Parallel Computing 104-105 (2021) 102786

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Improving the I/O of large geophysical models using PnetCDF and BeeGFS☆

Jared Brzenski [a,*], Christopher Paolini [b,a], Jose E. Castillo [a]

[a] Computational Science Research Center, San Diego State University, USA
[b] Electrical and Computer Engineering, San Diego State University, USA

ARTICLE INFO

Keywords:
HPC
BeeGFS
PnetCDF
Parallel
Computing
Modeling
IO
IOPS

ABSTRACT

Large scale geophysical modeling uses high performance computing systems to expedite the solutions of very large, complex systems. High disk latencies, low IOPS, and low read/write data transfer rates are relegating many numerical simulations to I/O bound jobs, where the run time is bound not by CPU rate, but by I/O rate. In this paper we seek to improve the I/O of two geophysical modeling applications and take full advantage of the parallel nature of the programs, as well as the file management system for the large output files. Parallelizing output for these programs is achieved using PnetCDF, a parallel implementation of the netCDF format, and BeeGFS, an open-source parallel file system. Using these solutions, we have significantly decreased the amount of time spent saving data to disk, and give analysis of the features used in relation to PnetCDF with BeeGFS I/O optimization.

1. Introduction

Large spatial and temporal scale geophysical simulations require very large data sets which must be read from and written to. Additionally, these large files may be striped across multiple hard drives requiring the use of a parallel file system (PFS). This I/O can become a significant bottleneck when running a simulation across multiple compute nodes in a high performance computing (HPC) environment. These are all currently areas of interest for us at SDSU and our geophysical models. We have two models, the General Curvilinear Coastal Ocean Model (GCCOM) and *Subflow*, which are large scale simulations designed to run across hundreds of compute nodes, yet process I/O serially using the netCDF API interfaces. As domain sizes increase to tens of millions of data points, the I/O becomes a bottleneck for scalability of both models. The netCDF format is the standard file format used by many organizations in the climate community for storing large data sets [1,2]. In its native format, netCDF does not support parallel reading and writing of its files, but has a parallel implementation *PnetCDF* that does [3].

While PnetCDF allows for almost unlimited file size capability, this leads to a typical situation in HPC of large files not being stored on a single physical drive. When implementing a new large parallel I/O scheme, the file system management must also be considered. A PFS acts as an intermediary between multiple physical drives and an overlying operating system. Lustre [4], GPFS [5], DeltaFS [6], and BeeGFS [7] are all variations of a PFS manager, whereby large files are broken into pieces and kept track of across multiple hard drives. A separate client manages this file tracking, allowing users to treat the multiple hard drives as one storage unit. Many of these PFS systems are fee based, which can be a limiting factor for educational organizations. BeeGFS is an open source (free) PFS that is touted as easy to deploy and maintain on existing HPC systems, which makes it the most cost-effective strategy for education institutions. There are a number of configuration options for each PFS, usually specific to the system running it and the general data being written.

This paper describes the implementation, analysis, and performance tuning specific to PnetCDF with BeeGFS and provides a framework for others working with similar programs and I/O conditions. While both PnetCDF and BeeGFS have been analyzed individually, we hope that studying them together will add insight into what realistic I/O patterns manifest from both systems and what options can be used to optimize I/O.

The paper is organized as follows. We start with a brief overview of PnetCDF and the BeeGFS standards, as well as the models Subflow and GCCOM in Section 2. Specifics about file I/O requirements, processing, and computing overhead will give a context to the results we find in Sections 4 and 5 . We introduce the testing system and timing protocols in Section 3, as well as the HPC benchmarks used to get the baseline performance and how we tracked improvements. BeeGFS file structure

☆ This work has been funded by NSF Office of Advanced Cyberinfrastructure (OAC) CC* Storage, United States Grant 1659169, *Implementation of a Distributed, Shareable, and Parallel Storage Resource at San Diego State University to Facilitate High-Performance Computing for Climate Science*.

* Corresponding author.
E-mail address: jbrzenski@sdsu.edu (J. Brzenski).

https://doi.org/10.1016/j.parco.2021.102786
Received 12 November 2020; Received in revised form 15 April 2021; Accepted 20 April 2021
Available online 28 April 2021

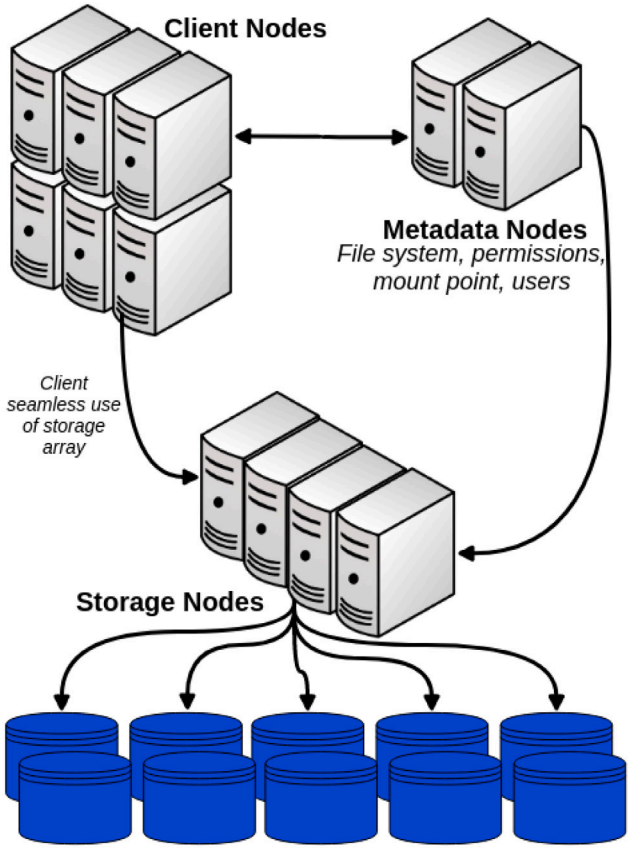

Fig. 1. BeeGFS system structure. Client nodes operate as if multiple drives are mounted as one. Bookkeeping is done by the BeeGFS metadata service to track file name, locations, permissions, and stripe and chunk settings.

and the impact on the benchmarks is also described here. Specifics about implementation as well as results for the individual models are given in Sections 4 and 5, where we discuss what specific PnetCDF calls were used and what MPI tuning was used to minimize the run time and maximize I/O for our specific models. Finally, we summarize the findings and discuss future work in Section 6.

2. Background

2.1. BeeGFS

BeeGFS is an open source parallel file system manager designed for flexibility and ease of administration [7]. It is designed to be used on any style of computing system, with little change to the underlying operating system configuration. BeeGFS is divided into four systems: management, metadata, client, and storage. The management service controls, and is a watchdog for, the other services. The metadata system retains all information about the directory and file structure. A single metadata service can be in charge of the entire file system, or multiple metadata services can be in charge of smaller portions of the file system. The storage system controls the writing and reading of files, and maintains the striping and chunking of the data, to accurately track files across multiple physical locations and multiple RAID structures. The client service manages the interface between BeeGFS and a Linux OS, to make reading and writing files no different than if the native OS were operating alone. A visual diagram of this relationship is given in Fig. 1.

While BeeGFS is currently deployed in European super computing centers and globally on private clusters, not much research is done specific to the BeeGFS file system. Optimization of stripe and chunk size for BeeGFS has been explored for small I/O packet size [8], but has not been formally explored for large file size simulations, especially coupled with another parallel I/O API. Research on MPI and BeeGFS has shown that using MPI decreased transmission time for large files [9]. File cacheing and TCP optimizations have been looked at [10,11], as well as the basic I/O speed of BeeGFS compared to other PFS [12]. However, specifics about the PFS settings and MPI parameters were not studied. Other PFS have been extensively tested [13–15], and some have MPI options specific to them [16]. The effects of MPI tuning options specific to BeeGFS has not been explored and is one of the main goals of this paper.

2.2. PnetCDF

Parallel netCDF is an I/O library for reading and writing netCDF files using parallel I/O. netCDF is a self describing data format, where variables stored in an array like structure have information about each variable in a header, with the multidimensional arrays following [1]. The header file describes the variables, how many dimensions they have, the units of measure, and the dimension of the variables. PnetCDF was chosen as the most likely candidate for parallel I/O because PnetCDF provides a similar interface to the serial netCDF API, but harnesses MPI-IO for concurrent reading and writing of data from multiple nodes [3]. PnetCDF has been used to parallelize output of the largest geophysical running models including the Global Cloud Resolving Model (GCRM) [17], WRF [18] and NCAR [19]. The API gives the user many different ways to output the data in parallel, giving it flexibility to be used regardless of the parallelized structure of the host program. PnetCDF allows for buffering of the reads and writes, allowing the underlying MPI structure to decide the best time to read or put data to disk. Some of the functions allow for blocking read and write operations, as well as optimized accessing of the same variable multiple times. PnetCDF can also simplify multiple I/O requests for a single variable by either queuing the entire variable, a sub-array, or a sub-selection of array indices. PnetCDF can be used in a purely serial mode, allowing one node to output data while the other nodes idle until I/O has completed. Exabyte file sizes, which are becoming more prevalent, required a new netCDF file format, CDF-5, which uses 64-bit indices and allows data structures that support greater than 2^{32} indices. netCDF provides the ability to use parallel file access with the netCDF-4 framework. This framework eases the transition to parallel file access with minor changes to netCDF commands. Building netCDF-4 includes the PnetCDF library and utilize a large number of the PnetCDF features. While possibly easier to implement, netCDF-4 does not include some features of PnetCDF, like the `flexible`, `nonblocking`, `vard` and `varn` APIs. It was decided to use PnetCDF for the possible utilization of all features and the trivial amount of code rewriting essentially matched that for a similar implementation using netCDF-4.

Because PnetCDF is built upon MPI_IO libraries, it allows hints to be passed to the underlying MPI_IO structures. These can be PnetCDF specific hints, or the more general ROMIO hints [16]. Hints for PFS striping, chunk sizes, and cache sizes can be passed, as well as PFS specific hints for Lustre and GPFS, and some of these MPI_IO hints can significantly change the I/O response of a program [16]. BeeGFS utilizes no ROMIO specific hints, so testing must be done to find the optimum hints for PnetCDF usage.

2.3. Subflow

At San Diego State University, we have developed a distributed parallel application named Subflow [20,21] to model subsurface CO_2 and hard, alkaline waste water injection into porous sandstone and shale reservoir systems. Subflow models the physical, mineralogical, and geochemical effects associated with CO_2 sequestration. As CO_2 is injected into deep geologic formations, the chemical and physical properties of the underlying rock are changed. As the processes can occur

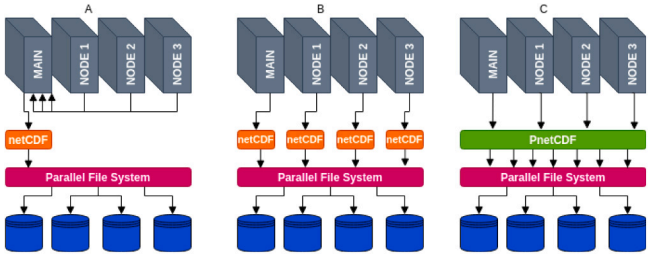

Fig. 2. Example of I/O for a work cluster. Fig. 2A: All nodes return values to the main node for writing out. Fig. 2B: All nodes each write their own data out to separate files, which are collected later. Fig. 2C: All nodes write their data to one file, using PnetCDF as the broker between the workers and the file system.

over many years, modeling of this phenomenon allows experimentation and analysis that are otherwise too time consuming and costly to occur in the field. Subflow's geophysical model [22] couples porosity, fluid pressure, fluid flux and poroelastic effects, together with mineral and solute reactions using a reactive transport and water–rock interaction model [23] based on elemental mass conservation.

Pressure, fluid flux, and rock displacements are calculated using constitutive relations from equations for fluid density, porosity and stress, then solving for a quasi-equilibrium state for each time step [22]. Rock stress and strain is determined by poroelastic relations. Chemical and physical reactions are modeled by applying a water–rock interaction and reactive-transport formalism [23], which provides elemental mass conservation equations. Fluid flux and conservation are solved using a finite element method (FEM) on a structured hexahedral grid. Rock displacements are solved using a standard continuous Galerkin FEM [24]. The FEM equations are solved using the Deal.ii finite element software library [25]. The resulting matrix equation is then solved using the PETSc [26] linear system solvers. This computation is done in parallel, and the resulting solution is passed back to the primary process.

The reactive transport model is solved using a finite difference based numerical formalism [23]. The data structures that store information on fluid and solid properties of the reservoir are initialized to match the quadrilateral grid used in our FEM formulation. Porosity, pressure, and fluid flux values in the reactive transport model are coupled with the results of the reservoir flow and poromechanical models. This is solved only by the primary process, and the results are distributed back to the parallel processes for the next time step calculation. All I/O in Subflow is serial (Fig. 2A). The primary process reads the initial netCDF file, and distributes it to the parallel processes. This is the only significant read operation. At each time step, data is aggregated by the serial chemistry solver, then the chemistry solution is redistributed to the parallel physical solvers. If this coincides with a writing time step, the primary process will write the data to the netCDF file, before continuing with the chemistry solution distribution. Currently there is an effort to parallelize the chemistry solver of Subflow. As such, any parallelization of output will necessarily be limited by the serial chemistry solver.

2.4. GCCOM model

The General Curvilinear Coastal Ocean Model (GCCOM) is a large eddy simulation Navier–Stokes solver that is fully curvilinear in a three dimensional coordinate system. GCCOM uses a Boussinesq approximation to find the computational non-hydrostatic pressure solving the 3D full Navier–Stokes equations. The run time discretization uses a Runge–Kutta 3rd order time scheme (RK3). GCCOM has been expanded over many years to resolve gravity waves [27], internal bores [28,29], and internal waves [28,30], and is currently using the Portable Extension Toolkit for Scientific Computing (PETSc) [31] for work distribution [32]. To resolve finer features, GCCOM has also been coupled with ROMS [28] to resolve turbulence and SWASH [33] to resolve surface features.

A driving force behind this project was the amount of time spent writing large data sets in GCCOM. Currently for GCCOM all I/O is serial, with the primary process reading and writing all data to a hard drive (see Fig. 2A). After the initial reading of the input conditions, all of the following I/O consists of writing data to disk, with reads only to confirm the write occurred. As problem size increases, time spent in I/O becomes the main use of computational resources. Not only did this increase as the size of the problem grew, but also as the number of compute nodes increased. For large simulations across multiple nodes, MPI processes and writing out data uses 90% of the computational time [32]. This is particularly frustrating because the code is fully parallelized with respect to solving the NS equations, so using parallel I/O is the next natural step for this model.

3. I/O benchmarking

Timings and analysis was done on the *mixcoatl.sdsu.edu* cluster in the College of Engineering at San Diego State University. It is a Linux based cluster with the following features:

- 256 Intel Xeon E5-2680 Processors
- 16 Nodes of 16 processors and 64GiB RAM per node
- 40 Gb/s Mellanox Infiniband MT4099 interconnects
- BeeGFS PFS, 2 manager, 2 metadata, 4 storage nodes
- 312 8TB Hitachi 7.2 K HDDs on BeeGFS storage nodes
- 12 1TB Intel 850 Pro SSDs on BeeGFS metadata nodes

Benchmarking the cluster gives a baseline for comparison to the write speeds of the geophysical programs. There are currently many standardized tests to benchmark I/O on HPC clusters [34] including IOZone [35], IOR [36], FLASH-IO [37] and WRFIO [38]. IOR was chosen for the ability to control data chunk size that could either match or mismatch with the PFS chunk size. This ability to match the size of the data sent with the chunking size of the PFS allows the I/O speed to approach its theoretical maximum. Knowing also that not all data is perfectly chunked, a more realistic benchmark is also used. FLASH-IO uses the PnetCDF based I/O routines of the FLASH stellar model program [39] to give a realistic speed for I/O with respect to the PnetCDF format [40].

Initial benchmarking with IOR was run on a variety of folders with different chunking and striping settings from 64 KiB to 2 MiB. IOR was setup to send a typical size file of 512 GiB from one node to the BeeGFS cluster using a transmission size that varied over 32, 64, 128, 256, and 512 KiB (Fig. 3). Previous work has shown that there is an optimum number of segments to choose to maximize I/O [41], but for simplicity the number of segments was set to one. The file size was chosen such that the entire file could not be stored across the write nodes total memory to prevent artificial speedup from cacheing. Write speeds are of most interest to us due to the nature of Subflow and GCCOM, so write speeds were prioritized over read speeds. 256 KiB transfer sizes has the fastest overall speeds at 2662 MiB/s, and the 64 KiB chunk size had better overall performance regardless of the transmission size of the packet sent from IOR. The 64 KiB transfer size was chosen even though it was slower (2658 MiB/s), it performed better than the 256 KiB transfers sizes across the different chunking sizes. The read and write speeds for sending 64 KiB packets to the different striping patterns is shown in Fig. 4.

An important part of parallel I/O are the MPI subroutines that are used. There are hints, based on the ROMIO_HINTS package that give extra information to the MPI calls. These can have a significant effect on I/O speeds. PFS such as Lustre and GPFS have specific hints optimized for use on those types of file systems [13,14]. Using the optimal chunking and striping patterns found from the IOR benchmark, different hints were tested to see if I/O speeds using BeeGFS could improve for the FLASH-IO benchmark as well as the geophysical models.

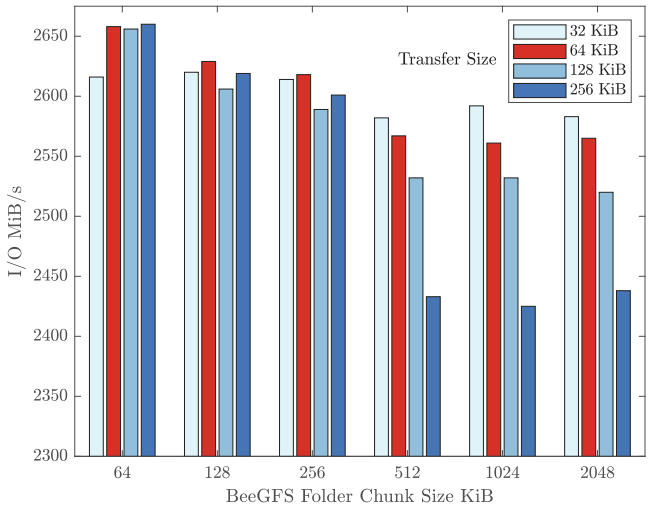

Fig. 3. Max write speeds for IOR with transfer sizes of 32, 64, 128, and 256KiB. The highest write speed is for sending 256 KiB packets to 64KiB chunked BeeGFS folder at 2662 MiB/s. Transfer sizes of 512, 1024 and 2048 KiB were tested as well, but fell far below the values here, mostly around 2000 MiB/s. The 64 KiB transfer size (2658 MiB/s) was chosen because it had better performance versus 256 KiB across different chunking sizes.

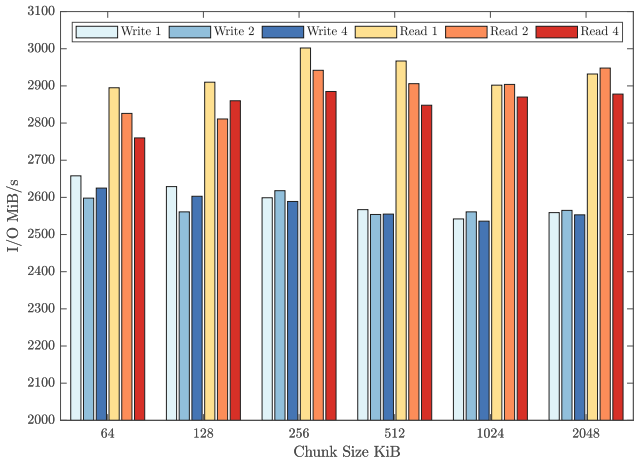

Fig. 4. IOR 64 KiB transmission size with speeds across different BeeGFS chunking values. I/O write speeds are given for reading and writing to 1, 2, or 4 storage targets.

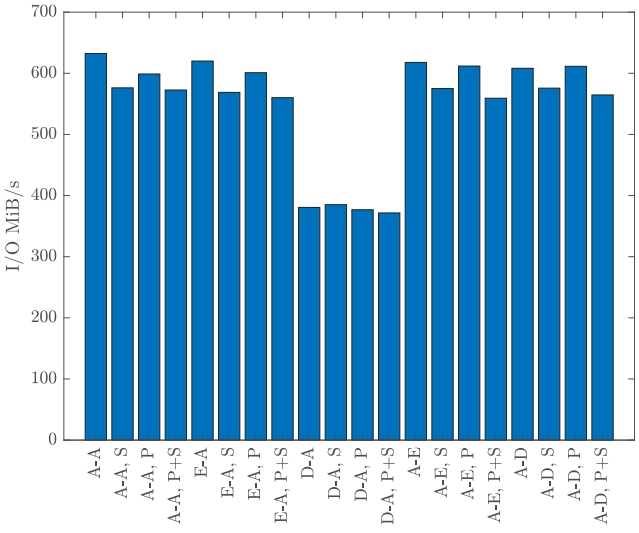

Fig. 5. Results for FLASH-IO run on 80 cores (16 nodes) with different ROMIO and PnetCDF hints used. See Table 1 for explanation of the code. Default values (no hints) is the leftmost column with the maximum write speed of 632 MiB/s. Disabling the `romio_cb_write` reduces write speed by about 1/3. There does not appear to be a great influence of giving PFS hints or PnetCDF hints to the program, but this could change with GCCOM or Subflows PnetCDF implementation.

The FLASH-IO benchmark was run on 10 nodes of 8 cores each, with a variety of ROMIO and PnetCDF hints given as command line arguments (Table 1). The block size was set to 16 × 16 × 16. FLASH-IO will create approximately 80 blocks per process, slightly varying the number to replicate real world data. A full explanation is found in [42]. Total I/O was 9661 MiB. The tests were run 5 times for each of the MPI hint combinations. Maximum throughput was significantly lower than the IOR benchmark, approximately 675MiB/s, which reflects the cost of writing non optimal data sets. Shown in Fig. 5 are the results of varying ROMIO hints across the same benchmark test. The most impacting ROMIO writing hints are `romio_cb_write`, which allows for collective buffering of the write data, and `romio_ds_write`, which allows for data sieving, a technique by which MPI can more easily handle disparate data [16]. These hints have three settings, automatic, enabled and disabled, represented by (A), (E), and (D) in Table 1. The first letter indicates the setting for `romio_cb_write` and the second letter is the setting for `romio_ds_write`. Reading settings were not changed due to the write heavy nature of the models tested. Striping settings can also be passed, setting `striping_unit` equal to the BeeGFS chunk size and `striping_factor` equal to the number of storage targets. Because BeeGFS does not have its own standard of interfacing with ROMIO, an additional hint `cb_nodes` was also used, which indicates the number of storage targets to use. Setting all three of these is indicated in figures with the (S). PnetCDF has hints of its own, in regards to how the variables are offset in the .nc file, and how big of a container should be used for transporting the data for the variables. Aligning these with the chunking size of BeeGFS could increase the I/O bandwidth. We set `nc_var_align_size` equal to the chunk size, and set `nc_header_align` to 512. These hints were passed together and are indicated with a (P) in figures.

The scale of the modeled system and its memory requirements is the main limiting factor to our model specific tests. Subflow and GCCOM are both very memory intensive models, with very large matrices being held in memory. Subflow tracks 73 different chemical and physical variables, while GCCOM utilizes over 100 matrices to solve its system of equations. For Subflow, a fully occupied node of 16 processes could run a maximum problem size of 1.6 million points, while GCCOM could run a problem size of 7.7 million points. While not optimal, these are sufficient to show speedup, PnetCDF implementation, and optimization of MPI and BeeGFS such that large scale tests can be run later on larger clusters.

Table 1
Hint code description for writing to a file location with 64 KiB chunks and four storage targets. Hint keys for collective buffering, data sieving, PnetCDF alignments, and folder settings of BeeGFS PFS.

ROMIO MPI_IO hint	Value(s) used	Key
romio_cb_write	(A)utomatic (E)nable (D)isable	**(A)**-A, P+S **(E)**-A, P+S **(D)**-A, P+S
romio_ds_write	(A)utomatic (E)nable (D)isable	A-**(A)**, P+S A-**(E)**, P+S A-**(D)**, P+S
nc_var_align_size nc_header_align_size	65536 512	A-A, **(P)**+S
cb_nodes striping_unit striping_factor	4 65536 4	A-A, P+**(S)**

4. Subflow implementation and results

Subflow utilizes different solvers for chemistry and physical processes. The physical solver is done in parallel, the chemistry solver is serial. Therefore, only the physical solution data write could be parallelized. An additional set of functions were written in C and the Subflow executable was compiled with both netCDF and PnetCDF libraries. Utilizing a command line switch -parrallel_out, the parallel output functions are called by the physical solvers to output data using the PnetCDF API. The parallel output subroutine is essentially a copy of the serial I/O routine, with some optional PnetCDF descriptions for in dependent and collective I/O. Code changes mostly came down to changing `ncf90_put_var` to `ncf90mpi_put_var` or equivalent. The use of a finite element solver for the physical parameters made discretizing the domain different between each step of the computation. Just because one core solved for a point on the domain it does not mean the core would solve for the same point in the next iteration. This location information is stored as an array in Subflow, and three `for` loops read it into the netCDF I/O routine. The parallel I/O subroutine uses the PnetCDF `ncmpi_put_varn` function. It is a single call, whereby you give the function an array of indices that corresponds to the physical location of the indices of the data you are sending. `ncmpi_put_varn` takes the location array from Subflow as-is, and uses it as the location data in the netCDF file for the data. And while this is a single call, this is equivalent to many individual single write calls in PnetCDF and saves iterating through those three `for` loops the serial I/O subroutine had to do.

The chemistry solver remained a serial solver, as it requires physics information from all worker nodes. This posed the most significant bottleneck in terms of run time and I/O optimization and parallelizing the chemistry solver is currently the subject of ongoing work. Because of the robustness of the PnetCDF API, the chemistry solver was still able to output in serial without the need for extensive code rewriting. This was done by putting PnetCDF in independent mode, and letting the main process write out the chemistry solutions, which could occur independently of the parallel writes. This allowed the chemistry solver to begin before the parallel data had been written to disk.

Subflow was run using a $100 \times 100 \times 16$ domain size running for one week simulated time. CO_2 was injected for the first day and then allowed to diffuse the remaining time. Timing of the I/O routines was done using calls to `MPI_Wtime` around each PnetCDF call, and summing and retaining this value at the end of each iteration. Timing and speedup are shown in Fig. 6 for processes between 2 and 64.

Comparing a pure serial I/O with the PnetCDF implementation, the time saved, while not apparent in Fig. 6, is effectively the difference between serial and parallel time savings in Fig. 7.

The plateauing effect of I/O time is due to the serial I/O, and the same number of MPI calls being made by the main process for the chemistry solver. MPI tuning identified two functions that could be optimized, `MPI_Bcast` and `MPI_Barrier`, yet neither yielded any appreciably different results from the non-tuned program. The write speed has a maximum of 332 MB/s, as is likely due to the problem size, not limitations of BeeGFS. Subflow read speed scales well as more processes are added. This is reflected by the structure of the function that reads in data. This function could make full use of the parallel structure of Subflow. The write speeds for Subflow are essentially unchanged above 16 processors, mostly due to the large amount of data that must be written out in serial by the chemistry solver. The output of the model shows no apparent difference to the totally serial model, Fig. 8, but small differences in the initial steps of the simulation propagate as the model increases.

Fig. 9 shows these differences correspond to the domain discretization of the finite element solver. The serial Subflow relied on CDF-1 format files, which are 32 bit based. The newer parallel output leverages the CDF-5 format file, which is 64 bit, and required using some higher precision variables in Subflow. These error could be the result

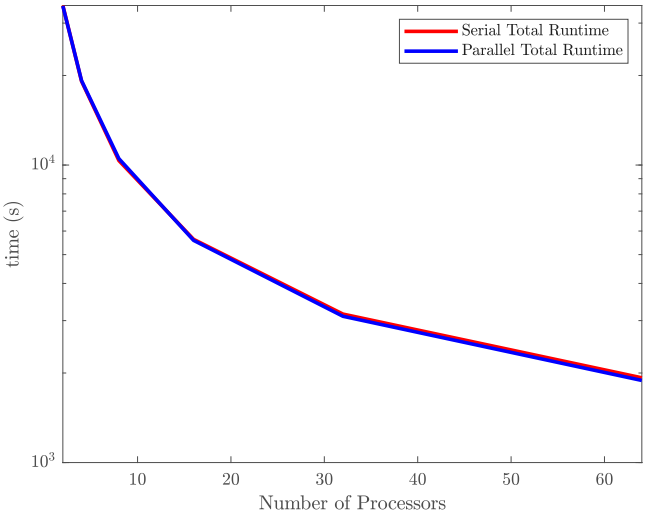

Fig. 6. Subflow timings showing the decrease in time as number of processors is increased. The decrease in I/O time is not apparent at this scale, but resolves in Fig. 7.

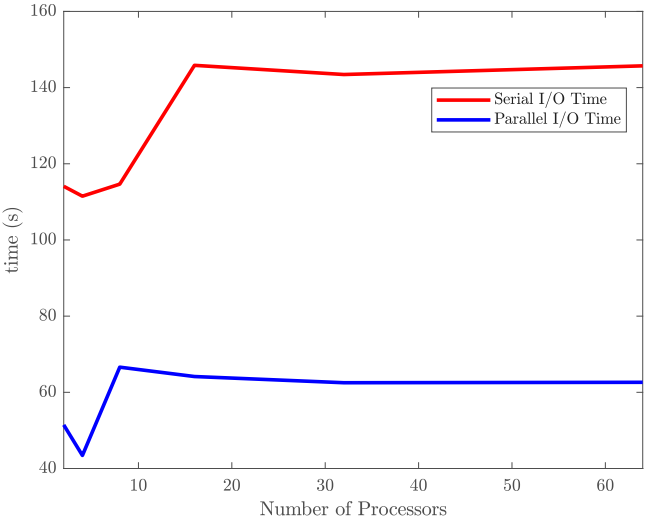

Fig. 7. Subflow I/O timings. A significant decrease is consistently seen. Many chemistry variables are solved in serial, shown by the consistent times across increasing processors.

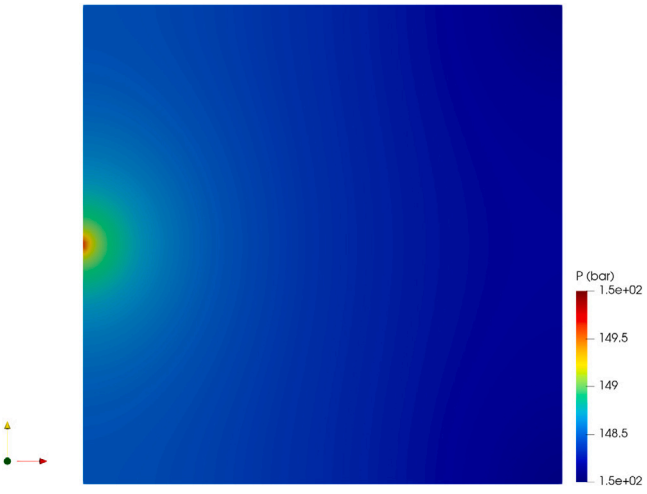

Fig. 8. Output for Subflow, showing pressure over the domain after 2.33 days. The wellhead is located on the left hand side, as the high source of pressure.

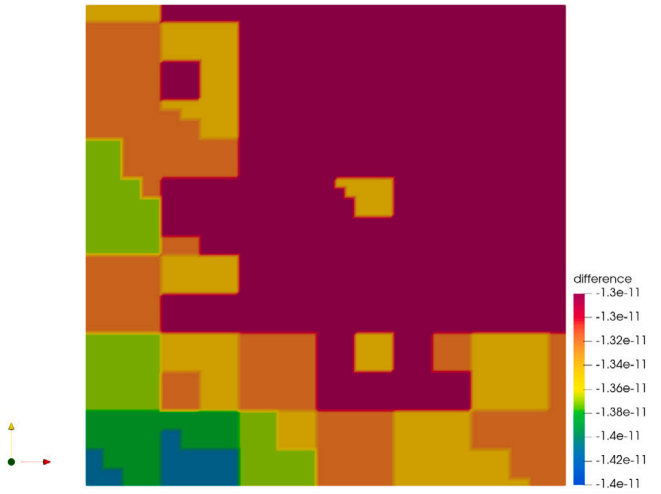

Fig. 9. Difference between the parallel and serial versions of Subflow. The differences correspond to the domain discretization and which compute process has been assigned to them from the finite element solver. The differences shown are on the order of 10^{-11}. The small differences are possibly representative of recasting to higher precision variables in Subflow.

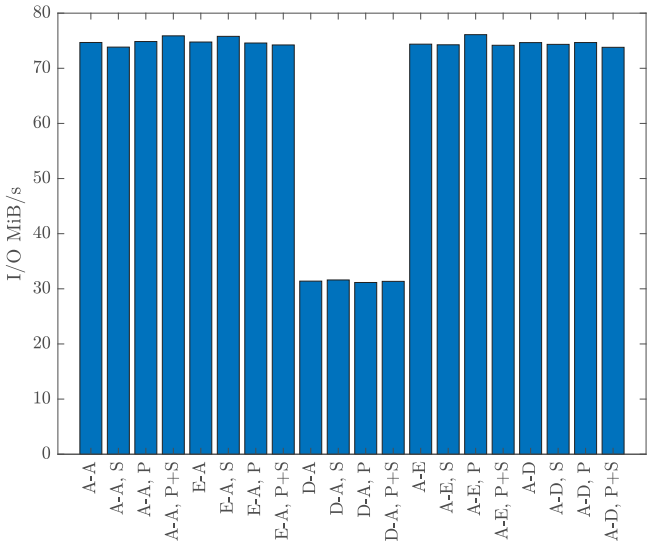

Fig. 10. Subflow I/O write speeds for different ROMIO MPI_IO hints. Default values (no hints) is the leftmost column. Differences are negligible unless data sieving is disabled, which leads to the four very low write speeds. Max I/O rate is 79.7 MiB/s for collective buffering set to automatic, data sieving enabled, and PnetCDF hints used (A-E, P). See Table 1 for full decoding.

of recasting those variables. Using the CDF-5 format with the original serial Subflow netCDF for comparison was not possible.

Varying the MPI hints given showed no appreciable difference in optimum writing speeds. If write caching was disabled, there was a significant reduction in writing speeds. The difference between sending PnetCDF hints and striping hints had no appreciable effect over the standard I/O speeds when MPI used the automatic settings for `romio_ds_write` and `romio_cb_write` (see Fig. 10).

5. GCCOM implementation and results

In GCCOM, I/O routines are run on all processes, with a switch in place for the main node to read or write out. In most cases, parallelizing the I/O entailed changing a few letters in a command. For example, inquiries to existing netCDF variables changed from

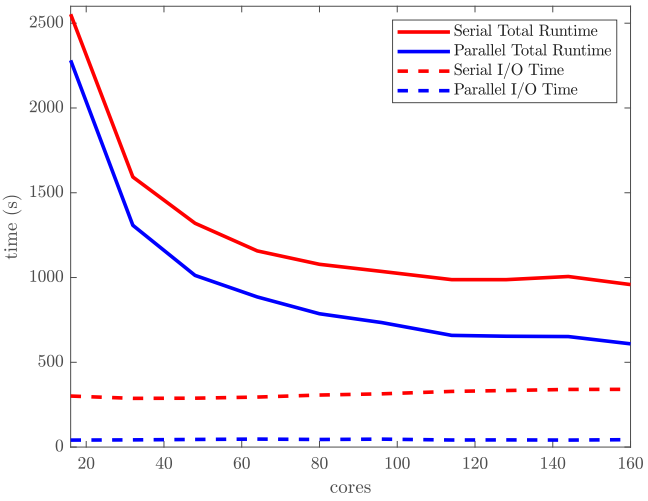

Fig. 11. Total time spent by GCCOM using serial and parallel I/O. The red lines are the serial runs. As the speedup scales with the number of cores used, the I/O stays constant. The difference between the times of the serial and parallel GCCOM is captured by the difference in I/O for serial netCDF versus PnetCDF.

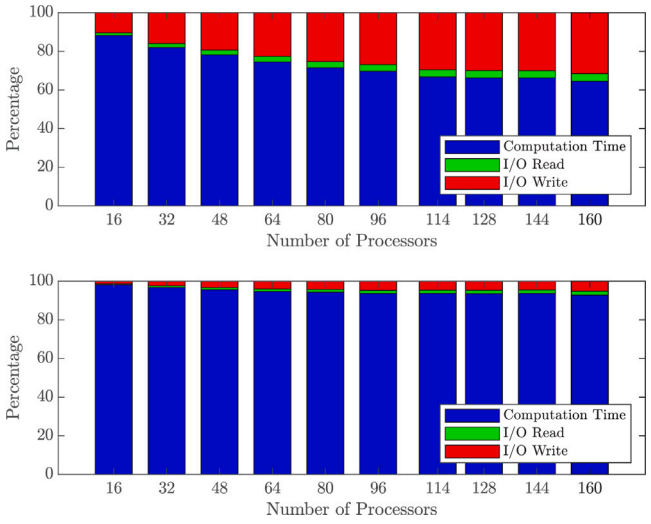

Fig. 12. Normalized percentage of time spent doing I/O, for the original serial I/O GCCOM model (top), and for the parallel I/O GCCOM (bottom).

`nf90_inq_varid` to `nf90mpi_inq_varid`. Writing to the netCDF variables code changes from 'nf_put_var' command to a `nfmpi_put_var`. Outside of the initial writing of the coordinate and latitude and longitude values, which was done in independent mode, all I/O is done in parallel. PETSc discretizes the domain into rectangular chunks, which leads to nice ordered arrays that remain consistent throughout the computation process. The arrays for the variable to be written are stored and indexed according to PETSc. Using these already stored values allowed directly giving the variable arrays to PnetCDF for writing with `nfmpi_put_vara`, using the indices from PETSc. This saved three `for` loops previously used for the x, y and z of each variable to order the indices, along with the associated gather and scatter calls PETSc would have used to get those variables onto the main process. An additional set of vectors needed to be created to store the size and count of the netCDF variable arrays for each processes sub-domain, but since the discretization does not change during the simulation, it only needed to be set once. The results of the PnetCDF implementation on run time are shown in Fig. 11 and as a percentage of run time in Fig. 12.

The problem tested had 7.6M points ($1601 \times 6 \times 801$), and each run output 10 times over one simulated second. The time savings

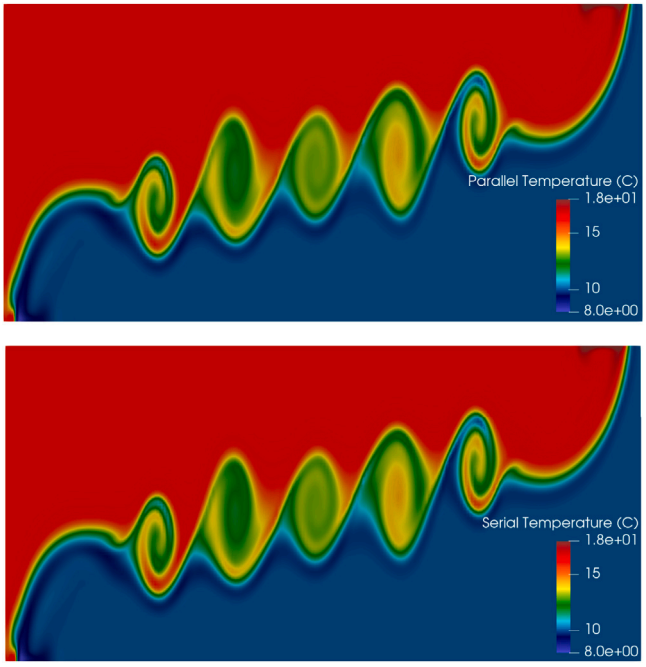

Fig. 13. Result from GCCOM for the lock release experiment comparing the parallel (top) output with the serial output (bottom) showing the result at 24 s of simulation. The numerical difference between these is exactly zero.

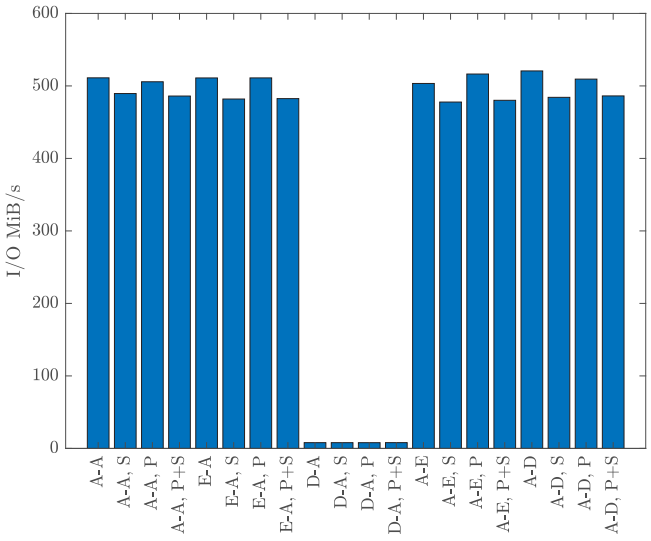

Fig. 14. Results for GCCOM run on 80 cores with different ROMIO and PnetCDF hints used. See Table 1 for explanation of the code. Default values (no hints) is the leftmost column. Disabling the `romio_cb_write` has a drastic effect on write speed. The max I/O rate is 545 MiB/s for collective buffering set to automatic, and data sieving enabled, with no hints passed.

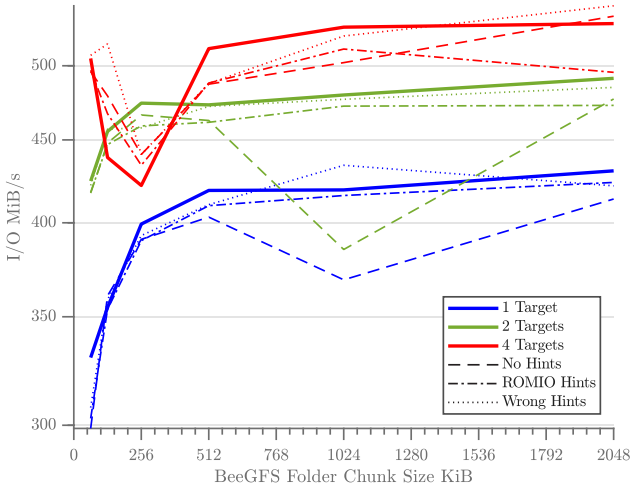

Fig. 15. GCCOM I/O, running on 5 nodes with 80 processes. The number of stripes and chunking was varied in BeeGFS, and hints were passed to the program. Bold lines represent the inclusion of PnetCDF hints. In general, ROMIO hints and PnetCDF hints resulted in the optimal speed for most BeeGFS chunking and striping sizes. Even when hints were passed that did not match the actual striping or chunking settings, results were generally better than no hints at all.

in the parallel I/O model using 160 cores is roughly equivalent to the reduction in time to write the data. The separation between the parallel and serial run times in Fig. 11 matches the relative percent used for I/O from Fig. 12. Some simple changes like removing a call to `nfmpi_sync` every write call saved an additional 20% of the parallel I/O time. Additionally, MPI features were tuned to decrease the run time, dependent on the number of processors, by an additional 1%–10%. MPI tuning was done by running `mpi-tune` on the executable file. To shorten the possible list of MPI functions to tune, a list of most used MPI functions was generated by enabling `MPI_STATS` and just using the most frequently used MPI commands. The two biggest users of MPI time were calls to `MPI_AllReduce` and `MPI_Waitall`. The `MPI_Allreduce` function had 6.6M calls in each execution, and used 81% of the time used by all MPI functions. The difference between the serial and parallel outputs is shown in Fig. 13, and the difference in outputs calculated by Paraview was exactly zero.

Further tuning was done using ROMIO hints, and PnetCDF specific hints which were passed as either command line arguments or environment variables during run time. The chunking of the folder where all I/O took place was set to 64 KiB chunks and striped across four targets. By varying the hints used, large differences could be seen in I/O speeds, as shown in Fig. 14.

As with Subflow, no major advantage could be seen at first in passing hints. The automatic output settings for ROMIO gives a write speed of 511 MiB/s, but there are major consequences when `romio_cb_write` is disabled. When disabled, write speeds dropped to 8 MiB/s. Again, like Subflow, this seems to negate the collective write of PnetCDF, resulting in an explosion of individual writes to disk. There also seem to be disadvantages to passing striping setting (S) to the model. When only stripe settings were passed, a drop averaging 25 MiB/s was observed (Fig. 14). To further explore the behavior of GCCOM, different hints were passed for different chunks sizes to see if other striping or chunking settings of BeeGFS resulted in better write speeds.

GCCOM was run in different folders of varying chunk and stripe settings. The simulation was run on 80 processes across 5 compute nodes. A baseline value for I/O speed was made for reading and writing data to folders of chunk sizes varying from 64 KiB to 2048 KiB by powers of two, and with the number of write targets being one, two or four. The same simulation was run passing ROMIO hints only to GCCOM including the number of write nodes, striping unit, and striping factor. A third run passed PnetCDF hints `nc_var_align_size` and `nc_header_read_chunk_size` about the writing directories to GCCOM. A fourth run passed the wrong hints to PnetCDF, consistently underestimating the number of `nc_var_align_size` and with ROMIO hints of striping unit and factor to match the PnetCDF hints. The results of this are in Fig. 15. It shows that for most scenarios, passing the PnetCDF hints about the striping and chunking performs better than no hints, or just the ROMIO hints alone.

The time savings in the parallel model was very close to the difference in I/O times between the models. I/O as a percentage of total run

time was also calculated. Here, computational time is given as $T_{total} = T_{Computational} + T_{I/O}$. The stacked bar charts of Fig. 12 show the serial I/O model compared to the parallel I/O model. Using 160 processors, the serial models spends 35% of the time doing I/O. This includes reading in the file and writing out each time step. Implementing PnetCDF reduces this down to 7.2% for the tuned optimum MPI model. As Fig. 12 shows, output time stays fairly consistent through the increase in processors. The small increases in time can be attributed to the increased number of MPI communications. Write speeds were measured and are shown in Fig. 14. Limitations of the size of the problem prevented testing larger file sizes but we expect this I/O performance to scale well to larger problems.

6. Conclusions and future work

Implementing parallel netCDF shows a significant decrease in I/O times versus a serial netCDF implementation. This scales well to two diverse geophysical models with different underlying architectures. Results show that a decrease in I/O time over serial can be achieved. It also shows that partial parallelization of I/O can yield significant results, and that PnetCDF has the capacity to do both serial and parallel I/O. While the ease of implementation cannot be expressed in charts or graphs, transitioning from netCDF to PnetCDF is relatively straightforward, and we hope this serves as a recommendation to transition any parallel code with serial I/O into full parallel I/O.

The interplay between BeeGFS and PnetCDF with the underlying MPI_IO routines seems to work well "out of the box". Disabling collective buffering has a detrimental effect on all tested programs, and should be avoided when using PnetCDF. Based on Fig. 15, passing PnetCDF hints about the underlying file structure generally results in better I/O for GCCOM, but had little effect for Subflow. This has many interesting features concerning performance. Of note is that the combination of ROMIO and PnetCDF hints is optimal regardless of BeeGFS chunking and striping settings. The effect is smallest when the chunk size is small, but for larger chunks, especially at 1024 KiB, the difference is as high as 75 MiB/s. There may be adjustments when dealing with large numbers of small reads and writes [8], but for general use and write speed optimization, this recommendation seems best.

It has also been shown that the GCCOM model scales well for larger problem sizes in terms of number of processors used. Previous results showed deadlock due to increased MPI and I/O traffic [32]. It remains to show this scaling continues for very large problems with GCCOM > 100M points, but this is reserved for future work as limitations of the RAM of the cluster prevented running larger problem sizes. Fig. 14 shows GCCOM gets roughly 5/6 the write speed of FLASH-IO. FLASH-IO uses cached writing exclusively yet does not suffer as badly the effects when collective buffering is disabled as much as GCCOM. However, Subflow only experiences a drop of 41% by disabling collective buffering and this could be due to the serial output. Further research into this difference is needed and possibly improving the memory handling in GCCOM to allow fully cached PnetCDF writes could improve the write speeds to be on equal that of FLASH-IO.

Implementation of PnetCDF to Subflow has shown a decrease in run time commensurate with the ability to parallelize output. Rewriting the code was straightforward, mostly involving translating netCDF commands into PnetCDF commands by adding an '`mpi`' to them. Small differences using a finite element domain discretization has shown to have an impact on reproducibility, but could be fixed by storing the variables as higher precision throughout the solution. Further parallelization of the Subflow model continues, in order to run larger simulations, and these will be sure to use parallel I/O. It is unclear if the lack of impact of PnetCDF hints to the I/O speed is due to the serial nature of some of the output, or something else. Future work intends to try and implement both serial and parallel output as buffered writes, to allow PnetCDF to write while Subflow continues its calculations.

CRediT authorship contribution statement

Jared Brzenski: Data curation, Writing - original draft, Investigation, Software, Validation. **Christopher Paolini:** Methodology, Conceptualization, Hardware. **Jose E. Castillo:** Supervision.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.parco.2021.102786.

References

[1] S.A. Brown, M. Folk, G. Goucher, R. Rew, P.F. Dubois, Software for portable scientific data management (NetCDF), Computers in Physics 7 (3) (1993) 304–308.

[2] W. Skamarock, J. Klemp, J. Dudhia, D. Gill, D. Barker, W. Wang, J. Powers, A Description of the Advanced Research WRF Version 3, Tech. Rep., 27, UCAR, 2008, pp. 3–27.

[3] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Parallel netCDF: A high-performance scientific I/O interface, in: SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, IEEE, 2003, p. 39.

[4] Lustre file system, 2010, http://opensfs.org/, accessed: 2010-09-30.

[5] F.B. Schmuck, R.L. Haskin, Gpfs: A shared-disk file system for large computing clusters, FAST 2 (2002).

[6] Q. Zheng, K. Ren, G. Gibson, B.W. Settlemyer, G. Grider, DeltafS: Exascale file systems scale better without dedicated servers, in: Proceedings of the 10th Parallel Data Storage Workshop, 2015, pp. 1–6.

[7] F. Herold, S. Breuner, J. Heichler, An introduction to beegfs, Tech. Rep., ThinkParQ, 2014.

[8] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, W. Yu, I/O characterization and performance evaluation of beegfs for deep learning, in: Proceedings of the 48th International Conference on Parallel Processing, 2019, pp. 1-10.

[9] N. Kondratyuk, G. Smirnov, E. Dlinnova, S. Biryukov, V. Stegailov, Hybrid supercomputer Desmos with Torus Angara interconnect: Efficiency analysis and optimization, in: L. Sokolinsky, M. Zymbler (Eds.), Parallel Computational Technologies, Springer International Publishing, Cham, 2018, pp. 77–91.

[10] D. Abramson, C. Jin, J. Luong, J. Carroll, A BeeGFS-based caching file system for data-intensive parallel computing, in: D.K. Panda (Ed.), Supercomputing Frontiers, Springer International Publishing, Cham, 2020, pp. 3–22.

[11] N. Mills, F.A. Feltus, W.B. Ligon III, Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks, Future Gener. Comput. Syst. 79 (2018) 190–198.

[12] J. Lüttgau, M. Kuhn, K. Duwe, Y. Alforov, E. Betke, J. Kunkel, T. Ludwig, Survey of storage systems for high-performance computing, Supercomp. Front. Innov. 5 (1) (2018) 31–58.

[13] P.M. Dickens, J. Logan, A high performance implementation of MPI-IO for a lustre file system environment, Concurr. Comput.: Pract. Exper. 22 (11) (2010) 1433–1449.

[14] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, A. Koniges, MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS, in: SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, IEEE, 2001, p. 58.

[15] R. Latham, R. Ross, R. Thakur, The impact of file systems on MPI-IO scalability, in: European Parallel Virtual Machine/Message Passing Interface Users Group Meeting, Springer, 2004, pp. 87–96.

[16] R. Thakur, E. Lusk, W. Gropp, Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation, Tech. Rep., Argonne National Lab., IL (United States), 1997.

[17] B. Palmer, A. Koontz, K. Schuchardt, R. Heikes, D. Randall, Efficient data IO for a parallel global cloud resolving model, Environ. Model. Softw. 26 (12) (2011) 1725–1735.

[18] P. Johnsen, Hurricane force supercomputing: Petascale simulations of sandy, 2013, Nov, URL https://www.hpcwire.com/2013/11/14/behind-blue-waters-hurricane-sandy-simulation/.

[19] Y.-H. Tseng, C. Ding, Efficient parallel I/O in community atmosphere model (CAM), Int. J. High Perform. Comput. Appl. 22 (2) (2008) 206–218.

[20] C. Paolini, A. Park, C. Binter, J. Castillo, An investigation of the variation in the sweep and diffusion front displacement as a function of reservoir temperature and seepage velocity with implications in CO2, in: 9th Annual International Energy Conversion Engineering Conference, IECEC 2011, 2011.

[21] C. Binter, C. Paolini, A.J. Park, J.E. Castillo, Utilization of reaction-transport modeling software to study the effects of reservoir temperature and seepage velocity on the sweep diffusion front displacement formed after CO_2-rich water injection, in: Tenth Annual Conference on Carbon Capture and Sequestration, 2011.

[22] B. Ganis, M.E. Mear, A. Sakhaee-Pour, M.F. Wheeler, T. Wick, Modeling fluid injection in fractures with a reservoir simulator coupled to a boundary element method, Comput. Geosci. 18 (5) (2014) 613–624.

[23] A.J. Park, Water-rock interaction and reactive-transport modeling using elemental mass-balance approach: I. The methodology, Am. J. Sci. 314 (3) (2014) 785–804.

[24] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, vol. 40, SIAM, 2002.

[25] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The deal.II library, version 9.0, J. Numer. Math. 26 (4) (2018) 173–183, http://dx.doi.org/10.1515/jnma-2018-0054.

[26] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[27] M. Abouali, J.E. Castillo, Unified curvilinear ocean atmosphere model (ucoam): A vertical velocity case study, Math. Comput. Modelling 57 (9–10) (2013) 2158–2168.

[28] P.F. Choboter, M. Garcia, D. De Cecchis, M. Thomas, R.K. Walter, J.E. Castillo, Nesting nonhydrostatic GCCOM within hydrostatic ROMS for multiscale coastal ocean modeling, in: OCEANS 2016 MTS/IEEE Monterey, IEEE, 2016, pp. 1–4.

[29] M. Garcia, J. Castllo, General Curvilinear Coastal Ocean Model (GCOM) User Guide, Computational Research Center, 2015, serucoam v3 2.

[30] M. Garcia, P.F. Choboter, R.K. Walter, J.E. Castillo, Validation of the nonhydrostatic general curvilinear coastal ocean model (GCCOM) for stratified flows, J. Comput. Sci. 30 (2019) 143–156.

[31] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object-oriented numerical software libraries, in: Modern Software Tools for Scientific Computing, Springer, 1997, pp. 163–202.

[32] M. Valera, M. Thomas, M. Garcia, J. Castillo, Parallel implementation of a PETSc-based framework for the general curvilinear coastal ocean model, J. Mar. Sci. Eng. 7 (2019) 185, http://dx.doi.org/10.3390/jmse7060185.

[33] J. Brzenski, M. Valera, J.E. Castillo, Coupling GCCOM, a curvilinear ocean model rigid lid simulation with SWASH for analysis of free surface conditions, in:OCEANS 2019 MTS/IEEE SEATTLE, 2019, pp. 1–8.

[34] J.M. Kunkel, E. Betke, M. Bryson, P. Carns, R. Francis, W. Frings, R. Laifer, S. Mendez, Tools for analyzing parallel I/O, in: International Conference on High Performance Computing, Springer, 2018, pp. 49–70.

[35] D. Capps, T. McNeal, Analyzing NFS client performance with IOzone, in: NFS Industry Conference, 2002.

[36] H. Shan, J. Shalf, Using IOR To Analyze the I/O Performance for HPC Platforms, Tech. Rep., Ernest Orlando Lawrence Berkeley NationalLaboratory, Berkeley, CA (US), 2007.

[37] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Parallel netcdf: A high-performance scientific I/O interface, in: SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, IEEE, 2003, p. 39.

[38] P. Malakar, V. Vishwanath, Hierarchical read-write optimizations for scientific applications with multi-variable structured datasets, Int. J. Parallel Program. 45 (1) (2017) 94–108.

[39] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, H. Tufo, FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes, Astrophys. J. Suppl. Ser. 131 (1) (2000) 273.

[40] R. Latham, C. Daley, W.-K. Liao, K. Gao, R. Ross, A. Dubey, A. Choudhary, A case study for scientific I/O: improving the FLASH astrophysics code, Comput. Sci. Discov. 5 (1) (2012) 015001.

[41] H. Song, Y. Yin, X.-H. Sun, R. Thakur, S. Lang, A segment-level adaptive data layout scheme for improved load balance in parallel file systems, in: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2011, pp. 414–423.

[42] W.-K. Liao, A. Choudhary, Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols, in: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE, 2008, pp. 1–12.