Supporting Computational Data Model Representation with High-performance I/O in Parallel netCDF

Kui Gao, Chen Jin, Alok Choudhary, and Wei-keng Liao
Electrical Engineering and Computer Science Department, Northwestern University
{*kgao, cji970, choudhar, wkliao*}*@eecs.northwestern.edu*

***Abstract*—Parallel computational scientific applications have been described by their computation and communication patterns. From a storage and I/O perspective, these applications can also be grouped into separate data models based on the way data is organized and accessed during simulation, analysis, and visualization. Parallel netCDF is a popular library used in many scientific applications to store scientific datasets and provides high-performance parallel I/O. Although the metadata-rich netCDF file format can effectively store and describe regular multi-dimensional array datasets, it does not address the full range of current and future computational science data models. In this paper, we present a new storage scheme in Parallel netCDF to represent a broad variety of data models used in modern computational scientific applications. This scheme also allows concurrent metadata construction for different data objects from multiple groups of application processes, an important feature in obtaining a high degree of I/O parallelism for data models exhibiting irregular data distribution. Furthermore, we employ non-blocking I/O functions to aggregate irregularly distributed data requests into large, contiguous data requests, to achieve high-performance I/O. Using an example of adaptive mesh refinement data model, we demonstrate the proposed scheme can produce scalable performance results for both data and metadata creation and access.**

***Keywords*-Parallel I/O, Parallel netCDF, Data Model**

I. Introduction

Modern parallel computers are increasingly used to solve large, data-intensive scientific applications, such as climate modeling, fusion, fluid dynamics, and computational biology. Parallel computational scientific applications have been described by their computation and communication patterns. The taxonomy consists of 13 data models [1], [2]. Each data model has a particular pattern of computation and communication. High I/O performance is critical from a performance and productivity perspective, to support interpretation of computational results and operation of these codes at fidelities enabled by extreme-scale computers. Codes across the range of computational data models are finding it difficult to perform efficient I/O on todays petascale architectures. The I/O inefficiency problem may limit the ability of these applications to achieve exascale performance.

The Network Common Data Format (netCDF) [3], [4] defines a set of I/O functions and a machine-independent file format to support the creation, access, and sharing of array-oriented scientific data. Data stored in netCDF format is defined as array variables with well-defined attributes, such as dimensions, data types, and annotations. NetCDF is widely used by many scientific applications to store data in a portable file format with the related metadata. To support parallel I/O, Parallel netCDF (PnetCDF) [5], a popular parallel I/O library in many scientific applications, defines a set of parallel functions of which implementation is built on top of MPI-IO [6] to provide high performance. Although the metadata-rich netCDF file format can effectively store and describe regular multi-dimensional datasets, it does not address the full range of current and future computational science data models.

Existing parallel I/O methods in PnetCDF concentrate on optimizing the process collaboration under a fairly evenly-distributed request pattern. However, these parallel I/O methods are not suitable for an irregular data set, because the underlying data distribution is highly irregular and dynamic. Process synchronization in the existing parallel I/O methods, such as MPI-IO, can penalize the I/O parallelism if the process collaboration is not carefully coordinated. Due to the irregularity of data distribution in some data models, the I/O workload can vary widely depending on computational region of interest during the runtime. This intensive and dynamic I/O behavior imposes a big challenge to achieve scalable and sustainable I/O performance when applications run on high-performance computing (HPC) systems.

Currently, there is no existing parallel I/O library that supports complex data models, i.e. arbitrary data relationships among data objects. Only HDF5 [7] supports a tree-structured model through group, but not for arbitrary graphs. PnetCDF and netCDF do not address the issue of irregular data sets. What this paper proposes is to allow applications to systematically store complex data relationships in files. We believe this paper is the first to propose a data model to support an arbitrary relationship in the high-level I/O libraries.

This paper's two main contributions are 1) a new metadata representation scheme in PnetCDF for supporting a broad variety of data models, and 2) new metadata and I/O methods that achieve high degrees of I/O parallelism and performance. In the new metadata representation scheme, we use a graph data structure that consists of a set of

vertices and edges connecting the vertices to depict the relationship of multiple data objects in a data model. A metadata format similar to the netCDF file header is also developed to describe the graphs. We store each graph as a regular netCDF variable, so the files created under the new scheme will conform with the netCDF file format. To improve metadata I/O parallelism for irregular data distribution exhibited in some data models, we make all metadata functions non-collective, which allows different data variables to be defined by different groups of processes concurrently. In the current form of PnetCDF, defining variables is a collective operation, even if the variable is only partitioned among a subset of processes. In the new scheme, a process will only include the define functions for the variables it accesses. We use the AMR (Adaptive mesh refinement) example to illustrate the complex relationship among data objects. These complex relationships are not addressed sufficiently or supported by existing parallel I/O libraries, such as PnetCDF and HDF5. Because different data models can potentially have different relationships, we propose the use of a general graph representation to store the relationship information in netCDF files. This general graph approach supports AMR but is not limited to AMR and tree-type data models. Representing irregular data models is more challenging than representing regular models which has already been addressed by PnetCDF and HDF5. To provide high-performance I/O for irregular distributed data, we use non-blocking I/O functions to aggregate the I/O requests into large, contiguous I/O requests. We evaluate this work by using the Chombo I/O benchmark, an example of the adaptive mesh refinement data model, and demonstrate scalable performance results achievable from the proposed scheme. The proposed non-collective I/O for metadata access also works for regular data models becaue the regular data models are special cases of the irregular data models.

The rest of this paper is organized as follows. The related work is discussed in Section II. Section III presents the design of new storage scheme. Section IV describes the implementation issues. Section V presents the performance evaluation and the conclusions are presented in Section VI.

II. Related Work

As described in [1], [2], there are thirteen popular data models in the computational scientific application field. The data model is a key part of HPC codes and strongly influences the efficiency of both computation and communication. At their core, scientific codes usually store the data model as multi-dimensional arrays bacause these structures are usually allocated and processed most efficiently on modern computer architectures. Scientific applications have begun to rely heavily on high-level I/O application programming interfaces (APIs) such as HDF5 [7] and PnetCDF [5] for their storage needs. These APIs allow scientists to describe their data in meaningful scientific terms as structured, typed data, and to store and retrieve this data in a manner that is portable across all the platforms. Because scientists have a richer language with which to describe their data, I/O for an application as a whole can be described in terms of the datatypes and organizations that the scientist is really using, rather than posing I/O operations in terms of the independent reads or writes of bytes on many processors. Recently, the computing community has focused a great deal of effort on describing the data models in high-level I/O libraries. The H5hut library [8] implement several data models for particle-based simulations on HDF5. The F5 library [9] supports a range of grids, meshes, and user-defined compound data types in HDF5. The Silo Library [10] also provides data model abstractions for representing many mesh types, variable types, parallel decompositions, on top of the HDF5 and NetCDF libraries.

A. Parallel netCDF

Dataset storage, exchange, and access play a critical role in scientific applications. NetCDF serves as a software library and self-describing machine independent data format that support the creation, access, and sharing of array-oriented scientific data. NetCDF stores data in an array-oriented dataset which contains dimensions, array variables, and attributes [4]. A netCDF file format, as show in Figure 1, is divided into three parts: file header, non-record array variables and record array variables. The netCDF file header stores metadata, such as array dimensions, names and sizes of dimensions, data types, and character strings for annotations. The dimension metadata are used to define the shapes and attributes of array variables. Non-record variables are arrays of fixed sizes in all dimensions. Record variables allow the most significant dimension to be defined as ”unlimited”, which means the arrays can grow along that dimension. Non-record variables are stored in the file prior to all the record variables.

Parallel netCDF (PnetCDF) defines a set of APIs for creating array objects, adding attributes, and accessing them in parallel. The APIs breaks file access into two modes, define and data modes. The define mode is used to define the data structure of array variables and the data mode is used for accessing the variables. The file header is read/written only by the root process, although a copy is cached in the local memory on each process. Header modifications are made in define mode. The root process fetches the file header, broadcasts it to all processes when opening a file, and writes the file header at the end of the define mode if any modifications occur in the header. The define mode functions, attribute functions, and inquiry functions all work on the local copy of the file header. All define mode and attribute functions are made collectively and require all the processes to provide the same arguments when adding, removing, or changing definitions so the local copies of the file header are guaranteed to be consistent across all

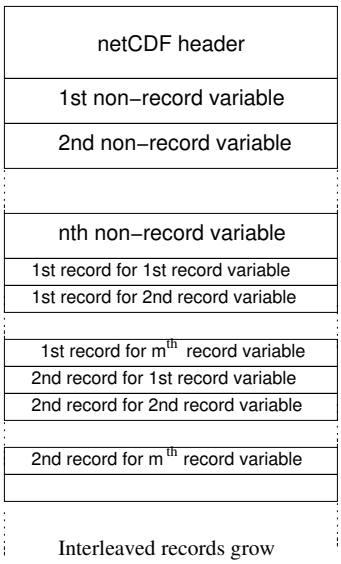


Figure 1. The netCDF file structure.

processes from the time the file is collectively opened until it is closed.

B. HDF5

Hierarchical Data Format 5 (HDF5) [7] is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format. HDF5 can store large numbers of large data objects, such as multidimensional arrays, tables, and computational meshes, and these can be mixed together in any way that suits a particular application. An HDF5 file is a container for storing a variety of scientific data is composed of two primary types of objects: groups and datasets [11]:

1) HDF5 group: a grouping structure containing HDF5 objects, together with supporting metadata;
2) HDF5 dataset: a multidimensional array, together with supporting metadata.

Any HDF5 group or dataset may have an associated attribute list. An HDF5 attribute is a user-defined structure that provides extra information about an HDF5 object. Working with groups and datasets is similar in many ways to working with directories and files in UNIX. HDF5 supports cross platform portability of the interface and corresponding file format, as well as ease of access for scientists and software developers.

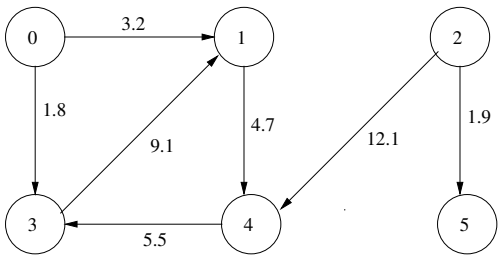


Figure 2. An example of directed graph that can be represented in the proposed scheme. The directed graph contains six vertices and seven edges. Each directed edge can have a different weight.

C. MPI-IO

MPI [12] is the building foundation of PnetCDF and HDF5 for parallel I/O operations. MPI-IO is an important feature of the MPI-2 standard [6], which allows multiple processes of a parallel program to access data in a shared file simultaneously. MPI-IO inherits two important MPI features: MPI communicators defining a set of processes for group operations, and MPI derived datatypes describing complex memory layouts. A communicator specifies the processes that participate in a collective operation for both inter-process communication and file I/O. When opening a file, the MPI communicator is a required argument to indicate the group of processes accessing the file. There are generally two types of functions defined in MPI-IO: collective and independent. The collective functions require participation of all processes that collectively open the file. Many collaboration strategies have been proposed and demonstrated their successes with significant performance improvements over uncoordinated I/O, such as two-phase I/O [13], [14], disk directed I/O [15], server-directed I/O [16], persistent file domain [17], [18], active buffering [19], and collaborative caching [20], [21]. However, there has not been much effort or demonstration in using MPI-IO for accessing hierarchical and irregularly distributed data sets in high performance. Ironically, the performance obstacle for such I/O patterns is mostly caused by the exact collectiveness requirement that enables process collaboration. Independent I/O functions, in contrast, require no coordination but make any collaborative optimization very difficult.

III. Design

A data model describes how simulation data is represented and accessed. From the application (or end-user) perspective, the spatial domain in the data model can be represented as a mesh, grid or tree, where each cell also includes all its lower-dimensional sub-cells. Fields are defined over d-dimensional cells, with multiple values for different time or other independent variable discretization points. The netCDF file format is designed for dense, multi-dimensional arrays. Describing complex relationships among multiple variables,

```
int ncmpi_def_graph(int ncid,
          const char *name,
          int *graphid);

int ncmpi_def_vertex(int ncid,
          const char *name
          int *vertexid);

int ncmpi_def_edge(int ncid,
          const char *name,
          double weight,
          int direction,
          int *edgeid);

int ncmpi_add_vertex(int ncid,
          int graphid,
          int vertexid).

int ncmpi_add_edge(int ncid,
          int vertexid,
          int edgeid,
          int adj_vertexid);

int ncmpi_add_var(int ncid,
          int vertexid,
          int varid);
```

Figure 3. A list of APIs for constructing graph objects.

such as a tree, is not supported in the current format. In order to augment PnetCDF with such capability, our new metadata scheme uses the graph representation to describe the data models.

A. New data objects

We add graphs as new data objects in PnetCDF. A graph $G = (V, E)$ consists of vertices V and directed edges E [22]. A vertex, also a new data object, is a container of variables and edges. Similar to the HDF5 group objects, vertices can contain multiple netCDF variables and have directed or undirected links to other vertices. An edge is also a new data object that connects two vertices as directed or undirected link. Edges can have associated real number weights. Figure 2 is just to illustrate a data model can have an arbitrary relationship among the data objects. Through the graph representation, netCDF variables can be grouped together and linked to different variables. Therefore, complex data models, such as trees, can be described. This paper proposes to store the relationship information in PnetCDF in a more systematic way.

B. Metadata format

Because the netCDF file format does not have the flexibility to add new user-defined data objects, we create a metadata format, shown in Figure 4, for describing the new data objects and their relationships. This metadata is saved as a regular netCDF non-record variable in text data type, which is treated like the file header. The new metadata format includes the information about graph, vertex, and edge objects, such as the number of these objects, their individual attributes, such as directions and weights for the edges, the number of vertices, and their connected relationships. A vertex's metadata contains a list of variable IDs, a list of edge IDs, and the corresponding vertex IDs connected by the edges. The strategy of saving the new metadata in a regular netCDF variable ensures the conformation of the conventional netCDF file format and hence the files created with this new data model feature can still be accessed through the netCDF and PnetCDF programs.

The graph idea proposed in this paper is to describe the relationships among multiple data objects (eg. variables). One netCDF file can have more than one graph. For example, two graph objects can be defined and the same variable can be added to both of the graphs.

C. Application programming interfaces

We add several PnetCDF APIs for accessing the new data objects. Figure 3 lists the new APIs for creating graph objects, connecting edges between vertices, and adding variables into vertices. The APIs for deleting or disassociating objects are similar, but not shown in the figure. The APIs that change the graph structures are required to be in the define mode. We also make the following requirements:

1) A vertex must be associated to at least one graph;
2) A netCDF variable can be added to one or more vertices, but the association is not required;
3) The same edge can be used to connect two different pairs of vertices.

All the new APIs are non-collective and allow their arguments to contain different values when used by different processes. However, if it is intended to define the same objects by a group of processes, then the defining processes must agree on the arguments. When the exiting the define mode is called (i.e. at the call to `ncmpi_enddef()`), the metadata consistency will be performed to check and merge the newly created objects. We use the object names as unique keys to tell if a group of defined calls is for the same object or not. Hence, defining the same objects for different processes the attributes must have the same argument values. Once the objects are merged, the header and graph metadata are duplicated in memories across all the processes.

IV. Implementation consideration for scalable I/O

Although high-level I/O libraries have made an important contribution to supporting large parallel applications, they do not always attain a high percentage of peak performance when used in these applications. Some of the performance issue reasons have been that (1) the underlying models, formats, and APIs in storage software did not explicitly

```
graph          = graph_magic graph_att_list edge_list vertex_list
graph_magic    = 'G' 'R' 'A' 'P' 'H' graph_type
graph_type     = \x01      //direction graph
                 \x02      //undirection graph
graph_att_list = att_list  //graph attribute
att_list       = ABSENT | NC_ATTRIBUTE nelems [attr ...]
edge_list      = ABSENT | NC_EDGE nelems [edge ...]
vertex_list    = ABSENT | NC_VERTEX nelems [vertex ...]
NC_ATTRIBUT    = \x00 \x00 \x00 \x0C //tag for list of attributes
NC_EDGE        = \x00 \x00 \x00 \x0D //tag for list of edges
NC_VERTEX      = \x00 \x00 \x00 \x0E //tag for list of vertexes
nelems         = NON_NEG             //nmber of elements in following sequence
edge           = name edge_weight
edge_weight    = name varid_list edgid_list vertex_att_list vertex_size begin
varid_list     = nelems [varid ...]
varid          = NON_NEG             //variable ID for graph data model
edgeid_list    = nelems [edgeid pair ...]
edgeid_pair    = edgeid adj_vertexid
edgeid         = NON_NEG             //edge ID for graph data model
vertex_id      = NON_NEG             //vertex ID for graph data model
vertex_att_list= att_list
vertex_size    = NON+NEG             //vertex size. The amount of space in Z bytes
                                     //allocated to the vertex
begin          = OFFSET              //vertex start location
```

Figure 4. Metadata format for graph representation.

consider parallelism in their original designs, (2) the parallelism subsequently introduced has been incremental and has often been driven to work around limitations of interfaces and underlying software (e.g., working around a specific file system performance bug), and (3) many more applications require, for scalability and algorithmic reasons, much more sophisticated data structures than those incorporated in the original designs (e.g., adaptive meshes, irregular datasets). Given the irregular data distribution patterns exhibited in many data models, we focus on the issues related to keeping high I/O parallelism.

1) Metadata I/O: All define mode functions are collective I/O mode in current PnetCDF and all processes in the same communicator must call them with the same argument values. While the consistency semantics of file systems are well-defined and strict, consistency semantics in high-level libraries have often been underspecified, leading to confusion on the part of users. These consistency semantics for high-level libraries are not suitable for an irregular data set, because the underlying data distribution is highly irregular and dynamic. Process synchronization in the existing parallel I/O methods can penalize the I/O parallelism if the process collaboration is not carefully coordinated. Due to the irregularity introduced by the some scientific applications, the I/O workload can vary widely depending on computational region of interest during the runtime. The intensive and dynamic I/O behavior presented by scientific applications imposes a big challenge to achieve scalable and sustainable I/O performance when applications run on a large scale of HPC systems.

Based on the understanding of the underlying data models, we relax the collectiveness requirement for the metadata I/O. Different processes can concurrently define different objects, such as variables, edges and vertices. At the end of the define mode, the root process collects metadata from all the processes and merge them into a consistent form. The advantage of this strategy is allowing processes to keep only the related data objects. It is not necessary for a process to know the metadata of those objects it never accesses. However if the amount of the metadata is large, having the root perform the metadata merge and check the consistency may create a performance bottleneck. We plan to investigate new approaches to overcome this potential problem.

2) Data I/O: Many I/O optimizations have been developed over the years including: two-phase I/O; data sieving; collective I/O with caching; etc. These optimizations have been incorporated into layers such as MPI-IO and have been shown to work in some cases, but fall short in many others, particularly in cases that require very irregular accesses, which may involve adaptive structures etc. One of most important goals is providing scalable I/O performance for the various complex structures and layouts. PnetCDF relies on MPI-IO for portability and for providing high performance, but MPI-IO alone is not efficient or flexible enough for handling the I/O requirements from the various data models discussed. Our preliminary results in combining multiple I/O requests in PnetCDF are encouraging [23] in that significant performance improvement was obtained over the traditional APIs. The combination is hidden under the non-blocking I/O interfaces, enabling data aggregation to achieve high I/O bandwidths. Combining many I/O operations into fewer, larger I/O operations has been a fundamental component of parallel I/O optimization for decades. Multiple benefits come with non-blocking I/O interfaces, as follows. First, it is implemented with standard MPI-IO library (ROMIO [24]),

which provides two-phase collective I/O to aggregate individual small I/O requests. Second, the non-blocking feature allows the I/O function calls to return right away so that computational process can proceed while the I/O subsystem is consuming all the I/O requests. Blocking I/O processes each logical write request one after another sequentially. Non-blocking I/O requests can be fired all at once. This essentially reduces the synchronization overhead among all the processes when all the issued logical writes can be aggregated not only among all the processes but also along programs execution order. Third, by spanning the I/O over the entire evolutional interval, the stress imposed on I/O systems will be reduced. Last but not the least, the total run time of applications will be reduced.

For data models exhibiting an irregular data distribution, PnetCDF non-blocking I/O is the best choice for obtaining high performance. The non-blocking APIs are non-collective, which allows processes to make different numbers of the calls for different variables. At the end, a collective wait call aggregates the posted requests and carries out the I/O using a single MPI collective I/O call.

V. Performance Improvement and Evaluation

To illustrate the irregular distribution nature of many data models that can be handled efficiently by our new scheme, we use Chombo I/O [29] as an example for performance evaluation. It is a representative example that has a hierarchical tree relationship among all data objects (multi-dimensional arrays).

A. Chombo I/O

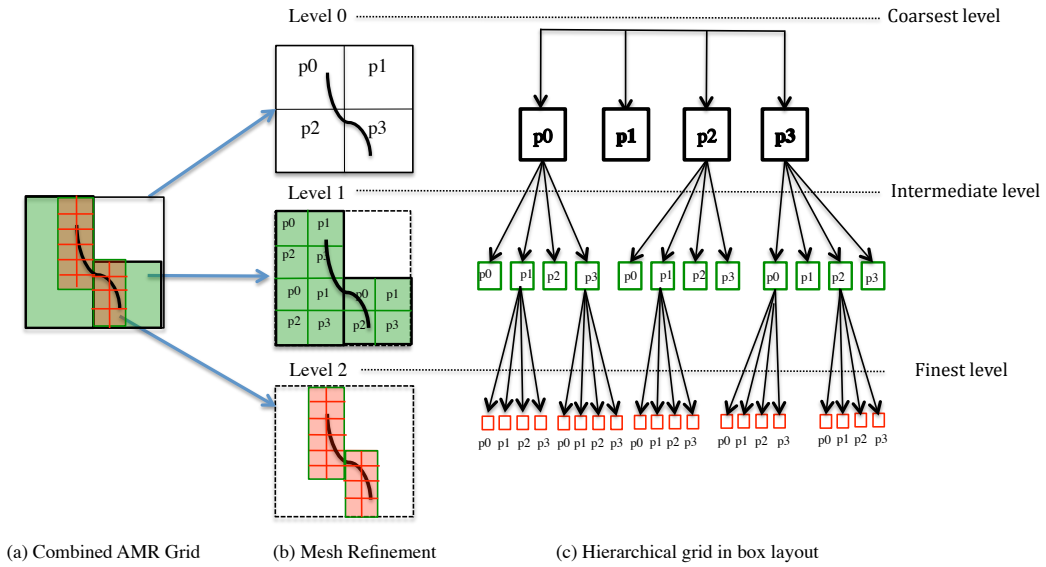
Chombo [30] developed at Lawrence Berkeley National Lab (LBNL) is a library for block-structured Adaptive mesh refinement (AMR) [31] applications. The AMR methodology has been successfully applied to numerical modeling of various physical problems that exhibit multi-scale behavior, such as those mentioned in [32]. The idea of AMR is quite straightforward, that is, to apply finer discretization only at places where higher resolution is needed. However, the simplicity of finite difference calculation on a uniform grid is a trade-off in AMR, where the irregularity comes from the boundaries between grids introduced by local mesh refinement. Although designed primarily for finite differences and computational fluid dynamics, Chombo can be used in other areas of computational science and engineering. By providing sufficient C++ abstract data types, Chombo enables user to build and manage the grid evolutions, decompose the computational data on the evolved grid, as well as process the I/O operations in either sequential or parallel mode. The Chombo I/O benchmark is derived from the framework mentioned above [29]. It creates simulated Chombo data structures and writes them to a single file using the HDF5 high-level I/O library [7]. Chombo accesses a variety of small auxiliary files at run time.

Typically, a structured AMR simulation starts from a uniform mesh that covers the entire computational domain. In order to compute or study the object (the black curve line as shown in Figure 5(a)), a union of patches (the shaded region) needs to be identified based on level 0's resolution and a finer mesh can be applied to this localized region of interest. This refinement process is repeated for the next level until the termination condition is satisfied. As a result, AMR simulation produces a dynamic, multi-level grid hierarchy with increasing refinement resolution. By overlying all the individual structured patches in the hierarchy, an adaptive refined mesh is formed as Figure 5(b). Since a patch can be further decomposed into boxes in Chombo framework, the union of patches at each level is reduced to a set of boxes on different processes, as shown in Figure 5(c), the number of boxes are changing between levels and the individual size for each box can be different at the same level or between levels.

B. Performance Improvement

One possible optimization is to use MPI derived data type to combine all the sequential I/O requests in the for-loop and replace it with a single logic I/O request. Another feasible optimization is to adopt collective I/O function calls. Note that the number of for-loop iterations on each process may not be the same, which results in some processes having more I/O requests than the other processes, thus, collective I/O would not work unless all the processes participate in each I/O request or all processes have the same number of I/O requests. The issue of varying number of I/O requests can be quickly solved by simply retrieving the maximum number of iterations across all the processes and using it as the for-loops termination variable, once the iteration index exceeds the number of boxes owned by a process.

It is possible that there are multiple components contained in a Box, since they are all serialized into a 1-D buffer, only a single logical I/O operation is required to write out this entire buffer. Fig. 6 illustrates an example grid of a particular level decomposed into 8 individual MPI processes, each holding the boxes in space-filling curve order. However, the boxes are organized in lexicographic order, in the final file layout. Therefore, a process will write multiple portions of data at noncontiguous regions in file. In the worst case scenario, the number of physical file seek and write requests issued by a process would be the same as the number of the boxes assigned for that process. The physical I/O requests can be further reduced by simply appending the list of boxes from each process in the order of the MPI process rank, which might cause applications analysis and visualization tools to rewrite their I/O interface in order to adopt the new file layout. We decided to leave this optional optimization to application scientists and look for more general solutions. Figure 6 only demonstrates a simple I/O access pattern, in reality, the file offsets and access sizes for



A 2-D patch-based hierarchical grid with three levels of refinement on 4 processes

Figure 5. An example of hierarchical grid in Chombo

each I/O request vary widely per evolution because of the irregular grid shape developed by AMR refinement scheme. Thus, it would be very difficult if not impossible to develop generalize solutions other than collective I/O by utilizing underlying specific but dynamically changeable I/O access patterns. This forces us to resort to some other I/O traits. Having observed that a burst of intensive I/O activity is presented after a period of CPU processing and it is repeated at certain intervals over the whole simulation process, we believe the relaxing the stress on I/O subsystem imposed by this high peak I/O bandwidth requirement would be one possible solution. One approach to reduce the stress on the I/O subsystem is to use non-blocking strategy, which allows the I/O operation to span over the computation interval until the next I/O request collection occurs. The idea of using the non-blocking I/O is because the traditional methods used by PnetCDF or HDF5 for accessing metadata and data are blocking I/O and per variable basis. Our implementation of non-blocking I/O is actually deferring the I/O, so multiple I/O requests can be aggregated into larger requests for better performance. Therefore, we believe the non-blocking I/O approach would performance at least as well as the blocking ones.

Although HDF5 actually uses the MPI IO library to implement parallel file access, Chombo I/O only adopts an independent mode, where I/O operations among processes are totally independent of each other. This implementation is also blocking, where the next iteration cannot proceed until the write or read operation in the current iteration has been finished and returned. The Chombo I/O benchmark abstracts all the I/O operations involved in simulation and leaves out the complex and time consuming computational component. Since its current version only has HDF5 I/O method, we added the PnetCDF method for the evaluation purpose. Two I/O methods are implemented, one using blocking APIs and the other using non-blocking APIs. The blocking I/O method mimics the approach used by the HDF5 method. The example grid

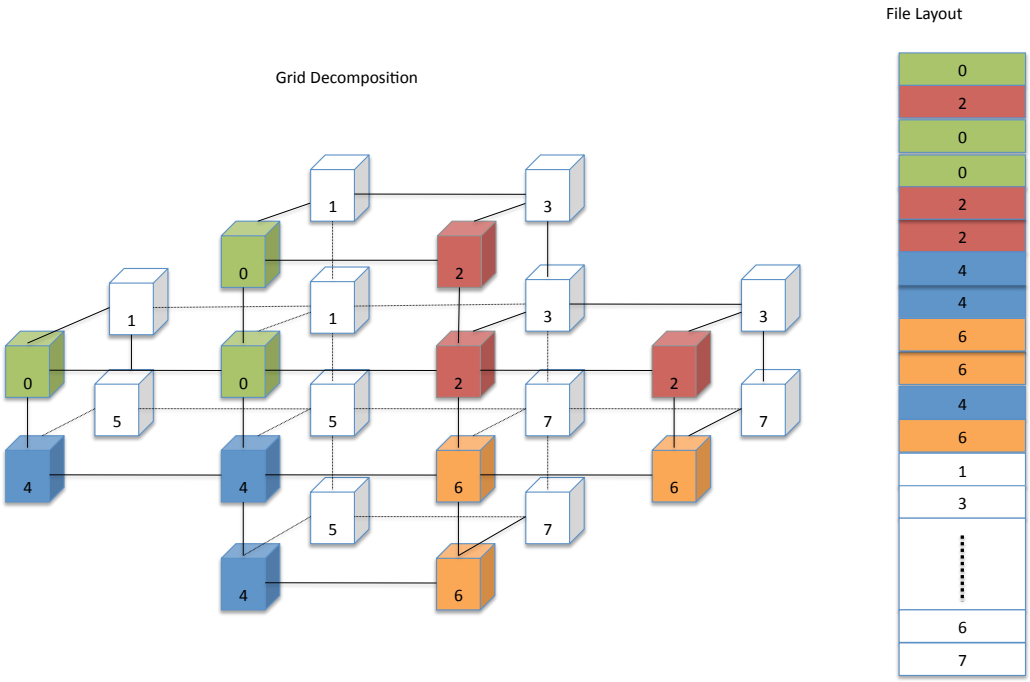


Figure 6. I/O patterns in Chombo

file, `s64x64x64L2r4i80b8-32o0f0.00100p1.abr`, provided by the benchmark is modified in order to run large numbers of processes. We keep the I/O amount proportional to the number of processes.

C. Experiments

Our experiments were conducted on Franklin [25], the Cray XT4 parallel machine at the National Energy Research Scientific Computing Center and Surveyor [26], the IBM Blue Gene/P system at Argonne National Laboratory. Franklin is a 9660-node SuSE Linux cluster where each compute node consists of a 2.3 GHz single socket, quad-core AMD Opteron processor with a theoretical peak performance of 9.2 GFlop/sec per core. Each compute node has 8 GBytes of memory. The parallel file system is Lustre [27] with 48 I/O servers (OSTs). The measured peak write performance on Franklin is 16 GB per second. Surveyor is a 4,096-core IBM Blue Gene/P system. The parallel file system is GPFS [28] with four file servers.

We conducted a series of experiments with up to 8192 MPI processes on Franklin. In order to maximize the possible I/O bandwidth, we use all 48 stripe counts (the total number of OSTs available on Franklin). The stripe size is set to 1 MB. Figure 7 shows the aggregated I/O bandwidth for HDF5 independent write, PnetCDF blocking, and non-blocking write, respectively. The independent blocking I/O for both PnetCDF and HDF5 performs poorly performance and does not scale at all. The maximum write bandwidth achieved was only 2GB/s. By utilizing the new metadata scheme, the PnetCDFs non-blocking I/O method dramatically improves the aggregated write bandwidth and sustains strong scalability. The best aggregate write bandwidth that our implementation achieved was a little more than 12 GB/s for 8192 processes on Franklin, very close to its measured peak performance. Similar scalabilities are observed on the Surveyor.

VI. Conclusions

The NetCDF file format can effectively store and describe regular multi-dimensional datasets, but it does not address the full range of current and future computational science data models. There is no existing parallel I/O library that supports complex data models, i.e. arbitrary data relationships among data objects. Only HDF5 supports a

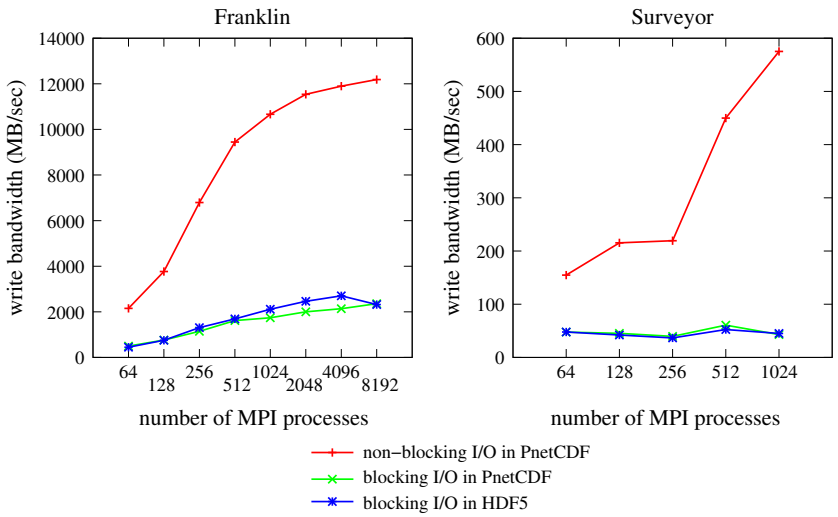


Figure 7. Performance results of Chombo I/O benchmark

tree-structured model through group, but not for arbitrary graphs. In this paper, we propose a storage mechanism in PnetCDF to support a broad variety of data models used in computational scientific applications. Our design principle for parallel construction and access the data models focuses on enabling metadata I/O parallelism as well as eliminating process synchronization. The performance results from using the Chombo I/O benchmark with irregularly distributed data objects, demonstrate that our approaches can achieve scalable I/O performance for parallel data and metadata creation and access. There are many other data models that post different challenges for finding efficient and effective mechanisms to provide scalable I/O performance in PnetCDF. Our future work includes designing a scalable method for merging parallel-defined metadata and performance evaluations using more application I/O kernels.

ACKNOWLEDGMENT

This work is supported in part by NSF award numbers: OCI-0724599, CNS-0830927, CCF-0621443, CCF-0833131, CCF-0938000, CCF-1029166, and CCF-1043085 and in part by DOE grants DE-FC02-07ER25808, DE-FG02-08ER25848, DE-SC0001283, DE-SC0005309, and DE-SC0005340. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams, and K.A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, University of California, Berkeley, December 2006.

[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J.D. Kubiatowicz, E.A. Lee, N. Morgan, G. Necula, D.A. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K.A. Yelick. The parallel computing laboratory at U.C. Berkeley: A research agenda based on the Berkeley view. Technical Report UCB/EECS-2008-23, University of California, Berkeley, March 2008.

[3] R. Rew and G. Davis. The Unidata netCDF: Softwarefor Scientific Data Access. Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, February 1990.

[4] R. Rew, G. Davis, S. Emmerson and H. Davies. NetCDF Users Guide for C. Unidata Program Center, June 1997. [Online]. Available: http://www.unidata.ucar.edu/packages/netcdf/guidec/.

[5] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A high-performance scientific I/O interface. In Proceedings of SC2003: High Performance Networking and Computing, Phoenix, AZ, November 2003. IEEE Computer Society Press.

[6] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. 1997. [Online]. Available: http://www.mpi-forum.org/docs/docs.html.

[7] HDF5 Home Page. The National Center for Supercomputing Applications. [Online]. Available: http://hdf.ncsa.uiuc.edu/HDF5/.

[8] Mark Howison, Andreas Adelmann, E. Wes Bethel, Achim Gsell, Benedikt Oswald and Prabhat. H5hut: A High-Performance I/O Library for Particle-based Simulations. The

Workshop on Interfaces and abstractions for Scientific Data Storage, September 2010.

[9] W. Benger, A. Hamilton, M. Folk, Q. Koziol, S. Su, E. Schnetter, M. Ritter, and G. Ritter, Using Geometric Algebra for Navigation in Riemannian and Hard Disc Space. Proceedings of Computer Graphics, Computer Vision and Mathematics, 2008.

[10] Silo: A mesh and field I/O library and scientific database. https://wci.llnl.gov/codes/silo.

[11] HDF5 Tutorial. [Online]. Available: http://www.hdfgroup.org/HDF5/Tutor.

[12] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. 1995. [Online]. Available: http://www.mpiforum.org/docs/docs.html.

[13] R. Thakur and A. Choudhary. An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays. Journal of Scientific Programming, 5(4):301, Winter 1996.

[14] R. Thakur, W. Gropp and E. Lusk. Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, October 1997.

[15] D. Kotz. Disk-directed I/O for MIMD Multiprocessors. ACM Transactions on Computer Systems, 1997. 15(1): p. 41-74.

[16] K. Seamons, Y. Chen, P. Jones, P.; J. Jozwiak, M. Winslett. Server-directed Collective I/O in Panda. In Supercomputing Conference. 1995.

[17] K. Coloma, A. Ching, A. Choudhary and W. Liao. A New Flexible MPI Collective I/O Implementation. in IEEE International Conference on Cluster Computing. 2006. Barcelona, Spain.

[18] K. Coloma, A. Choudhary, W. Liao, L. Ward, E. Russell, and N. Pundit. Scalable High-level Caching for Parallel I/O. International Parallel and Distributed Processing Symposium. 2004: New Mexico.

[19] X. Ma, M. Winslett, J. Lee and S. Yu. Improving MPI-IO Output Performance with Active Buffering Plus Threads. in the International Parallel and Distributed Processing Symposium. 2003.

[20] W. Liao, A. Ching, K. Coloma, A. Nisar, A. Choudhary, J. Chen, R. Sankaran and S. Klasky. Using MPI File Caching to Improve Parallel Write Performance for Large-Scale Scientific Applications. in International Conference for High Performance Computing, Networking, Storage and Analysis. 2007. Reno, Nevada.

[21] W. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and S. Tideman. Collective Caching: Application-aware Client-side File Caching. in 14th IEEE International Symposium on High Performance Distributed Computing. 2005. Research Triangle Park, NC.

[22] T. Cormen, C. Leiserson, R. Rivest and C. Stein. Introduction to Algorithms. Second Edition. MIT Press.

[23] K. Gao, W. Liao, A. Choudhary, R. Ross, and R. Latham. Combining I/O Operations for Multiple Array Variables in Parallel netCDF. In the Workshop on Interfaces and Architectures for Scientific Data Storage, September 2009.

[24] R. Thakur, W. Gropp and E. Lusk. Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, October 1997.

[25] Franklin, the Cray XT4 parallel computer at National Energy Research Scientific Computing Center. http://www.nersc.gov/nusers/resources/franklin/.

[26] Surveyor, the IBM Blue Gene/P system at Argonne National Laboratory. https://www.alcf.anl.gov/resources/storage.php.

[27] Cluster File System, Inc. Lustre: A Scalable, High Performance File System. http://www.Lustre.org/docs.html.

[28] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proceedings of the File and Storage Technologies (FAST 02), pp. 231-244, Jan. 2002.

[29] Chombo I/O benchmark. [Online]. Available: http://www.nersc.gov/∼ndk /ChomboBenchmarks/ chomboIOBenchmark.html.

[30] P. Colella, D. T. Graves, T. J. Ligocki, D. F. Martin, D. Modiano, D. B. Serafini, B. Van Straalen. Chombo software package for AMR applications design document. Applied Numerical Algorithms Group, NERSC Division, Lawrence Berkeley National Laboratory, Berkeley, California, September 12, 2003. [Online]. Available: http://seesar.lbl.gov/anag/chombo/.

[31] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. Journal of Computational Physics, Vol. 53, p.484, 1984.

[32] M. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydodynamics. Journal of Computational Physics, Vol. 82, No. 1, pp. 64-84, May 1989.

[33] G. Bryan. Fluids in the Universe: Adaptive Mesh Refinement in Cosmology. In Computing in Science and Engineering, 1(2):46-53, March/April, 1999.

[34] Wen, T, Su, J, and et. al. An Adaptive Mesh Refinement Benchmark for Modern Parallel Programming Languages. Proceedings of Supercomputing '07, November 10-16, 2007. Reno, Nevada, USA.

[35] E. L. Miller, and R. H. Katz. Input/Output Behavior of Supercomputing Applications. Proceedings of Supercomputing 91, November 1991.