Increasing the Scalability of PISM for High Resolution Ice Sheet Models

Phillip Dickens
School of Computing and Information Sciences
University of Maine
Orono, Maine
dickens@umcs.maine.edu

Timothy Morey
Department of Computer Science
University of Maine
Orono, Maine USA
timothy.morey@maine.edu

***Abstract*—The issue of global climate change is of great interest to scientist and a critical concern of society at large. One important piece of the climate puzzle is how the dynamics of large-scale ice sheets, such as those in Greenland and Antarctic, will react in response to such climate change. Domain scientists have developed several simulation models to predict and understand the behavior of large-scale ice sheets, but the depth of knowledge gained from such models is largely dependent upon the resolution at which they can be efficiently executed. The problem, however, is that relatively small increases in the resolution of the model result in very large increases in the size of the input and output data sets, and an explosion in the number of grid points that must be considered by the simulation. Thus increasing the resolution of ice-sheet models, in general, requires the use of supercomputing technologies and the application of tools and techniques developed within the high-performance computing research community. In this paper, we discuss our work in evaluating and increasing the performance of the Parallel Ice Sheet Model (PISM) [6, 25, 38], using a high-resolution model of the Greenland ice sheet, on a state-of-the-art supercomputer. In particular, we found that the computation performed by PISM was highly scalable, but that he I/O demands of the higher-resolution model were a significant drag on overall performance. We then performed a series of experiments to determine the cause of the relatively poor I/O performance and how such performance could be improved. By making simple changes to the PISM source code and one of the I/O libraries used by PISM we were able to provide an 8-fold increase in I/O performance.**

Keywords: High resolution Ice sheet models, parallel I/O, scalable scientific simulations.

I. INTRODUCTION

The Parallel Ice Sheet Model (PISM) [6, 25, 38] is a widely used parallel simulation model that provides researchers with insight into the past, current, and future behavior of large-scale ice sheets. As is the case with other scientific simulations, the depth of knowledge that can be gained from the simulation is, to a large extent, dependent upon the resolution at which the model can be efficiently executed. The problem, however, is that relatively small increases in the resolution of the model result in very large increases in the size of the input and output data sets, and an explosion in the number of grid points that must be considered by the simulation. Thus to be a tool for continued scientific discovery, PISM must be *scalable*. That is, PISM must be able to execute efficiently in the face of rapidly increasing problem sizes.

The tremendous challenges brought about by increasing the resolution of ice sheet models can be seen in Table 1. This table depicts some of the characteristics of three data sets at different resolutions. As can be seen, there is a dramatic change in the problem requirements when moving from the 5-KM to the 1-KM resolution of the Greenland ice sheet. In particular, the number of grid points increases by a factor of 50 (from approximately 34 million to over 1.6 billion), and the size of the output file increases by a factor of 25 (from 1.1 GB to 28 GB).

PISM is deigned to scale with increasing problem size by harnessing the computational power of supercomputing systems and by leveraging the scalable software libraries that have been developed by the high-performance computing research community. The scalability of the computation is largely derived from its use of the Portable Extensible Toolkit for Scientific Computation (PETSc), which is a widely used library of data structures and routines for the numerical solution of partial differential equations [2, 3, 4]. PETSc, in turn, derives its scalability by spreading its computation across multiple processors/cores and using MPI [20] for inter-process communication.

Our concern in this paper is with the performance of the computational and I/O components of the PISM model. That is, we are approaching the problem from the point of view of high-performance computing, and the ways in which such performance can be optimized. We chose PISM as the focus of our study primarily because it is an important parallel ice sheet model that is utilized in a large number of scientific studies [6, 31, 37, 38].

Until recently, PISM was unable to execute the 1-KM model of the Greenland ice sheet because the I/O software [24, 35] was unable to scale to the increased problem size. The current release of PISM (PISM 0.5) addresses this issue by utilizing the NetCDF4 I/O package [35], which is able to handle the demands of the 1-KM resolution model.

Model	x	y	z	Total Grid Points	Approx. File Size
G1km	1,501	2,801	401	1,685,924,701	28 GB
A10km	600	600	301	108,360,000	1.8 GB
G5km	301	561	201	33,941,061	1.1 GB

Table 1: This table shows statistics about each model we are using, including the number of data points in each spatial dimension (x, y, and z), the total number of data points in the computational grid, and an approximation of the size of the output files produced by the models.

The NetCDF4 library is the latest iteration of Unidata's NetCDF library, and it includes many new features such as large file support and parallel file access. Both of these features are implemented through the use of the HDF5 library [11], which is a powerful suite of tools designed to efficiently manage very large and complex data sets. NetCDF4 [35] provides an enhanced interface on top of the HDF5 storage layer. The goal of this approach is to leverage the widespread use and simplicity of NetCDF with the large file support and parallel I/O performance provided by HDF5. It should be noted that HDF5 utilizes MPI-IO [22] as the basis of its parallel I/O.

At the same time that support for NetCDF4 was added, the PISM developers also added support for the Parallel-NetCDF library (PNetCDF) [18, 24]. This provides users with an alternate way to perform parallel file access. While Parallel-NetCDF does provide support for large datasets via version 5 of the Common Data Format (CDF5) file format, this functionality was not exposed in PISM. Thus, the NetCDF4 library must currently be used when performing large simulations. The reason for the lack of CDF5 support in PISM is due to the lack of tools that support CDF5, which, in turn, is due to Unidata's choice to not yet support CDF5. One reason for this is that Unidata does not find the user request bank in large enough quantity to justify the expense.

Parallel I/O improves I/O performance in much the same way that PETSc improves computational performance. That is, the file data is spread across multiple processes, and, in the best case, each process can write its data to independent regions of a shared file concurrently. This requires an underlying parallel file system that can handle multiple concurrent accesses to the same file. In this paper, we use Lustre [19] as the parallel file system.

The problem, however, is that even with a parallel I/O environment and a powerful underlying parallel file system, the I/O performance of PISM still represents a significant drag on the execution efficiency of the model. This is a problem that is common to scientific simulations, and has been an active area of research in HPC over the past several years [7, 8, 14, 17, 18, 33, 34]. This is a major issue with PISM, and this work seeks to understand and improve the I/O scalability characteristics.

The primary concern of this paper is to study, understand, and improve the performance of a large-scale ice sheet model on a powerful supercomputer. We present our work as a series of steps, each of which improves the overall performance of the simulation. As will be discussed, the initial performance of PISM on Ranger was quite poor, and, in fact, overall performance decreased as the number of processes across which it was executed increased. We then iteratively investigated and improved performance, such that we end up with an 8-fold increase in performance.

We believe this paper provides two key contributions to the scientific modeling community:

- It provides an extensive analysis of the performance and scalability of an out-of-pocket scientific application in a critical domain. To the best of our knowledge, this is the first such analysis for a parallel ice sheet model.
- It explains the very poor performance of PNetCDF in the PISM model, makes minor changes to the source code to fix the problem, and then provides significantly better performance than the other approaches.

The rest of the paper is organized as follows. Section 2 provides background information on ice sheet models in general and PISM in particular. Section 3 discusses the experimental design, as well as the architecture and I/O subsection of the supercomputer. Section 4 covers the steps taken to understand and improve the I/O performance. Section 5 discusses related work, and we provide our conclusions and future directions in Section 6.

II. BACKGROUND

A. Ice Sheet Modeling

Ice sheet modeling is concerned with the laws that govern the ebb and flow of large glaciers on our planet, as well on alien planets. Glaciers have been shown to have a dramatic influence on the climate of our world, and they contribute to sea-level rise when they melt [5]. However, the excessively slow speed at which they move presents difficulties to researchers. Direct experimentation with ice sheets is not possible for many hypotheses, as glaciers may appear stationary from day to day. It is only on the scale of years that a glacier may show noticeable change. For this reason, a computational model, that can simulate thousands of years of ice sheet behavior in minutes or hours, is a valuable tool to researchers.

The extents of a glacier can be measured with remote sensing techniques, so that we may approximate its geometry in a discrete structure. Through geological records and core samples taken from the ice itself, we are able to estimate how the physical extent of the ice has changed over time, thus providing a basis for computational models. These core samples can also provide hints about the climate conditions that accompanied changes in the ice sheet extents, thus providing one basis for explaining the complex relationship between climate and glaciers. [5]

B. PISM

There are many modeling techniques for ice sheets that have been developed over the years, and many variations of each such approach. These range from the relatively simple shallow-ice approximation [6, 38] to the computationally expensive Full-Stokes model [6, 9, 38]. PISM makes use of several established techniques, including the shallow-ice and shallow-shelf approximations [6], and it provides ways to combine and customize them depending on the problem under consideration [26]. PISM is one of several open source ice sheet models available today [10, 16, 30] but it is one of a very few (but growing number) of models that have a parallel architecture designed to scale to large problem sizes [9, 28].

Within PISM, the model takes place in a rectangular computational box that consists of a collection data points in

three dimensions. This box represents the space that encloses a glacier, and in the case that the model is based on a real-world glacier, each of these data points may be mapped to a unique geographical location in or near that glacier. Since most glaciers exist very near the poles of the planet, it is typical to express the coordinates of these points in a projected coordinate reference system based on the polar stereographic projection. While the points may not describe a regular rectangular space in the real world, they do define a regular rectangular space in their native coordinate reference system.

In the *x* and *y* dimensions, the grid points are equally spaced, and have a relatively coarse resolution. The *z* dimension is given a relatively finer resolution, and the spacing of grid points along this dimension may vary within a given model. Each *x y* pair defines a single rectangular column of ice that is parallel with the force of gravity. The structure of the grid is shown in Figure 1. When we say that we are modeling the Greenland ice sheet at one-kilometer resolution, we mean that each column of ice is a one-kilometer by one-kilometer square column. In contrast, the *z* dimension might consist of equally spaced grid points 10 meters apart.

PISM takes the entire computational box and divides it into *n* rectangular sub-grids, where *n* is the number of processes being used for the simulation. It attempts to make the sub-grids as square as possible in the *x* and *y* dimensions because the calculation of a new value for a given grid point often only depends on the current values of variables at adjacent grid points. Therefore, the degree to which the computation of one sub-grid depends on results from another sub-grid is proportional to the perimeter of local grid, and the square has the minimum perimeter for any rectangle of area *n***.** However, for many computational box sizes and values of *n*, there is no way to evenly distribute the grid points to *n* non-intersecting squares. In such cases, PISM arranges the sub-grids into *r* rows and *c* columns with $n = rc$, with no row being more than one grid point wider than any other row, and no column being more than one grid point taller than any other column.

The form of parallelism used in the Parallel Ice Sheet Model is process parallelism, where independent processes, possibly on separate machines, communicate using the Message Passing Interface (MPI) [20]. PISM uses the single-program, multiple-data (SPMD) paradigm, where each process owns an independent region of the model's computational box. The computation within these independent regions, and the MPI communication between the processes that own such regions, is largely performed by PETSc. An example of how processes can be assigned to regions of the computational model is also provided in Figure 1.

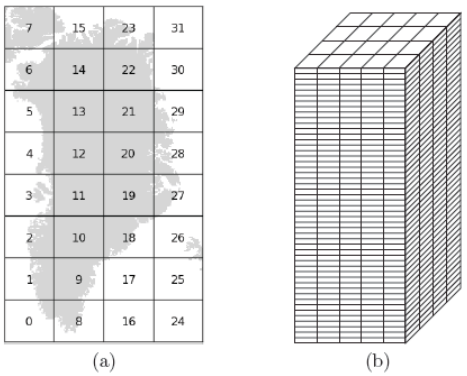

Figure 1: In (a), we see how regions of the computational box are mapped to MPI processes, identified by rank. Each process owns a rectangular sub-grid of the computational box that is structured like Figure (b), with the top face of the sub-grid in (b) corresponding to one of the rectangles in (a). Each space within the grid in (b) has many associated data values, such as temperature, which describe the current state of a block of ice, ocean, atmosphere, or ground that occupies that space.

It is important to note that the computation performed by PISM is highly scalable because each process only modifies data local to its region of the space (although it may have to read data from adjacent processes). This is a key point because all processes can largely perform their computation concurrently and independently. Thus the computation of PISM should scale close to linearly with the number of processes/cores, as long as there is enough work per process to mitigate the cost of the inter-process communication. This is confirmed through experimentation discussed below.

C. *PISM I/O*

Stated most simply, PISM reads input from, and writes output to, files defined by the Network Common Data Format (NetCDF) [35]. NetCDF is developed and maintained by the Unidata Program, which focuses on providing support for high-performance computing within the atmospheric and related geo-sciences [36]. At a lower level, PISM also takes advantage of HDF5 [11], developed by the HDF Group [13], which provides a data model, file format, and a set of tools to support large and complex scientific data sets. HDF5 also provides scalable parallel I/O, and a number of approaches (e.g., data chunking and chunk caching [12]), through which the efficiency of parallel I/O operations can be further enhanced. Both NetCDF and HDF5 files are known as *self-describing* because they maintain enough meta-data to describe the structure and meaning of the underlying data.

Until recently, PISM was unable to simulate the full Greenland ice sheet at 1-KM resolution because of the lack of support for large files. It now uses NetCDF4 [35], which was developed by Unidata to provide an enhanced NetCDF API on top of the HDF5 file format. In the remainder of the paper we term this approach NetCDF4/HDF5.

PISM also provides support for Parallel-NetCDF [18, 24] (termed PNetCDF), which was developed and is maintained at Argonne National Laboratory, and which provided the first support for parallel I/O for NetCDF files. However, PNetCDF can only scale to large files when utilizing the CDF5 file format [24], which is not currently supported by Unidata [35]. For reasons that will be explained later, when using PNetCDF to write data, PISM actually uses both NetCDF4 and PNetCDF to create and fill the file. Thus, PISM can only use earlier versions of the CDF format with PNetCDF, which allows us to use parallel I/O support, but which cannot scale to the large data sets required for higher-resolution models. We will refer to PISM's support of PNetCDF as PNetCDF/DS.

III. Experimental Design

A. Data Sets

The first model that we will be using is a model of the Greenland ice sheet, which is based on input data from the SeaRISE project. [29] This data can be used to model the ice sheet at a variety of horizontal resolutions, but we will be focusing on the one-kilometer resolution, which we refer to as the G1km model.

The G1km model consists of 1501 grid points in the *x* dimension, 2801 in the *y* dimension, and 401 in the *z* dimension, for a total of more than 1.6 billion grid points. A single three-dimensional variable, which consists of a double-precision floating-point value at each grid point, will require about 13 GB of storage space at G1km. A typical output file, which contains multiple one, two, and three-dimensional variables and all of the data necessary to restart the model, will be about 28 GB for G1km.

Due to the size of the G1km model, only the CDF5 and HDF5 file formats may be used at this resolution. Additionally, the size of G1km also presents problems at run-time, as the model requires more than a terabyte of memory when the simulation is running. Few workstations provide this amount of memory, so it is necessary to use a supercomputer for G1km.

As mentioned previously, the source data for the models was provided by the SeaRISE project [29], but this data was modified significantly to prepare it for our test runs. In particular, the data was preprocessed to put it into a format that PISM can use, and it was processed with the PISM spin-up procedure. This procedure transforms the preprocessed data into a PISM model state that represents the current state of the Greenland ice-sheet, and it is a very time consuming procedure.

Due to constraints on computational resources, performing a full spin-up for the G1km model was not practical, so we decided to simply re-sample the G5km model state to the G1km resolution. The resulting model may not have the predictive abilities that a properly spun-up model would have, but it has the correct number of data points and the data points do describe a glacier.

The results use the spun-up data as input and perform a predictive simulation. The parameters for this simulation are based on the parameters of the first experimental control run given in the PISM example scripts, which is described in [26]. The only modification made was to shorten the simulation to run for zero years, rather than the suggested 500 years. This was done so that we could more easily focus on the initialization and output procedures without spending all of our computation resources performing unnecessary calculations. However, even when running for zero years, PISM performs a single time step, which allows us to measure the compute performance.

We are also using a model of the Antarctic ice-sheet at 10 kilometer resolution, which we will refer to as A10km. As with the G1km and G5km models, the A10km model was based upon SeaRISE data [29], and was produced through the example pre-processing and spin-up scripts distributed with PISM. The parameters of our test runs were again based upon the first experimental control run provided in PISM's example scripts. [26] The size of the A10km model falls between G1km and G5km models, and is summarized in Table I above.

B. Supercomputer

All of our experimental work was performed on Ranger: a supercomputer housed at the Texas Advanced Computing Center at the University of Texas at Austin. Ranger consists of 3,936 compute nodes, each of which consists of 4 quad core AMD Operton processors and 32 GB of memory. Thus each compute node provides 16 processors, for a total of 62,976 compute cores. The system provides a total of 123 TB of memory, and 1.7 PB of raw global disk space. All Ranger nodes are interconnected using InfiniBand technology [15] in a full-CLOS topology providing 1GB/sec of point-to-point bandwidth. Ranger uses version 2.6.18.8 of the Linux kernel, and the software stack is built with the Intel compiler. Its MPI-IO implementation is MVAPICH2-1.8 [23].

C. Lustre Parallel I/O Subsystem

Lustre [19] consists of three primary components: file system clients (that request I/O services), object storage servers (OSSs) (that provide I/O services), and meta-data servers that manage the name space of the file system. Each OSS can support multiple Object Storage Targets (OSTs) that handle the duties of object storage and management. The scalability of Lustre is derived from two primary sources. First, file metadata operations are de-coupled from file I/O operations. The metadata is stored separately from the file data, and once a client has obtained the metadata it communicates directly with the OSSs in subsequent I/O operations. This provides significant parallelism because multiple clients can interact with multiple storage servers in parallel. The second driver for scalable performance is the striping of files across multiple OSTs, which provides parallel access to shared files by multiple processes.

Lustre provides APIs allowing the application to set the stripe size, the number of OSTs across which the file will be striped (the stripe width), the index of the OST in which the first stripe will be stored, and to retrieve the striping information for a given file. The stripe size is set when the file is opened and cannot be modified once set. Lustre

assigns stripes to OSTs in a round-robin fashion, beginning with the designated OST index.

The Lustre file system on Ranger consists of six Sun x4600 metadata servers, 72 Sun x4500 disk servers each containing 48 SATA drives for an aggregate storage capacity of 1.73 petabytes. The file system is partitioned into three Lustre file systems. On the Scratch file system used in these experiments, there were 50 OSSs, each of which hosted six OSTs, for a total of 300 OSTs. The bottleneck in the system was the 1-Gigabyte per second throughput from the OSSs to the Inifiband network.

IV. Experimental Progression

A. *Scalability of Smaller Resolution Models*

One advantage of using real-world applications is the ability to look at how the computation scales with increasing problem size, and whether there exists a trade-off between scalable computation and scalable I/O performance. The first set of experiments looked at the computational scalability.

In these experiments, we used the A10K model and measured the time taken to complete the first time-step of the model state. We varied the number of nodes between 4 and 64, and measure the compute time as a function of the number of processes. For each set of nodes, we varied the "wayness" utilized (i.e., core per node) at 1, 4, 8, and 16. Thus at the largest level, we utilized a total of 1024 processing cores.

The results are shown in Figure 2. As can be seen, the computational component of PISM scales exceptionally well as the number of compute core is increased. The reason for such excellent scalability was alluded to above: there was one process per core, each process was using PETSc for the computation, and every process wrote strictly into its region of the computational grid. PETSc performed the computation and managed all of the inter-process communication through MPI-IO.

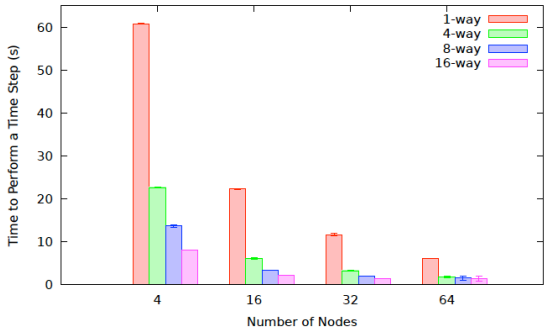

Figure 2: Time required to complete one time step of the model as a function of the number of nodes and number of processes per node (wayness) using the A10km data set.

B. *Scalability of Computation and Write Time*

The next set of experiments looked at the overall run time (i.e., time taken to initialize the model, compute the first time-step, and write the model state to disk) using the A10km data. The results are show in Figure 3. Two points are immediately clear. First, adding the I/O workload had a significant impact on the runtime, particularly as the number of processes/core per node increased. Second, PNetCDF/DS out-performed the NetCDF4 approach by a factor of two in the case of 1024 processes, 16 processes per node.

This is an interesting result, because it indicates that there is some sort of trade-off between maximizing computational scalability (by utilizing the maximum number of core) and I/O scalability (by utilizing fewer core). Further research is needed to understand this phenomenon, and why it appears to impact one approach more significantly than the other even though both are using MPI-IO.

C. *Scalability of Write Time*

We next focused only on the time required to write the model state at the end of the first time-step using the A10km model. The results are shown in Figure 4. As can be seen, there is a dramatic difference in performance, with PNetCDF/DS significantly outperforming NetCDF4. We also note that increasing the number of processes significantly decreases the I/O performance in both cases, but to much larger extent in NetCDF4.

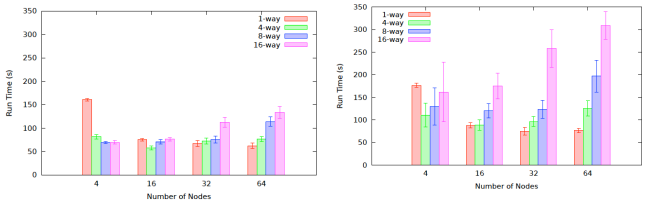

Figure 3: The graph on the left shows the overall runtime for a complete time step as a function of the number of nodes and processes per node using the PNetCDF/DS approach. The figure on the right shows the same information using NetCDF4. These results are from the A10km model.

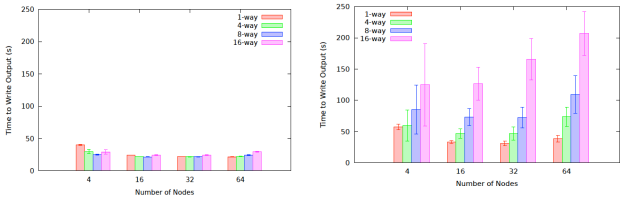

Figure 4: The graph on the left shows the time required to write out the model state as a function of the number of nodes and processes per node using the PNetCDF/DS approach. The figure on the right shows the same information using NetCDF4. These results are from the A10km model.

D. *Scaling to 1km Resolution*

Motivated by the relatively better performance of the PNetCDF library, we decided to implement large file support for PNetCDF. PISM currently only provides support for such high-resolution data via the NetCDF4 library. However, Argonne's Parallel-NetCDF library [18, 24] also provides support for such high-resolution data via the CDF-5 format, and it was not difficult to ensure that the PNetCDF library is used for writing file data and metadata using the CDF5 format. We term this approach PNetCDF/CDF5.

Moving to the CDF5 version revealed an interesting feature in PISM: the Parallel-NetCDF library was never being used to write the metadata. This was an interesting phenomenon, and we discovered comments in the PISM source code explained that PNetCDF was not being used to write metadata for performance reasons. The PISM developers had performed some experimentation and found that they were able to achieve better performance with a hybrid approach, rather than using PNetCDF directly. The hybrid approach involved using NetCDF4 in the legacy serial mode to create the file structure and then used PNetCDF to write the data.

After modifying PISM to use PNetCDF for both data and metadata, we attempted to run a G1km simulation, which requires the large file support. However, we found that PISM would become unresponsive while writing its output, and the run would not complete in a reasonable amount of time. Due to the cost of computational resources, we did not allow the simulation to run for an arbitrary amount of time to see if it would have completed, but rather stopped it after it had run for twice as long as it would have required for the equivalent run with NetCDF4 output.

In order to learn more about the new implementation, we attempted an A10km run, and allowed it to run to completion. After a lengthy pause during the output phase, the simulation did complete, and it produced valid output. The results are shown in figure 5, and, as can be seen, using PNetCDF/CDF-5 to write meta-data and data resulted in a *10-fold decrease in performance.* To understand the reason for such poor performance, it is necessary to briefly discuss the structure and operations of CDF5 files.

The NetCDF4 and PNetCDF libraries define two modes of execution when writing to a file: *define-mode* and *data-mode*. In define-mode, the user specifies the structure of the dataset by defining the variables metadata. The user must then switch to data-mode to write the actual variable data. However, switching from define-mode to data-mode can be an expensive operation, due to the structure of a CDF file.

The structure of a CDF5 file consists of three components: the header, which contains all of the metadata; the non-record variable section, which contains fixed-size variables, and the record section, which contains all record variables. Record variables are those that make use of the unlimited dimension and are therefore allowed to grow along that dimension. In contrast, non-record variables are those that have a fixed size, which must be specified when the variable is defined. This ordering of these three sections must be maintained. Thus, if the user writes data for a record variable and then switches to define mode to define a new non-record variable, then the file must be restructured. In particular, the non-record section must be expanded to accommodate the variable, which requires that the record data be moved. Thus one must take care to define all variables before writing any data.

Violating this pattern was the cause of the extremely poor performance. In the case of the G1km model, which could not run to completion, a single record is many gigabytes of data, all of which must be moved to accommodate the new file structure. A single record is much smaller in the A10km model, which did run to completion, but resulted in the 10-fold decrease in performance shown in Figure 5.

The PISM developers circumvented this problem by using the hybrid approach discussed above, and the problem does not occur with HDF-5. We remedied this problem by a slight modification to the PISM write pattern, which resulted in a tremendous increase in performance.

After diagnosing and correcting this problem, we returned to the 1KM Greenland model. Figure 6 shows the write performance using both NetCDF4 and PNetCDF/CDF5. As can be seen, using PNetCDF/CDF5 resulted in a speedup in performance by approximately a factor of 8.

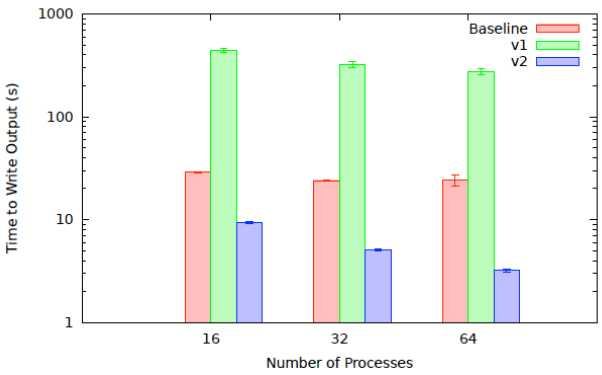

Figure 5: This shows the progression in our research related to the very poor I/O performance when using PNetCDF/CDF-5 to write the file data and metadata. The "baseline" data is the native performance of PISM 0.5 which uses the hybrid approach. The V1 data shows the increased cost when of using PNetCDF/CDF-5. "V2" shows the write performance when we corrected the write pattern. Note that the scale is logarithmic due to the wide variations in run time.

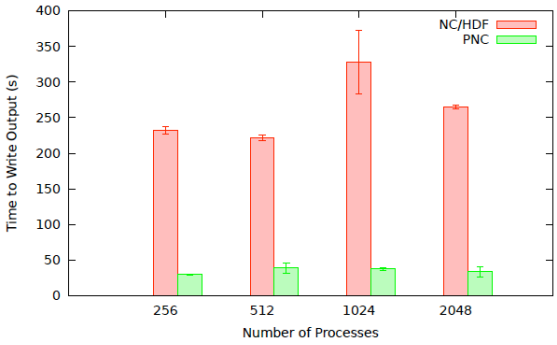

Figure 6: This shows the time to output model state using NetCDF4 (labeled NC/HDF) and PNetCDF/CDF-5 (labeled PNC), for the G1km resolution model.

V. Related Work

It is well known that I/O is a frequent bottleneck in the performance of parallel, scientific applications [7, 8, 14, 17, 18, 33, 34]. Through the years, a multitude of approaches have been developed to minimize the cost of I/O in parallel applications. MPI-IO [22] defines a standard interface for parallel I/O as well as collective file access semantics. The ROMIO [33] implementation of MPI-IO introduces optimizations for parallel I/O, in the form of data-sieving and two-phase I/O. [34]

The NetCDF4 [35], PNetCDF [18, 24], and HDF5 [11] libraries have been developed to make the semantics of MPI-IO and the optimizations of ROMIO more accessible to application developers. They allow the application developer to represent the application data in a format that is convenient on the application level, and they provide functionality to translate between the application's data-model and a serialized data-model that will fit in a file. The Adaptable Input/Output System (ADIOS) [1] is another system that seeks to reduce the cost of I/O in parallel applications. It introduces a new file format, but also provides tools to translate this format to and from the NetCDF and HDF5 formats.

In this work, we have made use of several of the libraries and techniques described above to analyze and improve the I/O performance of PISM. In [14], a similar study was performed, focusing on different parallel models and exclusively using the HDF5 format. In [18], the performance of the initial implementation of PNetCDF was compared to the (then serial) NetCDF library as well as the parallel HDF5 library using benchmark applications. To the best of our knowledge, no other studies of the I/O performance of ice sheet models have been performed.

VI. Discussion

It is important to re-iterate that our goal was to utilize PISM out of the box and did not attempt to optimize any of the I/O libraries with the one exception discussed above. Thus our research shows that in this important model, the NetCDF4 implementation provides significantly poorer performance than the PNetCDF implementation. There are several possibilities for optimization of both libraries, and HDF5 provides many mechanisms for performance enhancements [12, 14]. It would be interesting to look at the impact of such options, particularly since PISM is a widely used model in a critical domain.

Currently, PNetCDF is not available on PISM for large files. One likely reason is the very poor performance achieved when using the writing pattern discussed above. This study suggests that PNetCDF can, in fact, provide excellent performance.

There are several areas in which this research can be extended. First, we would like to utilize the optimizations that are available in the NetCDF4 library and determine their impact on performance. Similarly, we would like to spend more time optimizing PNetCDF through the use of Y-Lib [8], which has been shown to provide orders of magnitude performance increase over native ROMIO [33], which is the foundation upon which all of these libraries are based. Finally, we would like to utilize the ADIOS [1] platform to compare these and perhaps other optimization techniques.

Acknowledgments

We acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu

References

[1] Adios - Oak Ridge Leadership Computing Facility. www.olcf.ornl.gov/center-projects/adios/, 2012.

[2] Satish Balay, Jed Brown, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

[3] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2012. http://www.mcs.anl.gov/petsc.

[4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, Modern Software Tools in Scientific Computing, pages 163-202. Birkhauser Press, 1997.

[5] R. A. Bindschadler and 27 others. Ice-sheet model sensitivities to environmental forcing and their use in projecting future sea-level (the SeaRISE project). Journal of Glaciology, Submitted, 2012.

[6] E. Bueler and J. Brown. Shallow shelf approximation as a "sliding law" in a thermodynamically coupled ice sheet model. J. Geophys. Res., 114, 2009.

[7] Kenin Coloma, Avery Ching, Alok Choudhary, Wei keng Liao, Rob Ross, Rajeev Thakur, and Lee Ward. A new flexible MPI collective I/O implementation. In Proceedings of the IEEE International Conference on Cluster Computing, September 2006.

[8] Phillip M. Dickens and Jeremy Logan. A high performance implementation of MPI-IO for a Lustre file system environment. Concurrency and Computation: Practice and Experience - Grid Computing, High Performance and Distributed Application, Volume 22, August 2010.

[9] Elmer/ice. http://elmerice.elmerfem.org/, 2012.

[10] Glimmer community ice sheet model. http://glimmer-cism.berlios.de/, 2012.

[11] Hierarchical data format version 5. http://www.hdfgroup.org/HDF5, 2000-2010.

[12] HDF5 user's guide. http://www.hdfgroup.org/HDF5/doc/UG/, 2012.

[13] The hdf group. http://www.hdfgroup.org/, 2012.

[14] Mark Howison, Quincey Koziol, David Knaak, John Mainzer, and John Shalf. Tuning HDF5 for Lustre File Systems. In Proceedings of 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10), Heraklion, Crete, Greece, September 2010. LBNL-4803E.

[15] Infiniband trade association home. http://www.infinibandta.org/, 2012.

[16] ISSM: Ice Sheet System Model. http://issm.jpl.nasa.gov/, 2012.

[17] Wei keng Liao and Alok 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, 2008.

[18] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netcdf: A high-performance scientific i/o interface. In Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC '03, pages 39-, New York, NY, USA, 2003. ACM.

[19] Lustre. http://wiki.lustre.org/index.php/Main_Page, 2012.

[20] Message Passing Interface (MPI) Forum Home Page. http://www.mpi-forum.org/, 2012.

[21] MPICH. http://www.mpich.org/, 2012.

[22] MPI-2: Extensions to the message-passing interface. http://mpi-forum.org/docs/mpi-20-html/mpi2-report.html, 1997.

[23] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/, 2012.

[24] Parallel netCDF: A high performance api for NetCDF file access. www.mcs.anl.gov/parallel-netcdf, 2012.

[25] PISM, a Parallel Ice Sheet Model. http://www.pism-docs.org, 2012.

[26] PISM, a Parallel Ice Sheet Model: User's manual. http://www.pism-docs.org/wiki/lib/exe/fetch.php?media=manual.pdf, 2012.

[27] Repository for Parallel Ice Sheet Model (PISM). https://github.com/pism/pism, 2012.

[28] SEACISM: A Scalable, Efficeint and Accurate Community Ice Sheet Model. http://www.csm.ornl.gov/SEACISM/, 2012.

[29] Searise assessment. http://websrv.cs.umt.edu/isis/index.php /SeaRISE_Assessment/, 2012.

[30] Ice sheet model sicopolis. http://sicopolis.greveweb.net/, 2012.

[31] A. Solgaard and P. Langen. Multistability of the greenland ice sheet and the effects of an adaptive mass balance formulation. Climate Dynamics, 2012.

[32] Texas Advanced Computing Center. Ranger user guide. http://www.tacc.utexas.edu/user-services/user-guides/ranger-user-guide, 2012.

[33] Rajeev Thakur, William Gropp, and Ewing Lusk. Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Mathematics and Computer Science Division, Argonne National Laboratory, October 1997. ANL/MCS-TM-234.

[34] Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in mpi-io. Parallel Computing, 28:83-105, 2002.

[35] Unidata. NetCDF (Network Common Data Form). http://www.unidata.ucar.edu/software/netcdf/, 2012.

[36] Unidata. http://www.unidata.ucar.edu/, 2012.

[37] W. J. J. van Pelt and J. Oerlemans. Numerical simulations of cyclic behaviour in the parallel ice sheet model (PISM). Journal of Glaciology, 58(208):347{360, 2012.

[38] R. Winkelmann, M. A. Martin, M. Haselo_, T. Albrecht, E. Bueler, C. Khroulev, and A. Levermann. The Potsdam Parallel Ice Sheet Model (PISM-PIK) Part 1: Model description. The Cryosphere, 5:715-726, 2011.