Querying Large Scientific Data Sets with Adaptable IO System ADIOS

Junmin Gu[1], Scott Klasky[2], Norbert Podhorszki[2], Ji Qiang[1], and Kesheng Wu[1(✉)]

[1] Lawrence Berkeley National Laboratory (LBNL), Berkeley, USA
kwu@lbl.gov

[2] Oak Ridge National Laboratory (ORNL), Oak Ridge, USA

Abstract. When working with a large dataset, a relatively small fraction of data records are of interest in each analysis operation. For example, while examining a billion-particle dataset from an accelerator model, the scientists might focus on a few thousand fastest particles, or on the particle farthest from the beam center. In general, this type of selective data access is challenging because the selected data records could be anywhere in the dataset and require a significant amount of time to locate and retrieve. In this paper, we report our experience of addressing this data access challenge with the Adaptable IO System ADIOS. More specifically, we design a query interface for ADIOS to allow arbitrary combinations of range conditions on known variables, implement a number of different mechanisms for resolving these selection conditions, and devise strategies to reduce the time needed to retrieve the scattered data records. In many cases, the query mechanism can retrieve the selected data records orders of magnitude faster than the brute-force approach.

Our work relies heavily on the *in situ* data processing feature of ADIOS to allow user functions to be executed in the data transport pipeline. This feature allows us to build indexes for efficient query processing, and to perform other intricate analyses while the data is in memory.

1 Introduction

Modern scientific experiments such as large accelerators rely heavily on high-performance simulations for design, calibration and data analysis [13,24]. These simulation programs typically need to read and write a vast amount of data, for example to read the definition of the complex geometry of an accelerator design, to checkpoint the state of the simulation, and to produce analysis output [23]. The output from these simulations is used to understand the experimental observations and to guide the next experiment. Often, the critical information is only a small fraction of a large data collection. Reading and writing the necessary data records efficiently is the challenge we address in this work.

R. Yokota and W. Wu (Eds.): SCFA 2018, LNCS 10776, pp. 51–69, 2018.
https://doi.org/10.1007/978-3-319-69953-0_4

Current high-performance computing systems are typically massively parallel platforms consisting of millions of CPU cores and thousands of secondary storage devices. A significant amount of programming effort is needed to make effective uses of such a system. Programmers have to make difficult choices among available options. Take the checkpointing task as an example, an efficient solution to write a large amount data is to have each process of the simulation program write its own file. However, this option can create millions of files when millions of processes are used. On some file systems, an attempt to list these files will slow the metadata servers to a crawl, even crash the whole file system. Usually, the checkpoint files are also used for data analyses. During a typical analysis operation, only a fraction of the data records are needed for a specific analysis operation. Locating these data records from a petabyte data collection is a challenging task requiring auxiliary information such as indexes. It may be necessary to restart the simulation program with a different number of processes, for example, to reduce the computation time or to increase the fidelity of the simulation. When there is one checkpoint file per process, these checkpoint files may have to be combined in complex ways in order to restart the simulation properly. For these reasons, many researchers have explored options to simplify the IO operations for large simulation programs.

The process of locating and retrieving these selected records is typically through a query interface. The best known querying tools do not support scientific data, which is typically stored in large data files as numbers. For example, internet search engines are widely used by primarily designed to process text documents; and database systems could work on both numerical values and text string, but require the data to be under its full control. Scientific projects usually does not have the budget to pay for the extra storage and software license fee for database systems. The alternative we pursue is to add query interface to high-level IO libraries.

A number of high-level IO libraries are in wide use, the most commonly used are ADIOS [16], HDF5 [10], and netCDF [22]. Among these three popular libraries, netCDF is primarily used by the climate community and it is in the process of switching to use HDF5 as the backend storage layer. For this work, we primarily consider choosing between HDF5 and ADIOS. Both ADIOS and HDF5 are used in a variety of large scientific applications, and the authors have first-hand experience with both [5,7,16,26]. In this particular case, we plan to use ADIOS because it offers a distinctive feature that is not available in any other IO libraries. That is, ADIOS supports *in situ* processing, which allows us to build the indexes while generating the data files. This has a distinct advantage of as soon as the data files are available, the associated indexes are also available. This should make it much more efficient to select a relatively small number of critical data records from a large data collection. In addition, the *in situ* processing capability would also enable dynamic analysis capability to improve the usefulness of a simulation program. Thus, the core of this work is to develop a query interface for ADIOS.

The key contributions of this work are as follows:

- Design and develop a query interface for ADIOS.
- Evaluate strategies to minimize the time needed to retrieve scatter data records from a ADIOS file.
- Exercise the query interface with a large application dataset.
- Introduce *in situ* processing feature to an application that does not yet have this feature, and demonstrate the usefulness of this feature.
- Measure the performance of ADIOS in completing the checkpoint operations.

2 Related Work

In commercial applications, large datasets are typically managed by a data management system [15,18,25]. These systems take control of the user data and provide high-level languages for analysis tasks. In contrast, most scientific projects store their data as files and use the file systems as the primary tools for data management [24]. This file-based approach allows users full control of their data and their analysis tasks; however, it also requires the users to spend much more time to manage their data than using a data management system. In this work, we combine a number of techniques to reduce the data management time, more specifically the time to select a relatively small fraction of the data records. In this section, we briefly review the key technologies involved.

2.1 High-Level IO Libraries

Scientific datasets are frequently organized as multidimensional arrays and the commonly used IO libraries are designed to store and organize these arrays. Earlier we mentioned three: HDF5 [10], netCDF [22] and ADIOS [16]. Next, we provide a brief description for each of them.

HDF5 is a short-hand for Hierarchical Data Format version 5[1] [10,11]. It is highly flexible, efficient, portable and extensible. Many application domains have developed their own data organization standards based on HDF5 [10]. The recent releases of HDF5 software exposes the Virtual Object Layer (VOL) to make it even easier to extend the functionality of the software library, such as to provide query services [8] and to work with Burst Buffers [9].

NetCDF is a short-hand for network common data format[2] [22]. It is widely in the climate modeling community, with many petabytes of data stored in this format. Some of the largest collections are used to compile the assessment reports commissioned by the Intergovernmental Panel on Climate Change[3].

ADIOS is a short-hand for Adaptable IO System[4] [16,17]. First released in 2008 [17], it is a new library among the commonly used scientific formats. However, it has attracted much attention because its simplicity and efficiency [16].

[1] HDF5 software is available at https://support.hdfgroup.org/HDF5/.
[2] NetCDF software is available at https://www.unidata.ucar.edu/software/netcdf/.
[3] The most recent IPCC report AR5 is available at https://www.ipcc.ch/report/ar5/.
[4] Software available at https://www.olcf.ornl.gov/center-projects/adios/.

For example, it accepts an XML configuration file for users to describe the variables, their types, and the path to take from memory to disk. This capability allows the users to change how they process the data without changing the simulation program. This approach gives a level of adaptability that no other IO system could match. A special feature created by this flexibility is the *in situ* processing capability to be described next. To effectively support the querying capability over ADIOS files, we also utilize this *in situ* capability to create indexes, which reduces the effort required to generate indexes and ensures the indexes are available as soon as the data is available.

2.2 In Situ Processing

Writing to disk is generally much slower than writing the same data to memory, therefore, it is highly desirable to perform as many analysis operations as possible while the data is still in memory. This type of *in situ* processing also makes it possible to produce analysis results without storing the original data. On a HPC system, the IO operations typically need to pass data among the compute nodes, IO nodes, and disk systems. While the data is being transferred among these subsystems, it is possible to perform a considerable amount of analysis operations. ADIOS supports these options by separating API for data producers from that for data consumers.

A data producer outputs the data following a set of API styled after the familiar POSIX write interface. The content generated by the producer is sent to the downstream processing code by the ADIOS transport system based on the instructions provided by the user. The consumers of the data could be located on the same CPU as the producer, or elsewhere on the network. The ADIOS transport system will schedule the data movement in a reliable manner [17].

The benefits of *in situ* processing is widely recognized for large scientific simulations and a number of efforts are under way to develop alternatives to ADIOS. Bauer et al. have produced an excellent review of the state of art in 2016 [1]. As of this writing, ADIOS is the most mature system for *in situ* processing and has extensive IO capability, therefore, we have selected to use ADIOS for this work.

2.3 Querying and Indexing

A data management system such as a DBMS typically provides a query interface for accessing the data under its management. In contrast, the data access functions for a file generally need to follow the structure of the file, such as, move file pointer to a location and read the next 40 bytes. This structure-based access functions require the user to know more details about the file organization than most application users are familiar with, and therefore, not as user-friendly as the query-based data access methods. A query on a scientific dataset might be "finding all data records from collection A where temperature and pressure are in the specified ranges." In this example, the user only needs to know the name of the variables and quantities of interest, thus a query interface is much easier to use for an application scientist.

To effectively support the queries, the system needs to create indexes [12], such as, B-Tree [6], bitmap index [26], and hashing [28]. Because the scientific data collections are typically analyzed without modification (or with infrequently modifications), we plan to concentrate on indexing techniques that are designed for query-intensive workloads. The queries on scientific data typically returns a number of data records instead of a single record. Additionally, the users often explore a large variety of combinations of query conditions. From research literature, we see that bitmap indexes satisfy these requirements. To support newly designed ADIOS query interface, we choose to use an open source bitmap index library named FastBit[5] [26]. At the same time, we are also exploring additional indexing techniques that might be better suited for ADIOS [27].

2.4 Application Use Cases

In this work, we use a couple of large scientific applications to illustrate the functionality we are developing. Next, we give a brief description of IMPACT and S3D. They are selected as examples of large scientific simulation programs. These large simulations produce a large amount of data and require complex data analyses, where ADIOS indexing and querying capability could play important roles.

IMPACT[6] is a parallel particle-in-cell code designed to model the dynamics of multiple charged particle beams in accelerators. This program uses longitudinal position (z) as an independent variable and includes the effects of externally applied fields from magnets and accelerating cavities as well as the effect of self-fields (as space charge fields). It is written in Fortran 90 with MPI for interprocess communication. It has been applied to studies of halo formation and coupling resonance in high intensity beams, microbunching instability in high brightness electron linac, beam dynamics in SNS linac, JARPC linac, RIA driver linac, CERN superconducting linac, LEDA halo experiment, Proton Synchrotron at CERN, and so on [19–21]. In this work, we primarily use IMPACT to exercise the *in situ* processing capability because it is a capability not available in IMPACT yet.

S3D is a high-performance direct numerical simulation (DNS) of combustion with detailed chemistry [4,14]. It is designed to study the interaction between turbulence and combustion chemistry. It is extensively used to understand the flame characteristics of lean mixture flame in the next generation of diesel and alternative fuel engines, as well as the flame stability features in large industrial burners such as those for gas-fired power plants. A large run of this code typically divides its simulation domain into billions of cells and then follow the evolution of the combustion process for many thousands of time steps. The checkpoint files and the *in situ* analysis output could easily be many terabytes per run [4]. Since extensive work on *in situ* processing has been performed before with S3D, in this work we primarily use a set of S3D data to test the query processing capability to be developed.

[5] FastBit software is available at http://sdm.lbl.gov/fastbit/.
[6] IMPACT Software available at http://amac.lbl.gov/~jiqiang/IMPACT/.

3 ADIOS Overview

In the next three sections, we describe our work on ADIOS to address various IO challenges. We start with the basic bulk IO operation of checkpointing, then move on to *in situ* processing and querying in the next two sections.

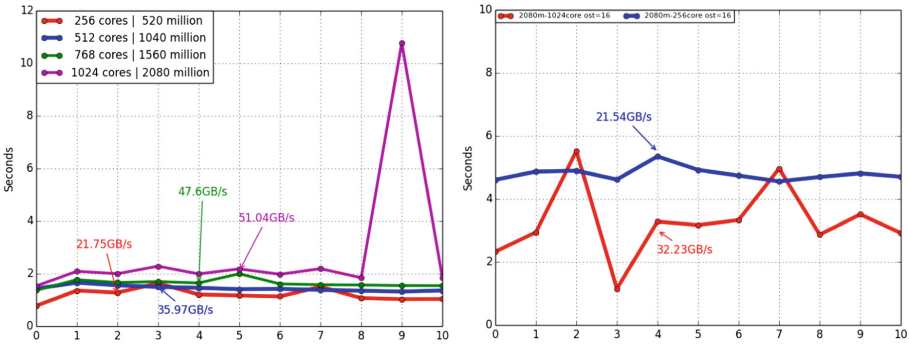
ADIOS is known for its simple API and high performance. The core insight guiding the ADIOS design is to separate the description of IO operations from the IO strategies employed for the actual lower level operations. This allows the application programming interface (API) to only describe what variables to read or write, while leaving the responsibility of selecting the actual transport operations to the ADIOS system. In particular, ADIOS has implemented a variety of transport mechanisms [16]. Its ability to seamlessly select the best transport mechanism is also at the root of its support for *in situ* operations. Other important factors contributing the high-performance include log-based file format, buffered writing, subfiling, asynchronous transport operations, and so on [16].

ADIOS was designed in 2005 to reduce the IO time for a number of mission critical applications [17]. Since then, ADIOS has been the leading software system for *in situ* data processing on many of largest high-performance computers. Some of the early success stories include improving the IO rate of S3D checkpoint operation by more than a factor of 10 from 3 GB/s to over 30 GB/s [16]. The developers of ADIOS have published a number of studies showing the dramatic improvement of IO performance for various applications. Next, we add our experience with the IMPACT code.

IMPACT employs the particle-in-cell paradigm to model the dynamics of particles. Each particle has immutable properties such as rest mass and charge, and dynamic properties such as position and momentum, recorded as x, y, z, p_x, p_y, and p_z. IMPACT produces two types of output for analyses: checkpoint files and particle statistics. We describe our work on checkpointing in this section and the work on utilizing *in situ* processing to accelerate the production of particle statistics in the next section.

Because the particles on each processor are independent from other particles, IMPACT produces its checkpoint files by writing one file per processor. This option has the advantage of minimizing the coordinate needed among the processors and could significantly reduce the time spent on IO operations.

ADIOS offers a variety of IO options and the parallel file system (Lustre) additionally offers a number of file system parameters; all of these parameters could affect IO performance. Instead of providing an exhaustive exploration of these parameters, in Fig. 1, we provide one set of performance measurements to show that ADIOS is able to support very efficient IO operations. This particular set of measurements was collected on Edison at NERSC. The measurements are conducted on Lustre file system with 24 OST and a peak IO rate of 168 GB/s. To avoid contention with other active jobs, we only used 16 OST for each ADIOS file, which have a nominal peak IO rate of about 112 GB/s.



(a) time (seconds) to write checkpoint files (b) time (seconds) to read checkpoint files

Fig. 1. Time to read and write checkpoint files with ADIOS.

The write tests were performed with a fixed number of particles on each MPI process. The reported IO rates in Fig. 1 are computed using the median observed IO time. The write operations reported in Fig. 1(a) all uses 16 OST and uses about 2 million particles per MPI process. Up to 1024 processes, the average write speeds rises to over 50 GB/s.

In Fig. 1(b), we reported the observed performance of reading the different checkpoint files. Clearly, the number of OST used to store the files has a strong influence on the observed read performance. One important feature we want to demonstrate is the reading of the same checkpoint files with a different number of processes. In this particular case, reading the same file with different number of processes took about the same amount of time and producing about the same aggregate IO speed.

4 *In Situ* Indexing

The checkpoint files capture the position and momentum of each particle periodically, but infrequently. To capture more dynamic behavior of the particles, IMPACT also compute high-level statistics about the particles at a much higher frequency. However, these statistics are programmed by the developers of IMPACT code and is difficult for the end users to modify to suit their own needs. The *in situ* processing capability of ADIOS is a flexible mechanism to introduce these custom statistics. It can also be used to provide asynchronous computation including index building, without blocking the main simulation computation. Next we describe a simple test to compute histograms at every simulation step to demonstrate the capability of ADIOS.

Using the ADIOS framework, IMPACT sends the positions and momentums to the *libsim* system, and a histogram function from *VTK* is attached to produce 1-D and 2-D histograms for each of the six variables. The histogram functions are instructed to divide the data records into 100 equal-width bins between the minimum and maximum values.

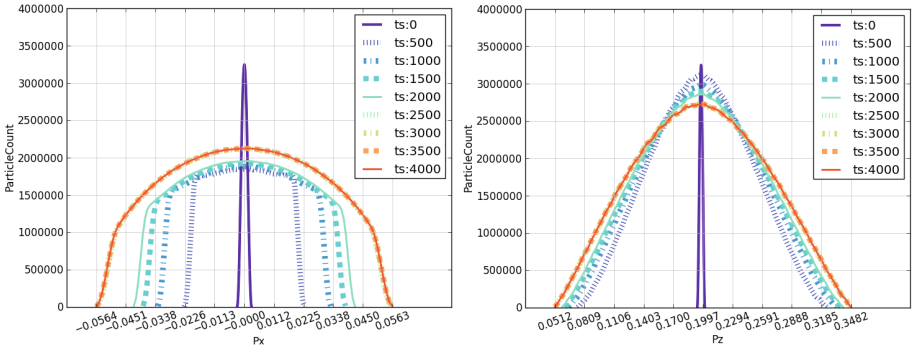


Fig. 2. Histogram of p_x and p_z at time steps 0, 500, 1000, 1500 and 2000 of an IMPACT run. Histogram of p_y is similar to that of p_x.

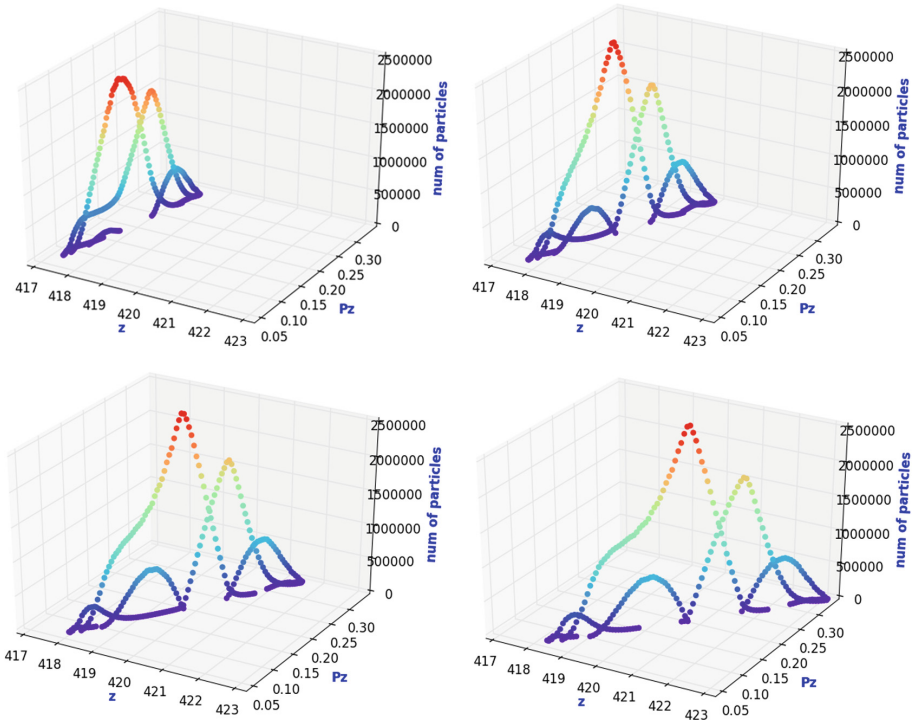


Fig. 3. Histogram of z and p_z at time steps 500, 1000, 1500, and 2000 of an IMPACT run (time progresses from left to right, top to bottom). The two tall peaks appear to move to the right indicating the bulk of the particles are moving along the z direction with increased momentum p_z.

Figure 2 shows a sample output from the histogram computation. In this simulation, the particles travel in the z direction. From the histograms of p_x, we see very sharp drop as momentums along x (and y) directions increase in magnitude reflecting the fact that the accelerators are designed to limit the motions perpendicular to z direction. In contrast, we see the histograms of p_z have a more gradual drop off as the momentum deviates from the center.

Figure 3 shows a series of 2D histograms of z and p_z. In this case, as time progresses (from left to right, top to bottom), we see that the peak of the curves move to the right indicating the bulk of the particles are moving along z direction as designed. There appears to be two groups of particles following different trajectories over time.

The above figures demonstrate two analysis options among many possible particle statistics could be computed with *in situ* analysis capability. We note two important ADIOS features in this use case. First, ADIOS *in situ* framework can effectively support analysis tasks with zero-copy data transfers. This is an important feature since the analysis task may require a large amount of data and copying the data would require extra memory and computer time. Second, we demonstrate that the ADIOS framework can easily work with a Fortran program. This is a useful feature since a large number of popular science codes are in Fortran.

Additionally, we have tested the options of the *in situ* processing capability of ADIOS to compute all the built-in statistics on a small set of separate compute nodes. Since the computation of the statistics involve a large number of global reductions, reducing the number of processors involved also reduce the overall cost of completing the simulation and the computation of the statistics. Using the common measure of CPU-core-hour (number of CPU cores used multiply the number of hours elapsed), the *in situ* processing option could reduce the overall CPU-core-hour by as much as 20% by overlapping the main simulation with the computation of the statistics.

Figure 4 shows a careful measurement of fraction of total execution time devoted to I/O operations with the standard I/O option (of writing data to files stored on a large parallel file system) and with the ADIOS *in situ* mechanism

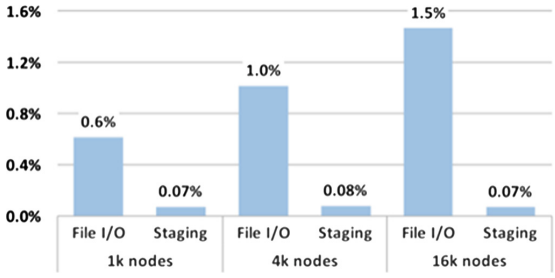


Fig. 4. The fraction of total execution time spent on I/O operations: using File I/O vs using ADIOS *in situ* capability to staging the output data before committing to disk.

to stage the data away from the compute nodes before committing the data to disk[7]. We note that the fraction of time spent on I/O operations is dramatically reduced. More importantly, it is possible to attach an index generation function and the above mentioned statistics computation to the *in situ* workflow without delaying the main simulation computation.

5 Query API

A common query interface is web search box on a web browser, where user enters a set of keywords to locate relevant pages on the web. A similar interface for finding interesting data records in large scientific data collection would also be very useful, however, this functionality is not widely available. An important reason for this lack of querying function is that most scientific datasets are stored as files. Because the POSIX file systems treat a file as a container of bytes, there is no general way of extracting meaningful data records for querying. The first step in breaking this limitation is to have a model to describe the data records. In this work, we are using the ADIOS library and will follow the array data model. In the remaining of this section, we will describe this data model and the query use cases. The latest version of ADIOS release contains the query interface and detailed description of how to use the functions is available in the user's manual[8].

5.1 Array Data Model

In ADIOS, the bulk of data is expressed as multi-dimensional arrays. In Fig. 5, we provide a simple illustration of a 3D array. Often such an array is produced from a simulation program, and each element of the array corresponds to a point or a cell from the 3D space being simulated. In such a case, there might be a number of different variables associated with each point or cell in space, e.g.,

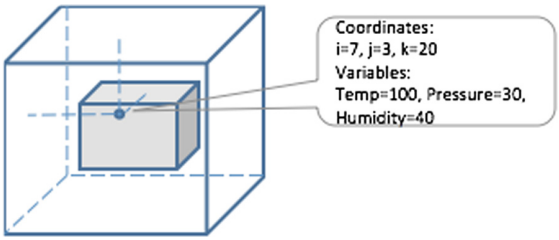


Fig. 5. Illustration of a 3D bounding box and a data record.

[7] This time measurement was obtained with a large XGC simulation running on titan at ORNL.

[8] ADIOS source code and documentation could be found at https://www.olcf.ornl.gov/center-projects/adios/.

temperature, pressure and humidity as shown in Fig. 5. We view all variables at a single point as a data record, which allows us to ask for the temperature values at points where pressure is between 20 and 40, and humidity is greater than 35.

On a parallel computer, a large multidimensional array is commonly divided onto different processors in blocks that could be expressed as bounding boxes. The existing ADIOS interface supports selective accesses to these bounding boxes. For example, to divide the above 3D array onto 1000 processors, each processor might have 1/1000th of the 3D array. A bounding box in ADIOS is expressed as an offset and extent. Say the 3D array has 1000 element along each of the three directions, then the bounding box for the entire array can be expressed as offset = [0, 0, 0] and size = [1000, 1000, 1000]. One way to divide this array into 1000 pieces might be to have each of the subarrays with the size of [100, 100, 100]. In a simple case, we can view the $1000 \times 1000 \times 1000$ array to be defined on a $1000 \times 1000 \times 1000$ mesh. To simplify the following discussion, we assume this is the case. However, the elements of an array may have a much more complex relationship with the underlying physical domain of the simulation. For example, irregular mesh points are often packed into 1D arrays with additional arrays used to describe the physical location of the mesh points and how the mesh points are connected.

5.2 Query Use Cases

Case 1: Regular mesh data, all variables are named explicitly. Given a dataset defined on m dimensions: $D_1, D_2, \ldots, D_m$, the n physical properties such as temperature, pressure and humidity, could be defined as separate m-dimensional arrays: $A_1, A_2, \ldots, A_n$. Each of these variables can be thought of as a column of a relational table and each point of the mesh as a row of the same table. Given this simple mapping between multidimensional data model and the relational data model, we can transplant all SQL queries to queries on mesh data. For example, "select humidity from mesh_data where temperature $>$ 280 and pressure $>$ 100000" is meant to select all mesh points where temperature and pressure values satisfy the specified conditions and then output the values of humidity on those mesh points.

Case 2: using bounding boxes to partition arrays of the same shape and size. Given a dataset defined on a 3-D mesh of size $10 \times 20 \times 30$, we might divide this mesh for 8 processors as a $2 \times 2 \times 2$ blocks. To accommodate this use case, we will define a set of 8 non-overlapping bounding boxes, one for each of the 8 processors. This would allow each processor to answer queries on 1/8th of the data, corresponding to mesh size $5 \times 10 \times 15$.

A query over this structure consists of 3 parts:

1. The selection box to limit the points considered,
2. The query conditions - in a form of query predicates connected with AND/OR operators,
3. The query output - the values of variables for the points that qualify.

An example of a query could be:

1. Selection box: starts = $[5, 0, 15]$, sizes = $[5, 10, 15]$
2. Query conditions: temperature > 5 AND pressure < 40
3. Query output: humidity

Case 3: Composite array structures. Users sometimes combine multiple variables into a single array. Continuing with the example involving temperature, pressure, and humidity, assume the mesh size to be $10 \times 20 \times 30$. The 3 dimensions could be linearized into a single dimension with 6000 elements. The three variables could then be put into a 6000×3 2D array illustrated in Fig. 6.

In such a case the individual variables could be specified with bounding boxes. Assuming the overall array is named A, the same query from the previous example could be expressed as follows, where temperature $\equiv A[0 : 6000, 0 : 1]$, pressure $\equiv A[0 : 6000, 1 : 2]$, and humidity $\equiv A[0 : 6000, 2 : 3]$; and the query specified in the previous use case could be expressed as,

SELECT $A[0 : 6000, 2 : 3]$ WHERE $A[0 : 6000, 0 : 1] > 5$ AND $A[0 : 6000, 1 : 2] < 40$.

Note that the bounding boxes are associated with each variable in the query expression separately, and the sizes of the bounding boxes must be the same; but the offsets (the starting positions) could be different.

Case 4: A general array structure. It is possible that some of the variables are packed together while others are not. More generally, the arrays may have more

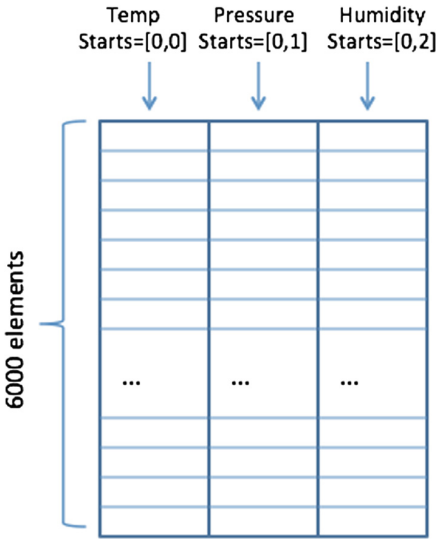


Fig. 6. Illustration of a use case with different variables packed as another dimension of the data array.

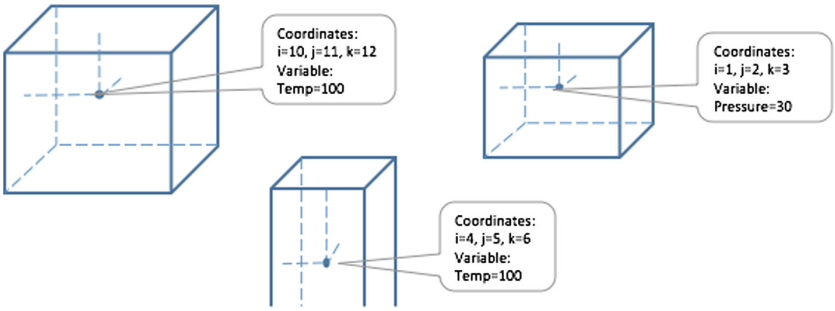


Fig. 7. Illustration of a user query involving multiple arrays of different shapes and sizes.

complex relationship than described above. For example, the values for temperature, pressure, and humidity, could be produced from different measuring instruments and recorded as different time resolutions in space and time, as illustrated in Fig. 7.

Now if we want to compare values at a particular city, we will need to use different bounding boxes on these arrays. This use case is similar to the previous one, the key difference is that the array names would be different. Again, the bounding boxes are required to be of the same size, i.e., having the same number of data points.

5.3 Additional Design Considerations

Reading Multiple Variables. To start with, the current design of the ADIOS query interface retrieves values from on variable at a time. If a use case requires multiple output variables, the caller needs to repeat the invocation of the read function. Introducing a mechanism to specify multiple output variables at once will increase the likelihood of additional optimization in the implementation. However, we choose to keep the interface relatively simple so that we can explore the implementation challenges associated with the basic tasks of integrating with indexing techniques. This and other performance optimization issues will be considered in the future.

Expressing Query Conditions. To avoid the need to introducing a query parser, we have opted to introduce a set of functions for users to compose query expressions instead of allowing the user to specify the query conditions in the string form, even though the string form is a more common form of query interface. This choice also has the benefit of not imposing any restrictions on the variable names. A typical database management system supports query in the SQL language, which imposes a number of restrictions on the variable names, such as, not allowing punctuations, which would introduce extra challenge is expressing the bounding boxes.

Integration with Indexing Software. An effective implementation of the query interface would need to connect to an indexing capability. Our implementation is designed to allow multiple indexing systems to be used. In later tests, we will explore two different ones: a bitmap indexing software library named FastBit [26] and a *MinMax* index capability built into the ADIOS software. A set of user specified conditions may select data records that are scattered randomly in a multidimensional array. Since reading randomly scattered values are much slower than reading consecutive values, some optimization is necessary to reduce the time needed to read the selected values. In our work, we have developed a set of heuristics to combine small random read operations into large sequential read operations.

6 Query Performance

The naive way of resolving a query would be to read through all data records to find those satisfying the user specified conditions. This option is generally known as *scanning*. In this work, we plan to use two different indexing techniques, FastBit index and block MinMax index, to accelerate the query answering process.

FastBit implements a number of different bitmap indexes that have been shown to work well for a number of scientific use case [26]. The block MinMax index is a structure that keeps the minimum and maximum for each variable in a data block. It is a mechanism developed to take advantage of the metadata already captured in the ADIOS BP file format. When processing a user query, this mechanism first examine each data block's header information to determine whether there are any possible entries satisfying the specified conditions using the minimum and maximum values. It only examines the data values in a block if there are possible hits. The mechanism allows us to avoid some blocks. For the data blocks with hits, since the minimum unit of an IO operation is a block, this query answering mechanism is reading the minimum number of blocks and performing the minimum amount of IO operations.

Figure 8 shows the time needed to resolve the queries with three different mechanisms: scanning the raw data, using FastBit indexes, and using the MinMax mechanism. We see that using the two indexing schemes could dramatically reduce the query processing time compared to scanning the raw data. Compared these two indexing mechanisms, we see that the FastBit indexing is typically faster, however, the FastBit indexes take up a lot more storage space than the MinMax mechanism, which can be regarding as not using any extra space in ADIOS BP format.

Another important observation from Fig. 8 is that the time needed to read the selected values (i.e., the difference between "FastBit + read" and "FastBit") is significantly longer than resolving the query using one of the indexing techniques. This is because the query results are typically randomly scattered in the data file. Extracting these randomly scattered values takes a long time. Often, reading a relatively small number of bounding boxes to encompass the randomly scattered points and then extract the selected values could reduce the overall time need to extract these values.

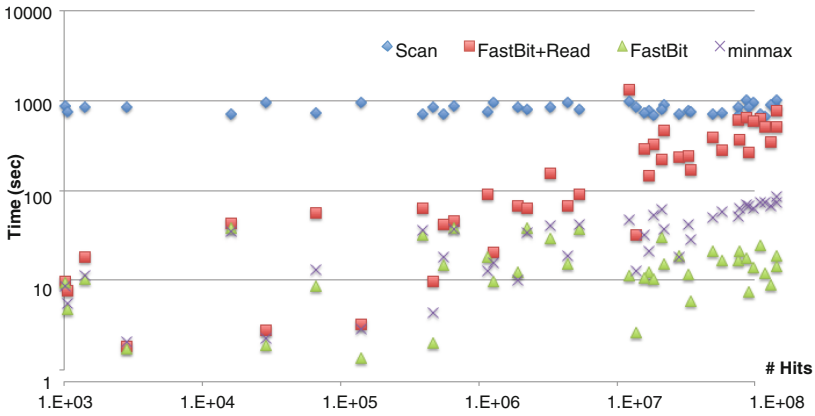


Fig. 8. Query processing time with a set of S3D data (total number of records is 1.67×10^9, organized into a 3D array of $1100 \times 1080 \times 1408$).

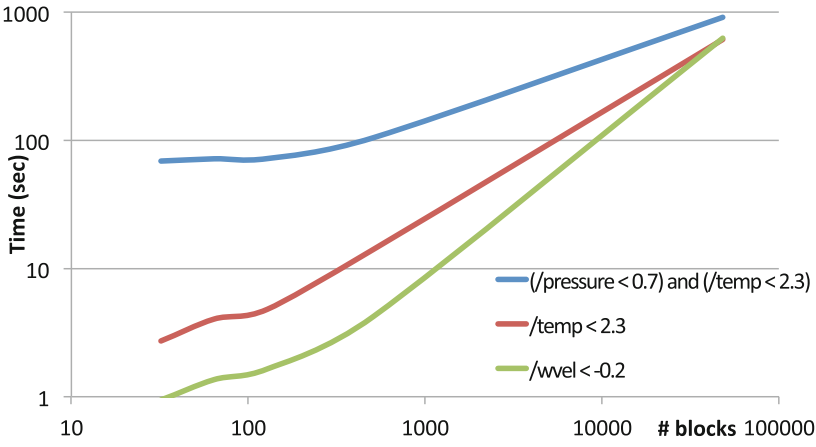


Fig. 9. Query processing time changes dramatically with the number of blocks.

When working with a large dataset, we typically employ multiple CPUs and process each data blocks independently on each CPU core. However, the query processing time could be dramatically affected by the number of blocks used to generated the indexes. Figure 9 shows the query processing time of a small number of queries when the FastBit indexes are generated on different number of blocks. Clearly, the more blocks are used the longer it takes to resolve a query. This is largely because the extra work needed to process each index block. On the other hand, using more processors can significantly reduce the query response time, as shown in Fig. 10. Additional studies are needed to further optimize these and other parameters affecting the performance of indexing and query processing techniques [27].

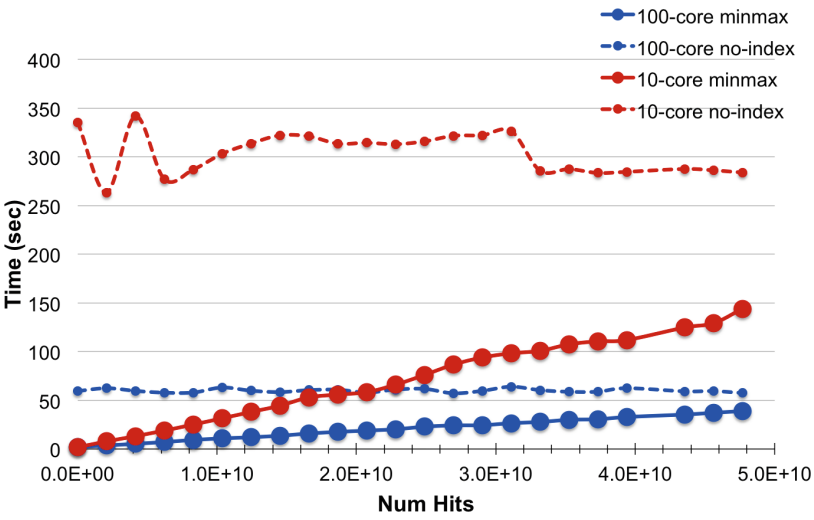


Fig. 10. Using more processors reduces the query processing time.

7 Summary

This work reports our experience in designing and implementing a query interface for ADIOS. We explored a number of different indexing data structures for supporting such a query interface. We observe that for queries that select a relatively small fraction of total number of records in a dataset, answering a query with these indexing methods could be dramatically faster than reading the whole data and then filtering the data records in memory.

In addition to using external indexing libraries, ADIOS also implements a block MinMax mechanism to take advantage of the built-in blocking structure. Tests show that it has the potential to significantly accelerate the query answering process. One challenge we have noticed is that the number of blocks has a strong impact on the overall system performance. We have started exploring possible options to select this and other parameters affecting the query processing time [27].

This work also demonstrates two useful capability of ADIOS in improving the IO pipeline of a simulation program called IMPACT: checkpointing and customizing analysis. In reading and writing of checkpoint files, ADIOS allows the user to manage the IO operations more efficiently. We rely on the *in situ* processing capability of ADIOS to enable IMPACT users to customize the particle statistics during the simulation process.

The *in situ* mechanism is also used to generate the indexes needed to accelerate the query processing algorithms, without increasing the elapsed time used by the simulation programs. It allows the indexes to be generated when the data file is generated, which means the indexes are available when the data is ready. This is very convenient for the users.

In the future, we plan to more fully explore the two capabilities described above. In addition, we plan to compare the query capability with well-known systems such as RasdaMan [2] and SciDB [3].

Acknowledgment. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 (for LBNL) and DE-AC05-00OR22725 Mod 877 (for ORNL). This research also used resources of the National Energy Research Scientific Computing Center supported by the same funding agency.

References

1. Bauer, A.C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., Moreland, K., O'Leary, P., Vishwanath, V., Whitlock, B., Bethel, E.W.: In situ methods, infrastructures, and applications on high performance computing platforms. Comput. Graph. Forum **35**(3), 577–597 (2016)
2. Baumann, P.: rasdaman - raster data manager, January 2018. rasdaman.org
3. Brown, P.G.: Overview of sciDB: large scale array storage, processing and analysis. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, pp. 963–968. ACM, New York (2010)
4. Chen, J.H., Choudhary, A., De Supinski, B., DeVries, M., Hawkes, E.R., Klasky, S., Liao, W.-K., Ma, K.-L., Mellor-Crummey, J., Podhorszki, N., et al.: Terascale direct numerical simulations of turbulent combustion using S3D. Comput. Sci. Discov. **2**(1), 015001 (2009)
5. Chou, J., Wu, K., Rübel, O., Howison, M., Qiang, J., Prabhat, Austin, B., Bethel, E.W., Ryne, R.D., Shoshani, A.: Parallel index and query for large scale data analysis. In: SC11 (2011)
6. Comer, D.: The ubiquitous B-tree. Comput. Surv. **11**(2), 121–137 (1979)
7. Dong, B., Byna, S., Wu, K.: Expediting scientific data analysis with reorganization of data. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–8, September 2013. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6702675
8. Dong, B., Byna, S., Wu, K.: SDS: a framework for scientific data services. In: Proceedings of the 8th Parallel Data Storage Workshop (2013). http://www.pdsw.org/pdsw13/papers/p27-pdsw13-dong.pdf
9. Dong, B., Byna, S., Wu, K., Prabhat, Johansen, H., Johnson, J.N., Keen, N.: Data elevator: low-contention data movement in hierarchical storage system. In: 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), pp. 152–161, December 2016
10. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the HDF5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, pp. 36–47. ACM (2011). http://www.hdfgroup.org/HDF5/
11. Gosink, L., Shalf, J., Stockinger, K., Wu, K., Bethel, W.: HDF5-FastQuery: accelerating complex queries on HDF datasets using fast bitmap indices. In: SSDBM 2006, Vienna, Austria, July 2006, pp. 149–158. IEEE Computer Society Press (2006)
12. Graefe, G.: Query evaluation techniques for large databases. ACM Comput. Surv. (CSUR) **25**(2), 73–169 (1993)

13. Hey, T., Tansley, S., Tolle, K. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft, Redmond (2009)
14. Im, H.G., Chen, J.H., Law, C.K.: Ignition of hydrogen/air mixing layer in turbulent flows. In: Twenty-Seventh Symposium (International) on Combustion, The Combustion Institute, pp. 1047–1056 (1998)
15. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with mapreduce: a survey. ACM SIGMOD Record **40**(4), 11–20 (2012)
16. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., Parashar, M., Samatova, N., Schwan, K., Shoshani, A., Wolf, M., Wu, K., Yu, W.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. Concurr. Comput. Pract. Exp. **26**, 1453–1473 (2014). https://www.olcf.ornl.gov/center-projects/adios/
17. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In: CLADE 2008, pp. 15–24. ACM, New York (2008)
18. Ozsu, M.T.: Principles of Distributed Database Systems, 3rd edn. Prentice Hall Press, Upper Saddle River (2007)
19. Qiang, J., Lidia, S., Ryne, R.D., Limborg-Deprey, C.: Three-dimensional quasistatic model for high brightness beam dynamics simulation. Phys. Rev. Spec. Topics-Accel. Beams **9**(4), 044204 (2006)
20. Qiang, J., Ryne, R.D., Habib, S., Decyk, V.: An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators. J. Comput. Phys. **163**(2), 434–451 (2000)
21. Qiang, J., Ryne, R.D., Venturini, M., Zholents, A.A., Pogorelov, I.V.: High resolution simulation of beam dynamics in electron linacs for X-ray free electron lasers. Phys. Rev. ST Accel. Beams **12**, 100702 (2009)
22. Rew, R., Davis, G.: NetCDF: an interface for scientific data access. IEEE Comput. Graphics Appl. **10**(4), 76–82 (1990). http://www.unidata.ucar.edu/software/netcdf/
23. Roman, E.: A survey of checkpoint/restart implementations. Technical report, Lawrence Berkeley National Laboratory (2002)
24. Shoshani, A., Rotem, D. (eds.): Scientific Data Management: Challenges, Technology, and Deployment. Chapman & Hall/CRC Press, Boca Raton (2010)
25. White, T.: Hadoop - The Definitive Guide: MapReduce for the Cloud. O'Reilly, Sebastopol (2009)
26. Wu, K., Ahern, S., Bethel, E.W., Chen, J., Childs, H., Cormier-Michel, E., Geddes, C., Gu, J., Hagen, H., Hamann, B., Koegler, W., Lauret, J., Meredith, J., Messmer, P., Otoo, E., Perevoztchikov, V., Poskanzer, A., Prabhat, Rübel, O., Shoshani, A., Sim, A., Stockinger, K., Weber, G., Zhang, W.-M.: FastBit: interactively searching massive data. In: SciDAC 2009. LBNL-2164E (2009)
27. Wu, T., Chou, J., Podhorszki, N., Gu, J., Tian, Y., Klasky, S., Wu, K.: Apply block index technique to scientific data analysis and I/O systems. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2017, pp. 865–871. IEEE Press, Piscataway, May 2017
28. Zhang, H., Wen, Y., Xie, H., Yu, N.: Distributed Hash Table: Theory, Platforms and Applications. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-9008-1