniges, A.: MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS, in: Proceedings of the 2001 ACM/IEEE conference on Supercomputing, pp. 58–58, New York, NY, USA, 2001.

Rew, R. and Davis, G.: NetCDF: an interface for scientific data access, IEEE Computer Graphics and Applications, 10, 76–82, 1990.

Smith, R., Jones, P., Briegleb, B., Bryan, F., Danabasoglu, G., Dennis, J., Dukowicz, J., Eden, C., Fox-Kemper, B., Gent, P., Hecht, M., et al.: The Parallel Ocean Program (POP) Reference Manual: Ocean Component of the Community Climate System Model (CCSM) and Community Earth System Model (CESM), Tech. Rep. LAUR-10-01853, Los Alamos National Laboratory, 2010.

Thakur, R., Gropp, W., and Lusk, E.: Data Sieving and Collective I/O in ROMIO, in: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation, pp. 182–189, Washington, DC, USA, 1999.

Tsujita, Y.: Effective nonblocking MPI-I/O in remote I/O operations using a multithreaded mechanism, in: Proceedings of the Second international conference on Parallel and Distributed Processing and Applications, pp. 34–43, Berlin, Heidelberg, 2004.

Yu, H., Sahoo, R., Howson, C., Almasi, G., Castanos, J., Gupta, M., Moreira, J., Parker, J., Engelsiepen, T., Ross, R., et al.: High performance file I/O for the Blue Gene/L supercomputer, in: International Symposium on High-Performance Computer Architecture, pp. 187–196, Austin, Texas, USA, 2006.

Yu, Y., Liu, H., and Lin, P.: A quasi-global 1/10° eddy-resolving ocean general circulation model and its preliminary results, Chinese Science Bulletin, 57, 3908–3916, 2012.